Variable-Centered Intelligent Rule System

Membangun Sistem Pakar yang Belajar, dari Nol hingga Aplikasi Nyata



Irfan Subakti

Profitina
profitina.com

Variable-Centered Intelligent Rule System

Membangun Sistem Pakar yang Belajar, dari Nol hingga Aplikasi Nyata

Irfan Subakti

Profitina

Hak cipta © 2026 Irfan Subakti. Hak cipta dilindungi undang-undang.

Diterbitkan oleh Profitina · profitina.com

Edisi pertama, September 2026.

Setiap program dalam buku ini sudah dikompilasi dan dijalankan sebelum keluarannya dicetak. Kode sumber lengkapnya ada di folder Code yang menyertai buku ini. Program-program tersebut disediakan untuk keperluan belajar-mengajar; Irfan Subakti dan penerbit tidak bertanggung jawab atas penggunaannya di luar keperluan itu.

Daftar Isi

Kata Pengantar ..... **xix**
Untuk siapa buku ini ditulis xix
Satu gagasan, dua bahasa xx
Satu mesin: vcirs-core xx
Isi buku ini xxi
Cara membacanya xxii
Tentang penulis xxii
Ucapan terima kasih xxiii

BAB 1 Aturan Cerdas dalam Keseharian, dan Meja Kerja Anda **1**
1.1 Pagi yang Penuh Aturan 1
1.2 Apa yang Membuat Aturan Menjadi Cerdas 2
1.2.1 Fakta, kondisi, dan konklusi 2
1.2.2 Tiga hal yang membuat aturan cerdas 2
1.2.3 Aturan yang sudah Anda percayai 3
1.3 Tiga Keluarga Sistem Aturan 3
1.3.1 Sistem berbasis aturan: jawaban kuat, sulit diubah 4
1.3.2 Ripple Down Rules: mudah diajari, jawabannya sederhana 4
1.3.3 VCIRS: yang terbaik dari keduanya 5
1.4 VCIRS dengan Kata-Kata Sederhana 5
1.4.1 Variabel, node, dan rule 5
1.4.2 Membangun pengetahuan kasus demi kasus 6
1.4.3 Menghitung apa yang dipakai 6
1.4.4 Menjawab disertai penjelasan 6
1.4.5 Satu mesin, dua bahasa 7
1.5 Mengapa Java, dan Mengapa Python 7
1.6 Perkakas di Meja Kerja Anda 8
1.7 Memasang di Windows, macOS, dan Linux 9
1.7.1 Windows, langkah demi langkah 10
1.7.2 macOS, langkah demi langkah 11
1.7.3 Linux, langkah demi langkah 11
1.7.4 VS Code dan kedua ekstensinya 12
1.7.5 Kode buku ini di komputer Anda 12
1.7.6 Bila ada yang tidak beres 13
1.8 Program Pertama Anda dalam Dua Bahasa 13
1.9 Aturan-Aturan Pertama Anda 15
1.10 Cicipan Pertama VCIRS 17
1.11 Memakainya Sungguh-Sungguh 19
1.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 19
1.13 Ringkasan Bab 20
1.14 Soal Latihan 21
1.15 Penyelesaian Soal Latihan 21
Daftar Pustaka 24

BAB 2 Penghemat Tagihan Listrik: Program Aturan Pertama Anda **26**
2.1 Tagihan Bu Sri 26
2.2 Gagasannya dengan Kata-Kata Sederhana 27
2.2.1 Daya, waktu, dan energi 27
2.2.2 Siaga: daya saat "mati" 27
2.2.3 Dari energi ke rupiah 27

2.2.4 Watt, volt-ampere, dan pemutus arus 28
2.2.5 Aturan penghematan 28
2.2.6 Mengapa aturan, bukan sekadar lembar kerja? 29
2.3 Langkah 1: Tagihan Tanpa Aturan 29
2.4 Langkah 2: Aturan Pertama, sebagai Pernyataan `if` 31
2.5 Langkah 3: Aturan sebagai Nilai 32
2.5.1 Rancangannya 32
2.5.2 Berkas rumah tangga 33
2.5.3 Hanya bilangan bulat 33
2.5.4 Sebuah peralatan 34
2.5.5 Rumah tangga dan tagihannya 36
2.5.6 Aturan adalah sebuah nilai 41
2.6 Menjalankan Penghemat 51
2.7 Membuatnya Lebih Baik 55
2.7.1 Semua aturan dalam satu halaman 55
2.7.2 Menambah sebuah aturan 56
2.7.3 Menguji aturan-aturan 56
2.7.4 Satu peralatan melewati mesin 56
2.7.5 Dari daftar aturan ke basis pengetahuan 57
2.7.6 Yang tidak dapat dilihat aturan 57
2.7.7 Mencocokkan penghemat dengan meteran 57
2.8 Memakainya Sungguh-Sungguh 58
2.9 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 60
2.10 Ringkasan Bab 61
2.11 Soal Latihan 61
2.12 Penyelesaian Soal Latihan 62
Daftar Pustaka 64

BAB 3 Pemecah Masalah Ponsel dan Wi-Fi yang Belajar dari Setiap Perbaikan 65
3.1 Konter Mas Anto 65
3.2 Gagasannya dengan Kata-Kata Sederhana 66
3.2.1 Mengapa daftar aturan yang datar rusak 66
3.2.2 Pengetahuan datang bersama konteksnya 69
3.2.3 Kasus cornerstone dan daftar perbedaan 69
3.2.4 Bentuk pengetahuan: except dan else 70
3.2.5 Keunggulan dan keterbatasan RDR 71
3.2.6 Ripple Down Rules dalam pekerjaan sungguhan 72
3.3 Ripple Down Rules dari Nol 72
3.4 Mas Anto Mengajar Lima Pelanggan 79
3.5 Dari Mesin Kita ke vcirs-core 83
3.6 Dokter Wi-Fi 84
3.6.1 Pengetahuan minggu pertama 84
3.6.2 Programnya 85
3.7 Empat Pelanggan di Konter 93
3.7.1 Basis pengetahuan setelah empat pelanggan 95
3.7.2 Hari Jumat Fajar 97
3.8 Membuatnya Lebih Baik 97
3.9 Memakainya Sungguh-Sungguh 98
3.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 99
3.11 Ringkasan Bab 100
3.12 Soal Latihan 101
3.13 Penyelesaian Soal Latihan 101
Daftar Pustaka 104

BAB 4 Penasihat Pemilihan Jurusan: Berpikir dengan Variabel 105
4.1 Laras dan Bu Wati 105
4.2 Gagasannya dengan Kata-Kata Sederhana 106
4.2.1 Dari nasihat ke variabel 106
4.2.2 Nilai dan pembanding 107
4.2.3 Node: rangkaian kondisi yang pendek 108
4.2.4 Rule: rangkaian node 108
4.2.5 Variable-Centered Rule Structure 109
4.2.6 Bagaimana sebuah saran ditemukan 110
4.3 Langkah 1: Nasihat sebagai Daftar Datar 111
4.4 Langkah 2: Variabel, Node, dan Rule dengan Tangan 115
4.5 Dari Struktur Kita ke vcirs-core 125
4.6 Penasihat Pemilihan Jurusan 126
4.6.1 Apa yang pertama kali diajarkan Bu Wati 126
4.6.2 Programnya 127
4.7 Empat Siswa di Meja Bu Wati 137
4.8 Membaca Strukturnya 140
4.9 Membuatnya Lebih Baik 142
4.10 Memakainya Sungguh-Sungguh 143
4.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 144
4.12 Ringkasan Bab 144
4.13 Soal Latihan 145
4.14 Penyelesaian Soal Latihan 146
Daftar Pustaka 149

BAB 5 Penasihat Cek Kesehatan Mandiri: Penyempurnaan Pengetahuan Kasus demi Kasus 150
5.1 Pak Darmo dan Ners Ratna 150
5.2 Gagasannya dengan Kata-Kata Sederhana 151
5.2.1 Keselamatan dulu, dan tidak dipelajari 151
5.2.2 Pengetahuan yang tumbuh, kasus demi kasus 152
5.2.3 Kasus cornerstone dan daftar perbedaan 152
5.2.4 Enam opsi 153
5.2.5 Memilih kondisi sendiri 153
5.2.6 Memeriksa apa yang akan diubah node baru 154
5.2.7 Tidak pernah menghapus, kadang menghentikan 154
5.3 Langkah 1: Enam Opsi dengan Tangan 155
5.4 Langkah 2: Ners Ratna Mengajari Basis Pengetahuan 161
5.5 Cek Kesehatan Mandiri 170
5.5.1 Titik awal aplikasi 170
5.5.2 Programnya 171
5.6 Empat Orang di Puskesmas 186
5.7 Membuatnya Lebih Baik 190
5.8 Memakainya Sungguh-Sungguh 191
5.9 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 192
5.10 Ringkasan Bab 192
5.11 Soal Latihan 193
5.12 Penyelesaian Soal Latihan 194
Daftar Pustaka 196

BAB 6 Penasihat Masalah Tanaman dan Pohon Buah untuk Petani: Pembangkitan Rule 198
6.1 Kebun Pak Slamet 198
6.2 Gagasannya dengan Kata-Kata Sederhana 199

6.2.1 Pengetahuan yang dipakai 199
6.2.2 Credit: seberapa sering sebuah kondisi disetujui 200
6.2.3 NS, CD, dan Weight: seberapa dibagi, seberapa dekat 200
6.2.4 VUR, NUR, dan RUR 201
6.2.5 Dari pemakaian ke pembangkitan 201
6.2.6 Urutan relatif 202
6.2.7 Pakar punya kata terakhir 203
6.3 Langkah 1: Tingkat Pemakaian dengan Tangan 203
6.4 Langkah 2: Pembangkitan pada Dokter Tani 210
6.5 Dokter Tanaman dan Pohon Buah 216
6.5.1 Titik awal dokter 216
6.5.2 Programnya 218
6.6 Lima Kunjungan 231
6.7 Membuatnya Lebih Baik 235
6.8 Memakainya Sungguh-Sungguh 236
6.9 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 237
6.10 Ringkasan Bab 238
6.11 Soal Latihan 238
6.12 Penyelesaian Soal Latihan 239
Daftar Pustaka 241

BAB 7 Pengarah Pesan Pelanggan Toko Kecil: Statistik Pemakaian dan Kredit 243
7.1 Ponsel Bu Rina 243
7.2 Gagasannya dengan Kata-Kata Sederhana 244
7.2.1 Sinyal, bukan pemahaman 244
7.2.2 Mengarahkan adalah mengklasifikasi 245
7.2.3 Credit: menghitung kesepakatan 245
7.2.4 Bobot, kedekatan, dan tiga tingkat pemakaian 246
7.2.5 Untuk apa angka-angka itu 247
7.2.6 Dua jenis hitungan 247
7.3 Tingkat Pemakaian dengan Tangan 248
7.4 Pendeteksi Sinyal 254
7.5 Pengarah di Atas vcirs-core 257
7.6 Aplikasinya 263
7.7 Minggu Pertama Bu Rina 275
7.8 Membaca Angka-Angkanya 277
7.9 Senin Pagi 279
7.10 Membuatnya Lebih Baik 282
7.10.1 Mengajarkan tiga kesalahan hari Senin 282
7.10.2 Mode suara 283
7.10.3 Merawat daftar kata 284
7.10.4 Pengujian 284
7.11 Memakainya Sungguh-Sungguh 284
7.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 285
7.13 Ringkasan Bab 286
7.14 Soal Latihan 287
7.15 Penyelesaian Soal Latihan 287
Daftar Pustaka 289

BAB 8 Pencari Kerusakan Motor untuk Pengemudi Ojek: Inferensi Maju dan Mundur dengan Penjelasan 290
8.1 Pagi Mas Yudi 290
8.2 Gagasannya dengan Kata-Kata Sederhana 291
8.2.1 Dua arah penalaran 291
8.2.2 Aturan yang memberi makan aturan 292

8.2.3 Apa, Bagaimana, dan Mengapa 292
8.2.4 Seberapa yakin jawabannya? 292
8.2.5 Dari kasus ke aturan 293
8.2.6 Saran, bukan perbaikan 293
8.2.7 Arah mana, kapan? 294
8.3 Perunutan dari Nol 294
8.4 Pengetahuan Cak Man 299
8.5 Pencari Kerusakan 301
8.6 Pagi-Pagi di Pangkalan 312
8.6.1 BeAT Mas Yudi tidak mau hidup 312
8.6.2 Pagi yang sama, secara maju 313
8.6.3 Vario 160 Pak Heru kehilangan tenaga 315
8.6.4 Lampu peringatan pada NMAX 316
8.6.5 Rem yang lembek 316
8.6.6 Menghitung pertanyaan 317
8.7 Membuatnya Lebih Baik 317
8.7.1 Kerusakan yang tak dapat dinamainya 317
8.7.2 Melihat setiap langkah 319
8.7.3 Menggabungkan bukti 320
8.7.4 Urutan aturan 320
8.7.5 Pengujian 320
8.8 Memakainya Sungguh-Sungguh 320
8.9 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 322
8.10 Ringkasan Bab 322
8.11 Soal Latihan 323
8.12 Penyelesaian Soal Latihan 324
Daftar Pustaka 325

BAB 9 Penasihat Kelayakan Pinjaman Koperasi: VCIRS Fuzzy 326
9.1 Pengurus Koperasi Sumber Makmur 326
9.2 Gagasannya dengan Kata-Kata Sederhana 327
9.2.1 Kata-kata yang bertepi lunak 327
9.2.2 Sangat, agak, bukan 328
9.2.3 Kondisi fuzzy dalam VCIRS 329
9.2.4 Dua jenis ketidaktepatan 329
9.2.5 Rasio angsuran dan garis 30 persen 330
9.2.6 Adil dan terbuka sejak rancangannya 330
9.2.7 Satu anggota, dari awal sampai akhir 332
9.2.8 Mengapa bukan skor kredit? 332
9.3 Derajat Keanggotaan dengan Tangan 333
9.4 Hitungan Koperasi 336
9.5 Penasihatnya 341
9.6 Rapat Pengurus 351
9.7 Mengajarkan Keputusan Pengurus 354
9.8 Membuatnya Lebih Baik 356
9.8.1 Menyetel kata-kata bersama pengurus 356
9.8.2 Ambang 356
9.8.3 Hedge dalam pengetahuan 356
9.8.4 Mencatat keputusan yang berbeda 356
9.8.5 Pengujian 356
9.9 Memakainya Sungguh-Sungguh 357
9.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 358
9.11 Ringkasan Bab 359
9.12 Soal Latihan 359
9.13 Penyelesaian Soal Latihan 360

Daftar Pustaka ... 361

BAB 10 Penasihat Penghilang Noda sebagai Aplikasi Web dengan Basis Data ... 363

10.1 Laundry Bu Rini ... 363
10.2 Gagasannya dengan Kata-Kata Sederhana ... 364
10.2.1 Dari satu komputer ke beberapa: aplikasi web ... 364
10.2.2 Mengapa basis data, dan mengapa SQLite ... 364
10.2.3 Basis pengetahuan sebagai tabel ... 365
10.2.4 Bertanya lewat web: pola jalankan-ulang ... 366
10.2.5 Apa yang diketahui penasihat, dan apa yang tidak boleh dilakukannya ... 367
10.3 Langkah 1: Basis Pengetahuan di dalam Basis Data ... 368
10.4 Langkah 2: Advisor, Tanpa Kode Web Apa Pun ... 373
10.5 Langkah 3: Sehari di Laundry, Tanpa Peramban ... 389
10.6 Langkah 4: Server Web ... 395
10.7 Langkah 5: Halaman-Halamannya ... 402
10.8 Menjalankan Penasihat Laundry ... 407
10.8.1 Berbicara dengan server tanpa peramban ... 410
10.8.2 Ponsel di meja depan ... 411
10.8.3 Pengujian ... 411
10.9 Membuatnya Lebih Baik ... 411
10.10 Memakainya Sungguh-Sungguh ... 412
10.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya ... 413
10.12 Ringkasan Bab ... 414
10.13 Soal Latihan ... 415
10.14 Penyelesaian Soal Latihan ... 416
Daftar Pustaka ... 418

BAB 11 Basis Pengetahuan Triase Klinik untuk Tim: Banyak Pakar, Versi, dan Audit ... 419

11.1 Meja Pendaftaran Klinik Sehat Bersama ... 419
11.2 Gagasannya dengan Kata-Kata Sederhana ... 420
11.2.1 Triase: memilah menurut kegawatan, bukan menamai penyakit ... 420
11.2.2 Banyak pakar, satu basis pengetahuan ... 421
11.2.3 Versi: sebuah potret setelah setiap perubahan ... 421
11.2.4 Pengujian: kasus cornerstone dan kasus emas ... 421
11.2.5 Empat mata: tinjauan oleh pakar lain ... 422
11.2.6 Jejak audit, dan mengapa ia dirangkai ... 423
11.2.7 Ketika para pakar tidak sepakat ... 424
11.3 Langkah 1: Dua Pakar, Satu Berkas ... 424
11.4 Langkah 2: Basis Pengetahuan Tim ... 427
11.5 Langkah 3: Penjalan Perintah ... 451
11.6 Seminggu di Klinik ... 458
11.6.1 Senin: versi pertama, dan pasien yang dinilai keliru ... 458
11.6.2 Rabu: usulan yang ditolak pengujian ... 459
11.6.3 Kamis: meja pendaftaran bekerja ... 460
11.6.4 Jumat: kekeliruan, rollback, dan pelajaran ... 461
11.6.5 Apa yang diingat berkas-berkasnya ... 462
11.7 Menengok ke Belakang ... 462
11.7.1 Pengujian ... 465
11.8 Membuatnya Lebih Baik ... 465
11.9 Memakainya Sungguh-Sungguh ... 466
11.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya ... 467
11.11 Ringkasan Bab ... 468
11.12 Soal Latihan ... 468
11.13 Penyelesaian Soal Latihan ... 469
Daftar Pustaka ... 471

BAB 12 Belajar Rule dari Data: dari Pohon Keputusan ke Basis Pengetahuan VCIRS 472
12.1 Klub Pencinta Alam Bu Lestari 472
12.2 Gagasannya dengan Kata-Kata Sederhana 473
12.2.1 Pengetahuan dari pakar, pengetahuan dari data 473
12.2.2 Pohon keputusan adalah permainan tebak-tebakan 473
12.2.3 Bagaimana pohon memilih pertanyaannya: kemurnian 474
12.2.4 Aritmetika yang tepat, agar Java dan Python sepakat 476
12.2.5 Pohon yang benar untuk semua yang pernah dilihatnya 476
12.2.6 Dari lintasan ke node VCIRS 477
12.2.7 Apa yang tidak dapat diketahui data 477
12.3 Langkah 1: CART dari Nol 478
12.4 Langkah 2: Dari Pohon ke Basis Pengetahuan 492
12.5 Langkah 3: Pelajaran dari Sang Guru 497
12.5.1 Enam hewan yang tidak pernah dilihat data 497
12.5.2 Programnya 497
12.5.3 Apa yang terjadi 502
12.5.4 Permainan di klub 505
12.6 Langkah 4: Pemeriksaan terhadap scikit-learn 505
12.6.1 Pengujian 508
12.7 Membuatnya Lebih Baik 508
12.8 Memakainya Sungguh-Sungguh 509
12.9 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 510
12.10 Ringkasan Bab 511
12.11 Soal Latihan 512
12.12 Penyelesaian Soal Latihan 512
Daftar Pustaka 514

BAB 13 Berbicara dengan Sistem Pakar Anda: Model Bahasa Lokal dengan Pengetahuan yang Diperiksa Manusia 515
13.1 Pesan-Pesan Pukul Sembilan Malam 515
13.2 Gagasannya dengan Kata-Kata Sederhana 516
13.2.1 Apa itu model bahasa, dan apa yang bukan 516
13.2.2 Dua tugas untuk model, satu untuk basis pengetahuan 516
13.2.3 Model di komputer Anda sendiri: Ollama 518
13.2.4 Meminta JSON, bukan prosa 518
13.2.5 Penjaga 519
13.3 Langkah 1: Bercakap dengan Ollama 519
13.4 Langkah 2: Pengganti yang Berjalan tanpa Internet 526
13.5 Langkah 3: Dokter Obrolan 534
13.6 Lima Percakapan 550
13.7 Menjalankannya dengan Model Sungguhan 552
13.8 Membuatnya Lebih Baik 554
13.9 Memakainya Sungguh-Sungguh 554
13.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 555
13.11 Ringkasan Bab 556
13.12 Soal Latihan 556
13.13 Penyelesaian Soal Latihan 557
Daftar Pustaka 558

BAB 14 Proyek Akhir: Pembangun Sistem Pakar VCIRS untuk Siapa Saja 560
14.1 Buku Catatan Bu Wati 560
14.2 Gagasannya dengan Kata-Kata Sederhana 561
14.2.1 Apa yang harus dapat dilakukan orang dengan pembangun 561
14.2.2 Statistik pemakaian dengan kata-kata sederhana 562
14.2.3 Usulan: mesin menyarankan, manusia memutuskan 563

14.2.4 Satu aplikasi, dua server, halaman yang sama 563
14.3 Langkah 1: Pembangun, tanpa Web 564
14.4 Langkah 2: Sesi Bu Wati, sebagai Skrip 575
14.5 Langkah 3: Halaman-Halaman 584
14.6 Langkah 4: Server-Server Web 600
14.7 Memakai Pembangun 611
14.8 Membuatnya Lebih Baik 616
14.9 Memakainya Sungguh-Sungguh 616
14.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 617
14.11 Ringkasan Bab 618
14.12 Soal Latihan 618
14.13 Penyelesaian Soal Latihan 619
Daftar Pustaka 621

BAB 15 Menerapkan, Merawat, dan Mengembangkan Sistem Pakar Anda 622
15.1 Pagi Ketika Laptop Tidak Mau Menyala 622
15.2 Gagasannya dengan Kata-Kata Sederhana 623
15.2.1 Dari program yang Anda jalankan menjadi layanan yang berjalan 623
15.2.2 Satu berkas untuk disalin 623
15.2.3 Basis pengetahuan adalah data: cadangan yang dapat dipercaya 623
15.2.4 Mengamati bagaimana pengetahuan dipakai 624
15.2.5 Data pribadi dan hukum 624
15.3 Langkah 1: Layanan Saran 625
15.4 Langkah 2: Mengemas dan Menerapkan 636
15.5 Langkah 3: Memeriksa Penerapan 640
15.6 Langkah 4: Cadangan 646
15.7 Langkah 5: Memantau Pemakaian Rule 654
15.8 Merawat dan Menumbuhkan Pengetahuan 663
15.9 Privasi, Etika, dan Keselamatan 665
15.9.1 Data pribadi, dalam praktik 665
15.9.2 Etika memberi saran lewat mesin 666
15.10 Ke Mana Selanjutnya 667
15.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 668
15.12 Ringkasan Bab 669
15.13 Soal Latihan 669
15.14 Penyelesaian Soal Latihan 670
Daftar Pustaka 672

Daftar Gambar

Gambar 1.1 Dua keluarga sistem aturan dan hibrida yang menyatukannya. VCIRS mengambil inferensi dan penjelasan dari sistem berbasis aturan serta pembangunan pengetahuan kasus demi kasus dari Ripple Down Rules, lalu menambahkan statistik pemakaian dan pembangkitan rule. 4

Gambar 1.2 Variable-Centered Rule Structure. Variabel dipakai di dalam node; node adalah rangkaian pendek kondisi beserta konklusinya; rule adalah rangkaian node, dimulai dari tingkat teratas. 5

Gambar 1.3 Meja kerja. Sumber Java dikompilasi oleh javac menjadi bytecode yang dijalankan oleh java; sumber Python dijalankan langsung oleh interpreter python; VS Code menampung keduanya. 8

Gambar 2.1 Penghemat tagihan listrik. Peralatan dan tarif menghasilkan energi dan tagihan; aturan-aturan hemat mengubahnya menjadi saran, masing-masing dengan alasan dan penghematannya. 33

Gambar 2.2 Energi bulanan per peralatan di rumah Bu Sri, sekarang dan bila semua saran yang dapat dihitung diikuti (rumah tangga contoh kita). 54

Gambar 3.1 Daftar perbedaan antara kasus cornerstone sebuah aturan dan sebuah kasus baru. Bagian A memuat apa yang hanya dimiliki kasus lama; bagian B apa yang hanya dimiliki kasus baru. Aturan baru sang pakar dibangun dari perbedaan-perbedaan ini. 70

Gambar 3.2 Pohon Ripple Down Rules klasifikasi tunggal. Aturan 0 selalu benar; cabang except memuat pengecualian, dicoba ketika sebuah aturan benar; cabang else memuat aturan berikutnya, dicoba ketika sebuah aturan salah. Jawabannya adalah konklusi aturan terakhir yang benar. 70

Gambar 4.1 SNBT 2026 dalam angka: kurang dari tiga dari sepuluh pendaftar diterima, dan sebagian besar yang diterima lolos lewat pilihan pertama atau kedua. Data dari siaran pers SNPMB 25 Mei 2026 [4]. 106

Gambar 4.2 Sebagian Variable-Centered Rule Structure nasihat Bu Wati. Setiap variabel disimpan sekali dan dihubungkan ke node-node yang memakainya; setiap node hanya memuat kondisinya sendiri; setiap rule adalah rangkaian node dari tingkat atas ke bawah. 109

Gambar 4.3 Medan-medan sebuah node dan sebuah rule dalam Variable-Centered Rule Structure, diisi untuk `medicine#1`. Rule mendaftar node-node-nya menurut urutan; setiap node mendaftar kondisi-kondisinya menurut urutan. 110

Gambar 5.1 Keselamatan dulu. Pertanyaan keselamatan yang tetap datang sebelum basis pengetahuan; satu jawaban *ya* langsung mengarahkan orang mencari pertolongan. Baru setelah itu basis pengetahuan, yang diajarkan perawat kasus demi kasus, dimintai saran perawatan mandiri. 152

Gambar 5.2 Apa yang akan dibangun tiap opsi untuk satu kasus di puskesmas. Opsi 1 sampai 3 menggantung pengecualian di bawah node lama; opsi 4 menaruh seluruh kasus di sampingnya dan opsi 5 di tingkat atas; pilihan perawat sendiri adalah satu kondisi yang menyatakan maksudnya. 154

Gambar 6.1 Cara menghitung tingkat pemakaian sebuah node. Variable Usage Rate setiap kondisi adalah Credit-nya dikali Weight-nya, dan Weight adalah banyaknya node yang berbagi variabel itu dikali kedekatan kondisi ke konklusi. Tingkat pemakaian node adalah rata-rata tingkat pemakaian kondisi-kondisinya. 200

Gambar 6.2 Pembangkitan rule dalam VCIRS. Tingkat pemakaian memberi urutan relatif pengetahuan; urutan itu serta node dan variabel terpenting memberi kandidat; pakar menyimpan yang benar, dan node baru menjawab kasus yang tak dapat dijawab node mana pun yang diajarkan. 202

Gambar 6.3 Tingkat pemakaian rule basis pengetahuan pertama Mbak Nur. Rule yang paling banyak dipakai, berwarna kuning tua, adalah rule yang node pertamanya punya pengecualian: banyak pengetahuan melewatinya. 215

Gambar 7.1 Perjalanan satu pesan melalui pengarah. Pendeteksi mengubah kata menjadi sinyal ya/tidak, tetapi hanya ketika basis pengetahuan menanyakannya; basis pengetahuan memilih

rute; ketika Bu Rina membetulkan rute yang salah, sebuah pengecualian baru ditambahkan dan kondisi-kondisi induknya memperoleh Credit. 244
Gambar 7.2 Variable Usage Rate terbesar setiap sinyal setelah minggu pertama (abu-abu) dan setelah tiga koreksi hari Senin (hijau toska). Payment, order, late, dan hours naik; yang lain tidak berubah. Sinyal `proof` tidak punya batang karena belum ada node yang memakainya. 283
Gambar 8.1 Dua aturan yang sama dipakai ke dua arah. Runut maju (kiri) naik dari fakta melalui aturan-aturan menuju konklusi; runut mundur (kanan) berangkat dari tujuan, mengubah kondisi aturan menjadi subtujuan, dan hanya menanyakan kepada pengemudi apa yang tidak dapat disimpulkan aturan. 291
Gambar 8.2 Empat tingkat kedaruratan saran pencari kerusakan, yang paling mendesak di atas. Kerusakan rem dan bahan bakar menghentikan perjalanan; sebagian besar kerusakan lain dibawa ke bengkel, hari ini atau minggu ini; hanya sedikit yang dikerjakan pengemudi sendiri. 294
Gambar 9.1 Tiga kata fuzzy untuk rasio angsuran, dengan hedge *very high* (bertitik). Garis putus-putus mendatar adalah ambang 0,5 tempat sebuah kondisi fuzzy dianggap benar; segitiga-segitiga menandai rasio lima anggota di Subbab 9.6. 328
Gambar 9.2 Perjalanan sebuah permohonan melalui penasihat. Hanya fakta tentang kemampuan membayar yang sampai ke mesin; penjaga keadilan menolak pengetahuan apa pun yang memakai atribut yang dilindungi; saran datang bersama CF, penjelasan, peringatan batas tipis, dan jumlah pengandaian; pengurus memutuskan, dan keputusannya diajarkan kembali. 331
Gambar 10.1 Tabel-tabel basis data penasihat noda. Tabel berwarna hijau toska menyimpan basis pengetahuan VCIRS seperti pada Lampiran E tesis; tabel berwarna kuning milik laundry: langkah-langkah setiap penanganan dan log konsultasi. 365
Gambar 10.2 Pola jalankan-ulang. Halaman mengirim setiap jawaban yang dimilikinya; server menjalankan mesin dari awal dan mengembalikan pertanyaan pertama yang belum dapat dijawab mesin, atau, bila tidak ada, sarannya. 366
Gambar 10.3 Penasihat noda sebagai aplikasi web. Peramban dan server berbicara dalam JSON lewat HTTP; Advisor berkonsultasi dengan dan mengajari basis pengetahuan vcirs-core, yang dimuat dari dan disimpan ke tabel-tabel VCIRS, serta mencatat konsultasi di tabel milik laundry. 395
Gambar 10.4 Penasihat mengajukan pertanyaan keduanya. Jawaban sejauh ini tampil sebagai label kecil; alasan pertanyaannya berasal dari mesin. 408
Gambar 10.5 Saran untuk noda teh yang sudah kering pada barang putih: langkah-langkah, peringatan, dan, dalam keadaan dibuka, penjelasan node yang menyala. 408
Gambar 10.6 Mengajari penasihat penanganan noda yang belum pernah dilihatnya. Kotak fakta tambahan dibiarkan kosong karena nodanya sendirilah yang baru. 409
Gambar 10.7 Halaman pengetahuan: angka-angka minggu ini, saran yang paling sering diberikan, pengetahuan sebagai pohon pengecualian, dan perubahan terakhir. 410
Gambar 11.1 Perjalanan sebuah perubahan. Usulan dicoba pada salinan versi yang berlaku lalu diuji; bila setiap kasus masih lolos, pakar lain meninjaunya; hanya perubahan yang disetujui yang menjadi versi baru. Setiap langkah, termasuk setiap penolakan, ditulis ke jejak audit. 422
Gambar 11.2 Jejak audit sebagai rantai. Setiap entri memuat sidik jari berkas versi yang diciptakannya dan sidik jari entri sebelumnya, dan sidik jarinya sendiri mencakup keduanya, sehingga mengubah entri lama atau berkas versi mana pun memutus setiap mata rantai sesudahnya. 423
Gambar 11.3 Tujuh versi dalam seminggu. P2 ditolak pengujian dan tidak pernah menjadi versi; versi 5 menghentikan node yang salah, dan versi 6 adalah salinan versi 4. 462
Gambar 12.1 Dari data ke basis pengetahuan yang sudah diperiksa. Sebuah pohon dipelajari dari tabel, setiap daunnya menjadi node VCIRS, sang guru menambahkan pengecualian, dan setiap pelajaran diperiksa terhadap semua hewan yang sudah dijawab basis pengetahuan. 473
Gambar 12.2 Pohon keputusan yang dipelajari dari 101 hewan dalam data Zoo. Setiap pertanyaan punya cabang ya dan cabang tidak; setiap daun memberikan golongan dan, dalam kurung, berapa hewan dalam data yang sampai ke sana. 474
Gambar 12.3 Pengecualian-pengecualian dari sang guru. Di bawah node ikan hasil belajar, pelajaran tentang penyu dan, di bawahnya, tentang ikan glodok; di bawah node amfibi hasil belajar, pelajaran tentang ekor yang ditolak dan pelajaran tentang sisik yang disimpan. Setiap node menyebut hewan cornerstone-nya. 504

Gambar 13.1 Antarmuka obrolan. Model bahasa (kuning) hanya mengubah kata menjadi fakta dan saran menjadi kalimat; semua keluarannya diperiksa. Basis pengetahuan VCIRS (biru tua) memutuskan dan menanyakan fakta yang masih dibutuhkannya. Penjaga hanya meloloskan jawaban yang memberikan saran basis pengetahuan dan tidak ada yang lain. 517
Gambar 13.2 Klien yang sama bercakap dengan Ollama sungguhan beserta model kecil, dengan server pengganti tanpa internet, atau, bila keduanya tidak menjawab, dengan siapa pun tidak: saat itu ia mengajukan pertanyaannya sendiri dan menulis jawabannya sendiri. 534
Gambar 14.1 Pembangun. Peramban bercakap dengan server web kecil (Flask di Python, HttpServer JDK di Java). Server hanya menghubungkan web dengan dua bagian yang sama sekali tidak mengenal web: halaman, yang menyusun HTML dari basis pengetahuan dan dari kata-kata di `ui-text.json`, dan pembangun, yang mengubah basis pengetahuan melalui vcirs-core dan menyimpannya setelah setiap perubahan. 563
Gambar 14.2 Halaman Beranda setelah mengimpor kasus-kasus kue: nama dan keterangan proyek, ukurannya, serta bagian ekspor, impor, dan proyek baru. “Tersimpan.” menegaskan perubahan terakhir. 611
Gambar 14.3 Halaman Konsultasi: saran dalam huruf besar, bagaimana saran itu ditemukan, tautan untuk mengajarkan jawaban yang benar bila saran itu salah, dan pertanyaan-pertanyaan dalam urutan yang paling membantu. 612
Gambar 14.4 Hasil mengajar: apa yang dijawab basis pengetahuan sebelumnya, apa yang terjadi, dan mengapa kasus baru berbeda dari kasus yang menciptakan node lama. 613
Gambar 14.5 Halaman Statistik: pertanyaan dari yang paling hingga paling tidak berguna beserta banyaknya node yang memakai masing-masing, pertanyaan dan node terpenting, dan Node Usage Rate, dari yang terbesar. 614
Gambar 14.6 Sebuah usulan node di halaman Usulan. Dari kondisi-kondisi yang dikumpulkan VCIRS dalam urutan yang baik, Bu Wati menyimpan satu, `masalah = kering`, untuk membuat node yang lebih pendek dan lebih umum. 615
Gambar 14.7 Halaman Pengetahuan: pohon node dan pengecualian, dengan tanda bila sebuah node hasil pembangkitan, serta tabel berisi siapa yang mengajarkan setiap node dan tombol Hentikan. 615
Gambar 15.1 Dokter kue yang sudah diterapkan. Layanan berjalan dari satu berkas di komputer toko dan menjawab tablet atau ponsel di etalase; ia memuat ulang basis pengetahuan ketika salinan yang sudah diperiksa datang dari laptop Bu Wati, menulis catatan pemakaian, dicadangkan setiap malam, dan dilaporkan setiap bulan. 639
Gambar 15.2 Seberapa sering setiap node memberi saran dalam 120 konsultasi, Juli sampai September 2026 (dari catatan yang diputar ulang), beserta node yang tidak pernah memberi saran dan konsultasi yang tidak mendapat jawaban. 662
Gambar 15.3 Siklus perawatan: layanan dipakai, setiap konsultasi dicatat, laporan bulanan menunjukkan apa yang tak terjawab dan tak terpakai, sang pakar mengajarkan kasus yang kurang di pembangun, basis pengetahuan yang baru diperiksa dan dicadangkan, dan salinan yang sudah diperiksa disalin ke layanan. 664

Daftar Tabel

Tabel 1.1 Peta jalan buku ini: setiap bab membangun sebuah aplikasi nyata dan menambahkan satu gagasan VCIRS. 7
Tabel 1.2 Meja kerja buku ini, sebagaimana diperiksa pada September 2026. Nomor pembaruan yang lebih baru selalu dapat dipakai. 9
Tabel 1.3 Meja kerja pada tiga sistem operasi, dari komputer kosong sampai penyunting yang siap dipakai. Di Fedora pakai `sudo dnf install java-25-openjdk-devel python3.14`; di Arch Linux, `sudo pacman -S jdk-openjdk python`. 10
Tabel 1.4 Folder-folder kode buku ini. Setiap folder bab memiliki bentuk yang sama. 13
Tabel 1.5 Masalah pemasangan yang lazim dan cara memperbaikinya. 13

Tabel 2.1 Tarif listrik rumah tangga PLN untuk Juli–September 2026, tidak berubah pada triwulan tersebut [2], [3]. 28
Tabel 2.2 Kesembilan aturan penghemat tagihan listrik: enam tentang satu peralatan dan tiga tentang rumah. 56
Tabel 3.1 Bagaimana gagasan mesin buatan tangan muncul dalam vcirs-core. 83
Tabel 3.2 Enam opsi untuk menambahkan pengetahuan dalam VCIRS, mengikuti revisi 2006 [8], [9]. 84
Tabel 4.1 Variabel-variabel dalam nasihat Bu Wati, beserta pertanyaan di balik masing-masing dan nilai-nilai lazimnya. 107
Tabel 4.2 Bagaimana struktur buatan tangan dari Langkah 2 muncul dalam vcirs-core. 126
Tabel 5.1 Enam opsi pembangunan pengetahuan VCIRS, beserta pertanyaan yang dijawab masing-masing. Opsi 1 sampai 3 membuat pengecualian node lama; opsi 4 dan 5 menyimpan seluruh kasus sebagai node baru di tempat lain. 153
Tabel 8.1 Apa yang diajarkan Cak Man kepada pencari kerusakan: gejala, penyebab, dan kedaruratan. Aturan tanpa CF dalam kurung ber-CF 100. 300
Tabel 8.2 Pertanyaan yang diajukan dalam sesi-sesi bab ini, termasuk cek keselamatan. 317
Tabel 10.1 Dua belas kasus benih penasihat noda, beserta peringatan yang menyertai setiap penanganan. 367
Tabel 10.2 Tiga panggilan penasihat noda dan apa yang dibawa masing-masing. 373
Tabel 11.1 Tiga tingkat kegawatan dalam basis pengetahuan meja pendaftaran, dan jawaban aman bawaan ketika tidak ada aturan yang cocok. 420
Tabel 11.2 Siapa boleh melakukan apa dalam basis pengetahuan tim klinik. 423
Tabel 12.1 Lima pertanyaan pertama terbaik untuk 101 hewan, menurut ketidakmurnian Gini tertimbang dari dua kelompok yang dihasilkannya. Susu menang. 475
Tabel 13.1 Model terbuka kecil di pustaka Ollama, September 2026, beserta ukuran unduhan versi bawaan setiap tag. 518
Tabel 14.1 Halaman-halaman pembangun, apa yang dilakukan orang di setiap halaman, dan gagasan VCIRS di baliknya. 561

Daftar Kode

Kode 1.1 `Hello.java`: program Java pertama; memeriksa apakah Java cukup baru. 14
Kode 1.2 `hello.py`: program Python pertama; memeriksa apakah Python cukup baru. 14
Kode 1.3 `MorningRules.java`: tiga aturan JIKA-MAKA sehari-hari, masing-masing dengan alasannya. 15
Kode 1.4 `morning_rules.py`: ketiga aturan yang sama dalam Python. 16
Kode 1.5 `FirstTaste.java`: basis pengetahuan mempelajari sebuah aturan lalu pengecualiannya. 17
Kode 1.6 `first_taste.py`: cicipan pertama yang sama dalam Python. 18
Kode 2.1 `BillCalculator.java`: tagihan tiga peralatan, belum ada aturan. 29
Kode 2.2 `bill_calculator.py`: tagihan yang sama dalam Python. 30
Kode 2.3 `FirstAdvice.java`: dua aturan penghematan yang ditulis langsung sebagai pernyataan `if`. 31
Kode 2.4 `first_advice.py`: kedua aturan yang sama dalam Python. 32
Kode 2.5 `Appliance.java`: satu peralatan, energi bulanannya, dan salinan-salinan yang menggambarkan saran. 34
Kode 2.6 `Household.java`: membaca kedua berkas, dan aritmetika tagihan dengan bilangan bulat. 36
Kode 2.7 `household.py`: peralatan, berkas rumah tangga, dan aritmetika tagihan dalam Python. 38
Kode 2.8 `SavingRule.java`: aturan penghematan sebagai nilai dengan lima bagian. 41
Kode 2.9 `BillSaver.java`: penghemat tagihan listrik, dengan aturan-aturannya sebagai nilai dan mesin kecil yang menerapkannya. 42
Kode 2.10 `bill_saver.py`: penghemat tagihan listrik dalam Python. 47

Kode 3.1 `FlatRules.java`: pemecah masalah sebagai daftar aturan yang datar, dicoba dalam tiga versi. 66
Kode 3.2 `flat_rules.py`: ketiga versi yang sama dalam Python. 67
Kode 3.3 `Rdr.java`: mesin Ripple Down Rules klasifikasi tunggal yang lengkap. 72
Kode 3.4 `rdr.py`: mesin yang sama dalam Python. 76
Kode 3.5 `RdrDemo.java`: lima pelanggan mengajari pemecah masalah yang masih kosong. 79
Kode 3.6 `rdr_demo.py`: kelima pelanggan yang sama dalam Python. 80
Kode 3.7 `WifiDoctor.java`: dokter Wi-Fi, pemecah masalah yang belajar dari setiap perbaikan, di atas vcirs-core. 85
Kode 3.8 `wifi_doctor.py`: dokter Wi-Fi dalam Python. 90
Kode 4.1 `FlatAdvice.java`: nasihat Bu Wati sebagai daftar aturan datar, diindeks per variabel. 111
Kode 4.2 `flat_advice.py`: daftar datar dan indeks yang sama dalam Python. 112
Kode 4.3 `MiniStructure.java`: tabel Variabel, Node Structure, dan Rule Structure, dibangun dengan tangan. 116
Kode 4.4 `mini_structure.py`: tiga tabel yang sama dalam Python. 120
Kode 4.5 `StudyAdvisor.java`: penasihat pemilihan jurusan di atas vcirs-core. 127
Kode 4.6 `study_advisor.py`: penasihat pemilihan jurusan dalam Python. 133
Kode 5.1 `SixOptions.java`: daftar perbedaan dan enam opsi, dikerjakan dengan tangan. 155
Kode 5.2 `six_options.py`: perhitungan yang sama dalam Python. 158
Kode 5.3 `NurseTeaches.java`: seorang perawat mengajari basis pengetahuan kasus demi kasus, di atas vcirs-core. 161
Kode 5.4 `nurse_teaches.py`: sesi mengajar yang sama dalam Python. 165
Kode 5.5 `HealthCheck.java`: cek kesehatan mandiri, keselamatan dulu, di atas vcirs-core. 171
Kode 5.6 `health_check.py`: cek kesehatan mandiri dalam Python. 181
Kode 6.1 `UsageByHand.java`: Credit, NS, CD, Weight, VUR, NUR, dan RUR dengan tangan, dicocokkan dengan vcirs-core. 203
Kode 6.2 `usage_by_hand.py`: perhitungan yang sama dalam Python. 207
Kode 6.3 `GenerateDemo.java`: pembangkitan rule dan node pada basis pengetahuan tani, dengan telaah penyuluh. 210
Kode 6.4 `generate_demo.py`: pembangkitan dan telaah yang sama dalam Python. 212
Kode 6.5 `FarmDoctor.java`: dokter tanaman dan pohon buah di atas vcirs-core, dengan tingkat pemakaian dan pembangkitan. 218
Kode 6.6 `farm_doctor.py`: dokter tanaman dan pohon buah dalam Python. 226
Kode 7.1 `UsageByHand.java`: tingkat pemakaian lima node dihitung dengan tangan, lalu oleh vcirs-core. 248
Kode 7.2 `usage_by_hand.py`: perhitungan yang sama dalam Python. 251
Kode 7.3 `Signals.java`: pendeteksi sinyal membaca daftar kata dan menemukan kata utuh serta frasa dalam sebuah pesan. 254
Kode 7.4 `signals.py`: pendeteksi yang sama dalam Python. 256
Kode 7.5 `Router.java`: membangun basis pengetahuan, mengarahkan pesan, dan mengajarkan koreksi. 257
Kode 7.6 `router.py`: tiga operasi yang sama dalam Python. 261
Kode 7.7 `ShopRouter.java`: pengarah pesan Toko Rina dengan lima perintahnya. 263
Kode 7.8 `shop_router.py`: aplikasi yang sama dalam Python. 270
Kode 8.1 `Chain.java`: runut maju dan runut mundur dari nol, pada empat aturan pencari kerusakan. 294
Kode 8.2 `chain.py`: kedua arah yang sama dalam Python. 297
Kode 8.3 `FaultFinder.java`: pencari kerusakan dengan cek keselamatannya dan tiga cara bernalar. 301
Kode 8.4 `fault_finder.py`: aplikasi yang sama dalam Python. 308
Kode 9.1 `FuzzyTerms.java`: derajat keanggotaan dan hedge dengan tangan, dicocokkan dengan vcirs-core. 333
Kode 9.2 `fuzzy_terms.py`: tabel yang sama dalam Python. 334

Kode 9.3 `Loans.java`: angsuran, rasio angsuran, dan jumlah pengandaian, dalam rupiah bulat. 336
Kode 9.4 `loans.py`: hitungan yang sama dalam Python. 339
Kode 9.5 `LoanAdvisor.java`: penasihat pinjaman fuzzy yang adil dan menjelaskan dirinya. 342
Kode 9.6 `loan_advisor.py`: penasihat yang sama dalam Python. 348
Kode 10.1 `db_tour.py`: basis pengetahuan disimpan di SQLite, dilihat dengan SQL, lalu dimuat kembali. 368
Kode 10.2 `DbTour.java`: penjelajahan basis data yang sama dalam Java, melalui JDBC. 369
Kode 10.3 `advisor.py`: logika penasihat noda, dengan pengetahuannya di SQLite. 373
Kode 10.4 `Advisor.java`: logika yang sama dalam Java, melalui JDBC. 379
Kode 10.5 `advisor_demo.py`: kelima belas permintaan tur, diputar ulang pada Advisor. 389
Kode 10.6 `AdvisorDemo.java`: tur yang sama dalam Java. 390
Kode 10.7 `LaundryServer.java`: server web dalam Java, di atas `HttpServer` bawaan JDK. 395
Kode 10.8 `laundry_app.py`: server web yang sama dalam Python, di atas Flask. 399
Kode 10.9 `index.html`: halaman penasihat, yang menyimpan jawaban-jawaban dan berbicara dengan server. 402
Kode 10.10 `kb.html`: halaman pengetahuan, berisi angka-angka minggu ini, pohon, perubahan terakhir, dan daftar penanganan. 405
Kode 10.11 `style.css`: tampilan kedua halaman. 407
Kode 11.1 `lost_update.py`: dua pakar mengubah satu berkas bersama, dan satu pelajaran lenyap. 424
Kode 11.2 `LostUpdate.java`: pelajaran yang hilang yang sama, dalam Java. 425
Kode 11.3 `team_kb.py`: basis pengetahuan tim, dengan versi, pengujian, tinjauan, penghentian, rollback, dan jejak audit yang dirangkai. 427
Kode 11.4 `TeamKb.java`: basis pengetahuan tim dalam Java. 437
Kode 11.5 `triage_team.py`: penjalan perintah, untuk skrip, perintah yang diketik, dan konsultasi pertanyaan demi pertanyaan. 451
Kode 11.6 `TriageTeam.java`: penjalan perintah yang sama dalam Java. 454
Kode 12.1 `cart.py`: pohon keputusan CART dengan aritmetika Gini yang tepat dan pengujian leave-one-out. 478
Kode 12.2 `Cart.java`: pembelajar CART yang sama dalam Java. 484
Kode 12.3 `tree_to_kb.py`: setiap daun pohon menjadi node tingkat atas, dengan lintasannya sebagai kondisi dan hewan pertamanya sebagai cornerstone. 492
Kode 12.4 `TreeToKb.java`: pengubahan yang sama dalam Java. 493
Kode 12.5 `refine_zoo.py`: pelajaran-pelajaran sang guru, masing-masing dicoba pada salinan dan diperiksa terhadap setiap hewan yang sudah diketahui sebelum disimpan. 497
Kode 12.6 `RefineZoo.java`: pelajaran-pelajaran sang guru dalam Java. 499
Kode 12.7 `sklearn_check.py`: pohon kita dibandingkan dengan `DecisionTreeClassifier` milik scikit-learn. 505
Kode 13.1 `OllamaClient.java`: klien minimal untuk API obrolan Ollama. 519
Kode 13.2 `ollama_client.py`: klien yang sama dalam Python, hanya pustaka standar. 522
Kode 13.3 `AskOllama.java`: langkah 1, satu pertanyaan kepada model. 523
Kode 13.4 `ask_ollama.py`: langkah 1 dalam Python. 524
Kode 13.5 `stub_ollama.py`: pengganti Ollama yang menjawab dari buku frasa. 526
Kode 13.6 `StubOllama.java`: server pengganti yang sama dalam Java. 529
Kode 13.7 `chat_doctor.py`: dokter obrolan. Model memahami dan merangkai kalimat; basis pengetahuan memutuskan; penjaga memeriksa. 534
Kode 13.8 `ChatDoctor.java`: dokter obrolan dalam Java. 541
Kode 14.1 `builder.py`: semua yang dapat dilakukan orang dengan basis pengetahuan, tanpa web. 564
Kode 14.2 `Builder.java`: pembangun yang sama dalam Java. 569
Kode 14.3 `builder_demo.py`: sesi Bu Wati, diperankan melalui pembangun. 575
Kode 14.4 `BuilderDemo.java`: sesi yang sama dalam Java. 577
Kode 14.5 `pages.py`: halaman-halaman HTML, disusun dari basis pengetahuan dan kata-kata di `ui-text.json`. 584

Kode 14.6 `Pages.java`: halaman-halaman yang sama dalam Java, bita demi bita. 592
Kode 14.7 `builder_app.py`: server web pembangun dengan Flask. 600
Kode 14.8 `BuilderServer.java`: server yang sama dengan HttpServer bawaan JDK. 605
Kode 15.1 `advice_service.py`: dokter kue sebagai layanan saran kecil dengan catatan pemakaian yang menjaga privasi. 625
Kode 15.2 `AdviceService.java`: layanan yang sama dalam Java. 630
Kode 15.3 `cake_doctor.py`: satu program dengan empat perintah, titik masuk berkas `.pyz` 636
Kode 15.4 `CakeDoctor.java`: keempat perintah yang sama dalam Java. 636
Kode 15.5 `build_pyz.py`: mengemas layanan, alat-alatnya, dan vcirs-core menjadi satu berkas `.pyz` 637
Kode 15.6 `service_check.py`: pemeriksaan setelah setiap penerapan, dan pemutaran ulang konsultasi yang tercatat. 641
Kode 15.7 `ServiceCheck.java`: pemeriksaan yang sama dalam Java. 642
Kode 15.8 `backup.py`: menyimpan, memeriksa, dan memulihkan cadangan basis pengetahuan, dengan sidik jari SHA-256. 646
Kode 15.9 `Backup.java`: alat pencadangan yang sama dalam Java. 649
Kode 15.10 `usage_report.py`: laporan pemakaian bulanan, dan penghapusan baris catatan yang lama. 654
Kode 15.11 `UsageReport.java`: laporan yang sama dalam Java. 657

Daftar Persamaan

(2.1) 27
(2.2) 27
(2.3) 28
(2.4) 34
(6.1) 200
(6.2) 200
(6.3) 201
(6.4) 201
(6.5) 201
(6.6) 201
(7.1) 246
(7.2) 246
(7.3) 246
(7.4) 246
(7.5) 246
(8.1) 293
(8.2) 293
(9.1) 327
(9.2) 328
(9.3) 329
(9.4) 330
(9.5) 330
(12.1) 475
(12.2) 475
(12.3) 476
(12.4) 477
(14.1) 562

(14.2).....562

Kata Pengantar

Pemilik konter ponsel yang tahu pertanyaan mana yang harus diajukan lebih dulu, perawat puskesmas yang tahu demam mana yang tidak boleh ditunda, mekanik di pangkalan ojek yang mendengar klakson lemah dan langsung menduga akinya: masing-masing membawa pengetahuan yang dapat ditulis sebagai *rule*. *Jika ini dan ini, maka itu — kecuali hal lain itu benar.* Tidak seorang pun mempelajari *rule* itu dari buku. *Rule* itu tumbuh satu kasus demi satu kasus, setiap kali *rule* lama bertemu pelanggan, pasien, atau sepeda motor yang tidak cocok dengannya. Buku ini membahas program komputer yang menyimpan pengetahuan semacam itu, menjelaskan setiap jawaban yang diberikannya, dan terus tumbuh dengan cara yang sama: kasus demi kasus, dari orang yang lebih tahu, tanpa merusak apa yang sudah berjalan baik.

Metode yang menjadi pusatnya adalah **Variable-Centered Intelligent Rule System**, disingkat VCIRS. Metode ini diusulkan dalam sebuah tesis magister bidang ilmu komputer dan rekayasa informasi yang diselesaikan pada Juni 2005, pertama kali diterbitkan pada *Information and Communication Technology Seminar* (ICTS) di ITS, Surabaya, pada Agustus 2005, lalu direvisi untuk ICTS 2006, pada tahun yang sama ketika edisi berbahasa Indonesianya ditulis untuk Jurusan Teknik Informatika ITS. Mahasiswa ITS kemudian memakainya untuk membangun sistem pakar yang sungguhan: VCIRS *fuzzy* yang membantu lulusan sekolah memilih jurusan, dan sistem pakar kesehatan untuk pemeriksaan awal secara mandiri atas keluhan yang umum. Gagasannya mudah dinyatakan. Sistem berbasis *rule* klasik menjawab dengan baik dan dapat menjelaskan dirinya, tetapi *rule*-nya hanya dapat diubah oleh pakar bersama seorang *knowledge engineer*. *Ripple Down Rules* memungkinkan pakar mengajari sistem satu kasus demi satu kasus, tetapi hanya menjawab dengan satu cara yang sederhana. VCIRS menyatukan keduanya: ia menyimpan pengetahuannya di sekitar **variabel** yang dibicarakan pakar, dikelompokkan menjadi **node** dan ***rule***; ia belajar kasus demi kasus; ia menghitung seberapa sering setiap variabel, node, dan *rule* dipakai; ia memakai hitungan itu untuk mengurutkan apa yang penting dan untuk mengusulkan *rule* baru yang lebih umum; dan ia dapat bernalar maju dan mundur serta menjawab *apa*, *bagaimana*, dan *mengapa*. Buku ini menceritakan kembali riset itu dari awal, ditulis ulang dari dasar untuk tahun 2026, dan mengubahnya menjadi aplikasi yang dapat dipakai orang setiap hari.

Untuk siapa buku ini ditulis

Anda dianggap belum tahu apa-apa. Bukan “belum tahu apa-apa tentang sistem pakar” — benar-benar belum tahu apa-apa. Jika Anda belum pernah menulis sebaris kode, belum pernah memasang bahasa pemrograman, dan belum pernah mendengar istilah “basis pengetahuan”, Andalah pembaca yang dibayangkan saat buku ini ditulis. Pengemudi becak yang demam dan bertanya-tanya apakah ia masih boleh bekerja, keluarga yang tagihan listriknya merayap melewati empat ratus ribu rupiah, petani durian yang memandangi bercak gelap dan basah di batang pohonnya, pengemudi ojek yang motornya tidak mau menyala pada pukul enam pagi, penjual siomay yang bersama putrinya memikirkan jurusan apa yang sebaiknya diambil selepas sekolah: mereka semua hadir di halaman-halaman ini, dan masing-masing semestinya dapat mengikuti bab yang ditulis tentang masalahnya. Setiap istilah diperkenalkan dengan kata-kata sederhana sebelum dipakai, setiap metode lebih dulu dikerjakan dengan tangan pada contoh kecil, dan setiap program dalam buku ini sudah dijalankan sebelum keluarannya dicetak di sini.

Mengapa sistem pakar yang belajar kasus demi kasus penting bagi pembaca seperti mereka? Karena pengetahuan yang paling menolong mereka bersifat setempat, dan pengetahuan itu berubah. Saran yang cocok untuk skuter di Surabaya, bedengan cabai di lereng desa, gerobak cilok di depan gerbang sekolah, atau antrean klinik pada Senin pagi diketahui oleh seseorang di dekatnya, dan tidak pernah selesai. Program yang harus ditulis ulang oleh pemrogram setiap kali sarannya keliru akan segera usang lalu ditinggalkan. Program yang dapat dikoreksi dalam satu menit oleh mekanik, perawat, atau pemilik toko, dengan kasus yang ada di depan mereka, tetap berguna selama bertahun-tahun, dan menyimpan catatan tertulis tentang alasan setiap sarannya.

Pembaca yang sudah bisa memprogram akan menemukan separuh lain buku ini: aplikasi yang utuh dan teruji, lengkap dengan struktur data, algoritma, pengukuran, dan pengujiannya, siap disesuaikan untuk toko, kebun, atau klinik yang sesungguhnya. Prinsip itu membuat buku ini tebal. Buku yang lebih tipis akan menganggap Anda sudah menguasai bagian-bagian yang sulit, dan buku semacam itu tidak berguna bagi orang yang belum menguasainya.

Satu gagasan, dua bahasa

Setiap aplikasi dalam buku ini dibangun dua kali, dalam **Java** dan dalam **Python**, dan kedua versi itu mengerjakan hal yang sama, membaca berkas data yang sama, dan mencetak keluaran yang sama. Java adalah bahasa sistem besar yang berumur panjang, bahasa ponsel Android, dan bahasa banyak layanan bank serta pemerintahan; kompilernya menangkap banyak kesalahan sebelum program dijalankan. Python adalah bahasa percobaan cepat, pengolahan data, dan sebagian besar riset kecerdasan buatan; ringkas dan mudah dibaca. Melihat satu gagasan ditulis dengan dua cara adalah jalan tercepat untuk memahami gagasan itu sendiri, bukan sekadar kebiasaan satu bahasa. Jika Anda belum mengenal keduanya, mulailah dengan Python, lalu baca versi Java-nya sesudah itu; perbandingan itu akan mengajari Anda Java dengan cepat.

Keduanya ditulis dan dijalankan di **Visual Studio Code**, editor gratis yang bekerja dengan cara yang sama di Windows, macOS, dan Linux. Bab 1 memasangnya, bersama JDK 25 dan Python 3.14, langkah demi langkah pada ketiga sistem itu. Hampir semua hal lain sudah tersedia dalam kedua bahasa itu sendiri: SQLite menyimpan basis pengetahuan bila sebuah bab memerlukan basis data, *web server* kecil bawaan tiap bahasa (atau Flask untuk Python) menyajikan halaman bila sebuah bab memerlukan peramban, dan di Bab 13 sebuah model bahasa berjalan di komputer Anda sendiri melalui Ollama, tanpa mengirim pesan siapa pun ke mana pun.

Satu mesin: vcirs-core

Sejak Bab 3, setiap aplikasi berdiri di atas satu mesin bersama, **vcirs-core**, yang disediakan buku ini dua kali: sebagai paket Java dan sebagai paket Python, dengan kelas yang sama, perilaku yang sama, teks keluaran yang sama, dan berkas simpanan yang sama. Basis pengetahuan yang diajarkan melalui versi Java sebuah aplikasi dapat dibuka oleh versi Python-nya dan memberikan jawaban yang sama. Mesin ini menyimpan pengetahuan dalam tiga struktur VCIRS — variabel, node, dan *rule* — belajar dari kasus dengan pilihan-pilihan dari revisi 2006, menghitung *Variable Usage Rate*, *Node Usage Rate*, dan *Rule Usage Rate*, membangkitkan kandidat *rule*, dan bernalar maju serta mundur dengan penjelasan. Mesin ini cukup kecil untuk dibaca, dan setia pada tesisnya: ketika dijalankan pada kasus-kasus Zookeeper yang dipakai tesis itu sebagai peragaan, mesin ini menghasilkan kembali tabel-tabel tesis tersebut, angka demi angka. Bab 2 dan 3 terlebih dahulu membangun mesin yang lebih sederhana dengan tangan, agar tidak ada bagian vcirs-core

yang terasa seperti sulap ketika kita beralih memakainya, dan setiap bab sesudahnya membuka tepat bagian mesin yang diperlukannya.

Isi buku ini

Bab 1 melihat *rule* yang sudah mengatur sebuah pagi biasa, menjelaskan apa yang membuat sekumpulan *rule* menjadi cerdas dan bagaimana tiga keluarganya — sistem berbasis *rule* klasik, *Ripple Down Rules*, dan VCIRS — saling berkaitan, lalu menyiapkan meja kerja: VS Code, JDK 25, dan Python 3.14 di Windows, macOS, dan Linux. Bab ini ditutup dengan cicipan pertama VCIRS, sebuah basis pengetahuan mungil yang mempelajari pengecualian atas *rule*-nya sendiri.

Bab 2 sampai 5 meletakkan dasarnya. Bab 2 membantu Bu Sri menjaga tagihan listrik keluarganya di bawah empat ratus ribu rupiah dengan program *rule* pertama dalam buku ini: saran IF–THEN yang disimpan terpisah dari program, masing-masing dengan alasan dan penghematannya dalam rupiah. Bab 3 membangun pemecah masalah ponsel dan Wi-Fi untuk konter Mas Anto di pasar dengan *Ripple Down Rules*, mula-mula dengan tangan lalu di atas vcirs-core, sehingga ia mempelajari perbaikan yang benar setiap kali sarannya gagal. Bab 4 membantu Laras dan guru BK-nya, Bu Wati, memikirkan pilihan jurusan, dan sekaligus memperkenalkan jantung VCIRS: saran yang diubah menjadi variabel, nilai, kondisi, node, dan *rule*, yang disimpan dalam *Variable-Centered Rule Structure*. Bab 5 membangun pemeriksa kesehatan mandiri yang menanyakan pertanyaan keselamatan lebih dulu dan menyatakan dengan jelas kapan harus ke dokter, serta memungkinkan perawat puskesmas menyempurnakan pengetahuannya kasus demi kasus dengan *list of difference*, kasus *cornerstone*, dan enam pilihan VCIRS.

Bab 6 sampai 9 membahas apa yang membuat VCIRS cerdas. Bab 6 membantu Pak Slamet menghadapi masalah durian, mangga, cabai, dan padinya, menghitung *Variable*, *Node*, dan *Rule Usage Rate*, lalu memakainya untuk membangkitkan *rule* baru yang lebih umum, yang ditinjau penyuluh pertanian desa sebelum disimpan. Bab 7 mengarahkan pesan WhatsApp sebuah toko kelontong ke orang yang tepat di keluarga pemiliknya, dan membaca *Credit* serta statistik pemakaian sebagaimana seorang pemilik toko membacanya. Bab 8 memberi pengemudi ojek pencari kerusakan motor yang menyingkirkan bahaya lebih dulu, bernalar maju dan mundur atas *rule* IF–THEN yang diturunkan dari basis pengetahuan, dan menjawab *apa*, *bagaimana*, dan *mengapa*. Bab 9 menyiapkan rapat bulanan sebuah koperasi simpan pinjam dengan VCIRS *fuzzy* yang memahami *rendah*, *sedang*, dan *tinggi*, memberi peringatan bila sebuah kasus berada di perbatasan, dan menolak belajar dari atribut yang dilindungi.

Bab 10 sampai 13 membawa pengetahuan itu ke dunia luar. Bab 10 menaruh penasihat penghilang noda sebuah binatu di web, dengan basis pengetahuan di dalam basis data SQLite, yang dilayani oleh *server* Java dan Python yang menjawab dengan *byte* yang sama. Bab 11 memungkinkan tim sebuah klinik kecil merawat bersama basis pengetahuan triase, dengan usulan, pengujian pada kasus emas, tinjauan pakar kedua, versi, dan jejak audit yang dirantai dengan SHA-256. Bab 12 mempelajari *rule* dari data, menumbuhkan pohon keputusan dari tabel hewan lalu mengubahnya menjadi basis pengetahuan VCIRS yang kemudian disempurnakan oleh seorang guru biologi. Bab 13 memungkinkan pelanggan berbicara dengan dokter Wi-Fi dalam kata-kata mereka sendiri melalui model bahasa lokal, sementara basis pengetahuan yang diperiksa manusia mengambil setiap keputusan dan sebuah penjaga menolak balasan apa pun yang tidak didukungnya.

Bab 14 adalah proyek akhir: aplikasi web yang memungkinkan seseorang yang belum pernah memprogram — Bu Wati si pembuat kue, dengan catatan sebelas tahun di sebuah buku tulis —

membangun sistem pakarnya sendiri, mengajarinya kasus demi kasus, membaca statistik pemakaiannya, dan menerima atau mengabaikan *rule* yang diusulkannya. Bab 15 menjaga sistem semacam itu tetap hidup: layanan saran yang dikemas menjadi satu berkas, pemeriksaan otomatis, cadangan yang dapat dipercaya, catatan pemakaian yang merekam saran dan bukan orangnya, laporan bulanan yang memberi tahu apa yang perlu diajarkan berikutnya, dan Undang-Undang Pelindungan Data Pribadi.

Setiap bab menempuh jalan yang dilalui sebuah proyek nyata: masalahnya dan siapa yang dibantu, gagasannya dalam kata-kata sederhana, aplikasinya dibangun langkah demi langkah, menjalankannya, menyempurnakannya, memakainya sungguh-sungguh di toko, kebun, klinik, atau rumah Anda sendiri, kesalahan yang sering terjadi, ringkasan, soal latihan dengan penyelesaian lengkap, dan daftar pustaka. Bila sebuah aplikasi memberi saran tentang kesehatan, uang, atau keselamatan, ia menyatakan dengan jelas kapan harus menemui dokter, bidan, mekanik, atau orang yang berwenang memutuskan.

Setiap program dalam buku ini sudah dijalankan sebelum keluarannya dicetak. Program-program itu dikumpulkan, bab demi bab, dalam folder `Code` yang menyertai buku ini, dari `Chapter01` sampai `Chapter15`, masing-masing dengan folder `java` dan `python`, data yang diperlukannya, dan README yang menjelaskan cara menjalankannya di terminal dan di VS Code, pada Windows, macOS, dan Linux; mesin bersamanya berada di sampingnya, di `vcirs-core`.

Cara membacanya

Dengan VS Code terbuka dan program-programnya berjalan. Bab 1 didahulukan bagi semua pembaca, karena bab itulah yang membangun meja kerja, dan Bab 3 serta 4 sebaiknya dibaca sebelum yang lain, karena keduanya memperkenalkan mesin dan struktur yang dipakai setiap bab sesudahnya; setelah itu bab-bab mendaki dari yang sangat mudah menuju yang sulit, dan setiap bab menyebutkan bab-bab sebelumnya yang menjadi pijakannya, sehingga Anda juga boleh langsung menuju aplikasi yang paling dekat dengan hidup Anda. Ketikkan contohnya, jangan hanya menyalinnya; saat mengetik itulah kesalahan terjadi, dan dari kesalahan itulah pelajaran didapat. Setelah sebuah program berjalan, ajarilah — satu kasus baru, satu koreksi, satu kondisi yang diubah dalam berkas datanya — lalu perhatikan bagaimana sarannya berubah: tidak ada cara yang lebih cepat untuk merasakan bagaimana sebuah sistem pakar yang belajar berpikir. Semua soal latihan di akhir tiap bab dilengkapi penyelesaian lengkap — gunakan sesudah usaha Anda sendiri, bukan sebagai penggantinya.

Gambar, tabel, kode, pseudokode, dan persamaan yang bernomor semuanya adalah tautan hidup: pada edisi elektronik, klik “Kode 5.3” atau “[4]” di bagian mana pun dan Anda langsung dibawa ke sana. Daftar Isi serta daftar-daftar di bagian depan buku bekerja dengan cara yang sama.

Tentang penulis

Irfan Subakti menggambarkan dirinya dalam satu baris — *Father | Educator | Entrepreneur | Bridging Tech & Nature: CS Lecturer × Crypto/Digital Business × AI/ML Specialist × Fruit Farmer | Storyteller* — ayah, pendidik, wirausahawan, penjembatan teknologi dan alam: dosen ilmu komputer, pelaku bisnis kripto dan digital, spesialis AI/ML, petani buah, dan pencerita. Setiap bagian dari baris itu ikut masuk ke dalam buku ini. Ia mengajar di Departemen Teknik Informatika ITS sejak 1998, dan VCIRS adalah risetnya sendiri: ia yang mengusulkannya, menerbitkannya, dan merevisinya, serta membimbing proyek mahasiswa yang pertama kali memakainya. Ia membimbing dan menguji tugas akhir mahasiswa sarjana, mengajar MLOps

dalam program IBM SkillsBuild, dan membangun produk: ia mendirikan Profitina, platform visibilitas AI. Jauh dari papan ketik, ia merawat kebun buah dan durian keluarga — di sanalah ia belajar bahwa perangkat lunak yang baik, seperti kebun yang baik, ditumbuhkan dengan sabar, satu bagian yang terawat demi satu bagian. Jiwa penceritanya adalah alasan setiap bab dalam buku ini dibuka dengan seseorang dan masalahnya, bukan dengan sebuah definisi.

Ucapan terima kasih

Kepada para mahasiswa yang membangun sistem pakar pertama di atas VCIRS dan memperlihatkan di mana ia kuat dan di mana ia masih tipis, serta kepada para mahasiswa yang mempelajari versi-versi awal materi ini dan mengajukan pertanyaan yang menyingkapkan di mana sebuah penjelasan masih perlu diperhalus. Kepada rekan-rekan di ITS yang bersama-sama mengajarkan dan menyempurnakan materi sistem cerdas dari tahun ke tahun. Kepada para penjaga konter, perawat, petani, pemilik toko, pengemudi, mekanik, guru, dan juru masak yang masalah sehari-harinya memberi setiap bab kisahnya. Standar buku ini juga berutang pada dokumentasi resmi Java, Python, SQLite, Flask, VS Code, dan Ollama, serta pada literatur riset tentang sistem berbasis *rule*, *Ripple Down Rules*, logika *fuzzy*, dan pembelajaran mesin, yang dikutip di setiap tempat penggunaannya.

Surabaya, September 2026



Irfan Subakti · yifana@gmail.com

Profitina

profitina.com

Bab 1

Aturan Cerdas dalam Keseharian, dan Meja Kerja Anda

Setelah menyelesaikan bab ini Anda akan mampu mengenali aturan-aturan yang sudah menjalankan hari-hari Anda, menjelaskan dengan kata-kata sederhana apa yang membuat sekumpulan aturan menjadi *cerdas*, dan menjelaskan tiga gagasan yang menjadi landasan seluruh buku ini: aturan yang dapat menjawab pertanyaan dan menjelaskan dirinya sendiri, aturan yang diajarkan kasus demi kasus, dan Variable-Centered Intelligent Rule System (VCIRS) yang menyatukan keduanya. Anda juga akan menyiapkan meja kerja: memasang Java Development Kit 25, Python 3.14, dan Visual Studio Code di Windows, macOS, atau Linux, memeriksa bahwa semuanya berjalan, lalu menjalankan tiga program pertama Anda dalam kedua bahasa. Program terakhir adalah basis pengetahuan kecil yang belajar membuat pengecualian atas aturannya sendiri.

1.1 Pagi yang Penuh Aturan

Pak Darto adalah pengemudi ojek di Sidoarjo. Sebelum berangkat pukul setengah enam pagi, ia menengok ke langit, merasakan udara, melirik ponselnya, dan menekan ban belakang dengan ibu jarinya. Ia tidak menganggap apa yang dilakukannya sebagai "menjalankan aturan", padahal itulah yang sebenarnya terjadi. *Jika langit gelap dan sedang musim hujan, bawa jas hujan. Jika baterai ponsel di bawah tiga puluh persen, isi dulu sebelum berangkat, sebab penumpang menemukannya lewat aplikasi. Jika ban terasa kempis, mampir ke bengkel pertama.* Setiap kalimat itu berbentuk sama: sebuah syarat yang dapat diperiksa, lalu tindakan atau kesimpulan yang mengikutinya.

Dua gang dari situ, Bu Sri berdiri di depan meteran listriknya sambil memegang struk bulan ini. Tagihannya lebih tinggi daripada bulan lalu. Ia pun tahu beberapa aturan: *jika AC menyala semalaman, tagihan naik; jika bohlam lama di dapur masih terpasang, ganti saja.* Namun ia tidak yakin aturan mana yang paling berpengaruh untuk rumahnya sendiri, dan tidak ada yang pernah memberitahunya berapa rupiah yang dapat dihemat oleh setiap perubahan.

Di sudut pasar, Mas Anto membuka konter HP. Sepanjang hari orang datang kepadanya sambil berkata "internetnya mati". Dari pelanggan ke pelanggan ia belajar apa yang biasanya memperbaikinya. *Jika simbol Wi-Fi mati, nyalakan Wi-Fi. Tetapi jika simbol pesawat kecil menyala, matikan dulu mode pesawat. Jika ponselnya baik-baik saja tetapi ponsel lain di rumah itu juga tidak mendapat internet, nyalakan ulang router — kecuali lampu router berwarna merah, sebab berarti kabel dari penyedia layanan yang bermasalah, dan menyalakan ulang tidak akan menolong.* Perhatikan kata *kecuali*. Pengetahuan Mas Anto tidak datang sebagai daftar yang sudah jadi; ia tumbuh, satu pengecualian demi satu, setiap kali seorang pelanggan membuktikan bahwa aturannya yang lama belum lengkap.

Ketiga orang ini adalah tokoh utama bab-bab awal buku ini. Tidak seorang pun dari mereka adalah pemrogram, dan tidak seorang pun perlu menjadi pemrogram untuk memperoleh manfaat dari apa yang akan kita bangun. Kesamaan mereka adalah bahwa keahlian mereka dapat

dituliskan sebagai aturan, dan aturan yang tertulis dapat diberikan kepada komputer, yang kemudian akan menerapkannya dengan setia, setiap kali, pada setiap kasus, sambil menyebutkan *mengapa* ia sampai pada jawabannya. Bila aturan-aturan itu juga dapat *dikoreksi* oleh orang yang lebih tahu, kasus demi kasus, tanpa merusak apa yang sudah benar, maka kita memperoleh jenis sistem yang menjadi pokok bahasan buku ini.

> **WAWASAN KUNCI:** Banyak keahlian sehari-hari sebenarnya sudah berupa sekumpulan aturan di kepala seseorang. Menuliskan aturan itu mengubah keahlian pribadi menjadi sesuatu yang dapat diperiksa, dibagikan, dijelaskan, dan diperbaiki.

1.2 Apa yang Membuat Aturan Menjadi Cerdas

1.2.1 Fakta, kondisi, dan konklusi

Mari kita beri nama bagian-bagian aturan pertama Pak Darto, karena nama-nama ini akan kita pakai di hampir setiap halaman berikutnya.

- **Fakta** adalah sesuatu yang diketahui tentang keadaan saat ini: *langit gelap*, *sedang musim hujan*, *baterai tinggal 25 persen*. Kita menuliskan fakta sebagai sebuah **variabel** dan sebuah **nilai**: `sky = dark`, `battery = 25`.
- **Kondisi** adalah uji terhadap sebuah fakta: `sky = dark` bernilai benar pagi ini; `battery < 30` juga benar. Kondisi memakai sebuah **operator**, salah satu dari `=`, `<>` (tidak sama dengan), `<`, `<=`, `>`, dan `>=`.
- **Konklusi** adalah apa yang dinyatakan aturan bila semua kondisinya benar: `bring = raincoat` (bawa jas hujan).
- **Aturan** (*rule*) menggabungkan semuanya: *JIKA* kondisi-kondisinya terpenuhi, *MAKA* konklusinya berlaku. Dalam bentuk yang dipakai di seluruh buku ini,

```
IF   sky = dark
AND  season = rainy
THEN bring = raincoat
```

Sekumpulan aturan seperti itu tentang satu pokok persoalan disebut **basis pengetahuan** (*knowledge base*), dan program yang mengambil fakta-fakta sebuah kasus, memeriksa aturan-aturan, lalu menarik kesimpulan disebut **mesin inferensi** (*inference engine*). Tempat menyimpan fakta-fakta kasus yang sedang ditangani selama mesin bekerja disebut **memori kerja** (*working memory*). Ketiganya membentuk **sistem berbasis aturan** (*rule-based system*), yang juga disebut **sistem pakar** (*expert system*): program yang memberi saran dalam bidang yang sempit dengan cara seorang pakar memberi saran [1].

1.2.2 Tiga hal yang membuat aturan cerdas

Sederet pernyataan `if` di dalam program belum tentu cerdas. Ada tiga kemampuan yang membedakannya, dan masing-masing akan mendapat satu bab penuh di bagian selanjutnya buku ini.

Ia menjelaskan dirinya sendiri. Tanyakan kepada Pak Darto mengapa ia membawa jas hujan, dan ia akan menjawab: “langitnya gelap, dan ini bulan Desember”. Sistem pakar yang baik melakukan hal yang sama. Ia dapat menjawab tiga pertanyaan tentang setiap kesimpulan [1], [2]: *Apa* (*What*) yang kamu simpulkan? *Bagaimana* (*How*) kamu sampai ke sana — aturan mana yang dijalankan, dengan fakta apa? Dan *Mengapa* (*Why*) — mengapa kamu menanyakan hal ini kepadaku, atau mengapa aturan ini berlaku? Sistem pakar kedokteran klasik MYCIN membuat hal ini terkenal pada tahun 1970-an: seorang dokter dapat mengetik WHY pada pertanyaan mana

pun dan program akan menampilkan aturan yang sedang dicoba dibuktikannya [2]. Penjelasan bukanlah kemewahan. Penjelasanlah yang membuat orang dapat memercayai sebuah saran, atau cepat melihat letak kesalahannya.

Ia belajar dari orang yang tahu. Pengetahuan selalu berubah. Setiap minggu Mas Anto bertemu jenis masalah baru. Dalam sistem berbasis aturan tradisional, menambah aturan berarti meminta bantuan seorang *knowledge engineer* (insinyur pengetahuan) — spesialis yang mewawancarai pakar dan menyunting basis aturan — sambil berharap aturan baru itu tidak merusak aturan lama. Sistem yang memungkinkan Mas Anto sendiri berkata "kamu salah di sini; untuk kasus ini perbaikan yang benar adalah ini, dan inilah yang membedakan kasus ini" belajar tepat di tempat yang membutuhkan, dalam konteks kasus yang nyata [3], [4]. Inilah gagasan **Ripple Down Rules**, yang dibangun dari nol di Bab 3.

Ia makin baik seiring waktu. Sistem yang menghitung fakta dan aturan mana yang paling sering dipakai dapat mengajukan pertanyaan paling berguna lebih dahulu, menunjuk pengetahuan yang paling penting, bahkan mengusulkan aturan baru untuk dikonfirmasi pakar. Inilah bagian yang ditambahkan oleh Variable-Centered Intelligent Rule System [5], [6], dan itulah sebabnya kata *intelligent* (cerdas) ada pada namanya: sistem ini "belajar" dari penggunanya selama mereka membangun dan memakainya.

> **DEFINISI.** *Sistem aturan cerdas* adalah sistem berbasis aturan yang (1) menarik kesimpulan dari fakta dengan aturan JIKA-MAKA, (2) menjelaskan apa yang disimpulkannya, bagaimana, dan mengapa, dan (3) memungkinkan penggunanya memperbaiki pengetahuannya kasus demi kasus, sambil mencatat pengetahuan mana yang dipakai dan berguna.

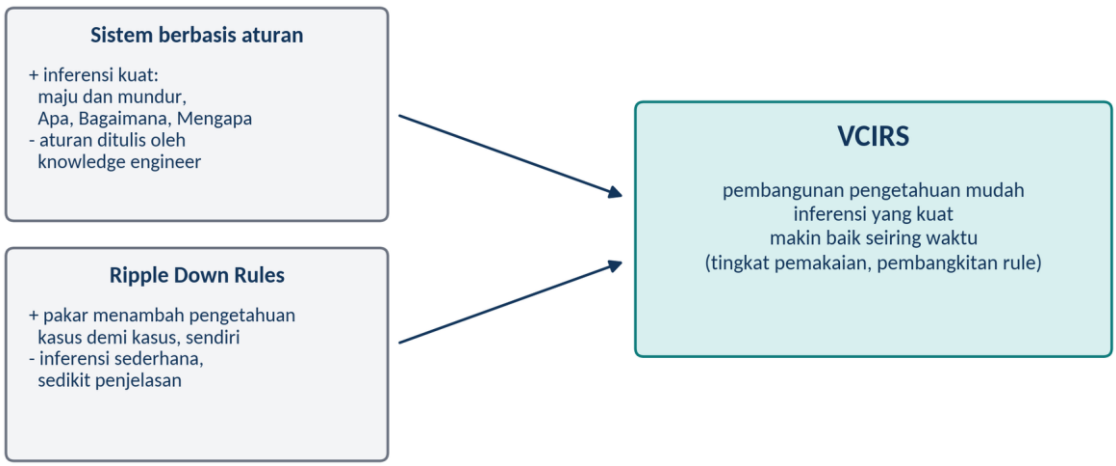
1.2.3 Aturan yang sudah Anda percayai

Aturan semacam ini ada di sekeliling Anda. Ponsel yang meredupkan layarnya ketika baterai lemah mengikuti sebuah aturan. Standar samping sepeda motor yang mematikan mesin bila masih turun saat Anda memasukkan gigi mengikuti sebuah aturan. Sistem bank yang menahan pembayaran kartu yang dilakukan di dua kota dalam sepuluh menit mengikuti sebuah aturan. Apoteker yang tidak mau menjual obat tertentu tanpa resep mengikuti sebuah aturan. Yang berbeda dari satu contoh ke contoh lain adalah berapa banyak aturannya, siapa yang boleh mengubahnya, dan apakah ada yang dapat bertanya *mengapa*.

Cara yang baik untuk menilai apakah sebuah masalah cocok ditangani sistem aturan adalah pertanyaan ini: *dapatkah seorang pakar yang sabar menjelaskan keputusannya kepada pemula dalam kalimat "jika ... maka ..."?* Memecahkan masalah ponsel, memilih tanaman, menentukan pesan pelanggan mana yang mendesak, mencari bagian sepeda motor yang rusak: semuanya memenuhi syarat ini, dan semuanya muncul sebagai aplikasi dalam buku ini. Mengenali wajah dalam foto atau memahami kata-kata yang diucapkan tidak memenuhinya — pengetahuannya tidak berbentuk kalimat — dan untuk itu kita memakai alat lain, beberapa di antaranya dihubungkan dengan sistem aturan di Bab 13.

1.3 Tiga Keluarga Sistem Aturan

Variable-Centered Intelligent Rule System diusulkan sebagai cara menggabungkan dua keluarga sistem aturan yang lebih tua, yang masing-masing kuat justru di tempat keluarga lainnya lemah [5], [6]. **Gambar 1.1** memperlihatkan gagasannya; sisa subbab ini menjelaskannya dengan kata-kata sederhana.



Gambar 1.1 Dua keluarga sistem aturan dan hibrida yang menyatukannya. VCIRS mengambil inferensi dan penjelasan dari sistem berbasis aturan serta pembangunan pengetahuan kasus demi kasus dari Ripple Down Rules, lalu menambahkan statistik pemakaian dan pembangkitan rule.

1.3.1 Sistem berbasis aturan: jawaban kuat, sulit diubah

Sistem berbasis aturan (*rule-based system*, RBS) klasik menyimpan kumpulan aturan JIKA-MAKA yang datar. Mesin inferensinya dapat bekerja dalam dua arah. Dalam **pelacakan maju** (*forward chaining*) ia berangkat dari fakta yang diketahuinya dan menjalankan setiap aturan yang kondisinya terpenuhi, menambahkan setiap konklusi baru ke memori kerja, sampai tidak ada lagi yang dapat disimpulkan; begitulah cara Pak Darto berpikir di pagi hari, dari apa yang dilihatnya ke apa yang harus dilakukannya. Dalam **pelacakan mundur** (*backward chaining*) ia berangkat dari sebuah pertanyaan — *apakah ini harimau?* — lalu mencari aturan yang dapat menyimpulkannya, kemudian berusaha memastikan kondisi-kondisinya, dan hanya bertanya kepada pengguna tentang hal yang tidak dapat diperoleh dari aturan mana pun; begitulah cara dokter bekerja dari dugaan penyakit ke pemeriksaan yang akan memastikannya. Keduanya memberi penjelasan yang kaya: Apa, Bagaimana, dan Mengapa [1].

Kelemahan RBS ada pada pemeliharaannya. Aturan-aturan saling memengaruhi. Aturan baru yang diletakkan di atas daftar dapat menyembunyikan aturan lama; bila diletakkan di bawah, mungkin tidak pernah tercapai. Menjaga agar basis aturan yang besar tetap konsisten adalah pekerjaan yang lambat dan menuntut keahlian, dan itulah sebabnya biasanya diperlukan seorang knowledge engineer [5]. Bab 3 memperlihatkan masalah ini terjadi pada pemecah masalah kecil yang hanya berisi tiga aturan.

1.3.2 Ripple Down Rules: mudah diajari, jawabannya sederhana

Ripple Down Rules (RDR) ditemukan di Sydney pada akhir 1980-an agar para pakar kedokteran dapat memelihara sendiri sebuah sistem pakar patologi [7], [4]. Gagasan intinya: seorang pakar tidak pernah perlu melihat atau menata ulang basis aturan. Sistem dipakai pada kasus-kasus nyata. Ketika sistem memberi jawaban yang salah, sang pakar menyebutkan jawaban yang benar lalu memilih, dari sebuah daftar perbedaan, ciri-ciri yang membedakan kasus ini dari kasus yang dahulu menciptakan aturan yang menjawab salah — yakni **kasus cornerstone** (*cornerstone case*, kasus batu penjuru). Aturan baru dipasang sebagai **pengecualian** (*exception*) tepat di tempat jawaban salah itu berasal, sehingga hanya dapat memengaruhi kasus-kasus yang serupa. Aturan tidak pernah dihapus, hanya diperhalus. Sistem yang dibangun dengan cara ini di Rumah Sakit St Vincent, Sydney, yaitu PEIRS, tumbuh sampai lebih dari dua ribu aturan yang seluruhnya ditambahkan oleh para ahli patologi [5], [4]; metode yang sama kini dipakai dalam sistem

komersial yang menafsirkan hasil laboratorium [4]. Varian berikutnya, Multiple Classification RDR, memungkinkan satu kasus menerima beberapa konklusi sekaligus, misalnya pasien dengan dua masalah yang tidak saling berkaitan [8].

Kelemahan RDR adalah sisi lain dari kekuatannya. Inferensinya hanya berupa penelusuran sederhana menyusuri pohon pengecualian, dan hanya sedikit menawarkan kemampuan menjawab pertanyaan seperti yang dimiliki RBS: tidak ada pelacakan mundur dari sebuah tujuan, dan penjelasannya tidak banyak melampaui "aturan ini dijalankan" [5].

1.3.3 VCIRS: yang terbaik dari keduanya

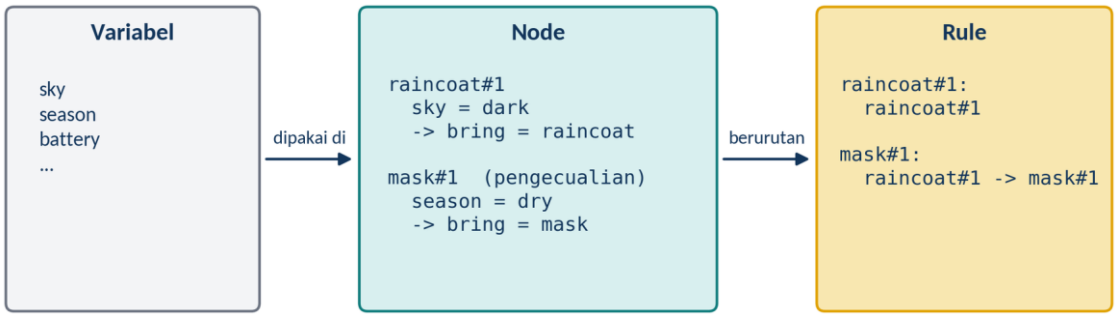
Variable-Centered Intelligent Rule System menyimpan pengetahuannya dalam struktur khusus — *Variable-Centered Rule Structure* — yang melayani kedua tujuan sekaligus [5], [6]. Pengetahuan ditambahkan dengan cara RDR, kasus demi kasus, dengan kasus cornerstone dan pengecualian, disertai enam pilihan yang terdefinisi jelas bagi sang pakar [3]. Pengetahuan yang sama dapat dibaca dengan cara RBS, diubah menjadi aturan JIKA-MAKA biasa, lalu dipakai untuk pelacakan maju dan mundur lengkap dengan penjelasannya. Dan karena strukturnya dibangun di sekitar *variabel*, sistem dapat menghitung seberapa sering setiap variabel, setiap node, dan setiap rule dipakai, mengurutkan mana yang penting, memandu pengguna ke pertanyaan yang paling berguna, dan membangkitkan kandidat rule untuk dikonfirmasi pakar.

Sisa buku ini membangun aplikasi-aplikasi nyata di atas gagasan tersebut, dari yang sangat mudah sampai yang sangat sulit. Sebelum membangun apa pun, mari kita tengok sekali ke dalam strukturnya, agar istilah-istilahnya menjadi akrab.

1.4 VCIRS dengan Kata-Kata Sederhana

1.4.1 Variabel, node, dan rule

Gambar 1.2 memperlihatkan tiga tingkat Variable-Centered Rule Structure pada contoh dua kasus yang akan kita jalankan di akhir bab ini.



Gambar 1.2 Variable-Centered Rule Structure. Variabel dipakai di dalam node; node adalah rangkaian pendek kondisi beserta konklusinya; rule adalah rangkaian node, dimulai dari tingkat teratas.

- **Variabel** adalah satu jenis fakta: `sky` (langit), `season` (musim), `battery` (baterai). Sistem menyimpan satu entri untuk setiap variabel, berapa pun banyaknya rule yang memakainya, dan mengingat di mana saja variabel itu dipakai.
- **Node** adalah rangkaian kecil kondisi dengan satu atau lebih konklusi. Node `raincoat#1` menyatakan: `sky = dark -> bring = raincoat`. Namanya diambil dari konklusi pertamanya, ditambah nomor urut agar setiap nama tetap unik.
- **Rule** adalah rangkaian node. Node pertama sebuah rule berada di tingkat teratas; setiap node berikutnya adalah pengecualian dari node sebelumnya. Rule `mask#1` adalah rangkaian

`raincoat#1 -> mask#1`: *jika langit gelap, bawa jas hujan — kecuali di musim kemarau, bawalah masker*, sebab langit gelap di musim kemarau biasanya adalah asap atau kabut. Sebuah rule diberi nama menurut node terakhirnya, dan konklusinya adalah konklusi node terakhir tersebut.

Beginilah persisnya tesis yang mengusulkan VCIRS mendefinisikan istilah-istilah itu: rule adalah rangkaian satu atau lebih node, node adalah rangkaian satu atau lebih variabel beserta nilainya, dan sebuah node atau rule dapat memiliki satu atau lebih konklusi [5].

1.4.2 Membangun pengetahuan kasus demi kasus

Kasus adalah apa yang dimasukkan pengguna: fakta-fakta dari sebuah keadaan nyata dan, ketika membangun pengetahuan, konklusi yang benar. Ketika menerima sebuah kasus, sistem mula-mula mencari **node yang layak** (*proper nodes*), yaitu node yang memiliki variabel atau konklusi yang sama dengan kasus itu, diurutkan dari node *sempurna* (variabel sama, nilai sama, konklusi sama) sampai node yang sekadar berbagi satu variabel [5]. Kemudian sistem membandingkan kasus baru dengan kasus cornerstone dari node yang dipilih dan mencetak **daftar perbedaan** (*list of difference*) dalam dua bagian: *A*, kondisi pada kasus lama yang tidak dimiliki kasus baru, dan *B*, kondisi pada kasus baru yang tidak dimiliki kasus lama [3]. Sang pakar memilih satu dari enam opsi: menerima pengetahuan yang sudah ada (opsi 0); membuat pengecualian dari negasi bagian A (1), dari bagian B (2), atau dari keduanya (3); atau menyimpan seluruh kasus sebagai node baru di samping node lama (4) atau di tingkat teratas (5) [3]. Bab 3 membahas daftar perbedaan perlahan-lahan, dan Bab 5 memakai setiap opsi pada sebuah aplikasi nyata.

1.4.3 Menghitung apa yang dipakai

Setiap kali sebuah kasus menyetujui variabel dari node yang sudah ada, variabel itu memperoleh satu **Credit**. Dari credit tersebut, dari banyaknya node yang berbagi setiap variabel, dan dari seberapa dekat setiap variabel dengan konklusi node-nya, sistem menghitung tiga **tingkat pemakaian** (*usage rate*): Variable Usage Rate (VUR), Node Usage Rate (NUR), dan Rule Usage Rate (RUR) [5]. Makin besar nilainya, makin sering pengetahuan itu dipakai. Tingkat pemakaian ini memberi tahu pengguna pertanyaan mana yang sebaiknya diajukan lebih dahulu, dan menggerakkan **pembangkitan rule** (*rule generation*): sistem menggabungkan node-node terpenting, dengan urutan yang menghormati rule yang sudah ada, menjadi kandidat rule yang dikonfirmasi atau ditolak oleh pakar. Bab 6 membangkitkan rule untuk petani, dan Bab 7 memanfaatkan statistik ini di sebuah toko kecil.

1.4.4 Menjawab disertai penjelasan

Terakhir, VCIRS dapat menjawab dengan tiga cara [5]. **Cara RDR** berangkat dari node-node yang layak untuk fakta yang ada, memastikan kondisi-kondisinya — menanyakan kepada pengguna apa pun yang belum diketahui — lalu mengikuti pengecualian-pengecualiannya sampai ke konklusi yang paling khusus. Cara **pelacakan maju** dan **pelacakan mundur** terlebih dahulu mengubah struktur itu menjadi aturan JIKA-MAKA biasa, langkah yang disebut **transformasi basis pengetahuan** (*knowledge base transformation*), kemudian melacak melalui aturan-aturan tersebut. Ketiganya mencatat setiap langkah, sehingga dapat mengatakan apa yang disimpulkan, bagaimana, dan mengapa. Bab 8 membangun pencari kerusakan sepeda motor dengan pelacakan maju dan mundur.

Tabel 1.1 memperlihatkan di mana setiap gagasan ini menjadi sebuah aplikasi.

Tabel 1.1 Peta jalan buku ini: setiap bab membangun sebuah aplikasi nyata dan menambahkan satu gagasan VCIRS.

Bab	Aplikasi	Gagasan VCIRS yang dibawanya
1	Aturan pagi dan cicipan pertama (bab ini)	fakta, kondisi, konklusi; meja kerja Anda
2	Penghemat tagihan listrik	aturan JIKA-MAKA sederhana dengan alasan dan penghematan
3	Pemecah masalah ponsel dan Wi-Fi yang belajar	Ripple Down Rules, kasus cornerstone, pengecualian
4	Penasihat pemilihan jurusan	berpikir dengan variabel: inti VCIRS
5	Penasihat cek kesehatan mandiri	enam opsi penyempurnaan kasus demi kasus
6	Penasihat masalah tanaman dan pohon buah	pembangkitan rule
7	Pengarah pesan pelanggan toko kecil	statistik pemakaian dan credit
8	Pencari kerusakan sepeda motor	inferensi maju dan mundur dengan penjelasan
9	Penasihat kelayakan pinjaman koperasi	VCIRS fuzzy
10	Aplikasi web penghilang noda cucian	basis pengetahuan VCIRS di dalam basis data
11	Basis pengetahuan triase klinik untuk tim	banyak pakar, versi, dan audit
12	Belajar rule dari data	dari pohon keputusan ke basis pengetahuan
13	Berbicara dengan sistem pakar Anda	model bahasa lokal di depan pengetahuan yang sudah diperiksa
14	Pembangun sistem pakar VCIRS	semuanya bersama, untuk siapa saja
15	Menerapkan dan merawat	menjaga sistem pakar tetap hidup

1.4.5 Satu mesin, dua bahasa

Setiap aplikasi sejak Bab 3 memakai satu mesin bersama, **vcirs-core**, yang disediakan buku ini dua kali: sebagai paket Java dan sebagai paket Python, dengan kelas yang sama, perilaku yang sama, teks keluaran yang sama, dan berkas simpanan yang sama. Basis pengetahuan yang diajarkan melalui versi Java sebuah aplikasi dapat dibuka oleh versi Python-nya dan memberikan jawaban yang sama. Mesin ini cukup kecil untuk dibaca, dan setia pada tesis yang mendefinisikan VCIRS: ketika dijalankan pada lima belas kasus Zookeeper — basis aturan kecil klasik karya Winston untuk mengenali hewan [9], yang dipakai tesis itu sebagai peragaan — mesin ini menghasilkan kembali tabel-tabel tingkat pemakaian dan rule hasil pembangkitan dalam tesis tersebut, angka demi angka [5]. Bab 2 dan 3 terlebih dahulu membangun versi-versi yang lebih sederhana dengan tangan, agar tidak ada bagian mesin yang terasa seperti sulap ketika kita beralih memakainya.

1.5 Mengapa Java, dan Mengapa Python

Buku ini menulis setiap program dua kali, dan itu disengaja.

Java adalah bahasa sistem-sistem besar yang berumur panjang: perbankan, layanan pemerintah, aplikasi Android, peladen banyak toko daring. Java *dikompilasi*: sebuah alat bernama `javac` memeriksa seluruh program Anda dan menerjemahkannya menjadi *bytecode*, yang kemudian dijalankan oleh Java Virtual Machine di Windows, macOS, maupun Linux. Java mengharuskan Anda menyebutkan tipe setiap variabel, yang mula-mula terasa ketat tetapi kelak menyelamatkan Anda dari berbagai golongan kesalahan. Java 25, yang dirilis pada September 2025, adalah versi **long-term support** (LTS): versi ini menerima pembaruan keamanan dan perbaikan galat selama

bertahun-tahun, itulah sebabnya buku ini memakainya [10], [11]. Versi non-LTS yang lebih baru, Java 27, muncul pada 15 September 2026 [12]; setiap program dalam buku ini juga berjalan di versi itu, dan di versi mana pun sejak 17.

Python adalah bahasa untuk alat-alat cepat, pengolahan data, dan kecerdasan artifisial. Python *diinterpretasikan*: program `python` membaca berkas Anda dan menjalankannya baris demi baris, sehingga tidak ada langkah kompilasi tersendiri. Kodenya pendek dan terbaca hampir seperti bahasa Inggris, sehingga ideal untuk mencoba sebuah gagasan dalam beberapa menit. Python 3.14 adalah versi yang berlaku saat ini; pembaruan terbarunya pada September 2026 adalah 3.14.7, yang dirilis pada Agustus 2026 [13].

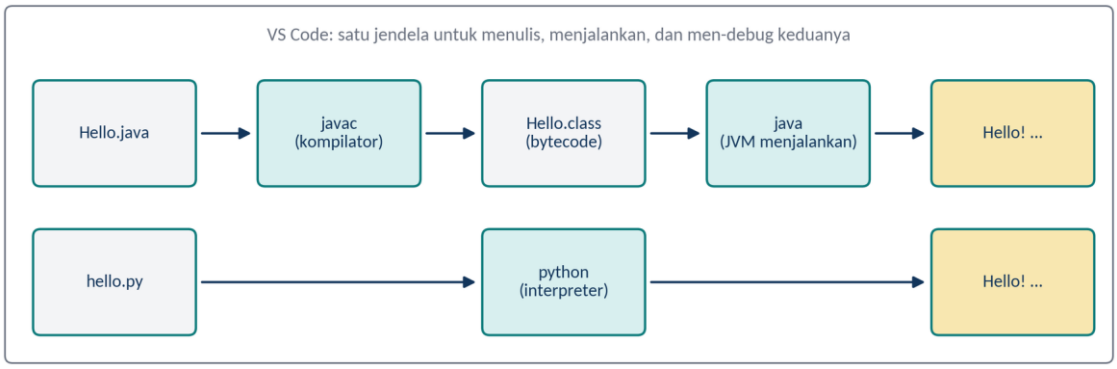
Melihat program yang sama dalam dua bahasa adalah cara tercepat untuk memahami mana yang *gagasan* dan mana yang sekadar *ejaan*. Aturan tetaplah aturan, entah ditulis `if (battery < 30)` atau `if battery < 30:`. Bila Anda dapat membaca keduanya, Anda dapat bergabung dengan tim yang memakai salah satunya, dan Anda dapat memindahkan basis pengetahuan dari purwarupa Python yang cepat ke layanan Java tanpa mengubah satu aturan pun.

1.6 Perkakas di Meja Kerja Anda

Anda membutuhkan empat hal, semuanya gratis.

1. **Java Development Kit (JDK) 25.** JDK berisi kompilator `javac`, program penjalan `java`, dan pustaka standar. Kita memakai **Eclipse Temurin**, distribusi JDK yang gratis dan bersumber terbuka yang dikelola proyek Adoptium. Pembaruan terbarunya pada September 2026 adalah 25.0.4, yang dirilis pada Agustus 2026, disusul pada 2 September 2026 oleh tambalan khusus keamanan, 25.0.4.1 [10], [11].
2. **Python 3.14**, dari python.org atau dari pengelola paket sistem Anda [13].
3. **Visual Studio Code** (VS Code), penyunting gratis dari Microsoft yang berjalan di ketiga sistem. VS Code sering memperbarui dirinya; versi yang dirilis pada 23 September 2026 adalah 1.139 [14].
4. **Dua ekstensi VS Code**: **Extension Pack for Java** dan ekstensi **Python**, keduanya diterbitkan oleh Microsoft [15], [16].

Gambar 1.3 memperlihatkan bagaimana perkakas ini saling berkaitan. Berkas sumber Java seperti `Hello.java` diterjemahkan oleh `javac` menjadi `Hello.class`, yang dijalankan oleh perintah `java`; berkas Python seperti `hello.py` dijalankan langsung oleh `python`. VS Code hanyalah jendela tempat Anda menulis berkas, menekan tombol yang menjalankan perkakas tersebut, dan membaca apa yang dicetaknya.



Gambar 1.3 Meja kerja. Sumber Java dikompilasi oleh javac menjadi bytecode yang dijalankan oleh java; sumber Python dijalankan langsung oleh interpreter python; VS Code menampung keduanya.

Tabel 1.2 merangkum versi-versi yang dipakai untuk menguji setiap program dalam buku ini dan versi terendah yang masih dapat menjalankannya.

Tabel 1.2 Meja kerja buku ini, sebagaimana diperiksa pada September 2026. Nomor pembaruan yang lebih baru selalu dapat dipakai.

Perkakas	Versi dalam buku ini	Juga berjalan dengan	Sumber
JDK (Eclipse Temurin)	25 LTS (25.0.4)	17 ke atas, termasuk 27	adoptium.net
Python	3.14 (3.14.7)	3.10 ke atas	python.org
VS Code	1.139 (September 2026)	versi terbaru mana pun	code.visualstudio.com
Extension Pack for Java	terbaru	—	tampilan Extensions di VS Code
Ekstensi Python	terbaru	—	tampilan Extensions di VS Code

Sepatah kata tentang nomor versi. Java 25 dan Python 3.14 menerima pembaruan berkala yang memperbaiki galat dan masalah keamanan: 25.0.1, 25.0.2, … dan 3.14.1, 3.14.2, … Selalu ambillah pembaruan terbaru dari versi yang disebutkan buku ini. Angka sesudah titik kedua di layar Anda akan lebih besar daripada yang tertulis di bab ini, dan memang seharusnya demikian.

1.7 Memasang di Windows, macOS, dan Linux

Pasang perkakas bahasanya terlebih dahulu, periksa di terminal, lalu pasang penyuntingnya, kemudian hubungkan keduanya. Memeriksa di terminal *sebelum* membuka VS Code itu penting: bila ada yang salah dengan Java atau Python, terminal memberi pesan yang pendek dan jelas, sedangkan penyunting mungkin memberi pesan yang membingungkan. **Terminal** adalah jendela tempat Anda mengetik perintah alih-alih mengeklik: *PowerShell* atau *Command Prompt* di Windows, *Terminal* di macOS dan Linux.

Tabel 1.3 merangkum seluruh prosedur untuk ketiga sistem dalam satu halaman, agar Anda dapat kembali kepadanya kelak; subbab-subbab sesudahnya menjelaskan setiap langkah.

Tabel 1.3 Meja kerja pada tiga sistem operasi, dari komputer kosong sampai penyunting yang siap dipakai. Di Fedora pakai `sudo dnf install java-25-openjdk-devel python3.14`; di Arch Linux, `sudo pacman -S jdk-openjdk python`.

Langkah	Windows 10/11	MacOS	Linux (Ubuntu/Debian)
1. JDK	Temurin 25 `.msi` dari adoptium.net, atau `winget install EclipseAdoptium.Temurin.25.JDK`	Temurin 25 `.pkg` dari adoptium.net, atau `brew install --cask temurin@25`	tambahkan repositori Adoptium, lalu `sudo apt install temurin-25-jdk`
2. Python	penginstal dari python.org (centang **Add python.exe to PATH**), atau `winget install Python.Python.3.14`	penginstal dari python.org, atau `brew install python@3.14`	`sudo apt install python3.14 python3.14-venv` (arsip deadsnakes di Ubuntu)
3. Periksa Java	`java -version`, `javac -version`	sama	sama
4. Periksa Python	`python --version` (atau `py --version`)	`python3 --version`	`python3.14 --version`
5. VS Code	penginstal dari code.visualstudio.com	unduh, seret ke *Applications*	`.deb` dari code.visualstudio.com, atau `sudo snap install code --classic`
6. Ekstensi	tampilan Extensions (`Ctrl+Shift+X`): **Extension Pack for Java**, **Python**	sama, dengan `Cmd+Shift+X`	sama, dengan `Ctrl+Shift+X`

1.7.1 Windows, langkah demi langkah

Pasang JDK.

1. Buka peramban web dan kunjungi **adoptium.net**. Halaman depannya menawarkan rilis LTS Temurin terbaru untuk komputer Anda; pastikan tertulis **25** dan **Windows x64** (atau **aarch64** pada laptop berprosesor ARM), lalu unduh penginstal **.msi** untuk **JDK**.
2. Jalankan penginstal itu. Pada halaman bernama *Custom Setup*, pastikan **Add to PATH** dan **Set JAVA_HOME variable** keduanya akan dipasang: klik masing-masing dan pilih *Will be installed on local hard drive*. *PATH* adalah daftar folder tempat Windows mencari perintah; menambahkan JDK ke dalamnya membuat Anda dapat mengetik `java` di terminal mana pun. Terima pilihan lainnya dan klik **Install**.
3. Bila Anda lebih suka mengetik daripada mengeklik, Windows 10 dan 11 memiliki pengelola paket bernama **winget**. Buka PowerShell dan ketik `winget install EclipseAdoptium.Temurin.25.JDK`; perintah ini mengunduh dan menjalankan penginstal yang sama.

Pasang Python.

4. Kunjungi **python.org**, buka **Downloads → Windows**, dan unduh Python **3.14**. Halaman itu menawarkan *Python install manager*, yang juga memasang perintah `py` untuk mengelola beberapa versi berdampingan, serta penginstal klasik. Keduanya sama baiknya. Dengan penginstal klasik, **centang kotak "Add python.exe to PATH"** di halaman pertamanya sebelum mengeklik *Install Now*; inilah hal yang paling sering terlewat oleh pemula. Dengan install manager, ikuti petunjuknya, dan ketik `py install 3.14` di terminal baru bila ia menanyakan versi mana yang akan dipasang.
5. Cara lain, di PowerShell: `winget install Python.Python.3.14`.

Periksa keduanya.

6. Tutup semua terminal yang sedang terbuka dan buka jendela PowerShell yang *baru*; jendela lama masih memakai PATH yang lama dan tidak akan menemukan perkakas yang baru. Ketik ketiga perintah berikut, tekan Enter setelah masing-masing:

```
java -version
javac -version
python --version
```

Anda akan melihat versi runtime (tiga baris yang diawali `openjdk version "25.0.4"` atau pembaruan yang lebih baru), versi kompilator (`javac 25.0.4`), dan versi Python (`Python 3.14.7`).

Bila `python --version` justru membuka Microsoft Store alih-alih mencetak versi, Windows sedang menjalankan program pengganti kecil yang disebut *app execution alias*, bukan Python yang sebenarnya. Cobalah `py --version`; bila berhasil, ketiklah `py` setiap kali buku ini menulis `python`. Agar `python` sendiri dapat dipakai, buka **Settings → Apps → Advanced app settings → App execution aliases**, matikan dua entri bernama *python.exe* dan *python3.exe*, lalu buka terminal baru.

1.7.2 macOS, langkah demi langkah

Pasang JDK. Kunjungi **adoptium.net**, pilih **Temurin 25**, **macOS**, dan prosesor yang tepat: **aarch64** untuk Mac dengan Apple silicon (M1 dan sesudahnya), **x64** untuk Mac Intel yang lebih lama. Bila ragu, buka menu Apple → **About This Mac**: di sana tertulis *Chip Apple M...* atau *Processor Intel...*. Unduh penginstal **.pkg** dan jalankan. JDK akan dipasang di `/Library/Java/JavaVirtualMachines/`, tempat macOS menemukannya sendiri; tidak ada PATH yang perlu diatur. Dengan pengelola paket **Homebrew**, `brew install --cask temurin@25` melakukan hal yang sama.

Pasang Python. macOS mungkin sudah memiliki perintah `python3`, tetapi sering kali perintah itu milik perkakas pengembang Apple dan versinya lebih tua; jangan mengandalkannya. Kunjungi **python.org → Downloads → macOS**, unduh penginstal **Python 3.14**, lalu jalankan. Di akhir pemasangan, penginstal membuka sebuah folder; klik dua kali **Install Certificates.command** di dalamnya, agar Python dapat membuat sambungan web yang aman. Dengan Homebrew: `brew install python@3.14`.

Periksa keduanya. Buka **Terminal** (tekan `Cmd+Space`, ketik *Terminal*, tekan Enter) dan ketik:

```
java -version
javac -version
python3 --version
```

Di macOS, ketik `python3`, bukan `python`: nama `python` bisa jadi tidak ada atau menunjuk ke tempat yang tidak berguna. Setiap kali buku ini menulis `python`, pengguna Mac mengetik `python3`.

1.7.3 Linux, langkah demi langkah

Distribusi Linux memasang perangkat lunak dari **repositori** melalui pengelola paket. Perintah-perintah di bawah ini untuk Ubuntu dan Debian; **Tabel 1.3** memberikan padanannya untuk Fedora dan Arch.

Pasang JDK. Distribusi Anda mungkin sudah menyediakan `openjdk-25-jdk`; cobalah `sudo apt install openjdk-25-jdk` terlebih dahulu. Bila tidak ditemukan, tambahkan repositori Adoptium, yang selalu memuat rilis Temurin terkini. Ketik baris-baris ini satu per satu; baris yang panjang adalah satu baris:

```
sudo apt install -y wget apt-transport-https gpg
wget -qO - https://packages.adoptium.net/artifactory/api/gpg/key/public | gpg --dearmor |
sudo tee /etc/apt/trusted.gpg.d/adoptium.gpg > /dev/null
echo "deb https://packages.adoptium.net/artifactory/deb $(awk -F=
'/^VERSION_CODENAME/{print$2}' /etc/os-release) main" | sudo tee
/etc/apt/sources.list.d/adoptium.list
sudo apt update
sudo apt install temurin-25-jdk
```

Baris pertama memasang tiga program bantu kecil; baris kedua mengunduh kunci yang dipakai Adoptium untuk menandatangani paketnya, agar sistem Anda dapat memeriksa keasliannya; baris ketiga memberi tahu pengelola paket letak repositorinya; dua baris terakhir menyegarkan daftar paket dan memasang JDK. Ketik kata sandi Anda ketika `sudo` memintanya — tidak ada yang muncul di layar saat Anda mengetik, dan itu wajar.

Pasang Python. Ubuntu menyertakan Python 3 sebagai bagian dari sistem, biasanya bukan versi terbaru, dan Anda tidak boleh sekali-kali menghapus atau mengganti Python milik sistem, karena sistem operasi sendiri memakainya. Pasang 3.14 *di sampingnya*. Di Ubuntu, sumber yang lazim adalah arsip *deadsnakes*:

```
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt update
sudo apt install python3.14 python3.14-venv
```

Sesudah itu `python3.14` menjalankan versi baru dan `python3` tetap menjalankan versi milik sistem. Bila distribusi Anda sudah memiliki Python 3.10 atau yang lebih baru sebagai `python3` — periksa dengan `python3 --version` — Anda boleh langsung memakainya untuk buku ini.

Periksa keduanya dengan `java -version`, `javac -version`, dan `python3.14 --version`. Bila `java -version` melaporkan Java lama yang sudah terpasang sebelumnya, pilih yang baru dengan `sudo update-alternatives --config java`, lalu ulangi untuk `javac`.

1.7.4 VS Code dan kedua ekstensinya

Unduh VS Code dari **code.visualstudio.com** dan pasang: di Windows jalankan penginstalnya dan centang **Add to PATH**; di macOS buka berkas unduhan dan seret *Visual Studio Code* ke *Applications*; di Ubuntu pasang berkas `.deb` (klik dua kali, atau `sudo apt install ./code_*.deb`) atau pakai `sudo snap install code --classic`.

Buka VS Code lalu klik ikon **Extensions** di sebelah kiri (empat kotak kecil), atau tekan `Ctrl+Shift+X` (`Cmd+Shift+X` di Mac). Cari **Extension Pack for Java** dan klik **Install**; lalu cari **Python** dan pasang ekstensi yang diterbitkan Microsoft [15], [16]. Paket Java membawa pelengkapan kode, pemeriksaan galat saat mengetik, tautan **Run** di atas setiap metode `main`, dan debugger; ekstensi Python melakukan hal yang sama untuk Python dan menambahkan tombol ▷ **Run Python File** di kanan atas penyunting.

Bilah status di bagian bawah jendela menunjukkan Python mana yang dipakai VS Code. Klik versi Python yang tertera di sana (atau tekan `Ctrl+Shift+P` dan ketik *Python: Select Interpreter*) dan pilih Python 3.14. Untuk Java, VS Code menemukan JDK melalui `JAVA_HOME` atau PATH; bila ia mengeluh tidak menemukan JDK, jalankan perintah *Java: Configure Java Runtime* dan arahkan ke JDK 25 yang telah Anda pasang.

1.7.5 Kode buku ini di komputer Anda

Setiap program dalam buku ini ada di folder bernama `Code_new`, dengan susunan seperti pada **Tabel 1.4**. Letakkan folder itu di tempat yang sederhana, misalnya `Documents\VCIRS\Code_new` di Windows atau `~/VCIRS/Code_new` di macOS dan Linux, lalu buka di VS Code dengan **File → Open**

Folder…. Biarkan folder-foldernya tetap berdampingan seperti aslinya: program setiap bab menemukan mesin bersama dengan naik dua folder lalu masuk ke `vcirs-core`.

Tabel 1.4 Folder-folder kode buku ini. Setiap folder bab memiliki bentuk yang sama.

Folder	Isinya
`Code_new/vcirs-core/java/src/vcirs`	mesin bersama dalam Java (paket `vcirs`)
`Code_new/vcirs-core/python/vcirs`	mesin yang sama dalam Python (paket `vcirs`)
`Code_new/vcirs-core/data`	kasus-kasus Zookeeper dari tesis dan basis pengetahuannya
`Code_new/ChapterNN/java`	program Java bab NN
`Code_new/ChapterNN/python`	program Python bab NN
`Code_new/ChapterNN/data`	berkas data bab NN
`Code_new/ChapterNN/README.md`	cara menjalankan semua program di bab itu

1.7.6 Bila ada yang tidak beres

Hampir semua masalah pemasangan termasuk salah satu dari segelintir masalah pada **Tabel 1.5**, lengkap dengan cara memperbaikinya. Aturan di balik kebanyakan masalah itu sama: setelah memasang apa pun, bukalah terminal *baru*.

Tabel 1.5 Masalah pemasangan yang lazim dan cara memperbaikinya.

Yang Anda lihat	Penyebabnya	Yang harus dilakukan
`'java' is not recognized…` atau `command not found: java`	JDK tidak ada di PATH	pasang ulang dengan **Add to PATH** dicentang (Windows), atau buka terminal baru
`java -version` menunjukkan 8, 11, atau 17	JDK lama ditemukan lebih dahulu	Windows: naikkan entri JDK 25 di *Environment Variables* → *Path*; Linux: `update-alternatives`; macOS: hapus JDK lama atau atur `JAVA_HOME`
`javac` tidak ditemukan tetapi `java` ada	yang terpasang runtime (JRE), bukan JDK	pasang **JDK**
`python` membuka Microsoft Store	app execution alias	pakai `py`, atau matikan alias-nya (Subbab 1.7.1)
`python3.14: command not found` di Ubuntu	arsip deadsnakes belum ditambahkan	ulangi tiga perintah di Subbab 1.7.3
VS Code: *No Java runtime found*	VS Code tidak melihat JDK	*Java: Configure Java Runtime*, lalu mulai ulang VS Code
VS Code menjalankan Python yang salah	interpreter lain yang terpilih	*Python: Select Interpreter* → 3.14
`error: class Hello is public, should be declared in a file named Hello.java`	nama berkas berbeda dengan nama kelas	beri nama berkas persis seperti nama kelasnya, termasuk huruf besarnya

1.8 Program Pertama Anda dalam Dua Bahasa

Program pertama dalam buku mana pun berguna untuk memeriksa bahwa meja kerja sudah berjalan. **Kode 1.1** adalah program pertama kita dalam Java. Selain menyapa, program ini

menanyakan kepada runtime Java versi berapa dirinya dan menyatakan apakah versi itu cukup baru untuk buku ini.

Kode 1.1 `Hello.java`: program Java pertama; memeriksa apakah Java cukup baru.

```java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello! Java is working on this computer.");
        int version = Runtime.version().feature();   // 25 untuk JDK 25
        if (version >= 17) {
            System.out.println("It is new enough for every program in this book.");
        } else {
            System.out.println("It is too old: please install JDK 25.");
        }
    }
}
```

Bacalah baris demi baris. `public class Hello` mendeklarasikan sebuah **kelas**, satuan tempat Java menyimpan kode; namanya harus persis sama dengan nama berkasnya, termasuk huruf besarnya. `public static void main(String[] args)` adalah **metode main**, tempat setiap program Java mulai berjalan. `System.out.println(...)` mencetak satu baris. `Runtime.version().feature()` menanyakan nomor versi utama Java yang sedang berjalan, misalnya 25, lalu menyimpannya di variabel `version` yang bertipe `int` (bilangan bulat). Kemudian datanglah aturan pertama buku ini, ditulis dalam Java: *jika* versinya paling sedikit 17, katakan bahwa versi itu cukup baru, *jika tidak* mintalah JDK 25.

Untuk menjalankannya, bukalah terminal di folder Java bab ini — di VS Code, **Terminal → New Terminal**, lalu `cd Chapter01/java` (`cd Chapter01\java` di Windows) — dan ketik dua perintah. Perintah pertama mengompilasi berkas; perintah kedua menjalankan kelasnya, disebut tanpa `.java`:

```
javac Hello.java
java Hello
```

```
Hello! Java is working on this computer.
It is new enough for every program in this book.
```

`javac` tidak mencetak apa-apa bila semuanya beres, dan meninggalkan berkas `Hello.class` di samping berkas sumbernya. Anda juga dapat mengeklik **Run** di atas `main` di VS Code, yang melakukan kedua langkah itu untuk Anda, atau menjalankan program satu berkas dalam satu langkah dengan `java Hello.java`.

Kode 1.2 adalah program yang sama dalam Python.

Kode 1.2 `hello.py`: program Python pertama; memeriksa apakah Python cukup baru.

```python
import sys

print("Hello! Python is working on this computer.")
if sys.version_info >= (3, 10):        # (3, 14) untuk Python 3.14
    print("It is new enough for every program in this book.")
else:
    print("It is too old: please install Python 3.14.")
```

Di sini tidak ada kelas dan tidak ada `main`: Python cukup menjalankan berkas dari atas ke bawah. `import sys` menyediakan modul `sys`, dan `sys.version_info` adalah versi Python sebagai sebuah tuple seperti `(3, 14, 7, ...)`. Membandingkannya dengan `(3, 10)` sama dengan bertanya "apakah ini 3.10 atau lebih baru?". Baris-baris yang menjorok ke dalam menjadi milik `if` atau

milik `else`; dalam Python, **indentasi** adalah bagian dari bahasa, bukan hiasan. Jalankan dengan `python hello.py` (`python3 hello.py` di macOS dan Linux) atau dengan tombol ▷:

```
Hello! Python is working on this computer.
It is new enough for every program in this book.
```

Bila kedua program mencetak baris-baris ini, meja kerja Anda sudah siap.

1.9 Aturan-Aturan Pertama Anda

Sekarang mari kita tuliskan pagi Pak Darto sebagai program. **Kode 1.3** menyimpan fakta-fakta satu pagi dalam empat variabel dan memeriksa tiga aturan. Setiap aturan yang terpenuhi mencetak apa yang harus dilakukan dan, di baris berikutnya, *because* — alasannya.

Kode 1.3 `MorningRules.java`: tiga aturan JIKA-MAKA sehari-hari, masing-masing dengan alasannya.

```
public class MorningRules {
    public static void main(String[] args) {
        // Fakta pagi ini. Ubahlah, lalu jalankan programnya lagi.
        String sky = "dark";          // "clear", "cloudy", atau "dark"
        boolean rainySeason = true;   // November-Maret di sebagian besar Jawa
        int batteryPercent = 25;      // baterai ponsel
        int tyrePressure = 26;        // ban belakang, dalam psi

        int fired = 0;
        // Aturan 1
        if (sky.equals("dark") && rainySeason) {
            System.out.println("Rule 1: bring the raincoat.");
            System.out.println("  because the sky is dark and it is the "
                    + "rainy season");
            fired++;
        }
        // Aturan 2
        if (batteryPercent < 30) {
            System.out.println("Rule 2: charge the phone before leaving.");
            System.out.println("  because the battery is at " + batteryPercent
                    + "%, below 30%");
            fired++;
        }
        // Aturan 3
        if (tyrePressure < 29) {
            System.out.println("Rule 3: pump the rear tyre at the first workshop.");
            System.out.println("  because the pressure is " + tyrePressure
                    + " psi, below 29 psi");
            fired++;
        }
        if (fired == 0) {
            System.out.println("No rule fired: have a good ride!");
        }
    }
}
```

Fakta-fakta datang lebih dahulu: sebuah `String` (teks) untuk langit, sebuah `boolean` (benar atau salah) untuk musim, dan dua `int`. Setiap aturan adalah pernyataan `if` yang kondisinya menggabungkan uji-uji dengan `&&`, yaitu *dan* dalam Java. Bila sebuah aturan terpenuhi, program mencetak konklusi dan alasannya lalu menambahkan satu ke `fired`, yang menghitung aturan yang terpenuhi; bila tidak ada satu pun, program mengucapkan selamat jalan. Menjalankannya mencetak:

```
Rule 1: bring the raincoat.
  because the sky is dark and it is the rainy season
Rule 2: charge the phone before leaving.
  because the battery is at 25%, below 30%
Rule 3: pump the rear tyre at the first workshop.
  because the pressure is 26 psi, below 29 psi
```

Kode 1.4 adalah program yang sama dalam Python, tempat *dan* ditulis `and` dan penghitung dinaikkan dengan `fired += 1`.

Kode 1.4 `morning_rules.py`: ketiga aturan yang sama dalam Python.

```
# Fakta pagi ini. Ubahlah, lalu jalankan programnya lagi.
sky = "dark"              # "clear", "cloudy", atau "dark"
rainy_season = True       # November-Maret di sebagian besar Jawa
battery_percent = 25      # baterai ponsel
tyre_pressure = 26        # ban belakang, dalam psi

fired = 0
# Aturan 1
if sky == "dark" and rainy_season:
    print("Rule 1: bring the raincoat.")
    print("  because the sky is dark and it is the rainy season")
    fired += 1
# Aturan 2
if battery_percent < 30:
    print("Rule 2: charge the phone before leaving.")
    print("  because the battery is at " + str(battery_percent)
          + "%, below 30%")
    fired += 1
# Aturan 3
if tyre_pressure < 29:
    print("Rule 3: pump the rear tyre at the first workshop.")
    print("  because the pressure is " + str(tyre_pressure)
          + " psi, below 29 psi")
    fired += 1
if fired == 0:
    print("No rule fired: have a good ride!")
```

Keluarannya persis sama, baris demi baris:

```
Rule 1: bring the raincoat.
  because the sky is dark and it is the rainy season
Rule 2: charge the phone before leaving.
  because the battery is at 25%, below 30%
Rule 3: pump the rear tyre at the first workshop.
  because the pressure is 26 psi, below 29 psi
```

Walaupun kecil, program ini sudah memiliki tiga bagian yang dimiliki setiap sistem aturan dalam buku ini: **fakta** (keempat variabel di atas), **aturan** (ketiga pernyataan `if`), dan **penjelasan** (baris-baris *because*). Apa yang tidak dimilikinya juga sama pentingnya untuk dipelajari. Aturan-aturannya bercampur dengan kode, sehingga mengubah aturan berarti mengubah program. Program ini tidak dapat menjawab pertanyaan seperti “apa yang membuatmu menyuruhku membawa jas hujan?”, yang merupakan pelacakan mundur. Dan program ini tidak dapat belajar: bila Pak Darto mendapati bahwa di bulan Agustus langit gelap berarti kabut asap, bukan hujan, seseorang harus menyunting kode sumbernya. Bab 2 memisahkan aturan dari program; subbab berikut memberi cicipan pertama sebuah basis pengetahuan yang dapat belajar.

> **JEBAKAN:** Dalam Java, teks dibandingkan dengan `.equals`, seperti `sky.equals("dark")`, jangan pernah dengan `==`. Operator `==` menanyakan apakah dua variabel menunjuk ke *objek yang sama* di memori, yang bisa saja bernilai salah walaupun katanya sama. `==` dalam Python membandingkan kata-katanya sendiri, sehingga `sky == "dark"` benar di sana.

1.10 Cicipan Pertama VCIRS

Program terakhir bab ini memakai mesin bersama. Program ini membuat basis pengetahuan kosong, mengajarinya dua kasus, lalu menanyakan kepadanya tentang sebuah pagi yang baru. **Kode 1.5** adalah versi Java-nya.

Kode 1.5 `FirstTaste.java`: basis pengetahuan mempelajari sebuah aturan lalu pengecualiannya.

```
import java.util.List;

import vcirs.Case;
import vcirs.Facts;
import vcirs.InferenceResult;
import vcirs.KnowledgeBase;
import vcirs.LearnResult;

public class FirstTaste {
    public static void main(String[] args) {
        KnowledgeBase kb = new KnowledgeBase("morning");
        // Kasus 1: langit gelap berarti hujan akan turun.
        LearnResult r1 = kb.learn(Case.of(
                List.of("sky = dark"),
                List.of("bring = raincoat")));
        System.out.println("Case 1: " + r1);
        // Kasus 2: di musim kemarau langit gelap hanya asap atau kabut.
        LearnResult r2 = kb.learn(Case.of(
                List.of("sky = dark", "season = dry"),
                List.of("bring = mask")));
        System.out.println("Case 2: " + r2);
        // Tanyakan pagi yang baru kepada basis pengetahuan.
        InferenceResult answer = kb.infer(Facts.of("sky = dark", "season = dry"));
        for (String line : answer.what()) {
            System.out.println(line);
        }
        for (String line : answer.how()) {
            System.out.println(line);
        }
    }
}
```

Baris-baris `import` mendatangkan lima kelas dari paket `vcirs`. `Case.of` membangun sebuah kasus dari dua daftar: fakta-fakta, yang ditulis sebagai `"variabel = nilai"`, dan konklusi yang benar. `kb.learn(...)` adalah seluruh putaran Ripple Down Rules dalam satu panggilan: basis pengetahuan mencoba kasus itu; bila jawabannya sudah benar, kasus itu diterima; bila ada node yang memberi jawaban salah, sebuah pengecualian ditambahkan di bawah node tersebut; bila tidak ada node yang menjawab, node baru ditambahkan di tingkat teratas. Kasus pertama jatuh di tingkat teratas, karena basis pengetahuannya masih kosong; kasus kedua — langit gelap di musim kemarau — dijawab *raincoat* oleh node pertama, yang salah, sehingga kasus itu menjadi pengecualian. Akhirnya `kb.infer(...)` menanyakan tentang pagi dengan langit gelap di musim kemarau; `what()` memberi jawabannya dan `how()` penjelasannya.

Karena program ini memakai mesin, kompilator harus diberi tahu letak berkas-berkas sumber mesin tersebut. Dari folder `Chapter01/java`:

```
javac -d out -sourcepath ../../vcirs-core/java/src *.java
java -cp out FirstTaste
```

`-d out` menaruh semua kelas hasil kompilasi di folder bernama `out`; `-sourcepath` menyebutkan tempat menemukan kelas-kelas `vcirs` yang diimpor program, dan `javac` hanya mengompilasi kelas yang dibutuhkan; `-cp out` (*class path*) memberi tahu `java` letak kelas-kelas hasil kompilasi. Keluarannya:

```
Case 1: added: raincoat#1 added at the top level
Case 2: refined: mask#1 added as an exception of raincoat#1
Conclusion: bring = mask  (rule mask#1, CF 100)
  on the way: bring = raincoat
mask#1 fired because:
  sky = dark  (given, CF 100)
  season = dry  (given, CF 100)
  so bring = mask  [rule CF 100, conclusion CF 100]
```

Bacalah dari atas. Kasus 1 membuat node `raincoat#1` di tingkat teratas. Kasus 2 membuat `mask#1` sebagai pengecualian dari `raincoat#1` — basis pengetahuan menyadari sendiri bahwa jawabannya salah dan meletakkan pengetahuan baru tepat di tempat jawaban salah itu berasal. Jawaban untuk pagi yang baru adalah *bring = mask* (bawa masker), ditemukan melalui rule `mask#1`; "di tengah jalan" (*on the way*) ia melewati konklusi node yang lebih umum, *raincoat*, yang kemudian digantikan oleh pengecualiannya. Penjelasannya mendaftar dua fakta yang membuat rule itu terpenuhi dan tingkat kepastian konklusinya: 100, yang tertinggi, karena kita tidak memberikan fakta yang tidak pasti. **Gambar 1.2** adalah gambar persis basis pengetahuan ini.

Kode 1.6 adalah versi Python-nya. Baris-baris pertamanya memberi tahu Python letak mesinnya, naik dua folder lalu masuk ke `vcirs-core/python`; sesudah itu, isinya terbaca seperti versi Java.

Kode 1.6 `first_taste.py`: cicipan pertama yang sama dalam Python.

```python
import pathlib
import sys

# Beri tahu Python letak mesin vcirs-core (naik dua folder, lalu masuk).
sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import Case, KnowledgeBase

kb = KnowledgeBase("morning")
# Kasus 1: langit gelap berarti hujan akan turun.
r1 = kb.learn(Case.of(["sky = dark"], ["bring = raincoat"]))
print("Case 1: " + str(r1))
# Kasus 2: di musim kemarau langit gelap hanya asap atau kabut.
r2 = kb.learn(Case.of(["sky = dark", "season = dry"], ["bring = mask"]))
print("Case 2: " + str(r2))
# Tanyakan pagi yang baru kepada basis pengetahuan.
answer = kb.infer({"sky": "dark", "season": "dry"})
for line in answer.what():
    print(line)
for line in answer.how():
    print(line)
```

Jalankan dari folder `Chapter01/python` dengan `python first_taste.py`. Keluarannya sama dengan keluaran Java, karakter demi karakter:

```
Case 1: added: raincoat#1 added at the top level
Case 2: refined: mask#1 added as an exception of raincoat#1
Conclusion: bring = mask  (rule mask#1, CF 100)
  on the way: bring = raincoat
mask#1 fired because:
  sky = dark  (given, CF 100)
  season = dry  (given, CF 100)
  so bring = mask  [rule CF 100, conclusion CF 100]
```

> **WAWASAN KUNCI:** Dalam **Kode 1.5** tidak seorang pun menulis pernyataan `if` tentang masker atau kabut asap. Pengetahuannya datang dari sebuah kasus, dan mesinlah yang memutuskan di mana tempatnya. Itulah perbedaan antara memprogram aturan dan mengajarkannya.

1.11 Memakainya Sungguh-Sungguh

Program pagi ini kecil, tetapi kebiasaan yang diajarkannya itulah yang penting: tuliskan keahlian Anda sebagai aturan, dan berikan alasan pada setiap aturan. Berikut beberapa cara untuk memanfaatkannya mulai hari ini.

Pagi Anda sendiri. Gantilah fakta dan aturan Pak Darto dengan milik Anda. Seorang penjual siomay mungkin menulis: *jika hari Jumat, siapkan porsi lebih banyak, karena jamaah masjid membeli selepas salat; jika tabung gas terasa ringan, bawa cadangan; jika semalam hujan, bawa terpal.* Seorang juru parkir mungkin menulis aturan tentang baris mana yang penuh lebih dahulu pada hari pasar. Mulailah dengan tiga aturan, jalankan selama seminggu, dan tambahkan aturan keempat setiap kali kenyataan mengejutkan Anda. Anda akan melihat bahwa kejutan-kejutan itu cenderung berupa *pengecualian* atas aturan yang sudah ada, bukan aturan yang sama sekali baru — tepat itulah alasan Ripple Down Rules dibangun di sekitar pengecualian.

Aturan belanja stok sebuah toko. Toko kecil dapat menulis *jika minyak goreng tinggal kurang dari lima botol dan besok hari pasar, pesan dua karton.* Faktanya berasal dari menghitung rak; konklusinya adalah sebuah pesanan. Bila aturan-aturan itu tersimpan dalam program, keponakan pemilik yang menjaga toko pada hari Minggu menerapkan pengetahuan yang sama dengan sang pemilik.

Cek kesehatan mandiri. VCIRS sudah pernah dipakai untuk membangun aplikasi diagnosis mandiri dari sebuah buku panduan kesehatan rumah tangga, yang menanyakan gejala dan menyarankan langkah pertama [17]. Bab 5 membangun penasihat semacam itu dengan satu aturan yang lebih penting daripada aturan lain mana pun: bila sebuah gejala serius, satu-satunya saran adalah segera menemui dokter atau bidan.

Kebun keluarga. Di kebun buah dan durian, aturan seperti *jika daun durian menguning dari ujungnya dan tanah sudah basah selama seminggu, periksa akarnya dari kemungkinan busuk* adalah jenis pengetahuan yang diubah oleh Bab 6 menjadi sebuah penasihat, dengan aturan-aturan yang diusulkan sendiri oleh sistem untuk dikonfirmasi pekebunnya.

Dalam setiap contoh, mulailah dengan aturan yang Anda yakini, simpan alasannya, dan biarkan kasus-kasus yang Anda temui menunjukkan di mana aturan-aturan itu belum lengkap. Aplikasi-aplikasi di bab-bab berikutnya melakukan persis hal ini, setiap kali dengan sedikit lebih banyak bantuan dari mesin.

1.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Memakai terminal lama setelah memasang. Terminal yang dibuka sebelum pemasangan tidak melihat PATH yang baru. *Hindari:* tutup semua terminal dan buka yang baru setiap kali selesai memasang.

Memasang JRE, bukan JDK. *Runtime* Java dapat menjalankan program tetapi tidak memiliki `javac`. *Hindari:* unduh **JDK** Temurin 25 dan periksa dengan `javac -version`.

Lupa "Add python.exe to PATH" di Windows. Python terpasang, tetapi tidak ada terminal yang dapat menemukannya. *Hindari:* jalankan penginstalnya lagi, pilih *Modify*, lalu centang opsi tersebut; atau pakai perintah `py`.

Mengetik `python` di macOS atau Linux. Nama itu mungkin tidak ada atau menunjuk ke Python lama. *Hindari:* ketik `python3` (atau `python3.14`).

Memberi nama berkas yang berbeda dengan nama kelas. Java mengharuskan `public class Hello` berada di `Hello.java`, dengan huruf besar yang sama. *Hindari:* ganti nama berkas atau kelas agar keduanya cocok.

Menjalankan `java Hello.java` untuk program yang membutuhkan kelas lain. Cara satu langkah hanya mengompilasi satu berkas, sehingga `FirstTaste` gagal karena tidak menemukan `vcirs`. *Hindari:* kompilasi dengan `javac -d out -sourcepath ../../vcirs-core/java/src *.java` dan jalankan dengan `java -cp out FirstTaste`.

Memindahkan folder bab menjauh dari `vcirs-core`. Program-program mencari mesinnya dua folder di atasnya. *Hindari:* biarkan folder `Code_new` tetap utuh.

Membandingkan teks dengan `==` di Java. Operator itu membandingkan objek, bukan kata. *Hindari:* pakai `.equals`.

Mencampur tab dan spasi di Python. Python menentukan baris mana yang menjadi milik sebuah `if` dari indentasinya, dan campuran keduanya membingungkannya. *Hindari:* biarkan VS Code yang mengatur indentasi; ia memakai empat spasi.

Menulis aturan tanpa alasan. Aturan tanpa *because* tidak dapat diperiksa orang lain dan kelak tidak dapat menjelaskan dirinya. *Hindari:* untuk setiap aturan, tuliskan alasannya pada saat yang sama dengan aturannya.

1.13 Ringkasan Bab

- Banyak keahlian sehari-hari sudah berupa **aturan**: JIKA kondisi-kondisi atas **fakta** terpenuhi, MAKA sebuah **konklusi** berlaku. Sebuah kondisi terdiri atas **variabel**, **operator** (`= <> < <= > >=`), dan **nilai**.
- **Sistem berbasis aturan** (sistem pakar) memiliki **basis pengetahuan** berisi aturan, **mesin inferensi** yang menerapkannya, dan **memori kerja** untuk fakta-fakta kasus yang sedang ditangani. Sistem itu *cerdas* bila ia menjelaskan **apa** yang disimpulkannya, **bagaimana**, dan **mengapa**, bila penggunanya dapat memperbaiki pengetahuannya kasus demi kasus, dan bila ia mempelajari pengetahuan mana yang dipakai.
- **Sistem berbasis aturan** klasik berinferensi dengan kuat, melalui **pelacakan maju** (dari fakta ke konklusi) dan **pelacakan mundur** (dari tujuan ke fakta yang dibutuhkannya), tetapi sulit dipelihara. **Ripple Down Rules** memungkinkan pakar menambah pengetahuan kasus demi kasus sebagai **pengecualian** yang diperiksa terhadap **kasus cornerstone**, tetapi inferensinya sederhana (**Gambar 1.1**).
- **VCIRS** menyatukan keduanya dalam **Variable-Centered Rule Structure** yang terdiri atas **variabel**, **node**, dan **rule** (**Gambar 1.2**). VCIRS membangun pengetahuan dengan **daftar perbedaan** dan enam opsi, menghitung pemakaian dengan **Credit**, **VUR**, **NUR**, dan **RUR**, **membangkitkan** kandidat rule, dan berinferensi dengan cara RDR atau dengan pelacakan maju dan mundur setelah **transformasi basis pengetahuan**.
- Setiap aplikasi dalam buku ini memakai mesin bersama **vcirs-core**, ditulis dalam Java dan Python dengan perilaku yang identik; **Tabel 1.1** memperlihatkan gagasan yang ditambahkan setiap bab.
- Meja kerjanya adalah **JDK 25** (Temurin), **Python 3.14**, dan **VS Code** dengan dua ekstensi (**Tabel 1.2**, **Tabel 1.3**); **Tabel 1.5** mendaftar masalah-masalah yang lazim. Java dikompilasi dengan `javac` dan dijalankan dengan `java`; Python dijalankan dengan `python` (**Gambar 1.3**).

- Program-program pertama memeriksa meja kerja (**Kode 1.1**, **Kode 1.2**), menuliskan tiga aturan beserta alasannya (**Kode 1.3**, **Kode 1.4**), dan mengajari basis pengetahuan sebuah aturan beserta pengecualiannya (**Kode 1.5**, **Kode 1.6**).

1.14 Soal Latihan

1.1 Tuliskan lima aturan yang Anda atau anggota keluarga Anda pakai setiap hari, masing-masing dalam bentuk *JIKA ... DAN ... MAKA ..., karena ...*. Untuk setiap aturan, sebutkan variabel, operator, nilai, dan konklusinya.

1.2 Untuk setiap masalah berikut, tentukan apakah masalah itu cocok ditangani sistem aturan dan jelaskan alasannya: (a) memilih pupuk untuk pohon durian yang daunnya menguning; (b) mengenali suara seorang teman di telepon; (c) memutuskan apakah pesanan cucian dapat selesai besok; (d) menebak kurs rupiah besok; (e) memutuskan bus mana dari tiga bus yang paling dahulu tiba di terminal, bila rutenya diketahui.

1.3 Aturan-aturan Pak Darto diterapkan dari apa yang dilihatnya ke apa yang dilakukannya. Tuliskan ulang Aturan 1 dari **Kode 1.3** sebagai pertanyaan untuk **pelacakan mundur**: dari tujuan apa mesin akan berangkat, aturan mana yang akan dicobanya, dan fakta apa saja yang akan ditanyakannya?

1.4 Tambahkan aturan keempat pada **Kode 1.3** dan **Kode 1.4**: *jika penunjuk bensin tinggal kurang dari dua garis, isi bensin di SPBU pertama, karena penumpang mungkin meminta perjalanan jauh.* Biarkan fakta tetap di atas dan alasan di barisnya sendiri. Jalankan kedua program dengan bensin tinggal satu garis.

1.5 Ubahlah fakta-fakta di **Kode 1.4** sehingga tidak ada aturan yang terpenuhi, lalu periksa bahwa program mencetak pesan selamat jalan. Fakta mana yang Anda ubah, dan menjadi apa?

1.6 Dengan memakai **Gambar 1.2**, jelaskan dengan kata-kata Anda sendiri mengapa rule `mask#1` berisi dua node sedangkan rule `raincoat#1` berisi satu node. Konklusi apa yang diberikan masing-masing rule?

1.7 Perluas **Kode 1.6** (atau **Kode 1.5**) dengan kasus ketiga: di musim kemarau, langit gelap *dan* bau hujan berarti hujan juga (`sky = dark`, `season = dry`, `smell = rain`, konklusi `bring = raincoat`). Di mana mesin meletakkan node baru itu, dan apa jawabannya untuk pagi yang gelap, kering, dan berbau hujan?

1.8 Teman Anda memasang Python di Windows, tetapi `python --version` membuka Microsoft Store dan `py --version` mencetak `Python 3.12.4`. Jelaskan apa yang sedang terjadi dan apa yang sebaiknya ia lakukan. Apakah 3.12 cukup untuk buku ini?

1.15 Penyelesaian Soal Latihan

Penyelesaian 1.1. Jawaban dapat berbeda-beda. Aturan yang baik menyebut fakta yang dapat diperiksa dan konklusi yang jelas, misalnya: *JIKA beras di tempayan kurang dari dua cangkir DAN besok hari Minggu MAKA beli lima kilogram hari ini, karena toko tutup hari Minggu.* Variabelnya adalah `sisa beras` dan `hari`; operatornya `<` dan `=`; nilainya `2 cangkir` dan `Minggu`; konklusinya `beli = 5 kg`. Aturan seperti "JIKA saya ingin MAKA memasak" tidak dapat dipakai, karena tidak ada orang lain yang dapat memeriksa kondisinya.

Penyelesaian 1.2. (a) Cocok: pekebun yang berpengalaman dapat menjelaskan pemeriksaannya — kelembapan tanah, pola menguningnya daun, pupuk terakhir — sebagai kalimat JIKA-MAKA;

Bab 6 membangun persis hal ini. (b) Tidak cocok: mengenali suara bukan pengetahuan yang dapat dituangkan orang ke dalam kalimat; pengetahuan itu dipelajari dari contoh dan termasuk wilayah pembelajaran mesin. (c) Cocok: jawabannya bergantung pada berat, jenis layanan, mesin yang kosong, dan waktu, dan pemilik usaha dapat merumuskan aturannya. (d) Tidak cocok: kurs bergantung pada terlalu banyak faktor yang tidak diketahui, dan tidak ada sekumpulan aturan yang dapat diandalkan; tidak ada sistem aturan yang pantas berpura-pura sebaliknya. (e) Sebagian: jawabannya berasal dari perhitungan jadwal, yang lebih baik dikerjakan program biasa daripada aturan, walaupun aturan dapat menyatakan pengecualian seperti “pada hari pasar rute B lambat”.

Penyelesaian 1.3. Tujuannya adalah `bring = raincoat`. Mesin mencari aturan yang konklusinya `bring = raincoat` dan menemukan Aturan 1. Untuk membuktikannya, mesin harus memastikan kondisi-kondisinya secara berurutan: pertama `sky = dark` — tidak ada aturan yang menyimpulkan `sky`, sehingga mesin bertanya kepada pengguna “seperti apa langitnya?” — lalu `season = rainy`, yang juga ditanyakannya (atau dihitung dari bulan, bila ada aturan untuk itu). Bila keduanya terpenuhi, tujuan terbukti; bila pengguna menjawab langitnya cerah, Aturan 1 gagal dan, karena tidak ada aturan lain untuk tujuan itu, jawabannya adalah “tidak terbukti”. Bab 8 membangun penalaran semacam ini.

Penyelesaian 1.4. Tambahkan sebuah fakta dan sebuah aturan. Program lengkapnya adalah `MorningRulesFuel.java` dan `morning_rules_fuel.py` di folder `Chapter01`; versi Java-nya:

```java
public class MorningRulesFuel {
    public static void main(String[] args) {
        // Fakta pagi ini. Ubahlah, lalu jalankan programnya lagi.
        String sky = "dark";          // "clear", "cloudy", atau "dark"
        boolean rainySeason = true;   // November-Maret di sebagian besar Jawa
        int batteryPercent = 25;      // baterai ponsel
        int tyrePressure = 26;        // ban belakang, dalam psi
        int fuelBars = 1;             // garis pada penunjuk bensin

        int fired = 0;
        // Aturan 1
        if (sky.equals("dark") && rainySeason) {
            System.out.println("Rule 1: bring the raincoat.");
            System.out.println("  because the sky is dark and it is the "
                    + "rainy season");
            fired++;
        }
        // Aturan 2
        if (batteryPercent < 30) {
            System.out.println("Rule 2: charge the phone before leaving.");
            System.out.println("  because the battery is at " + batteryPercent
                    + "%, below 30%");
            fired++;
        }
        // Aturan 3
        if (tyrePressure < 29) {
            System.out.println("Rule 3: pump the rear tyre at the first workshop.");
            System.out.println("  because the pressure is " + tyrePressure
                    + " psi, below 29 psi");
            fired++;
        }
        // Aturan 4
        if (fuelBars < 2) {
            System.out.println("Rule 4: fill up at the first petrol station.");
            System.out.println("  because the gauge shows " + fuelBars
                    + " bar, and a passenger may want a long trip");
            fired++;
        }
        if (fired == 0) {
            System.out.println("No rule fired: have a good ride!");
        }
    }
}
```

Dengan `fuelBars = 1` program mencetak keempat aturan; dua baris terakhirnya:

```
Rule 4: fill up at the first petrol station.
  because the gauge shows 1 bar, and a passenger may want a long trip
```

Versi Python menambahkan `fuel_bars = 1` pada fakta-fakta dan aturan yang sama dengan `if fuel_bars < 2:`; keluarannya identik.

Penyelesaian 1.5. Misalnya `sky = "clear"`, `battery_percent = 80`, dan `tyre_pressure = 30`: Aturan 1 membutuhkan langit gelap, Aturan 2 baterai di bawah 30, dan Aturan 3 tekanan di bawah 29, sehingga tidak ada yang terpenuhi dan program mencetak `No rule fired: have a good ride!`. Mengubah musim menjadi `False` saja tidak cukup, karena Aturan 2 dan 3 masih terpenuhi.

Penyelesaian 1.6. Sebuah rule adalah rangkaian node dari tingkat teratas sampai node terakhirnya. `raincoat#1` adalah node tingkat teratas, sehingga rule-nya hanya berisi node itu dan menyimpulkan `bring = raincoat`. `mask#1` dibuat sebagai pengecualian dari `raincoat#1`, sehingga rule-nya adalah `raincoat#1 -> mask#1`: kedua node harus terpenuhi — langit gelap dan musim kemarau — dan konklusinya adalah konklusi node terakhir, `bring = mask`.

Penyelesaian 1.7. Program penyelesaiannya adalah `SecondTaste.java` dan `second_taste.py`. Kasus ketiga dijawab *mask* oleh `mask#1`, yang salah, sehingga `learn` menambahkan `raincoat#2` sebagai pengecualian dari `mask#1` dengan satu kondisi pembeda, `smell = rain`. Keluaran versi Python-nya:

```
Case 1: added: raincoat#1 added at the top level
Case 2: refined: mask#1 added as an exception of raincoat#1
Case 3: refined: raincoat#2 added as an exception of mask#1
Conclusion: bring = raincoat  (rule raincoat#2, CF 100)
  on the way: bring = raincoat, bring = mask
raincoat#2 fired because:
  sky = dark  (given, CF 100)
  season = dry  (given, CF 100)
  smell = rain  (given, CF 100)
  so bring = raincoat  [rule CF 100, conclusion CF 100]
```

Rule `raincoat#2` kini adalah rangkaian `raincoat#1 -> mask#1 -> raincoat#2`: langit gelap, musim kemarau, bau hujan; konklusi-konklusi yang lebih umum, *raincoat* dan *mask*, muncul "di tengah jalan".

Penyelesaian 1.8. Perintah `python` adalah pengganti dari Microsoft Store (sebuah app execution alias), sedangkan `py` menemukan Python yang sungguhan, versi 3.12. Teman Anda dapat memakai `py` di mana-mana, atau mematikan alias-nya (Subbab 1.7.1) dan memastikan folder Python ada di PATH. Python 3.12 lebih baru daripada 3.10, sehingga setiap program dalam buku ini berjalan di sana; memasang 3.14 tetap disarankan, dan `py install 3.14` (atau penginstal dari python.org) melakukannya tanpa menghapus 3.12.

Daftar Pustaka

[1] G. R. Baur and D. V. Pigford, *Expert Systems for Business: Concepts and Applications*. Boston, MA, USA: Boyd & Fraser, 1990.

[2] B. G. Buchanan and E. H. Shortliffe, Eds., *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Reading, MA, USA: Addison-Wesley, 1984.

[3] I. Subakti, "Some revisions in VCIRS and cases reconstructing perspectives," in *Proc. Information and Communication Technology Seminar (ICTS 2006)*, Surabaya, Indonesia, 2006.

[4] P. Compton and B. H. Kang, *Ripple-Down Rules: The Alternative to Machine Learning*. Boca Raton, FL, USA: CRC Press, 2021.

[5] M. M. I. Subakti, "A Variable-Centered Intelligent Rule System," M.Sc.Eng. thesis, Dept. of Computer Science and Information Engineering, Taipei, Taiwan, Jun. 2005.

[6] I. Subakti, "A variable-centered intelligent rule system," in *Proc. 1st Annual Int. Conf. Information and Communication Technology Seminar (ICTS 2005)*, vol. 1, no. 1, Surabaya, Indonesia, Aug. 2005, pp. 167–174.

[7] P. Compton, G. Edwards, B. Kang, L. Lazarus, R. Malor, T. Menzies, P. Preston, A. Srinivasan, and C. Sammut, "Ripple down rules: possibilities and limitations," in *Proc. 6th AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1991, pp. 6.1–6.20.

[8] B. H. Kang, P. Compton, and P. Preston, "Multiple classification ripple down rules: evaluation and possibilities," in *Proc. 9th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1995, pp. 17.1–17.20.

[9] P. H. Winston, *Artificial Intelligence*, 3rd ed. Reading, MA, USA: Addison-Wesley, 1992.

[10] Adoptium, "Eclipse Temurin 8u502, 11.0.32, 17.0.20, 21.0.12, 25.0.4 and 26.0.2 available," Aug. 2026. [Online]. Available: https://adoptium.net/news/2026/08/eclipse-temurin-8u502-11032-17020-21012-2504-2602-available

[11] Adoptium, “Eclipse Temurin 8u504, 11.0.32.1, 17.0.20.1, 21.0.12.1, 25.0.4.1 and 26.0.2.1 available,” Sep. 2, 2026. [Online]. Available: https://adoptium.net/news/2026/09/eclipse-temurin-8u504-110321-170201-210121-25041-26021-available

[12] Oracle, “The arrival of Java 27!,” *Inside.java*, Sep. 15, 2026. [Online]. Available: https://inside.java/2026/09/15/jdk-27-available/

[13] Python Software Foundation, “Python 3.14.7 and 3.13.15 are now available,” *Python Insider*, Aug. 2026. [Online]. Available: https://blog.python.org/2026/08/python-3147-31315/

[14] Microsoft, “Visual Studio Code 1.139,” release notes, Sep. 23, 2026. [Online]. Available: https://code.visualstudio.com/updates/v1_139

[15] Microsoft, “Extension Pack for Java,” Visual Studio Marketplace. [Online]. Available: https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-pack

[16] Microsoft, “Python,” Visual Studio Marketplace. [Online]. Available: https://marketplace.visualstudio.com/items?itemName=ms-python.python

[17] I. Subakti and R. Hidayatullah, “Aplikasi sistem pakar untuk diagnosis awal gangguan kesehatan secara mandiri menggunakan Variable-Centered Intelligent Rule System,” vol. 6, no. 1, pp. 11–16, Jan. 2007.

Bab 2

Penghemat Tagihan Listrik: Program Aturan Pertama Anda

Setelah menyelesaikan bab ini Anda akan mampu menghitung tagihan listrik bulanan sebuah rumah tangga, dengan tangan maupun dengan program, dari daya setiap peralatan, lama pemakaiannya, dan tarif PLN; menuliskan saran penghematan sebagai aturan JIKA-MAKA yang memberikan alasan dan besarnya penghematan untuk setiap saran; memisahkan aturan dari program, sebagai nilai-nilai dalam sebuah daftar yang diterapkan oleh sebuah perulangan kecil pada setiap peralatan; mengerjakan aritmetika uang dengan bilangan bulat agar Java dan Python sepakat sampai ke rupiah terakhir; dan menyesuaikan aplikasinya untuk rumah Anda sendiri, sebuah kamar kos, atau sebuah toko kecil. Aplikasi yang kita bangun, **penghemat tagihan listrik**, membaca daftar peralatan, mencetak tagihan per peralatan, memberikan tiga belas saran untuk rumah Bu Sri, dan memperlihatkan bahwa mengikuti saran-saran itu akan menurunkan tagihannya dari Rp 407.600 menjadi Rp 317.603 sebulan.

2.1 Tagihan Bu Sri

Bu Sri tinggal bersama suami dan dua anaknya yang sudah remaja di sebuah rumah kecil dengan sambungan listrik 1300 VA. Bulan lalu tagihannya sedikit di atas empat ratus ribu rupiah, padahal keluarga itu sudah sepakat untuk menjaganya di bawah empat ratus ribu. Para tetangga punya banyak saran: matikan AC, belilah lampu LED, jangan biarkan rice cooker menghangatkan nasi seharian, cabut steker televisi. Semuanya mungkin benar. Yang tidak dapat dikatakan siapa pun kepadanya adalah *berapa* nilai setiap saran itu di rumahnya, saran mana yang harus dilakukan lebih dahulu, dan apakah melakukan semuanya sudah cukup.

Itulah pekerjaan yang cocok untuk aturan. Seorang teknisi listrik atau penasihat energi yang berkunjung ke rumah berpikir persis seperti ini: *jika ada AC yang disetel 20 derajat, sarankan 25; jika bohlam lama di dapur menyala empat belas jam sehari, sarankan LED dan mematikannya ketika siang; jika rice cooker menghangatkan nasi sepanjang hari, sarankan memasak lebih dekat ke waktu makan.* Setiap aturan berbentuk seperti yang selalu dipakai sistem berbasis aturan — klausa-klausa *JIKA* yang dihubungkan dengan *DAN*, *MAKA* sebuah konklusi [1] — dengan kondisi yang dapat diperiksa pada satu peralatan, sebuah saran, sebuah alasan, dan — karena listrik itu terukur — penghematan yang dapat dihitung. Program yang menyimpan aturan-aturan ini dapat "mengunjungi" setiap rumah di kampung, dan ia tidak pernah melupakan satu aturan pun atau salah berhitung.

Bab ini membangun program itu dalam tiga langkah. Pertama, kita menghitung tagihan tanpa aturan sama sekali, agar yakin angka-angkanya benar. Kemudian kita menuliskan dua saran sebagai pernyataan `if` biasa, seperti cara Bab 1 menuliskan pagi Pak Darto. Terakhir, kita mengeluarkan aturan-aturan itu dari logika program dan menjadikannya **nilai**: sebuah daftar aturan, masing-masing dengan kondisi, saran, alasan, dan akibatnya, yang diterapkan oleh satu perulangan kecil. Langkah terakhir itu — memisahkan pengetahuan dari mesin yang memakainya — adalah gagasan terpenting bab ini, dan setiap bab sesudahnya bergantung padanya.

Ingatlah apa aplikasi ini dan apa yang bukan. Ia adalah penasihat yang berhitung dengan angka yang Anda berikan, dan hasilnya hanya sebaik angka-angka itu. Ia memberi saran, bukan jaminan, dan ia tidak pernah menyentuh instalasi listrik: apa pun yang berhubungan dengan meteran, pemutus arus, atau kabel adalah pekerjaan PLN atau instalatir listrik bersertifikat, dan salah satu aturan kita menyatakannya.

2.2 Gagasannya dengan Kata-Kata Sederhana

2.2.1 Daya, waktu, dan energi

Setiap peralatan memiliki **daya** yang tertulis pada labelnya dalam satuan **watt** (W): lampu LED kecil memakai 9 W, kipas angin 45 W, rice cooker 350 W ketika memasak. Daya menyatakan seberapa cepat peralatan memakai listrik selama menyala. Berapa banyak yang dipakainya dalam sebulan bergantung juga pada **berapa lama** ia menyala. Kalikan daya dengan jam dan Anda memperoleh **energi** dalam **watt-jam** (Wh); seribu watt-jam sama dengan satu **kilowatt-jam** (kWh), satuan yang tertera pada meteran listrik dan pada tagihan.

Untuk satu jenis peralatan yang dipakai h jam sehari, sebanyak n buah, masing-masing berdaya P watt, energi dalam sebulan yang terdiri atas 30 hari adalah

$$E = \frac{P \times n \times h \times 30}{1000} \text{ kWh} \tag{2.1}$$

Mari kita coba dengan tangan pada AC Bu Sri. Labelnya menyebut 400 W; jumlahnya satu; ia menyala 9 jam setiap malam. Menurut Persamaan (2.1), $E = 400 \times 1 \times 9 \times 30/1000 = 108{,}000/1000 = 108$ kWh sebulan. Enam lampu LED-nya yang masing-masing 9 W dan menyala 7 jam sehari memakai $9 \times 6 \times 7 \times 30/1000 = 11.34$ kWh — kurang dari sepersepuluh satu AC.

2.2.2 Siaga: daya saat “mati”

Banyak peralatan tidak pernah benar-benar mati selama stekernya tercolok. Televisi menunggu sinyal dari remote, mesin cuci menghidupkan layar kecilnya, pengisi daya ponsel yang dibiarkan tercolok tetap hangat. Daya **siaga** (*standby*) ini kecil — sekitar 1 W dalam contoh kita, meskipun beberapa peralatan lama memakai lebih banyak — tetapi berlangsung selama semua jam ketika peralatan itu *tidak* dipakai. Untuk peralatan dengan daya siaga S watt, energi sebulan adalah

$$E = \frac{(P \times h + S \times (24 - h)) \times n \times 30}{1000} \text{ kWh} \tag{2.2}$$

AC tadi memiliki daya siaga 1 W selama 15 jam ketika mati, yang menambah $1 \times 15 \times 30/1000 = 0.45$ kWh sebulan, sehingga total sebenarnya adalah 108.45 kWh.

2.2.3 Dari energi ke rupiah

Harga satu kWh bergantung pada **tarif** pelanggan, yang bergantung pada daya sambungan dalam **volt-ampere** (VA) — 450, 900, 1300, 2200 VA, dan seterusnya — dan pada apakah rumah tangga itu menerima subsidi. Pemerintah meninjau tarif rumah tangga setiap tiga bulan. Untuk Juli sampai September 2026 tarif itu tidak diubah [2], [3]; **Tabel 2.1** memperlihatkannya. Pelanggan pascabayar membayar tarif ini untuk kWh yang dihitung meteran; pelanggan prabayar membeli token dengan tarif yang sama.

Tabel 2.1 Tarif listrik rumah tangga PLN untuk Juli–September 2026, tidak berubah pada triwulan tersebut [2], [3].

Kode dalam berkas kita	Tarif	VA	Rp per kWh
`R1-450S`	R-1/TR 450 VA, bersubsidi	450	415,00
`R1-900S`	R-1/TR 900 VA, bersubsidi	900	605,00
`R1-900`	R-1/TR 900 VA, nonsubsidi	900	1.352,00
`R1-1300`	R-1/TR 1300 VA	1300	1.444,70
`R1-2200`	R-1/TR 2200 VA	2200	1.444,70
`R2-3500`	R-2/TR 3500–5500 VA	3500	1.699,53
`R3-6600`	R-3/TR 6600 VA ke atas	6600	1.699,53

Tagihan untuk energi E dengan tarif r rupiah per kWh adalah

$$B = E \times r \tag{2.3}$$

yang dibulatkan ke rupiah utuh. Bu Sri memakai `R1-1300`, sehingga AC-nya menelan biaya $108.45 \times 1444.70 = 156{,}677.7$, sekitar Rp 156.678 sebulan. Tagihan sungguhan juga dapat memuat Pajak Penerangan Jalan, yang ditetapkan oleh setiap kabupaten atau kota, dan bea meterai kecil; tagihan pascabayar memiliki tagihan minimum untuk rumah yang sangat sedikit memakai listrik. Subbab 2.8 menunjukkan cara menambahkannya. Semua hal lain dalam bab ini berasal dari Persamaan (2.1), Persamaan (2.2), dan Persamaan (2.3).

2.2.4 Watt, volt-ampere, dan pemutus arus

Angka pada sambungan, 1300 VA untuk Bu Sri, bukanlah jumlah energi; angka itu adalah batas *daya*: seberapa besar yang boleh dipakai rumah pada satu saat. **Pemutus arus** di kotak meteran, MCB, memadamkan rumah bila batas itu terlampaui, dan itulah yang dimaksud orang ketika berkata “listriknya turun” setelah menyalakan setrika dan pompa bersamaan. Volt-ampere dan watt mengukur hal yang hampir sama untuk lampu atau ketel, dan sedikit kurang sama untuk peralatan bermotor, seperti kulkas atau pompa air, yang wattnya agak di bawah volt-amperenya. Untuk peringatan rumah tangga, menganggap satu watt sama dengan satu volt-ampere sudah cukup dekat, dan untuk kebanyakan peralatan justru berada di sisi yang aman.

Batas ini memengaruhi tagihan hanya secara tidak langsung: tarif dengan VA lebih tinggi dapat lebih mahal per kWh, dan sambungan yang sering turun menggoda orang meminta daya yang lebih besar. Sering kali jawaban yang lebih murah adalah tidak menyalakan peralatan-peralatan berat pada saat yang sama, dan itulah persisnya yang akan dikatakan salah satu aturan kita.

2.2.5 Aturan penghematan

Sebuah aturan penghematan melihat satu peralatan dan mengajukan pertanyaan tentangnya. Bila jawabannya ya, aturan itu memberi saran, alasannya, dan — setiap kali mungkin — peralatan itu *sebagaimana jadinya bila saran itu diikuti*: jam yang lebih sedikit, daya yang lebih rendah, tanpa siaga. Penghematannya kemudian hanyalah energi sebelum dikurangi energi sesudah, keduanya menurut Persamaan (2.2). Beberapa saran tidak dapat dihitung secara jujur tanpa mengukur ruangannya: menyetel AC ke 25 °C alih-alih 20 °C jelas menghemat energi, dan itu adalah saran Kementerian ESDM sendiri [4], tetapi besarnya bergantung pada ruangan, insulasinya, dan cuaca, sehingga aturan kita memberi saran itu dan terus terang menyatakan bahwa penghematannya

"bergantung pada ruangan dan cuaca". Aturan yang mengarang angka lebih buruk daripada tidak ada aturan sama sekali.

2.2.6 Mengapa aturan, bukan sekadar lembar kerja?

Sebuah lembar kerja (*spreadsheet*) dapat menghitung Persamaan (2.1) dan Persamaan (2.3) untuk setiap peralatan, dan banyak keluarga sudah memilikinya. Yang tidak dilakukan lembar kerja adalah *memutuskan*. Ia akan menunjukkan bahwa lampu dapur menelan Rp 36.406 sebulan, tetapi ia tidak akan berkata "ini bohlam lama; LED akan menghemat sebagian besarnya", dan tidak akan menyebutkan alasannya. Keputusan itulah tempat keahlian berada: tahu peralatan mana yang layak diperhatikan, apa yang realistis untuk diubah, dan berapa nilai perubahannya. Aturan menangkap persis bagian itu, dan menangkapnya dalam bentuk yang dapat dibaca, diperdebatkan, dan dikoreksi oleh orang yang tahu. Penasihat energi yang tidak setuju dengan aturan 25 °C kita dapat melihatnya, mengubah angkanya, dan melihat akibatnya pada setiap rumah sekaligus.

Ada alasan kedua yang lebih diam-diam. Rumus-rumus lembar kerja tersembunyi di dalam sel-selnya, masing-masing seperti pulau kecil. Program aturan memiliki satu tempat untuk seluruh pengetahuannya — di sini, satu daftar — dan satu mesin yang menerapkannya dengan cara yang sama kepada semua orang. Bila sebuah keluarga bertanya "mengapa kami disarankan mematikan lampu teras?", jawabannya adalah sebuah aturan dengan nama, kondisi, dan alasan, bukan rujukan ke sebuah sel. Sifat ini, *pengetahuan yang dapat ditunjuk*, adalah fondasi bagi sisa buku ini.

> **CONTOH:** Bohlam dapur Bu Sri: 60 W, 14 jam sehari. Menurut Persamaan (2.1) bohlam itu memakai $60 \times 14 \times 30/1000 = 25.2$ kWh sebulan, yang dengan tarif Rp 1.444,70 berharga sekitar Rp 36.406. LED 9 W selama 14 jam yang sama akan memakai $9 \times 14 \times 30/1000 = 3.78$ kWh, seharga sekitar Rp 5.461. Penghematannya $25.2 - 3.78 = 21.42$ kWh, atau sekitar Rp 30.945 sebulan. Bila lampu itu juga dimatikan ketika siang, sehingga menyala 12 jam, bukan 14, LED itu memakai $9 \times 12 \times 30/1000 = 3.24$ kWh, lebih hemat 0.54 kWh lagi. Angka-angka inilah yang akan dicetak aplikasi, dan menghitungnya sekali dengan tangan adalah cara terbaik untuk memercayainya.

2.3 Langkah 1: Tagihan Tanpa Aturan

Sebelum menulis satu aturan pun, kita pastikan aritmetikanya benar. **Kode 2.1** menghitung energi dan biaya tiga peralatan Bu Sri dengan Persamaan (2.1) dan Persamaan (2.3).

Kode 2.1 `BillCalculator.java`: tagihan tiga peralatan, belum ada aturan.

```java
import java.util.Locale;

public class BillCalculator {
    public static void main(String[] args) {
        double rate = 1444.70;                  // Rp per kWh, R-1/TR 1300 VA
        String[] names = {"Air conditioner", "Refrigerator", "LED lamps"};
        double[] watts = {400, 90, 9};          // daya satu peralatan
        int[] counts = {1, 1, 6};               // berapa buah
        double[] hours = {9, 10, 7};            // jam pemakaian per hari

        double totalKwh = 0;
        for (int i = 0; i < names.length; i++) {
            // watt x jam = watt-jam sehari; x 30 hari; / 1000 = kWh
            double kwh = watts[i] * counts[i] * hours[i] * 30 / 1000;
            totalKwh += kwh;
            System.out.printf(Locale.US, "%-16s %6.1f kWh  Rp %,9.0f%n",
                    names[i], kwh, kwh * rate);
        }
        System.out.printf(Locale.US, "%-16s %6.1f kWh  Rp %,9.0f%n",
                "Total", totalKwh, totalKwh * rate);
    }
}
```

Peralatan-peralatan itu digambarkan oleh empat **larik** (*array*), satu larik untuk setiap sifat, dan posisi `i` di setiap larik milik peralatan ke-`i`. Perulangannya menghitung kWh setiap peralatan dengan Persamaan (2.1), menambahkannya ke total, lalu mencetak satu baris. `printf` mencetak dengan **format**: `%-16s` adalah teks yang dilebarkan menjadi 16 karakter rata kiri, `%6.1f` bilangan dalam 6 karakter dengan satu desimal, `%,9.0f` bilangan dalam 9 karakter tanpa desimal dengan koma di antara ribuan, dan `%n` pergantian baris [5]. `Locale.US` meminta koma sebagai pemisah ribuan apa pun bahasa komputer Anda; tanpanya, komputer berbahasa Indonesia akan mencetak `156.028`, cara Indonesia. Kompilasi dan jalankan dengan `javac BillCalculator.java` dan `java BillCalculator`:

```
Air conditioner   108.0 kWh  Rp   156,028
Refrigerator       27.0 kWh  Rp    39,007
LED lamps          11.3 kWh  Rp    16,383
Total             146.3 kWh  Rp   211,417
```

Periksa baris pertama dengan perhitungan tangan kita: 108.0 kWh untuk AC (tanpa daya siaganya, yang diabaikan versi sederhana ini) dan 108 × 1444.70 = 156,027.6, dicetak sebagai Rp 156,028. Baris LED menunjukkan 11.3 kWh, yaitu 11.34 kita yang dibulatkan ke satu desimal. (Program mencetak angka dengan cara internasional, koma untuk ribuan dan titik untuk desimal; kita akan tetap memakai cara itu dalam keluaran program, agar keluarannya sama di setiap komputer.)

Kode 2.2 adalah perhitungan yang sama dalam Python. **f-string**-nya (`f"..."`) melakukan pekerjaan `printf`: `{names[i]:<16}` melebarkan menjadi 16 karakter rata kiri, `{kwh:6.1f}` mencetak satu desimal dalam 6 karakter, dan `{kwh * rate:9,.0f}` mencetak tanpa desimal dengan ribuan dipisahkan koma [6].

Kode 2.2 `bill_calculator.py`: tagihan yang sama dalam Python.

```
rate = 1444.70                          # Rp per kWh, R-1/TR 1300 VA
names = ["Air conditioner", "Refrigerator", "LED lamps"]
watts = [400, 90, 9]                    # daya satu peralatan
counts = [1, 1, 6]                      # berapa buah
hours = [9, 10, 7]                      # jam pemakaian per hari

total_kwh = 0
for i in range(len(names)):
    # watt x jam = watt-jam sehari; x 30 hari; / 1000 = kWh
    kwh = watts[i] * counts[i] * hours[i] * 30 / 1000
    total_kwh += kwh
    print(f"{names[i]:<16} {kwh:6.1f} kWh  Rp {kwh * rate:9,.0f}")
print(f"{'Total':<16} {total_kwh:6.1f} kWh  Rp {total_kwh * rate:9,.0f}")
```

Keluarannya identik:

```
Air conditioner   108.0 kWh  Rp   156,028
Refrigerator       27.0 kWh  Rp    39,007
LED lamps          11.3 kWh  Rp    16,383
Total             146.3 kWh  Rp   211,417
```

2.4 Langkah 2: Aturan Pertama, sebagai Pernyataan `if`

Sekarang mari kita tambahkan saran, dengan cara paling sederhana, untuk satu peralatan. **Kode 2.3** menggambarkan AC melalui fakta-faktanya dan menuliskan dua aturan sebagai pernyataan `if`, persis seperti cara Bab 1 menuliskan pagi Pak Darto.

Kode 2.3 `FirstAdvice.java`: dua aturan penghematan yang ditulis langsung sebagai pernyataan `if`.

```
public class FirstAdvice {
    public static void main(String[] args) {
        // Satu peralatan, digambarkan oleh fakta-faktanya.
        String name = "Bedroom air conditioner";
        String kind = "ac";
        int watts = 400;
        double hours = 9;
        int setting = 20;                       // derajat Celsius

        // Aturan 1: JIKA ini AC DAN disetel di bawah 25 C
        //           MAKA sarankan 25 C.
        if (kind.equals("ac") && setting < 25) {
            System.out.println(name + ": set it to 25 C instead of "
                    + setting + " C.");
        }
        // Aturan 2: JIKA ini AC DAN menyala lebih dari 8 jam
        //           MAKA sarankan timer, dan sebutkan penghematannya.
        if (kind.equals("ac") && hours > 8) {
            double savedKwh = watts * (hours - 8) * 30 / 1000;
            System.out.println(name + ": switch it off with the timer after"
                    + " 8 hours; that saves " + savedKwh + " kWh a month.");
        }
    }
}
```

Aturan 1 memiliki dua kondisi yang dihubungkan dengan *dan*: peralatannya AC, dan disetel di bawah 25 °C. Aturan 2 juga menghitung penghematan: kelebihan jam di atas delapan, dikali daya, dikali 30 hari, dibagi 1000 — Persamaan (2.1) yang diterapkan pada jam yang dihemat. Keluarannya:

```
Bedroom air conditioner: set it to 25 C instead of 20 C.
Bedroom air conditioner: switch it off with the timer after 8 hours; that saves 12.0 kWh a
month.
```

Kode 2.4 adalah versi Python-nya, dengan keluaran yang sama.

Kode 2.4 `first_advice.py`: kedua aturan yang sama dalam Python.

```
# Satu peralatan, digambarkan oleh fakta-faktanya.
name = "Bedroom air conditioner"
kind = "ac"
watts = 400
hours = 9
setting = 20                            # derajat Celsius

# Aturan 1: JIKA ini AC DAN disetel di bawah 25 C
#           MAKA sarankan 25 C.
if kind == "ac" and setting < 25:
    print(name + ": set it to 25 C instead of " + str(setting) + " C.")
# Aturan 2: JIKA ini AC DAN menyala lebih dari 8 jam
#           MAKA sarankan timer, dan sebutkan penghematannya.
if kind == "ac" and hours > 8:
    saved_kwh = watts * (hours - 8) * 30 / 1000
    print(name + ": switch it off with the timer after 8 hours; "
          + "that saves " + str(saved_kwh) + " kWh a month.")
```

```
Bedroom air conditioner: set it to 25 C instead of 20 C.
Bedroom air conditioner: switch it off with the timer after 8 hours; that saves 12.0 kWh a
month.
```

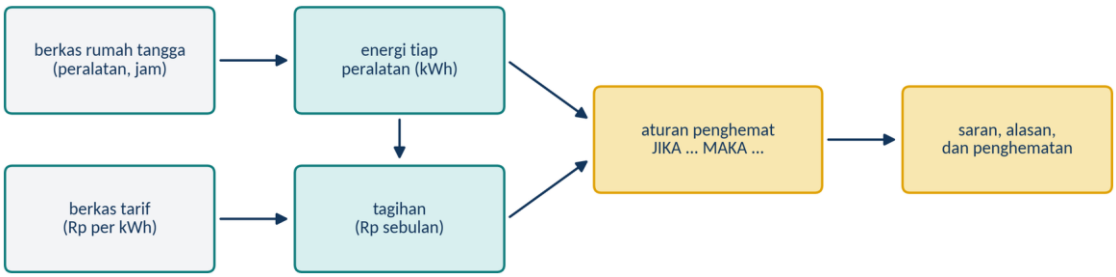
Program ini berjalan, dan untuk dua aturan dan satu peralatan, besarnya sudah pas. Namun bayangkan seluruh penasihat ditulis dengan cara ini. Bu Sri memiliki empat belas peralatan, dan penasihat yang baik mengetahui paling sedikit selusin aturan. Setiap aturan harus diulang untuk setiap peralatan, atau dibungkus dalam sebuah perulangan dengan kekusutan `if` yang terus bertambah di dalamnya. Untuk menambah aturan Anda harus menyunting bagian tengah program; untuk melihat semua aturan sekaligus Anda harus membaca seluruh program; dan untuk menghitung penghematan *total*, Anda harus memastikan bahwa dua aturan tentang lampu yang sama — "ganti dengan LED" dan "matikan ketika siang" — tidak menghitung kilowatt-jam yang sama dua kali. Pengetahuan dan mesinnya bercampur aduk.

> **WAWASAN KUNCI:** Program aturan menjadi mudah dikembangkan bila aturannya berupa *data*: sebuah daftar nilai, masing-masing memuat kondisi, saran, alasan, dan akibatnya. Program yang menerapkannya — mesin inferensi — kemudian cukup berupa perulangan kecil yang tidak pernah berubah ketika sebuah aturan ditambahkan.

2.5 Langkah 3: Aturan sebagai Nilai

2.5.1 Rancangannya

Gambar 2.1 memperlihatkan rancangan aplikasi yang sudah jadi. Dua berkas masuk: **berkas rumah tangga** yang mendaftar peralatan dan cara pemakaiannya, dan **berkas tarif** yang berisi harga per kWh setiap tarif. Program menghitung energi setiap peralatan dan tagihannya. Kemudian ia menjalankan setiap **aturan hemat** pada setiap peralatan, serta tiga **aturan rumah** pada rumah secara keseluruhan, dan mencetak setiap saran bersama alasan dan penghematannya.



Gambar 2.1 Penghemat tagihan listrik. Peralatan dan tarif menghasilkan energi dan tagihan; aturan-aturan hemat mengubahnya menjadi saran, masing-masing dengan alasan dan penghematannya.

2.5.2 Berkas rumah tangga

Berkas rumah tangga adalah berkas teks biasa dalam format **CSV** (*comma-separated values*, nilai yang dipisahkan koma), yang dapat disimpan oleh aplikasi lembar kerja apa pun dan diubah dengan penyunting teks apa pun. Berkas milik Bu Sri adalah:

```
# Bu Sri's house: a family of four, 1300 VA, postpaid
house,Bu Sri's house
tariff,R1-1300
budget,400000
name,kind,watts,count,hours,standby,setting
Bedroom air conditioner,ac,400,1,9,1,20
Refrigerator,fridge,90,1,10,0,
LED lamps,lamp-led,9,6,7,0,
Kitchen lamp (old bulb),lamp-bulb,60,1,14,0,
Terrace lamp (old bulb),lamp-bulb,40,1,12,0,
Television,tv,60,1,7,1,
Rice cooker (cooking),rice-cook,350,1,1,0,
Rice cooker (keep warm),rice-warm,45,1,14,0,
Water pump,pump,250,1,1.5,0,
Standing fan,fan,45,2,6,0,
Phone chargers,charger,10,3,3,0.5,
Iron,iron,350,1,0.5,0,
Washing machine,washer,300,1,1,1,
Wi-Fi router,router,10,1,24,0,
```

Baris yang diawali `#` adalah komentar. Tiga baris pertama memuat nama rumah, kode tarif dari **Tabel 2.1**, dan anggaran bulanan dalam rupiah. Kemudian datang tabelnya: baris judul `name,kind,watts,count,hours,standby,setting`, lalu satu baris untuk setiap peralatan. Kolom **kind** (jenis) memberi tahu aturan-aturan apa peralatannya — `ac`, `fridge` (kulkas), `lamp-led`, `lamp-bulb` (bohlam pijar lama), `rice-cook`, `rice-warm` (rice cooker yang sama ketika menghangatkan nasi, dicatat sebagai baris tersendiri karena daya dan jamnya berbeda), dan seterusnya. `hours` adalah jam pemakaian sehari; `standby` adalah daya siaga dalam watt; `setting` adalah suhu AC, dan dibiarkan kosong untuk peralatan lainnya.

Dari mana angka-angka itu berasal? Wattnya tertera pada label atau buku petunjuk setiap peralatan. Jamnya adalah perkiraan keluarga itu sendiri; untuk kulkas, yang motornya menyala dan mati sendiri, kita menulis 10 jam sebagai perkiraan lamanya motor benar-benar bekerja, dan Subbab 2.8 menjelaskan cara mengukurnya. Berkas tarif, `tariff-2026-q3.csv`, menyimpan **Tabel 2.1** dalam format yang sama.

2.5.3 Hanya bilangan bulat

Program pertama kita memakai bilangan desimal (`double` di Java, `float` di Python), dan untuk perkiraan cepat itu sudah memadai. Untuk aplikasi yang sesungguhnya kita menginginkan sesuatu yang lebih kuat: jawaban yang sama, sampai ke rupiah terakhir, di Java dan di Python, di setiap komputer, setiap kali. Bilangan desimal di komputer disimpan dalam bentuk biner, dan

bilangan seperti 1444.70 tidak dapat disimpan secara tepat; galatnya yang kecil tidak berbahaya, tetapi bila sebuah hasil jatuh tepat di tengah-tengah dua rupiah, kedua bahasa dapat membulatkannya secara berbeda. Karena itu aplikasi ini mengerjakan seluruh aritmetikanya dengan **bilangan bulat**, dalam satuan yang cukup kecil sehingga tidak memerlukan desimal:

- daya dalam **sepersepuluh watt**: 400 W menjadi `4000`, 0,5 W menjadi `5`;
- waktu dalam **menit per hari**: 1,5 jam menjadi `90`;
- energi dalam **desiwatt-menit**: sepersepuluh watt dikali menit. Satu kWh adalah $1000\,\text{W} \times 60\,\text{min} = 60{,}000$ watt-menit, yaitu 600,000 desiwatt-menit;
- tarif dalam **sen**, seperseratus rupiah: Rp 1.444,70 menjadi `144470`.

Harga energi e (dalam desiwatt-menit) dengan tarif s (dalam sen per kWh) kemudian, dalam sen utuh, adalah

$$\text{sen} = \left\lfloor \frac{e \times s + 300{,}000}{600{,}000} \right\rfloor \tag{2.4}$$

dengan $\lfloor x \rfloor$ berarti "x dibulatkan ke bawah menjadi bilangan bulat". Menambahkan separuh pembagi sebelum membagi membuat pembagian itu membulatkan ke bilangan bulat *terdekat*, dengan setengah dibulatkan ke atas. Cara yang sama mengubah sen menjadi rupiah (tambahkan 50, bagi 100) dan energi menjadi persepuluhan kWh untuk dicetak (tambahkan 30.000, bagi 60.000). Pembagian bilangan bulat itu tepat di kedua bahasa, sehingga hasilnya identik. Periksalah pada AC: $e = 4000 \times 540 \times 30 + 10 \times 900 \times 30 = 65{,}070{,}000$, dan menurut Persamaan (2.4) biayanya adalah $(65{,}070{,}000 \times 144{,}470 + 300{,}000)/600{,}000$, dibulatkan ke bawah: 15,667,771 sen, yaitu Rp 156.678 — sama dengan perhitungan tangan kita dengan Persamaan (2.3).

2.5.4 Sebuah peralatan

Kode 2.5 adalah kelas yang menyimpan satu baris berkas rumah tangga. Metodenya, `energy()`, adalah Persamaan (2.2) dalam satuan-satuan ini, dan tiga metode kecil membuat *salinan yang diubah* dari peralatan itu, yang menjadi cara sebuah aturan menggambarkan akibat sarannya: `withMinutes` untuk "kurangi pemakaiannya", `withDeciWatts` untuk "ganti dengan yang dayanya lebih kecil", dan `withoutStandby` untuk "cabut stekernya".

Kode 2.5 `Appliance.java`: satu peralatan, energi bulanannya, dan salinan-salinan yang menggambarkan saran.

```
/**
 * Satu baris berkas rumah tangga: sebuah peralatan dan cara memakainya.
 * Daya disimpan dalam sepersepuluh watt dan waktu dalam menit, sehingga
 * setiap perhitungan memakai bilangan bulat dan hasilnya sama di Java dan
 * di Python.
 */
public class Appliance {
    public static final int DAYS = 30;          // sebulan = 30 hari

    public final String name;
    public final String kind;         // ac, fridge, lamp-led, lamp-bulb, ...
    public final long deciWatts;      // daya saat menyala, dalam 0,1 W
    public final int count;           // berapa buah
    public final long minutes;        // menit pemakaian per hari
    public final long standbyDeciWatts;   // daya saat "mati" tetapi tercolok
    public final String setting;      // mis. suhu AC, atau ""

    public Appliance(String name, String kind, long deciWatts, int count,
                     long minutes, long standbyDeciWatts, String setting) {
        this.name = name;
        this.kind = kind;
        this.deciWatts = deciWatts;
        this.count = count;
        this.minutes = minutes;
        this.standbyDeciWatts = standbyDeciWatts;
        this.setting = setting;
    }

    /** Energi sebulan, dalam desiwatt-menit (600.000 di antaranya = 1 kWh). */
    public long energy() {
        long on = deciWatts * count * minutes * DAYS;
        long standby = standbyDeciWatts * count * (24 * 60 - minutes) * DAYS;
        return on + standby;
    }

    /** Salinan dengan menit yang berbeda (saran "kurangi pemakaian"). */
    public Appliance withMinutes(long newMinutes) {
        return new Appliance(name, kind, deciWatts, count, newMinutes,
                standbyDeciWatts, setting);
    }
```

Kode 2.5 (Lanjutan) *Appliance.java: satu peralatan, energi bulanannya, dan salinan-salinan yang menggambarkan saran.*

```
    /** Salinan dengan daya lain (saran "ganti peralatannya"). */
    public Appliance withDeciWatts(long newDeciWatts) {
        return new Appliance(name, kind, newDeciWatts, count, minutes,
                standbyDeciWatts, setting);
    }

    /** Salinan tanpa daya siaga (saran "cabut stekernya"). */
    public Appliance withoutStandby() {
        return new Appliance(name, kind, deciWatts, count, minutes, 0, setting);
    }

    /** Setelan sebagai bilangan, atau -1 bila tidak ada. */
    public int settingNumber() {
        return setting.isEmpty() ? -1 : Integer.parseInt(setting);
    }
}
```

Field-field-nya dideklarasikan `final`: sekali sebuah peralatan dibuat, ia tidak pernah berubah. Sebuah aturan tidak pernah *mengubah* AC Bu Sri; ia membuat `Appliance` baru yang menggambarkan AC itu sebagaimana jadinya bila timer disetel delapan jam, dan penghematannya

adalah selisih kedua energi tersebut. Membiarkan yang asli tetap utuh itulah yang memungkinkan beberapa aturan memeriksa peralatan yang sama tanpa saling mengganggu.

2.5.5 Rumah tangga dan tagihannya

Kode 2.6 membaca berkas rumah tangga dan berkas tarif, serta menyimpan aritmetika tagihan dari Subbab 2.5.3.

Kode 2.6 `Household.java`: membaca kedua berkas, dan aritmetika tagihan dengan bilangan bulat.

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;

/** Berkas rumah tangga, tarifnya, dan aritmetika tagihan. */
public class Household {
    public String name = "";
    public String tariffCode = "";
    public String tariffName = "";
    public long va;                    // daya tersambung, mis. 1300 VA
    public long rateSen;               // Rp per kWh, dalam sen (1/100 rupiah)
    public long budget;                // rupiah sebulan, 0 = tanpa anggaran
    public final List<Appliance> appliances = new ArrayList<>();

    /** "1.5" dengan 2 desimal menjadi 150; "400" menjadi 40000. */
    public static long scaled(String text, int places) {
        String t = text.strip();
        String whole = t;
        String part = "";
        int dot = t.indexOf('.');
        if (dot >= 0) {
            whole = t.substring(0, dot);
            part = t.substring(dot + 1);
        }
        if (part.length() > places) {
            throw new IllegalArgumentException("too many decimals: " + text);
        }
        while (part.length() < places) {
            part = part + "0";
        }
        return Long.parseLong(whole + part);
    }
```

Kode 2.6 (Lanjutan) *Household.java: membaca kedua berkas, dan aritmetika tagihan dengan bilangan bulat.*

```
    public static Household load(String path, String tariffPath)
            throws IOException {
        Household h = new Household();
        boolean inTable = false;
        for (String line : Files.readAllLines(Path.of(path),
                StandardCharsets.UTF_8)) {
            if (line.isBlank() || line.startsWith("#")) {
                continue;
            }
            String[] f = line.split(",", -1);
            if (f[0].equals("name")) {
                inTable = true;               // baris judul tabel
            } else if (inTable) {
                long hundredths = scaled(f[4], 2);         // jam x 100
                if (hundredths * 60 % 100 != 0) {
                    throw new IllegalArgumentException("hours must be whole "
                            + "minutes: " + f[4]);
                }
                h.appliances.add(new Appliance(f[0], f[1], scaled(f[2], 1),
                        Integer.parseInt(f[3]), hundredths * 60 / 100,
                        scaled(f[5], 1), f[6]));
            } else if (f[0].equals("house")) {
                h.name = f[1];
            } else if (f[0].equals("tariff")) {
                h.tariffCode = f[1];
            } else if (f[0].equals("budget")) {
                h.budget = Long.parseLong(f[1]);
            }
        }
        for (String line : Files.readAllLines(Path.of(tariffPath),
                StandardCharsets.UTF_8)) {
            String[] f = line.split(",", -1);
            if (f[0].equals(h.tariffCode)) {
                h.va = Long.parseLong(f[1]);
                h.rateSen = scaled(f[2], 2);
                h.tariffName = f[3];
            }
        }
        if (h.rateSen == 0) {
            throw new IllegalArgumentException("unknown tariff " + h.tariffCode);
        }
        return h;
    }
```

Kode 2.6 (Lanjutan) *Household.java: membaca kedua berkas, dan aritmetika tagihan dengan bilangan bulat.*

```
    /** Harga sejumlah energi, dalam rupiah utuh (dibulatkan). */
    public long rupiah(long energy) {
        long sen = (energy * rateSen + 300_000) / 600_000;
        return (sen + 50) / 100;
    }

    public long totalEnergy() {
        long sum = 0;
        for (Appliance a : appliances) {
            sum += a.energy();
        }
        return sum;
    }

    /** Energi sebagai kWh dengan satu desimal, mis. "108.0". */
    public static String kwh(long energy) {
        long tenths = (energy + 30_000) / 60_000;
        return (tenths / 10) + "." + (tenths % 10);
    }

    /** Rupiah dengan pemisah ribuan, mis. "Rp 156,028". */
    public static String rp(long rupiah) {
        return "Rp " + String.format(Locale.US, "%,d", rupiah);
    }

    public String rate() {
        return "Rp " + String.format(Locale.US, "%,d", rateSen / 100) + "."
                + String.format("%02d", rateSen % 100);
    }
}
```

`scaled` mengubah bilangan desimal yang tertulis di berkas menjadi bilangan bulat dalam satuan yang tepat tanpa pernah melewati `double`: `"1.5"` dengan dua angka desimal menjadi `150`, yaitu jam dikali 100; dikali 60 lalu dibagi 100 menghasilkan 90 menit. Bila jamnya bukan bilangan menit yang utuh (seperti 0.33), program menolak baris itu dengan pesan yang jelas, alih-alih membulatkannya diam-diam. `load` membaca berkas rumah tangga baris demi baris: sebelum baris judul ia mencari baris `house`, `tariff`, dan `budget`; sesudahnya, setiap baris adalah sebuah peralatan. `split(",", -1)` memecah baris pada tanda koma dan tetap menyimpan field kosong di akhir, seperti `setting` yang kosong pada lampu. Kemudian berkas tarif ditelusuri untuk mencari kode tarif rumah tangga tersebut. `rupiah` adalah Persamaan (2.4) yang diikuti pembulatan dari sen ke rupiah, sedangkan `kwh`, `rp`, dan `rate` memformat angka dengan cara yang sama di setiap komputer.

Dalam Python, peralatan dan rumah tangga berada dalam satu modul, **Kode 2.7**.

Kode 2.7 `household.py`: peralatan, berkas rumah tangga, dan aritmetika tagihan dalam Python.

```
"""Berkas rumah tangga, peralatannya, tarifnya, dan aritmetika tagihan.

Daya disimpan dalam sepersepuluh watt dan waktu dalam menit, sehingga
setiap perhitungan memakai bilangan bulat dan hasilnya sama di Python dan
di Java."""

DAYS = 30                                  # sebulan = 30 hari


class Appliance:
    def __init__(self, name, kind, deci_watts, count, minutes,
                 standby_deci_watts, setting):
        self.name = name
        self.kind = kind                   # ac, fridge, lamp-led, lamp-bulb, ...
        self.deci_watts = deci_watts       # daya saat menyala, dalam 0,1 W
        self.count = count                 # berapa buah
        self.minutes = minutes             # menit pemakaian per hari
        self.standby_deci_watts = standby_deci_watts  # mati tetapi tercolok
        self.setting = setting             # mis. suhu AC, atau ""

    def energy(self):
        """Energi sebulan, dalam desiwatt-menit (600.000 = 1 kWh)."""
        on = self.deci_watts * self.count * self.minutes * DAYS
        standby = (self.standby_deci_watts * self.count
                   * (24 * 60 - self.minutes) * DAYS)
        return on + standby

    def with_minutes(self, new_minutes):
        return Appliance(self.name, self.kind, self.deci_watts, self.count,
                         new_minutes, self.standby_deci_watts, self.setting)

    def with_deci_watts(self, new_deci_watts):
        return Appliance(self.name, self.kind, new_deci_watts, self.count,
                         self.minutes, self.standby_deci_watts, self.setting)

    def without_standby(self):
        return Appliance(self.name, self.kind, self.deci_watts, self.count,
                         self.minutes, 0, self.setting)

    def setting_number(self):
        """Setelan sebagai bilangan, atau -1 bila tidak ada."""
        return int(self.setting) if self.setting else -1
```

Kode 2.7 (Lanjutan) *household.py: peralatan, berkas rumah tangga, dan aritmetika tagihan dalam Python.*

```
def scaled(text, places):
    """'1.5' dengan 2 desimal menjadi 150; '400' dengan 1 desimal 4000."""
    text = text.strip()
    whole, _, part = text.partition(".")
    if len(part) > places:
        raise ValueError("too many decimals: " + text)
    return int(whole + part.ljust(places, "0"))


class Household:
    def __init__(self):
        self.name = ""
        self.tariff_code = ""
        self.tariff_name = ""
        self.va = 0                     # daya tersambung, mis. 1300 VA
        self.rate_sen = 0               # Rp per kWh, dalam sen (1/100 rupiah)
        self.budget = 0                 # rupiah sebulan, 0 = tanpa anggaran
        self.appliances = []

    @staticmethod
    def load(path, tariff_path):
        h = Household()
        in_table = False
        with open(path, encoding="utf-8") as f:
            for line in f.read().splitlines():
                if not line.strip() or line.startswith("#"):
                    continue
                f = line.split(",")
                if f[0] == "name":
                    in_table = True      # baris judul tabel
                elif in_table:
                    hundredths = scaled(f[4], 2)           # jam x 100
                    if hundredths * 60 % 100 != 0:
                        raise ValueError("hours must be whole minutes: " + f[4])
                    h.appliances.append(Appliance(
                        f[0], f[1], scaled(f[2], 1), int(f[3]),
                        hundredths * 60 // 100, scaled(f[5], 1), f[6]))
                elif f[0] == "house":
                    h.name = f[1]
                elif f[0] == "tariff":
                    h.tariff_code = f[1]
                elif f[0] == "budget":
                    h.budget = int(f[1])
        with open(tariff_path, encoding="utf-8") as f:
            for line in f.read().splitlines():
                f = line.split(",")
```

Kode 2.7 (Lanjutan) *household.py: peralatan, berkas rumah tangga, dan aritmetika tagihan dalam Python.*

```
                if f[0] == h.tariff_code:
                    h.va = int(f[1])
                    h.rate_sen = scaled(f[2], 2)
                    h.tariff_name = f[3]
        if h.rate_sen == 0:
            raise ValueError("unknown tariff " + h.tariff_code)
        return h

    def rupiah(self, energy):
        """Harga sejumlah energi, dalam rupiah utuh (dibulatkan)."""
        sen = (energy * self.rate_sen + 300_000) // 600_000
        return (sen + 50) // 100

    def total_energy(self):
        return sum(a.energy() for a in self.appliances)

    def rate(self):
        return f"Rp {self.rate_sen // 100:,d}.{self.rate_sen % 100:02d}"


def kwh(energy):
    """Energi sebagai kWh dengan satu desimal, mis. '108.5'."""
    tenths = (energy + 30_000) // 60_000
    return str(tenths // 10) + "." + str(tenths % 10)


def rp(rupiah):
    """Rupiah dengan pemisah ribuan, mis. 'Rp 156,678'."""
    return f"Rp {rupiah:,d}"
```

Ada dua perbedaan yang patut diperhatikan. Bilangan bulat Python tidak memiliki batas ukuran dan `//` adalah pembagian bilangan bulat, sehingga rumus-rumusnya tampak persis seperti Persamaan (2.4). Dan `text.partition(".")` memecah `"1.5"` menjadi `"1"`, `"."`, dan `"5"` dalam satu langkah, sedangkan Java memerlukan `indexOf` dan `substring`.

2.5.6 Aturan adalah sebuah nilai

Kini kita sampai di jantung bab ini. **Kode 2.8** mendefinisikan apa sebenarnya sebuah aturan penghematan: sebuah pengenal, sebuah kondisi, cara menghasilkan saran, cara menghasilkan alasan, dan cara menghasilkan peralatan yang diubah.

Kode 2.8 `SavingRule.java`: aturan penghematan sebagai nilai dengan lima bagian.

```
import java.util.function.Function;
import java.util.function.Predicate;

/**
 * Aturan JIKA-MAKA tentang satu peralatan. "when" adalah bagian JIKA;
 * "advice" dan "why" adalah isi aturan; "change" adalah cara peralatan
 * dipakai bila sarannya diikuti (null bila tidak dapat dihitung).
 */
public class SavingRule {
    public final String id;
    public final Predicate<Appliance> when;
    public final Function<Appliance, String> advice;
    public final Function<Appliance, String> why;
    public final Function<Appliance, Appliance> change;

    public SavingRule(String id, Predicate<Appliance> when,
                      Function<Appliance, String> advice,
                      Function<Appliance, String> why,
                      Function<Appliance, Appliance> change) {
        this.id = id;
        this.when = when;
        this.advice = advice;
        this.why = why;
        this.change = change;
    }
}
```

Bagian-bagiannya adalah **fungsi yang disimpan dalam field**. `Predicate<Appliance>` adalah fungsi yang menerima sebuah peralatan dan menjawab benar atau salah: bagian JIKA. `Function<Appliance, String>` menerima sebuah peralatan dan mengembalikan teks: saran dan alasannya. `Function<Appliance, Appliance>` menerima sebuah peralatan dan mengembalikan salinan yang diubah: akibat sarannya. Dalam Java fungsi semacam itu dapat ditulis sebagai **ekspresi lambda**, `a -> a.kind.equals("ac")`, yang dapat Anda baca keras-keras sebagai "diberikan sebuah peralatan `a`, jawablah apakah jenisnya `ac`".

Kode 2.9 adalah seluruh aplikasinya. Metode `rules()` membangun daftar enam aturan peralatan; `main` membaca berkas-berkas, mencetak tagihan, menjalankan aturan-aturan, menjalankan tiga aturan rumah, lalu mencetak totalnya.

Kode 2.9 `BillSaver.java`: penghemat tagihan listrik, dengan aturan-aturannya sebagai nilai dan mesin kecil yang menerapkannya.

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

/**
 * Penghemat tagihan listrik: membaca berkas rumah tangga, menghitung tagihan
 * bulanan, menjalankan setiap aturan hemat pada setiap peralatan, lalu tiga
 * aturan tentang seluruh rumah, dan menyebutkan penghematan tiap saran.
 * Jalankan: java BillSaver ../data/home-sri.csv ../data/tariff-2026-q3.csv
 */
public class BillSaver {
    static final int AC_BEST = 25;            // derajat C ("3M" dari ESDM)
    static final long AC_HOURS = 8 * 60;      // menit sehari
    static final long LIGHT_HOURS = 12 * 60;
    static final long WARM_HOURS = 4 * 60;
    static final int LED_PERCENT = 15;        // LED 9 W ~ bohlam 60 W
    static final long HEAVY = 2500;           // 250 W, dalam sepersepuluh watt

    static List<SavingRule> rules() {
        List<SavingRule> r = new ArrayList<>();
        r.add(new SavingRule("ac-setpoint",
                a -> a.kind.equals("ac") && a.settingNumber() >= 0
                        && a.settingNumber() < AC_BEST,
                a -> "set it to " + AC_BEST + " C instead of " + a.setting + " C",
                a -> "it is an air conditioner set below " + AC_BEST + " C",
                null));
        r.add(new SavingRule("ac-timer",
                a -> a.kind.equals("ac") && a.minutes > AC_HOURS,
                a -> "let the timer switch it off after 8 hours",
                a -> "it is an air conditioner used " + a.minutes / 60.0
                        + " hours a day",
                a -> a.withMinutes(AC_HOURS)));
        r.add(new SavingRule("old-bulb",
                a -> a.kind.equals("lamp-bulb"),
                a -> "replace it with an LED lamp of about "
                        + ledDeciWatts(a) / 10 + " W",
                a -> "it is an old bulb of " + a.deciWatts / 10 + " W",
                a -> a.withDeciWatts(ledDeciWatts(a))));
        r.add(new SavingRule("lights-daytime",
                a -> a.kind.startsWith("lamp") && a.minutes > LIGHT_HOURS,
                a -> "switch it off in daylight; 12 hours a day is enough",
                a -> "it is a lamp on for " + a.minutes / 60.0 + " hours a day",
                a -> a.withMinutes(LIGHT_HOURS)));
        r.add(new SavingRule("keep-warm",
                a -> a.kind.equals("rice-warm") && a.minutes > WARM_HOURS,
                a -> "cook closer to mealtime; keep rice warm 4 hours at most",
```

Kode 2.9 (Lanjutan) *BillSaver.java: penghemat tagihan listrik, dengan aturan-aturannya sebagai nilai dan mesin kecil yang menerapkannya.*

```
                a -> "the rice cooker keeps warm " + a.minutes / 60.0
                        + " hours a day",
                a -> a.withMinutes(WARM_HOURS)));
        r.add(new SavingRule("standby",
                a -> a.standbyDeciWatts > 0 && a.minutes < 24 * 60,
                a -> "switch it off at the socket when you are not using it",
                a -> "it uses " + a.standbyDeciWatts / 10.0
                        + " W even when it is off",
                a -> a.withoutStandby()));
        return r;
    }

    static long ledDeciWatts(Appliance a) {
        return (a.deciWatts * LED_PERCENT + 50) / 100;
    }

    public static void main(String[] args) throws IOException {
        String file = args.length > 0 ? args[0] : "../data/home-sri.csv";
        String tariff = args.length > 1 ? args[1] : "../data/tariff-2026-q3.csv";
        Household h = Household.load(file, tariff);
        long total = h.totalEnergy();
        long bill = h.rupiah(total);

        System.out.println("Electricity-bill saver: " + h.name);
        System.out.println("Tariff: " + h.tariffName);
        System.out.println("Price:  " + h.rate()
                + " per kWh (July-September 2026)");
        System.out.println();
        System.out.printf("%-26s %9s %10s %6s%n", "Appliance", "kWh/month",
                "Rp/month", "Share");
        System.out.println("-".repeat(54));
        for (Appliance a : h.appliances) {
            long e = a.energy();
            long share = (e * 100 + total / 2) / total;
            System.out.printf("%-26s %9s %10s %5d%%%n", a.name, Household.kwh(e),
                    Household.rp(h.rupiah(e)).substring(3), share);
        }
        System.out.println("-".repeat(54));
        System.out.printf("%-26s %9s %10s%n", "Total", Household.kwh(total),
                Household.rp(bill).substring(3));
        System.out.println();
```

Kode 2.9 (Lanjutan) *BillSaver.java: penghemat tagihan listrik, dengan aturan-aturannya sebagai nilai dan mesin kecil yang menerapkannya.*

```
        // Jalankan setiap aturan peralatan pada setiap peralatan.
        List<SavingRule> rules = rules();
        List<Appliance> after = new ArrayList<>();
        int number = 0;
        System.out.println("Advice (" + rules.size() + " appliance rules and "
                + "3 house rules checked):");
        for (Appliance a : h.appliances) {
            Appliance changed = a;
            for (SavingRule rule : rules) {
                if (!rule.when.test(a)) {
                    continue;                // bagian JIKA bernilai salah
                }
                number++;
                System.out.println(String.format("%2d. [%s] %s", number,
                        rule.id, a.name));
                System.out.println("    advice: " + rule.advice.apply(a));
                System.out.println("    why: " + rule.why.apply(a));
                if (rule.change == null) {
                    System.out.println("    saving: depends on the room and "
                            + "the weather");
                } else {
                    long saved = a.energy() - rule.change.apply(a).energy();
                    System.out.println("    saving: " + Household.kwh(saved)
                            + " kWh = " + Household.rp(h.rupiah(saved))
                            + " a month");
                    changed = rule.change.apply(changed);
                }
            }
            after.add(changed);
        }

        // Aturan rumah 1: peralatan paling boros.
        for (Appliance a : h.appliances) {
            long share = (a.energy() * 100 + total / 2) / total;
            if (share >= 30) {
                number++;
                System.out.println(String.format("%2d. [biggest-eater] %s",
                        number, a.name));
                System.out.println("    advice: look at it first; it uses "
                        + share + "% of your electricity");
            }
        }
        // Aturan rumah 2: semua dinyalakan bersamaan.
        long peak = 0;
        for (Appliance a : h.appliances) {
            peak += a.deciWatts * a.count / 10;
```

Kode 2.9 (Lanjutan) *BillSaver.java: penghemat tagihan listrik, dengan aturan-aturannya sebagai nilai dan mesin kecil yang menerapkannya.*

```
        }
        if (peak > h.va) {
            number++;
            System.out.println(String.format("%2d. [overload] the whole house",
                    number));
            System.out.println("    advice: do not run these heavy appliances "
                    + "at the same time:");
            for (Appliance a : h.appliances) {
                if (a.deciWatts >= HEAVY) {
                    System.out.println("      - " + a.name);
                }
            }
            System.out.println("    why: everything on at once needs about "
                    + peak + " W, more than your " + h.va + " VA");
            System.out.println("    safety: never change the MCB yourself; "
                    + "ask PLN if you need more power");
        }
        // Aturan rumah 3: anggaran.
        long newTotal = 0;
        for (Appliance a : after) {
            newTotal += a.energy();
        }
        long newBill = h.rupiah(newTotal);
        if (h.budget > 0 && bill > h.budget) {
            number++;
            System.out.println(String.format("%2d. [budget] the whole house",
                    number));
            System.out.println("    why: the bill is " + Household.rp(bill
                    - h.budget) + " over your budget of " + Household.rp(h.budget));
            System.out.println("    with the advice above it becomes "
                    + Household.rp(newBill) + (newBill <= h.budget
                    ? ", within the budget" : ", still over the budget"));
        }
        System.out.println();
        System.out.println("If you follow the advice: " + Household.kwh(
                total - newTotal) + " kWh and " + Household.rp(bill - newBill)
                + " less each month,");
        System.out.println("that is " + Household.rp(newBill) + " instead of "
                + Household.rp(bill) + ".");
    }
}
```

Bacalah aturan-aturannya lebih dahulu, karena merekalah pengetahuannya. `ac-setpoint` terpenuhi untuk AC yang setelannya di bawah 25; ia memberi saran dan alasan tetapi tanpa perubahan, karena penghematannya tidak dapat dihitung secara jujur. `ac-timer` terpenuhi untuk AC yang dipakai lebih dari delapan jam dan mengubah menitnya menjadi 480. `old-bulb` terpenuhi untuk setiap bohlam lama dan mengubah dayanya menjadi daya LED: 15 persen dari daya bohlam, dibulatkan, yang mengubah 60 W menjadi 9 W — kesetaraan yang lazim tercetak pada kemasan LED. `lights-daytime` terpenuhi untuk lampu mana pun yang menyala lebih dari dua belas jam. `keep-warm` terpenuhi untuk rice cooker yang menghangatkan lebih dari empat jam. `standby` terpenuhi untuk apa pun yang memiliki daya siaga dan tidak dipakai sepanjang hari. Konstanta di bagian atas — `AC_BEST`, `AC_HOURS`, `LED_PERCENT`, dan yang lainnya — adalah angka-angka yang menjadi tumpuan aturan, dikumpulkan di satu tempat sehingga seorang penasihat dapat mengubahnya.

Lalu bacalah mesinnya: perulangan *untuk setiap peralatan, untuk setiap aturan*. Bila kondisi sebuah aturan salah, `continue` beralih ke aturan berikutnya. Bila benar, perulangan mencetak pengenal aturan, peralatannya, saran, dan alasannya; bila aturan itu memiliki perubahan, ia

mencetak penghematannya — energi peralatan dikurangi energi salinan yang diubah — dan menerapkan perubahan itu pada salinan berjalan bernama `changed`. Salinan berjalan itulah jawaban atas kekhawatiran penghitungan ganda di Subbab 2.4: ketika dua aturan terpenuhi pada lampu dapur, aturan pertama mengubahnya menjadi LED 9 W, dan aturan kedua memperpendek jam *LED itu*, sehingga penghematan total menghitung setiap kilowatt-jam satu kali saja. Setelah perulangan, daftar `after` menyimpan setiap peralatan sebagaimana jadinya bila semua saran diikuti.

Ketiga **aturan rumah** melihat rumah secara keseluruhan. `biggest-eater` menunjuk peralatan mana pun yang memakai 30 persen atau lebih dari listrik. `overload` menjumlahkan daya semua peralatan, seolah-olah semuanya dinyalakan bersamaan, lalu membandingkannya dengan VA sambungan; watt dan volt-ampere tidak persis sama, tetapi untuk peralatan rumah tangga keduanya cukup dekat untuk sebuah peringatan. Bila jumlahnya lebih besar, aturan ini mendaftar peralatan-peralatan berat yang tidak boleh berjalan bersamaan — dan menambahkan satu baris keselamatan, karena MCB yang turun sedang menjalankan tugasnya, dan menggantinya dengan yang lebih besar berbahaya serta melanggar ketentuan PLN. `budget` membandingkan tagihan dengan anggaran keluarga dan menyatakan apakah saran-saran itu akan membawanya ke dalam anggaran.

Kode 2.10 adalah versi Python-nya. Aturan-aturannya adalah nilai lima bagian yang sama, dibangun dengan `lambda` Python dan sebuah kelas kecil `SavingRule`, dan mesinnya adalah perulangan ganda yang sama.

Kode 2.10 `bill_saver.py`: penghemat tagihan listrik dalam Python.

```
"""Penghemat tagihan listrik: membaca berkas rumah tangga, menghitung tagihan
bulanan, menjalankan setiap aturan hemat pada setiap peralatan, lalu tiga
aturan tentang seluruh rumah, dan menyebutkan penghematan tiap saran.
Jalankan: python bill_saver.py ../data/home-sri.csv ../data/tariff-2026-q3.csv
"""

import sys

from household import Household, kwh, rp

AC_BEST = 25                            # derajat C ("3M" dari ESDM)
AC_HOURS = 8 * 60                       # menit sehari
LIGHT_HOURS = 12 * 60
WARM_HOURS = 4 * 60
LED_PERCENT = 15                        # LED 9 W ~ bohlam 60 W
HEAVY = 2500                            # 250 W, dalam sepersepuluh watt


class SavingRule:
    """Aturan JIKA-MAKA tentang satu peralatan. `when` adalah bagian JIKA;
    `advice` dan `why` adalah isi aturan; `change` adalah cara peralatan
    dipakai bila sarannya diikuti (None bila tidak dapat dihitung)."""

    def __init__(self, rule_id, when, advice, why, change):
        self.id = rule_id
        self.when = when
        self.advice = advice
        self.why = why
        self.change = change


def led_deci_watts(a):
    return (a.deci_watts * LED_PERCENT + 50) // 100


def rules():
    return [
        SavingRule(
            "ac-setpoint",
            lambda a: a.kind == "ac" and 0 <= a.setting_number() < AC_BEST,
            lambda a: f"set it to {AC_BEST} C instead of {a.setting} C",
            lambda a: f"it is an air conditioner set below {AC_BEST} C",
            None),
        SavingRule(
            "ac-timer",
            lambda a: a.kind == "ac" and a.minutes > AC_HOURS,
```

Kode 2.10 (Lanjutan) *bill_saver.py: penghemat tagihan listrik dalam Python.*

```
            lambda a: "let the timer switch it off after 8 hours",
            lambda a: "it is an air conditioner used "
                      + str(a.minutes / 60) + " hours a day",
            lambda a: a.with_minutes(AC_HOURS)),
        SavingRule(
            "old-bulb",
            lambda a: a.kind == "lamp-bulb",
            lambda a: "replace it with an LED lamp of about "
                      + str(led_deci_watts(a) // 10) + " W",
            lambda a: "it is an old bulb of " + str(a.deci_watts // 10) + " W",
            lambda a: a.with_deci_watts(led_deci_watts(a))),
        SavingRule(
            "lights-daytime",
            lambda a: a.kind.startswith("lamp") and a.minutes > LIGHT_HOURS,
            lambda a: "switch it off in daylight; 12 hours a day is enough",
            lambda a: "it is a lamp on for " + str(a.minutes / 60)
                      + " hours a day",
            lambda a: a.with_minutes(LIGHT_HOURS)),
        SavingRule(
            "keep-warm",
            lambda a: a.kind == "rice-warm" and a.minutes > WARM_HOURS,
            lambda a: "cook closer to mealtime; keep rice warm 4 hours at most",
            lambda a: "the rice cooker keeps warm " + str(a.minutes / 60)
                      + " hours a day",
            lambda a: a.with_minutes(WARM_HOURS)),
        SavingRule(
            "standby",
            lambda a: a.standby_deci_watts > 0 and a.minutes < 24 * 60,
            lambda a: "switch it off at the socket when you are not using it",
            lambda a: "it uses " + str(a.standby_deci_watts / 10)
                      + " W even when it is off",
            lambda a: a.without_standby()),
    ]


def main(args):
    path = args[0] if args else "../data/home-sri.csv"
    tariff = args[1] if len(args) > 1 else "../data/tariff-2026-q3.csv"
    h = Household.load(path, tariff)
    total = h.total_energy()
    bill = h.rupiah(total)
```

Kode 2.10 (Lanjutan) *bill_saver.py: penghemat tagihan listrik dalam Python.*

```
print("Electricity-bill saver: " + h.name)
print("Tariff: " + h.tariff_name)
print("Price:  " + h.rate() + " per kWh (July-September 2026)")
print()
print(f"{'Appliance':<26} {'kWh/month':>9} {'Rp/month':>10} {'Share':>6}")
print("-" * 54)
for a in h.appliances:
    e = a.energy()
    share = (e * 100 + total // 2) // total
    print(f"{a.name:<26} {kwh(e):>9} {rp(h.rupiah(e))[3:]:>10} "
          f"{share:>5}%")
print("-" * 54)
print(f"{'Total':<26} {kwh(total):>9} {rp(bill)[3:]:>10}")
print()

# Jalankan setiap aturan peralatan pada setiap peralatan.
all_rules = rules()
after = []
number = 0
print("Advice (" + str(len(all_rules)) + " appliance rules and "
      + "3 house rules checked):")
for a in h.appliances:
    changed = a
    for rule in all_rules:
        if not rule.when(a):
            continue                # bagian JIKA bernilai salah
        number += 1
        print(f"{number:2d}. [{rule.id}] {a.name}")
        print("    advice: " + rule.advice(a))
        print("    why: " + rule.why(a))
        if rule.change is None:
            print("    saving: depends on the room and the weather")
        else:
            saved = a.energy() - rule.change(a).energy()
            print("    saving: " + kwh(saved) + " kWh = "
                  + rp(h.rupiah(saved)) + " a month")
            changed = rule.change(changed)
    after.append(changed)
```

Kode 2.10 (Lanjutan) *bill_saver.py: penghemat tagihan listrik dalam Python.*

```
    # Aturan rumah 1: peralatan paling boros.
    for a in h.appliances:
        share = (a.energy() * 100 + total // 2) // total
        if share >= 30:
            number += 1
            print(f"{number:2d}. [biggest-eater] {a.name}")
            print("    advice: look at it first; it uses " + str(share)
                  + "% of your electricity")
    # Aturan rumah 2: semua dinyalakan bersamaan.
    peak = sum(a.deci_watts * a.count // 10 for a in h.appliances)
    if peak > h.va:
        number += 1
        print(f"{number:2d}. [overload] the whole house")
        print("    advice: do not run these heavy appliances at the same time:")
        for a in h.appliances:
            if a.deci_watts >= HEAVY:
                print("      - " + a.name)
        print("    why: everything on at once needs about " + str(peak)
              + " W, more than your " + str(h.va) + " VA")
        print("    safety: never change the MCB yourself; ask PLN if you "
              "need more power")
    # Aturan rumah 3: anggaran.
    new_total = sum(a.energy() for a in after)
    new_bill = h.rupiah(new_total)
    if h.budget > 0 and bill > h.budget:
        number += 1
        print(f"{number:2d}. [budget] the whole house")
        print("    why: the bill is " + rp(bill - h.budget)
              + " over your budget of " + rp(h.budget))
        print("    with the advice above it becomes " + rp(new_bill)
              + (", within the budget" if new_bill <= h.budget
                 else ", still over the budget"))
    print()
    print("If you follow the advice: " + kwh(total - new_total) + " kWh and "
          + rp(bill - new_bill) + " less each month,")
    print("that is " + rp(new_bill) + " instead of " + rp(bill) + ".")


if __name__ == "__main__":
    main(sys.argv[1:])
```

2.6 Menjalankan Penghemat

Dari folder `Chapter02/java`, kompilasi semuanya dan jalankan penghemat pada rumah Bu Sri:

```
javac -d out *.java
java -cp out BillSaver ../data/home-sri.csv ../data/tariff-2026-q3.csv
```

Dalam Python, dari `Chapter02/python`: `python bill_saver.py ../data/home-sri.csv ../data/tariff-2026-q3.csv`. Keduanya mencetak laporan yang persis sama:

```
Electricity-bill saver: Bu Sri's house
Tariff: R-1/TR 1300 VA
Price:  Rp 1,444.70 per kWh (July-September 2026)

Appliance                  kWh/month   Rp/month  Share
-----------------------------------------------------
Bedroom air conditioner        108.5    156,678    38%
Refrigerator                    27.0     39,007    10%
LED lamps                       11.3     16,383     4%
Kitchen lamp (old bulb)         25.2     36,406     9%
Terrace lamp (old bulb)         14.4     20,804     5%
Television                      13.1     18,940     5%
Rice cooker (cooking)           10.5     15,169     4%
Rice cooker (keep warm)         18.9     27,305     7%
Water pump                      11.3     16,253     4%
Standing fan                    16.2     23,404     6%
Phone chargers                   3.6      5,266     1%
Iron                             5.3      7,585     2%
Washing machine                  9.7     13,999     3%
Wi-Fi router                     7.2     10,402     3%
-----------------------------------------------------
Total                          282.1    407,600

Advice (6 appliance rules and 3 house rules checked):
 1. [ac-setpoint] Bedroom air conditioner
    advice: set it to 25 C instead of 20 C
    why: it is an air conditioner set below 25 C
    saving: depends on the room and the weather
 2. [ac-timer] Bedroom air conditioner
    advice: let the timer switch it off after 8 hours
    why: it is an air conditioner used 9.0 hours a day
    saving: 12.0 kWh = Rp 17,293 a month
 3. [standby] Bedroom air conditioner
    advice: switch it off at the socket when you are not using it
    why: it uses 1.0 W even when it is off
    saving: 0.5 kWh = Rp 650 a month
 4. [old-bulb] Kitchen lamp (old bulb)
    advice: replace it with an LED lamp of about 9 W
    why: it is an old bulb of 60 W
    saving: 21.4 kWh = Rp 30,945 a month
 5. [lights-daytime] Kitchen lamp (old bulb)
    advice: switch it off in daylight; 12 hours a day is enough
    why: it is a lamp on for 14.0 hours a day
    saving: 3.6 kWh = Rp 5,201 a month
 6. [old-bulb] Terrace lamp (old bulb)
    advice: replace it with an LED lamp of about 6 W
    why: it is an old bulb of 40 W
    saving: 12.2 kWh = Rp 17,683 a month
 7. [standby] Television
    advice: switch it off at the socket when you are not using it
    why: it uses 1.0 W even when it is off
    saving: 0.5 kWh = Rp 737 a month
```

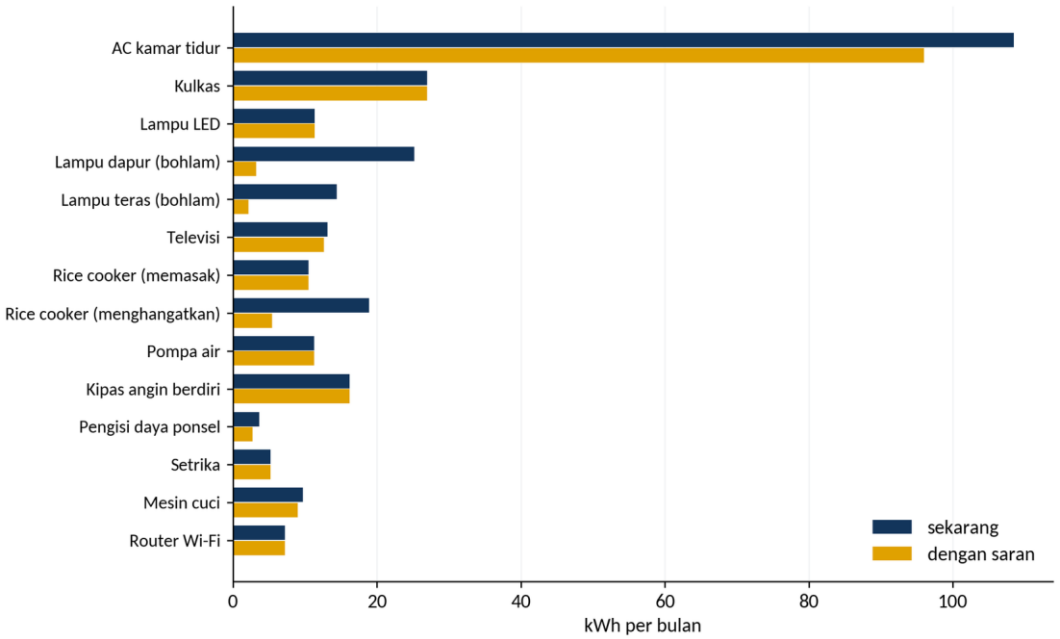
```
 8. [keep-warm] Rice cooker (keep warm)
    advice: cook closer to mealtime; keep rice warm 4 hours at most
    why: the rice cooker keeps warm 14.0 hours a day
    saving: 13.5 kWh = Rp 19,503 a month
 9. [standby] Phone chargers
    advice: switch it off at the socket when you are not using it
    why: it uses 0.5 W even when it is off
    saving: 0.9 kWh = Rp 1,365 a month
10. [standby] Washing machine
    advice: switch it off at the socket when you are not using it
    why: it uses 1.0 W even when it is off
    saving: 0.7 kWh = Rp 997 a month
11. [biggest-eater] Bedroom air conditioner
    advice: look at it first; it uses 38% of your electricity
12. [overload] the whole house
    advice: do not run these heavy appliances at the same time:
      - Bedroom air conditioner
      - Rice cooker (cooking)
      - Water pump
      - Iron
      - Washing machine
    why: everything on at once needs about 2129 W, more than your 1300 VA
    safety: never change the MCB yourself; ask PLN if you need more power
13. [budget] the whole house
    why: the bill is Rp 7,600 over your budget of Rp 400,000
    with the advice above it becomes Rp 317,603, within the budget

If you follow the advice: 62.3 kWh and Rp 89,997 less each month,
that is Rp 317,603 instead of Rp 407,600.
```

Bacalah laporan itu seperti Bu Sri membacanya. Tabelnya menyatakan bahwa AC saja menghabiskan 38 persen tagihan, Rp 156.678 dari Rp 407.600; bahwa bohlam lama di dapur menelan biaya lebih besar daripada dugaan, Rp 36.406, karena menyala empat belas jam sehari dengan daya 60 W; dan bahwa menghangatkan nasi selama empat belas jam menghabiskan Rp 27.305 sebulan, hampir dua kali biaya memasaknya.

Kemudian saran-sarannya, tiga belas butir dari sembilan aturan. Penghematan tunggal terbesar adalah mengganti bohlam dapur dengan LED 9 W, Rp 30.945 sebulan untuk lampu yang harganya hanya sebagian kecil dari angka itu. Menghangatkan nasi empat jam alih-alih empat belas jam menghemat Rp 19.503; timer AC menghemat Rp 17.293. Butir-butir siaga kecil, masing-masing beberapa ratus rupiah; semuanya layak dilakukan karena tidak memerlukan biaya, tetapi tidak akan menyelamatkan anggaran. Aturan `overload` mengingatkan bahwa semua peralatan sekaligus membutuhkan sekitar 2129 W pada sambungan 1300 VA, dan mendaftar lima peralatan berat yang tidak boleh berjalan bersamaan. Dan aturan `budget` memberi putusannya: tagihan melampaui anggaran sebesar Rp 7.600, dan dengan saran-saran itu akan menjadi Rp 317.603, jauh di dalam anggaran — penghematan Rp 89.997 sebulan, atau lebih dari satu juta rupiah setahun.

Gambar 2.2 memperlihatkan hasil yang sama sebagai gambar: energi bulanan setiap peralatan sekarang, dan dengan saran-saran itu.



Gambar 2.2 Energi bulanan per peralatan di rumah Bu Sri, sekarang dan bila semua saran yang dapat dihitung diikuti (rumah tangga contoh kita).

Program yang sama, tanpa perubahan sedikit pun, memberi saran kepada Dina, mahasiswi yang menempati kamar kos dengan meteran prabayar 900 VA sendiri (`room-dina.csv`):

```
Electricity-bill saver: Dina's room
Tariff: R-1/TR 900 VA (non-subsidized)
Price:  Rp 1,352.00 per kWh (July-September 2026)

Appliance                  kWh/month   Rp/month  Share
-----------------------------------------------------
Ceiling lamp                     2.2      2,920     3%
Desk lamp (old bulb)             3.8      5,070     6%
Laptop                          12.2     16,548    18%
Phone charger                    1.2      1,643     2%
Small fridge                    14.4     19,469    22%
Rice cooker (cooking)            4.5      6,084     7%
Rice cooker (keep warm)         12.0     16,224    18%
Fan                             12.0     16,224    18%
Electric kettle                  4.5      6,084     7%
-----------------------------------------------------
Total                           66.8     90,266

Advice (6 appliance rules and 3 house rules checked):
 1. [old-bulb] Desk lamp (old bulb)
    advice: replace it with an LED lamp of about 3 W
    why: it is an old bulb of 25 W
    saving: 3.2 kWh = Rp 4,299 a month
 2. [standby] Laptop
    advice: switch it off at the socket when you are not using it
    why: it uses 1.0 W even when it is off
    saving: 0.5 kWh = Rp 730 a month
 3. [standby] Phone charger
    advice: switch it off at the socket when you are not using it
    why: it uses 0.5 W even when it is off
    saving: 0.3 kWh = Rp 426 a month
 4. [keep-warm] Rice cooker (keep warm)
    advice: cook closer to mealtime; keep rice warm 4 hours at most
    why: the rice cooker keeps warm 10.0 hours a day
    saving: 7.2 kWh = Rp 9,734 a month
 5. [overload] the whole house
    advice: do not run these heavy appliances at the same time:
      - Rice cooker (cooking)
      - Electric kettle
    why: everything on at once needs about 1152 W, more than your 900 VA
    safety: never change the MCB yourself; ask PLN if you need more power

If you follow the advice: 11.2 kWh and Rp 15,189 less each month,
that is Rp 75,077 instead of Rp 90,266.
```

Hanya empat aturan peralatan yang terpenuhi, dan saran terbesarnya lagi-lagi soal menghangatkan nasi. Peringatan beban lebih muncul karena ketel listrik 600 W dan rice cooker 300 W bersamaan sudah memenuhi sambungan 900 VA; Dina tidak memiliki masalah anggaran, sehingga aturan anggaran diam saja. Perhatikan bahwa aturan-aturannya tidak berubah di antara kedua rumah. Hanya faktanya yang berubah. Itulah arti aturan yang terpisah dari program.

2.7 Membuatnya Lebih Baik

2.7.1 Semua aturan dalam satu halaman

Karena aturan-aturan itu adalah nilai, kita dapat menggambarkannya sebagai tabel, cara seorang penasihat meninjaunya. **Tabel 2.2** mendaftar kesembilan aturan aplikasi ini.

Tabel 2.2 Kesembilan aturan penghemat tagihan listrik: enam tentang satu peralatan dan tiga tentang rumah.

Aturan	JIKA	MAKA (saran)	Akibat untuk menghitung penghematan
`ac-setpoint`	jenisnya `ac` dan disetel di bawah 25 °C	setel ke 25 °C	tidak ada (bergantung pada ruangan)
`ac-timer`	jenisnya `ac` dan dipakai lebih dari 8 jam	timer mati setelah 8 jam	jam menjadi 8
`old-bulb`	jenisnya `lamp-bulb`	ganti dengan LED	daya menjadi 15 %
`lights-daytime`	lampu apa pun menyala lebih dari 12 jam	matikan ketika siang	jam menjadi 12
`keep-warm`	jenisnya `rice-warm`, lebih dari 4 jam	masak lebih dekat ke waktu makan	jam menjadi 4
`standby`	ada daya siaga dan tidak dipakai sepanjang hari	matikan di stopkontak	siaga menjadi 0
`biggest-eater`	satu peralatan memakai 30 % atau lebih	periksa ini lebih dahulu	tidak ada
`overload`	jumlah seluruh daya melampaui VA	jangan nyalakan peralatan berat bersamaan	tidak ada (keselamatan)
`budget`	tagihan di atas anggaran	tagihan baru dengan saran	jumlah semua akibat

2.7.2 Menambah sebuah aturan

Misalkan seorang penasihat menginginkan aturan untuk pemanas air: *jika pemanas air listrik menyala lebih dari dua jam sehari, sarankan menyalakannya hanya setengah jam sebelum mandi.* Dalam rancangan kita, ini adalah satu nilai baru dalam daftar yang dikembalikan `rules()`, dengan jenis `heater`, dan tidak satu baris pun di mesinnya berubah. Hal yang sama berlaku untuk menghapus aturan, atau mengubah angka seperti 25 °C. Soal Latihan 2.5 meminta Anda menambahkan aturan ini; penyelesaiannya menunjukkan betapa sedikit baris yang dibutuhkan.

2.7.3 Menguji aturan-aturan

Program aturan yang diandalkan orang memerlukan **uji**: program-program kecil yang memeriksa aritmetika dan aturan secara otomatis, sehingga perubahan yang merusak sesuatu langsung ketahuan. Folder bab ini memiliki `BillSaverTest.java` dan `test_bill_saver.py`. Keduanya memeriksa bahwa `"1.5"` menjadi 150, bahwa AC memakai tepat 65.070.000 desiwatt-menit, bahwa tagihan Bu Sri Rp 407.600, dan bahwa aturan-aturan terpenuhi dalam urutan yang persis sama dengan laporan. Jalankan dengan `java -cp out BillSaverTest` dan `python -m unittest test_bill_saver -v`; uji Java berakhir dengan:

```
ok    LED for 60 W
all tests passed
```

2.7.4 Satu peralatan melewati mesin

Menelusuri satu peralatan melewati perulangan ganda dengan tangan, seperti yang dikerjakan komputer, sangat membantu. Ambillah lampu dapur, baris keempat berkas: jenis `lamp-bulb`, 600 sepersepuluh watt, satu lampu, 840 menit sehari, tanpa siaga. Pada awal gilirannya, `changed` adalah lampu itu sendiri.

1. `ac-setpoint`: apakah jenisnya `ac`? Bukan. Kondisinya salah; `continue`.

2. `ac-timer`: jenis `ac`? Bukan; `continue`.
3. `old-bulb`: jenis `lamp-bulb`? Ya. Aturan terpenuhi: butir 4 laporan. Daya LED-nya $(600 \times 15 + 50)/100 = 90$ sepersepuluh watt, 9 W. Penghematan yang dicetak adalah energi lampu dikurangi energi salinan 9 W: $600 \times 840 \times 30 - 90 \times 840 \times 30 = 15{,}120{,}000 - 2{,}268{,}000 = 12{,}852{,}000$ desiwatt-menit, yaitu 21,42 kWh, dicetak 21.4. Kini `changed` menjadi salinan 9 W.
4. `lights-daytime`: apakah jenisnya diawali `lamp`, dan menyala lebih dari 720 menit? Ya, 840. Aturan terpenuhi: butir 5. Penghematan yang *dicetak* dihitung pada lampu aslinya, $600 \times 120 \times 30 = 2{,}160{,}000$, atau 3,6 kWh — yang akan dihemat saran ini sendiri. Tetapi `changed` kini menjadi salinan 9 W dengan 720 menit, sehingga dalam *total*, aturan ini hanya dihitung sebesar 0.54 kWh yang masih dipakai LED itu dalam dua jam tersebut.
5. `keep-warm`: jenis `rice-warm`? Bukan.
6. `standby`: daya siaga di atas nol? Tidak.

Entri lampu ini di `after` adalah LED 9 W yang menyala 12 jam. Kedua penghematan yang dicetak, 21,4 dan 3,6 kWh, memberi tahu Bu Sri nilai setiap saran *bila ia hanya melakukan saran itu*; total di bagian akhir memberi tahu nilainya *bila dilakukan bersama-sama*. Keduanya adalah jawaban jujur atas pertanyaan yang berbeda, dan laporannya berhati-hati memberikan keduanya.

2.7.5 Dari daftar aturan ke basis pengetahuan

Lihatlah **Tabel 2.2** sekali lagi dengan kosakata Bab 1. Setiap aturan adalah rangkaian **kondisi** atas **variabel** sebuah peralatan — `kind`, `minutes`, `setting`, `standby` — dengan **operator** dan **nilai**, serta sebuah **konklusi**, yaitu sarannya. Dalam bahasa VCIRS, setiap baris hampir sama dengan sebuah **node**. Yang tidak dimiliki daftar kita adalah struktur di antara aturan-aturan itu. Kedua aturan lampu sama-sama terpenuhi pada lampu dapur, dan tidak ada yang mencatat bahwa aturan kedua baru penting setelah aturan pertama; bila seorang penasihat ingin berkata "kecuali bila lampu itu lampu penerangan untuk keamanan, jangan pernah sarankan mematikannya pada malam hari", ia harus menulis ulang aturannya, bukan menambahkan sebuah pengecualian. Dan aturan-aturan ini tidak dapat diajari: setiap perubahan adalah perubahan pada program. Bab berikutnya membalik hal itu. Bab itu mempertahankan pemisahan pengetahuan dari mesin yang kita bangun di sini, dan menambahkan cara bagi orang yang tahu untuk mengoreksi pengetahuan itu satu kasus demi satu kasus.

2.7.6 Yang tidak dapat dilihat aturan

Kita perlu jujur tentang batas-batasnya. Aturan hanya mengetahui apa yang dikatakan berkas rumah tangga. Aturan tidak tahu bahwa lampu teras menyala sepanjang malam demi keamanan, atau bahwa kulkasnya sudah berumur lima belas tahun dan karet pintunya robek; orang yang membaca saran itu tetap harus menilainya. Aturan tidak mengurutkan saran menurut besarnya usaha, padahal sebuah keluarga lebih memilih mengganti bohlam daripada mengatur ulang jadwal memasaknya. Dan aturan tidak dapat belajar: bila seorang penasihat menemukan keadaan baru, seseorang harus menuliskan nilai baru di dalam program. Bab 3 menghapus batas terakhir itu dengan basis pengetahuan yang belajar dari setiap kasus; Bab 7 menunjukkan bagaimana menghitung aturan mana yang benar-benar dipakai dapat mengurutkannya.

2.7.7 Mencocokkan penghemat dengan meteran

Penghemat yang tidak pernah diperiksa hanyalah tebakan yang memakai angka desimal. Untungnya rumah memiliki saksi yang jujur: meteran listrik. Bagi pelanggan pascabayar, tagihan itu sendiri memperlihatkan kWh yang terpakai dalam sebulan, yaitu selisih dua kali pembacaan

meteran; bagi pelanggan prabayar, kWh yang dibeli dengan setiap token tercetak pada struk atau tampil di aplikasi pembayaran, dan layar meteran menghitung mundur ketika kWh itu dipakai. Dengan cara mana pun, keluarga dapat meletakkan total dari penghemat di samping angka yang tidak dihasilkan oleh penghemat itu sendiri.

Misalkan tagihan Bu Sri bulan lalu menunjukkan 305 kWh, sedangkan penghemat menyebut 282,1 kWh. Selisihnya $305 - 282.1 = 22.9$ kWh, sekitar 7,5 persen dari tagihan. Itu kecocokan yang baik untuk berkas yang disusun dari label dan perkiraan: sebulan tidak selalu tepat tiga puluh hari, kulkas bekerja lebih keras ketika cuaca panas, dan tidak ada yang menghitung malam-malam ketika televisi menyala lebih lama dari biasanya. Bila selisihnya 30 atau 40 persen, berarti ada sesuatu yang terlewat dalam berkas itu, dan ada urutan yang baik untuk mencarinya.

1. **Peralatan yang terlupa.** Dispenser air yang memanaskan dan mendinginkan sepanjang hari, pompa akuarium, kulkas kedua di warung depan, sepeda motor listrik yang diisi dayanya pada malam hari. Kelilingi rumah sekali lagi dan tambahkan setiap steker yang dipakai.
2. **Peralatan terbesar lebih dahulu.** Kesalahan satu jam sehari pada AC 400 W berarti $400 \times 1 \times 30/1000 = 12$ kWh sebulan; kesalahan yang sama pada lampu 9 W hanya 0,27 kWh. Periksa jam pemakaian tiga baris teratas laporan sebelum yang lainnya.
3. **Kulkas.** Jam kulkas paling sulit ditebak, karena motornya menyala dan mati sendiri. Ukurlah dengan pengukur energi colok selama sehari penuh, seperti yang dijelaskan Subbab 2.8, alih-alih menebaknya.
4. **Label yang berupa rentang atau nilai maksimum.** Sebagian label menyebut daya pada beban penuh, dan sebagian peralatan, seperti kipas pada kecepatan terendah, memakai lebih sedikit. Penghemat yang disusun dari nilai maksimum biasanya menghasilkan angka sedikit *di atas* meteran, bukan di bawahnya.

Bila berkas dan meteran cocok sampai kira-kira sepersepuluh, saran-sarannya dapat dipercaya dalam arti yang penting: penghematan yang dicetaknya adalah penghematan pada tagihan yang mirip dengan tagihan sebenarnya. Bila tidak cocok, langkah yang benar adalah memperbaiki faktanya, bukan aturannya. Pembagian kerja ini sama dengan yang menjadi tumpuan seluruh buku: aturan menyimpan apa yang diketahui seorang penasihat secara umum, fakta menyimpan apa yang benar tentang rumah yang satu ini, dan masing-masing diperbaiki di tempatnya sendiri.

Meteran juga memberi tahu keluarga apakah saran-saran itu dijalankan dan berhasil. Bacalah meteran pada tanggal yang sama setiap bulan, atau catat kWh token yang dibeli, lalu tuliskan angkanya di samping nama bulan. Bila bohlam dapur sudah diganti dan rice cooker kini hanya menghangatkan selama empat jam, bulan-bulan berikutnya seharusnya menunjukkan kira-kira 60 kWh lebih sedikit daripada sebelumnya; bila tidak, ada sesuatu di rumah yang tidak berubah sesuai rencana, dan keluarga itu tahu di mana harus mencarinya.

2.8 Memakainya Sungguh-Sungguh

Rumah Anda sendiri. Salinlah `home-sri.csv`, pertahankan baris-baris pertamanya dan baris judulnya, lalu tuliskan satu baris untuk setiap peralatan. Kelilingi rumah dengan buku catatan: bacalah label setiap peralatan (dayanya tertulis dalam W, kadang sebagai rentang seperti 350–400 W; ambillah angka yang lebih besar), dan tanyakan kepada keluarga berapa jam sehari setiap peralatan menyala. Untuk jenisnya, pakai jenis-jenis dalam **Tabel 2.2** bila cocok; jenis lain, seperti `laptop` atau `kettle`, tetap dihitung dalam tagihan meskipun tidak ada aturan tentangnya.

Ukur, jangan menebak. Alat pengukur energi colok, yang dijual murah di toko elektronik, dipasang di antara stopkontak dan peralatan, lalu menampilkan daya dan kWh yang terpakai.

Colokkan kulkas melaluinya selama sehari penuh: kWh yang ditampilkannya, dibagi daya pada label, adalah jumlah jam sebenarnya motor kulkas bekerja. Alat yang sama menunjukkan daya siaga televisi atau pengisi daya ketika "mati". Kementerian ESDM juga menyediakan aplikasi kalkulator hemat energi rumah tangga yang memperkirakan pemakaian dari daftar serupa [7].

Membeli peralatan baru. Ketika sebuah peralatan memang harus diganti, label hemat energi pada AC dan peralatan lain, dengan bintang-bintangnya, menunjukkan seberapa efisien setiap model; pemerintah mewajibkan label seperti itu dan standar kinerja energi minimum untuk AC [8]. Masukkan daya model yang baru ke dalam berkas, dan penghemat akan memperlihatkan selisihnya.

Triwulan tarif yang baru. Ketika pemerintah mengumumkan tarif untuk triwulan berikutnya, salinlah `tariff-2026-q3.csv` dengan nama baru, ubah tarifnya, lalu berikan berkas baru itu sebagai argumen kedua. Tidak ada hal lain yang berubah. Hal yang sama berlaku untuk baris-baris pajak pada tagihan sungguhan: untuk memasukkan Pajak Penerangan Jalan daerah sebesar, misalnya, 3 persen, kalikan tagihan dengan 103 lalu bagi dengan 100 (dengan pembulatan dari Subbab 2.5.3) — Soal Latihan 2.6 mengerjakan persis hal ini.

Meteran prabayar. Pelanggan prabayar (*token*) membayar harga per kWh yang sama, tetapi di muka. kWh bulanan dari penghemat kemudian adalah jumlah kWh token yang dibutuhkan sebulan, dan penghematannya menyatakan berapa banyak token yang tidak perlu dibeli.

Warung atau laundry kecil. Format berkas yang sama berlaku untuk toko: kulkas pajangan, lampu, kipas, blender; atau untuk laundry: mesin cuci, pengering, dan setrika, yang dayanya besar dan jam pemakaiannya panjang. Tambahkan aturan yang sesuai dengan usahanya — *jika kulkas pajangan setengah kosong, naikkan setelan suhunya pada malam hari* — sebagai nilai-nilai baru.

Malam hemat energi di lingkungan. Sebuah kelompok warga, pengurus masjid, atau pertemuan bulanan RT dapat menjalankan penghemat untuk beberapa rumah dalam satu malam: seorang relawan membaca label, seorang lagi menanyakan jam pemakaian, orang ketiga mengetik berkasnya di laptop dan mencetak laporannya. Karena aturannya sama untuk semua orang, keluarga-keluarga dapat membandingkan laporan mereka, dan saran yang paling umum — biasanya bohlam lama dan rice cooker — dapat dilaksanakan bersama, misalnya dengan membeli lampu LED secara borongan. Simpanlah berkas setiap keluarga hanya untuk keluarga itu; angka-angkanya memperlihatkan kebiasaan sehari-hari, dan sebuah laporan hanya untuk rumah tangga yang digambarkannya.

Menjaga aturan tetap mutakhir. Angka-angka di dalam aturan juga menua. Lampu LED terus menjadi lebih efisien, sehingga 15 persen dalam `LED_PERCENT` mungkin menjadi 12; rice cooker yang baru mungkin menghangatkan dengan daya yang lebih kecil; seorang penasihat mungkin memutuskan bahwa sembilan jam AC, bukan delapan, adalah malam yang wajar. Setiap perubahan seperti itu hanyalah satu konstanta di bagian atas `BillSaver.java` atau `bill_saver.py`, dan setelah mengubahnya, jalankan uji-ujinya: uji akan gagal persis di tempat angka-angka lama tertulis di dalamnya, yang menjadi pengingat untuk memeriksa saran yang baru dengan tangan sebelum memercayainya. Simpanlah catatan singkat di folder bab tentang apa yang diubah, kapan, dan mengapa, sehingga siapa pun yang merawat penghemat ini tahun depan tahu dari mana setiap angka berasal.

Apa saran ini dan apa yang bukan. Penghemat ini memperkirakan; ia tidak menjanjikan. Angka-angkanya hanya sebaik watt dan jam yang Anda berikan, dan tagihan sungguhan juga bergantung pada tegangan, umur peralatan, dan cuaca. Ia juga bukan saran keuangan tentang apa yang sebaiknya dibeli: apakah AC inverter akan balik modal adalah keputusan keluarga, dengan

harga dari toko. Dan ia tidak pernah menggantikan instalatir: bila lampu berkedip, stopkontak terasa hangat, atau MCB sering turun, hubungi PLN atau instalatir listrik bersertifikat.

> **JEBAKAN:** Jangan pernah "memperbaiki" pemutus arus yang sering turun dengan menggantinya dengan yang lebih besar atau dengan menjumpernya. Pemutus arus melindungi kabel di dalam dinding dari kepanasan. Bila sebuah rumah benar-benar membutuhkan daya lebih besar, sambungannya dapat ditambah secara resmi melalui PLN.

2.9 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Mencampuradukkan watt dan watt-jam. Watt menyatakan seberapa cepat; watt-jam menyatakan seberapa banyak. Bohlam 60 W tidak "memakai 60 W sebulan". *Hindari:* selalu kalikan dengan jam, memakai Persamaan (2.1).

Lupa membagi 1000. Watt-jam seribu kali lebih besar daripada kilowatt-jam; tagihan yang seribu kali terlalu besar adalah tandanya. *Hindari:* periksa hasilnya dengan tangan pada satu peralatan, seperti di Subbab 2.2.1. (Versi pertama metode `kwh` aplikasi ini melakukan kesalahan serupa dengan desiwatt-menit dan mencetak 1084.5 kWh untuk AC; membandingkannya dengan perhitungan tangan langsung menemukannya.)

Menghitung penghematan yang sama dua kali. Dua aturan tentang satu lampu masing-masing dapat mengklaim kilowatt-jam yang sudah dihemat aturan lainnya. *Hindari:* terapkan perubahan satu per satu pada salinan berjalan, seperti yang dilakukan mesin dengan `changed`.

Membiarkan bahasa komputer mengubah angka. Tanpa `Locale.US`, Java mencetak `156.028` di komputer berbahasa Indonesia. *Hindari:* sebutkan locale secara eksplisit bila keluarannya harus sama di mana pun.

Membulatkan desimal secara berbeda di dua bahasa. Pecahan biner dapat membuat Java dan Python membulatkan angka setengah secara berbeda. *Hindari:* untuk uang, hitunglah dengan bilangan bulat dalam satuan terkecil (sen), seperti Persamaan (2.4).

Aturan tanpa alasan. Saran seperti "matikan" tanpa alasan akan diabaikan. *Hindari:* setiap aturan dalam daftar memiliki `why`.

Aturan yang mengarang penghematan. Menebak penghematan dari perubahan suhu akan menyesatkan. *Hindari:* bila penghematan tidak dapat dihitung dari fakta, katakan demikian, seperti yang dilakukan `ac-setpoint`.

Menaruh aturan di dalam mesin. `if` khusus untuk satu peralatan yang disembunyikan di dalam perulangan tidak terlihat dalam daftar aturan dan merusak rancangannya. *Hindari:* setiap saran adalah sebuah nilai dalam `rules()`.

Membandingkan teks dengan `==` di Java. Jenis peralatan adalah teks; bandingkan dengan `.equals`, seperti yang dilakukan aturan-aturan itu.

Lebih memercayai perkiraan daripada meteran. Bila penghemat dan tagihan sungguhan berselisih jauh, yang salah adalah berkas rumah tangganya, bukan meterannya. *Hindari:* ukurlah peralatan-peralatan terbesar dengan pengukur energi colok dan perbaiki berkasnya.

Membandingkan perkiraan tiga puluh hari dengan tagihan tiga puluh satu hari. Penghemat selalu menghitung 30 hari; tagihan di antara dua pembacaan meteran dapat meliputi 28 sampai 31 hari. *Hindari:* kalikan total penghemat dengan jumlah hari pada tagihan dibagi 30 sebelum membandingkannya, seperti dalam Soal Latihan 2.9.

2.10 Ringkasan Bab

- **Energi** bulanan sebuah peralatan adalah daya × jumlah × jam × 30 / 1000 kWh (Persamaan (2.1)); **siaga** menambahkan daya siaga selama jam-jam ketika tidak dipakai (Persamaan (2.2)); **tagihan** adalah energi × tarif (Persamaan (2.3)), dengan tarif PLN Juli–September 2026 pada **Tabel 2.1**.
- **Aturan penghematan** memeriksa satu peralatan, memberi **saran** dan **alasan**, dan — bila memang dapat dilakukan secara jujur — menggambarkan peralatan itu sebagaimana jadinya setelah saran diikuti; **penghematannya** adalah energi sebelum dikurangi energi sesudah.
- Aturan yang ditulis langsung sebagai pernyataan `if` (**Kode 2.3**, **Kode 2.4**) mencampur pengetahuan dengan mesin. Bila ditulis sebagai **nilai** — kondisi, saran, alasan, akibat — aturan-aturan itu berada dalam satu daftar dan sebuah perulangan kecil menerapkannya (**Kode 2.8**, **Kode 2.9**, **Kode 2.10**); menambah aturan mengubah daftarnya, tidak pernah mesinnya (**Gambar 2.1**, **Tabel 2.2**).
- **Salinan yang diubah** dari peralatan memungkinkan beberapa aturan memeriksa peralatan yang sama, dan menerapkan perubahan satu per satu pada salinan berjalan menghitung setiap penghematan sekali saja.
- Aritmetika uang dengan **bilangan bulat** — sepersepuluh watt, menit, sen, dan pembulatan dengan menambahkan separuh pembagi (Persamaan (2.4)) — membuat Java dan Python sepakat sampai ke rupiah terakhir.
- Untuk rumah Bu Sri penghemat memberikan tiga belas saran dan menunjukkan bahwa tagihan akan turun dari Rp 407.600 menjadi Rp 317.603 sebulan (**Gambar 2.2**); aturan yang sama memberi saran untuk kamar Dina tanpa perubahan.
- Aplikasi ini memberi saran; ia tidak menjamin, tidak memberi saran keuangan, dan tidak menggantikan instalatir listrik.

2.11 Soal Latihan

2.1 Dengan tangan, memakai Persamaan (2.1) dan Persamaan (2.3), hitunglah energi dan biaya bulanan pompa air Bu Sri (250 W, 1,5 jam sehari) dengan tarif `R1-1300`. Bandingkan dengan laporan.

2.2 Sebuah keluarga dengan tarif 900 VA bersubsidi (Rp 605 per kWh) mengganti empat bohlam 40 W, masing-masing menyala 6 jam sehari, dengan LED 6 W. Berapa kWh dan berapa rupiah yang dihemat sebulan? Dan dengan tarif 900 VA nonsubsidi?

2.3 Dengan memakai Persamaan (2.4), hitunglah dengan tangan, dengan bilangan bulat, biaya router Wi-Fi (10 W, 24 jam sehari) dalam sen dan rupiah dengan tarif `R1-1300`. Periksa jawaban Anda dengan laporan.

2.4 Mengapa aturan `standby` mensyaratkan `minutes < 24 * 60`? Sebutkan sebuah peralatan dalam berkas Bu Sri yang terkait dengan kondisi ini.

2.5 Tambahkan aturan dari Subbab 2.7.2 pada kedua versi: *JIKA jenisnya `heater` DAN menyala lebih dari 2 jam sehari MAKA nyalakan hanya 30 menit sebelum mandi*, lengkap dengan alasan dan akibatnya. Tambahkan pemanas air 350 W yang dipakai 3 jam sehari ke salinan berkas Bu Sri dan jalankan penghemat.

2.6 Banyak daerah menambahkan Pajak Penerangan Jalan pada tagihan. Ubahlah program agar berkas rumah tangga boleh memuat baris `tax,3`, yang berarti 3 persen, dan laporan mencetak tagihan beserta pajaknya. Pakailah aritmetika bilangan bulat.

2.7 Tuliskan berkas rumah tangga untuk rumah Anda sendiri (atau rumah yang Anda kenal), jalankan penghemat, dan tuliskan tiga kalimat untuk keluarga itu: peralatan paling boros, saran terbaik, dan apakah saran-saran itu sepadan dengan usahanya.

2.8 Aturan `overload` menjumlahkan daya *semua* peralatan, padahal rumah sungguhan tidak pernah menyalakan semuanya sekaligus. Usulkan aturan yang lebih baik, dengan kata-kata: fakta apa yang dibutuhkannya, dan apa yang akan dihitungnya?

2.9 Tagihan Bu Sri berikutnya meliputi 31 hari dan menunjukkan 300 kWh. (a) Apakah perkiraan penghemat sebesar 282,1 kWh, yang menghitung 30 hari, cocok? (b) Setelah keluarga itu mengatakan sudah menjalankan semua saran, tagihan 30 hari menunjukkan 233,3 kWh. Penghemat memperkirakan $282.1 - 62.3 = 219.8$ kWh. Saran mana yang paling mungkin tidak dijalankan, dan apa yang sebaiknya dilakukan keluarga itu untuk memastikannya?

2.12 Penyelesaian Soal Latihan

Penyelesaian 2.1. $E = 250 \times 1 \times 1.5 \times 30/1000 = 11.25$ kWh, dan $11.25 \times 1444.70 = 16{,}252.9$, jadi sekitar Rp 16.253. Laporan mencetak 11.3 kWh (11,25 dibulatkan ke satu desimal, setengah ke atas) dan Rp 16,253.

Penyelesaian 2.2. Sebelum: $40 \times 4 \times 6 \times 30/1000 = 28.8$ kWh; sesudah: $6 \times 4 \times 6 \times 30/1000 = 4.32$ kWh; penghematannya 24.48 kWh sebulan. Dengan Rp 605 per kWh, itu $24.48 \times 605 = 14{,}810.4$, sekitar Rp 14.810; dengan Rp 1.352, sekitar Rp 33.097 (dibulatkan dari 33,096.96). LED yang sama menghemat uang lebih dari dua kali lipat bagi rumah tangga nonsubsidi.

Penyelesaian 2.3. Dalam satuan aplikasi, router memakai $e = 100 \times 1440 \times 30 = 4{,}320{,}000$ desiwatt-menit (10 W adalah 100 sepersepuluh watt; 24 jam adalah 1440 menit), yaitu 7,2 kWh. Menurut Persamaan (2.4): $(4{,}320{,}000 \times 144{,}470 + 300{,}000)/600{,}000 = 624{,}110{,}700{,}000/600{,}000 = 1{,}040{,}184.5$, dibulatkan ke bawah menjadi 1,040,184 sen. Dalam rupiah: $(1{,}040{,}184 + 50)/100 = 10{,}402.34$, dibulatkan ke bawah menjadi Rp 10.402 — angka dalam laporan.

Penyelesaian 2.4. Peralatan yang dipakai sepanjang 24 jam tidak pernah "mati": ia tidak memiliki jam siaga, sehingga menurut Persamaan (2.2) energi siaganya nol dan mencabut stekernya tidak menghemat apa pun — malah menghentikan peralatan yang memang harus bekerja sepanjang hari. Dalam berkas Bu Sri, router Wi-Fi menyala 24 jam; field siaganya 0, sehingga aturan itu tidak terpenuhi dengan atau tanpa kondisi tersebut, tetapi router yang labelnya menyebut, misalnya, daya siaga 0,5 W akan mendapat saran yang janggal "matikan di stopkontak ketika tidak dipakai" tanpa kondisi `minutes < 24 * 60`.

Penyelesaian 2.5. Dalam Java, tambahkan pada `rules()`:

```
/** Soal Latihan 2.5: aturan hemat untuk pemanas air listrik. */
public class HeaterRule {
    static SavingRule rule() {
        return new SavingRule("heater",
                a -> a.kind.equals("heater") && a.minutes > 2 * 60,
                a -> "switch it on only 30 minutes before bathing",
                a -> "it is a water heater on " + a.minutes / 60.0
                        + " hours a day",
                a -> a.withMinutes(30));
    }
}
```

dan tambahkan `r.add(HeaterRule.rule());` sebelum `return r;` (folder penyelesaian menyimpan aturan itu dalam kelasnya sendiri, `HeaterRule.java`, agar `BillSaver.java` tetap seperti yang tercetak di buku). Penyelesaian Python menambahkan, di `rules()`:

```
"""Soal Latihan 2.5: aturan hemat untuk pemanas air listrik."""

from bill_saver import SavingRule


def heater_rule():
    return SavingRule(
        "heater",
        lambda a: a.kind == "heater" and a.minutes > 2 * 60,
        lambda a: "switch it on only 30 minutes before bathing",
        lambda a: "it is a water heater on " + str(a.minutes / 60)
                  + " hours a day",
        lambda a: a.with_minutes(30))
```

Dengan baris `Water heater,heater,350,1,3,0,` ditambahkan ke salinan berkas (`home-sri-heater.csv`), penghemat yang menjalankan aturan tambahan itu (`BillSaverHeater`, `bill_saver_heater.py`) mencetak, di antara saran-sarannya:

```
11. [heater] Water heater
    advice: switch it on only 30 minutes before bathing
    why: it is a water heater on 3.0 hours a day
    saving: 26.3 kWh = Rp 37,923 a month
```

Penyelesaian 2.6. Bacalah baris itu di `Household.load` (`tax` → sebuah field `taxPercent`), dan setelah menghitung `bill`, hitunglah pajaknya dengan bilangan bulat: `tax = (bill * taxPercent + 50) / 100` di Java, `(bill * tax_percent + 50) // 100` di Python, dibulatkan ke rupiah terdekat. Cetak `bill`, `tax`, dan `bill + tax`. Untuk Bu Sri dengan 3 persen, pajaknya $(407{,}600 \times 3 + 50)/100 = 12{,}228.5$, dibulatkan ke bawah menjadi Rp 12.228, dan totalnya Rp 419.828. Penghematan juga memperkecil pajaknya, sebesar 3 persen dari penghematan itu.

Penyelesaian 2.7. Jawaban dapat berbeda-beda. Jawaban yang baik menyebut peralatan dengan bagian terbesar, saran dengan penghematan terbesar untuk setiap satuan usaha (biasanya mengganti bohlam lama, atau mengurangi jam menghangatkan nasi atau jam AC), dan membandingkan penghematan total dengan usaha dan biaya yang diperlukan, misalnya harga lampu LED, yang biasanya balik modal dalam beberapa bulan.

Penyelesaian 2.8. Aturan yang lebih baik membutuhkan, untuk setiap peralatan berat, waktu pemakaiannya dalam sehari (misalnya "setrika 19.00–19.30", "pompa 05.00–06.30"), lalu menghitung, untuk setiap saat dalam sehari, jumlah daya peralatan yang menyala pada saat itu; aturan itu hanya memperingatkan bila jumlah terbesar melampaui VA, dan menyebutkan peralatan-peralatan yang tumpang-tindih pada saat itu. Dengan hanya mengetahui jam pemakaian, seperti dalam berkas kita, hal yang jujur adalah aturan yang sekarang: sebuah peringatan, bukan kepastian.

Penyelesaian 2.9. (a) Untuk 31 hari, perkiraan penghemat menjadi $282.1 \times 31/30 \approx 291.5$ kWh. Meteran menunjukkan $300 - 291.5 = 8.5$ kWh lebih banyak, di bawah 3 persen: kecocokan yang sangat baik, lebih baik daripada 7,5 persen di Subbab 2.7.7 setelah hari tambahan ikut dihitung. (b) Meteran menunjukkan $233.3 - 219.8 = 13.5$ kWh lebih banyak daripada perkiraan. Angka itu persis penghematan butir 8, menghangatkan nasi selama empat jam alih-alih empat belas jam, dan dekat dengan 12,0 kWh dari timer AC, butir 2; butir-butir siaga yang kecil tidak dapat menjelaskannya. Jawaban yang paling mungkin adalah rice cooker, tetapi angka-angka itu saja tidak dapat membedakan keduanya. Keluarga itu sebaiknya bertanya siapa mengerjakan apa di

rumah, atau mencolokkan rice cooker melalui pengukur energi colok selama sehari dan membaca berapa kWh yang dipakainya; sehari dengan empat belas jam menghangatkan menunjukkan sekitar 0,63 kWh, sehari dengan empat jam sekitar 0,18 kWh.

Daftar Pustaka

[1] M. M. I. Subakti, "A Variable-Centered Intelligent Rule System," M.Sc.Eng. thesis, Dept. of Computer Science and Information Engineering, Taipei, Taiwan, Jun. 2005.

[2] Fortune Indonesia, "Daftar tarif listrik terbaru Agustus 2026, tidak naik," Aug. 2026. [Online]. Available: https://www.fortuneidn.com/news/tarif-listrik-terbaru-agustus-2026-d7w02-00-lflgw-prph33

[3] CNBC Indonesia, "Tarif listrik terbaru PLN per kWh, berlaku hingga September 2026," Aug. 10, 2026. [Online]. Available: https://www.cnbcindonesia.com/news/20260810101545-4-757861/tarif-listrik-terbaru-pln-per-kwh-berlaku-hingga-september-2026

[4] Kementerian Energi dan Sumber Daya Mineral, "3M, resep hemat energi dari Kementerian ESDM," Ditjen EBTKE, Nov. 8, 2017. [Online]. Available: https://www.esdm.go.id/id/berita-unit/direktorat-jenderal-ebtke/3m-resep-hemat-energi-dari-kementerian-esdm

[5] Oracle, "Class Formatter," *Java SE 25 & JDK 25 API Specification*, 2025. [Online]. Available: https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/util/Formatter.html

[6] Python Software Foundation, "Format specification mini-language," *The Python Standard Library*, 3.14. [Online]. Available: https://docs.python.org/3/library/string.html#formatspec

[7] Kementerian Energi dan Sumber Daya Mineral, "Ayo unduh aplikasi kalkulator hemat energi rumah tangga," Ditjen EBTKE. [Online]. Available: https://www.esdm.go.id/id/berita-unit/direktorat-jenderal-ebtke/ayo-unduh-aplikasi-kalkulator-hemat-energi-rumah-tangga

[8] Kementerian Energi dan Sumber Daya Mineral, "Pemerintah terapkan SKEM dan pencantuman label hemat energi untuk peranti pengkondisi udara," Ditjen EBTKE. [Online]. Available: https://www.esdm.go.id/id/berita-unit/directorate-general-ebtke/pemerintah-terapkan-skem-dan-pencantuman-label-hemat-energi-untuk-peranti-pengkondisi-udara

Bab 3

Pemecah Masalah Ponsel dan Wi-Fi yang Belajar dari Setiap Perbaikan

Setelah menyelesaikan bab ini Anda akan mampu menjelaskan mengapa daftar aturan yang datar semakin sulit dirawat setiap kali sebuah aturan ditambahkan; menguraikan jawaban Ripple Down Rules — pengecualian yang dipasang dalam konteks tempat sebuah kesalahan terjadi, diperiksa terhadap kasus-kasus *cornerstone*, dan dipilih dari daftar perbedaan; membangun mesin Ripple Down Rules klasifikasi tunggal yang lengkap dari nol dalam Java dan Python, dalam sekitar seratus baris; dan memakai mesin bersama vcirs-core untuk menjalankan **dokter Wi-Fi**, pemecah masalah ponsel dan Wi-Fi yang hanya menanyakan pertanyaan yang diperlukannya, menjelaskan sarannya, dan belajar perbaikan yang benar dari orang di balik etalase setiap kali sarannya gagal. Anda akan menjalankan empat percakapan sungguhan dengannya, menyaksikan pengetahuannya bertambah, dan melihat basis pengetahuan tersimpan yang dibaca oleh versi Java maupun versi Python.

3.1 Konter Mas Anto

Konter ponsel Mas Anto adalah etalase kaca sepanjang dua meter di sudut pasar: casing ponsel, pengisi daya, kartu SIM, dan sebuah bangku kecil untuk pelanggan yang butuh bantuan. Menurut hitungannya sendiri, separuh orang yang duduk di bangku itu mengatakan hal yang sama: *internetnya mati*. Pada 2026 Indonesia memiliki sekitar 235 juta pengguna internet, lebih dari delapan dari sepuluh penduduk [1], dan bagi kebanyakan dari mereka internet *adalah* ponsel. Ketika internet berhenti, mereka datang kepada orang-orang seperti Mas Anto.

Selama bertahun-tahun ia belajar apa yang harus ditanyakan. *Pakai Wi-Fi atau data seluler? Simbol Wi-Fi-nya menyala? Pesawat kecilnya menyala? Ponsel lain di rumah bisa internetan? Lampu router-nya warna apa?* Dan ia belajar apa yang harus dilakukan dengan jawaban-jawaban itu. Pengetahuannya tidak berasal dari buku. Pengetahuan itu berasal dari pelanggan, dan dalam urutan yang sangat khas: mula-mula sebuah aturan sederhana yang sering berhasil, lalu seorang pelanggan yang tidak tertolong olehnya, lalu koreksi untuk pelanggan *jenis itu*. “Nyalakan Wi-Fi” — kecuali bila mode pesawat menyala. “Restart router-nya” — kecuali bila lampu router merah, karena berarti masalahnya ada pada kabel penyedia layanan, dan me-restart hanya membuang waktu.

Mas Anto menginginkan dua hal. Ia ingin keponakannya, Fajar, yang menjaga konter setiap hari Jumat, memberikan saran yang sama dengan sarannya. Dan ia ingin pengetahuannya terus bertambah tanpa ada yang harus menulisnya ulang, karena setiap minggu datang jenis masalah baru. Bab ini membangun program yang melakukan keduanya. Inilah aplikasi pertama dalam buku ini yang *belajar*: setiap kali sarannya gagal, orang di balik etalase memberitahunya apa yang akhirnya memperbaiki masalah dan apa yang berbeda pada kasus ini, dan program menaruh pengetahuan itu tepat di tempatnya.

3.2 Gagasannya dengan Kata-Kata Sederhana

3.2.1 Mengapa daftar aturan yang datar rusak

Mari kita mulai dari rancangan yang paling jelas dan melihatnya gagal. Sebuah pemecah masalah dapat berupa daftar aturan yang dicoba dari atas; aturan pertama yang kondisinya cocok dengan fakta pelanggan memberikan jawabannya. **Kode 3.1** mencoba tiga versi daftar semacam itu pada empat pelanggan yang jawaban benarnya kita ketahui.

Kode 3.1 `FlatRules.java`: pemecah masalah sebagai daftar aturan yang datar, dicoba dalam tiga versi.

```
import java.util.List;
import java.util.Map;

/** Pemecah masalah berupa daftar aturan datar: aturan cocok pertama menang. */
public class FlatRules {
    record Rule(Map<String, String> when, String fix) {
        boolean matches(Map<String, String> facts) {
            for (Map.Entry<String, String> c : when.entrySet()) {
                if (!c.getValue().equals(facts.get(c.getKey()))) {
                    return false;
                }
            }
            return true;
        }
    }

    static String advise(List<Rule> rules, Map<String, String> facts) {
        for (Rule r : rules) {
            if (r.matches(facts)) {
                return r.fix();
            }
        }
        return "no advice";
    }

    public static void main(String[] args) {
        Rule wifiOff = new Rule(Map.of("wifi icon", "off"), "turn on Wi-Fi");
        Rule router = new Rule(Map.of("other devices online", "no"),
                "restart the router");
        Rule airplane = new Rule(Map.of("airplane mode", "on"),
                "turn off airplane mode");

        List<Map<String, String>> cases = List.of(
                Map.of("wifi icon", "off", "airplane mode", "off"),
                Map.of("wifi icon", "on", "other devices online", "no"),
                Map.of("wifi icon", "off", "airplane mode", "on"),
                Map.of("wifi icon", "on", "airplane mode", "on",
                        "other devices online", "no"));
        List<String> right = List.of("turn on Wi-Fi", "restart the router",
                "turn off airplane mode", "restart the router");
```

Kode 3.1 (Lanjutan) *FlatRules.java: pemecah masalah sebagai daftar aturan yang datar, dicoba dalam tiga versi.*

```
        List<List<Rule>> versions = List.of(
                List.of(wifiOff, router),
                List.of(wifiOff, router, airplane),
                List.of(airplane, wifiOff, router));
        List<String> names = List.of("Two rules", "New rule at the end",
                "New rule at the front");
        for (int v = 0; v < versions.size(); v++) {
            System.out.println(names.get(v) + ":");
            for (int i = 0; i < cases.size(); i++) {
                String got = advise(versions.get(v), cases.get(i));
                String mark = got.equals(right.get(i)) ? "right" : "WRONG";
                System.out.printf("  case %d: %-24s %s%n", i + 1, got, mark);
            }
        }
    }
}
```

`record` adalah cara singkat menulis kelas Java yang hanya menyimpan data: `Rule` menyimpan kondisi (`when`, sebuah map dari variabel ke nilai) dan perbaikannya. `matches` memeriksa bahwa nilai setiap kondisi sama dengan fakta pelanggan. `advise` mengembalikan perbaikan dari aturan pertama yang cocok. Para pelanggan adalah map berisi fakta, dan `right` menyimpan perbaikan yang benar-benar berhasil untuk masing-masing. **Kode 3.2** adalah hal yang sama dalam Python, dengan aturan yang cukup berupa pasangan sebuah dictionary dan sebuah perbaikan.

Kode 3.2 `flat_rules.py`: ketiga versi yang sama dalam Python.

```
"""Pemecah masalah berupa daftar aturan datar: aturan cocok pertama menang."""


def matches(when, facts):
    return all(facts.get(var) == value for var, value in when.items())


def advise(rules, facts):
    for when, fix in rules:
        if matches(when, facts):
            return fix
    return "no advice"


wifi_off = ({"wifi icon": "off"}, "turn on Wi-Fi")
router = ({"other devices online": "no"}, "restart the router")
airplane = ({"airplane mode": "on"}, "turn off airplane mode")

cases = [
    {"wifi icon": "off", "airplane mode": "off"},
    {"wifi icon": "on", "other devices online": "no"},
    {"wifi icon": "off", "airplane mode": "on"},
    {"wifi icon": "on", "airplane mode": "on", "other devices online": "no"},
]
right = ["turn on Wi-Fi", "restart the router", "turn off airplane mode",
         "restart the router"]

versions = [[wifi_off, router], [wifi_off, router, airplane],
            [airplane, wifi_off, router]]
names = ["Two rules", "New rule at the end", "New rule at the front"]
for name, rules in zip(names, versions):
    print(name + ":")
    for i, facts in enumerate(cases):
        got = advise(rules, facts)
        mark = "right" if got == right[i] else "WRONG"
        print(f"  case {i + 1}: {got:<24} {mark}")
```

Keduanya mencetak:

```
Two rules:
  case 1: turn on Wi-Fi            right
  case 2: restart the router       right
  case 3: turn on Wi-Fi            WRONG
  case 4: restart the router       right
New rule at the end:
  case 1: turn on Wi-Fi            right
  case 2: restart the router       right
  case 3: turn on Wi-Fi            WRONG
  case 4: restart the router       right
New rule at the front:
  case 1: turn on Wi-Fi            right
  case 2: restart the router       right
  case 3: turn off airplane mode   right
  case 4: turn off airplane mode   WRONG
```

Dengan dua aturan, pelanggan 3 — yang simbol Wi-Fi-nya mati *karena mode pesawat menyala* — mendapat saran yang salah. Mas Anto menambahkan aturan yang jelas, “jika mode pesawat menyala, matikan”. Bila ditambahkan di **akhir** daftar, aturan itu tidak pernah terpenuhi untuk pelanggan 3, karena aturan Wi-Fi di atasnya lebih dahulu cocok. Bila ditambahkan di **depan**, aturan itu memperbaiki pelanggan 3 — dan merusak pelanggan 4, yang ponselnya kebetulan berada dalam mode pesawat dengan Wi-Fi menyala (ponsel modern memperbolehkannya) sementara masalah sebenarnya ada pada router. Aturan yang benar untuk seorang pelanggan,

bila ditaruh dalam sebuah daftar, mengubah jawaban bagi pelanggan-pelanggan yang tidak pernah dimaksudkannya.

Inilah masalah perawatan sistem berbasis aturan dalam bentuk mini [2], [3]. Dengan tiga aturan kita dapat melihatnya; dengan tiga ratus, tidak ada yang bisa. Setiap aturan baru harus diperiksa terhadap setiap aturan lama, dan urutan aturan menjadi pengetahuan tersendiri, tak terlihat dan rapuh. Dalam praktik, inilah sebabnya basis aturan yang besar membutuhkan *knowledge engineer*, dan mengapa para pakar merasa begitu sulit menjaganya tetap mutakhir [4].

3.2.2 Pengetahuan datang bersama konteksnya

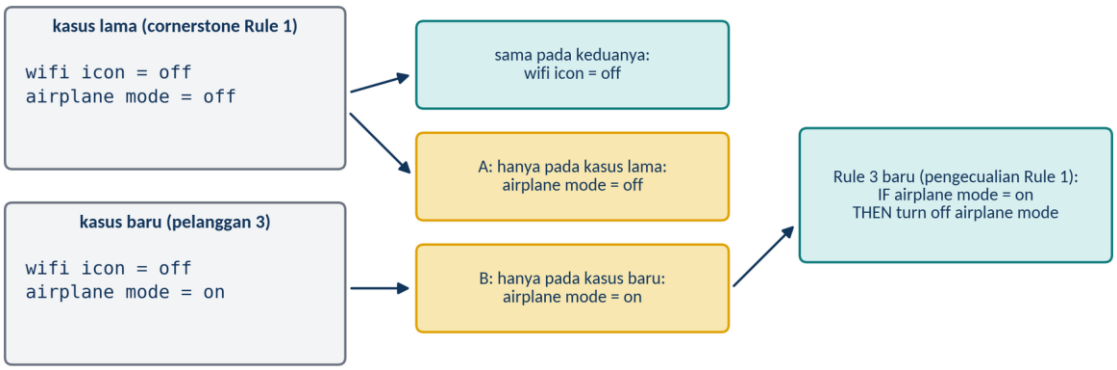
Sekarang lihatlah bagaimana Mas Anto sebenarnya berpikir. Ia tidak pernah berkata "jika mode pesawat menyala, matikan" secara abstrak. Ia berkata: *ketika saya menyuruh pelanggan menyalakan Wi-Fi dan itu tidak menolong, bedanya adalah mode pesawat sedang menyala.* Aturan barunya adalah **pengecualian** terhadap aturan lama, dan hanya bermakna dalam konteks aturan lama itu: di antara pelanggan yang simbol Wi-Fi-nya mati. Pelanggan 4, yang Wi-Fi-nya menyala, tidak pernah sampai ke sana.

Pengamatan ini adalah jantung **Ripple Down Rules** (RDR), yang diusulkan pada akhir 1980-an oleh Paul Compton dan rekan-rekannya untuk sebuah sistem pakar patologi [5], [2], lalu dipakai untuk membangun PEIRS, yang menafsirkan laporan patologi kimia di St Vincent's Hospital, Sydney, dengan pengetahuan yang seluruhnya ditambahkan oleh para ahli patologi [6]. Menurut mereka, pengetahuan seorang pakar selalu berupa pembenaran yang diberikan dalam suatu konteks: konklusi *ini*, bukan *itu*, untuk kasus *ini*. Maka sistem sebaiknya menyimpan pengetahuan persis dalam bentuk itu. Ketika sistem memberikan jawaban yang salah untuk sebuah kasus, sang pakar menambahkan aturan baru yang hanya berlaku di tempat jawaban salah itu berasal — sebagai pengecualian dari aturan yang memberikannya. Aturan tidak pernah disunting atau dihapus; aturan hanya diperhalus, dan penghalusannya "beriak turun" (*ripple down*) di bawahnya [4], [7].

> **DEFINISI.** Dalam *Ripple Down Rules*, setiap aturan dapat memiliki *pengecualian*: aturan-aturan yang hanya dicoba bila aturan itu benar, dan yang konklusinya menggantikan konklusi aturan itu bila mereka juga benar. Aturan baru selalu ditambahkan sebagai pengecualian di tempat konklusi yang salah dicapai, sehingga ia hanya dapat mengubah jawaban bagi kasus-kasus yang sampai ke tempat itu.

3.2.3 Kasus cornerstone dan daftar perbedaan

Bagaimana pakar memilih kondisi pengecualian itu? RDR menyimpan, bersama setiap aturan, kasus yang menyebabkan aturan itu dibuat: **kasus cornerstone**-nya (kasus batu penjuru) [2], [4]. Ketika sebuah kasus baru dijawab salah oleh suatu aturan, sistem membandingkan kasus baru itu dengan kasus cornerstone aturan tersebut dan menampilkan **perbedaan**-perbedaannya. Pakar cukup memilih, di antara perbedaan itu, ciri-ciri yang menjelaskan mengapa kasus baru membutuhkan jawaban lain. **Gambar 3.1** memperlihatkannya untuk pelanggan 3: kasus cornerstone aturan "nyalakan Wi-Fi" memiliki mode pesawat *off*; kasus baru memilikinya *on*. Perbedaan itulah alasannya, dan ia menjadi kondisi pengecualiannya.



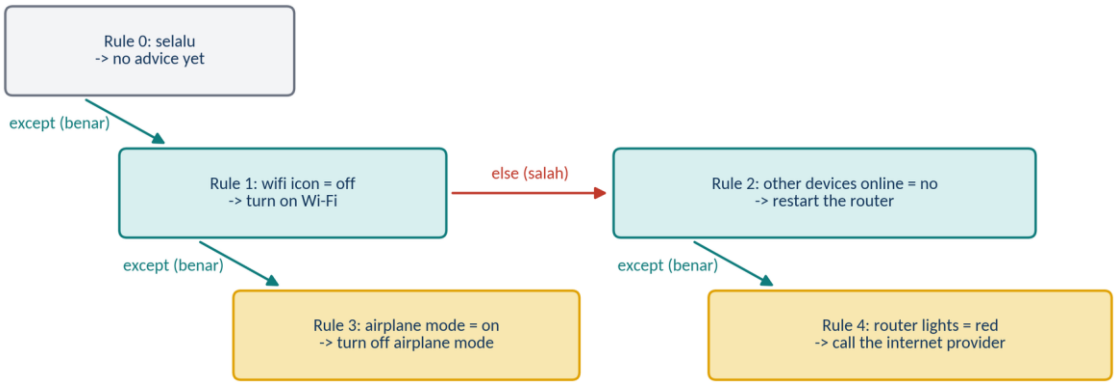
Gambar 3.1 Daftar perbedaan antara kasus cornerstone sebuah aturan dan sebuah kasus baru. Bagian A memuat apa yang hanya dimiliki kasus lama; bagian B apa yang hanya dimiliki kasus baru. Aturan baru sang pakar dibangun dari perbedaan-perbedaan ini.

Revisi VCIRS tahun 2006 menamai kedua sisi perbandingan ini dengan tepat [8], [9]. Bagian **A** memuat kondisi yang hanya ada pada kasus lama (cornerstone); bagian **B** memuat kondisi yang hanya ada pada kasus baru. Pengecualian baru dapat dibuat dari negasi bagian A (*mode pesawat tidak off*), dari bagian B (*mode pesawat on*), atau dari keduanya. Mesin buatan tangan kita dalam bab ini memakai bagian B, pilihan yang paling sederhana dan paling lazim; mesin vcirs-core menyediakan semuanya.

Mengapa kasus cornerstone menjamin bahwa tidak ada yang rusak? Karena kondisi yang baru, menurut cara pembuatannya, *salah* untuk kasus lama: kasus lama tidak memiliki mode pesawat yang menyala. Jadi aturan yang dahulu menjawab kasus lama dengan benar tetap menjawabnya dengan benar. Setiap aturan dalam basis pengetahuan RDR membawa kasus yang membenarkannya, dan setiap aturan baru dipilih agar tidak mengganggu kasus itu. Itulah yang memungkinkan seorang pakar yang tidak pernah melihat basis aturannya menambahkan pengetahuan dengan aman [4].

3.2.4 Bentuk pengetahuan: except dan else

Dalam bentuk aslinya, RDR klasifikasi tunggal, basis pengetahuan adalah sebuah **pohon biner** [2]. Setiap aturan memiliki dua cabang. Cabang **except** dicoba ketika aturan itu benar: ia memuat pengecualian aturan tersebut. Cabang **else** dicoba ketika aturan itu salah: ia memuat aturan berikutnya yang harus dicoba. **Gambar 3.2** memperlihatkan pohon yang dibangun Mas Anto di Subbab 3.4 bersama lima pelanggannya.



Gambar 3.2 Pohon Ripple Down Rules klasifikasi tunggal. Aturan 0 selalu benar; cabang except memuat pengecualian, dicoba ketika sebuah aturan benar; cabang else memuat aturan berikutnya, dicoba ketika sebuah aturan salah. Jawabannya adalah konklusi aturan terakhir yang benar.

Untuk menjawab sebuah kasus, mulailah dari Aturan 0, yang selalu benar dan memberikan jawaban bawaan *belum ada saran*. Turunlah melalui cabang except-nya. Pada setiap aturan, bila aturan itu benar, ingat konklusinya dan ikuti cabang except-nya; bila salah, ikuti cabang else-nya. Bila tidak ada cabang yang dapat diikuti, berhentilah. Jawabannya adalah konklusi **aturan terakhir yang benar**.

Cobalah pada pelanggan 4 di **Kode 3.1**: Wi-Fi on, mode pesawat on, perangkat lain tidak online. Aturan 0 benar (jawaban sementara: *belum ada saran*); Aturan 1, *wifi icon = off*, salah, jadi kita mengambil cabang else-nya; Aturan 2, *other devices online = no*, benar (jawaban: *restart router*); kita mengambil cabang except-nya ke Aturan 4, *router lights = red*, yang salah dan tidak memiliki cabang else, jadi kita berhenti. Jawabannya *restart router* — benar, karena aturan pesawat, Aturan 3, berada di bawah Aturan 1, dan pelanggan 4 tidak pernah ke sana.

Lalu ke mana sebuah aturan *baru* ditaruh? Tepat di tempat pencarian berhenti. Bila aturan terakhir yang dicoba benar, aturan baru menjadi cabang except-nya; bila salah, menjadi cabang else-nya. Dengan cara mana pun, hanya kasus-kasus yang mengikuti jalur yang sama dengan kasus yang sedang dikoreksi yang akan pernah mencapainya.

3.2.5 Keunggulan dan keterbatasan RDR

Ripple Down Rules memiliki watak tertentu, dan ada gunanya mengenalnya sebelum memercayai pemecah masalah yang dibangun di atasnya [3], [4]. Jawabannya **presisi**: karena setiap aturan dibuat untuk memperbaiki kasus nyata, dan dibuat agar tidak mengganggu kasus-kasus sebelumnya, basis pengetahuan cenderung menjawab dengan benar kasus-kasus yang pernah dilihatnya dan kasus-kasus yang mirip dengannya. Harganya adalah **cakupan**: basis pengetahuan RDR yang masih muda bersikap hati-hati. Ia menjawab apa yang sudah diajarkan kepadanya dan, untuk situasi yang benar-benar baru, berkata "belum ada saran" alih-alih menebak. Dibandingkan dengan metode pembelajaran mesin yang dilatih dengan ribuan contoh, sistem RDR biasanya unggul dalam presisi dan kalah dalam cakupan, terutama pada awalnya [3]. Untuk sebuah konter ponsel, watak inilah yang tepat. Jawaban yang salah membuang waktu pelanggan dan nama baik Mas Anto; "saya belum tahu — biar saya periksa" tidak merugikan apa pun, dan pelanggan berikutnya dengan masalah yang sama akan mendapat jawabannya.

Kedua, aturan cenderung **terlalu umum** pada awalnya: aturan pertama Mas Anto, *jika simbol Wi-Fi mati, nyalakan Wi-Fi*, sering benar dan kadang-kadang salah. Itu bukan cacat; begitulah pengetahuan dibangun. Setiap pengecualian mempersempit sebuah aturan tepat di tempat aturan itu terlalu umum, dan hanya di sana.

Ketiga, RDR klasifikasi tunggal memberikan **satu jawaban** untuk setiap kasus. Seorang pelanggan dapat memiliki dua masalah yang tidak berhubungan — kuota habis *dan* sinyal lemah — dan satu jawaban hanya dapat menyebut salah satunya. Varian yang disebut Multiple Classification RDR mengevaluasi setiap cabang yang berlaku dan memberikan semua konklusi yang ditemukannya [10]; mesin vcirs-core dapat melakukan hal yang sama bila diminta (`multiple = true`), dan Bab 5 memakainya untuk penasihat kesehatan, tempat dua temuan dapat sama-sama penting.

Terakhir, RDR sendiri **bernalar secara sederhana**. Ia menelusuri pohon; ia tidak dapat berangkat dari sebuah tujuan, seperti "apakah masalahnya di router?", lalu hanya menanyakan pertanyaan yang diperlukan untuk membuktikannya. Itulah *backward chaining*, kekuatan sistem berbasis aturan klasik, dan itulah persisnya yang dikembalikan oleh struktur VCIRS yang berpusat pada variabel [3], [11]. Bab 8 memakainya.

3.2.6 Ripple Down Rules dalam pekerjaan sungguhan

Semua ini bukan sekadar gagasan yang rapi di atas kertas. Setelah PEIRS, Ripple Down Rules dipakai dalam sistem komersial di laboratorium patologi, tempat para ahli patologi sendiri, tanpa pemrogram, menambahkan aturan satu kasus demi satu kasus sambil menjalankan pekerjaan sehari-hari mereka, dan basis pengetahuan yang tumbuh dengan cara ini mencapai ribuan aturan [4]. Yang mencolok dari laporan-laporan itu adalah biayanya: menambahkan satu aturan memerlukan beberapa menit, dan waktu itu hampir tidak bertambah ketika basis pengetahuan membesar, karena sang pakar tidak pernah perlu melihat seluruh basis aturan — cukup kasus yang di depannya, kasus cornerstone di sampingnya, dan daftar perbedaannya [4], [7].

Pola yang sama muncul di bidang-bidang lain yang dikumpulkan dalam tinjauan dua dasawarsa riset RDR: layanan *help desk*, klasifikasi surel dan dokumen, serta konfigurasi [7]. Kesamaannya selalu sama dengan kesamaan pada konter Mas Anto. Ada seseorang yang tahu jawabannya ketika melihat sebuah kasus, tetapi tidak dapat menuliskan seluruh pengetahuannya di muka; ada aliran kasus yang terus datang; dan jawaban yang salah dapat dikenali, sehingga dapat dikoreksi. Di mana pun ketiga hal itu ada, pengetahuan dapat tumbuh dari pekerjaan itu sendiri.

Ada juga pelajaran tentang apa yang *tidak* diperlukan. Sistem-sistem itu tidak menunggu sampai basis pengetahuannya “selesai” sebelum dipakai. Mereka mulai dengan beberapa aturan, menjawab apa yang dapat mereka jawab, dan menyerahkan sisanya kepada sang pakar, yang kemudian mengajarkannya. Dokter Wi-Fi kita akan bekerja persis dengan cara itu.

3.3 Ripple Down Rules dari Nol

Sekarang kita membangun mesin ini sendiri. **Kode 3.3** adalah seluruhnya dalam Java.

Kode 3.3 `Rdr.java`: mesin Ripple Down Rules klasifikasi tunggal yang lengkap.

```
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

/**
 * Ripple Down Rules klasifikasi tunggal, dibangun dari nol. Setiap aturan
 * punya cabang "except" (dicoba bila aturan benar) dan cabang "else"
 * (dicoba bila salah); jawabannya adalah konklusi aturan terakhir yang
 * benar. Aturan baru digantung tepat di tempat pencarian berhenti.
 */
public class Rdr {
    public static class Rule {
        public final int number;
        public final Map<String, String> conditions;
        public final String conclusion;
        public final Map<String, String> cornerstone;   // kasus yang membuatnya
        public Rule ifTrue;                             // cabang "except"
        public Rule ifFalse;                            // cabang "else"

        Rule(int number, Map<String, String> conditions, String conclusion,
             Map<String, String> cornerstone) {
            this.number = number;
            this.conditions = conditions;
            this.conclusion = conclusion;
            this.cornerstone = cornerstone;
        }

        boolean matches(Map<String, String> facts) {
            for (Map.Entry<String, String> c : conditions.entrySet()) {
                if (!c.getValue().equals(facts.get(c.getKey()))) {
                    return false;
                }
            }
            return true;
        }
```

Kode 3.3 (Lanjutan) *Rdr.java: mesin Ripple Down Rules klasifikasi tunggal yang lengkap.*

```
        public String text() {
            List<String> parts = new ArrayList<>();
            for (Map.Entry<String, String> c : conditions.entrySet()) {
                parts.add(c.getKey() + " = " + c.getValue());
            }
            String when = parts.isEmpty() ? "(always)"
                    : String.join(" AND ", parts);
            return "Rule " + number + ": IF " + when + " THEN " + conclusion;
        }
    }

    /** Jawaban, aturan yang memberikannya, dan tempat pencarian berhenti. */
    public static class Answer {
        public String conclusion;
        public Rule lastTrue;
        public Rule lastTried;
        public boolean lastResult;
        public final List<String> trace = new ArrayList<>();
    }

    public final Rule root = new Rule(0, new LinkedHashMap<>(), "no advice yet",
            new LinkedHashMap<>());
    private int count = 0;

    public Answer classify(Map<String, String> facts) {
        Answer a = new Answer();
        a.lastTrue = root;
        a.lastTried = root;
        a.lastResult = true;
        Rule rule = root.ifTrue;
        while (rule != null) {
            boolean ok = rule.matches(facts);
            a.trace.add("Rule " + rule.number + (ok ? " is true" : " is false"));
            a.lastTried = rule;
            a.lastResult = ok;
            if (ok) {
                a.lastTrue = rule;
                rule = rule.ifTrue;
            } else {
                rule = rule.ifFalse;
            }
        }
        a.conclusion = a.lastTrue.conclusion;
        return a;
    }
```

Kode 3.3 (Lanjutan) *Rdr.java: mesin Ripple Down Rules klasifikasi tunggal yang lengkap.*

```
    /** Fakta kasus baru yang tidak dimiliki kasus cornerstone-nya. */
    public static Map<String, String> differences(Map<String, String> facts,
                                                  Map<String, String> cornerstone) {
        Map<String, String> out = new LinkedHashMap<>();
        for (Map.Entry<String, String> f : facts.entrySet()) {
            if (!f.getValue().equals(cornerstone.get(f.getKey()))) {
                out.put(f.getKey(), f.getValue());
            }
        }
        return out;
    }

    /** Gantungkan aturan baru di tempat pencarian kasus ini berhenti. */
    public Rule learn(Map<String, String> facts, String conclusion,
                      Map<String, String> conditions) {
        Answer a = classify(facts);
        Rule rule = new Rule(++count, conditions, conclusion, facts);
        if (a.lastResult) {
            a.lastTried.ifTrue = rule;
        } else {
            a.lastTried.ifFalse = rule;
        }
        return rule;
    }

    /** "wifi icon = off, airplane mode = on" */
    public static String show(Map<String, String> facts) {
        List<String> parts = new ArrayList<>();
        for (Map.Entry<String, String> f : facts.entrySet()) {
            parts.add(f.getKey() + " = " + f.getValue());
        }
        return parts.isEmpty() ? "(nothing)" : String.join(", ", parts);
    }

    public List<String> tree() {
        List<String> lines = new ArrayList<>();
        draw(root, "", "", lines);
        return lines;
    }
```

Kode 3.3 (Lanjutan) *Rdr.java: mesin Ripple Down Rules klasifikasi tunggal yang lengkap.*

```
    private void draw(Rule rule, String indent, String label, List<String> out) {
        out.add(indent + label + rule.text());
        if (rule.ifTrue != null) {
            draw(rule.ifTrue, indent + "  ", "except: ", out);
        }
        if (rule.ifFalse != null) {
            draw(rule.ifFalse, indent, "else:   ", out);
        }
    }
}
```

Bacalah dalam empat bagian.

Aturan. Sebuah `Rule` memiliki nomor, kondisi-kondisinya (map dari variabel ke nilai, yang semuanya harus cocok), konklusinya, kasus cornerstone-nya, serta dua cabang `ifTrue` (except) dan `ifFalse` (else), yang berisi aturan lain atau `null`. `matches` memeriksa kondisi terhadap fakta sebuah kasus; `text` menuliskan aturan dengan cara buku ini menuliskan aturan.

Jawaban. `Answer` mencatat konklusi, `lastTrue` (aturan yang memberikannya), `lastTried` dan `lastResult` (aturan tempat pencarian berhenti, dan apakah aturan itu benar di sana), serta `trace`

berisi setiap aturan yang dicoba, yang merupakan penjelasan mesin tentang *bagaimana* ia mencapai jawabannya.

Mengklasifikasi. `classify` adalah penelusuran Subbab 3.2.4 dalam selusin baris. Ia mulai di bawah `root`, yang mewakili Aturan 0, dan mengulang: uji aturan, catat dalam jejak, ingat posisi kita; bila aturan benar, ingat sebagai `lastTrue` lalu pergi ke `ifTrue`, bila tidak, pergi ke `ifFalse`; berhenti pada `null`. Jawabannya adalah `lastTrue.conclusion`.

Belajar. `differences` mendaftar fakta-fakta kasus baru yang nilainya berbeda dari (atau tidak ada di) sebuah kasus cornerstone: bagian B dari daftar perbedaan. `learn` mengklasifikasi kasus itu sekali lagi untuk menemukan tempat pencarian berhenti, membuat aturan baru dengan kasus tersebut sebagai cornerstone-nya, lalu menggantungkannya pada cabang `ifTrue` aturan terakhir yang dicoba bila aturan itu benar, atau pada cabang `ifFalse`-nya bila salah. Hanya itu yang dibutuhkan RDR. `tree` mencetak basis pengetahuan, menjorokkan pengecualian di bawah aturannya dan menuliskan cabang else pada kedalaman yang sama.

Kode 3.4 adalah mesin yang sama dalam Python.

Kode 3.4 `rdr.py`: mesin yang sama dalam Python.

```
"""Ripple Down Rules klasifikasi tunggal, dibangun dari nol. Setiap aturan
punya cabang "except" (dicoba bila aturan benar) dan cabang "else" (dicoba
bila salah); jawabannya adalah konklusi aturan terakhir yang benar. Aturan
baru digantung tepat di tempat pencarian berhenti."""


class Rule:
    def __init__(self, number, conditions, conclusion, cornerstone):
        self.number = number
        self.conditions = conditions
        self.conclusion = conclusion
        self.cornerstone = cornerstone   # kasus yang membuatnya
        self.if_true = None              # cabang "except"
        self.if_false = None             # cabang "else"

    def matches(self, facts):
        return all(facts.get(var) == value
                   for var, value in self.conditions.items())

    def text(self):
        parts = [var + " = " + value for var, value in self.conditions.items()]
        when = " AND ".join(parts) if parts else "(always)"
        return "Rule " + str(self.number) + ": IF " + when + " THEN " \
            + self.conclusion


class Answer:
    """Jawaban, aturan yang memberikannya, dan tempat pencarian berhenti."""

    def __init__(self, root):
        self.conclusion = None
        self.last_true = root
        self.last_tried = root
        self.last_result = True
        self.trace = []


def show(facts):
    """'wifi icon = off, airplane mode = on'"""
    parts = [var + " = " + value for var, value in facts.items()]
    return ", ".join(parts) if parts else "(nothing)"
```

Kode 3.4 (Lanjutan) *rdr.py: mesin yang sama dalam Python.*

```
def differences(facts, cornerstone):
    """Fakta kasus baru yang tidak dimiliki kasus cornerstone-nya."""
    return {var: value for var, value in facts.items()
            if cornerstone.get(var) != value}


class Rdr:
    def __init__(self):
        self.root = Rule(0, {}, "no advice yet", {})
        self.count = 0

    def classify(self, facts):
        a = Answer(self.root)
        rule = self.root.if_true
        while rule is not None:
            ok = rule.matches(facts)
            a.trace.append("Rule " + str(rule.number)
                           + (" is true" if ok else " is false"))
            a.last_tried = rule
            a.last_result = ok
            if ok:
                a.last_true = rule
                rule = rule.if_true
            else:
                rule = rule.if_false
        a.conclusion = a.last_true.conclusion
        return a

    def learn(self, facts, conclusion, conditions):
        """Gantungkan aturan baru di tempat pencarian kasus ini berhenti."""
        a = self.classify(facts)
        self.count += 1
        rule = Rule(self.count, conditions, conclusion, facts)
        if a.last_result:
            a.last_tried.if_true = rule
        else:
            a.last_tried.if_false = rule
        return rule

    def tree(self):
        lines = []
```

Kode 3.4 (Lanjutan) *rdr.py: mesin yang sama dalam Python.*

```
        def draw(rule, indent, label):
            lines.append(indent + label + rule.text())
            if rule.if_true is not None:
                draw(rule.if_true, indent + "  ", "except: ")
            if rule.if_false is not None:
                draw(rule.if_false, indent, "else:   ")

        draw(self.root, "", "")
        return lines
```

Versi Python sedikit lebih pendek: sebuah *dictionary comprehension* membangun perbedaan dalam satu baris, dan `all(...)` memeriksa setiap kondisi dalam satu ekspresi. Logikanya identik, baris demi baris, demikian pula keluaran program yang memakainya.

> **WAWASAN KUNCI:** Mesin tidak pernah melihat makna sebuah aturan. Ia hanya mengikuti cabang benar dan salah, lalu menggantungkan aturan baru di tempat pencarian berhenti. Seluruh pengetahuan pemecahan masalah berada di dalam aturan, dan semuanya berasal dari kasus.

3.4 Mas Anto Mengajar Lima Pelanggan

Kode 3.5 memutar ulang lima pelanggan pertama Mas Anto pada suatu pagi. Untuk setiap pelanggan, program menunjukkan pencarian, jawaban, dan — bila jawabannya salah — perbaikan yang benar, perbedaan dari kasus cornerstone aturan yang menjawab, serta aturan yang ditambahkan Mas Anto, yang dibangun dari perbedaan yang dipilihnya. Di akhir, program mencetak basis pengetahuan dan mencoba lagi setiap pelanggan sebelumnya.

Kode 3.5 `RdrDemo.java`: lima pelanggan mengajari pemecah masalah yang masih kosong.

```
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

/** Mas Anto mengajari pemecah masalah, satu pelanggan demi satu. */
public class RdrDemo {
    /** facts("wifi icon=off", "airplane mode=on") menjaga urutan yang diberikan. */
    static Map<String, String> facts(String... pairs) {
        Map<String, String> m = new LinkedHashMap<>();
        for (String p : pairs) {
            String[] kv = p.split("=");
            m.put(kv[0], kv[1]);
        }
        return m;
    }

    static void customer(Rdr kb, int n, Map<String, String> facts, String right,
                         String... chosen) {
        Rdr.Answer a = kb.classify(facts);
        System.out.println("Customer " + n + ": " + Rdr.show(facts));
        System.out.println("  search: " + (a.trace.isEmpty() ? "(no rules yet)"
                : String.join(", ", a.trace)));
        System.out.println("  answer: " + a.conclusion);
        if (a.conclusion.equals(right)) {
            System.out.println("  right!");
            return;
        }
        Map<String, String> diff = Rdr.differences(facts,
                a.lastTrue.cornerstone);
        System.out.println("  wrong; the right fix is: " + right);
        System.out.println("  differences from the case of Rule "
                + a.lastTrue.number + ": " + Rdr.show(diff));
        Rdr.Rule rule = kb.learn(facts, right, facts(chosen));
        System.out.println("  learned " + rule.text());
    }
```

Kode 3.5 (Lanjutan) *RdrDemo.java: lima pelanggan mengajari pemecah masalah yang masih kosong.*

```
    public static void main(String[] args) {
        Rdr kb = new Rdr();
        List<Map<String, String>> seen = List.of(
                facts("wifi icon=off", "airplane mode=off"),
                facts("wifi icon=on", "other devices online=no"),
                facts("wifi icon=off", "airplane mode=on"),
                facts("wifi icon=on", "airplane mode=on",
                        "other devices online=no"),
                facts("wifi icon=on", "other devices online=no",
                        "router lights=red"));
        List<String> right = List.of("turn on Wi-Fi", "restart the router",
                "turn off airplane mode", "restart the router",
                "call the internet provider");
        customer(kb, 1, seen.get(0), right.get(0), "wifi icon=off");
        customer(kb, 2, seen.get(1), right.get(1), "other devices online=no");
        customer(kb, 3, seen.get(2), right.get(2), "airplane mode=on");
        customer(kb, 4, seen.get(3), right.get(3));
        customer(kb, 5, seen.get(4), right.get(4), "router lights=red");

        System.out.println();
        System.out.println("The knowledge base now:");
        for (String line : kb.tree()) {
            System.out.println("  " + line);
        }
        System.out.println();
        System.out.println("Every earlier customer again:");
        for (int i = 0; i < seen.size(); i++) {
            String got = kb.classify(seen.get(i)).conclusion;
            System.out.printf("  customer %d: %-28s %s%n", i + 1, got,
                    got.equals(right.get(i)) ? "right" : "WRONG");
        }
    }
}
```

`facts` membangun map yang mempertahankan urutan fakta sebagaimana ditulis, sehingga perbedaan dicetak dalam urutan yang dapat diperkirakan. `customer` mengerjakan satu langkah pengajaran: klasifikasi, cetak, dan bila jawabannya salah, hitung perbedaan terhadap `a.lastTrue.cornerstone` lalu panggil `learn` dengan kondisi yang dipilih Mas Anto (diberikan sebagai argumen terakhir). **Kode 3.6** adalah versi Python-nya.

Kode 3.6 `rdr_demo.py`: kelima pelanggan yang sama dalam Python.

```
"""Mas Anto mengajari pemecah masalah, satu pelanggan demi satu."""

from rdr import Rdr, differences, show


def facts(*pairs):
    """facts("wifi icon=off", "airplane mode=on") menjaga urutan yang diberikan."""
    return dict(p.split("=") for p in pairs)


def customer(kb, n, case, right, *chosen):
    a = kb.classify(case)
    print("Customer " + str(n) + ": " + show(case))
    print("  search: " + (", ".join(a.trace) if a.trace else "(no rules yet)"))
    print("  answer: " + a.conclusion)
    if a.conclusion == right:
        print("  right!")
        return
    diff = differences(case, a.last_true.cornerstone)
    print("  wrong; the right fix is: " + right)
    print("  differences from the case of Rule " + str(a.last_true.number)
          + ": " + show(diff))
    rule = kb.learn(case, right, facts(*chosen))
    print("  learned " + rule.text())


kb = Rdr()
seen = [facts("wifi icon=off", "airplane mode=off"),
        facts("wifi icon=on", "other devices online=no"),
        facts("wifi icon=off", "airplane mode=on"),
        facts("wifi icon=on", "airplane mode=on", "other devices online=no"),
        facts("wifi icon=on", "other devices online=no", "router lights=red")]
right = ["turn on Wi-Fi", "restart the router", "turn off airplane mode",
         "restart the router", "call the internet provider"]
customer(kb, 1, seen[0], right[0], "wifi icon=off")
customer(kb, 2, seen[1], right[1], "other devices online=no")
customer(kb, 3, seen[2], right[2], "airplane mode=on")
customer(kb, 4, seen[3], right[3])
customer(kb, 5, seen[4], right[4], "router lights=red")
```

Kode 3.6 (Lanjutan) *rdr_demo.py: kelima pelanggan yang sama dalam Python.*

```
print()
print("The knowledge base now:")
for line in kb.tree():
    print("  " + line)
print()
print("Every earlier customer again:")
for i, case in enumerate(seen):
    got = kb.classify(case).conclusion
    print(f"  customer {i + 1}: {got:<28} "
          + ("right" if got == right[i] else "WRONG"))
```

Jalankan `java -cp out RdrDemo` (setelah `javac -d out Rdr.java RdrDemo.java`) atau `python rdr_demo.py`:

```
Customer 1: wifi icon = off, airplane mode = off
  search: (no rules yet)
  answer: no advice yet
  wrong; the right fix is: turn on Wi-Fi
  differences from the case of Rule 0: wifi icon = off, airplane mode = off
  learned Rule 1: IF wifi icon = off THEN turn on Wi-Fi
Customer 2: wifi icon = on, other devices online = no
  search: Rule 1 is false
  answer: no advice yet
  wrong; the right fix is: restart the router
  differences from the case of Rule 0: wifi icon = on, other devices online = no
  learned Rule 2: IF other devices online = no THEN restart the router
Customer 3: wifi icon = off, airplane mode = on
  search: Rule 1 is true
  answer: turn on Wi-Fi
  wrong; the right fix is: turn off airplane mode
  differences from the case of Rule 1: airplane mode = on
  learned Rule 3: IF airplane mode = on THEN turn off airplane mode
Customer 4: wifi icon = on, airplane mode = on, other devices online = no
  search: Rule 1 is false, Rule 2 is true
  answer: restart the router
  right!
Customer 5: wifi icon = on, other devices online = no, router lights = red
  search: Rule 1 is false, Rule 2 is true
  answer: restart the router
  wrong; the right fix is: call the internet provider
  differences from the case of Rule 2: router lights = red
  learned Rule 4: IF router lights = red THEN call the internet provider

The knowledge base now:
  Rule 0: IF (always) THEN no advice yet
    except: Rule 1: IF wifi icon = off THEN turn on Wi-Fi
      except: Rule 3: IF airplane mode = on THEN turn off airplane mode
    else:   Rule 2: IF other devices online = no THEN restart the router
      except: Rule 4: IF router lights = red THEN call the internet provider

Every earlier customer again:
  customer 1: turn on Wi-Fi                right
  customer 2: restart the router           right
  customer 3: turn off airplane mode       right
  customer 4: restart the router           right
  customer 5: call the internet provider   right
```

Ikutilah pengetahuan itu seiring pertumbuhannya.

Pelanggan 1 datang ke basis pengetahuan yang kosong. Pencarian tidak mencoba aturan apa pun; jawabannya adalah *belum ada saran* dari Aturan 0. Perbedaan dari cornerstone Aturan 0 (yang kosong) adalah semua yang dikatakan pelanggan. Mas Anto hanya memilih `wifi icon = off` — mode pesawat yang mati bukanlah alasannya — dan Aturan 1 digantungkan pada cabang except Aturan 0.

Pelanggan 2 Wi-Fi-nya menyala. Aturan 1 salah; pencarian berhenti di sana, sehingga Aturan 2 yang baru, *other devices online = no → restart router*, digantungkan pada cabang **else** Aturan 1. Aturan itu akan dicoba untuk setiap pelanggan yang Wi-Fi-nya menyala.

Pelanggan 3 adalah pelanggan mode pesawat yang merusak daftar datar. Aturan 1 benar, sehingga jawabannya *nyalakan Wi-Fi*, yang salah. Perbedaan dari kasus cornerstone Aturan 1 (pelanggan 1) tinggal satu fakta, `airplane mode = on`, dan Aturan 3 digantungkan pada cabang **except** Aturan 1: sebuah pengecualian, dalam konteks tempat ia seharusnya berada.

Pelanggan 4 adalah pelanggan yang dijawab salah oleh daftar datar setelah "diperbaiki". Di sini tidak ada yang perlu diajarkan: Aturan 1 salah, Aturan 2 benar, dan jawabannya benar. Aturan pesawat tidak pernah dilihat, karena pelanggan 4 tidak pernah memasuki konteks Aturan 1.

Pelanggan 5 lampu router-nya merah. Aturan 2 menjawab *restart router*, yang tidak akan menolong: jalur penyedia layanan sedang putus. Satu-satunya perbedaan dari cornerstone Aturan 2 (pelanggan 2) adalah `router lights = red`, dan Aturan 4 menjadi pengecualian Aturan 2.

Basis pengetahuan yang dicetak di akhir adalah pohon **Gambar 3.2**. Dan baris-baris terakhir adalah bukti yang penting: setiap pelanggan sebelumnya, ketika dicoba lagi, tetap mendapat jawaban yang benar. Tidak ada aturan yang diubah urutannya, tidak ada aturan yang disunting, dan tidak satu pun jawaban lama berubah. Setiap aturan baru dipilih, dari perbedaan-perbedaan, agar salah bagi kasus cornerstone di sampingnya.

3.5 Dari Mesin Kita ke vcirs-core

Mesin buatan tangan kita sudah lengkap, tetapi sengaja dibuat kecil. Ia menyimpan pengetahuannya hanya di memori, tidak dapat bertanya, hanya menyediakan satu cara membangun pengecualian, dan penjelasannya berupa daftar aturan yang benar dan salah. Mesin bersama, vcirs-core, mempertahankan jantung Ripple-Down-Rules yang sama dan menambahkan apa yang dibutuhkan aplikasi sungguhan. **Tabel 3.1** memperlihatkan bagaimana gagasan-gagasan mesin kita muncul di dalamnya.

Tabel 3.1 Bagaimana gagasan mesin buatan tangan muncul dalam vcirs-core.

Mesin RDR kita (Subbab 3.3)	Vcirs-core
aturan dengan kondisi dan sebuah konklusi	sebuah **node**: rangkaian kondisi dengan satu atau lebih konklusi
cabang except sebuah aturan	**anak-anak** (*children*) sebuah node (pengecualiannya)
cabang else	urutan node tingkat atas dan urutan anak-anak
jalur dari Aturan 0 ke sebuah aturan	sebuah **rule** dalam Rule Structure: rangkaian node
kasus cornerstone	`node.cornerstone`, kasus yang menciptakan node itu
`differences` (hanya bagian B)	`differences(node, case)`: bagian **A** dan **B**
`learn(case, conclusion, conditions)`	`learn(case)` (otomatis) atau `refine(case, option, node, chosen)`
`classify` dan jejaknya	`infer(facts, asker)`: menanyakan fakta yang belum diketahui, menjelaskan How dan Why
pohon di memori	basis pengetahuan yang disimpan sebagai JSON (atau SQLite)

Tambahan terbesarnya ada tiga. Pertama, **pertanyaan**. Mesin kita membutuhkan semua fakta di muka; vcirs-core menanyakan sebuah fakta hanya ketika sebuah node membutuhkannya, sehingga pelanggan menjawab tiga atau empat pertanyaan alih-alih mengisi formulir lengkap. Kedua, **enam opsi** pembangunan pengetahuan dari revisi VCIRS [8], [9], yang dirangkum dalam **Tabel 3.2**. `learn(case)` yang cukup dipanggil sekali memilih di antaranya seperti sebuah sistem RDR: menerima kasus bila jawabannya sudah benar (opsi 0), menambahkan pengecualian dari kedua bagian daftar perbedaan bila sebuah node memberi jawaban yang salah (opsi 3), dan menambahkan node tingkat atas yang baru bila tidak ada node yang menjawab sama sekali (opsi 5). Ketiga, pengetahuannya **berpusat pada variabel** [3], [11]: setiap kondisi disimpan bersama variabelnya, setiap pemakaian sebuah variabel dihitung, dan pengetahuan yang sama kelak dapat

dibaca sebagai aturan JIKA-MAKA biasa untuk *forward chaining* dan *backward chaining*. Bab ini hanya memakai dua yang pertama; bab-bab berikutnya memakai sisanya.

Tabel 3.2 Enam opsi untuk menambahkan pengetahuan dalam VCIRS, mengikuti revisi 2006 [8], [9].

Opsi	Node baru dibuat dari	Tempatnya
0	tidak ada: jawaban yang ada diterima	— (pemeriksaannya dicatat)
1	negasi bagian A daftar perbedaan	sebagai pengecualian node lama
2	bagian B daftar perbedaan	sebagai pengecualian node lama
3	opsi 1 dan 2 bersama-sama	sebagai pengecualian node lama
4	semua fakta kasus baru	di samping node lama (induk yang sama)
5	semua fakta kasus baru	di tingkat atas

Satu tambahan lagi penting untuk keamanan. Sebelum menambahkan pengecualian, vcirs-core dapat mendaftar **cornerstone yang terdampak**: kasus-kasus cornerstone node lain yang juga akan tertangkap oleh node baru, sehingga jawabannya akan berubah [4]. Untuk pengecualian yang dipilih dengan baik, daftar ini kosong. Bila tidak kosong, pakar melihat dengan tepat kasus-kasus lama mana yang harus diperhatikan, dan dapat menambahkan fakta pembeda lainnya — inilah perulangan RDR "ulangi sampai pakar menerima aturan dan akibat-akibatnya", yang dikerjakan oleh mesin.

3.6 Dokter Wi-Fi

3.6.1 Pengetahuan minggu pertama

Aplikasi ini berangkat dari apa yang diajarkan Mas Anto pada minggu pertamanya, enam kasus yang disimpan dalam sebuah berkas kasus, `wifi-seed-cases.json`:

```
{
  "name": "Phone and Wi-Fi doctor",
  "description": "What Mas Anto taught the troubleshooter in his first week at the phone
counter.",
  "cases": [
    {"facts": ["connection = wifi", "wifi icon = off"], "conclusions": ["fix = turn on
Wi-Fi"], "option": -1, "author": "Anto"},
    {"facts": ["connection = wifi", "wifi icon = off", "airplane mode = on"],
"conclusions": ["fix = turn off airplane mode"], "option": -1, "author": "Anto"},
    {"facts": ["connection = wifi", "wifi icon = on", "other devices online = no"],
"conclusions": ["fix = restart the router"], "option": -1, "author": "Anto"},
    {"facts": ["connection = wifi", "wifi icon = on", "other devices online = no",
"router lights = red"], "conclusions": ["fix = call the internet provider"], "option": -
1, "author": "Anto"},
    {"facts": ["connection = mobile data", "mobile data = off"], "conclusions": ["fix =
turn on mobile data"], "option": -1, "author": "Anto"},
    {"facts": ["connection = mobile data", "mobile data = on", "quota left = no"],
"conclusions": ["fix = buy a data package"], "option": -1, "author": "Anto"}
  ]
}
```

Setiap kasus mendaftar fakta-fakta dan perbaikan yang benar; `"option": -1` berarti "biarkan `learn` yang memutuskan", sehingga basis pengetahuan dibangun persis seolah-olah Mas Anto mengajarkan keenam kasus itu satu demi satu di konter. Saat pertama kali dijalankan, aplikasi membangun basis pengetahuannya dari berkas ini dan menyimpannya sebagai `wifi-kb.json`; sesudahnya ia selalu memuat dan memperbarui berkas tersimpan itu.

3.6.2 Programnya

Kode 3.7 adalah dokter Wi-Fi dalam Java.

Kode 3.7 `WifiDoctor.java`: dokter Wi-Fi, pemecah masalah yang belajar dari setiap perbaikan, di atas vcirs-core.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;

import vcirs.Case;
import vcirs.CaseFile;
import vcirs.Conclusion;
import vcirs.Condition;
import vcirs.DifferenceList;
import vcirs.Fact;
import vcirs.Facts;
import vcirs.InferenceResult;
import vcirs.KnowledgeBase;
import vcirs.LearnResult;
import vcirs.Node;
import vcirs.Question;
import vcirs.Report;
import vcirs.VcirsException;

/**
 * Dokter ponsel dan Wi-Fi: pemecah masalah yang belajar dari tiap perbaikan.
 * java WifiDoctor [--echo] [--tree] [knowledge-base.json]
 * Program menanyakan gejala, memberi perbaikan yang diketahui basis
 * pengetahuan, menanyakan apakah berhasil, dan bila tidak, mempelajari
 * perbaikan yang benar sebagai pengecualian Ripple Down Rules baru. --echo
 * mengulang jawaban yang dibaca, agar sesi dari berkas terbaca seperti
 * percakapan.
 */
public class WifiDoctor {
    static final String[][] QUESTIONS = {
        {"connection", "Are you using Wi-Fi or mobile data?",
            "wifi", "mobile data"},
        {"wifi icon", "Is the Wi-Fi symbol at the top of the screen on or off?",
            "on", "off"},
        {"airplane mode", "Is airplane mode (the little plane) on?", "on", "off"},
        {"other devices online",
            "Do other phones get internet on the same Wi-Fi?", "yes", "no"},
        {"router lights", "Are the router's lights normal, red or off?",
            "normal", "red", "off"},
        {"mobile data", "Is mobile data switched on?", "on", "off"},
        {"quota left", "Is there still internet quota on the SIM card?",
```

Kode 3.7 (Lanjutan) *WifiDoctor.java: dokter Wi-Fi, pemecah masalah yang belajar dari setiap perbaikan, di atas vcirs-core.*

```
            "yes", "no"},
    };

    static BufferedReader in = new BufferedReader(
            new InputStreamReader(System.in, StandardCharsets.UTF_8));
    static boolean echo = false;
    static KnowledgeBase kb;

    /** Membaca jawaban pengguna; bila echo aktif, mencetaknya juga. */
    static String read(String prompt) {
        System.out.print(prompt);
        String line;
        try {
            line = in.readLine();
        } catch (IOException e) {
            line = null;
        }
        String text = line == null ? "" : line.strip();
        if (echo) {
            System.out.println(text);
        }
        return text;
    }

    static Fact ask(Question q) {
        String info = kb.variables.containsKey(q.variable)
                ? kb.variables.get(q.variable).question : "";
        String text = read((info.isEmpty() ? q.variable : info) + " ("
                + String.join("/", q.choices) + ") > ");
        return text.isEmpty() ? null : new Fact(q.variable, text);  // Enter = ?
    }

    /** Muat basis pengetahuan, atau bangun dari kasus-kasus awal. */
    static KnowledgeBase openKb(String path) {
        if (Files.exists(Path.of(path))) {
            return KnowledgeBase.load(path);
        }
        Path parent = Path.of(path).toAbsolutePath().getParent();
        CaseFile seed = CaseFile.read(parent.resolve("wifi-seed-cases.json")
                .toString());
        KnowledgeBase k = new KnowledgeBase(seed.name, seed.description);
        for (String[] q : QUESTIONS) {
            List<String> choices = new ArrayList<>();
            for (int i = 2; i < q.length; i++) {
                choices.add(q[i]);
            }
```

Kode 3.7 (Lanjutan) *WifiDoctor.java: dokter Wi-Fi, pemecah masalah yang belajar dari setiap perbaikan, di atas vcirs-core.*

```
            k.declare(q[0], q[1], choices);
        }
        seed.build(k);
        k.save(path);
        return k;
    }

    public static void main(String[] args) {
        String path = "../data/wifi-kb.json";
        boolean tree = false;
        for (String a : args) {
            if (a.equals("--echo")) {
                echo = true;
            } else if (a.equals("--tree")) {
                tree = true;
            } else {
                path = a;
            }
        }
        kb = openKb(path);
        if (tree) {
            for (String line : Report.tree(kb)) {
                System.out.println(line);
            }
            return;
        }
        System.out.println("Phone and Wi-Fi doctor: " + kb.nodes.size()
                + " fixes learned so far.");
        System.out.println("Answer each question; press Enter if you do not "
                + "know.");
        InferenceResult result = kb.infer(Facts.none(), WifiDoctor::ask);
        Node fired = result.paths.isEmpty() ? null : result.paths.get(0).last();
        if (fired != null && !result.conclusions().isEmpty()) {
            System.out.println("Advice: " + result.conclusions().get(0).value);
            System.out.println("Why:");
            for (String line : result.how()) {
                System.out.println("  " + line);
            }
        } else {
            System.out.println("I have no advice for this problem yet.");
        }
```

Kode 3.7 (Lanjutan) *WifiDoctor.java: dokter Wi-Fi, pemecah masalah yang belajar dari setiap perbaikan, di atas vcirs-core.*

```
        List<Condition> facts = new ArrayList<>();
        for (Fact f : result.memory.facts.values()) {
            if (!f.variable.equals("fix")) {
                facts.add(new Condition(f.variable, "=", f.value));
            }
        }
        boolean worked = fired != null
                && read("Did it fix the problem? (yes/no) > ").equals("yes");
        if (worked) {
            Case c = new Case(facts, result.conclusions(), 100);
            kb.learn(c, "counter");              // dicatat: kasus terverifikasi
            kb.save(path);
            System.out.println("Good. The knowledge base was right; nothing "
                    + "new to learn.");
            return;
        }
        String fix = read("What fixed it in the end? (Enter if nothing did) > ");
        if (fix.isEmpty()) {
            System.out.println("Then there is nothing to learn this time.");
            return;
        }
        while (true) {
            String extra = read("Another fact that was different this time? "
                    + "(variable = value, Enter when done) > ");
            if (extra.isEmpty()) {
                break;
            }
            try {
                facts.add(Condition.parse(extra));
            } catch (VcirsException e) {
                System.out.println("Please write it as: variable = value");
            }
        }
        Case c = new Case(facts, List.of(new Conclusion("fix", fix)), 100);
        if (fired != null) {
            DifferenceList d = kb.differences(fired.id, c);
            for (String line : d.lines()) {
                System.out.println(line);
            }
            List<Condition> conds = new ArrayList<>();
            for (Condition x : d.a) {
                conds.add(x.negated());
            }
            conds.addAll(d.b);
            List<String> hit = kb.affectedCornerstones(fired.id, conds,
                    c.conclusions);
```

Kode 3.7 (Lanjutan) *WifiDoctor.java: dokter Wi-Fi, pemecah masalah yang belajar dari setiap perbaikan, di atas vcirs-core.*

```
            System.out.println("Earlier cases whose answer would change: "
                    + (hit.isEmpty() ? "none" : String.join(", ", hit)));
        }
        LearnResult learned;
        try {
            learned = kb.learn(c, "counter");
        } catch (VcirsException e) {
            System.out.println("Cannot learn: " + e.getMessage());
            return;
        }
        System.out.println("Learned: " + learned);
        kb.save(path);
        System.out.println("Saved " + path + ": " + kb.summary());
    }
}
```

Telusurilah `main` langkah demi langkah, karena di sanalah seluruh riwayat seorang pelanggan.

1. **Buka pengetahuannya.** `openKb` memuat `wifi-kb.json`, atau, pada kali pertama, membangunnya: ia mendeklarasikan ketujuh variabel beserta pertanyaan yang diajukan untuk masing-masing dan jawaban-jawaban yang lazim (`QUESTIONS`), lalu memberikan keenam kasus awal kepada mesin dengan `seed.build(k)`, dan menyimpan hasilnya.
2. **Berkonsultasi.** `kb.infer(Facts.none(), WifiDoctor::ask)` mulai tanpa fakta sama sekali. Mesin menelusuri basis pengetahuan dari node-node tingkat atasnya, dan setiap kali sebuah node membutuhkan fakta yang belum dimilikinya, mesin memanggil `ask`, yang mencetak pertanyaan yang dideklarasikan beserta jawaban-jawaban lazimnya lalu membaca balasannya. Balasan kosong berarti “saya tidak tahu”, dan node yang membutuhkannya tidak terpenuhi. Ketika sebuah node terpenuhi, mesin mengikuti pengecualian-pengecualiannya, dan hanya menanyakan apa yang mereka butuhkan.
3. **Beri saran dan jelaskan.** Jawabannya adalah konklusi node terakhir yang terpenuhi; `result.how()` menjelaskannya, fakta demi fakta.
4. **Berhasil?** Bila pelanggan menjawab ya, kasus itu diberikan kepada `learn`, yang mendapati jawabannya sudah benar dan mencatatnya sebagai kasus terverifikasi (opsi 0) — verifikasi sambil jalan yang menjadi ciri RDR [4].
5. **Belajar.** Bila sarannya gagal (atau tidak ada saran), program menanyakan apa yang akhirnya memperbaiki masalah, lalu, satu per satu, fakta lain apa pun yang berbeda kali ini. Di sini Mas Anto dapat memasukkan variabel yang belum pernah dikenal basis pengetahuan. Program menampilkan daftar perbedaan terhadap kasus cornerstone node yang menjawab, serta cornerstone yang terdampak; kemudian `kb.learn(c, "counter")` menambahkan pengetahuan itu dan berkasnya disimpan.

Mengapa dokter menanyakan koneksi lebih dahulu? Karena ia menelusuri node tingkat atas dalam urutan pengajarannya, dan memeriksa kondisi setiap node dalam urutannya; kasus awal yang pertama kebetulan dimulai dengan `connection`. Node yang kondisi pertamanya salah langsung ditinggalkan, tanpa menanyakan sisanya, sehingga pelanggan yang memakai data seluler hanya ditanya tentang simbol Wi-Fi bila ada node tentang data seluler yang membutuhkannya. Dan sebuah fakta, sekali diketahui, tidak pernah ditanyakan lagi: memori kerja konsultasi menyimpan setiap jawaban, dan setiap node yang membutuhkannya membacanya dari sana. Pada sesi 2 di bawah Anda akan melihat mesin melewati satu node utuh setelah satu jawaban lalu melanjutkan.

Opsi `--echo` mengulang setiap jawaban yang dibaca dari masukan. Opsi ini ada agar sebuah sesi dapat diketik dalam berkas lalu diputar ulang, dan dengan cara itulah transkrip-transkrip dalam bab ini dihasilkan; di konter Anda menjalankan program tanpa opsi itu dan mengetik jawabannya. **Kode 3.8** adalah aplikasi yang sama dalam Python.

Kode 3.8 `wifi_doctor.py`: dokter Wi-Fi dalam Python.

```
"""Dokter ponsel dan Wi-Fi: pemecah masalah yang belajar dari tiap perbaikan.

    python wifi_doctor.py [--echo] [--tree] [knowledge-base.json]

Program menanyakan gejala, memberi perbaikan yang diketahui basis
pengetahuan, menanyakan apakah berhasil, dan bila tidak, mempelajari
perbaikan yang benar sebagai pengecualian Ripple Down Rules baru. --echo
mengulang jawaban yang dibaca, agar sesi dari berkas terbaca seperti
percakapan. --tree menampilkan aturan-aturannya.
"""

import os
import pathlib
import sys

# Beri tahu Python letak mesin vcirs-core (naik dua folder, lalu masuk).
sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import (BOTH, Case, Condition, Conclusion, KnowledgeBase,
                   VcirsError, casefile, report)

QUESTIONS = [
    ("connection", "Are you using Wi-Fi or mobile data?",
     ["wifi", "mobile data"]),
    ("wifi icon", "Is the Wi-Fi symbol at the top of the screen on or off?",
     ["on", "off"]),
    ("airplane mode", "Is airplane mode (the little plane) on?", ["on", "off"]),
    ("other devices online",
     "Do other phones get internet on the same Wi-Fi?", ["yes", "no"]),
    ("router lights", "Are the router's lights normal, red or off?",
     ["normal", "red", "off"]),
    ("mobile data", "Is mobile data switched on?", ["on", "off"]),
    ("quota left", "Is there still internet quota on the SIM card?",
     ["yes", "no"]),
]


class Console:
    """Membaca jawaban pengguna; bila echo aktif, mencetaknya juga."""

    def __init__(self, echo):
        self.echo = echo
```

Kode 3.8 (Lanjutan) *wifi_doctor.py: dokter Wi-Fi dalam Python.*

```
    def read(self, prompt):
        print(prompt, end="")
        line = sys.stdin.readline()
        text = line.strip()
        if self.echo:
            print(text)
        return text

    def ask(self, question):
        var = question.variable
        info = KB.variables[var].question if var in KB.variables else ""
        choices = "/".join(question.choices)
        text = self.read((info or var) + " (" + choices + ") > ")
        return text or None                    # Enter = "tidak tahu"


def open_kb(path):
    """Muat basis pengetahuan, atau bangun dari kasus-kasus awal."""
    if os.path.exists(path):
        return KnowledgeBase.load(path)
    seed = os.path.join(os.path.dirname(path), "wifi-seed-cases.json")
    name, description, steps = casefile.read_cases(seed)
    kb = KnowledgeBase(name, description)
    for var, question, choices in QUESTIONS:
        kb.declare(var, question, choices)
    casefile.build(kb, steps)
    kb.save(path)
    return kb


def main(args):
    global KB
    echo = "--echo" in args
    files = [a for a in args if not a.startswith("--")]
    path = files[0] if files else "../data/wifi-kb.json"
    KB = kb = open_kb(path)
    if "--tree" in args:
        for line in report.tree(kb):
            print(line)
        return
    con = Console(echo)
    print("Phone and Wi-Fi doctor: " + str(len(kb.nodes))
          + " fixes learned so far.")
    print("Answer each question; press Enter if you do not know.")
    result = kb.infer({}, con.ask)
    fired = result.paths[0].last() if result.paths else None
```

Kode 3.8 (Lanjutan) *wifi_doctor.py: dokter Wi-Fi dalam Python.*

```
    if fired is not None and result.conclusions():
        print("Advice: " + result.conclusions()[0].value)
        print("Why:")
        for line in result.how():
            print("  " + line)
    else:
        print("I have no advice for this problem yet.")

    facts = [Condition(f.variable, "=", f.value)
             for f in result.memory.facts.values() if f.variable != "fix"]
    worked = fired is not None and con.read(
        "Did it fix the problem? (yes/no) > ") == "yes"
    if worked:
        case = Case(facts, list(result.conclusions()))
        kb.learn(case, "counter")               # dicatat: kasus terverifikasi
        kb.save(path)
        print("Good. The knowledge base was right; nothing new to learn.")
        return
    fix = con.read("What fixed it in the end? (Enter if nothing did) > ")
    if not fix:
        print("Then there is nothing to learn this time.")
        return
    while True:
        extra = con.read("Another fact that was different this time? "
                         "(variable = value, Enter when done) > ")
        if not extra:
            break
        try:
            facts.append(Condition.parse(extra))
        except VcirsError:
            print("Please write it as: variable = value")
    case = Case(facts, [Conclusion("fix", fix)])
    if fired is not None:
        d = kb.differences(fired.id, case)
        for line in d.lines():
            print(line)
        conds = [c.negated() for c in d.a] + d.b
        hit = kb.affected_cornerstones(fired.id, conds, case.conclusions)
        print("Earlier cases whose answer would change: "
              + (", ".join(hit) if hit else "none"))
    try:
        learned = kb.learn(case, "counter")
    except VcirsError as e:
        print("Cannot learn: " + str(e))
        return
    print("Learned: " + str(learned))
```

Kode 3.8 (Lanjutan) *wifi_doctor.py: dokter Wi-Fi dalam Python.*

```
    kb.save(path)
    print("Saved " + path + ": " + kb.summary())


if __name__ == "__main__":
    main(sys.argv[1:])
```

Penanya (*asker*) dalam Python adalah metode `ask` dari objek kecil `Console`; fungsi apa pun yang menerima sebuah pertanyaan dan mengembalikan jawaban (atau `None`) dapat dipakai, dan begitulah mesin tetap tidak bergantung pada dari mana jawaban berasal — sebuah konsol hari ini, halaman web di Bab 10, jendela obrolan di Bab 13.

3.7 Empat Pelanggan di Konter

Folder `data` bab ini menyimpan empat percakapan, `session-1.txt` sampai `session-4.txt`, satu jawaban per baris. Mulai dari basis pengetahuan yang baru, jalankan secara berurutan dari folder `Chapter03/java`:

```
javac -d out -sourcepath ../../vcirs-core/java/src *.java
java -cp out WifiDoctor --echo < ../data/session-1.txt
```

dan seterusnya untuk sesi 2, 3, dan 4; atau, dari `Chapter03/python`, `python wifi_doctor.py --echo < ../data/session-1.txt`. (Di Windows PowerShell, yang tidak mengenal `<`, ketik `Get-Content ..\data\session-1.txt | java -cp out WifiDoctor --echo`.) Kedua bahasa mencetak transkrip yang sama dan menulis `wifi-kb.json` yang sama, byte demi byte.

Sesi 1: sarannya berhasil. Seorang pelanggan yang memakai Wi-Fi dan simbol Wi-Fi-nya mati.

```
Phone and Wi-Fi doctor: 6 fixes learned so far.
Answer each question; press Enter if you do not know.
Are you using Wi-Fi or mobile data? (wifi/mobile data) > wifi
Is the Wi-Fi symbol at the top of the screen on or off? (on/off) > off
Is airplane mode (the little plane) on? (on/off) > on
Advice: turn off airplane mode
Why:
  turn off airplane mode#1 fired because:
    connection = wifi  (answered by the user, CF 100)
    wifi icon = off  (answered by the user, CF 100)
    airplane mode = on  (answered by the user, CF 100)
    so fix = turn off airplane mode  [rule CF 100, conclusion CF 100]
Did it fix the problem? (yes/no) > yes
Good. The knowledge base was right; nothing new to learn.
```

Mesin mengajukan tiga pertanyaan, tidak lebih. Node tingkat atas yang pertama, *turn on Wi-Fi*, membutuhkan `connection = wifi` dan `wifi icon = off`; keduanya terpenuhi, sehingga node itu terpenuhi, dan mesin melanjutkan ke pengecualiannya, yang membutuhkan `airplane mode = on` — ditanyakan, dibenarkan, terpenuhi. Jawabannya berasal dari pengecualian itu, dan penjelasannya mendaftar ketiga fakta yang menghasilkannya. Pelanggan membenarkan, dan kasus itu dicatat sebagai terverifikasi: tidak ada yang baru untuk dipelajari, tetapi riwayat basis pengetahuan kini menunjukkan bahwa pengetahuan ini sudah diperiksa pada pelanggan sungguhan.

Sesi 2: sarannya gagal, dan dokter belajar. Wi-Fi menyala, ponsel-ponsel lain tidak bisa internetan, dan lampu router *mati*.

```
Phone and Wi-Fi doctor: 6 fixes learned so far.
Answer each question; press Enter if you do not know.
Are you using Wi-Fi or mobile data? (wifi/mobile data) > wifi
Is the Wi-Fi symbol at the top of the screen on or off? (on/off) > on
Do other phones get internet on the same Wi-Fi? (yes/no) > no
Are the router's lights normal, red or off? (normal/red/off) > off
Advice: restart the router
Why:
  restart the router#1 fired because:
    connection = wifi  (answered by the user, CF 100)
    wifi icon = on  (answered by the user, CF 100)
    other devices online = no  (answered by the user, CF 100)
    so fix = restart the router  [rule CF 100, conclusion CF 100]
Did it fix the problem? (yes/no) > no
What fixed it in the end? (Enter if nothing did) > switch the router on
Another fact that was different this time? (variable = value, Enter when done) >
Differences with the cornerstone case of restart the router#1:
  A (only in the old case): (none)
  B (only in the new case): router lights = off
Earlier cases whose answer would change: none
Learned: refined: switch the router on#1 added as an exception of restart the router#1
Saved ../data/wifi-kb.json: Phone and Wi-Fi doctor: 8 variables, 7 nodes, 7 rules
```

Kali ini node pertama gagal pada kondisi keduanya (simbol Wi-Fi menyala), sehingga mesin beralih ke node tingkat atas berikutnya, *restart the router*, dan menanyakan satu fakta yang belum dimilikinya. Node itu terpenuhi; pengecualiannya, *call the internet provider*, membutuhkan lampu merah, sehingga mesin bertanya tentang lampunya — mati — dan pengecualian itu tidak terpenuhi. Sarannya *restart the router*, dan penjelasannya menunjukkan alasannya. Tetapi router-nya tidak rusak; stekernya dicabut dari stopkontak. Mas Anto mengetik perbaikan yang sebenarnya. Ia tidak menambahkan fakta lain, karena fakta yang penting, lampu yang mati, sudah ditanyakan. Daftar perbedaan terhadap cornerstone *restart the router* menunjukkan persis satu fakta itu di bagian B; tidak ada kasus lama yang jawabannya berubah; dan dokter menambahkan `switch the router on#1` sebagai pengecualian `restart the router#1` — opsi 3 dari **Tabel 3.2**, dalam bahasa VCIRS.

Sesi 3: tidak ada saran sama sekali, dan sebuah variabel baru. Seorang pelanggan yang memakai data seluler, data selulernya menyala dan kuotanya masih ada, tetapi sinyalnya lemah.

```
Phone and Wi-Fi doctor: 7 fixes learned so far.
Answer each question; press Enter if you do not know.
Are you using Wi-Fi or mobile data? (wifi/mobile data) > mobile data
Is mobile data switched on? (on/off) > on
Is there still internet quota on the SIM card? (yes/no) > yes
I have no advice for this problem yet.
What fixed it in the end? (Enter if nothing did) > find a stronger signal
Another fact that was different this time? (variable = value, Enter when done) > signal
bars = 1
Another fact that was different this time? (variable = value, Enter when done) >
Learned: added: find a stronger signal#1 added at the top level
Saved ../data/wifi-kb.json: Phone and Wi-Fi doctor: 9 variables, 8 nodes, 8 rules
```

Tidak ada node yang terpenuhi: kedua node data seluler membutuhkan data seluler yang mati, atau kuota yang habis. Dokter mengatakannya dengan jujur dan bertanya apa yang memperbaikinya. Mas Anto memindahkan pelanggan ke dekat pintu, tempat sinyalnya lebih baik, dan ia menambahkan fakta yang belum pernah ditanyakan oleh pertanyaan mana pun: `signal bars = 1`. Mesin membuat variabel baru untuknya dan menambahkan kasus itu sebagai node tingkat atas yang baru (opsi 5). Mulai sekarang, dokter mengenal batang sinyal.

Sesi 4: masalah yang sama lagi. Pelanggan sesi 2 datang kembali seminggu kemudian, atau salah seorang tetangganya.

```
Phone and Wi-Fi doctor: 8 fixes learned so far.
Answer each question; press Enter if you do not know.
Are you using Wi-Fi or mobile data? (wifi/mobile data) > wifi
Is the Wi-Fi symbol at the top of the screen on or off? (on/off) > on
Do other phones get internet on the same Wi-Fi? (yes/no) > no
Are the router's lights normal, red or off? (normal/red/off) > off
Advice: switch the router on
Why:
  switch the router on#1 fired because:
    connection = wifi  (answered by the user, CF 100)
    wifi icon = on  (answered by the user, CF 100)
    other devices online = no  (answered by the user, CF 100)
    router lights = off  (answered by the user, CF 100)
    so fix = switch the router on  [rule CF 100, conclusion CF 100]
Did it fix the problem? (yes/no) > yes
Good. The knowledge base was right; nothing new to learn.
```

Dokter kini langsung memberikan jawaban yang benar, dan dapat menjelaskannya: empat fakta, termasuk yang dipelajarinya pada sesi 2. Inilah seluruh janji bab ini dalam satu transkrip. Tidak ada yang menulis satu baris kode pun di antara sesi 2 dan sesi 4. Pengetahuan itu datang dari orang yang tahu, dalam konteks sebuah kasus nyata, dan ia masuk tepat ke tempatnya.

3.7.1 Basis pengetahuan setelah empat pelanggan

`--tree` mencetak basis pengetahuan sebagai pohon node dan pengecualian:

```
(root)
+- turn on Wi-Fi#1: connection = wifi, wifi icon = off -> fix = turn on Wi-Fi
|  +- turn off airplane mode#1: airplane mode = on -> fix = turn off airplane mode
+- restart the router#1: connection = wifi, wifi icon = on, other devices online = no ->
fix = restart the router
|  +- call the internet provider#1: router lights = red -> fix = call the internet provider
|  +- switch the router on#1: router lights = off -> fix = switch the router on
+- turn on mobile data#1: connection = mobile data, mobile data = off -> fix = turn on
mobile data
+- buy a data package#1: connection = mobile data, mobile data = on, quota left = no -> fix
= buy a data package
+- find a stronger signal#1: connection = mobile data, mobile data = on, quota left = yes,
signal bars = 1 -> fix = find a stronger signal
```

Bandingkan dengan pohon biner **Gambar 3.2**. Pohon vcirs-core menampilkan setiap node dengan *semua* kondisinya dan meletakkan pengecualian langsung di bawah node-nya; cabang else mesin buatan tangan kita kini cukup menjadi urutan pencobaan node. Node yang dipelajari pada sesi 2 berada di samping node dari minggu pertama — dua pengecualian dari *restart the router*, satu untuk lampu merah dan satu untuk lampu yang mati sama sekali.

Berkas tersimpan, `wifi-kb.json`, adalah teks biasa yang dapat dibaca siapa saja. Inilah node yang dipelajari pada sesi 2, sebagaimana disimpan dalam berkas:

```
{
  "id": "switch the router on#1",
  "parent": "restart the router#1",
  "cf": 100,
  "conditions": [
    {
      "variable": "router lights",
      "operator": "=",
      "value": "off",
      "cf": 100,
      "credit": 1
    }
  ],
  "conclusions": [
    {
      "variable": "fix",
      "value": "switch the router on"
    }
  ],
  "cornerstone": {
    "facts": [
      {
        "variable": "connection",
        "operator": "=",
        "value": "wifi",
        "cf": 100
      },
      {
        "variable": "wifi icon",
        "operator": "=",
        "value": "on",
        "cf": 100
      },
      {
        "variable": "other devices online",
        "operator": "=",
        "value": "no",
        "cf": 100
      },
      {
        "variable": "router lights",
        "operator": "=",
        "value": "off",
        "cf": 100
      }
    ],
    "conclusions": [
      {
        "variable": "fix",
        "value": "switch the router on"
      }
    ],
    "cf": 100
  },
  "stopped": false,
  "generated": false,
  "author": "counter",
  "seq": 7
}
```


Node itu mencatat induknya, satu kondisinya dengan variabel, operator, nilai, faktor kepastian (*certainty factor*), dan Credit, konklusinya, seluruh kasus cornerstone (empat fakta pelanggan pada sesi 2), dan siapa yang mengajarkannya. Berkas itu juga menyimpan `history` (riwayat) setiap perubahan; entri-entri terakhirnya menyatakan bahwa sesi 2 menambahkan node ini

dengan opsi 3, sesi 3 menambahkan sebuah node dengan opsi 5, dan sesi 4 menerima jawaban node ini (opsi 0).

3.7.2 Hari Jumat Fajar

Pada hari Jumat Fajar menjaga konter sendirian, dengan basis pengetahuan dari keempat sesi tadi. Seorang pemuda duduk: ponselnya memakai Wi-Fi, simbol Wi-Fi-nya menyala, dan ponsel-ponsel lain di tempat kosnya bisa internetan — hanya ponselnya yang tidak. Dokter mengajukan tiga pertanyaan. Node pertama gagal pada simbol Wi-Fi; node kedua, *restart the router*, gagal pada perangkat-perangkat lain, yang ternyata online; node-node data seluler dan node sinyal langsung gagal pada koneksi, yang sudah diketahui. Tidak ada node yang terpenuhi, dan dokter menyatakan belum punya saran.

Fajar ingat bahwa tempat kos itu mengganti kata sandi Wi-Fi-nya minggu ini. Ia membuat ponsel itu melupakan jaringannya lalu bergabung lagi dengan kata sandi yang baru, dan internetnya kembali. Ia mengetik perbaikannya, *forget the network and enter the new password*, dan satu fakta lagi yang tidak ditanyakan oleh pertanyaan mana pun: `wifi password changed = yes`. Seperti pada sesi 3, tidak ada yang menjawab, sehingga `learn` menambahkan kasus itu sebagai node tingkat atas yang baru, dengan keempat faktanya sebagai kondisi.

Pada hari Senin Mas Anto mencetak pohonnya dengan `--tree` dan menemukan node baru itu di bagian bawah, bersama kasus cornerstone-nya dan entri riwayat yang mencatat kapan node itu ditambahkan. Ia tidak perlu berada di konter untuk memperoleh pengetahuan itu, dan Fajar tidak perlu tahu apa pun tentang aturan untuk memberikannya. Bila Mas Anto menganggap node itu terlalu sempit — barangkali perangkat lain tidak penting bila kata sandinya sudah berganti — ia dapat mengajarkan node yang lebih baik ketika pelanggan seperti itu datang lagi, dan menghentikan node yang ini.

3.8 Membuatnya Lebih Baik

Angka, bukan hanya kata. Node yang dipelajari pada sesi 3 berbunyi `signal bars = 1`. Pelanggan dengan *nol* batang sinyal memiliki masalah yang sama tetapi tidak akan cocok. Pemanggilan `learn` membangun kondisi dari fakta sebagaimana diberikan, dan itulah bawaan yang aman. Bila pakar tahu lebih baik, `refine` menerima kondisi-kondisi yang dipilihnya, dan sebuah kondisi boleh memakai operator mana pun dari mesin, seperti `signal bars <= 1`. Soal Latihan 3.6 mengerjakan persis hal ini.

Pertanyaan dalam urutan yang lebih baik. Dokter mengajukan pertanyaan dalam urutan node-nya. VCIRS dapat berbuat lebih baik: tingkat pemakaian di Bab 7 mengurutkan variabel menurut seberapa banyak variabel itu dipakai, dan `questionOrder()` milik mesin menaruh pertanyaan yang paling berguna di depan. Dengan beberapa ratus node, selisih antara menanyakan pertanyaan yang tepat lebih dahulu dan yang keliru adalah beberapa pertanyaan untuk setiap pelanggan.

Menghentikan saran yang buruk. Misalkan Mas Anto memutuskan bahwa menyuruh pelanggan menelepon penyedia layanan tidak ada gunanya karena penyedia itu tidak pernah menjawab, dan ia lebih suka menyarankan kunjungan ke kantor penyedia layanan. RDR tidak pernah menghapus node, tetapi dapat **menghentikannya**: `kb.stop(nodeId)` mempertahankan node, cornerstone-nya, dan pengecualian-pengecualiannya, tetapi konklusinya tidak lagi diberikan. Node baru dengan saran yang lebih baik kemudian dapat diajarkan seperti biasa.

Meninjau cornerstone yang terdampak. Dalam sesi-sesi kita, daftar cornerstone yang terdampak selalu kosong. Bila tidak kosong, aplikasi sebaiknya menampilkan kasus-kasus itu dan menanyakan apakah jawabannya memang harus berubah, lalu mengizinkan pakar menambahkan fakta lain bila tidak. Ini hanya beberapa baris lagi yang sejenis dengan daftar perbedaan, dan Soal Latihan 3.8 memintanya.

Lebih dari satu jawaban yang benar. Bila seorang pelanggan kehabisan kuota sekaligus mendapat sinyal lemah, dokter memberikan perbaikan pertama yang ditemukannya. Memanggil `kb.infer(facts, asker, true)` (Python: `multiple=True`) membuat mesin melanjutkan setelah jawaban pertama dan mengikuti setiap cabang yang berlaku, lalu mengembalikan satu jalur untuk setiap konklusi. Untuk sebuah konter, kebiasaannya adalah mencoba perbaikan pertama, lalu kembali ke dokter bila masalahnya masih ada, dan dokter kemudian belajar dari situ. Untuk penasihat kesehatan lain ceritanya, dan Bab 5 memakai mode ganda ini.

> **JEBAKAN:** Basis pengetahuan yang diajari oleh satu orang mencerminkan kebiasaan orang itu, yang baik maupun yang buruk. Bila Mas Anto selalu me-restart router lebih dahulu karena cepat, dokter akan belajar menyarankannya bahkan ketika sebuah pertanyaan tentang lampu akan menemukan penyebabnya lebih cepat. Tinjaulah pohonnya sesekali, dan hentikan node yang sarannya sudah menjadi kebiasaan yang buruk.

Dua bahasa, satu basis pengetahuan. Karena kedua versi mesin membaca dan menulis JSON yang sama, Fajar dapat menjalankan versi Python di laptop tua konter sementara Mas Anto menjalankan versi Java di rumah, dan mereka dapat menyalin `wifi-kb.json` di antara keduanya. Berkasnya kecil; satu baris untuk setiap fakta.

Uji. `RdrTest.java` dan `test_rdr.py` memeriksa mesin buatan tangan (sebuah pengecualian terpenuhi, kasus lama mempertahankan jawabannya, perbedaan-perbedaannya benar) dan basis pengetahuan awal (enam node, node penyedia layanan di bawah node router, jawaban yang benar untuk lampu merah, dan bahwa `learn` melakukan penghalusan bila memang seharusnya). Keduanya berakhir dengan:

```
all tests passed
```

3.9 Memakainya Sungguh-Sungguh

Di konter ponsel. Jalankan dokter di komputer konter dan jawablah pertanyaan-pertanyaannya sambil berbicara dengan pelanggan. Ketika sarannya gagal, ajarilah sebelum pelanggan pergi, selagi kasusnya masih segar; hanya perlu kurang dari satu menit. Setelah sebulan, basis pengetahuan menyimpan pengalaman Anda selama bulan itu, dan pembantu Anda memberikan saran Anda setiap hari Jumat. Simpanlah salinan `wifi-kb.json` setiap minggu: berkasnya kecil, dan itulah pengetahuan Anda.

Berapa banyak kasus untuk memulai? Lebih sedikit daripada yang Anda kira. Keenam kasus awal dalam bab ini hanya memerlukan sepuluh menit bagi Mas Anto untuk menuliskannya, dan keenamnya sudah menjawab keluhan-keluhan yang paling umum. Tuliskan tiga sampai sepuluh situasi yang Anda temui setiap minggu sebagai kasus awal, satu per baris dalam berkas kasus, dan biarkan hal lainnya diajarkan di konter. Tahanlah godaan untuk menuliskan di muka setiap aturan yang dapat Anda bayangkan; aturan yang ditulis tanpa kasus nyata tidak memiliki cornerstone yang pernah diperiksa, dan aturan semacam itu membawa kembali masalah daftar datar. Setelah beberapa minggu, cetaklah pohonnya dengan `--tree` dan bacalah bersama seorang

rekan: node yang tidak dikenali siapa pun, atau dua node yang mengatakan hal yang sama dengan kata-kata berbeda, layak dibicarakan.

Apakah dokter semakin pintar? Riwayat dalam `wifi-kb.json` menjawab pertanyaan ini tanpa usaha tambahan. Setiap konsultasi yang berhasil dicatat sebagai opsi 0, setiap pengecualian baru sebagai opsi 3, dan setiap node tingkat atas yang baru sebagai opsi 5. Hitunglah ketiganya setiap minggu. Pada minggu-minggu pertama, opsi 3 dan 5 akan sering muncul: dokter masih banyak belajar. Seiring waktu, bagian opsi 0 seharusnya bertambah besar, karena semakin banyak pelanggan datang dengan masalah yang sudah dikenalnya. Bila suatu minggu opsi 5 tiba-tiba melonjak, ada jenis masalah baru — pembaruan sistem ponsel, gangguan jaringan di daerah itu — dan itulah saat yang tepat untuk membaca node-node yang baru bersama-sama. Angka-angka ini bukan ukuran yang sempurna, karena pelanggan yang tidak kembali tidak pernah mengatakan apakah sarannya berhasil, tetapi angka-angka itu jujur dan murah, dan menunjukkan arah yang benar.

Pemecah masalah lainnya. Tidak ada bagian `WifiDoctor` yang khusus tentang Wi-Fi kecuali daftar pertanyaan dan berkas kasus awalnya. Sebuah toko servis laptop, bengkel sepeda motor, petugas komputer di sekolah, tukang servis pompa air — masing-masing dapat menuliskan pertanyaannya sendiri dan beberapa kasus awal, lalu mulai mengajar. Bab 8 membangun pencari kerusakan sepeda motor dengan *forward chaining* dan *backward chaining* milik mesin; Bab 14 membangun alat yang memungkinkan siapa pun membuat pemecah masalah seperti ini tanpa pemrograman.

Banyak pengajar. Ketika beberapa orang mengajari basis pengetahuan yang sama, setiap node mencatat pengajarnya dan riwayat mencatat setiap perubahan, sehingga sebuah ketidaksepakatan dapat dilacak ke sebuah kasus dan seseorang. Bab 11 membangun di atas hal ini untuk tim sebuah klinik.

Saran yang aman. Sebuah pemecah masalah hanya boleh menyarankan apa yang dapat dilakukan pelanggan dengan aman. Jangan ajarkan perbaikan yang berarti membuka ponsel, mengisi daya baterai yang menggembung, atau memanjat ke router di tiang; ajarkan untuk berkata “bawalah ke pusat servis”. Baterai yang menggembung atau panas berisiko terbakar, dan saran yang benar adalah berhenti memakai ponsel itu.

Hanya fakta yang diperlukan. Kasus-kasus yang disimpan dalam basis pengetahuan adalah fakta teknis, seperti keadaan simbol Wi-Fi atau lampu router. Tidak ada alasan menyimpan nama, nomor, atau pesan pelanggan di dalamnya, dan ada alasan kuat untuk tidak melakukannya: basis pengetahuan dimaksudkan untuk dibagikan, dan ia seharusnya menyimpan pengetahuan tentang masalah, bukan tentang orang.

3.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Menaruh aturan baru di depan, atau di belakang, daftar yang datar. Keduanya dapat mengubah jawaban bagi pelanggan yang tidak dimaksudkan oleh aturan itu, seperti ditunjukkan **Kode 3.1**. *Hindari:* tambahkan pengetahuan sebagai pengecualian dalam konteks tempat jawaban yang salah berasal.

Memilih terlalu banyak perbedaan. Bila Mas Anto memilih *semua* perbedaan untuk pelanggan 1 (`wifi icon = off` dan `airplane mode = off`), Aturan 1 tidak akan terpenuhi untuk pelanggan-pelanggan berikutnya yang tidak menyebut mode pesawat. *Hindari:* pilih fakta-fakta yang *menjelaskan* jawaban yang berbeda, bukan semua yang kebetulan berbeda.

Memilih terlalu sedikit. Kondisi yang juga benar untuk kasus cornerstone tidak dapat membedakan kedua kasus; mesin menolak belajar bila daftar perbedaannya kosong. *Hindari:* ketika mesin menyatakan tidak dapat membedakan kasusnya, tambahkan fakta yang memang berbeda.

Menyunting atau menghapus aturan lama. Hal ini merusak jaminan bahwa kasus-kasus lama mempertahankan jawabannya. *Hindari:* perhalus dengan pengecualian; untuk mengistirahatkan sebuah saran, hentikan node-nya.

Mengajarkan perbaikan yang belum terbukti. Bila "perbaikan" itu tidak benar-benar menyelesaikan masalah, basis pengetahuan mempelajari sebuah kesalahan. *Hindari:* ajarkan hanya setelah masalah pelanggan benar-benar terselesaikan.

Mengajar sambil terburu-buru. Ketika antrean panjang, godaannya adalah mengetik perbaikan tanpa memikirkan fakta mana yang benar-benar berbeda. *Hindari:* bila tidak ada waktu, catat kasusnya di kertas dan ajarkan setelah konter sepi; pengetahuan yang diajarkan satu jam kemudian tetap sama bergunanya.

Menjawab pertanyaan yang tidak Anda yakini. Fakta yang salah membuat node yang salah terpenuhi. *Hindari:* tekan Enter untuk "saya tidak tahu"; mesin memperlakukan fakta yang tidak diketahui sebagai tidak cocok.

Kata-kata berbeda untuk hal yang sama. `wifi`, `Wi-Fi`, dan `WiFi` adalah tiga nilai berbeda bagi mesin. *Hindari:* pakai jawaban-jawaban lazim yang ditawarkan pertanyaan, yang ditampilkan aplikasi dalam tanda kurung; mesin mengumpulkannya dari basis pengetahuan [3].

Memindahkan berkas basis pengetahuan. Aplikasi mencari `../data/wifi-kb.json` dan berkas kasus awal di sampingnya. *Hindari:* pertahankan susunan foldernya, atau berikan jalurnya sebagai argumen.

Lupa bahwa pohonnya dibangun dari kasus. Pohon indah yang diketik dengan tangan adalah basis aturan datar yang menyamar, lengkap dengan semua masalahnya. *Hindari:* biarkan pengetahuan tumbuh dari kasus-kasus nyata.

3.11 Ringkasan Bab

- **Daftar datar** aturan, yang dicoba dari atas, membiarkan aturan baru mengubah jawaban kasus-kasus yang tidak dimaksudkannya; urutannya menjadi pengetahuan tersembunyi yang rapuh (**Kode 3.1**, **Kode 3.2**).
- **Ripple Down Rules** menyimpan pengetahuan sebagai **pengecualian** dalam konteks tempat sebuah kesalahan terjadi. Setiap aturan menyimpan **kasus cornerstone**-nya; aturan baru dibangun dari **perbedaan** antara kasus baru dan cornerstone itu, sehingga jawaban kasus lama tidak berubah (**Gambar 3.1**).
- Dalam RDR klasifikasi tunggal, basis pengetahuan adalah pohon biner dengan cabang **except** (dicoba ketika benar) dan **else** (dicoba ketika salah); jawabannya adalah konklusi aturan terakhir yang benar, dan aturan baru digantungkan di tempat pencarian berhenti (**Gambar 3.2**).
- Seluruh mesinnya muat dalam seratus baris (**Kode 3.3**, **Kode 3.4**); lima pelanggan mengajarinya empat aturan tanpa mengubah satu pun jawaban sebelumnya (**Kode 3.5**, **Kode 3.6**).
- **vcirs-core** mempertahankan jantung yang sama dan menambahkan pertanyaan yang diajukan hanya bila diperlukan, **daftar perbedaan** dengan bagian A dan B, **enam opsi** pembangunan

pengetahuan VCIRS, **cornerstone yang terdampak**, penjelasan, dan basis pengetahuan JSON yang dipakai bersama oleh Java dan Python (**Tabel 3.1**, **Tabel 3.2**).

- **Dokter Wi-Fi** (**Kode 3.7**, **Kode 3.8**) mengajukan tiga sampai empat pertanyaan untuk setiap pelanggan, menjelaskan setiap saran, mencatat kasus terverifikasi, mempelajari pengecualian ketika sarannya gagal, mempelajari node tingkat atas baru dengan variabel baru ketika tidak punya saran, dan langsung memberikan jawaban yang benar kepada pelanggan berikutnya dengan masalah yang sama.

3.12 Soal Latihan

3.1 Dengan tangan, memakai pohon **Gambar 3.2**, temukan jawabannya dan daftarkan aturan-aturan yang dicoba untuk: (a) Wi-Fi off, mode pesawat off; (b) Wi-Fi on, perangkat lain online yes; (c) Wi-Fi on, perangkat lain online no, lampu router red, mode pesawat on.

3.2 Dalam **Kode 3.1**, adakah urutan *apa pun* dari ketiga aturan yang memberikan jawaban benar kepada keempat pelanggan? Cobalah keenam urutan dengan tangan atau dengan sedikit mengubah program, lalu jelaskan hasilnya.

3.3 Lanjutkan **Kode 3.6** dengan pelanggan keenam: Wi-Fi off, mode pesawat off, *battery saver on*, dan perbaikan yang benar adalah *matikan battery saver* (sebagian ponsel mematikan Wi-Fi untuk menghemat daya). Aturan mana yang menjawab lebih dahulu, apa perbedaan-perbedaannya, dan ke mana aturan baru ditaruh? Gambarlah pohon yang baru.

3.4 Dalam mesin buatan tangan, mengapa `learn` harus mengklasifikasi kasus itu sekali lagi alih-alih memakai jawaban yang sudah dicetak program? Apa yang bisa salah bila pohonnya berubah di antara keduanya?

3.5 Untuk sesi 2 di Subbab 3.7, tuliskan daftar perbedaannya dengan tangan: kasus cornerstone `restart the router#1` dan fakta-fakta kasus baru. Mengapa bagian A kosong?

3.6 Sesi 3 mengajarkan node `signal bars = 1`. Dengan memakai `refine` dan kondisi pilihan, ajarkan sebagai gantinya sebuah node dengan kondisi `signal bars <= 1`, mulai dari basis pengetahuan awal, dan tunjukkan bahwa node itu menjawab untuk 0 dan 1 batang tetapi tidak untuk 3.

3.7 Jelaskan dengan kata-kata Anda sendiri perbedaan antara opsi 2 dan opsi 5 dalam **Tabel 3.2**, dengan sesi 2 dan sesi 3 sebagai contoh. Mengapa `learn` memilih opsi 3 pada sesi 2, bukan opsi 2?

3.8 Uraikan bagaimana Anda akan memperluas `WifiDoctor` sehingga, bila daftar cornerstone yang terdampak tidak kosong, program menampilkan setiap kasus yang terdampak dan mengizinkan Mas Anto menambahkan fakta lain sebelum basis pengetahuan berubah. Pemanggilan vcirs-core apa saja yang akan Anda pakai?

3.9 Pada Hari Jumat Fajar (Subbab 3.7.2), node baru memiliki empat kondisi. Kondisi mana yang akan Anda pertahankan bila boleh memilih, dan mengapa? Dengan urutan penelusuran node tingkat atas oleh dokter, jelaskan pertanyaan apa saja yang akan diajukan kepada pelanggan berikutnya dengan masalah yang sama, dan apakah `wifi password changed` pernah ditanyakan.

3.13 Penyelesaian Soal Latihan

Penyelesaian 3.1. (a) Aturan 0 benar; Aturan 1 (*wifi icon = off*) benar; Aturan 3 (*airplane mode = on*) salah, tanpa cabang else: berhenti. Jawabannya *nyalakan Wi-Fi* (Aturan 1). (b) Aturan 0; Aturan 1 salah → else; Aturan 2 (*other devices online = no*) salah, tanpa cabang else: berhenti.

Aturan terakhir yang benar adalah Aturan 0: *belum ada saran*. (c) Aturan 0; Aturan 1 salah → else; Aturan 2 benar → except; Aturan 4 (*router lights = red*) benar, tanpa cabang except: berhenti. Jawabannya *hubungi penyedia layanan internet*; mode pesawat tidak pernah dilihat.

Penyelesaian 3.2. Sebut aturan-aturannya W (*wifi icon = off*), R (*other devices online = no*), dan A (*airplane mode = on*). Pelanggan 3 cocok dengan W dan A, sehingga A harus berada sebelum W; pelanggan 4 cocok dengan R dan A, sehingga R harus berada sebelum A; pelanggan 1 dan 2 masing-masing hanya cocok dengan satu aturan. Dari keenam urutan, tepat satu yang memenuhi kedua batasan itu: R, A, W — dan urutan itu memang memberikan jawaban benar kepada keempat pelanggan. Namun pelajarannya ada pada apa yang terjadi sesudahnya. Pelanggan kelima dengan Wi-Fi off, mode pesawat on, dan ponsel-ponsel lain tidak online, yang perbaikannya adalah mematikan mode pesawat, cocok dengan R lebih dahulu dan mendapat jawaban yang salah; memperbaikinya membutuhkan A sebelum R, yang bertentangan dengan pelanggan 4. Setiap kali datang pelanggan baru, pencarian urutan dimulai lagi, dan kadang-kadang tidak ada urutan yang memenuhi. Pengecualian menghindari pencarian itu sama sekali: dalam pohon RDR, aturan pelanggan 5 berada dalam konteks Wi-Fi yang mati.

Penyelesaian 3.3. Aturan 1 menjawab *nyalakan Wi-Fi* (Aturan 3 salah, karena mode pesawat off). Perbedaan dari kasus cornerstone Aturan 1 (Wi-Fi off, pesawat off) hanyalah `battery saver = on`. Pencarian berhenti di Aturan 3, yang salah, sehingga Aturan 5 yang baru, *battery saver = on* → *matikan battery saver*, digantungkan pada cabang **else** Aturan 3: aturan itu tetap merupakan pengecualian Aturan 1, yang dicoba setelah pengecualian pesawat. Pohonnya menjadi: Aturan 0 → except Aturan 1 → except Aturan 3, else Aturan 5; Aturan 1 → else Aturan 2 → except Aturan 4.

Penyelesaian 3.4. Tempat menggantungkan aturan baru adalah tempat pencarian untuk kasus *ini* berhenti, yang bergantung pada seluruh pohon. Mengklasifikasi ulang menjamin bahwa tempat itu dihitung pada pohon yang sekarang. Bila ada aturan lain yang ditambahkan di antaranya (misalnya oleh seorang rekan pada basis pengetahuan yang sama), titik berhenti jawaban lama bisa keliru, dan aturan baru dapat digantungkan di tempat yang tidak pernah dicapai kasus ini — sehingga aturan itu tidak akan pernah terpenuhi.

Penyelesaian 3.5. Cornerstone `restart the router#1` adalah kasus awal: `connection = wifi`, `wifi icon = on`, `other devices online = no`. Kasus baru memiliki ketiga fakta itu dan `router lights = off`. Bagian A (hanya pada kasus lama) kosong karena setiap fakta kasus lama juga ada pada kasus baru; bagian B memuat `router lights = off`. Jadi di sini opsi 1 dan 3 memberikan node yang sama dengan opsi 2.

Penyelesaian 3.6. Program penyelesaiannya adalah `SignalRule.java` dan `signal_rule.py`. Versi Python-nya:

```
"""Soal Latihan 3.6: ajarkan aturan dengan kondisi angka, dipilih sendiri."""

import pathlib
import sys

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import TOP_LEVEL, Case, Condition, KnowledgeBase, casefile

name, description, steps = casefile.read_cases("../data/wifi-seed-cases.json")
kb = casefile.build(KnowledgeBase(name, description), steps)
case = Case.of(["connection = mobile data", "mobile data = on",
                "quota left = yes", "signal bars = 1"],
               ["fix = find a stronger signal"])
# Pakar memakai fakta kasus, tetapi "signal bars = 1" dilebarkan jadi "<= 1".
chosen = [Condition.parse("connection = mobile data"),
          Condition.parse("mobile data = on"),
          Condition.parse("signal bars <= 1")]
node = kb.refine(case, TOP_LEVEL, chosen=chosen, author="Anto")
print("New node: " + str(node))
for bars in ["0", "1", "3"]:
    answer = kb.infer({"connection": "mobile data", "mobile data": "on",
                       "quota left": "yes", "signal bars": bars})
    print("signal bars = " + bars + ": " + answer.what()[0])
```

Program itu mencetak:

```
New node: find a stronger signal#1: connection = mobile data, mobile data = on, signal bars
<= 1 -> fix = find a stronger signal
signal bars = 0: Conclusion: fix = find a stronger signal  (rule find a stronger signal#1,
CF 100)
signal bars = 1: Conclusion: fix = find a stronger signal  (rule find a stronger signal#1,
CF 100)
signal bars = 3: No node matched the case.
```

Dengan tiga batang, kondisi `signal bars <= 1` salah, dan tidak ada node yang cocok — sebagaimana mestinya, karena sinyal yang baik bukanlah masalah ini. Mesin membandingkan `signal bars` sebagai bilangan karena kedua sisinya bilangan.

Penyelesaian 3.7. Opsi 2 membangun node baru dari bagian B daftar perbedaan dan menaruhnya *di bawah* node lama, sebagai pengecualian; opsi 5 membangunnya dari *semua* fakta kasus dan menaruhnya di *tingkat atas*, tanpa induk. Pada sesi 2 sebuah node (*restart the router*) memberikan jawaban yang salah, sehingga pengetahuan yang baru adalah pengecualian darinya; `learn` memakai opsi 3, gabungan negasi bagian A dan bagian B, karena itulah pilihan aman yang paling spesifik — dan di sini bagian A kosong, sehingga hasilnya sama dengan opsi 2. Pada sesi 3 tidak ada node yang menjawab sama sekali, sehingga tidak ada yang dapat dijadikan induk pengecualian, dan `learn` memakai opsi 5.

Penyelesaian 3.8. Setelah menghitung kondisi-kondisinya (`conds`), lakukan perulangan: panggil `kb.affectedCornerstones(fired.id, conds, c.conclusions)`; bila daftarnya kosong, panggil `learn` (atau `refine(c, BOTH, fired.id, conds, "counter")`). Bila tidak kosong, cetak setiap node yang terdampak dengan `kb.node(id).cornerstone` dan konklusinya, lalu tanyakan apakah jawabannya memang harus berubah. Bila Mas Anto menjawab tidak, mintalah fakta lain, tambahkan ke kasus dan ke `conds`, lalu ulangi. Bila ia menerima, panggil `refine` dengan kondisi-kondisi yang dipilih. Inilah prosedur RDR "ulangi sampai pakar menerima aturan dan akibat-akibatnya" [4].

Penyelesaian 3.9. Kondisi yang menjelaskan perbaikannya adalah `wifi password changed = yes`, bersama `connection = wifi`; `wifi icon = on` dan `other devices online = yes` hanya

menggambarkan pelanggan ini. Node yang hanya memiliki kedua kondisi itu juga akan menangkap pelanggan yang perangkat-perangkat lainnya offline karena router-nya mati, tetapi node-node router dicoba lebih dahulu, sehingga mereka akan menjawab pelanggan seperti itu sebelum node ini tercapai. Dengan keempat kondisi sebagaimana dipelajari, pelanggan berikutnya ditanya tentang koneksi, simbol Wi-Fi, dan perangkat lain, persis seperti pemuda tadi; bila semuanya cocok, mesin sampai ke node baru dan mengajukan pertanyaan baru, `wifi password changed`, yang variabelnya tidak memiliki pertanyaan yang dideklarasikan, sehingga dokter menanyakannya dengan namanya. Pertanyaan itu tidak pernah diajukan kepada pelanggan yang memakai data seluler.

Daftar Pustaka

[1] ANTARA News, “Survei APJII: Penetrasi internet di Indonesia 2026 capai 81,7 persen,” May 2026. [Online]. Available: https://www.antaranews.com/berita/5576225/survei-apjii-penetrasi-internet-di-indonesia-2026-capai-817-persen

[2] P. Compton, G. Edwards, B. Kang, L. Lazarus, R. Malor, T. Menzies, P. Preston, A. Srinivasan, and C. Sammut, “Ripple down rules: possibilities and limitations,” in *Proc. 6th AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1991, pp. 6.1–6.20.

[3] M. M. I. Subakti, “A Variable-Centered Intelligent Rule System,” M.Sc.Eng. thesis, Dept. of Computer Science and Information Engineering, Taipei, Taiwan, Jun. 2005.

[4] P. Compton and B. H. Kang, *Ripple-Down Rules: The Alternative to Machine Learning*. Boca Raton, FL, USA: CRC Press, 2021.

[5] P. Compton, R. Horn, R. Quinlan, and L. Lazarus, “Maintaining an expert system,” in *Applications of Expert Systems*, J. R. Quinlan, Ed. London, U.K.: Addison-Wesley, 1989, pp. 366–385.

[6] G. Edwards, P. Compton, R. Malor, A. Srinivasan, and L. Lazarus, “PEIRS: a pathologist-maintained expert system for the interpretation of chemical pathology reports,” *Pathology*, vol. 25, no. 1, pp. 27–34, 1993.

[7] D. Richards, “Two decades of ripple down rules research,” *The Knowledge Engineering Review*, vol. 24, no. 2, pp. 159–184, 2009.

[8] I. Subakti, “Some revisions in VCIRS and cases reconstructing perspectives,” in *Proc. Information and Communication Technology Seminar (ICTS 2006)*, Surabaya, Indonesia, 2006.

[9] I. Subakti, *Variable-Centered Intelligent Rule System*, Indonesian ed. (resume). Surabaya, Indonesia: Dept. of Informatics, Institut Teknologi Sepuluh Nopember, 2006.

[10] B. H. Kang, P. Compton, and P. Preston, “Multiple classification ripple down rules: evaluation and possibilities,” in *Proc. 9th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1995, pp. 17.1–17.20.

[11] I. Subakti, “A variable-centered intelligent rule system,” in *Proc. 1st Annual Int. Conf. Information and Communication Technology Seminar (ICTS 2005)*, vol. 1, no. 1, Surabaya, Indonesia, Aug. 2005, pp. 167–174.

Bab 4

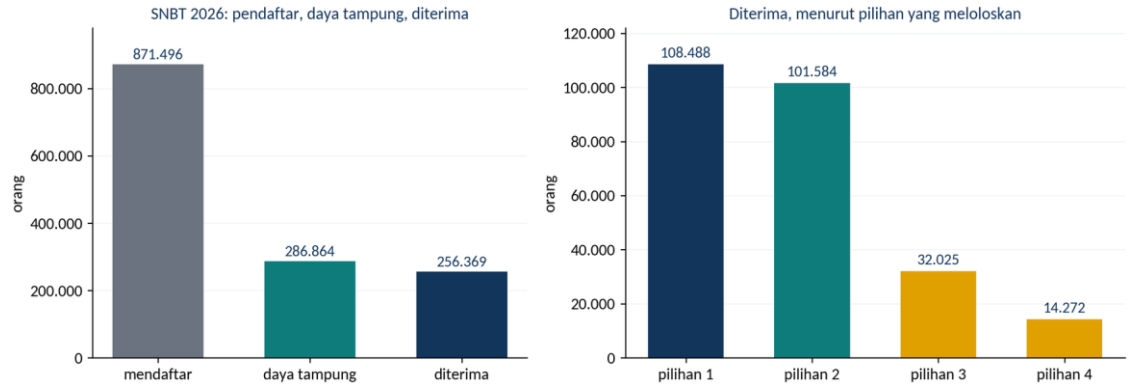
Penasihat Pemilihan Jurusan: Berpikir dengan Variabel

Di akhir bab ini Anda akan mampu mengambil nasihat seorang yang berpengalaman, yang ditulis sebagai kalimat biasa, lalu mengubahnya menjadi **variabel**, **nilai**, **kondisi**, **node**, dan **rule**; menjelaskan apa itu *Variable-Centered Rule Structure* dan mengapa menyimpan pengetahuan dengan berpusat pada variabel membuatnya lebih kecil, lebih mudah diperiksa, dan lebih mudah dikembangkan; membangun struktur itu dengan tangan, dalam Java dan Python, sebagai tiga tabel kecil — tabel Variabel, Node Structure, dan Rule Structure; serta menjalankan **penasihat pemilihan jurusan**, aplikasi di atas mesin bersama vcirs-core yang membantu lulusan SMA memikirkan program studi yang cocok baginya, hanya menanyakan yang perlu, menjelaskan setiap sarannya, menunjukkan program-program yang dilewati di jalan, dan belajar dari guru BK ketika ia tidak setuju. Anda juga akan membaca struktur yang disimpan mesin dan melihat, lewat tabel-tabelnya, variabel dan node mana yang paling penting — langkah pertama menuju pembangkitan rule di Bab 6.

4.1 Laras dan Bu Wati

Setiap siang pukul setengah dua, Pak Joko memarkir gerobak siomaynya di gerbang sebuah SMA negeri di pinggir kota. Pukul dua para siswa keluar, dan pukul setengah tiga dandangnya hampir kosong. Putrinya, Laras, bersekolah di SMA yang sama, di kelas XII, dan tahun ini pertanyaan di tikar makan malam keluarga itu selalu sama: *Laras mau melanjutkan ke program studi apa, dan di mana?*

Pertanyaan itu lebih sulit daripada kedengarannya, dan angka-angkanya menunjukkan sebabnya. Perguruan tinggi negeri di Indonesia menerima mahasiswa baru terutama lewat dua seleksi nasional yang diselenggarakan SNPMB. Yang pertama, **SNBP**, berdasarkan prestasi di sekolah: hanya siswa terbaik tiap sekolah yang *eligible* — 40 persen terbaik di sekolah berakreditasi A, 25 persen di sekolah berakreditasi B, 5 persen di sekolah berakreditasi C — dan tiap siswa eligible memilih paling banyak dua program studi; pada 2026 hasilnya diumumkan tanggal 31 Maret [1]. Yang kedua, **SNBT**, berdasarkan tes tertulis berbasis komputer, UTBK. Pada 2026 pendaftarannya berlangsung 25 Maret sampai 7 April, tesnya sendiri 21 sampai 30 April, dan hasilnya diumumkan 25 Mei [2], [3]. Seorang siswa boleh memilih sampai empat program studi, dengan aturan tentang berapa yang boleh program akademik (S1) dan berapa yang vokasi, dan ia hanya boleh mengikuti UTBK sekali pada 2026 [3]. **Gambar 4.1** memperlihatkan hasilnya. Dari 871.496 pendaftar SNBT 2026, 256.369 diterima — 29,42 persen — pada 286.864 kursi yang tersedia; 86.118 di antara yang diterima, sepertiganya, adalah pendaftar KIP Kuliah, yang biaya studinya dibantu pemerintah [4].



Gambar 4.1 SNBT 2026 dalam angka: kurang dari tiga dari sepuluh pendaftar diterima, dan sebagian besar yang diterima lolos lewat pilihan pertama atau kedua. Data dari siaran pers SNPMB 25 Mei 2026 [4].

Separuh kanan gambar itu memuat pelajaran yang paling penting bagi keluarga seperti keluarga Pak Joko. Dari yang diterima, 108.488 orang lolos lewat pilihan pertama dan 101.584 lewat pilihan kedua; hanya 46.297 lewat pilihan ketiga atau keempat [4]. Pilihan di urutan teratas mengerjakan hampir seluruh pekerjaannya. Memilihnya dengan baik — program yang cocok dengan siswa, yang dapat dijangkaunya, dan yang dapat didukung keluarganya — jauh lebih berharga daripada kiat apa pun pada hari tes.

Di sinilah Bu Wati berperan. Sudah sembilan belas tahun ia menjadi guru BK di sekolah itu, dan ia telah duduk bersama ratusan siswa seperti Laras. Ia tidak tahu masa depan, tetapi ia mengenal pola. *Anak perempuan yang menyukai makhluk hidup dan tidak takut darah sering berhasil di keperawatan, dan bila nilai IPA-nya kuat, ia sebaiknya melihat kedokteran juga. Anak laki-laki yang menyukai komputer tetapi keluarganya membutuhkan ia segera bekerja sebaiknya melihat diploma tiga (D3) informatika, bukan hanya sarjana empat tahun.* Ia menyampaikan hal-hal itu satu per satu kepada siswa, di ruangan kecilnya di samping perpustakaan, dan tiap tahun antrean di depan pintunya lebih panjang daripada waktu yang ia punya.

Bu Wati menginginkan dua hal. Ia ingin setiap siswa mendengar nasihatnya, bahkan yang pemalu dan tidak pernah mengetuk pintunya. Dan ia ingin melihat pengetahuannya sendiri terbentang di depannya, agar ia dapat memeriksanya, membetulkannya, dan mewariskannya kepada guru BK muda yang kelak menggantikannya. Bab ini membangun program yang melakukan keduanya. Di sepanjang jalan, bab ini mengajarkan gagasan yang memberi VCIRS namanya: **berpikir dengan variabel**.

> **JEBAKAN:** Penasihat dalam bab ini memberikan saran seorang guru BK, bukan keputusan. Persyaratan resmi setiap program studi — nilai minimum, mata pelajaran pendukung, apakah program itu menerima pendaftar dengan buta warna, daya tampungnya — diumumkan oleh SNPMB dan oleh tiap perguruan tinggi, dan berubah dari tahun ke tahun. Siswa selalu memeriksanya di sana sebelum memilih [2].

4.2 Gagasannya dengan Kata-Kata Sederhana

4.2.1 Dari nasihat ke variabel

Dengarkan sekali lagi salah satu kalimat Bu Wati: *anak perempuan yang menyukai makhluk hidup dan tidak takut darah sering berhasil di keperawatan.* Ada tiga hal berbeda di dalamnya. Ada sesuatu yang ditanyakan Bu Wati — apa yang disukai siswa. Ada jawaban yang ia tunggu — makhluk hidup. Dan ada nasihat yang ia berikan bila jawaban-jawabannya cocok — keperawatan.

VCIRS memberi nama kepada masing-masing. Yang ditanyakan adalah **variabel**: `likes`, `blood ok`, `science score`. Jawabannya adalah **nilai** variabel itu: `living things`, `yes`, `88`. Sebuah variabel bersama sebuah pembanding dan sebuah nilai adalah **kondisi**: `likes = living things`, `science score >= 85`. Dan nasihatnya adalah **konklusi**, yang juga berupa variabel dengan nilainya: `program = nursing`. (Nama variabel dan nilai dalam program ditulis dalam bahasa Inggris, sama untuk kedua edisi buku ini, agar basis pengetahuannya dapat dipakai bersama; artinya dijelaskan di **Tabel 4.1**.)

Pemisahan ini tampak terlalu sederhana untuk diberi nama, tetapi di sinilah seluruh kiat bab ini. Selama pengetahuan tinggal di dalam kalimat, komputer hanya dapat menyimpan dan mencetaknya kembali. Begitu dipecah menjadi variabel dan nilai, komputer dapat *menanyakan* nilai sebuah variabel, *membandingkan* jawaban dengan sebuah kondisi, *menghitung* seberapa sering sebuah variabel dipakai, dan *menemukan* setiap nasihat yang bergantung padanya. Tesis yang mengusulkan VCIRS menaruh variabel tepat di pusat karena alasan ini: elemen dasar Variable-Centered Rule Structure adalah variabel yang dimasukkan pengguna, dan sistem mengelola variabel itu dengan cermat — nilainya, strukturnya, dan kemunculannya [5], [6].

Tabel 4.1 mendaftar variabel-variabel dalam nasihat Bu Wati, beserta pertanyaan di balik masing-masing. Tabel seperti ini layak ditulis sebelum menulis kode apa pun. Inilah kosakata basis pengetahuan, dan basis pengetahuan yang kosakatanya kacau — `likes` di satu tempat, `enjoys` di tempat lain, `hobby` di tempat ketiga — tidak dapat ditelusuri atau dihitung.

Tabel 4.1 Variabel-variabel dalam nasihat Bu Wati, beserta pertanyaan di balik masing-masing dan nilai-nilai lazimnya.

Variabel	Pertanyaan di baliknya	Nilai yang lazim
`likes`	Apa yang paling kamu sukai?	computers (komputer), living things (makhluk hidup), numbers (angka), people (orang), building things (membangun sesuatu), drawing (menggambar)
`wants`	Setelah lulus, kamu ingin segera bekerja atau belajar lebih lama?	work soon, study longer
`blood ok`	Apakah kamu tidak apa-apa melihat darah dan luka?	yes, no
`science score`	Rata-rata nilai rapor IPA-mu (0–100)?	sebuah angka
`math score`	Rata-rata nilai rapor matematikamu (0–100)?	sebuah angka
`patient with children`	Apakah kamu sabar menghadapi anak kecil?	yes, no
`drawing`	Seberapa baik kamu menggambar?	good, average
`program`	(konklusinya: program studi yang disarankan)	informatics, nursing, medicine, …

4.2.2 Nilai dan pembanding

Kebanyakan nilai berupa kata: `living things`, `work soon`, `yes`. Untuk nilai seperti ini, pembanding yang masuk akal hanyalah *sama dengan* (`=`) dan *tidak sama dengan* (`<>`). Sebagian nilai berupa angka, misalnya nilai rapor 88, dan untuknya masuk akal bertanya *paling sedikit* (`>=`), *lebih dari* (`>`), *paling banyak* (`<=`), dan *kurang dari* (`<`). VCIRS mengizinkan tepat enam operator ini, `=`, `<>`, `<`, `<=`, `>`, dan `>=`, dan membandingkan dua nilai sebagai angka hanya bila keduanya tampak seperti angka [5]. Jadi `science score >= 85` benar untuk siswa dengan nilai 88 dan salah

untuk siswa dengan nilai 80, sedangkan `likes >= computers` tidak pernah benar, karena kata-kata tidak punya urutan yang dapat dipercaya mesin.

Kondisi yang memakai angka adalah langkah kecil tetapi nyata dari aturan-aturan Bab 3. Bu Wati tidak akan pernah berkata "bila nilai IPA-nya 88"; ia berkata "bila nilai IPA-nya *kuat*", dan angka 85 adalah caranya membuat *kuat* menjadi tepat. Bab 9 melangkah lebih jauh dan membuat mesin memahami *kuat* itu sendiri, sebagai nilai fuzzy.

4.2.3 Node: rangkaian kondisi yang pendek

Node adalah satu butir nasihat: rangkaian pendek kondisi, dengan urutan tetap, bersama satu atau lebih konklusi [5]. Node untuk nasihat makhluk hidup cukup

```
likes = living things -> program = biology
```

dan node untuk keperawatan, yang merupakan *pengecualian* dari node biologi, hanya memuat yang baru:

```
blood ok = yes -> program = nursing
```

Inilah gagasan Ripple Down Rules dari Bab 3 dalam suasana baru. Node keperawatan hanya dicoba untuk siswa yang sudah cocok dengan node biologi, jadi ia tidak perlu menyebut `likes = living things` lagi. Dan pengecualiannya sendiri, node kedokteran, hanya membutuhkan `science score >= 85`. Bila ditulis sebagai daftar datar, ketiga butir nasihat itu mengulang `likes = living things` tiga kali dan `blood ok = yes` dua kali; bila disimpan sebagai node, setiap kondisi ditulis sekali, di tempat yang semestinya.

Urutan kondisi di dalam node itu penting. Itulah urutan mesin memeriksanya — dan menanyakannya — dan juga, seperti ditunjukkan Bab 6, sepotong kecil pengetahuan tersendiri: kondisi yang lebih belakang dalam sebuah node berdiri *lebih dekat ke konklusi*, dan VCIRS memberinya bobot lebih besar ketika menghitung pemakaian pengetahuan [5]. Posisi sebuah kondisi di dalam node-nya disebut **urutan variabel** (*variable order*).

Setiap node punya nama, **NodeID**, yang dibuat dari konklusi pertamanya dan sebuah nomor urut: node yang menyimpulkan `program = nursing` adalah `nursing#1`; node kedua dengan konklusi yang sama menjadi `nursing#2` [5]. Nama-nama ini bukan hiasan. Riwayat basis pengetahuan, penjelasan, dan laporan semuanya merujuk node dengan nama ini, sehingga seorang guru BK dapat berkata "`nursing#1` tepat untuk Laras" dan semua orang tahu nasihat mana yang ia maksud.

4.2.4 Rule: rangkaian node

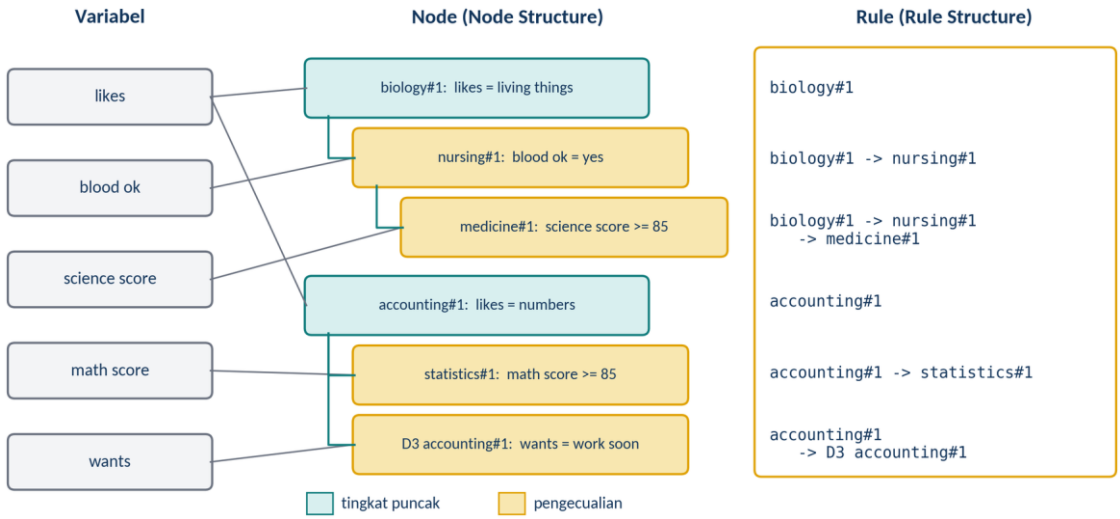
Rule dalam VCIRS adalah rangkaian node, dari tingkat atas ke bawah: `biology#1 -> nursing#1 -> medicine#1` adalah satu rule [5], [6]. Bacaannya: *bila siswa menyukai makhluk hidup, dan ia tidak apa-apa melihat darah, dan nilai IPA-nya paling sedikit 85, maka program yang layak dilihat adalah kedokteran.* Konklusi sebuah rule adalah konklusi node terakhirnya, dan rule itu juga dinamai menurut node terakhirnya, jadi rule ini adalah `medicine#1`.

Setiap node adalah node terakhir dari tepat satu rule — jalur yang menuju kepadanya — sehingga basis pengetahuan dengan tiga belas node punya tiga belas rule. Tetapi sebuah node dapat muncul dalam *banyak* rule: `biology#1` adalah node pertama dari `biology#1`, dari `nursing#1`, dan dari `medicine#1`. Tesis menyebut hal ini **berbagi** (*sharing*): node yang dipakai bersama oleh banyak rule, seperti variabel yang dipakai bersama oleh banyak node, dipakai di banyak tempat dan karena itu kemungkinan besar penting [5]. Ingatlah kata ini; tabel-tabel di Subbab 4.8 menghitungnya.

Node-node yang dilewati di sepanjang sebuah rule juga membawa konklusi: sebelum `medicine#1` dijalankan, `biology#1` dan `nursing#1` sudah dijalankan dan menyimpulkan *biology* dan *nursing*. Mesin menyimpan ini sebagai **konklusi antara** (*intermediate conclusions*) [5]. Untuk diagnosis dokter, konklusi antara mungkin hanya mengganggu; untuk siswa yang memilih program studi, justru inilah pilihan cadangan yang, menurut **Gambar 4.1**, sangat ia butuhkan.

4.2.5 Variable-Centered Rule Structure

Satukan potongan-potongan itu, dan Anda memperoleh struktur yang memberi VCIRS namanya. **Gambar 4.2** memperlihatkan sebagian strukturnya untuk nasihat Bu Wati.

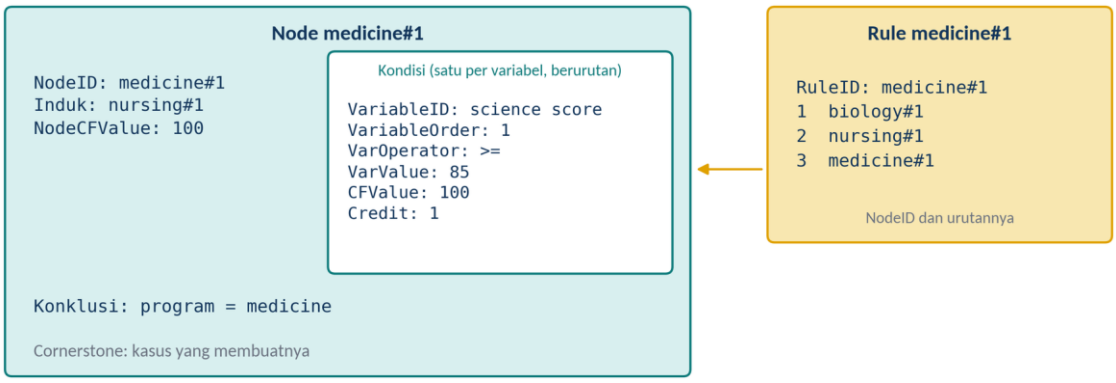


Gambar 4.2 Sebagian Variable-Centered Rule Structure nasihat Bu Wati. Setiap variabel disimpan sekali dan dihubungkan ke node-node yang memakainya; setiap node hanya memuat kondisinya sendiri; setiap rule adalah rangkaian node dari tingkat atas ke bawah.

Struktur ini punya tiga bagian [5], [7]:

- **tabel Variabel** menyimpan setiap variabel sekali, berapa pun banyaknya node yang memakainya, bersama pertanyaannya dan nilai-nilai lazimnya, dan mengingat di mana ia dipakai dan pada posisi keberapa;
- **Node Structure** menyimpan setiap node: NodeID-nya, induknya (tidak ada untuk node tingkat atas), kondisi-kondisinya menurut urutan, konklusinya, kepastiannya (NodeCFValue, 100 kecuali pakarnya kurang yakin), dan **kasus cornerstone** yang membuatnya;
- **Rule Structure** menyimpan setiap rule sebagai daftar NodeID, masing-masing dengan posisinya di dalam rule.

Gambar 4.3 memperlihatkan medan-medan (*fields*) sebuah node dan sebuah rule, sebagaimana didefinisikan tesis. Dua medan di sana belum dijelaskan. **Nilai CF** setiap kondisi menyatakan seberapa pasti kondisi itu, pada skala 0 sampai 100 yang dipakai sistem berbasis aturan klasik; semua kondisi Bu Wati pasti, jadi semuanya 100. **Credit** sebuah kondisi menghitung berapa kali sebuah kasus yang datang kemudian *menyetujui* kondisi itu; nilainya mulai dari 1, dan Bab 6 menunjukkan bagaimana ia bertambah dan untuk apa ia dipakai.



Gambar 4.3 Medan-medan sebuah node dan sebuah rule dalam Variable-Centered Rule Structure, diisi untuk `medicine#1`. Rule mendaftar node-node-nya menurut urutan; setiap node mendaftar kondisi-kondisinya menurut urutan.

Mengapa bersusah payah begini, padahal Ripple Down Rules di Bab 3 sudah cukup baik sebagai pohon? Karena pohon hanya menjawab satu jenis pertanyaan: *dengan fakta-fakta ini, apa jawabannya?* Tiga tabel menjawab jauh lebih banyak. *Nasihat mana yang bergantung pada nilai IPA?* Cari `science score` di tabel Variabel. *Nasihat mana yang ikut berubah bila node biologi keliru?* Cari rule-rule yang berbagi `biology#1`. *Pertanyaan mana yang sebaiknya diajukan lebih dulu?* Hitung seberapa banyak tiap variabel dipakai. *Dapatkah pengetahuan yang sama dibaca sebagai aturan JIKA-MAKA biasa, untuk penalaran maju dan mundur?* Dapat: setiap rule adalah jalur node, dan kondisi-kondisinya dapat dikumpulkan menjadi satu aturan JIKA-MAKA, seperti yang dilakukan Bab 8. Tesis merangkumnya sebagai kemudahan membangun pengetahuan, inferensi pengetahuan yang kuat, dan perbaikan unjuk kerja sistem yang terus berkembang, sekaligus [6], [5], dan sebuah penasihat pemilihan jurusan fuzzy yang dibangun di atas VCIRS, dengan jurusan-jurusan di Chinese University of Hong Kong sebagai studi kasusnya, memakai struktur yang persis sama [8].

> **DEFINISI.** *Variable-Centered Rule Structure* menyimpan basis pengetahuan sebagai tiga tabel yang saling terhubung: *variabel* (masing-masing disimpan sekali, bersama tempat-tempat ia dipakai), *node* (rangkaian kondisi pendek yang berurutan beserta konklusinya, masing-masing pengecualian dari induknya), dan *rule* (rangkaian node dari tingkat atas ke bawah, dinamai menurut node terakhirnya).

4.2.6 Bagaimana sebuah saran ditemukan

Menjawab seorang siswa berlangsung seperti di Bab 3, dengan satu perbaikan. Mesin mencoba node-node tingkat atas secara berurutan; untuk tiap node ia memeriksa kondisi-kondisinya secara berurutan dan menanyakan kepada siswa setiap nilai yang belum diketahuinya. Node yang kondisi pertamanya salah langsung ditinggalkan, tanpa menanyakan sisanya. Ketika sebuah node **dijalankan** (*fires*) — semua kondisinya terpenuhi — mesin turun ke pengecualian-pengecualiannya dan melakukan hal yang sama di sana, sedalam mungkin. Sarannya adalah konklusi node terakhir yang dijalankan, dan konklusi node-node di atasnya adalah konklusi antara [5].

Perbaikannya: sebuah nilai, sekali diketahui, tidak pernah ditanyakan lagi. Nilai itu tetap berada di **memori kerja** (*working memory*) konsultasi, yaitu daftar fakta yang sudah diketahui, dan setiap node yang membutuhkannya membacanya dari sana. Untuk Laras, node pertama menanyakan `likes`; ia menjawab *living things*; node informatika langsung gagal; node biologi dijalankan tanpa pertanyaan baru; pengecualiannya, keperawatan, menanyakan `blood ok`; dan

seterusnya. Laras menjawab tiga pertanyaan, bukan delapan. Itulah yang dilakukan guru BK yang baik, dan itulah yang tidak pernah dilakukan formulir yang memuat semua pertanyaan.

4.3 Langkah 1: Nasihat sebagai Daftar Datar

Sebelum membangun strukturnya, mari kita lihat pengetahuan itu seperti kebanyakan program menyimpannya: sebagai daftar datar aturan lengkap, masing-masing dengan semua kondisinya. **Kode 4.1** menyimpan tiga belas butir nasihat Bu Wati dengan cara itu, lalu melakukan hal pertama yang dilakukan VCIRS terhadap pengetahuan apa pun: **mengindeksnya per variabel**.

Kode 4.1 `FlatAdvice.java`: nasihat Bu Wati sebagai daftar aturan datar, diindeks per variabel.

```
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

/** Saran Bu Wati sebagai daftar aturan datar, lalu diindeks per variabel. */
public class FlatAdvice {
    record Rule(List<String> conditions, String program) { }

    static final List<Rule> RULES = List.of(
            new Rule(List.of("likes = computers"), "informatics"),
            new Rule(List.of("likes = computers", "wants = work soon"),
                    "D3 informatics"),
            new Rule(List.of("likes = living things"), "biology"),
            new Rule(List.of("likes = living things", "blood ok = yes"),
                    "nursing"),
            new Rule(List.of("likes = living things", "blood ok = yes",
                    "science score >= 85"), "medicine"),
            new Rule(List.of("likes = numbers"), "accounting"),
            new Rule(List.of("likes = numbers", "math score >= 85"),
                    "statistics"),
            new Rule(List.of("likes = numbers", "wants = work soon"),
                    "D3 accounting"),
            new Rule(List.of("likes = people"), "communication science"),
            new Rule(List.of("likes = people", "patient with children = yes"),
                    "primary teaching"),
            new Rule(List.of("likes = building things"), "civil engineering"),
            new Rule(List.of("likes = building things", "drawing = good",
                    "math score >= 75"), "architecture"),
            new Rule(List.of("likes = drawing"), "visual communication design"));

    static final String[] OPERATORS = {" <= ", " >= ", " <> ", " = ", " < ",
        " > "};

    /** "math score >= 85" -> {"math score", ">=", "85"} */
    static String[] split(String condition) {
        for (String op : OPERATORS) {
            int at = condition.indexOf(op);
            if (at > 0) {
                return new String[] {condition.substring(0, at), op.strip(),
                    condition.substring(at + op.length())};
            }
        }
        throw new IllegalArgumentException("no operator in " + condition);
    }
```

Kode 4.1 (Lanjutan) *FlatAdvice.java: nasihat Bu Wati sebagai daftar aturan datar, diindeks per variabel.*

```
    static String times(int n) {
        return n + (n == 1 ? " time" : " times");
    }

    public static void main(String[] args) {
        Map<String, List<String>> values = new LinkedHashMap<>();
        Map<String, List<Integer>> usedIn = new LinkedHashMap<>();
        Map<String, Integer> count = new LinkedHashMap<>();
        int total = 0;
        for (int i = 0; i < RULES.size(); i++) {
            for (String c : RULES.get(i).conditions()) {
                String[] p = split(c);
                String shown = p[1].equals("=") ? p[2] : p[1] + " " + p[2];
                values.putIfAbsent(p[0], new ArrayList<>());
                usedIn.putIfAbsent(p[0], new ArrayList<>());
                if (!values.get(p[0]).contains(shown)) {
                    values.get(p[0]).add(shown);
                }
                usedIn.get(p[0]).add(i + 1);
                count.merge(c, 1, Integer::sum);
                total++;
            }
        }
        System.out.println(RULES.size() + " rules, " + total + " conditions, "
                + values.size() + " variables.");
        System.out.println();
        for (String var : values.keySet()) {
            List<String> numbers = new ArrayList<>();
            for (int n : usedIn.get(var)) {
                numbers.add(String.valueOf(n));
            }
            System.out.println(var + " (used " + times(numbers.size()) + ")");
            System.out.println("  values: " + String.join(", ", values.get(var)));
            System.out.println("  rules:  " + String.join(", ", numbers));
        }
        System.out.println();
        System.out.println("Conditions written more than once:");
        for (Map.Entry<String, Integer> e : count.entrySet()) {
            if (e.getValue() > 1) {
                System.out.println("  " + e.getKey() + ": " + times(e.getValue()));
            }
        }
    }
}
```

Sebuah `record Rule` menyimpan kondisi-kondisi (sebagai teks, seperti `"math score >= 85"`) dan programnya. `split` memotong sebuah kondisi menjadi tiga bagian: variabel, operator, dan nilai. Ia mencari setiap operator *dengan spasi di kedua sisinya*, sehingga `" >= "` ditemukan dalam `"math score >= 85"` dan `" = "` tidak disangka sebagai bagian dari `>=`. `main` lalu menelusuri setiap kondisi dari setiap aturan dan mengisi tiga peta (*map*): nilai-nilai yang terlihat untuk tiap variabel, aturan-aturan yang memakainya, dan berapa kali tiap kondisi utuh ditulis. `LinkedHashMap` adalah peta yang mengingat urutan kunci-kuncinya pertama kali dimasukkan, sehingga variabel-variabel keluar sesuai urutan kemunculan pertamanya dalam nasihat. **Kode 4.2** adalah program yang sama dalam Python.

Kode 4.2 `flat_advice.py`: daftar datar dan indeks yang sama dalam Python.

```
"""Saran Bu Wati sebagai daftar aturan datar, lalu diindeks per variabel."""

RULES = [
    (["likes = computers"], "informatics"),
    (["likes = computers", "wants = work soon"], "D3 informatics"),
    (["likes = living things"], "biology"),
    (["likes = living things", "blood ok = yes"], "nursing"),
    (["likes = living things", "blood ok = yes", "science score >= 85"],
     "medicine"),
    (["likes = numbers"], "accounting"),
    (["likes = numbers", "math score >= 85"], "statistics"),
    (["likes = numbers", "wants = work soon"], "D3 accounting"),
    (["likes = people"], "communication science"),
    (["likes = people", "patient with children = yes"],
     "primary teaching"),
    (["likes = building things"], "civil engineering"),
    (["likes = building things", "drawing = good", "math score >= 75"],
     "architecture"),
    (["likes = drawing"], "visual communication design"),
]

OPERATORS = [" <= ", " >= ", " <> ", " = ", " < ", " > "]


def split(condition):
    """'math score >= 85' -> ['math score', '>=', '85']"""
    for op in OPERATORS:
        at = condition.find(op)
        if at > 0:
            return [condition[:at], op.strip(), condition[at + len(op):]]
    raise ValueError("no operator in " + condition)


def times(n):
    return str(n) + (" time" if n == 1 else " times")
```

Kode 4.2 (Lanjutan) *flat_advice.py: daftar datar dan indeks yang sama dalam Python.*

```
def main():
    index = {}                # variabel -> {"values": [...], "rules": [...]}
    count = {}                # kondisi -> berapa aturan menulisnya
    for number, (conditions, program) in enumerate(RULES, start=1):
        for c in conditions:
            var, op, value = split(c)
            entry = index.setdefault(var, {"values": [], "rules": []})
            shown = value if op == "=" else op + " " + value
            if shown not in entry["values"]:
                entry["values"].append(shown)
            entry["rules"].append(number)
            count[c] = count.get(c, 0) + 1

    total = sum(len(conditions) for conditions, _ in RULES)
    print(str(len(RULES)) + " rules, " + str(total) + " conditions, "
          + str(len(index)) + " variables.")
    print()
    for var, entry in index.items():
        print(var + " (used " + times(len(entry["rules"])) + ")")
        print("  values: " + ", ".join(entry["values"]))
        print("  rules:  " + ", ".join(str(n) for n in entry["rules"]))
    print()
    print("Conditions written more than once:")
    for c, n in count.items():
        if n > 1:
            print("  " + c + ": " + times(n))


if __name__ == "__main__":
    main()
```

Versi Python memakai kamus berisi kamus-kamus kecil untuk indeksnya, dan `setdefault`, yang mengembalikan entri sebuah variabel dan membuatnya lebih dulu bila belum ada. Baris-baris terakhir, `if __name__ == "__main__": main()`, menjalankan `main` hanya bila berkas dijalankan sebagai program, bukan ketika berkas lain mengimpor `split` darinya (pengujian di Subbab 4.9 melakukannya). Jalankan `java -cp out FlatAdvice` (setelah mengompilasi, lihat Subbab 4.7) atau `python flat_advice.py`; keduanya mencetak:

```
13 rules, 22 conditions, 7 variables.

likes (used 13 times)
  values: computers, living things, numbers, people, building things, drawing
  rules:  1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
wants (used 2 times)
  values: work soon
  rules:  2, 8
blood ok (used 2 times)
  values: yes
  rules:  4, 5
science score (used 1 time)
  values: >= 85
  rules:  5
math score (used 2 times)
  values: >= 85, >= 75
  rules:  7, 12
patient with children (used 1 time)
  values: yes
  rules:  10
drawing (used 1 time)
  values: good
  rules:  12

Conditions written more than once:
  likes = computers: 2 times
  wants = work soon: 2 times
  likes = living things: 3 times
  blood ok = yes: 2 times
  likes = numbers: 3 times
  likes = people: 2 times
  likes = building things: 2 times
```

Bacalah indeks ini seperti seorang guru BK membacanya. Baris pertama menyatakan bahwa tiga belas butir nasihat hanya membutuhkan tujuh variabel: pengetahuannya jauh lebih kecil daripada kelihatannya. Variabel `likes` muncul di setiap aturan, yang menandakan bahwa ialah yang pertama ditanyakan dan tulang punggung seluruh basis pengetahuan. `math score` dipakai dua kali, dengan dua ambang yang berbeda — 85 untuk statistika, 75 untuk arsitektur — yang mungkin ingin dilihat seorang guru BK berdampingan. Dan blok terakhir memperlihatkan harga daftar datar: `likes = living things` ditulis tiga kali, `likes = numbers` tiga kali, `blood ok = yes` dua kali. Setiap pengulangan adalah tempat pengetahuan dapat bergeser saling menjauh. Bila suatu hari Bu Wati memutuskan bahwa "living things" sebaiknya menjadi "living things and nature", ia harus menemukan dan mengubahnya di tiga aturan, dan bila satu terlewat, basis pengetahuannya diam-diam tidak sepakat dengan dirinya sendiri.

Perhatikan satu baris yang *bukan* pengulangan dalam arti yang sama: `wants = work soon` muncul dua kali, di aturan untuk D3 informatika dan di aturan untuk D3 akuntansi. Keduanya dua butir nasihat berbeda yang kebetulan mengajukan pertanyaan yang sama, dan keduanya akan tetap menjadi dua node dalam struktur. VCIRS menyimpan *variabel* `wants` sekali dan menghubungkannya ke kedua node; tabel Variabel pada langkah berikutnya memperlihatkannya.

4.4 Langkah 2: Variabel, Node, dan Rule dengan Tangan

Kode 4.3 menyimpan nasihat yang sama dengan cara VCIRS. Setiap node hanya memuat kondisinya sendiri dan menyebut induknya; program membangun tabel Variabel, Node Structure, dan Rule Structure sambil berjalan, lalu memakainya untuk menjawab empat siswa.

Kode 4.3 `MiniStructure.java`: tabel Variabel, Node Structure, dan Rule Structure, dibangun dengan tangan.

```
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

/**
 * Saran Bu Wati disimpan dengan cara VCIRS: tabel Variabel, Node Structure
 * dan Rule Structure, dibuat dengan tangan. Sebuah node hanya memuat
 * kondisinya sendiri; kondisi induknya tidak ditulis ulang.
 */
public class MiniStructure {
    static final String[] OPERATORS = {" <= ", " >= ", " <> ", " = ", " < ",
        " > "};

    /** "math score >= 85" -> {"math score", ">=", "85"} */
    static String[] split(String condition) {
        for (String op : OPERATORS) {
            int at = condition.indexOf(op);
            if (at > 0) {
                return new String[] {condition.substring(0, at), op.strip(),
                    condition.substring(at + op.length())};
            }
        }
        throw new IllegalArgumentException("no operator in " + condition);
    }

    static boolean holds(String condition, Map<String, String> facts) {
        String[] p = split(condition);
        String fact = facts.get(p[0]);
        if (fact == null) {
            return false;                    // fakta tak dikenal tak pernah cocok
        }
        switch (p[1]) {
            case "=": return fact.equals(p[2]);
            case "<>": return !fact.equals(p[2]);
            default: break;
        }
        double a = Double.parseDouble(fact);  // yang lain membandingkan angka
        double b = Double.parseDouble(p[2]);
        switch (p[1]) {
            case "<": return a < b;
            case "<=": return a <= b;
            case ">": return a > b;
            default: return a >= b;
        }
    }
```

Kode 4.3 (Lanjutan) *MiniStructure.java: tabel Variabel, Node Structure, dan Rule Structure, dibangun dengan tangan.*

```
    record Node(String id, String parent, List<String> conditions,
                String program) { }

    static class Structure {
        final Map<String, Node> nodes = new LinkedHashMap<>();   // Node Str.
        final Map<String, List<String>> rules = new LinkedHashMap<>();
        final Map<String, List<String>> variables = new LinkedHashMap<>();

        String add(String parent, List<String> conditions, String program) {
            int n = 1;
            while (nodes.containsKey(program + "#" + n)) {
                n++;
            }
            Node node = new Node(program + "#" + n, parent, conditions,
                    program);
            nodes.put(node.id(), node);
            List<String> path = new ArrayList<>(List.of(node.id()));
            String p = parent;
            while (p != null) {              // rule adalah jalur dari puncak
                path.add(0, p);
                p = nodes.get(p).parent();
            }
            rules.put(node.id(), path);      // dinamai menurut node terakhirnya
            for (int i = 0; i < conditions.size(); i++) {
                String var = split(conditions.get(i))[0];
                variables.putIfAbsent(var, new ArrayList<>());
                variables.get(var).add(node.id() + ", position " + (i + 1));
            }
            return node.id();
        }

        List<Node> children(String parent) {
            List<Node> out = new ArrayList<>();
            for (Node n : nodes.values()) {
                if (parent == null ? n.parent() == null
                        : parent.equals(n.parent())) {
                    out.add(n);
                }
            }
            return out;
        }
```

Kode 4.3 (Lanjutan) *MiniStructure.java: tabel Variabel, Node Structure, dan Rule Structure, dibangun dengan tangan.*

```
        /**
         * Jalankan node tingkat puncak pertama yang cocok, lalu pengecualian
         * pertamanya yang cocok, dan seterusnya ke bawah. Node terakhir yang
         * dijalankan memberi jawabannya.
         */
        List<String> answer(Map<String, String> facts) {
            List<String> path = new ArrayList<>();
            List<Node> level = children(null);
            while (true) {
                Node fired = null;
                for (Node node : level) {
                    boolean all = true;
                    for (String c : node.conditions()) {
                        all = all && holds(c, facts);
                    }
                    if (all) {
                        fired = node;
                        break;
                    }
                }
                if (fired == null) {
                    return path;
                }
                path.add(fired.id());
                level = children(fired.id());
            }
        }
    }

    /** Tiga belas saran Bu Wati; pengecualian menyebut induknya. */
    static Structure buWati() {
        Structure s = new Structure();
        String inf = s.add(null, List.of("likes = computers"), "informatics");
        s.add(inf, List.of("wants = work soon"), "D3 informatics");
        String bio = s.add(null, List.of("likes = living things"), "biology");
        String nur = s.add(bio, List.of("blood ok = yes"), "nursing");
        s.add(nur, List.of("science score >= 85"), "medicine");
        String acc = s.add(null, List.of("likes = numbers"), "accounting");
        s.add(acc, List.of("math score >= 85"), "statistics");
        s.add(acc, List.of("wants = work soon"), "D3 accounting");
        String com = s.add(null, List.of("likes = people"),
                "communication science");
        s.add(com, List.of("patient with children = yes"), "primary teaching");
        String civ = s.add(null, List.of("likes = building things"),
                "civil engineering");
        s.add(civ, List.of("drawing = good", "math score >= 75"),
```

Kode 4.3 (Lanjutan) *MiniStructure.java: tabel Variabel, Node Structure, dan Rule Structure, dibangun dengan tangan.*

```
                "architecture");
        s.add(null, List.of("likes = drawing"), "visual communication design");
        return s;
    }

    public static void main(String[] args) {
        Structure s = buWati();
        int stored = 0;
        for (Node n : s.nodes.values()) {
            stored += n.conditions().size();
        }
        System.out.println("Node Structure: " + s.nodes.size() + " nodes, "
                + stored + " conditions");
        for (Node n : s.nodes.values()) {
            System.out.println("  " + n.id() + (n.parent() != null
                    ? "  (exception of " + n.parent() + ")" : "  (top level)"));
            System.out.println("      " + String.join(", ", n.conditions())
                    + " -> program = " + n.program());
        }
        System.out.println("Rule Structure: " + s.rules.size() + " rules");
        for (Map.Entry<String, List<String>> r : s.rules.entrySet()) {
            System.out.println("  " + r.getKey() + ": "
                    + String.join(" -> ", r.getValue()));
        }
        System.out.println("Variable table: " + s.variables.size()
                + " variables");
        for (Map.Entry<String, List<String>> v : s.variables.entrySet()) {
            System.out.println("  " + v.getKey() + ", used in "
                    + v.getValue().size() + " node(s):");
            for (String where : v.getValue()) {
                System.out.println("      " + where);
            }
        }

        System.out.println();
        String[] names = {"Laras", "Bima", "Sari", "Tono"};
        List<Map<String, String>> students = List.of(
                Map.of("likes", "living things", "blood ok", "yes",
                        "science score", "88"),
                Map.of("likes", "computers", "wants", "work soon"),
                Map.of("likes", "numbers", "math score", "80",
                        "wants", "study longer"),
                Map.of("likes", "music"));
        for (int i = 0; i < names.length; i++) {
            List<String> path = s.answer(students.get(i));
            if (path.isEmpty()) {
```

Kode 4.3 (Lanjutan) *MiniStructure.java: tabel Variabel, Node Structure, dan Rule Structure, dibangun dengan tangan.*

```
                System.out.println(names[i] + ": no advice yet");
            } else {
                System.out.println(names[i] + ": "
                        + s.nodes.get(path.get(path.size() - 1)).program()
                        + "  (path: " + String.join(" -> ", path) + ")");
            }
        }
    }
}
```

Bacalah dari atas.

Membandingkan. `split` sama seperti di Langkah 1. `holds` memutuskan apakah satu kondisi benar untuk fakta-fakta seorang siswa. Fakta yang tidak diberikan siswa tidak pernah cocok. Kata dibandingkan dengan `=` dan `<>`; untuk empat operator urutan, kedua sisi diubah menjadi angka dengan `Double.parseDouble` dan dibandingkan sebagai angka.

Tiga tabel. Sebuah `Node` punya ID, induk (`null` untuk node tingkat atas), kondisi-kondisinya, dan programnya. `Structure` menyimpan ketiga tabel sebagai peta: `nodes` (Node Structure), `rules` (setiap ID rule dengan daftar ID node-nya, Rule Structure), dan `variables` (setiap variabel dengan tempat-tempat ia dipakai, tabel Variabel).

Menambah node. `add` mengerjakan, dalam dua puluh baris, apa yang dijelaskan tesis untuk Node Structure dan Rule Structure [5]. Ia membuat NodeID dari nama program dan nomor urut bebas pertama (`medicine#1`, dan `medicine#2` bila yang pertama sudah terpakai). Ia menyimpan node itu. Ia membangun rule milik node itu dengan berjalan naik dari node baru melewati induk-induknya sampai ke tingkat atas, menaruh masing-masing di depan jalur, lalu menyimpan jalur itu dengan ID node itu sendiri — rule dinamai menurut node terakhirnya. Terakhir ia mencatat, untuk tiap kondisi, variabelnya, node-nya, dan posisinya, yaitu urutan variabel.

Menjawab. `answer` mulai dengan node-node tingkat atas (anak-anak `null`), menjalankan yang pertama yang semua kondisinya terpenuhi, lalu mencari di antara anak-anak node *itu*, dan seterusnya ke bawah, sambil mengumpulkan jalurnya. Ketika tidak ada node pada tingkat itu yang dijalankan, jalur sejauh itu adalah jawabannya, dan node terakhir di jalur memberi programnya. Jalur kosong berarti *belum ada saran*.

Pengetahuan Bu Wati. `buWati` menambahkan tiga belas node. Lihatlah betapa sedikit yang dikatakan tiap pengecualian: `add(nur, List.of("science score >= 85"), "medicine")` menggantung kedokteran di bawah keperawatan dengan satu kondisi. `main` mencetak ketiga tabel dan menjawab empat siswa. **Kode 4.4** adalah versi Python-nya.

Kode 4.4 `mini_structure.py`: tiga tabel yang sama dalam Python.

```
"""Saran Bu Wati disimpan dengan cara VCIRS: tabel Variabel, Node Structure
dan Rule Structure, dibuat dengan tangan. Sebuah node hanya memuat
kondisinya sendiri; kondisi induknya tidak ditulis ulang."""

OPERATORS = [" <= ", " >= ", " <> ", " = ", " < ", " > "]


def split(condition):
    """'math score >= 85' -> ['math score', '>=', '85']"""
    for op in OPERATORS:
        at = condition.find(op)
        if at > 0:
            return [condition[:at], op.strip(), condition[at + len(op):]]
    raise ValueError("no operator in " + condition)


def holds(condition, facts):
    var, op, value = split(condition)
    if var not in facts:
        return False                    # fakta tak dikenal tak pernah cocok
    fact = facts[var]
    if op == "=":
        return fact == value
    if op == "<>":
        return fact != value
    a, b = float(fact), float(value)    # operator lain membandingkan angka
    return {"<": a < b, "<=": a <= b, ">": a > b, ">=": a >= b}[op]


class Node:
    def __init__(self, node_id, parent, conditions, program):
        self.id = node_id
        self.parent = parent            # None untuk node tingkat puncak
        self.conditions = conditions
        self.program = program


class Structure:
    def __init__(self):
        self.nodes = {}       # node id -> Node          (Node Structure)
        self.rules = {}       # rule id -> [node ids]    (Rule Structure)
        self.variables = {}   # variabel -> [(id node, posisi)]
```

Kode 4.4 (Lanjutan) *mini_structure.py: tiga tabel yang sama dalam Python.*

```
    def add(self, parent, conditions, program):
        n = 1
        while program + "#" + str(n) in self.nodes:
            n += 1
        node = Node(program + "#" + str(n), parent, conditions, program)
        self.nodes[node.id] = node
        path, p = [node.id], parent
        while p is not None:             # rule adalah jalur dari puncak
            path.insert(0, p)
            p = self.nodes[p].parent
        self.rules[node.id] = path       # dinamai menurut node terakhirnya
        for position, c in enumerate(conditions, start=1):
            var = split(c)[0]
            self.variables.setdefault(var, []).append((node.id, position))
        return node.id

    def children(self, parent):
        return [n for n in self.nodes.values() if n.parent == parent]

    def answer(self, facts):
        """Jalankan node tingkat puncak pertama yang cocok, lalu pengecualian
        pertamanya yang cocok, dan seterusnya ke bawah. Node terakhir yang
        dijalankan memberi jawabannya."""
        path, level = [], self.children(None)
        while True:
            fired = None
            for node in level:
                if all(holds(c, facts) for c in node.conditions):
                    fired = node
                    break
            if fired is None:
                return path
            path.append(fired.id)
            level = self.children(fired.id)
```

Kode 4.4 (Lanjutan) *mini_structure.py: tiga tabel yang sama dalam Python.*

```
def bu_wati():
    """Tiga belas saran Bu Wati; pengecualian menyebut induknya."""
    s = Structure()
    inf = s.add(None, ["likes = computers"], "informatics")
    s.add(inf, ["wants = work soon"], "D3 informatics")
    bio = s.add(None, ["likes = living things"], "biology")
    nur = s.add(bio, ["blood ok = yes"], "nursing")
    s.add(nur, ["science score >= 85"], "medicine")
    acc = s.add(None, ["likes = numbers"], "accounting")
    s.add(acc, ["math score >= 85"], "statistics")
    s.add(acc, ["wants = work soon"], "D3 accounting")
    com = s.add(None, ["likes = people"], "communication science")
    s.add(com, ["patient with children = yes"], "primary teaching")
    civ = s.add(None, ["likes = building things"], "civil engineering")
    s.add(civ, ["drawing = good", "math score >= 75"], "architecture")
    s.add(None, ["likes = drawing"], "visual communication design")
    return s


def main():
    s = bu_wati()
    stored = sum(len(n.conditions) for n in s.nodes.values())
    print("Node Structure: " + str(len(s.nodes)) + " nodes, " + str(stored)
          + " conditions")
    for n in s.nodes.values():
        print("  " + n.id + ("  (exception of " + n.parent + ")" if n.parent
                             else "  (top level)"))
        print("      " + ", ".join(n.conditions) + " -> program = " + n.program)
    print("Rule Structure: " + str(len(s.rules)) + " rules")
    for rule_id, path in s.rules.items():
        print("  " + rule_id + ": " + " -> ".join(path))
    print("Variable table: " + str(len(s.variables)) + " variables")
    for var, used in s.variables.items():
        print("  " + var + ", used in " + str(len(used)) + " node(s):")
        for nid, pos in used:
            print("      " + nid + ", position " + str(pos))
```

Kode 4.4 (Lanjutan) *mini_structure.py: tiga tabel yang sama dalam Python.*

```
    print()
    students = [
        ("Laras", {"likes": "living things", "blood ok": "yes",
                   "science score": "88"}),
        ("Bima", {"likes": "computers", "wants": "work soon"}),
        ("Sari", {"likes": "numbers", "math score": "80",
                  "wants": "study longer"}),
        ("Tono", {"likes": "music"}),
    ]
    for name, facts in students:
        path = s.answer(facts)
        if not path:
            print(name + ": no advice yet")
        else:
            print(name + ": " + s.nodes[path[-1]].program + "  (path: "
                  + " -> ".join(path) + ")")


if __name__ == "__main__":
    main()
```

Dalam Python, `holds` memilih perbandingan yang tepat dari sebuah kamus kecil berisi empat hasil, dan `answer` memakai `all(...)` untuk memeriksa semua kondisi dalam satu ekspresi. Kedua program mencetak:

```
Node Structure: 13 nodes, 14 conditions
  informatics#1  (top level)
      likes = computers -> program = informatics
  D3 informatics#1  (exception of informatics#1)
      wants = work soon -> program = D3 informatics
  biology#1  (top level)
      likes = living things -> program = biology
  nursing#1  (exception of biology#1)
      blood ok = yes -> program = nursing
  medicine#1  (exception of nursing#1)
      science score >= 85 -> program = medicine
  accounting#1  (top level)
      likes = numbers -> program = accounting
  statistics#1  (exception of accounting#1)
      math score >= 85 -> program = statistics
  D3 accounting#1  (exception of accounting#1)
      wants = work soon -> program = D3 accounting
  communication science#1  (top level)
      likes = people -> program = communication science
  primary teaching#1  (exception of communication science#1)
      patient with children = yes -> program = primary teaching
  civil engineering#1  (top level)
      likes = building things -> program = civil engineering
  architecture#1  (exception of civil engineering#1)
      drawing = good, math score >= 75 -> program = architecture
  visual communication design#1  (top level)
      likes = drawing -> program = visual communication design
Rule Structure: 13 rules
  informatics#1: informatics#1
  D3 informatics#1: informatics#1 -> D3 informatics#1
  biology#1: biology#1
  nursing#1: biology#1 -> nursing#1
  medicine#1: biology#1 -> nursing#1 -> medicine#1
  accounting#1: accounting#1
  statistics#1: accounting#1 -> statistics#1
  D3 accounting#1: accounting#1 -> D3 accounting#1
  communication science#1: communication science#1
  primary teaching#1: communication science#1 -> primary teaching#1
  civil engineering#1: civil engineering#1
  architecture#1: civil engineering#1 -> architecture#1
  visual communication design#1: visual communication design#1
Variable table: 7 variables
  likes, used in 6 node(s):
      informatics#1, position 1
      biology#1, position 1
      accounting#1, position 1
      communication science#1, position 1
      civil engineering#1, position 1
      visual communication design#1, position 1
  wants, used in 2 node(s):
      D3 informatics#1, position 1
      D3 accounting#1, position 1
```

```
  blood ok, used in 1 node(s):
      nursing#1, position 1
  science score, used in 1 node(s):
      medicine#1, position 1
  math score, used in 2 node(s):
      statistics#1, position 1
      architecture#1, position 2
  patient with children, used in 1 node(s):
      primary teaching#1, position 1
  drawing, used in 1 node(s):
      architecture#1, position 1

Laras: medicine  (path: biology#1 -> nursing#1 -> medicine#1)
Bima: D3 informatics  (path: informatics#1 -> D3 informatics#1)
Sari: accounting  (path: accounting#1)
Tono: no advice yet
```

Bandingkan baris pertama dengan Langkah 1: tiga belas butir nasihat yang sama sekarang hanya membutuhkan **14 kondisi, bukan 22**. Tidak ada yang hilang; delapan kondisi yang tidak tertulis adalah kondisi yang diwarisi sebuah pengecualian dari node-node di atasnya. Rule Structure memperlihatkan sisi lain dari mata uang yang sama: rule `medicine#1` adalah rangkaian `biology#1 -> nursing#1 -> medicine#1`, dan menelusurinya mengumpulkan tepat tiga kondisi yang harus ditulis lengkap oleh aturan datar.

Tabel Variabel kini terbaca seperti indeks sebuah buku. `likes` dipakai di enam node, selalu di posisi 1 — ialah pertanyaan pertama setiap node tingkat atas. `wants` dipakai di dua node, D3 informatika dan D3 akuntansi: satu variabel, dua butir nasihat. `math score` juga dipakai di dua node, di posisi 1 `statistics#1` dan di posisi 2 `architecture#1`, sesudah `drawing = good`. Bila Bu Wati ingin tahu setiap nasihat yang bergantung pada nilai matematika, jawabannya ada dalam satu baris.

Dan keempat siswa mendapat jawaban yang masuk akal. **Laras** menyukai makhluk hidup, tidak apa-apa melihat darah, dan nilai IPA-nya 88: jalurnya `biology#1 -> nursing#1 -> medicine#1`, dan sarannya kedokteran — dengan biologi dan keperawatan dilewati di jalan. **Bima** menyukai komputer dan ingin segera bekerja: D3 informatika. **Sari** menyukai angka, tetapi nilai matematikanya 80, di bawah 85 milik node statistika, dan ia ingin belajar lebih lama, sehingga tidak ada pengecualian akuntansi yang dijalankan: akuntansi. **Tono** menyukai musik, dan belum ada yang mengajari basis pengetahuan apa pun tentang musik: belum ada saran. Itulah jawaban yang jujur, dan Subbab 4.7 menunjukkan bagaimana guru BK mengisi celah seperti itu.

> **WAWASAN KUNCI:** Menyimpan pengetahuan dengan berpusat pada variabel tidak mengubah apa yang *dikatakan* pengetahuan itu; yang berubah adalah apa yang *dapat Anda lakukan* dengannya. Tiga belas butir nasihat yang sama, disimpan sebagai tiga tabel yang terhubung, menjadi lebih kecil, tidak dapat bertentangan dengan dirinya sendiri, dan dapat ditelusuri per variabel, per node, dan per rule.

4.5 Dari Struktur Kita ke vcirs-core

Struktur buatan tangan itu sudah lengkap sebagai gagasan, tetapi tiga hal belum ada untuk sebuah penasihat sungguhan. Ia membutuhkan semua fakta di muka, sehingga siswa harus mengisi setiap pertanyaan; ia tidak dapat belajar, sehingga Bu Wati harus menyunting program untuk menambah nasihat; dan ia melupakan semuanya ketika program berakhir. Mesin bersama, vcirs-core, menyimpan tiga tabel yang persis sama dan menambahkan bagian-bagian yang hilang itu. **Tabel 4.2** memperlihatkan bagaimana gagasan-gagasan Langkah 2 muncul di dalamnya.

Tabel 4.2 Bagaimana struktur buatan tangan dari Langkah 2 muncul dalam vcirs-core.

Struktur kita (Subbab 4.4)	Vcirs-core
`split` dan `holds`	`Condition.parse("math score >= 85")` dan `kb.evaluate(condition, facts)`
`Node` dengan ID, induk, kondisi, dan program	`Node`: `id`, `parent`, `cf`, `conditions`, `conclusions`, `cornerstone`
peta `nodes` (Node Structure)	`kb.nodes`
peta `rules` (Rule Structure)	`kb.rules`, masing-masing sebuah `Rule` dengan `nodes`
peta `variables` (tabel Variabel)	`kb.variables` (dengan `question` dan `choices`) dan `kb.usage().variableSharing()`
`add(parent, conditions, program)`	`kb.refine(case, option, parent, chosen)`, atau cukup `kb.learn(case)`
`answer(facts)`	`kb.infer(facts, asker)`: menanyakan nilai yang belum diketahui, menjelaskan How dan Why
tabel-tabel yang dicetak `main`	`Report.variableSharing`, `Report.nodeSharing`, `Report.rules`, `Report.tree`
(hilang ketika program berakhir)	`kb.save(path)` dan `KnowledgeBase.load(path)`, satu berkas JSON untuk kedua bahasa

Dua perbedaan layak dibahas. Pertama, dalam vcirs-core sebuah kondisi adalah objek, bukan sepotong teks: `Condition` punya variabel, operator, nilai, CF, dan Credit, tepat seperti medan-medan pada **Gambar 4.3**, sehingga tidak ada yang perlu dipotong berulang-ulang. Kedua, node tidak ditambahkan dengan menyebut induknya dengan tangan. Guru BK memberikan sebuah **kasus** — fakta-fakta seorang siswa sungguhan dan program yang tepat — dan `learn` menemukan tempatnya, seperti di Bab 3: bila sebuah node sudah memberikan program yang tepat, kasus itu dicatat sebagai pemeriksaan; bila sebuah node memberikan program yang keliru, node baru menjadi pengecualiannya, dibangun dari perbedaan dengan kasus cornerstone node itu; bila tidak ada node yang menjawab, node baru masuk ke tingkat atas [9], [7]. Jadi Bu Wati tidak pernah perlu tahu node mana induk node mana. Ia hanya perlu tahu jawaban yang tepat untuk siswa di depannya.

4.6 Penasihat Pemilihan Jurusan

4.6.1 Apa yang pertama kali diajarkan Bu Wati

Penasihat mulai dari tiga belas butir nasihat Bu Wati, ditulis sebagai tiga belas kasus dalam sebuah berkas kasus, `study-seed-cases.json`:

```
{
  "name": "Study-program advisor",
  "description": "What Bu Wati, the school counselor, taught the advisor from last year's
school leavers.",
  "cases": [
    {"facts": ["likes = computers"], "conclusions": ["program = informatics"], "option":
-1, "author": "Wati"},
    {"facts": ["likes = computers", "wants = work soon"], "conclusions": ["program = D3
informatics"], "option": -1, "author": "Wati"},
    {"facts": ["likes = living things"], "conclusions": ["program = biology"], "option":
-1, "author": "Wati"},
    {"facts": ["likes = living things", "blood ok = yes"], "conclusions": ["program =
nursing"], "option": -1, "author": "Wati"},
    {"facts": ["likes = living things", "blood ok = yes", "science score >= 85"],
"conclusions": ["program = medicine"], "option": -1, "author": "Wati"},
    {"facts": ["likes = numbers"], "conclusions": ["program = accounting"], "option": -1,
"author": "Wati"},
    {"facts": ["likes = numbers", "math score >= 85"], "conclusions": ["program =
statistics"], "option": -1, "author": "Wati"},
    {"facts": ["likes = numbers", "wants = work soon"], "conclusions": ["program = D3
accounting"], "option": -1, "author": "Wati"},
    {"facts": ["likes = people"], "conclusions": ["program = communication science"],
"option": -1, "author": "Wati"},
    {"facts": ["likes = people", "patient with children = yes"], "conclusions": ["program
= primary teaching"], "option": -1, "author": "Wati"},
    {"facts": ["likes = building things"], "conclusions": ["program = civil
engineering"], "option": -1, "author": "Wati"},
    {"facts": ["likes = building things", "drawing = good", "math score >= 75"],
"conclusions": ["program = architecture"], "option": -1, "author": "Wati"},
    {"facts": ["likes = drawing"], "conclusions": ["program = visual communication
design"], "option": -1, "author": "Wati"}
  ]
}
```

Setiap kasus adalah seorang siswa yang diingat Bu Wati, diringkas menjadi fakta-fakta yang penting dan program yang ia sarankan. Kasus kedokteran, misalnya, berbunyi: *menyukai makhluk hidup, tidak apa-apa melihat darah, nilai IPA paling sedikit 85: kedokteran.* Kondisi dengan angka ditulis bersama operatornya, `science score >= 85`, karena itulah yang dimaksud guru BK. `"option": -1` meminta mesin memutuskan sendiri tempat tiap kasus, lewat `learn`, persis seperti bila Bu Wati membawa siswa-siswa itu satu per satu ke penasihat dengan urutan ini. Karena urutan itu, kasus keperawatan jatuh di bawah biologi, kasus kedokteran di bawah keperawatan, dan kasus statistika serta D3 akuntansi keduanya di bawah akuntansi — pohon pada **Gambar 4.2**, dibangun tanpa seorang pun menyebut induk.

Pertama kali penasihat dijalankan, ia membangun basis pengetahuannya dari berkas ini dan menyimpannya sebagai `study-kb.json` di folder yang sama; setelah itu ia memuat dan memperbarui berkas yang tersimpan itu.

4.6.2 Programnya

Kode 4.5 adalah penasihat itu dalam Java.

Kode 4.5 `StudyAdvisor.java`: penasihat pemilihan jurusan di atas vcirs-core.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;

import vcirs.Case;
import vcirs.CaseFile;
import vcirs.Conclusion;
import vcirs.Condition;
import vcirs.Fact;
import vcirs.Facts;
import vcirs.InferenceResult;
import vcirs.KnowledgeBase;
import vcirs.LearnResult;
import vcirs.Node;
import vcirs.Question;
import vcirs.Report;
import vcirs.Usage;
import vcirs.Variable;
import vcirs.VcirsException;

/**
 * Penasihat pemilihan jurusan: pengetahuan Bu Wati, disimpan sebagai
 * variabel, node, dan rule.
 * java StudyAdvisor [--echo] [--tree] [--tables] [knowledge-base.json]
 * Program hanya menanyakan yang perlu kepada lulusan sekolah, menyarankan
 * program studi beserta program yang dilewati di jalan, menjelaskan
 * alasannya, dan belajar dari guru BK bila ia tidak setuju. --tables
 * mencetak Variable-Centered Rule Structure: di mana tiap variabel dipakai,
 * rule mana yang berbagi tiap node, dan setiap rule sebagai rangkaian node.
 */
public class StudyAdvisor {
    static final String[][] QUESTIONS = {
        {"likes", "What do you enjoy most?", "computers", "living things",
            "numbers", "people", "building things", "drawing"},
        {"wants", "After school, do you want to work soon or study longer?",
            "work soon", "study longer"},
        {"blood ok", "Are you all right with seeing blood and wounds?",
            "yes", "no"},
        {"science score", "Your average report score in science (0-100)?"},
        {"math score", "Your average report score in mathematics (0-100)?"},
        {"patient with children", "Are you patient with small children?",
```

Kode 4.5 (Lanjutan) *StudyAdvisor.java: penasihat pemilihan jurusan di atas vcirs-core.*

```
            "yes", "no"},
        {"drawing", "How good are you at drawing?", "good", "average"},
        {"family farm", "Does your family have a farm, orchard or fish pond?",
            "yes", "no"},
    };

    static BufferedReader in = new BufferedReader(
            new InputStreamReader(System.in, StandardCharsets.UTF_8));
    static boolean echo = false;
    static KnowledgeBase kb;

    /** Membaca jawaban pengguna; bila echo aktif, mencetaknya juga. */
    static String read(String prompt) {
        System.out.print(prompt);
        String line;
        try {
            line = in.readLine();
        } catch (IOException e) {
            line = null;
        }
        String text = line == null ? "" : line.strip();
        if (echo) {
            System.out.println(text);
        }
        return text;
    }

    static Fact ask(Question q) {
        Variable var = kb.variables.get(q.variable);
        String text = q.variable;
        List<String> choices = q.choices;
        if (var != null && !var.question.isEmpty()) {
            text = var.question;                    // seperti ditulis Bu Wati
            choices = var.choices;
        }
        if (!choices.isEmpty()) {
            text += " (" + String.join("/", choices) + ")";
        }
        String answer = read(text + " > ");
        return answer.isEmpty() ? null : new Fact(q.variable, answer); // Enter
    }
```

Kode 4.5 (Lanjutan) *StudyAdvisor.java: penasihat pemilihan jurusan di atas vcirs-core.*

```
    /** Muat basis pengetahuan, atau bangun dari kasus-kasus awal. */
    static KnowledgeBase openKb(String path) {
        if (Files.exists(Path.of(path))) {
            return KnowledgeBase.load(path);
        }
        Path parent = Path.of(path).toAbsolutePath().getParent();
        CaseFile seed = CaseFile.read(parent.resolve("study-seed-cases.json")
                .toString());
        KnowledgeBase k = new KnowledgeBase(seed.name, seed.description);
        for (String[] q : QUESTIONS) {
            List<String> choices = new ArrayList<>();
            for (int i = 2; i < q.length; i++) {
                choices.add(q[i]);
            }
            k.declare(q[0], q[1], choices);
        }
        seed.build(k);
        k.save(path);
        return k;
    }

    static void printTables() {
        System.out.println("Where each variable is used (the Variable table):");
        for (String line : Report.variableSharing(kb)) {
            System.out.println("  " + line);
        }
        System.out.println("Which rules share each node:");
        for (String line : Report.nodeSharing(kb)) {
            System.out.println("  " + line);
        }
        System.out.println("Every rule as a sequence of nodes "
                + "(the Rule Structure):");
        for (String line : Report.rules(kb)) {
            System.out.println("  " + line);
        }
        Usage usage = kb.usage();
        System.out.println("Most important variables: "
                + String.join(", ", usage.importantVariables()));
        System.out.println("Most important nodes: "
                + String.join(", ", usage.importantNodes()));
    }
```

Kode 4.5 (Lanjutan) *StudyAdvisor.java: penasihat pemilihan jurusan di atas vcirs-core.*

```
    public static void main(String[] args) {
        String path = "../data/study-kb.json";
        boolean tree = false;
        boolean tables = false;
        for (String a : args) {
            if (a.equals("--echo")) {
                echo = true;
            } else if (a.equals("--tree")) {
                tree = true;
            } else if (a.equals("--tables")) {
                tables = true;
            } else {
                path = a;
            }
        }
        kb = openKb(path);
        if (tree) {
            for (String line : Report.tree(kb)) {
                System.out.println(line);
            }
            return;
        }
        if (tables) {
            printTables();
            return;
        }
        System.out.println("Study-program advisor: " + kb.nodes.size()
                + " pieces of advice so far.");
        System.out.println("Answer each question; press Enter if you do not "
                + "know.");
        InferenceResult result = kb.infer(Facts.none(), StudyAdvisor::ask);
        Node fired = result.paths.isEmpty() ? null : result.paths.get(0).last();
        if (fired != null && !result.conclusions().isEmpty()) {
            System.out.println("Suggested program: "
                    + result.conclusions().get(0).value);
            List<String> onWay = new ArrayList<>();
            for (Conclusion c : result.paths.get(0).intermediate()) {
                onWay.add(c.value);
            }
            if (!onWay.isEmpty()) {
                System.out.println("Also worth a look: "
                        + String.join(", ", onWay));
            }
            System.out.println("Why:");
            for (String line : result.how()) {
                System.out.println("  " + line);
```

Kode 4.5 (Lanjutan) *StudyAdvisor.java: penasihat pemilihan jurusan di atas vcirs-core.*

```
            }
        } else {
            System.out.println("I have no suggestion for this student yet.");
        }

        List<Condition> facts = new ArrayList<>();
        for (Fact f : result.memory.facts.values()) {
            if (!f.variable.equals("program")) {
                facts.add(new Condition(f.variable, "=", f.value));
            }
        }
        boolean agree = fired != null
                && read("Does the counselor agree? (yes/no) > ").equals("yes");
        if (agree) {
            kb.learn(new Case(facts, result.conclusions(), 100), "counselor");
            kb.save(path);
            System.out.println("Good. The case is recorded as checked.");
            return;
        }
        String program = read("Which program does the counselor suggest? "
                + "(Enter if none) > ");
        if (program.isEmpty()) {
            System.out.println("Then there is nothing to learn this time.");
            return;
        }
        while (true) {
            String extra = read("Another fact that matters? "
                    + "(variable = value, Enter when done) > ");
            if (extra.isEmpty()) {
                break;
            }
            try {
                facts.add(Condition.parse(extra));
            } catch (VcirsException e) {
                System.out.println("Please write it as: variable = value");
            }
        }
        Case c = new Case(facts, List.of(new Conclusion("program", program)),
                100);
        if (fired != null) {
            for (String line : kb.differences(fired.id, c).lines()) {
                System.out.println(line);
            }
        }
        LearnResult learned;
        try {
```

Kode 4.5 (Lanjutan) *StudyAdvisor.java: penasihat pemilihan jurusan di atas vcirs-core.*

```
            learned = kb.learn(c, "counselor");
        } catch (VcirsException e) {
            System.out.println("Cannot learn: " + e.getMessage());
            return;
        }
        System.out.println("Learned: " + learned);
        kb.save(path);
        System.out.println("Saved " + path + ": " + kb.summary());
    }
}
```

Inilah perjalanan satu konsultasi, sebagaimana dijalankan `main`.

1. **Buka basis pengetahuan.** `openKb` memuat `study-kb.json`, atau, pada kali pertama, membuatnya: ia **mendeklarasikan** delapan variabel dalam `QUESTIONS` — masing-masing

dengan pertanyaan yang diajukan dan jawaban-jawaban lazimnya — lalu menyuapkan kasus-kasus awal ke mesin dengan `seed.build(k)`. Mendeklarasikan variabel adalah cara tabel Variabel pada **Gambar 4.2** mendapatkan pertanyaan-pertanyaannya. Sebuah variabel dapat dideklarasikan sebelum ada node yang memakainya: `family farm` dideklarasikan di sini, walaupun tidak ada kasus awal yang menyebutnya, agar penasihat sudah tahu cara menanyakannya pada hari sebuah node membutuhkannya.

2. **Berkonsultasi.** `kb.infer(Facts.none(), StudyAdvisor::ask)` dimulai dengan memori kerja kosong. Setiap kali sebuah node membutuhkan nilai yang belum dimiliki mesin, mesin memanggil `ask` dengan sebuah `Question`. `ask` mencari variabelnya di tabel Variabel (`kb.variables`) dan mencetak pertanyaan guru BK bersama jawaban-jawaban lazimnya; balasan kosong berarti "saya tidak tahu", dan node yang membutuhkan nilai itu tidak dijalankan.
3. **Menyarankan dan menjelaskan.** Jalur pertama hasilnya, `result.paths.get(0)`, adalah rule yang menjawab. Node terakhirnya memberi program yang disarankan; `intermediate()` memberi konklusi node-node di atasnya, dicetak sebagai "Also worth a look" (juga layak dilihat); dan `result.how()` menjelaskan saran itu kondisi demi kondisi, menyebutkan asal setiap nilai.
4. **Memeriksa bersama guru BK.** Bila Bu Wati setuju, kasusnya diberikan kepada `learn`, yang mendapati jawabannya sudah benar dan mencatatnya dalam riwayat sebagai pemeriksaan. Bila tidak, program menanyakan program yang akan ia sarankan dan fakta lain yang penting, memperlihatkan daftar perbedaan terhadap kasus cornerstone node yang menjawab, dan membiarkan `learn` menambahkan pengetahuan baru. Berkasnya disimpan.

Opsi di akhir baris perintah membuka basis pengetahuan dengan cara lain: `--tree` mencetaknya sebagai pohon node dan pengecualian, dan `--tables` mencetak tiga tabel Variable-Centered Rule Structure, bersama variabel dan node terpenting (Subbab 4.8). `--echo` mengulang setiap jawaban yang dibaca program, sehingga konsultasi yang diketik ke sebuah berkas terbaca seperti percakapan ketika diputar ulang; begitulah transkrip-transkrip bab ini dibuat. **Kode 4.6** adalah aplikasi yang sama dalam Python.

Kode 4.6 `study_advisor.py`: penasihat pemilihan jurusan dalam Python.

```
"""Penasihat pemilihan jurusan: pengetahuan Bu Wati, disimpan sebagai
variabel, node, dan rule.

    python study_advisor.py [--echo] [--tree] [--tables] [knowledge-base.json]

Program hanya menanyakan yang perlu kepada lulusan sekolah, menyarankan
program studi beserta program yang dilewati di jalan, menjelaskan alasannya,
dan belajar dari guru BK bila ia tidak setuju. --tables mencetak
Variable-Centered Rule Structure: di mana tiap variabel dipakai, rule mana
yang berbagi tiap node, dan setiap rule sebagai rangkaian node.
"""

import os
import pathlib
import sys

# Beri tahu Python letak mesin vcirs-core (naik dua folder, lalu masuk).
sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import (Case, Condition, Conclusion, KnowledgeBase, VcirsError,
                   casefile, report)

QUESTIONS = [
    ("likes", "What do you enjoy most?",
     ["computers", "living things", "numbers", "people", "building things",
      "drawing"]),
    ("wants", "After school, do you want to work soon or study longer?",
     ["work soon", "study longer"]),
    ("blood ok", "Are you all right with seeing blood and wounds?",
     ["yes", "no"]),
    ("science score", "Your average report score in science (0-100)?", []),
    ("math score", "Your average report score in mathematics (0-100)?", []),
    ("patient with children", "Are you patient with small children?",
     ["yes", "no"]),
    ("drawing", "How good are you at drawing?", ["good", "average"]),
    ("family farm", "Does your family have a farm, orchard or fish pond?",
     ["yes", "no"]),
]


class Console:
    """Membaca jawaban pengguna; bila echo aktif, mencetaknya juga."""
```

Kode 4.6 (Lanjutan) *study_advisor.py: penasihat pemilihan jurusan dalam Python.*

```
    def __init__(self, kb, echo):
        self.kb = kb
        self.echo = echo

    def read(self, prompt):
        print(prompt, end="")
        text = sys.stdin.readline().strip()
        if self.echo:
            print(text)
        return text

    def ask(self, question):
        var = self.kb.variables.get(question.variable)
        text, choices = question.variable, question.choices
        if var is not None and var.question:
            text, choices = var.question, var.choices   # seperti ditulis Bu Wati
        if choices:
            text += " (" + "/".join(choices) + ")"
        return self.read(text + " > ") or None    # Enter = "saya tidak tahu"


def open_kb(path):
    """Muat basis pengetahuan, atau bangun dari kasus-kasus awal."""
    if os.path.exists(path):
        return KnowledgeBase.load(path)
    seed = os.path.join(os.path.dirname(path), "study-seed-cases.json")
    name, description, steps = casefile.read_cases(seed)
    kb = KnowledgeBase(name, description)
    for var, question, choices in QUESTIONS:
        kb.declare(var, question, choices)
    casefile.build(kb, steps)
    kb.save(path)
    return kb


def print_tables(kb):
    print("Where each variable is used (the Variable table):")
    for line in report.variable_sharing(kb):
        print("  " + line)
    print("Which rules share each node:")
    for line in report.node_sharing(kb):
        print("  " + line)
    print("Every rule as a sequence of nodes (the Rule Structure):")
    for line in report.rules(kb):
        print("  " + line)
    usage = kb.usage()
```

Kode 4.6 (Lanjutan) *study_advisor.py: penasihat pemilihan jurusan dalam Python.*

```
    print("Most important variables: " + ", ".join(usage.important_variables()))
    print("Most important nodes: " + ", ".join(usage.important_nodes()))


def main(args):
    echo = "--echo" in args
    files = [a for a in args if not a.startswith("--")]
    path = files[0] if files else "../data/study-kb.json"
    kb = open_kb(path)
    if "--tree" in args:
        for line in report.tree(kb):
            print(line)
        return
    if "--tables" in args:
        print_tables(kb)
        return
    con = Console(kb, echo)
    print("Study-program advisor: " + str(len(kb.nodes))
          + " pieces of advice so far.")
    print("Answer each question; press Enter if you do not know.")
    result = kb.infer({}, con.ask)
    fired = result.paths[0].last() if result.paths else None
    if fired is not None and result.conclusions():
        print("Suggested program: " + result.conclusions()[0].value)
        on_way = result.paths[0].intermediate()
        if on_way:
            print("Also worth a look: "
                  + ", ".join(c.value for c in on_way))
        print("Why:")
        for line in result.how():
            print("  " + line)
    else:
        print("I have no suggestion for this student yet.")

    facts = [Condition(f.variable, "=", f.value)
             for f in result.memory.facts.values() if f.variable != "program"]
    agree = fired is not None and con.read(
        "Does the counselor agree? (yes/no) > ") == "yes"
    if agree:
        kb.learn(Case(facts, list(result.conclusions())), "counselor")
        kb.save(path)
        print("Good. The case is recorded as checked.")
        return
    program = con.read("Which program does the counselor suggest? "
                       "(Enter if none) > ")
    if not program:
```

Kode 4.6 (Lanjutan) *study_advisor.py: penasihat pemilihan jurusan dalam Python.*

```
        print("Then there is nothing to learn this time.")
        return
    while True:
        extra = con.read("Another fact that matters? "
                         "(variable = value, Enter when done) > ")
        if not extra:
            break
        try:
            facts.append(Condition.parse(extra))
        except VcirsError:
            print("Please write it as: variable = value")
    case = Case(facts, [Conclusion("program", program)])
    if fired is not None:
        for line in kb.differences(fired.id, case).lines():
            print(line)
    try:
        learned = kb.learn(case, "counselor")
    except VcirsError as e:
        print("Cannot learn: " + str(e))
        return
    print("Learned: " + str(learned))
    kb.save(path)
    print("Saved " + path + ": " + kb.summary())


if __name__ == "__main__":
    main(sys.argv[1:])
```

Penanya (*asker*) dalam Python adalah metode `ask` milik objek kecil `Console`, yang juga memegang basis pengetahuan agar dapat mencari pertanyaan tiap variabel. Fungsi apa pun yang menerima pertanyaan dan mengembalikan jawaban dapat dipakai; Bab 10 menaruh basis pengetahuan yang sama di balik sebuah halaman web, dan hanya penanyanya yang berubah.

4.7 Empat Siswa di Meja Bu Wati

Kompilasi dan jalankan penasihat dari folder `Chapter04/java`:

```
javac -d out -sourcepath ../../vcirs-core/java/src *.java
java -cp out StudyAdvisor --echo < ../data/session-1.txt
```

atau, dari `Chapter04/python`, `python study_advisor.py --echo < ../data/session-1.txt`. (Windows PowerShell tidak mengenal `<`; ketikkan `Get-Content ..\data\session-1.txt | java -cp out StudyAdvisor --echo` sebagai gantinya.) Folder `data` memuat empat konsultasi subbab ini, satu jawaban per baris. Jalankan secara berurutan pada basis pengetahuan yang baru — hapus dulu `study-kb.json` bila ada — dan Anda akan melihat transkrip yang persis sama, dari Java maupun Python, dan `study-kb.json` yang sama, bita demi bita. (Pertanyaan dan keluaran program dalam bahasa Inggris, sama untuk kedua edisi.)

Sesi 1: Laras. Ia menyukai makhluk hidup, sudah bertahun-tahun membantu ibunya membersihkan lutut adiknya yang lecet, dan rata-rata IPA-nya 88.

```
Study-program advisor: 13 pieces of advice so far.
Answer each question; press Enter if you do not know.
What do you enjoy most? (computers/living things/numbers/people/building things/drawing) >
living things
Are you all right with seeing blood and wounds? (yes/no) > yes
Your average report score in science (0-100)? > 88
Suggested program: medicine
Also worth a look: biology, nursing
Why:
  medicine#1 fired because:
    likes = living things  (answered by the user, CF 100)
    blood ok = yes  (answered by the user, CF 100)
    science score >= 85  (science score = 88, answered by the user, CF 100)
    so program = medicine  [rule CF 100, conclusion CF 100]
Does the counselor agree? (yes/no) > yes
Good. The case is recorded as checked.
```

Tiga pertanyaan, tidak lebih. Node tingkat atas pertama, informatika, membutuhkan `likes = computers`; mesin menanyakan `likes`, dan jawaban *living things* langsung menggagalkannya. Node biologi membutuhkan variabel yang sama, yang sudah ada di memori kerja, sehingga ia dijalankan tanpa pertanyaan baru. Pengecualian pertamanya, keperawatan, membutuhkan `blood ok`, yang ditanyakan; pengecualian keperawatan, kedokteran, membutuhkan `science score`, yang ditanyakan dan dibandingkan sebagai angka, 88 terhadap 85. Penjelasannya memperlihatkan persis hal itu: `science score >= 85  (science score = 88, ...)`. Sarannya kedokteran, dan biologi serta keperawatan — node-node yang dilewati di jalan — ditawarkan sebagai program lain yang layak dilihat. Bagian keluaran inilah yang paling perlu dibaca cermat oleh keluarga Laras. Kedokteran di perguruan tinggi negeri termasuk pilihan yang paling ketat persaingannya; keperawatan dan biologi adalah program yang juga cocok dengannya, dan **Gambar 4.1** menunjukkan bahwa pilihan kedua hampir sama pentingnya dengan pilihan pertama. Bu Wati setuju, dan kasusnya dicatat sebagai sudah diperiksa.

Sesi 2: Bima. Ia menyukai komputer, dan ia ingin segera bekerja: ayahnya pengemudi ojek daring, dan Bima anak sulung dari empat bersaudara.

```
Study-program advisor: 13 pieces of advice so far.
Answer each question; press Enter if you do not know.
What do you enjoy most? (computers/living things/numbers/people/building things/drawing) >
computers
After school, do you want to work soon or study longer? (work soon/study longer) > work
soon
Suggested program: D3 informatics
Also worth a look: informatics
Why:
  D3 informatics#1 fired because:
    likes = computers  (answered by the user, CF 100)
    wants = work soon  (answered by the user, CF 100)
    so program = D3 informatics  [rule CF 100, conclusion CF 100]
Does the counselor agree? (yes/no) > yes
Good. The case is recorded as checked.
```

Dua pertanyaan. Node informatika dijalankan pada jawaban pertama, pengecualiannya D3 informatika dijalankan pada jawaban kedua, dan sarannya diploma tiga tahun, dengan sarjana empat tahun sebagai program yang layak dilihat. Bagi Bima ini lebih dari sekadar soal teknis. Menurut aturan SNBT 2026, siswa yang memilih tiga atau empat program harus menyertakan program vokasi — dan, dengan empat pilihan, paling sedikit satu D3 [3] — sehingga D3 bukan hadiah hiburan; ia bagian dari rencana yang disusun dengan baik.

Sesi 3: Rudi, dan sebuah celah dalam pengetahuan. Rudi juga menyukai makhluk hidup, tetapi ia pingsan bila melihat darah. Keluarganya menanam cabai dan punya kebun durian kecil di belakang rumah, dan ia sudah membantu di ladang sejak bisa berjalan.

```
Study-program advisor: 13 pieces of advice so far.
Answer each question; press Enter if you do not know.
What do you enjoy most? (computers/living things/numbers/people/building things/drawing) >
living things
Are you all right with seeing blood and wounds? (yes/no) > no
Suggested program: biology
Why:
  biology#1 fired because:
    likes = living things  (answered by the user, CF 100)
    so program = biology  [rule CF 100, conclusion CF 100]
Does the counselor agree? (yes/no) > no
Which program does the counselor suggest? (Enter if none) > agrotechnology
Another fact that matters? (variable = value, Enter when done) > family farm = yes
Another fact that matters? (variable = value, Enter when done) >
Differences with the cornerstone case of biology#1:
  A (only in the old case): (none)
  B (only in the new case): blood ok = no, family farm = yes
Learned: refined: agrotechnology#1 added as an exception of biology#1
Saved ../data/study-kb.json: Study-program advisor: 9 variables, 14 nodes, 14 rules
```

Penasihat menyarankan biologi: pengecualian keperawatan membutuhkan `blood ok = yes` dan gagal, sehingga konklusi node biologi sendiri yang berlaku. Bu Wati tidak setuju. Untuk anak yang tumbuh di ladang dan mencintainya, ia akan menyarankan **agroteknologi**, program yang mempelajari cara tanaman dibudidayakan, dilindungi, dan diperbaiki. Ia mengetik `agrotechnology`, dan satu fakta lagi yang belum pernah ditanyakan: `family farm = yes`.

Daftar perbedaan membandingkan Rudi dengan kasus cornerstone `biology#1` — kasus awal yang hanya memuat `likes = living things`. Bagian A, fakta yang hanya dimiliki kasus lama, kosong. Bagian B memuat dua fakta yang hanya dimiliki Rudi: `blood ok = no` dan `family farm = yes`. `learn` membangun node baru dari kedua bagian — opsi 3 dari enam opsi yang dikenal di Bab 3 — dan menggantungnya di bawah `biology#1`. Basis pengetahuan kini punya empat belas node.

Perhatikan baik-baik node baru itu, karena ia tidak persis seperti yang dimaksud Bu Wati. Ia akan menyarankan agroteknologi kepada anak petani entah ia takut darah atau tidak; kondisi `blood ok = no` ikut terbawa hanya karena kebetulan berbeda dengan kasus cornerstone. Node itu *benar* — ia tidak akan pernah memberi jawaban keliru kepada siswa seperti Rudi — tetapi ia *lebih sempit* daripada seharusnya: anak petani yang tidak apa-apa melihat darah akan diarahkan ke keperawatan dan tidak pernah mencapainya. Memilih perbedaan mana yang benar-benar penting adalah tugas pakar, dan Bab 5 memberi pakar alat untuk melakukannya: enam opsi, kondisi yang dipilih dengan tangan, dan pemeriksaan kasus-kasus yang akan terdampak oleh node baru.

Sesi 4: Wulan, seminggu kemudian. Orang tua Wulan punya kolam ikan dan kebun buah kecil; ia menyukai makhluk hidup, dan seperti Rudi ia lebih suka tidak melihat darah.

```
Study-program advisor: 14 pieces of advice so far.
Answer each question; press Enter if you do not know.
What do you enjoy most? (computers/living things/numbers/people/building things/drawing) >
living things
Are you all right with seeing blood and wounds? (yes/no) > no
Does your family have a farm, orchard or fish pond? (yes/no) > yes
Suggested program: agrotechnology
Also worth a look: biology
Why:
  agrotechnology#1 fired because:
    likes = living things  (answered by the user, CF 100)
    blood ok = no  (answered by the user, CF 100)
    family farm = yes  (answered by the user, CF 100)
    so program = agrotechnology  [rule CF 100, conclusion CF 100]
Does the counselor agree? (yes/no) > yes
Good. The case is recorded as checked.
```

Sekarang penasihat mengajukan pertanyaan baru — `Does your family have a farm, orchard or fish pond?` (apakah keluargamu punya ladang, kebun, atau kolam ikan?) — dengan kata-kata yang dideklarasikan Bu Wati, karena node yang dipelajari pada sesi 3 membutuhkan variabel `family farm`. Sarannya agroteknologi, dengan biologi di jalan, dan penjelasannya mendaftar tiga fakta yang menghasilkannya. Tidak ada satu baris kode pun yang diubah antara Rudi dan Wulan.

`--tree` mencetak basis pengetahuan setelah keempat siswa:

```
(root)
+- informatics#1: likes = computers -> program = informatics
|  +- D3 informatics#1: wants = work soon -> program = D3 informatics
+- biology#1: likes = living things -> program = biology
|  +- nursing#1: blood ok = yes -> program = nursing
|  |  +- medicine#1: science score >= 85 -> program = medicine
|  +- agrotechnology#1: blood ok = no, family farm = yes -> program = agrotechnology
+- accounting#1: likes = numbers -> program = accounting
|  +- statistics#1: math score >= 85 -> program = statistics
|  +- D3 accounting#1: wants = work soon -> program = D3 accounting
+- communication science#1: likes = people -> program = communication science
|  +- primary teaching#1: patient with children = yes -> program = primary teaching
+- civil engineering#1: likes = building things -> program = civil engineering
|  +- architecture#1: drawing = good, math score >= 75 -> program = architecture
+- visual communication design#1: likes = drawing -> program = visual communication design
```

Bandingkan dengan **Gambar 4.2**: bentuknya sama, dengan satu pengecualian baru di bawah `biology#1` di samping `nursing#1`. Node-node tingkat atas dicoba sesuai urutan diajarkannya, begitu pula pengecualian-pengecualian tiap node.

4.8 Membaca Strukturnya

Pohon memperlihatkan pengetahuan sebagaimana mesin *menelusurinya*. Tiga tabel memperlihatkannya sebagaimana mesin *menyimpannya*, dan di sinilah berpikir dengan variabel memberi hasil. `--tables` mencetaknya:

```
Where each variable is used (the Variable table):
  VariableID             NumOfNodeIDs  NodeID Used                    VariableOrder
  ---------------------  ------------  ----------------------------  -------------
  science score          1             medicine#1                     1
  patient with children  1             primary teaching#1             1
  drawing                1             architecture#1                 1
  family farm            1             agrotechnology#1               2
  wants                  2             D3 informatics#1               1
                                       D3 accounting#1                1
  blood ok               2             nursing#1                      1
                                       agrotechnology#1               1
  math score             2             statistics#1                   1
                                       architecture#1                 2
  likes                  6             informatics#1                  1
                                       biology#1                      1
                                       accounting#1                   1
                                       communication science#1        1
                                       civil engineering#1            1
                                       visual communication design#1  1
Which rules share each node:
  NodeID                         NumOfRuleIDs  RuleID Used                    NodeOrder
  -----------------------------  ------------  ----------------------------  ---------
  D3 informatics#1               1             D3 informatics#1               2
  medicine#1                     1             medicine#1                     3
  statistics#1                   1             statistics#1                   2
  D3 accounting#1                1             D3 accounting#1                2
  primary teaching#1             1             primary teaching#1             2
  architecture#1                 1             architecture#1                 2
  visual communication design#1  1             visual communication design#1  1
  agrotechnology#1               1             agrotechnology#1               2
  informatics#1                  2             informatics#1                  1
                                               D3 informatics#1               1
  nursing#1                      2             nursing#1                      2
                                               medicine#1                     2
  communication science#1        2             communication science#1        1
                                               primary teaching#1             1
  civil engineering#1            2             civil engineering#1            1
                                               architecture#1                 1
  accounting#1                   3             accounting#1                   1
                                               statistics#1                   1
                                               D3 accounting#1                1
  biology#1                      4             biology#1                      1
                                               nursing#1                      1
                                               medicine#1                     1
                                               agrotechnology#1               1
Every rule as a sequence of nodes (the Rule Structure):
  informatics#1: informatics#1
  D3 informatics#1: informatics#1 -> D3 informatics#1
  biology#1: biology#1
  nursing#1: biology#1 -> nursing#1
  medicine#1: biology#1 -> nursing#1 -> medicine#1
  accounting#1: accounting#1
  statistics#1: accounting#1 -> statistics#1
  D3 accounting#1: accounting#1 -> D3 accounting#1
  communication science#1: communication science#1
  primary teaching#1: communication science#1 -> primary teaching#1
  civil engineering#1: civil engineering#1
  architecture#1: civil engineering#1 -> architecture#1
  visual communication design#1: visual communication design#1
  agrotechnology#1: biology#1 -> agrotechnology#1
Most important variables: likes
Most important nodes: biology#1
```

Tabel Variabel. Tabel pertama mendaftar setiap variabel yang dipakai suatu node, dengan jumlah node yang memakainya (NumOfNodeIDs), tiap node itu, dan posisi variabel di dalamnya (VariableOrder). Urutannya dari variabel yang paling sedikit dibagi ke yang paling banyak, seperti

cara tesis mencetaknya [5]. Empat variabel masing-masing dipakai satu node; `wants`, `math score`, dan — sejak Rudi — `blood ok` dipakai dua node; dan `likes` enam node. Dua hal layak diperhatikan. `blood ok` kini dipakai bersama oleh `nursing#1` (sebagai `yes`) dan oleh `agrotechnology#1` (sebagai `no`): pertanyaan yang sama melayani dua butir nasihat. Dan `family farm`, yang sudah dideklarasikan sejak awal, baru muncul sekarang, karena baru sekarang ada node yang memakainya.

Rule mana yang berbagi tiap node. Tabel kedua mendaftar setiap node dengan jumlah rule yang memuatnya (NumOfRuleIDs), tiap rule itu, dan posisi node di dalamnya (NodeOrder). Node yang hanya menjadi node terakhir rule-nya sendiri, seperti `medicine#1`, dipakai satu rule. `biology#1` dipakai empat: rule-nya sendiri, dan rule keperawatan, kedokteran, dan agroteknologi, tempat ia selalu menjadi node 1. Makin banyak rule yang berbagi sebuah node, makin banyak pengetahuan yang bergantung padanya: bila `biology#1` keliru, empat saran ikut keliru [5].

Setiap rule sebagai rangkaian node. Tabel ketiga adalah Rule Structure: empat belas rule, satu per node, masing-masing rangkaian node dari tingkat atas ke bawah. Isinya sama dengan pohon, ditulis seperti cara penalaran maju dan mundur di Bab 8 akan membacanya.

Variabel dan node terpenting. Dua baris terakhir adalah hasil pertama dari apa yang disebut tesis **analisis variabel**: variabel terpenting adalah yang dipakai bersama oleh node terbanyak, dan node terpenting adalah yang dipakai bersama oleh rule terbanyak [5]. Di sini jawabannya `likes` dan `biology#1` — tidak mengejutkan Bu Wati, yang tahu bahwa apa yang disukai siswa adalah hal pertama yang ditanyakan, dan bahwa sebagian besar kasus sulitnya adalah siswa yang menyukai makhluk hidup. Dalam basis pengetahuan berisi beberapa ratus node yang dibangun beberapa guru BK, tidak seorang pun akan tahu jawabannya tanpa tabel-tabel ini. Bab 6 memakai node dan variabel terpenting ini sebagai titik awal **pembangkitan rule**: mesin mengusulkan rule dan node baru yang dibangun di sekitarnya, dan pakar mengonfirmasi yang masuk akal. Bab 7 menambahkan separuh lainnya, **analisis nilai**, yang menimbang tiap variabel menurut Credit dan posisinya dan mengubah hitungan-hitungan itu menjadi tingkat pemakaian.

4.9 Membuatnya Lebih Baik

Urutan pertanyaan yang lebih baik. Penasihat menanyakan `likes` lebih dulu karena kasus awal pertama kebetulan dimulai dengannya. Untuk basis pengetahuan ini, itu juga urutan terbaik, tetapi secara umum belum tentu. Mesin dapat mengurutkan variabel menurut seberapa banyak ia dipakai — `kb.usage().questionOrder()` — dan sebuah aplikasi dapat mengajukan pertanyaan yang paling berguna lebih dulu. Bab 7 melakukannya untuk pesan-pesan pelanggan sebuah toko.

Angka yang dipilih guru BK. Kasus-kasus awal memakai `science score >= 85` karena Bu Wati menulisnya begitu. Tetapi ketika seorang siswa diajarkan di meja, fakta-faktanya berasal dari konsultasi, dan sebuah angka menjadi kondisi persis seperti `science score = 91`, yang hanya akan cocok dengan siswa bernilai tepat 91. Ketika node baru membutuhkan angka, guru BK sebaiknya memilih kondisinya sendiri, dengan `refine` dan daftar kondisi pilihan; Soal 3.6 di Bab 3 menunjukkan caranya, dan Bab 5 menjadikannya bagian dari layar mengajar.

Program cadangan. Baris “Also worth a look” hanya mendaftar program yang dilewati di sepanjang satu rule. Seorang siswa dapat cocok dengan dua rule sekaligus — Sari, yang menyukai angka, bernilai matematika 90, dan ingin segera bekerja, cocok dengan statistika maupun D3 akuntansi. `kb.infer(facts, asker, true)` (Python: `multiple=True`) membuat mesin mengikuti setiap cabang yang cocok dan mengembalikan satu jalur per saran, tepat yang dibutuhkan daftar sampai empat pilihan SNBT. Soal 4.6 meminta Anda mencobanya.

Memeriksa rencana terhadap aturan. Begitu seorang siswa punya daftar program, beberapa aturan biasa dapat memeriksanya terhadap ketentuan SNBT 2026: paling banyak empat program, paling banyak dua yang akademik, dan dengan tiga atau empat pilihan, campuran program vokasi yang tepat serta paling sedikit satu D3 pada sebagian kombinasi [3]. Aturan-aturan ini tidak berasal dari kasus; ia berasal dari peraturan, dan paling baik ditulis sebagai kode biasa, seperti aturan penghematan di Bab 2. Soal 4.9 membangun pemeriksanya.

Pengujian. `StudyTest.java` dan `test_study.py` memeriksa bagian-bagian bab ini: bahwa `split` memotong kondisi dengan benar, bahwa struktur buatan tangan punya tiga belas node dengan rule dan jawaban yang tepat, bahwa basis pengetahuan awal punya tiga belas node dengan `medicine#1` di bawah `nursing#1` dan `likes` sebagai variabel terpentingnya, bahwa `learn` menggantung kasus Rudi di bawah `biology#1`, dan bahwa pemeriksa rencana dari Soal 4.9 menerima dan menolak rencana yang tepat. Keduanya diakhiri dengan:

```
all tests passed
```

4.10 Memakainya Sungguh-Sungguh

Di sekolah. Jalankan penasihat di komputer ruang BK, atau di laptop pada hari orientasi sekolah, dan biarkan siswa berkonsultasi dengannya sebelum menemui guru BK. Ia tidak menggantikan percakapan; ia menyiapkannya. Siswa yang datang dengan "penasihat menyarankan keperawatan, dengan biologi di jalan — dan saya tidak tahu keperawatan punya D3" mengajukan pertanyaan yang lebih baik dan memperoleh lebih banyak dari lima belas menit. Ketika guru BK tidak setuju dengan sebuah saran, ia mengajari penasihat saat itu juga; setelah satu tahun ajaran, basis pengetahuan menyimpan pengalamannya tahun itu.

Mulailah dari lulusan tahun lalu. Kasus awal terbaik adalah siswa sungguhan: kelas XII tahun lalu, ke mana mereka mendaftar, di mana mereka diterima, dan bagaimana kabar mereka sekarang. Tulislah sepuluh atau lima belas yang paling khas sebagai kasus, seperti dalam `study-seed-cases.json`, dan biarkan sisanya diajarkan di meja. Batasi fakta-faktanya pada hal yang dapat dijawab siswa dengan jujur tentang dirinya — apa yang ia sukai, rata-rata rapornya, apakah ia tahan melihat darah, apa yang dibutuhkan keluarganya — dan jangan menyimpan nama di dalam kasus. Basis pengetahuan ini dimaksudkan untuk diwariskan kepada guru BK berikutnya; ia semestinya menyimpan pengetahuan tentang jenis-jenis siswa, bukan tentang orang tertentu.

Simpan fakta resmi di luar penasihat. Program studi, daya tampungnya, persyaratannya, dan aturan SNBP serta SNBT berubah setiap tahun [3], [2]. Semua itu tempatnya di portal SNPMB dan di halaman penerimaan tiap perguruan tinggi, bukan di basis pengetahuan berisi nasihat guru BK. Yang disimpan penasihat adalah pengetahuan yang *tidak* berubah setiap tahun: jenis siswa mana yang berkembang di jenis program mana.

Soal biaya. Bagi banyak keluarga, termasuk keluarga Pak Joko, pertanyaan pertama bukan *program apa* melainkan *sanggupkah kami membiayainya*. Sampaikan dengan terang kepada setiap siswa: sepertiga siswa yang diterima lewat SNBT 2026 adalah pendaftar KIP Kuliah [4]. Variabel seperti `needs KIP Kuliah` dengan saran *daftarkan KIP Kuliah saat mendaftar* adalah salah satu nasihat paling berguna yang dapat diberikan sekolah, dan Soal 4.8 menanyakan di mana tempatnya dalam penasihat.

Di luar sekolah. Tidak ada bagian `StudyAdvisor` yang khusus tentang program studi kecuali daftar pertanyaan dan berkas kasus awalnya. Balai latihan kerja dapat menyarankan pelatihan yang cocok bagi pencari kerja; koperasi dapat menyarankan produk simpanan yang cocok bagi

anggotanya; remaja masjid dapat menyarankan peran relawan yang cocok bagi anggota baru. Masing-masing menulis variabelnya lebih dulu, seperti pada **Tabel 4.1**, lalu selusin kasus awal, dan mengajarkan sisanya sambil berjalan. Bab 14 membangun alat yang melakukannya tanpa pemrograman.

4.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Dua nama untuk satu variabel. `likes`, `enjoys`, dan `hobby` dalam kasus-kasus yang berbeda menjadi tiga variabel, dan tabel Variabel tidak lagi dapat memperlihatkan di mana pengetahuan tentang minat berada. *Hindari:* tulis tabel variabel lebih dulu (**Tabel 4.1**) dan patuhi; deklarasikan setiap variabel bersama pertanyaannya.

Dua ejaan untuk satu nilai. `living things`, `Living things`, dan `living-things` adalah tiga nilai berbeda bagi mesin. *Hindari:* deklarasikan nilai-nilai lazim bersama setiap variabel; penasihat menampilkannya dalam tanda kurung, dan siswa memilih di antaranya.

Satu variabel untuk dua hal. Variabel `interest` dengan nilai seperti `computers`, `yes`, dan `work soon` mencampur tiga pertanyaan menjadi satu, dan tidak ada kondisi di atasnya yang dapat dibaca. *Hindari:* satu variabel, satu pertanyaan.

Angka persis padahal yang dimaksud ambang. `math score = 90` hanya cocok dengan 90. *Hindari:* pakai `>=` atau `<=` dalam kasus-kasus awal, dan pilih kondisinya dengan tangan ketika node baru membutuhkan angka.

Lupa bahwa urutan kondisi itu penting. Mesin memeriksa, dan menanyakan, kondisi-kondisi sesuai urutan kondisi itu ditulis. Node yang ditulis `science score >= 85, likes = living things` akan menanyakan nilai IPA setiap siswa sebelum menanyakan apa yang ia sukai. *Hindari:* taruh lebih dulu kondisi yang menyaring siswa paling banyak.

Menerima setiap perbedaan. Pada sesi 3, `blood ok = no` masuk ke node Rudi hanya karena berbeda dengan kasus cornerstone, dan membuat node itu lebih sempit daripada maksud Bu Wati. *Hindari:* baca daftar perbedaan sebelum menerima node baru; Bab 5 menunjukkan cara menyimpan hanya fakta-fakta yang menjelaskan nasihat baru.

Memperlakukan saran sebagai keputusan. Penasihat tahu apa yang diketahui Bu Wati, tidak lebih. *Hindari:* sajikan keluarannya sebagai "program yang layak dilihat", dan selalu periksa persyaratan resminya [2].

Memasukkan aturan tahunan ke basis pengetahuan. Node yang berbunyi *pilih paling banyak dua program S1* adalah peraturan, bukan pengalaman guru BK, dan ia akan keliru pada tahun peraturannya berubah. *Hindari:* simpan peraturan dalam kode biasa (Soal 4.9), dan pengalaman dalam basis pengetahuan.

4.12 Ringkasan Bab

- Nasihat menjadi pengetahuan yang dapat dipakai komputer ketika dipecah menjadi **variabel** (yang ditanyakan), **nilai** (jawabannya), **kondisi** (variabel, salah satu operator `=`, `<>`, `<`, `<=`, `>`, `>=`, dan nilai), dan **konklusi** (**Tabel 4.1**).
- **Node** adalah rangkaian kondisi pendek yang berurutan beserta konklusinya, dinamai menurut konklusi pertamanya (`nursing#1`); pengecualian hanya memuat kondisinya sendiri. **Rule** adalah rangkaian node dari tingkat atas ke bawah, dinamai menurut node terakhirnya; konklusinya adalah konklusi node terakhir, dan node-node di atasnya memberi konklusi antara.

- **Variable-Centered Rule Structure** menyimpan tiga tabel yang terhubung — variabel, node, rule (**Gambar 4.2**, **Gambar 4.3**). Mengindeks nasihat per variabel memperlihatkan kosakata dan pengulangannya (**Kode 4.1**, **Kode 4.2**); menyimpannya sebagai node memangkas 22 kondisi menjadi 14 dan membuatnya dapat ditelusuri per variabel, node, dan rule (**Kode 4.3**, **Kode 4.4**).
- vcirs-core menyimpan tabel-tabel yang sama dan menambahkan pertanyaan yang diajukan hanya bila perlu, belajar dari kasus, penjelasan, dan berkas JSON yang dipakai bersama Java dan Python (**Tabel 4.2**).
- **Penasihat pemilihan jurusan** (**Kode 4.5**, **Kode 4.6**) mengajukan dua atau tiga pertanyaan per siswa, menyarankan program beserta cadangan yang dilewati di jalan, menjelaskan setiap saran, mempelajari pengecualian baru dari guru BK dengan variabel baru, dan mengajukan pertanyaan baru itu kepada siswa berikutnya.
- **Analisis variabel** membaca tabel-tabel itu: variabel yang paling banyak dibagi (`likes`) dan node yang paling banyak dibagi (`biology#1`) adalah yang terpenting, dan di sanalah pembangkitan rule Bab 6 dimulai. Dengan kurang dari tiga dari sepuluh pendaftar SNBT 2026 yang diterima, dan hampir semuanya lewat dua pilihan pertama (**Gambar 4.1**), membantu siswa memilih dengan baik sepadan dengan usahanya.

4.13 Soal Latihan

4.1 Pecahlah setiap nasihat berikut menjadi variabel, kondisi, dan konklusi, dengan memakai variabel-variabel pada **Tabel 4.1** sedapat mungkin. Sebutkan apakah masing-masing akan menjadi node tingkat atas atau pengecualian, dan pengecualian dari node mana. (a) *Siswa yang menyukai orang dan sabar menghadapi anak kecil sebaiknya melihat pendidikan guru sekolah dasar.* (b) *Bila siswa menyukai membangun sesuatu, pandai menggambar, dan rata-rata matematikanya paling sedikit 75, arsitektur layak dilihat.* (c) *Siswa yang menyukai komputer dan juga pandai menggambar sebaiknya melihat desain komunikasi visual, bukan informatika.*

4.2 Dengan tangan, memakai struktur yang dicetak **Kode 4.4**, temukan jalur dan saran untuk: (a) `likes = building things`, `drawing = good`, `math score = 70`; (b) `likes = people`, `patient with children = yes`; (c) `likes = numbers`, `math score = 90`, `wants = work soon`.

4.3 Tuliskan rule `medicine#1` sebagai satu aturan JIKA-MAKA datar. Lalu andaikan Bu Wati mengajarkan tiga pengecualian lagi di bawah `medicine#1`, masing-masing dengan satu kondisi. Berapa kondisi yang bertambah dalam struktur, dan berapa yang bertambah dalam daftar datar aturan lengkap?

4.4 Setelah sesi 3, tabel Variabel menyatakan bahwa `blood ok` dipakai dua node. Langkah 1 mengeluhkan bahwa `blood ok = yes` ditulis dua kali dalam daftar datar. Apakah pemakaian bersama yang baru ini masalah yang sama? Jelaskan.

4.5 Pada sesi 3, andaikan Rudi menekan Enter pada pertanyaan tentang darah alih-alih menjawab *no*. Fakta apa saja yang akan dimuat kasusnya, seperti apa daftar perbedaannya, dan node apa yang akan dibuat `learn`? Apakah node itu akan menjawab anak petani yang *tidak apa-apa* melihat darah?

4.6 Sari menyukai angka, rata-rata matematikanya 90, dan ia ingin segera bekerja. Tulislah program kecil yang menjalankan basis pengetahuan awal pada fakta-faktanya dua kali, sekali dengan cara biasa dan sekali dengan `multiple` dinyalakan, dan mencetak setiap rule yang menjawab. Jelaskan perbedaannya, dan mengapa itu penting bagi rencana SNBT.

4.7 Node mana yang akan menjadi node terpenting, dalam arti Subbab 4.8, bila Bu Wati mengajarkan tiga pengecualian lagi di bawah `D3 informatics#1`? Hitung rule-rule yang akan berbagi tiap kandidat.

4.8 Bu Wati ingin setiap siswa yang membutuhkan bantuan biaya diberi tahu agar mendaftar KIP Kuliah. Di mana nasihat ini sebaiknya ditaruh? Pertimbangkan (a) kasus dengan konklusi `note = apply for KIP Kuliah` yang diajarkan kepada penasihat, (b) aturan biasa dalam kode aplikasi, dan (c) basis pengetahuan kecil kedua. Jelaskan apa yang akan dilakukan `learn` dengan (a).

4.9 Tulislah program yang memeriksa daftar pilihan SNBT 2026 seorang siswa — masing-masing program dengan jenjangnya, S1, D4, atau D3 — terhadap aturan di Subbab 4.1 dan Subbab 4.9 [3], dan mencetak *allowed* (boleh) atau apa yang salah. Cobalah pada lima rencana.

4.14 Penyelesaian Soal Latihan

Penyelesaian 4.1. (a) Variabel `likes` dan `patient with children`; kondisi `likes = people` dan `patient with children = yes`; konklusi `program = primary teaching`. Ia pengecualian dari `communication science#1`, yang hanya memuat `patient with children = yes` — tepat node pada Langkah 2. (b) Variabel `likes`, `drawing`, dan `math score`; kondisi `likes = building things`, `drawing = good`, `math score >= 75`; konklusi `program = architecture`; pengecualian dari `civil engineering#1` yang memuat dua kondisi terakhir, dengan urutan itu. (c) Variabel `likes` dan `drawing`; kondisi `likes = computers` dan `drawing = good`; konklusi `program = visual communication design`. Ia pengecualian dari `informatics#1` yang hanya memuat `drawing = good`. Variabel `drawing` lalu dipakai bersama dua node, `architecture#1` dan node baru itu — pertanyaan yang sudah dideklarasikan melayani nasihat kedua.

Penyelesaian 4.2. (a) `civil engineering#1` dijalankan pada `likes = building things`. Pengecualiannya `architecture#1` memeriksa `drawing = good` (benar) lalu `math score >= 75`, yang salah untuk 70, sehingga ia tidak dijalankan. Jalurnya `civil engineering#1`, dan sarannya teknik sipil. (b) `communication science#1` dijalankan, lalu `primary teaching#1`: pendidikan guru sekolah dasar. (c) `accounting#1` dijalankan; pengecualian pertamanya, `statistics#1`, membutuhkan `math score >= 85`, benar untuk 90, dan dijalankan. Pencarian turun ke bawah, bukan ke samping, sehingga `D3 accounting#1` tidak pernah dicoba: sarannya statistika. Soal 4.6 menunjukkan cara melihat keduanya.

Penyelesaian 4.3. JIKA `likes = living things` DAN `blood ok = yes` DAN `science score >= 85` MAKA `program = medicine`. Tiga pengecualian yang masing-masing berisi satu kondisi menambah tiga kondisi pada struktur. Dalam daftar datar, setiap aturan baru harus mengulang tiga kondisi `medicine#1` dan menambah kondisinya sendiri: 3 × 4 = 12 kondisi.

Penyelesaian 4.4. Bukan. Daftar datar menulis kondisi yang *sama*, `blood ok = yes`, dua kali, sebagai bagian dari dua aturan, sehingga perubahan padanya harus dilakukan di dua tempat. Setelah sesi 3, kedua node memakai *variabel* yang sama dengan nilai berbeda — `blood ok = yes` untuk keperawatan dan `blood ok = no` untuk agroteknologi — sebagai dua butir nasihat yang berbeda. Variabelnya disimpan sekali di tabel Variabel, bersama pertanyaannya; hanya kondisinya yang berbeda. Itulah berbagi, dan tabel Variabel yang membuatnya terlihat.

Penyelesaian 4.5. Jawaban yang tidak diketahui tidak dimasukkan ke memori kerja, sehingga kasusnya memuat `likes = living things` dari konsultasi dan `family farm = yes` yang diketik Bu Wati. Daftar perbedaan terhadap kasus cornerstone `biology#1` akan berisi bagian A yang kosong dan `family farm = yes` sendirian di bagian B, dan `learn` akan membuat node

`agrotechnology#1: family farm = yes -> program = agrotechnology` di bawah `biology#1` — node yang dimaksud Bu Wati. Node itu tetap tidak akan menjawab anak petani yang tidak apa-apa melihat darah: `nursing#1` berada lebih dulu di antara pengecualian `biology#1`, dan ia dijalankan untuk anak itu, sehingga mesin turun ke keperawatan dan tidak pernah mencoba node agroteknologi. Urutan pengecualian juga merupakan pengetahuan; Bab 5 menunjukkan bagaimana pakar menangani kasus seperti itu.

Penyelesaian 4.6. Program penyelesaiannya adalah `BackupPrograms.java` dan `backup_programs.py`. Versi Python-nya:

```
"""Soal 4.6: semua program yang cocok bagi siswa, dengan multiple=True."""

import pathlib
import sys

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import KnowledgeBase, casefile

name, description, steps = casefile.read_cases("../data/study-seed-cases.json")
kb = casefile.build(KnowledgeBase(name, description), steps)
sari = {"likes": "numbers", "math score": "90", "wants": "work soon"}
for multiple in (False, True):
    result = kb.infer(sari, multiple=multiple)
    print("every rule that fits:" if multiple else "the first rule only:")
    for path in result.paths:
        print("  " + path.conclusions()[0].value + "  (rule "
              + " -> ".join(n.id for n in path.nodes) + ")")
```

Program ini mencetak:

```
the first rule only:
  statistics  (rule accounting#1 -> statistics#1)
every rule that fits:
  statistics  (rule accounting#1 -> statistics#1)
  D3 accounting  (rule accounting#1 -> D3 accounting#1)
```

Biasanya mesin berhenti pada rule pertama yang menjawab, `statistics#1`, karena pengecualian akuntansi pertama yang dijalankan yang menang. Dengan `multiple` dinyalakan, mesin mengikuti setiap pengecualian yang dijalankan dan mengembalikan satu jalur per rule, seperti Multiple Classification RDR: statistika *dan* D3 akuntansi. Untuk rencana SNBT sampai empat pilihan, yang harus mencampur program akademik dan vokasi, daftar kedua inilah yang perlu Sari bicarakan dengan Bu Wati.

Penyelesaian 4.7. Saat ini `biology#1` dipakai bersama empat rule (setelah sesi 3), `accounting#1` tiga, dan `informatics#1` dua. Tiga pengecualian baru di bawah `D3 informatics#1` menciptakan tiga rule baru, masing-masing diawali `informatics#1 -> D3 informatics#1 -> …`. Maka `informatics#1` dipakai bersama 2 + 3 = 5 rule, `D3 informatics#1` 1 + 3 = 4, dan `biology#1` tetap 4. Node terpentingnya menjadi `informatics#1`.

Penyelesaian 4.8. (a) adalah jebakan. `learn` membandingkan jawaban basis pengetahuan (`program = ...`) dengan konklusi kasus (`note = apply for KIP Kuliah`); keduanya berbeda, sehingga ia menambahkan pengecualian di bawah node yang menjawab, dan sejak itu siswa-siswa node tersebut mendapat catatan itu *sebagai ganti* sebuah program. Catatan itu tidak ada hubungannya dengan program mana yang cocok bagi siswa. (b) adalah jawaban benar yang paling sederhana: aturan biasa dalam aplikasi, seperti aturan-aturan di Bab 2, dicetak setelah saran — *bila needs KIP Kuliah = yes, daftarkan KIP Kuliah ketika mendaftar SNBP atau SNBT*. (c) tepat bila sekolah punya banyak nasihat semacam itu tentang biaya, tempat tinggal, atau

beasiswa: basis pengetahuan kecil kedua, dikonsultasikan setelah yang pertama, memisahkannya dan dapat diajari kasus demi kasus seperti yang pertama.

Penyelesaian 4.9. Program penyelesaiannya adalah `SnbtChoices.java` dan `snbt_choices.py`. Versi Python-nya:

```
"""Soal 4.9: periksa rencana pilihan program studi SNBT 2026 terhadap
aturan yang diumumkan SNPMB untuk 2026."""


def problems(choices):
    """choices: daftar (program, jenjang) dengan jenjang "S1" (akademik),
    "D4" atau "D3" (vokasi). Jawaban kosong berarti rencananya boleh."""
    n = len(choices)
    academic = sum(1 for _, level in choices if level == "S1")
    d3 = sum(1 for _, level in choices if level == "D3")
    out = []
    if n < 1 or n > 4:
        out.append("choose one to four programs")
    if academic > 2:
        out.append("at most two academic (S1) programs")
    if n == 3 and academic == 0 and d3 == 0:
        out.append("three vocational choices need at least one D3 program")
    if n == 4 and academic == 0:
        out.append("four choices need one or two academic programs")
    if n == 4 and d3 == 0:
        out.append("four choices need at least one D3 program")
    return out


PLANS = [
    ("Laras, plan 1", [("medicine", "S1"), ("nursing", "S1")]),
    ("Laras, plan 2", [("medicine", "S1"), ("nursing", "S1"),
                       ("biology", "S1")]),
    ("Laras, plan 3", [("medicine", "S1"), ("nursing", "S1"),
                       ("D3 nursing", "D3"), ("applied midwifery", "D4")]),
    ("Bima, plan 1", [("software engineering", "D4"), ("multimedia", "D4"),
                      ("computer networks", "D4")]),
    ("Bima, plan 2", [("informatics", "S1"), ("D3 informatics", "D3"),
                      ("software engineering", "D4"), ("multimedia", "D4")]),
]

if __name__ == "__main__":
    for name, plan in PLANS:
        found = problems(plan)
        print(name + ": " + ("allowed" if not found else "; ".join(found)))
```

Program ini mencetak:

```
Laras, plan 1: allowed
Laras, plan 2: at most two academic (S1) programs
Laras, plan 3: allowed
Bima, plan 1: three vocational choices need at least one D3 program
Bima, plan 2: allowed
```

Aturan-aturannya diambil langsung dari halaman SNPMB untuk 2026 [3]: satu atau dua pilihan bebas; tiga pilihan boleh dua akademik dan satu vokasi, satu akademik dan dua vokasi, atau tiga vokasi dengan paling sedikit satu D3; empat pilihan harus dua akademik dan dua vokasi, atau satu akademik dan tiga vokasi, selalu dengan paling sedikit satu D3. Fungsi itu menuliskannya sebagai lima pemeriksaan. Karena peraturan berubah, pemeriksaan-pemeriksaan itu duduk dalam satu fungsi kecil yang mudah diperbarui tahun depan, dan tidak di dalam basis pengetahuan.

Daftar Pustaka

[1] Universitas Negeri Yogyakarta, "Seleksi Nasional Berdasarkan Prestasi (SNBP) 2026," PMB UNY, 2026. [Online]. Available: https://pmb.uny.ac.id/jalur/SNBP

[2] Kementerian Pendidikan Dasar dan Menengah, "Panduan umum dan ketentuan UTBK-SNBT 2026," Pusat Informasi Rumah Pendidikan, 2026. [Online]. Available: https://pusatinformasi.rumahpendidikan.kemendikdasmen.go.id/hc/id/articles/55040921152537-Panduan-Umum-dan-Ketentuan-UTBK-SNBT-2026

[3] SNPMB, "Informasi umum UTBK-SNBT 2026," Seleksi Nasional Penerimaan Mahasiswa Baru, 2026. [Online]. Available: https://snpmb.id/utbk-snbt/informasi-umum

[4] SNPMB, "Siaran pers 02/sipers/snpmb/V/2026 tentang pengumuman hasil SNBT 2026," May 25, 2026. [Online]. Available: https://files.snpmb.id/web2026/02_Sipers%20Pengumuman%20SNBT%202026.pdf

[5] M. M. I. Subakti, "A Variable-Centered Intelligent Rule System," M.Sc.Eng. thesis, Dept. of Computer Science and Information Engineering, Taipei, Taiwan, Jun. 2005.

[6] I. Subakti, "A variable-centered intelligent rule system," in *Proc. 1st Annual Int. Conf. Information and Communication Technology Seminar (ICTS 2005)*, vol. 1, no. 1, Surabaya, Indonesia, Aug. 2005, pp. 167–174.

[7] I. Subakti, *Variable-Centered Intelligent Rule System*, Indonesian ed. (resume). Surabaya, Indonesia: Dept. of Informatics, Institut Teknologi Sepuluh Nopember, 2006.

[8] I. Subakti and O. Wijayanto, "Penerapan konsep fuzzy dalam Variable-Centered Intelligent Rule System (studi kasus: pemilihan jurusan di Chinese University of Hongkong)," Dept. of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia.

[9] I. Subakti, "Some revisions in VCIRS and cases reconstructing perspectives," in *Proc. Information and Communication Technology Seminar (ICTS 2006)*, Surabaya, Indonesia, 2006.

Bab 5

Penasihat Cek Kesehatan Mandiri: Penyempurnaan Pengetahuan Kasus demi Kasus

Di akhir bab ini Anda akan mampu menjelaskan mengapa pemeriksaan kesehatan mandiri harus menaruh **aturan keselamatan** yang tetap di depan apa pun yang telah dipelajarinya; menghitung, dengan tangan maupun dengan mesin, **daftar perbedaan** antara kasus baru dan **kasus cornerstone** sebuah node; menyebutkan apa yang dibangun oleh masing-masing dari **enam opsi** VCIRS, dan memilih di antaranya; memilih sendiri kondisi-kondisi node baru ketika Anda lebih tahu daripada daftar perbedaan; memeriksa **kasus-kasus lama** mana yang akan diubah sebuah node baru sebelum Anda menambahkannya; mengistirahatkan sebuah saran dengan **menghentikan** node-nya alih-alih menghapusnya; serta menjalankan **cek kesehatan mandiri**, aplikasi dalam Java dan Python yang mengajukan pertanyaan keselamatan lebih dulu, lalu memberikan setiap saran perawatan mandiri yang berlaku beserta kondisi di baliknya, dan membiarkan perawat di puskesmas mengajarinya — memilih node, opsi, dan kondisinya — setiap kali sarannya kurang atau keliru.

5.1 Pak Darmo dan Ners Ratna

Sudah tiga puluh tahun Pak Darmo mengayuh becak di kampung yang sama di Surabaya. Senin pagi ia bangun dengan badan panas dan pegal, dan sampai Rabu badannya masih panas. Istrinya berkata *ke puskesmas saja*; Pak Darmo menjawab bahwa becak yang tidak jalan tidak menghasilkan apa-apa, dan demam selalu turun sendiri. Di seberang jalan, anak tetangganya demam pada minggu yang sama dan akhirnya dirawat di rumah sakit karena demam berdarah.

Pilihan Pak Darmo adalah pilihan yang diambil jutaan orang setiap minggu: *apakah ini bisa saya rawat sendiri di rumah, atau saya perlu menemui seseorang?* Pertanyaan itu penting, karena penyakit yang umum dan yang berbahaya bermula dengan cara yang sama. Demam biasanya infeksi ringan yang lewat, dan kadang demam berdarah dengue: pada 2024 dunia mencatat lebih dari 14,6 juta kasus dengue dan lebih dari 12.000 kematian, dan tanda-tanda bahaya dengue berat — nyeri perut hebat, muntah yang tak berhenti, gusi atau hidung berdarah, darah dalam muntahan atau tinja — sering muncul justru *sesudah* demamnya turun [1]. Sakit kepala biasanya karena lelah atau kurang minum, dan kadang karena tekanan darah tinggi, yang diperkirakan dialami 1,4 miliar orang dewasa berusia 30 sampai 79 tahun pada 2024, 44 persen di antaranya tanpa menyadarinya [2]. Batuk biasanya selesma, dan kadang tuberkulosis, dan Indonesia menyumbang sekitar satu dari sepuluh kasus baru TBC dunia pada 2024, kedua setelah India [3]. Diare biasanya karena salah makan, dan juga salah satu pembunuh utama anak kecil di seluruh dunia, itulah sebabnya setiap tenaga kesehatan tahu nilai oralit [4].

Di puskesmas dua gang dari situ, Ners Ratna mengelola *posbindu* bulanan, pos tempat orang dewasa memeriksakan tekanan darah, berat badan, dan gula darahnya. Ia sudah menjawab pertanyaan Pak Darmo ribuan kali, dengan cara yang sama cermatnya: pertama tanda-tanda bahaya, yang berarti *berangkat sekarang*; lalu apa yang dapat dilakukan sendiri; dan selalu *kapan*

harus kembali. Ia ingin caranya yang cermat itu sampai kepada orang-orang yang tak pernah datang ke posbindu — tukang becak, pedagang pasar, juru parkir yang tidak bisa meninggalkan tempatnya. Dan ia ingin terus menyempurnakannya, satu pasien sungguhan demi satu pasien, karena setiap minggu ia bertemu keadaan yang tidak terduga oleh sarannya.

Bab ini membangun pemeriksaan mandiri itu. Inilah aplikasi paling berhati-hati dalam buku ini sejauh ini, dan ia mengajarkan bagian VCIRS yang berurusan dengan *kecermatan*: bagaimana seorang pakar menyempurnakan pengetahuan kasus demi kasus, memilih dengan tepat apa yang ditambahkan dan memeriksa apa yang akan diubahnya sebelum menambahkannya.

> **JEBAKAN:** Pemeriksaan mandiri memberikan saran perawatan mandiri dan memberi tahu kapan harus menemui tenaga kesehatan. Ia tidak mendiagnosis, tidak meresepkan obat, dan tidak pernah menggantikan dokter, perawat, atau bidan. Setiap saran dalam bab ini berakhir ke arah yang sama: bila ragu, atau bila keadaan memburuk, pergilah ke puskesmas atau telepon 119 [5].

5.2 Gagasannya dengan Kata-Kata Sederhana

5.2.1 Keselamatan dulu, dan tidak dipelajari

Aturan terpenting dalam pemeriksaan kesehatan mandiri sama sekali tidak dipelajari. Aturan-aturan itu ditulis sekali, di dalam program, dan ditanyakan sebelum apa pun:

- **Tanda stroke.** Wajah yang turun sebelah, lemah atau kebas mendadak pada satu sisi tubuh, bicara pelo atau tiba-tiba tidak dapat bicara, penglihatan kabur mendadak, sakit kepala mendadak yang lebih hebat daripada yang pernah dirasakan. Kementerian Kesehatan mengajarkannya dengan singkatan **SeGeRa Ke RS** — *senyum tidak simetris, gerak tubuh melemah, bicara pelo, kebas, rabun, sakit kepala hebat* — yang sekaligus menyatakan apa yang harus dilakukan: segera ke rumah sakit, karena pengobatan paling berhasil dalam sekitar empat setengah jam [6].
- **Nyeri atau tekanan di dada, atau sesak napas.**
- **Tanda bahaya dengue berat** pada orang yang demam: nyeri perut hebat, muntah tiga kali atau lebih sehari, gusi atau hidung berdarah, darah dalam muntahan atau tinja, sangat lemas atau gelisah. Orang dengan tanda-tanda ini membutuhkan IGD, bukan saran [1], [7].
- **Tanda kehilangan terlalu banyak cairan**: tidak dapat menahan minum, atau sangat sedikit buang air kecil [4].

Bila jawaban atas salah satunya *ya*, pemeriksaan mandiri hanya mengatakan satu hal — telepon **119**, nomor gawat darurat Kementerian Kesehatan, atau pergi ke instalasi gawat darurat (IGD) rumah sakit terdekat sekarang — dan tidak bertanya apa-apa lagi [5]. (Nomor darurat umum **112** juga menyambungkan ke pertolongan [8].) Pemeriksaan mandiri juga jujur tentang untuk siapa ia dibuat: untuk orang dewasa yang tidak sedang hamil. Untuk anak, atau selama kehamilan, ia langsung mengarahkan ke bidan atau puskesmas.

Mengapa aturan-aturan ini *tetap*, padahal inti VCIRS adalah pengetahuan yang tumbuh? Karena hal yang membuat belajar kasus demi kasus begitu kuat — pakar mana pun dapat mengubah jawaban untuk jenis kasus baru — justru hal yang tidak boleh pernah terjadi pada tanda bahaya. Perawat yang lelah di ujung hari panjang bisa saja mengajarkan pengecualian yang, tanpa ia sadari, mengarahkan sejenis nyeri dada ke “istirahat di rumah”. Basis pengetahuan yang dapat belajar juga dapat mempelajari kesalahan. Karena itu aturan keselamatan duduk *sebelum* basis pengetahuan, dalam kode biasa, di tempat yang tak terjangkau pengajaran, dan hanya berubah

bila programnya diubah dengan sengaja, dengan sumber-sumbernya diperiksa ulang. **Gambar 5.1** memperlihatkan susunannya.



Gambar 5.1 Keselamatan dulu. Pertanyaan keselamatan yang tetap datang sebelum basis pengetahuan; satu jawaban *ya* langsung mengarahkan orang mencari pertolongan. Baru setelah itu basis pengetahuan, yang diajarkan perawat kasus demi kasus, dimintai saran perawatan mandiri.

Sistem pakar kesehatan mandiri VCIRS yang asli, yang dibangun dari diagram-diagram sebuah buku kesehatan rumahan, mengambil pilihan serupa: pengetahuannya dikelompokkan menurut siapa pasiennya — anak-anak, orang dewasa, pria, wanita — karena keluhan yang sama membutuhkan penanganan berbeda pada tiap kelompok, dan pertanyaan untuk orang dewasa tidak boleh diajukan tentang bayi [9].

5.2.2 Pengetahuan yang tumbuh, kasus demi kasus

Di balik pertanyaan keselamatan ada basis pengetahuan VCIRS, dan di sini Ners Ratna adalah pakarnya. Awalnya kecil: *demam: istirahat, minum, parasetamol sesuai label; batuk: minum hangat, tutup mulut saat batuk*. Lalu, satu pasien sungguhan demi satu, ia menyempurnakannya. Demam yang sudah berlangsung dua hari atau lebih perlu diperiksa di puskesmas, karena pada saat itulah dengue harus disingkirkan [7]. Batuk dua minggu atau lebih perlu pemeriksaan TBC, dan pemeriksaan serta obat TBC di puskesmas gratis [10]. Sakit kepala dengan hasil ukur tekanan darah 140 atau lebih perlu diukur lagi pada hari lain, karena itulah batas yang dipakai WHO untuk menyebut tekanan darah tinggi bila ditemukan pada dua hari yang berbeda [2].

Inilah Ripple Down Rules seperti diuraikan Bab 3: basis pengetahuan dipakai pada kasus sungguhan; ketika jawabannya kurang atau keliru, pakar menambahkan pengetahuan dalam konteks jawaban itu; tidak ada yang pernah dihapus [11], [12]. Dan setiap kali jawabannya benar, perawat dapat menyatakannya, dan VCIRS mencatat pemeriksaan itu — *verifikasi sambil jalan* (*verification on the fly*), menurut istilah tesis [13].

Dua hal berbeda dari dokter Wi-Fi, dan keduanya berasal dari sifat kesehatan. Pertama, seseorang dapat punya beberapa keluhan sekaligus: Pak Darmo demam *dan* sakit kepala, dan keduanya butuh saran. Jadi pemeriksaan mandiri mengikuti setiap cabang yang berlaku, seperti Multiple Classification RDR [14], dan memberi satu saran per cabang. Kedua, harga sebuah aturan baru yang ceroboh lebih mahal, sehingga pakar membutuhkan kendali lebih atas apa yang ditambahkan. Kendali itulah pokok empat subbab berikut.

5.2.3 Kasus cornerstone dan daftar perbedaan

Setiap node menyimpan kasus yang membuatnya, **kasus cornerstone**-nya [11], [13]. Ketika sebuah kasus baru mendapat jawaban yang keliru atau kurang dari sebuah node, VCIRS

meletakkan kasus baru itu di samping kasus cornerstone node tersebut dan mendaftar perbedaannya, dalam dua bagian, seperti didefinisikan revisi 2006 [15], [16]:

- **Bagian A** memuat fakta yang hanya dimiliki kasus *lama*;
- **Bagian B** memuat fakta yang hanya dimiliki kasus *baru*.

Fakta yang dimiliki kedua kasus tidak didaftar: fakta seperti itu tidak dapat menjelaskan mengapa kedua kasus membutuhkan saran berbeda. Daftar perbedaan adalah lembar kerja pakar. Semua yang mungkin ia tulis ke dalam node baru ada di daftar itu atau, bila ia tahu sesuatu yang tidak tampak pada kasus-kasusnya, ditambahkan olehnya.

5.2.4 Enam opsi

Dengan daftar perbedaan di depannya, pakar memilih salah satu dari enam opsi [15], [16]. **Tabel 5.1** mendaftarnya, beserta pertanyaan yang dijawab masing-masing.

Tabel 5.1 Enam opsi pembangunan pengetahuan VCIRS, beserta pertanyaan yang dijawab masing-masing. Opsi 1 sampai 3 membuat pengecualian node lama; opsi 4 dan 5 menyimpan seluruh kasus sebagai node baru di tempat lain.

Opsi	Node baru memuat	Tempatnya	Pakai bila
0	tidak ada: jawabannya diterima	— (pemeriksaannya dicatat)	sarannya sudah benar
1	negasi bagian A	di bawah node lama, sebagai pengecualian	yang *dimiliki* kasus lama adalah alasan perbedaannya
2	bagian B	di bawah node lama, sebagai pengecualian	yang *dimiliki* kasus baru adalah alasannya
3	opsi 1 dan 2 bersama-sama	di bawah node lama, sebagai pengecualian	keduanya bagian dari alasan (yang dilakukan `learn`)
4	semua fakta kasus baru	di samping node lama, di bawah induk yang sama	kasus baru adalah keadaan lain pada tingkat yang sama
5	semua fakta kasus baru	di tingkat atas	tak ada bagian basis pengetahuan yang terkait

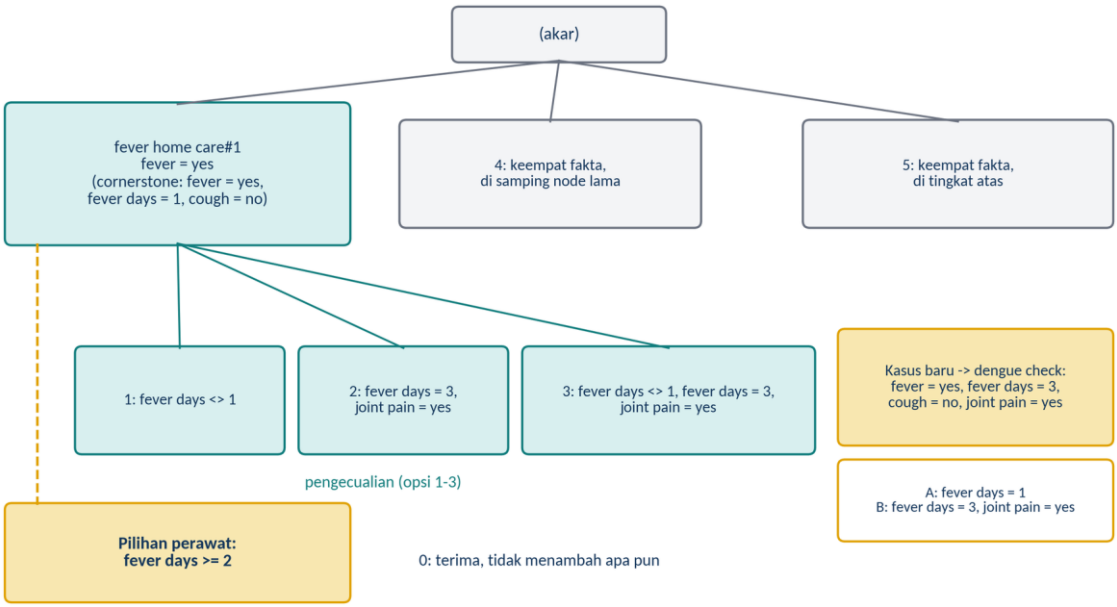
Opsi 1, 2, dan 3 adalah jantung Ripple Down Rules: node baru adalah pengecualian, hanya dicoba untuk kasus yang mencapai node lama, sehingga ia tidak dapat mengganggu yang lain. Opsi 3, kedua bagian bersama-sama, adalah pilihan aman yang paling spesifik, dan itulah yang dipakai `learn` sekali-panggil di Bab 3 dan 4. Opsi 4 dan 5 sama sekali tidak membuat pengecualian: keduanya menyimpan seluruh kasus sebagai node baru, di samping node lama atau di puncak, yang tepat ketika kasus baru bukan penghalusan saran lama melainkan keadaan yang lain.

5.2.5 Memilih kondisi sendiri

Keenam opsi membangun node dari fakta-fakta dua kasus, dan fakta itu spesifik: *fever days = 3*, *cough weeks = 1*. Seorang pakar biasanya tahu aturan di balik kasus itu. Ners Ratna tidak bermaksud "demam tepat tiga hari"; ia bermaksud "demam dua hari atau lebih". Karena itu VCIRS membolehkan pakar **memilih kondisinya sendiri**, dengan urutannya sendiri, alih-alih mengambil daftar opsi apa adanya [13]. Tesis sudah membolehkan pengguna mengurutkan kondisi-kondisinya; mesin melangkah selangkah lebih jauh dan menerima kondisi apa pun yang ia tulis.

Di sinilah seninya. Node yang **terlalu spesifik** — *fever days = 3, joint pain = yes* — akan benar untuk kasus yang membuatnya dan diam bagi hampir semua orang lain; basis pengetahuannya tumbuh, tetapi cakupannya tidak. Node yang **terlalu umum** — *fever days <> 1* — akan

menangkap kasus yang tak pernah dimaksudkan untuknya. **Gambar 5.2** memperlihatkan enam opsi dan pilihan perawat sendiri untuk satu kasus, dan Subbab 5.3 menghitung, untuk lima pasien berikutnya, siapa saja yang akan ditangkap tiap node.



Gambar 5.2 Apa yang akan dibangun tiap opsi untuk satu kasus di puskesmas. Opsi 1 sampai 3 menggantung pengecualian di bawah node lama; opsi 4 menaruh seluruh kasus di sampingnya dan opsi 5 di tingkat atas; pilihan perawat sendiri adalah satu kondisi yang menyatakan maksudnya.

5.2.6 Memeriksa apa yang akan diubah node baru

Sebuah pengecualian hanya dapat memengaruhi kasus yang mencapai induknya, tetapi *di antara* kasus-kasus itu ia masih dapat menangkap lebih banyak daripada yang dimaksud pakar. Jawaban RDR adalah melihat kasus-kasus cornerstone: sebelum node baru ditambahkan, sistem memeriksa setiap kasus cornerstone yang tersimpan terhadap seluruh jalur kondisi node baru, dan mendaftar kasus yang kini akan mendapat jawaban berbeda [11]. Inilah **cornerstone yang terdampak**. Pakar melihat masing-masing dan memutuskan: jawabannya memang sepatutnya berubah, atau ia menambahkan kondisi lain yang membedakannya, dan pemeriksaan dijalankan lagi. Untuk node yang dipilih dengan baik, daftarnya kosong.

Ini murah bagi mesin dan tak ternilai bagi pakar. Ia tidak pernah perlu membaca seluruh basis pengetahuan; ia hanya membaca beberapa kasus lama yang akan disentuh node barunya. Inilah sebabnya basis pengetahuan RDR dapat dirawat oleh para pakarnya sendiri, satu kasus demi satu kasus, bertahun-tahun [11], [17].

5.2.7 Tidak pernah menghapus, kadang menghentikan

Tidak ada apa pun dalam basis pengetahuan VCIRS yang pernah dihapus: node yang pernah ditambahkan tetap menyimpan kasus cornerstone-nya, pengecualian-pengecualiannya, dan tempatnya dalam riwayat [13]. Ketika sebuah saran menjadi keliru — obat ditarik, layanan pindah, pedoman berubah — pakar **menghentikan** (*stop*) node-nya. Node yang dihentikan tetap ikut dalam pencarian, dan pengecualian-pengecualiannya tetap bekerja, tetapi konklusinya sendiri tidak lagi diberikan. Saran yang lebih baik lalu diajarkan sebagai node baru. Soal 5.7 melakukan persis hal ini.

> **DEFINISI.** *Penyempurnaan pengetahuan kasus demi kasus* adalah perulangan tempat seorang pakar, untuk setiap kasus sungguhan yang jawabannya kurang atau keliru, membaca daftar

> perbedaan terhadap kasus cornerstone node yang menjawab, memilih salah satu dari enam opsi (dan, bila ia lebih tahu, kondisinya sendiri), memeriksa cornerstone yang terdampak, dan baru setelah itu menambahkan node baru. Jawaban yang benar dicatat sebagai pemeriksaan; tidak ada yang dihapus.

5.3 Langkah 1: Enam Opsi dengan Tangan

Sebelum membiarkan mesin mengerjakannya, mari kita hitung sendiri daftar perbedaan dan keenam opsinya untuk kasus pada **Gambar 5.2**. Node lamanya `fever home care#1`, yang kasus cornerstone-nya demam satu hari tanpa batuk. Kasus barunya demam tiga hari, tanpa batuk, dengan nyeri sendi, dan saran yang tepat adalah pemeriksaan dengue. **Kode 5.1** membangun setiap opsi, menambahkan pilihan Ners Ratna sendiri, dan mencoba setiap node yang dihasilkan pada lima pasien berikutnya.

Kode 5.1 `SixOptions.java`: daftar perbedaan dan enam opsi, dikerjakan dengan tangan.

```
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

/**
 * Daftar perbedaan dan enam opsi VCIRS, dikerjakan dengan tangan pada satu
 * kasus dari puskesmas: node apa yang dibangun tiap opsi, dan pasien mana
 * yang kelak akan ditangkap node itu?
 */
public class SixOptions {
    static final String[] OPERATORS = {" <= ", " >= ", " <> ", " = ", " < ",
        " > "};
    static final Map<String, String> NEGATION = Map.of("=", "<>", "<>", "=",
            "<", ">=", ">=", "<", ">", "<=", "<=", ">");

    static String[] split(String condition) {
        for (String op : OPERATORS) {
            int at = condition.indexOf(op);
            if (at > 0) {
                return new String[] {condition.substring(0, at), op.strip(),
                    condition.substring(at + op.length())};
            }
        }
        throw new IllegalArgumentException("no operator in " + condition);
    }

    static String negate(String condition) {
        String[] p = split(condition);
        return p[0] + " " + NEGATION.get(p[1]) + " " + p[2];
    }

    static boolean holds(String condition, Map<String, String> facts) {
        String[] p = split(condition);
        String fact = facts.get(p[0]);
        if (fact == null) {
            return false;
        }
        if (p[1].equals("=") || p[1].equals("<>")) {
            return fact.equals(p[2]) == p[1].equals("=");
        }
        double a = Double.parseDouble(fact);
        double b = Double.parseDouble(p[2]);
        switch (p[1]) {
            case "<": return a < b;
            case "<=": return a <= b;
```

Kode 5.1 (Lanjutan) *SixOptions.java: daftar perbedaan dan enam opsi, dikerjakan dengan tangan.*

```
            case ">": return a > b;
            default: return a >= b;
        }
    }

    static String show(List<String> conditions) {
        return conditions.isEmpty() ? "(none)" : String.join(", ", conditions);
    }

    record Option(String name, String place, List<String> conditions) { }

    static Map<String, String> facts(String... pairs) {
        Map<String, String> m = new LinkedHashMap<>();
        for (int i = 0; i < pairs.length; i += 2) {
            m.put(pairs[i], pairs[i + 1]);
        }
        return m;
    }

    public static void main(String[] args) {
        String oldId = "fever home care#1";
        List<String> oldCase = List.of("fever = yes", "fever days = 1",
                "cough = no");
        List<String> newCase = List.of("fever = yes", "fever days = 3",
                "cough = no", "joint pain = yes");
        List<String> a = new ArrayList<>();          // hanya pada kasus lama
        for (String c : oldCase) {
            if (!newCase.contains(c)) {
                a.add(c);
            }
        }
        List<String> b = new ArrayList<>();          // hanya pada kasus baru
        for (String c : newCase) {
            if (!oldCase.contains(c)) {
                b.add(c);
            }
        }
        List<String> negA = new ArrayList<>();
        for (String c : a) {
            negA.add(negate(c));
        }
        List<String> both = new ArrayList<>(negA);
        both.addAll(b);
```

Kode 5.1 (Lanjutan) *SixOptions.java: daftar perbedaan dan enam opsi, dikerjakan dengan tangan.*

```
        String under = "under " + oldId;             // pengecualian
        List<Option> options = List.of(
                new Option("Option 1", under, negA),
                new Option("Option 2", under, b),
                new Option("Option 3", under, both),
                new Option("Option 4", "beside " + oldId, newCase),
                new Option("Option 5", "at the top level", newCase),
                new Option("Nurse's choice", under, List.of("fever days >= 2")));

        System.out.println("Old node " + oldId + ", made by the case:");
        System.out.println("   " + show(oldCase));
        System.out.println("New case, whose right advice is dengue check:");
        System.out.println("   " + show(newCase));
        System.out.println("List of difference:");
        System.out.println("  A (only in the old case): " + show(a));
        System.out.println("  B (only in the new case): " + show(b));
        System.out.println();
        System.out.println("Option 0: nothing new; the old advice is accepted");
        for (Option o : options) {
            System.out.println(o.name() + ": " + o.place() + ": "
                    + show(o.conditions()));
        }

        Map<String, Map<String, String>> patients = new LinkedHashMap<>();
        patients.put("fever 2 days", facts("fever", "yes", "fever days", "2"));
        patients.put("3 days + sore joints", facts("fever", "yes",
                "fever days", "3", "joint pain", "yes", "cough", "no"));
        patients.put("fever 5 days", facts("fever", "yes", "fever days", "5"));
        patients.put("1 day + sore joints", facts("fever", "yes",
                "fever days", "1", "joint pain", "yes"));
        patients.put("fever since today", facts("fever", "yes",
                "fever days", "0"));
        System.out.println();
        System.out.println("Which later patients would the new node send to "
                + "dengue check?");
        for (Option o : options) {
            List<String> caught = new ArrayList<>();
            for (Map.Entry<String, Map<String, String>> p : patients.entrySet()) {
                boolean all = true;
                for (String c : o.conditions()) {
                    all = all && holds(c, p.getValue());
                }
                if (all) {
                    caught.add(p.getKey());
                }
            }
```

Kode 5.1 (Lanjutan) *SixOptions.java: daftar perbedaan dan enam opsi, dikerjakan dengan tangan.*

```
            System.out.println("  " + String.format("%-16s", o.name() + ":")
                    + (caught.isEmpty() ? "nobody" : String.join(", ", caught)));
        }
    }
}
```

Fungsi-fungsi pembantunya sama dengan Bab 4 dengan satu tambahan. `NEGATION` memetakan setiap operator ke lawannya (`=` ke `<>`, `>=` ke `<`, dan seterusnya), dan `negate` memakainya untuk mengubah sebuah kondisi menjadi negasinya: `fever days = 1` menjadi `fever days <> 1`. `holds` memperlakukan `=` dan `<>` sebagai perbandingan kata dan empat lainnya sebagai perbandingan angka.

`main` menghitung bagian A sebagai fakta kasus lama yang tidak dimiliki kasus baru, dan bagian B sebagai fakta kasus baru yang tidak dimiliki kasus lama, persis seperti definisinya. Keenam opsi

lalu dibangun dalam enam baris: opsi 1 dari negasi A, opsi 2 dari B, opsi 3 dari keduanya, opsi 4 dan 5 dari seluruh kasus baru, ditaruh di samping node lama atau di tingkat atas. Pilihan perawat adalah satu kondisi yang ia tulis sendiri. Terakhir, setiap node dicoba pada lima pasien yang mungkin datang kemudian — semuanya sudah demam, jadi semuanya mencapai node lama — dan program mencetak pasien mana yang akan dikirim tiap node ke pemeriksaan dengue. **Kode 5.2** adalah versi Python-nya.

Kode 5.2 `six_options.py`: perhitungan yang sama dalam Python.

```
"""Daftar perbedaan dan enam opsi VCIRS, dikerjakan dengan tangan pada
satu kasus dari puskesmas: node apa yang dibangun tiap opsi, dan pasien
mana yang kelak akan ditangkap node itu?"""

OPERATORS = [" <= ", " >= ", " <> ", " = ", " < ", " > "]
NEGATION = {"=": "<>", "<>": "=", "<": ">=", ">=": "<", ">": "<=", "<=": ">"}


def split(condition):
    for op in OPERATORS:
        at = condition.find(op)
        if at > 0:
            return [condition[:at], op.strip(), condition[at + len(op):]]
    raise ValueError("no operator in " + condition)


def negate(condition):
    var, op, value = split(condition)
    return var + " " + NEGATION[op] + " " + value


def holds(condition, facts):
    var, op, value = split(condition)
    if var not in facts:
        return False
    fact = facts[var]
    if op in ("=", "<>"):
        return (fact == value) == (op == "=")
    a, b = float(fact), float(value)
    return {"<": a < b, "<=": a <= b, ">": a > b, ">=": a >= b}[op]


def show(conditions):
    return ", ".join(conditions) if conditions else "(none)"


def main():
    old_id = "fever home care#1"
    old_case = ["fever = yes", "fever days = 1", "cough = no"]
    new_case = ["fever = yes", "fever days = 3", "cough = no", "joint pain = yes"]
    a = [c for c in old_case if c not in new_case]     # hanya pada kasus lama
    b = [c for c in new_case if c not in old_case]     # hanya pada kasus baru
```

Kode 5.2 (Lanjutan) *six_options.py: perhitungan yang sama dalam Python.*

```
    under = "under " + old_id                          # pengecualian
    options = [
        ("Option 1", under, [negate(c) for c in a]),
        ("Option 2", under, b),
        ("Option 3", under, [negate(c) for c in a] + b),
        ("Option 4", "beside " + old_id, new_case),
        ("Option 5", "at the top level", new_case),
        ("Nurse's choice", under, ["fever days >= 2"]),
    ]

    print("Old node " + old_id + ", made by the case:")
    print("  " + show(old_case))
    print("New case, whose right advice is dengue check:")
    print("  " + show(new_case))
    print("List of difference:")
    print("  A (only in the old case): " + show(a))
    print("  B (only in the new case): " + show(b))
    print()
    print("Option 0: nothing new; the old advice is accepted")
    for name, place, conditions in options:
        print(name + ": " + place + ": " + show(conditions))

    patients = [
        ("fever 2 days", {"fever": "yes", "fever days": "2"}),
        ("3 days + sore joints", {"fever": "yes", "fever days": "3",
                                  "joint pain": "yes", "cough": "no"}),
        ("fever 5 days", {"fever": "yes", "fever days": "5"}),
        ("1 day + sore joints", {"fever": "yes", "fever days": "1",
                                 "joint pain": "yes"}),
        ("fever since today", {"fever": "yes", "fever days": "0"}),
    ]
    print()
    print("Which later patients would the new node send to dengue check?")
    for name, _, conditions in options:
        caught = [p for p, facts in patients
                  if all(holds(c, facts) for c in conditions)]
        print("  " + (name + ":").ljust(16) + (", ".join(caught) or "nobody"))


if __name__ == "__main__":
    main()
```

Keduanya mencetak:

```
Old node fever home care#1, made by the case:
  fever = yes, fever days = 1, cough = no
New case, whose right advice is dengue check:
  fever = yes, fever days = 3, cough = no, joint pain = yes
List of difference:
  A (only in the old case): fever days = 1
  B (only in the new case): fever days = 3, joint pain = yes

Option 0: nothing new; the old advice is accepted
Option 1: under fever home care#1: fever days <> 1
Option 2: under fever home care#1: fever days = 3, joint pain = yes
Option 3: under fever home care#1: fever days <> 1, fever days = 3, joint pain = yes
Option 4: beside fever home care#1: fever = yes, fever days = 3, cough = no, joint pain =
yes
Option 5: at the top level: fever = yes, fever days = 3, cough = no, joint pain = yes
Nurse's choice: under fever home care#1: fever days >= 2

Which later patients would the new node send to dengue check?
  Option 1:       fever 2 days, 3 days + sore joints, fever 5 days, fever since today
  Option 2:       3 days + sore joints
  Option 3:       3 days + sore joints
  Option 4:       3 days + sore joints
  Option 5:       3 days + sore joints
  Nurse's choice: fever 2 days, 3 days + sore joints, fever 5 days
```

Bacalah blok terakhir perlahan-lahan, karena di sanalah seluruh pelajaran bab ini dalam enam baris.

Opsi 2, 3, 4, dan 5 masing-masing menangkap satu pasien: yang demam tiga hari dengan nyeri sendi — kasus baru itu sendiri. Semuanya *benar*: tidak ada yang mengirim orang ke pemeriksaan dengue padahal tidak perlu. Tetapi semuanya tak berguna bagi pasien yang demam dua hari, dan bagi yang demam lima hari, yang juga butuh pemeriksaan. Node-node ini **terlalu spesifik**: mereka mengingat satu kasus alih-alih mempelajari aturan.

Opsi 1, `fever days <> 1`, menangkap pasien yang demam dua, tiga, dan lima hari — dan juga pasien yang demamnya baru mulai *hari ini*, karena nol bukan satu. Ia **terlalu umum**: ia mempelajari "bukan satu hari" padahal pelajarannya "paling sedikit dua hari". Kesalahan ini tidak akan terlihat berminggu-minggu, sampai seseorang yang demam sejak pagi disuruh tes darah.

Pilihan perawat, `fever days >= 2`, menangkap tepat tiga pasien yang butuh pemeriksaan dengue dan tidak satu pun yang lain. Kondisi itu tidak ada di daftar perbedaan; ia adalah apa yang diketahui perawat. Daftar perbedaan memberi tahu *di mana* alasannya — pada jumlah hari demam — dan ia menuliskan *apa* alasannya.

Mengapa tidak membiarkan mesin memilih ambangnya sendiri? Dengan kasus yang cukup, mesin bisa: program yang sudah melihat seratus kasus demam, masing-masing diberi label *perlu pemeriksaan dengue* atau *tidak*, dapat mencari jumlah hari yang paling baik memisahkan kedua kelompok. Bab 12 melakukan persis itu, mempelajari aturan dari data lalu menyerahkannya kepada VCIRS. Tetapi perhatikan apa yang dibutuhkan program seperti itu: seratus kasus berlabel sebelum ia dapat mengatakan apa pun, dan labelnya sendiri berasal dari orang-orang seperti Ners Ratna. Di puskesmas yang baru memulai pemeriksaan mandirinya, tidak ada seratus kasus, dan ada seorang perawat yang sudah tahu jawabannya, dari pendidikannya, dari panduan Kementerian Kesehatan [7], dan dari pasien bertahun-tahun. Penyempurnaan kasus demi kasus membiarkannya memasukkan pengetahuan itu sejak hari pertama, dari kasus pertama, dan membiarkan setiap kasus berikutnya mengujinya. Kedua pendekatan itu bukan saingan. Data dapat mengusulkan ambang; pakar mengonfirmasinya, atau membetulkannya, dengan sebuah kasus di depannya.

Ada pula soal kepercayaan. Ketika sebuah node berbunyi `fever days >= 2`, perawat dapat membacanya, dokter dapat mencocokkannya dengan panduan, dan putri pasien dapat memahaminya. Ambang yang ditemukan program di dalam tabel angka mungkin sama baiknya, tetapi sampai seseorang membacanya dan mengaitkannya dengan sebuah alasan, ia hanyalah angka tanpa alasan. Dalam kesehatan, alasan adalah bagian dari saran.

> **WAWASAN KUNCI:** Daftar perbedaan menunjukkan kepada pakar ke mana harus melihat; opsi-opsi mengubahnya menjadi node secara mekanis; tetapi hanya pakar yang tahu aturan di balik kasusnya. Node terbaik sering dibangun dari satu fakta dalam daftar itu, ditulis ulang sebagai aturan yang sungguh ia pakai.

5.4 Langkah 2: Ners Ratna Mengajari Basis Pengetahuan

Sekarang pekerjaan yang sama di atas mesin, untuk satu sesi mengajar utuh. Dalam **Kode 5.3**, Ners Ratna membangun basis pengetahuan pemeriksaan mandiri dari sepuluh kasus sungguhan. Untuk tiap kasus, program memperlihatkan apa kata basis pengetahuan, daftar perbedaan, apa yang akan dibangun opsi pilihannya, kondisi yang ia simpan sebagai gantinya (bila ada), dan cornerstone yang terdampak; lalu ia menambahkan node-nya.

Kode 5.3 `NurseTeaches.java`: seorang perawat mengajari basis pengetahuan kasus demi kasus, di atas vcirs-core.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import vcirs.Case;
import vcirs.Conclusion;
import vcirs.Condition;
import vcirs.DifferenceList;
import vcirs.InferenceResult;
import vcirs.KnowledgeBase;
import vcirs.Node;
import vcirs.Report;

/**
 * Ners Ratna mengajari pemeriksaan kesehatan mandiri, kasus demi kasus.
 * Untuk tiap kasus ia melihat apa kata basis pengetahuan, membaca daftar
 * perbedaan, memilih opsi (dan, bila ia lebih tahu, kondisinya sendiri),
 * dan memeriksa kasus lama mana yang akan terdampak oleh node baru.
 */
public class NurseTeaches {
    static final int ACCEPT = KnowledgeBase.ACCEPT;
    static final int ADD_NEW = KnowledgeBase.ADD_NEW;
    static final int BOTH = KnowledgeBase.BOTH;
    static final int TOP_LEVEL = KnowledgeBase.TOP_LEVEL;

    static String show(List<Condition> conds) {
        List<String> parts = new ArrayList<>();
        for (Condition c : conds) {
            parts.add(c.toString());
        }
        return String.join(", ", parts);
    }

    static List<Condition> parse(List<String> texts) {
        List<Condition> out = new ArrayList<>();
        for (String t : texts) {
            out.add(Condition.parse(t));
        }
        return out;
    }
```

Kode 5.3 (Lanjutan) *NurseTeaches.java: seorang perawat mengajari basis pengetahuan kasus demi kasus, di atas vcirs-core.*

```
    /** Kondisi yang dibangun sebuah opsi, sebelum diubah perawat. */
    static List<Condition> builtBy(KnowledgeBase kb, int option, Node fired,
                                   Case c) {
        List<Condition> out = new ArrayList<>();
        if (option >= 1 && option <= 3) {
            DifferenceList d = kb.differences(fired.id, c);
            if (option == 1 || option == 3) {
                for (Condition x : d.a) {
                    out.add(x.negated());
                }
            }
            if (option == 2 || option == 3) {
                out.addAll(d.b);
            }
            return out;
        }
        out.addAll(c.facts);
        return out;
    }

    static void teach(KnowledgeBase kb, List<String> facts, String advice,
                      int option, List<String> keep, List<String> more) {
        Case c = Case.of(facts, List.of("advice = " + advice));
        System.out.println("Case: " + String.join(", ", facts) + " -> "
                + advice);
        InferenceResult result = kb.infer(c.known());
        Node fired = result.paths.isEmpty() ? null : result.paths.get(0).last();
        System.out.println("  knowledge base says: " + (fired != null
                ? fired.conclusions.get(0).value + " (" + fired.id + ")"
                : "nothing"));
        if (option == ACCEPT) {
            kb.refine(c, ACCEPT, fired.id, null, "Ratna");
            System.out.println("  option 0: " + KnowledgeBase.OPTION_TEXT.get(0));
            return;
        }
        if (fired != null) {
            for (String line : kb.differences(fired.id, c).lines()) {
                System.out.println("  " + line);
            }
        }
        List<Condition> conds = builtBy(kb, option, fired, c);
        System.out.println("  option " + option + " builds: " + show(conds));
        if (keep != null) {
            conds = parse(keep);
            System.out.println("  Ratna keeps instead: " + String.join(", ", keep));
        }
```

Kode 5.3 (Lanjutan) *NurseTeaches.java: seorang perawat mengajari basis pengetahuan kasus demi kasus, di atas vcirs-core.*

```
        String parent = option == TOP_LEVEL ? null : fired.id;
        List<String> hit = kb.affectedCornerstones(parent, conds, c.conclusions);
        System.out.println("  earlier cases whose advice would change: "
                + (hit.isEmpty() ? "none" : String.join(", ", hit)));
        if (!hit.isEmpty() && more != null) {
            conds.addAll(parse(more));
            System.out.println("  Ratna adds: " + String.join(", ", more));
            hit = kb.affectedCornerstones(parent, conds, c.conclusions);
            System.out.println("  earlier cases whose advice would change: "
                    + (hit.isEmpty() ? "none" : String.join(", ", hit)));
        }
        Node node = kb.refine(c, option, parent, conds, "Ratna");
        System.out.println("  new node " + node);
    }

    static void teachAll(KnowledgeBase kb) {
        teach(kb, List.of("fever = yes"), "fever home care", TOP_LEVEL, null,
                null);
        teach(kb, List.of("cough = yes", "cough weeks = 1"), "cough home care",
                TOP_LEVEL, List.of("cough = yes"), null);
        teach(kb, List.of("diarrhea = yes"), "ORS at home", TOP_LEVEL, null,
                null);
        teach(kb, List.of("headache = yes"), "headache home care", TOP_LEVEL,
                null, null);
        teach(kb, List.of("fever = yes", "fever days = 3"), "dengue check",
                ADD_NEW, List.of("fever days >= 2"), null);
        teach(kb, List.of("fever = yes", "fever days = 1"), "fever home care",
                ACCEPT, null, null);
        teach(kb, List.of("cough = yes", "cough weeks = 3", "smoker = yes"),
                "TB check", BOTH, List.of("cough weeks >= 2"), null);
        teach(kb, List.of("cough = yes", "cough weeks = 1", "smoker = yes"),
                "quit smoking help", ADD_NEW, null, List.of("cough weeks < 2"));
        teach(kb, List.of("headache = yes", "blood pressure = 150"),
                "blood pressure check", ADD_NEW,
                List.of("blood pressure >= 140"), null);
        teach(kb, List.of("diarrhea = yes", "diarrhea days = 4"),
                "diarrhea check", ADD_NEW, List.of("diarrhea days >= 3"), null);
    }
```

Kode 5.3 (Lanjutan) *NurseTeaches.java: seorang perawat mengajari basis pengetahuan kasus demi kasus, di atas vcirs-core.*

```
    public static void main(String[] args) {
        KnowledgeBase kb = new KnowledgeBase("Health self-check");
        teachAll(kb);
        System.out.println();
        for (String line : Report.tree(kb)) {
            System.out.println(line);
        }
        int kept = 0;
        for (Node n : kb.nodes.values()) {
            for (Conclusion c : kb.infer(n.cornerstone.known()).conclusions()) {
                if (c.same(n.conclusions.get(0))) {
                    kept++;
                    break;
                }
            }
        }
        System.out.println("Cornerstone cases that still get their own advice: "
                + kept + " of " + kb.nodes.size());
        List<String> used = new ArrayList<>();
        for (Map<String, Object> h : kb.history) {
            used.add(String.valueOf(h.get("option")));
        }
        System.out.println("Options used: " + String.join(", ", used));
    }
}
```

`builtBy` adalah perhitungan enam opsi dari Langkah 1 yang dikerjakan dengan potongan-potongan milik mesin sendiri: `kb.differences(fired.id, c)` mengembalikan daftar perbedaan sebagai `DifferenceList` dengan bagian `a` dan `b`, dan `negated()` menegasikan sebuah kondisi. `teach` adalah satu langkah mengajar. Ia menanyakan jawaban basis pengetahuan dengan `kb.infer(c.known())`, fakta-fakta kasus yang diketahui; menerimanya dengan opsi 0 bila jawabannya benar; selain itu mencetak daftar perbedaan dan apa yang akan dibangun opsinya, mengganti kondisinya dengan kondisi perawat sendiri bila ia memberikannya (`keep`), dan meminta mesin mendaftar cornerstone yang terdampak dengan `kb.affectedCornerstones(parent, conds, c.conclusions)`. Bila daftarnya tidak kosong dan ia sudah menyiapkan kondisi lain (`more`), kondisi itu ditambahkan dan diperiksa lagi. Terakhir `kb.refine(c, option, parent, conds, "Ratna")` menambahkan node-nya. `refine` adalah panggilan umum mesin untuk membangun pengetahuan: sebuah kasus, salah satu dari enam opsi, node lama, kondisi pilihan, dan nama pakarnya [13], [15].

`teachAll` adalah sesinya sendiri — sepuluh kasus sesuai urutan Ners Ratna menjumpainya — dan `main` menjalankannya, mencetak pohonnya, memeriksa bahwa setiap kasus cornerstone masih mendapat sarannya sendiri, dan mendaftar opsi-opsi yang dipakai, sebagaimana dicatat riwayat. **Kode 5.4** adalah versi Python-nya.

Kode 5.4 `nurse_teaches.py`: sesi mengajar yang sama dalam Python.

```
"""Ners Ratna mengajari pemeriksaan kesehatan mandiri, kasus demi kasus.
Untuk tiap kasus ia melihat apa kata basis pengetahuan, membaca daftar
perbedaan, memilih opsi (dan, bila ia lebih tahu, kondisinya sendiri), dan
memeriksa kasus lama mana yang akan terdampak oleh node baru."""

import pathlib
import sys

# Beri tahu Python letak mesin vcirs-core (naik dua folder, lalu masuk).
sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import (ACCEPT, ADD_NEW, BOTH, OPTION_TEXT, TOP_LEVEL, Case,
                   Condition, KnowledgeBase, report)


def built_by(kb, option, fired, case):
    """Kondisi yang dibangun sebuah opsi, sebelum diubah perawat."""
    if option in (1, 2, 3):
        d = kb.differences(fired.id, case)
        neg = [c.negated() for c in d.a] if option in (1, 3) else []
        return neg + (d.b if option in (2, 3) else [])
    return list(case.facts)


def teach(kb, facts, advice, option, keep=None, more=None):
    case = Case.of(facts, ["advice = " + advice])
    print("Case: " + ", ".join(facts) + " -> " + advice)
    result = kb.infer(case)
    fired = result.paths[0].last() if result.paths else None
    print("  knowledge base says: " + (fired.conclusions[0].value + " ("
          + fired.id + ")" if fired else "nothing"))
    if option == ACCEPT:
        kb.refine(case, ACCEPT, fired.id, author="Ratna")
        print("  option 0: " + OPTION_TEXT[0])
        return
    if fired is not None:
        for line in kb.differences(fired.id, case).lines():
            print("  " + line)
    conds = built_by(kb, option, fired, case)
    print("  option " + str(option) + " builds: "
          + ", ".join(str(c) for c in conds))
    if keep is not None:
        conds = [Condition.parse(c) for c in keep]
        print("  Ratna keeps instead: " + ", ".join(keep))
    parent = None if option == TOP_LEVEL else fired.id
    hit = kb.affected_cornerstones(parent, conds, case.conclusions)
```

Kode 5.4 (Lanjutan) *nurse_teaches.py: sesi mengajar yang sama dalam Python.*

```
    print("  earlier cases whose advice would change: "
          + (", ".join(hit) if hit else "none"))
    if hit and more:
        conds += [Condition.parse(c) for c in more]
        print("  Ratna adds: " + ", ".join(more))
        hit = kb.affected_cornerstones(parent, conds, case.conclusions)
        print("  earlier cases whose advice would change: "
              + (", ".join(hit) if hit else "none"))
    node = kb.refine(case, option, parent, chosen=conds, author="Ratna")
    print("  new node " + str(node))


def teach_all(kb):
    teach(kb, ["fever = yes"], "fever home care", TOP_LEVEL)
    teach(kb, ["cough = yes", "cough weeks = 1"], "cough home care",
          TOP_LEVEL, keep=["cough = yes"])
    teach(kb, ["diarrhea = yes"], "ORS at home", TOP_LEVEL)
    teach(kb, ["headache = yes"], "headache home care", TOP_LEVEL)
    teach(kb, ["fever = yes", "fever days = 3"], "dengue check", ADD_NEW,
          keep=["fever days >= 2"])
    teach(kb, ["fever = yes", "fever days = 1"], "fever home care", ACCEPT)
    teach(kb, ["cough = yes", "cough weeks = 3", "smoker = yes"], "TB check",
          BOTH, keep=["cough weeks >= 2"])
    teach(kb, ["cough = yes", "cough weeks = 1", "smoker = yes"],
          "quit smoking help", ADD_NEW, more=["cough weeks < 2"])
    teach(kb, ["headache = yes", "blood pressure = 150"],
          "blood pressure check", ADD_NEW, keep=["blood pressure >= 140"])
    teach(kb, ["diarrhea = yes", "diarrhea days = 4"], "diarrhea check",
          ADD_NEW, keep=["diarrhea days >= 3"])


if __name__ == "__main__":
    kb = KnowledgeBase("Health self-check")
    teach_all(kb)
    print()
    for line in report.tree(kb):
        print(line)
    kept = [n for n in kb.nodes.values()
            if any(c.same(n.conclusions[0])
                   for c in kb.infer(n.cornerstone).conclusions())]
    print("Cornerstone cases that still get their own advice: "
          + str(len(kept)) + " of " + str(len(kb.nodes)))
    used = [h["option"] for h in kb.history]
    print("Options used: " + ", ".join(str(o) for o in used))
```

Keduanya mencetak:

```
Case: fever = yes -> fever home care
  knowledge base says: nothing
  option 5 builds: fever = yes
  earlier cases whose advice would change: none
  new node fever home care#1: fever = yes -> advice = fever home care
Case: cough = yes, cough weeks = 1 -> cough home care
  knowledge base says: nothing
  option 5 builds: cough = yes, cough weeks = 1
  Ratna keeps instead: cough = yes
  earlier cases whose advice would change: none
  new node cough home care#1: cough = yes -> advice = cough home care
Case: diarrhea = yes -> ORS at home
  knowledge base says: nothing
  option 5 builds: diarrhea = yes
  earlier cases whose advice would change: none
  new node ORS at home#1: diarrhea = yes -> advice = ORS at home
Case: headache = yes -> headache home care
  knowledge base says: nothing
  option 5 builds: headache = yes
  earlier cases whose advice would change: none
  new node headache home care#1: headache = yes -> advice = headache home care
Case: fever = yes, fever days = 3 -> dengue check
  knowledge base says: fever home care (fever home care#1)
  Differences with the cornerstone case of fever home care#1:
    A (only in the old case): (none)
    B (only in the new case): fever days = 3
  option 2 builds: fever days = 3
  Ratna keeps instead: fever days >= 2
  earlier cases whose advice would change: none
  new node dengue check#1: fever days >= 2 -> advice = dengue check
Case: fever = yes, fever days = 1 -> fever home care
  knowledge base says: fever home care (fever home care#1)
  option 0: accept the existing knowledge (inference only)
Case: cough = yes, cough weeks = 3, smoker = yes -> TB check
  knowledge base says: cough home care (cough home care#1)
  Differences with the cornerstone case of cough home care#1:
    A (only in the old case): cough weeks = 1
    B (only in the new case): cough weeks = 3, smoker = yes
  option 3 builds: cough weeks <> 1, cough weeks = 3, smoker = yes
  Ratna keeps instead: cough weeks >= 2
  earlier cases whose advice would change: none
  new node TB check#1: cough weeks >= 2 -> advice = TB check
Case: cough = yes, cough weeks = 1, smoker = yes -> quit smoking help
  knowledge base says: cough home care (cough home care#1)
  Differences with the cornerstone case of cough home care#1:
    A (only in the old case): (none)
    B (only in the new case): smoker = yes
  option 2 builds: smoker = yes
  earlier cases whose advice would change: TB check#1
  Ratna adds: cough weeks < 2
  earlier cases whose advice would change: none
  new node quit smoking help#1: smoker = yes, cough weeks < 2 -> advice = quit smoking help
```

```
Case: headache = yes, blood pressure = 150 -> blood pressure check
  knowledge base says: headache home care (headache home care#1)
  Differences with the cornerstone case of headache home care#1:
    A (only in the old case): (none)
    B (only in the new case): blood pressure = 150
  option 2 builds: blood pressure = 150
  Ratna keeps instead: blood pressure >= 140
  earlier cases whose advice would change: none
  new node blood pressure check#1: blood pressure >= 140 -> advice = blood pressure check
Case: diarrhea = yes, diarrhea days = 4 -> diarrhea check
  knowledge base says: ORS at home (ORS at home#1)
  Differences with the cornerstone case of ORS at home#1:
    A (only in the old case): (none)
    B (only in the new case): diarrhea days = 4
  option 2 builds: diarrhea days = 4
  Ratna keeps instead: diarrhea days >= 3
  earlier cases whose advice would change: none
  new node diarrhea check#1: diarrhea days >= 3 -> advice = diarrhea check

(root)
+- fever home care#1: fever = yes -> advice = fever home care
|  +- dengue check#1: fever days >= 2 -> advice = dengue check
+- cough home care#1: cough = yes -> advice = cough home care
|  +- TB check#1: cough weeks >= 2 -> advice = TB check
|  +- quit smoking help#1: smoker = yes, cough weeks < 2 -> advice = quit smoking help
+- ORS at home#1: diarrhea = yes -> advice = ORS at home
|  +- diarrhea check#1: diarrhea days >= 3 -> advice = diarrhea check
+- headache home care#1: headache = yes -> advice = headache home care
   +- blood pressure check#1: blood pressure >= 140 -> advice = blood pressure check
Cornerstone cases that still get their own advice: 9 of 9
Options used: 5, 5, 5, 5, 2, 0, 3, 2, 2, 2
```

Ikutilah sesi ini bersama Ners Ratna.

Kasus 1 sampai 4 memulai basis pengetahuan. Setiap kali basis pengetahuan berkata *nothing* (tidak ada), dan ia memakai opsi 5, node tingkat atas baru, satu per keluhan. Pada kasus 2 opsinya akan membangun `cough = yes, cough weeks = 1`, tetapi ia hanya menyimpan `cough = yes`: batuk berapa lama pun layak mendapat saran perawatan di rumah, dan lamanya termasuk urusan pengecualian. Kasus cornerstone-nya tetap mengingat seluruh kasus, batuk satu minggu, dan itu akan penting pada kasus 7.

Kasus 5 adalah Langkah 1 lagi, dengan mesin. Basis pengetahuan berkata *fever home care*; daftar perbedaannya punya bagian A yang kosong (cornerstone `fever home care#1` hanya `fever = yes`) dan `fever days = 3` di bagian B; opsi 2 akan membangun persis itu; ia menyimpan `fever days >= 2` sebagai gantinya. Tidak ada kasus lama yang sarannya berubah, dan `dengue check#1` digantung di bawah `fever home care#1`.

Kasus 6 adalah demam satu hari, dan basis pengetahuan memberi saran perawatan di rumah, yang benar. Ia menerimanya: opsi 0. Tidak ada yang ditambahkan, tetapi riwayat kini menunjukkan bahwa node ini sudah diperiksa pada pasien sungguhan.

Kasus 7 adalah perokok dengan batuk tiga minggu, yang butuh pemeriksaan TBC. Kasus cornerstone `cough home care#1` punya batuk satu minggu, jadi kali ini bagian A tidak kosong: `cough weeks = 1`. Opsi 3 akan membangun `cough weeks <> 1, cough weeks = 3, smoker = yes` — negasi A dan seluruh B — yang hanya akan menangkap perokok dengan batuk tepat tiga minggu. Ners Ratna menyimpan `cough weeks >= 2`: lamanya adalah alasannya, merokok bukan.

Kasus 8 adalah tempat pemeriksaan cornerstone terdampak membuktikan gunanya. Perokok dengan batuk satu minggu sebaiknya ditawari bantuan berhenti merokok, di puskesmas atau lewat layanan konseling gratis Kementerian Kesehatan [18]. Opsi 2 membangun `smoker = yes`,

yang tampak benar. Tetapi mesin mendapati bahwa kasus cornerstone `TB check#1` — perokok dengan batuk tiga minggu dari kasus 7 — juga akan tertangkap node baru, dan akan diberi bantuan berhenti merokok sebagai ganti, atau di samping, pemeriksaan TBC yang ia butuhkan. Ia menambahkan `cough weeks < 2`, pemeriksaan dijalankan lagi, dan daftarnya kosong. Node baru kini berbunyi *perokok, dan batuk kurang dari dua minggu*: batuk yang lama tetap ke pemeriksaan TBC.

Kasus 9 dan 10 mengikuti pola kasus 5: sakit kepala dengan hasil ukur tekanan darah 150 mengajarkan `blood pressure >= 140` [2], dan diare empat hari mengajarkan `diarrhea days >= 3`.

Tiga bagian terakhir keluaran adalah bukti bahwa tidak ada yang rusak. Pohonnya punya empat node tingkat atas dan lima pengecualian, masing-masing hanya memuat yang ditambahkannya. Setiap dari sembilan kasus cornerstone, bila dijalankan pada basis pengetahuan yang sudah jadi, masih mendapat sarannya sendiri — jaminan Ripple Down Rules, diperiksa oleh mesin. Dan riwayat mendaftar opsi tepat seperti dipakai: empat kali opsi 5, sekali opsi 0, sekali opsi 3, dan empat kali opsi 2, dengan urutan itu. Inilah catatan yang dapat ditunjukkan Ners Ratna kepada kepala puskesmasnya: siapa mengajarkan apa, bagaimana, dan kapan.

5.5 Cek Kesehatan Mandiri

5.5.1 Titik awal aplikasi

Aplikasi mulai dari pengetahuan bulan pertama Ners Ratna, yang ditulis sebagai sembilan kasus awal dalam `health-seed-cases.json`:

```
{
  "name": "Health self-check",
  "description": "Self-care advice for adults, taught by Ners Ratna at the puskesmas. Not
a diagnosis.",
  "cases": [
    {"facts": ["fever = yes"], "conclusions": ["advice = fever home care"], "option": 5,
"author": "Ratna"},
    {"facts": ["cough = yes"], "conclusions": ["advice = cough home care"], "option": 5,
"author": "Ratna"},
    {"facts": ["diarrhea = yes"], "conclusions": ["advice = ORS at home"], "option": 5,
"author": "Ratna"},
    {"facts": ["headache = yes"], "conclusions": ["advice = headache home care"],
"option": 5, "author": "Ratna"},
    {"facts": ["fever = yes", "fever days >= 2"], "conclusions": ["advice = dengue
check"], "option": 2, "node": "fever home care#1", "author": "Ratna"},
    {"facts": ["cough = yes", "cough weeks >= 2"], "conclusions": ["advice = TB check"],
"option": 2, "node": "cough home care#1", "author": "Ratna"},
    {"facts": ["cough = yes", "smoker = yes", "cough weeks < 2"], "conclusions": ["advice
= quit smoking help"], "option": 2, "node": "cough home care#1", "author": "Ratna"},
    {"facts": ["headache = yes", "blood pressure >= 140"], "conclusions": ["advice =
blood pressure check"], "option": 2, "node": "headache home care#1", "author": "Ratna"},
    {"facts": ["diarrhea = yes", "diarrhea days >= 3"], "conclusions": ["advice =
diarrhea check"], "option": 2, "node": "ORS at home#1", "author": "Ratna"}
  ]
}
```

Berkas ini membangun pohon yang sama dengan sesi mengajar di Subbab 5.4. Di tempat perawat memilih kondisinya, kasus awal menyatakannya langsung — `fever days >= 2` — dan menyebut opsinya (2) serta node yang dihaluskannya (`"node": "fever home care#1"`), sehingga mesin membangun persis pengecualian yang ia pilih.

Konklusinya berupa label pendek, seperti `dengue check`, bukan kalimat utuh. Ini menjaga nama node tetap terbaca (`dengue check#1`) dan membuat puskesmas dapat mengubah *kata-kata* sebuah saran tanpa menyentuh *pengetahuannya*. Kata-kata itu tinggal di berkas terpisah, `advice-texts.txt`, satu baris per label:

```
# The wording of each piece of advice, kept apart from the knowledge.
# One line per advice: label = text. Lines starting with # are notes.
fever home care = Rest, drink plenty of water, and take paracetamol as written on the
label. Measure your temperature morning and evening.
dengue check = A fever of two days or more needs a check: go to the puskesmas today and
ask whether a blood test is needed. Until then take paracetamol only, not ibuprofen or
aspirin.
cough home care = Rest, drink warm water, and cover your mouth when you cough. Wear a
mask near babies and old people.
TB check = A cough of two weeks or more should be checked at the puskesmas, including for
TB. TB tests and medicine there are free.
quit smoking help = Smoking keeps a cough going. Ask the puskesmas about its quit-smoking
service, or call the free quitline 0-800-177-6565.
ORS at home = Drink oralit (ORS) after every loose stool, keep eating small meals, and
wash your hands with soap.
diarrhea check = Diarrhea for three days or more needs a check at the puskesmas.
report food poisoning = Several people sick after the same food may be food poisoning:
tell the puskesmas today, so that others can be warned.
headache home care = Rest, drink water, and take paracetamol as written on the label.
blood pressure check = A top number of 140 or more is high. Measure again on another day
at the posbindu or puskesmas; high blood pressure needs a doctor's care.
```

Ketika seorang dokter di puskesmas meminta sebuah frasa diubah, perawat menyunting satu baris berkas ini; basis pengetahuan, cornerstone-nya, dan riwayatnya tetap seperti semula. (Kata-kata saran di sini dalam bahasa Inggris agar kodenya sama untuk kedua edisi; di puskesmas sungguhan, berkas inilah yang ditulis dalam bahasa pasien.)

5.5.2 Programnya

Kode 5.5 adalah pemeriksaan mandiri itu dalam Java.

Kode 5.5 `HealthCheck.java`: cek kesehatan mandiri, keselamatan dulu, di atas vcirs-core.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

import vcirs.Case;
import vcirs.CaseFile;
import vcirs.Conclusion;
import vcirs.Condition;
import vcirs.DifferenceList;
import vcirs.Fact;
import vcirs.Facts;
import vcirs.InferenceResult;
import vcirs.KnowledgeBase;
import vcirs.Node;
import vcirs.Question;
import vcirs.RdrPath;
import vcirs.Report;
import vcirs.Variable;
import vcirs.VcirsException;

/**
 * Pemeriksaan kesehatan mandiri: keselamatan dulu, lalu saran perawatan
 * mandiri yang diajarkan perawat kasus demi kasus. Ini bukan diagnosis.
 * java HealthCheck [--echo] [--nurse] [--tree] [knowledge-base.json]
 * Pertanyaan keselamatan datang lebih dulu dan tetap di dalam program: bila
 * salah satunya dijawab ya, program langsung mengarahkan orang itu mencari
 * pertolongan dan tidak bertanya lagi. Baru setelah itu program
 * berkonsultasi dengan basis pengetahuan, mengikuti setiap saran yang
 * berlaku. Dengan --nurse, perawat memeriksa saran dan mengajari basis
 * pengetahuan: ia memilih node, opsi, dan kondisinya, dan melihat kasus lama
 * mana yang akan terdampak.
 */
public class HealthCheck {
    // Keselamatan dulu: aturan tetap yang tak dapat diubah pengajaran.
    static final String ADULT = "Is this self-check for an adult (18 or older) "
            + "who is not pregnant?";
    static final String NOT_ADULT = "This self-check is for adults who are not "
            + "pregnant. For a child, or during pregnancy, see the bidan or the "
            + "puskesmas.";
```

Kode 5.5 (Lanjutan) *HealthCheck.java: cek kesehatan mandiri, keselamatan dulu, di atas vcirs-core.*

```
    static final String[][] RED_FLAGS = {
        {"Right now: sudden weakness or numbness on one side, a drooping face, "
            + "slurred speech, sudden trouble seeing, or the worst headache ever?",
            "EMERGENCY: these can be signs of a stroke. Call 119 or go to the "
            + "nearest hospital emergency room (IGD) now."},
        {"Chest pain or pressure, or trouble breathing?",
            "EMERGENCY: call 119 or go to the IGD now."},
        {"Fever with any of: severe belly pain, vomiting 3 or more times a day, "
            + "bleeding gums or nose, blood in vomit or stool, great weakness?",
            "URGENT: these are warning signs of severe dengue. Go to the IGD now."},
        {"Unable to keep water down, or passing very little urine?",
            "URGENT: the body may be losing too much water. Go to the puskesmas "
            + "or the IGD now."},
    };
    static final String[][] QUESTIONS = {
        {"fever", "Do you have a fever?", "yes", "no"},
        {"fever days", "For how many days have you had the fever?"},
        {"cough", "Do you have a cough?", "yes", "no"},
        {"cough weeks", "For how many weeks have you been coughing?"},
        {"smoker", "Do you smoke?", "yes", "no"},
        {"diarrhea", "Diarrhea (three or more loose stools a day)?", "yes", "no"},
        {"diarrhea days", "For how many days have you had diarrhea?"},
        {"others ill", "Are others who ate the same food sick too?", "yes", "no"},
        {"headache", "Do you have a headache?", "yes", "no"},
        {"blood pressure", "Top number of your blood pressure, if measured today?"},
    };

    static BufferedReader in = new BufferedReader(
            new InputStreamReader(System.in, StandardCharsets.UTF_8));
    static boolean echo = false;
    static KnowledgeBase kb;

    /** Pecah teks menjadi baris paling panjang width karakter, di spasi. */
    static List<String> wrap(String text, int width) {
        List<String> lines = new ArrayList<>();
        String line = "";
        for (String word : text.split(" ", -1)) {
            if (!line.isEmpty() && line.length() + 1 + word.length() > width) {
                lines.add(line);
                line = word;
            } else {
                line = line.isEmpty() ? word : line + " " + word;
            }
        }
        lines.add(line);
        return lines;
```

Kode 5.5 (Lanjutan) *HealthCheck.java: cek kesehatan mandiri, keselamatan dulu, di atas vcirs-core.*

```
    }

    /** Cetak teks terbungkus: baris pertama sesudah first, sisanya sesudah rest. */
    static void show(String first, String rest, String text) {
        String body = text.stripLeading();
        String lead = " ".repeat(text.length() - body.length());  // indennya
        List<String> lines = wrap(body, 72);
        for (int i = 0; i < lines.size(); i++) {
            System.out.println((i == 0 ? first + lead : rest) + lines.get(i));
        }
    }

    /** Membaca jawaban pengguna; bila echo aktif, mencetaknya juga. */
    static String read(String prompt) {
        String bare = prompt.replaceAll("[ >]+$", "");
        List<String> lines = wrap(bare, 74);              // pertanyaan panjang
        for (int i = 0; i < lines.size() - 1; i++) {
            System.out.println(lines.get(i));
        }
        System.out.print(lines.get(lines.size() - 1) + " > ");
        String line;
        try {
            line = in.readLine();
        } catch (IOException e) {
            line = null;
        }
        String text = line == null ? "" : line.strip();
        if (echo) {
            System.out.println(text);
        }
        return text;
    }

    static boolean yes(String question) {
        return read(question + " (yes/no) > ").equals("yes");
    }
```

Kode 5.5 (Lanjutan) *HealthCheck.java: cek kesehatan mandiri, keselamatan dulu, di atas vcirs-core.*

```
    static Fact ask(Question q) {
        Variable var = kb.variables.get(q.variable);
        String text = q.variable;
        List<String> choices = q.choices;
        if (var != null && !var.question.isEmpty()) {
            text = var.question;
            choices = var.choices;
        }
        if (!choices.isEmpty()) {
            text += " (" + String.join("/", choices) + ")";
        }
        String answer = read(text + " > ");
        return answer.isEmpty() ? null : new Fact(q.variable, answer); // Enter
    }

    /** advice-texts.txt: satu "label = teks" per baris. */
    static Map<String, String> readTexts(Path path) {
        Map<String, String> texts = new LinkedHashMap<>();
        try {
            for (String line : Files.readAllLines(path, StandardCharsets.UTF_8)) {
                int at = line.indexOf('=');
                if (at >= 0 && !line.startsWith("#")) {
                    texts.put(line.substring(0, at).strip(),
                            line.substring(at + 1).strip());
                }
            }
        } catch (IOException e) {
            throw new VcirsException("cannot read " + path);
        }
        return texts;
    }

    /** Muat basis pengetahuan, atau bangun dari kasus-kasus awal. */
    static KnowledgeBase openKb(String path) {
        if (Files.exists(Path.of(path))) {
            return KnowledgeBase.load(path);
        }
        Path parent = Path.of(path).toAbsolutePath().getParent();
        CaseFile seed = CaseFile.read(parent.resolve("health-seed-cases.json")
                .toString());
        KnowledgeBase k = new KnowledgeBase(seed.name, seed.description);
        for (String[] q : QUESTIONS) {
            List<String> choices = new ArrayList<>();
            for (int i = 2; i < q.length; i++) {
                choices.add(q[i]);
            }
```

Kode 5.5 (Lanjutan) *HealthCheck.java: cek kesehatan mandiri, keselamatan dulu, di atas vcirs-core.*

```
            k.declare(q[0], q[1], choices);
        }
        seed.build(k);
        k.save(path);
        return k;
    }

    /** Pertanyaan keselamatan tetap. true bila saran mandiri boleh menyusul. */
    static boolean safe() {
        if (!yes(ADULT)) {
            show("", "", NOT_ADULT);
            return false;
        }
        for (String[] flag : RED_FLAGS) {
            if (yes(flag[0])) {
                show("", "", flag[1]);
                return false;
            }
        }
        return true;
    }

    static String join(List<Condition> conds) {
        List<String> parts = new ArrayList<>();
        for (Condition c : conds) {
            parts.add(c.toString());
        }
        return String.join(", ", parts);
    }

    /** Kondisi yang dibangun sebuah opsi dari daftar perbedaan. */
    static List<Condition> builtBy(int option, Node node, Case c) {
        List<Condition> out = new ArrayList<>();
        if (option >= 1 && option <= 3) {
            DifferenceList d = kb.differences(node.id, c);
            if (option == 1 || option == 3) {
                for (Condition x : d.a) {
                    out.add(x.negated());
                }
            }
            if (option == 2 || option == 3) {
                out.addAll(d.b);
            }
            return out;
        }
        out.addAll(c.facts);                          // opsi 4 dan 5
```

Kode 5.5 (Lanjutan) *HealthCheck.java: cek kesehatan mandiri, keselamatan dulu, di atas vcirs-core.*

```
        return out;
    }

    static List<Condition> factsOf(InferenceResult result) {
        List<Condition> facts = new ArrayList<>();
        for (Fact f : result.memory.facts.values()) {
            if (!f.variable.equals("advice")) {
                facts.add(new Condition(f.variable, "=", f.value));
            }
        }
        return facts;
    }

    /** Perawat membetulkan atau melengkapi saran, memilih semuanya. */
    static void teach(InferenceResult result, String path) {
        String advice = read("Which advice is right or missing? (label) > ");
        if (advice.isEmpty()) {
            return;
        }
        List<Node> fired = new ArrayList<>();
        for (RdrPath p : result.paths) {
            fired.add(p.last());
        }
        for (int i = 0; i < fired.size(); i++) {
            System.out.println("  " + (i + 1) + ") " + fired.get(i).id);
        }
        String pick = read("Which node gave wrong or incomplete advice? "
                + "(number, Enter for none) > ");
        Node node = pick.isEmpty() ? null : fired.get(Integer.parseInt(pick) - 1);
        List<Condition> facts = factsOf(result);
        while (true) {
            String extra = read("Another fact that matters? "
                    + "(variable = value, Enter when done) > ");
            if (extra.isEmpty()) {
                break;
            }
            try {
                facts.add(Condition.parse(extra));
            } catch (VcirsException e) {
                System.out.println("Please write it as: variable = value");
            }
        }
        Case c = new Case(facts, List.of(new Conclusion("advice", advice)), 100);
        if (node != null) {
            for (String line : kb.differences(node.id, c).lines()) {
                show("", "    ", line);
```

Kode 5.5 (Lanjutan) *HealthCheck.java: cek kesehatan mandiri, keselamatan dulu, di atas vcirs-core.*

```
            }
        }
        int option = Integer.parseInt(read("Option (0 accept, 1-3 exception, "
                + "4 beside, 5 top level) > "));
        if (option != 5 && node == null) {
            System.out.println("Options 0 to 4 need a node; nothing was changed.");
            return;
        }
        if (option == KnowledgeBase.ACCEPT) {
            kb.refine(c, KnowledgeBase.ACCEPT, node.id, null, "nurse");
            kb.save(path);
            System.out.println("Accepted; the check is recorded.");
            return;
        }
        List<Condition> conds = builtBy(option, node, c);
        show("", "  ", "Option " + option + " builds: " + join(conds));
        String keep = read("Conditions to keep, separated by ';' "
                + "(Enter = as built) > ");
        if (!keep.isEmpty()) {
            conds = new ArrayList<>();
            for (String k : keep.split(";")) {
                conds.add(Condition.parse(k));
            }
        }
        String parent = option == 5 ? null
                : option == 4 ? node.parent : node.id;
        while (true) {
            List<String> hit = kb.affectedCornerstones(parent, conds,
                    c.conclusions);
            System.out.println("Earlier cases whose advice would change: "
                    + (hit.isEmpty() ? "none" : String.join(", ", hit)));
            String more = hit.isEmpty() ? "" : read("Add a condition to tell "
                    + "them apart? (Enter to go ahead) > ");
            if (more.isEmpty()) {
                break;
            }
            conds = new ArrayList<>(conds);
            conds.add(Condition.parse(more));
        }
        Node learned;
        try {
            learned = kb.refine(c, option, node == null ? null : node.id, conds,
                    "nurse");
        } catch (VcirsException e) {
            System.out.println("Cannot learn: " + e.getMessage());
            return;
```

Kode 5.5 (Lanjutan) *HealthCheck.java: cek kesehatan mandiri, keselamatan dulu, di atas vcirs-core.*

```
        }
        System.out.println("Learned: " + learned);
        kb.save(path);
        System.out.println("Saved " + path + ": " + kb.summary());
    }

    public static void main(String[] args) {
        String path = "../data/health-kb.json";
        boolean tree = false;
        boolean nurse = false;
        for (String a : args) {
            if (a.equals("--echo")) {
                echo = true;
            } else if (a.equals("--tree")) {
                tree = true;
            } else if (a.equals("--nurse")) {
                nurse = true;
            } else {
                path = a;
            }
        }
        kb = openKb(path);
        if (tree) {
            for (String line : Report.tree(kb)) {
                System.out.println(line);
            }
            return;
        }
        Map<String, String> texts = readTexts(Path.of(path).toAbsolutePath()
                .getParent().resolve("advice-texts.txt"));
        System.out.println("Health self-check for adults. It is not a "
                + "diagnosis: it tells you");
        System.out.println("what you can do yourself, and when to see a health "
                + "worker.");
        System.out.println("First, the safety questions.");
        if (!safe()) {
            return;
        }
        System.out.println("Now about how you feel. Press Enter if you do not "
                + "know.");
        InferenceResult result = kb.infer(Facts.none(), HealthCheck::ask, true);
        if (result.paths.isEmpty()) {
            System.out.println("The self-check has no advice for this. If you "
                    + "are worried, see the puskesmas.");
        }
        for (RdrPath p : result.paths) {
```

Kode 5.5 (Lanjutan) *HealthCheck.java: cek kesehatan mandiri, keselamatan dulu, di atas vcirs-core.*

```
            for (Conclusion c : p.conclusions()) {
                show("* ", "  ", texts.getOrDefault(c.value, c.value));
            }
            for (Conclusion c : p.intermediate()) {
                show("  and: ", "  ", texts.getOrDefault(c.value, c.value));
            }
            List<Condition> used = new ArrayList<>();
            for (Node n : p.nodes) {
                used.addAll(n.conditions);
            }
            show("  (", "  ", p.ruleId() + ": " + join(used) + ")");
        }
        show("", "", "If anything gets worse or a warning sign appears, see the "
                + "puskesmas at once or call 119.");
        if (!nurse) {
            return;
        }
        if (yes("Nurse: is the advice right and complete?")) {
            Case c = new Case(factsOf(result), result.conclusions(), 100);
            for (RdrPath p : result.paths) {
                kb.refine(c, KnowledgeBase.ACCEPT, p.last().id, null, "nurse");
            }
            kb.save(path);
            System.out.println("Good. The check is recorded.");
        } else {
            teach(result, path);
        }
    }
}
```

Ini program terpanjang sejauh ini, jadi ambillah dalam lima bagian.

Keselamatan. `ADULT`, `NOT_ADULT`, dan `RED_FLAGS` adalah aturan keselamatan tetap dari Subbab 5.2.1, masing-masing sebuah pertanyaan dan saran yang diberikan bila jawabannya *ya*. `safe` menanyakannya berurutan dan berhenti pada *ya* pertama. Semuanya konstanta di dalam program, bukan node di dalam basis pengetahuan, dan tidak ada pengajaran yang dapat mengubahnya.

Kata-kata di layar. `wrap` memecah teks panjang menjadi baris-baris yang paling panjang selebar yang ditentukan, di spasi, dan `show` mencetak teks yang dibungkus dengan awalan di baris pertama (misalnya `*`) dan inden di baris lainnya, sambil mempertahankan inden yang sudah dimiliki teks itu. `read` membungkus pertanyaan panjang dengan cara yang sama. Semua ini demi manusia: pertanyaan atau saran yang terpotong di tepi layar ponsel tidak akan dibaca.

Basis pengetahuan. `openKb` mendeklarasikan sepuluh variabel dalam `QUESTIONS` dan membangun basis pengetahuan dari kasus awal pada kali pertama, persis seperti di Bab 4. `readTexts` membaca kata-kata saran. `main` lalu berkonsultasi dengan basis pengetahuan lewat `kb.infer(Facts.none(), HealthCheck::ask, true)` — argumen terakhir, `true`, meminta **setiap** cabang yang berlaku. Untuk setiap jalur, program mencetak saran node terakhir sesudah `*`, saran node-node di atasnya sesudah `and:`, dan, dalam tanda kurung, rule dan kondisi yang menghasilkannya, agar orang itu — dan perawatnya — dapat melihat alasannya.

Memeriksa. Dengan `--nurse`, program menanyakan kepada perawat apakah sarannya benar dan lengkap. Bila ya, setiap node yang menjawab dicatat sebagai sudah diperiksa dengan opsi 0.

Mengajar. Bila tidak, `teach` menuntun perawat melewati perulangan penyempurnaan di Subbab 5.2: saran yang benar atau yang kurang; node yang memberi saran keliru atau kurang, dipilih dari node-node yang menjawab (atau tidak ada, untuk node tingkat atas baru); fakta lain yang penting;

daftar perbedaan; opsinya; apa yang akan dibangun opsi itu (`builtBy`); kondisi yang disimpan, yang boleh ia ketik sendiri, dipisahkan titik koma; lalu cornerstone yang terdampak, berulang-ulang, sampai daftarnya kosong atau ia memutuskan untuk meneruskan. Baru setelah itu `refine` menambahkan node-nya, dan berkasnya disimpan.

Kode 5.6 adalah aplikasi yang sama dalam Python.

Kode 5.6 `health_check.py`: cek kesehatan mandiri dalam Python.

```
"""Pemeriksaan kesehatan mandiri: keselamatan dulu, lalu saran perawatan
mandiri yang diajarkan perawat kasus demi kasus. Ini bukan diagnosis.

    python health_check.py [--echo] [--nurse] [--tree] [knowledge-base.json]

Pertanyaan keselamatan datang lebih dulu dan tetap di dalam program: bila
salah satunya dijawab ya, program langsung mengarahkan orang itu mencari
pertolongan dan tidak bertanya lagi. Baru setelah itu program berkonsultasi
dengan basis pengetahuan, mengikuti setiap saran yang berlaku. Dengan
--nurse, perawat memeriksa saran dan mengajari basis pengetahuan: ia memilih
node, opsi, dan kondisinya, dan melihat kasus lama mana yang akan terdampak.
"""

import os
import pathlib
import sys

# Beri tahu Python letak mesin vcirs-core (naik dua folder, lalu masuk).
sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import (ACCEPT, Case, Condition, Conclusion, KnowledgeBase,
                   VcirsError, casefile, report)

# Keselamatan dulu: aturan tetap yang tak dapat diubah atau dihapus pengajaran.
ADULT = "Is this self-check for an adult (18 or older) who is not pregnant?"
NOT_ADULT = ("This self-check is for adults who are not pregnant. For a "
             "child, or during pregnancy, see the bidan or the puskesmas.")
RED_FLAGS = [
    ("Right now: sudden weakness or numbness on one side, a drooping face, "
     "slurred speech, sudden trouble seeing, or the worst headache ever?",
     "EMERGENCY: these can be signs of a stroke. Call 119 or go to the "
     "nearest hospital emergency room (IGD) now."),
    ("Chest pain or pressure, or trouble breathing?",
     "EMERGENCY: call 119 or go to the IGD now."),
    ("Fever with any of: severe belly pain, vomiting 3 or more times a day, "
     "bleeding gums or nose, blood in vomit or stool, great weakness?",
     "URGENT: these are warning signs of severe dengue. Go to the IGD now."),
    ("Unable to keep water down, or passing very little urine?",
     "URGENT: the body may be losing too much water. Go to the puskesmas or "
     "the IGD now."),
]
QUESTIONS = [
    ("fever", "Do you have a fever?", ["yes", "no"]),
    ("fever days", "For how many days have you had the fever?", []),
    ("cough", "Do you have a cough?", ["yes", "no"]),
    ("cough weeks", "For how many weeks have you been coughing?", []),
```

Kode 5.6 (Lanjutan) *health_check.py: cek kesehatan mandiri dalam Python.*

```
    ("smoker", "Do you smoke?", ["yes", "no"]),
    ("diarrhea", "Diarrhea (three or more loose stools a day)?", ["yes", "no"]),
    ("diarrhea days", "For how many days have you had diarrhea?", []),
    ("others ill", "Are others who ate the same food sick too?", ["yes", "no"]),
    ("headache", "Do you have a headache?", ["yes", "no"]),
    ("blood pressure", "Top number of your blood pressure, if measured today?",
     []),
]


def wrap(text, width):
    """Pecah teks menjadi baris paling panjang width karakter, di spasi."""
    lines, line = [], ""
    for word in text.split(" "):
        if line and len(line) + 1 + len(word) > width:
            lines.append(line)
            line = word
        else:
            line = word if not line else line + " " + word
    return lines + [line]


def show(first, rest, text):
    """Cetak teks terbungkus, baris pertama sesudah 'first', sisanya sesudah
    'rest' (misalnya '* ' dan '  ')."""
    lead = " " * (len(text) - len(text.lstrip(" ")))  # pertahankan inden
    for i, line in enumerate(wrap(text.lstrip(" "), 72)):
        print((first + lead if i == 0 else rest) + line)


class Console:
    """Membaca jawaban pengguna; bila echo aktif, mencetaknya juga."""

    def __init__(self, kb, echo):
        self.kb = kb
        self.echo = echo
```

Kode 5.6 (Lanjutan) *health_check.py: cek kesehatan mandiri dalam Python.*

```
    def read(self, prompt):
        lines = wrap(prompt.rstrip(" >"), 74)       # pertanyaan panjang dibungkus
        for line in lines[:-1]:
            print(line)
        print(lines[-1] + " > ", end="")
        text = sys.stdin.readline().strip()
        if self.echo:
            print(text)
        return text

    def yes(self, question):
        return self.read(question + " (yes/no) > ") == "yes"

    def ask(self, question):
        var = self.kb.variables.get(question.variable)
        text, choices = question.variable, question.choices
        if var is not None and var.question:
            text, choices = var.question, var.choices
        if choices:
            text += " (" + "/".join(choices) + ")"
        return self.read(text + " > ") or None    # Enter = "saya tidak tahu"


def read_texts(path):
    """advice-texts.txt: satu 'label = teks' per baris."""
    texts = {}
    with open(path, encoding="utf-8") as f:
        for line in f:
            if "=" in line and not line.startswith("#"):
                label, text = line.split("=", 1)
                texts[label.strip()] = text.strip()
    return texts


def open_kb(path):
    """Muat basis pengetahuan, atau bangun dari kasus-kasus awal."""
    if os.path.exists(path):
        return KnowledgeBase.load(path)
    seed = os.path.join(os.path.dirname(path), "health-seed-cases.json")
    name, description, steps = casefile.read_cases(seed)
    kb = KnowledgeBase(name, description)
    for var, question, choices in QUESTIONS:
        kb.declare(var, question, choices)
    casefile.build(kb, steps)
    kb.save(path)
    return kb
```

Kode 5.6 (Lanjutan) *health_check.py: cek kesehatan mandiri dalam Python.*

```
def safe(con):
    """Pertanyaan keselamatan tetap. True bila saran mandiri boleh menyusul."""
    if not con.yes(ADULT):
        show("", "", NOT_ADULT)
        return False
    for question, advice in RED_FLAGS:
        if con.yes(question):
            show("", "", advice)
            return False
    return True


def built_by(kb, option, node, case):
    """Kondisi yang dibangun sebuah opsi dari daftar perbedaan."""
    if option in (1, 2, 3):
        d = kb.differences(node.id, case)
        neg = [c.negated() for c in d.a] if option in (1, 3) else []
        return neg + (d.b if option in (2, 3) else [])
    return list(case.facts)                      # opsi 4 dan 5


def teach(kb, con, result, path):
    """Perawat membetulkan atau melengkapi saran, memilih semuanya."""
    advice = con.read("Which advice is right or missing? (label) > ")
    if not advice:
        return
    fired = [p.last() for p in result.paths]
    for i, node in enumerate(fired):
        print("  " + str(i + 1) + ") " + node.id)
    pick = con.read("Which node gave wrong or incomplete advice? "
                    "(number, Enter for none) > ")
    node = fired[int(pick) - 1] if pick else None
    facts = [Condition(f.variable, "=", f.value)
             for f in result.memory.facts.values() if f.variable != "advice"]
    while True:
        extra = con.read("Another fact that matters? "
                         "(variable = value, Enter when done) > ")
        if not extra:
            break
        try:
            facts.append(Condition.parse(extra))
        except VcirsError:
            print("Please write it as: variable = value")
    case = Case(facts, [Conclusion("advice", advice)])
```

Kode 5.6 (Lanjutan) *health_check.py: cek kesehatan mandiri dalam Python.*

```
    if node is not None:
        for line in kb.differences(node.id, case).lines():
            show("", "    ", line)
    option = int(con.read("Option (0 accept, 1-3 exception, 4 beside, "
                          "5 top level) > "))
    if option != 5 and node is None:
        print("Options 0 to 4 need a node; nothing was changed.")
        return
    if option == ACCEPT:
        kb.refine(case, ACCEPT, node.id, author="nurse")
        kb.save(path)
        print("Accepted; the check is recorded.")
        return
    conds = built_by(kb, option, node, case)
    show("", "  ", "Option " + str(option) + " builds: "
         + ", ".join(str(c) for c in conds))
    keep = con.read("Conditions to keep, separated by ';' "
                    "(Enter = as built) > ")
    if keep:
        conds = [Condition.parse(c) for c in keep.split(";")]
    parent = {4: node.parent if node else None, 5: None}.get(
        option, node.id if node else None)
    while True:
        hit = kb.affected_cornerstones(parent, conds, case.conclusions)
        print("Earlier cases whose advice would change: "
              + (", ".join(hit) if hit else "none"))
        more = con.read("Add a condition to tell them apart? "
                        "(Enter to go ahead) > ") if hit else ""
        if not more:
            break
        conds = conds + [Condition.parse(more)]
    try:
        new = kb.refine(case, option, node.id if node else None,
                        chosen=conds, author="nurse")
    except VcirsError as e:
        print("Cannot learn: " + str(e))
        return
    print("Learned: " + str(new))
    kb.save(path)
    print("Saved " + path + ": " + kb.summary())
```

Kode 5.6 (Lanjutan) *health_check.py: cek kesehatan mandiri dalam Python.*

```
def main(args):
    echo = "--echo" in args
    files = [a for a in args if not a.startswith("--")]
    path = files[0] if files else "../data/health-kb.json"
    kb = open_kb(path)
    if "--tree" in args:
        for line in report.tree(kb):
            print(line)
        return
    texts = read_texts(os.path.join(os.path.dirname(path),
                                    "advice-texts.txt"))
    con = Console(kb, echo)
    print("Health self-check for adults. It is not a diagnosis: it tells you")
    print("what you can do yourself, and when to see a health worker.")
    print("First, the safety questions.")
    if not safe(con):
        return
    print("Now about how you feel. Press Enter if you do not know.")
    result = kb.infer({}, con.ask, multiple=True)
    if not result.paths:
        print("The self-check has no advice for this. If you are worried, "
              "see the puskesmas.")
    for p in result.paths:
        for c in p.conclusions():
            show("* ", "  ", texts.get(c.value, c.value))
        for c in p.intermediate():
            show("  and: ", "  ", texts.get(c.value, c.value))
        show("  (", "  ", p.rule_id() + ": " + ", ".join(
            str(c) for n in p.nodes for c in n.conditions) + ")")
    show("", "", "If anything gets worse or a warning sign appears, see the "
         "puskesmas at once or call 119.")
    if "--nurse" not in args:
        return
    if con.yes("Nurse: is the advice right and complete?"):
        case = Case([Condition(f.variable, "=", f.value)
                     for f in result.memory.facts.values()
                     if f.variable != "advice"], result.conclusions())
        for p in result.paths:
            kb.refine(case, ACCEPT, p.last().id, author="nurse")
        kb.save(path)
        print("Good. The check is recorded.")
    else:
        teach(kb, con, result, path)
```

Kode 5.6 (Lanjutan) *health_check.py: cek kesehatan mandiri dalam Python.*

```
if __name__ == "__main__":
    main(sys.argv[1:])
```

5.6 Empat Orang di Puskesmas

Kompilasi dan jalankan dari `Chapter05/java`:

```
javac -d out -sourcepath ../../vcirs-core/java/src *.java
java -cp out HealthCheck --echo < ../data/session-1.txt
```

atau, dari `Chapter05/python`, `python health_check.py --echo < ../data/session-1.txt`. Sesi 3 dan 4 dijalankan dengan `--nurse` juga. Jalankan keempat sesi berurutan pada basis pengetahuan yang baru, dan transkrip serta `health-kb.json` yang tersimpan sama, bita demi bita, dalam kedua bahasa.

Sesi 1: Pak Darmo, pada hari ketiga demamnya. Putrinya menjalankan pemeriksaan mandiri di ponselnya untuk ayahnya.

```
Health self-check for adults. It is not a diagnosis: it tells you
what you can do yourself, and when to see a health worker.
First, the safety questions.
Is this self-check for an adult (18 or older) who is not pregnant?
(yes/no) > yes
Right now: sudden weakness or numbness on one side, a drooping face,
slurred speech, sudden trouble seeing, or the worst headache ever?
(yes/no) > no
Chest pain or pressure, or trouble breathing? (yes/no) > no
Fever with any of: severe belly pain, vomiting 3 or more times a day,
bleeding gums or nose, blood in vomit or stool, great weakness? (yes/no) > no
Unable to keep water down, or passing very little urine? (yes/no) > no
Now about how you feel. Press Enter if you do not know.
Do you have a fever? (yes/no) > yes
For how many days have you had the fever? > 3
Do you have a cough? (yes/no) > no
Diarrhea (three or more loose stools a day)? (yes/no) > no
Do you have a headache? (yes/no) > yes
Top number of your blood pressure, if measured today? >
* A fever of two days or more needs a check: go to the puskesmas today and
  ask whether a blood test is needed. Until then take paracetamol only,
  not ibuprofen or aspirin.
  and: Rest, drink plenty of water, and take paracetamol as written on the
  label. Measure your temperature morning and evening.
  (dengue check#1: fever = yes, fever days >= 2)
* Rest, drink water, and take paracetamol as written on the label.
  (headache home care#1: headache = yes)
If anything gets worse or a warning sign appears, see the puskesmas at
once or call 119.
```

Kelima pertanyaan keselamatan datang lebih dulu, dan semua jawabannya *no* (tidak). Lalu basis pengetahuan hanya menanyakan apa yang dibutuhkan node-node-nya: demam dan berapa lama, batuk, diare, sakit kepala, dan — karena node sakit kepala punya pengecualian tentang tekanan darah — angka atas tekanan darahnya, yang tidak diukur hari itu, sehingga ia menekan Enter. Dua saran keluar, satu per cabang. Yang pertama pemeriksaan dengue, karena demamnya sudah tiga hari, dengan saran perawatan demam dari node di atasnya; yang kedua perawatan sakit kepala di rumah. Baris dalam tanda kurung di bawah masing-masing menunjukkan rule dan kondisinya: `dengue check#1: fever = yes, fever days >= 2`. Baris terakhir sama untuk semua orang: bila keadaan memburuk, berangkatlah.

Sarannya berkata *hari ini*, dan *parasetamol saja, bukan ibuprofen atau aspirin*, karena WHO memperingatkan bahwa dua obat itu meningkatkan risiko perdarahan pada dengue [1]. Putri Pak Darmo mengantarnya ke puskesmas sore itu.

Sesi 2: Bu Sum, penjual jamu, dengan tanda bahaya. Ia sudah demam empat hari, dan pagi ini gusinya berdarah ketika menggosok gigi.

```
Health self-check for adults. It is not a diagnosis: it tells you
what you can do yourself, and when to see a health worker.
First, the safety questions.
Is this self-check for an adult (18 or older) who is not pregnant?
(yes/no) > yes
Right now: sudden weakness or numbness on one side, a drooping face,
slurred speech, sudden trouble seeing, or the worst headache ever?
(yes/no) > no
Chest pain or pressure, or trouble breathing? (yes/no) > no
Fever with any of: severe belly pain, vomiting 3 or more times a day,
bleeding gums or nose, blood in vomit or stool, great weakness? (yes/no) > yes
URGENT: these are warning signs of severe dengue. Go to the IGD now.
```

Pertanyaan keselamatan ketiga menangkapnya, dan pemeriksaan mandiri hanya mengatakan satu hal: ini tanda bahaya dengue berat; pergilah ke IGD sekarang [1], [7]. Ia tidak menanyakan batuk atau sakit kepalanya, dan sama sekali tidak berkonsultasi dengan basis pengetahuan. Inilah susunan **Gambar 5.1** yang bekerja sebagaimana dirancang.

Sesi 3: Mas Yanto, dan perawat mengajar. Mas Yanto, juru parkir di pasar, diare sejak kemarin. Ia datang ke posbindu, dan Ners Ratna menjalankan pemeriksaan mandiri bersamanya dalam mode perawat.

```
Health self-check for adults. It is not a diagnosis: it tells you
what you can do yourself, and when to see a health worker.
First, the safety questions.
Is this self-check for an adult (18 or older) who is not pregnant?
(yes/no) > yes
Right now: sudden weakness or numbness on one side, a drooping face,
slurred speech, sudden trouble seeing, or the worst headache ever?
(yes/no) > no
Chest pain or pressure, or trouble breathing? (yes/no) > no
Fever with any of: severe belly pain, vomiting 3 or more times a day,
bleeding gums or nose, blood in vomit or stool, great weakness? (yes/no) > no
Unable to keep water down, or passing very little urine? (yes/no) > no
Now about how you feel. Press Enter if you do not know.
Do you have a fever? (yes/no) > no
Do you have a cough? (yes/no) > no
Diarrhea (three or more loose stools a day)? (yes/no) > yes
For how many days have you had diarrhea? > 1
Do you have a headache? (yes/no) > no
* Drink oralit (ORS) after every loose stool, keep eating small meals, and
  wash your hands with soap.
  (ORS at home#1: diarrhea = yes)
If anything gets worse or a warning sign appears, see the puskesmas at
once or call 119.
Nurse: is the advice right and complete? (yes/no) > no
Which advice is right or missing? (label) > report food poisoning
  1) ORS at home#1
Which node gave wrong or incomplete advice? (number, Enter for none) > 1
Another fact that matters? (variable = value, Enter when done) > others ill = yes
Another fact that matters? (variable = value, Enter when done) >
Differences with the cornerstone case of ORS at home#1:
  A (only in the old case): (none)
  B (only in the new case): fever = no, cough = no, diarrhea days = 1,
    headache = no, others ill = yes
Option (0 accept, 1-3 exception, 4 beside, 5 top level) > 2
Option 2 builds: fever = no, cough = no, diarrhea days = 1, headache =
  no, others ill = yes
Conditions to keep, separated by ';' (Enter = as built) > others ill = yes
Earlier cases whose advice would change: none
Learned: report food poisoning#1: others ill = yes -> advice = report food poisoning
Saved ../data/health-kb.json: Health self-check: 11 variables, 10 nodes, 10 rules
```

Basis pengetahuan memberi saran oralit, yang benar sejauh itu. Tetapi sambil berbicara dengannya, Ners Ratna mengetahui bahwa semua yang makan *nasi kotak* di pesta pernikahan semalam juga sakit. Itu bukan sekadar diare; mungkin keracunan makanan, dan puskesmas harus tahu hari ini, agar tamu-tamu lain dapat diperingatkan. Sarannya kurang, jadi ia menjawab *no*.

Layar mengajar mengerjakan sisanya. Ia mengetik saran yang kurang, `report food poisoning` (laporkan keracunan makanan); memilih node yang memberi saran kurang, `ORS at home#1`; dan menambahkan fakta yang penting, `others ill = yes`. Daftar perbedaan terhadap cornerstone `ORS at home#1` (yang hanya `diarrhea = yes`) punya bagian A kosong dan bagian B yang panjang: setiap jawaban Mas Yanto — *tidak demam, tidak batuk, diare satu hari, tidak sakit kepala* — dan fakta yang ia tambahkan. Inilah harga menanyakan setiap keluhan: sebagian besar fakta sebuah kasus nyata tidak ada hubungannya dengan saran baru. Opsi 2 akan membangun kelimanya ke dalam node, dan node itu hanya akan pernah menyala untuk orang dengan pola *persis* ini. Ia mengetik satu kondisi yang menjadi alasannya, `others ill = yes`. Tidak ada kasus lama yang sarannya berubah, dan `report food poisoning#1` ditambahkan di bawah `ORS at home#1`.

Sesi 4: Bu Tini, keesokan paginya. Ia hadir di pernikahan yang sama.

```
Health self-check for adults. It is not a diagnosis: it tells you
what you can do yourself, and when to see a health worker.
First, the safety questions.
Is this self-check for an adult (18 or older) who is not pregnant?
(yes/no) > yes
Right now: sudden weakness or numbness on one side, a drooping face,
slurred speech, sudden trouble seeing, or the worst headache ever?
(yes/no) > no
Chest pain or pressure, or trouble breathing? (yes/no) > no
Fever with any of: severe belly pain, vomiting 3 or more times a day,
bleeding gums or nose, blood in vomit or stool, great weakness? (yes/no) > no
Unable to keep water down, or passing very little urine? (yes/no) > no
Now about how you feel. Press Enter if you do not know.
Do you have a fever? (yes/no) > no
Do you have a cough? (yes/no) > no
Diarrhea (three or more loose stools a day)? (yes/no) > yes
For how many days have you had diarrhea? > 2
Are others who ate the same food sick too? (yes/no) > yes
Do you have a headache? (yes/no) > no
* Several people sick after the same food may be food poisoning: tell the
  puskesmas today, so that others can be warned.
  and: Drink oralit (ORS) after every loose stool, keep eating small meals, and
  wash your hands with soap.
  (report food poisoning#1: diarrhea = yes, others ill = yes)
If anything gets worse or a warning sign appears, see the puskesmas at
once or call 119.
Nurse: is the advice right and complete? (yes/no) > yes
Good. The check is recorded.
```

Kini pemeriksaan mandiri mengajukan pertanyaan yang belum pernah diajukannya — *Are others who ate the same food sick too?* (apakah orang lain yang makan makanan yang sama juga sakit?) — karena node yang dipelajari kemarin membutuhkannya, dan kata-kata pertanyaannya sudah dideklarasikan. Sarannya meminta memberi tahu puskesmas hari ini, dengan saran oralit dari node di atasnya. Ners Ratna mengonfirmasinya, dan pemeriksaannya dicatat. Menjelang siang puskesmas sudah mendengar dari enam tamu, dan dinas kesehatan kabupaten mulai memeriksa usaha katering itu.

`--tree` memperlihatkan basis pengetahuan setelah keempat sesi:

```
(root)
+- fever home care#1: fever = yes -> advice = fever home care
|  +- dengue check#1: fever days >= 2 -> advice = dengue check
+- cough home care#1: cough = yes -> advice = cough home care
|  +- TB check#1: cough weeks >= 2 -> advice = TB check
|  +- quit smoking help#1: smoker = yes, cough weeks < 2 -> advice = quit smoking help
+- ORS at home#1: diarrhea = yes -> advice = ORS at home
|  +- diarrhea check#1: diarrhea days >= 3 -> advice = diarrhea check
|  +- report food poisoning#1: others ill = yes -> advice = report food poisoning
+- headache home care#1: headache = yes -> advice = headache home care
   +- blood pressure check#1: blood pressure >= 140 -> advice = blood pressure check
```

Satu-satunya perubahan sejak sesi mengajar di Subbab 5.4 adalah pengecualian baru di bawah `ORS at home#1`, di samping `diarrhea check#1`: keduanya dicoba untuk setiap orang yang diare, dan dalam mode banyak-konklusi keduanya dapat menjawab.

5.7 Membuatnya Lebih Baik

Menghentikan saran yang sudah keliru. Andaikan dokter puskesmas meminta agar saran sakit kepala tidak lagi menyebut parasetamol, karena terlalu banyak orang meminumnya setiap hari. Ners Ratna menghentikan `headache home care#1` dan mengajarkan saran baru sebagai node baru. Soal 5.7 menunjukkan bahwa node yang dihentikan tetap bekerja sebagai konteks bagi pengecualiannya — pemeriksaan tekanan darah tetap menyala — sementara sarannya sendiri tidak lagi diberikan. Tidak ada yang dihapus, dan riwayat mencatat penghentiannya.

Anak dan kehamilan. Pemeriksaan mandiri mengarahkan anak dan ibu hamil langsung ke bidan atau puskesmas, karena tanda bahaya mereka berbeda dan harus diajarkan oleh orang yang mengenalnya. Cara yang tepat untuk mencakup mereka bukan menambahkan beberapa node ke basis pengetahuan ini, melainkan membangun basis pengetahuan tersendiri, dengan pertanyaan keselamatan tetapnya sendiri, diajarkan oleh bidan — pengelompokan menurut pasien yang dipakai sistem kesehatan VCIRS yang asli [9].

Angka dari alat. Kini banyak keluarga punya tensimeter digital dan termometer. Hasil ukurnya dapat diketik sebagai angka, dan node seperti `blood pressure >= 140` membandingkannya sebagai angka. Satu kali ukur belum membuktikan banyak; WHO meminta dua kali ukur pada hari berbeda sebelum menyebut tekanan darah tinggi [2], dan itulah yang dikatakan sarannya.

Pengetahuan sama, pertanyaan lebih baik. Pak Darmo ditanya tentang batuk, diare, dan sakit kepala walaupun ia datang karena demam, karena mode banyak-konklusi mencoba setiap cabang. Untuk daftar keluhan yang panjang, pertanyaan pertama — *apa yang paling mengganggu Anda?* — dan basis pengetahuan yang dikelompokkan per keluhan membuat pemeriksaan mandiri tetap singkat. Tabel Variabel dari Bab 4 menunjukkan variabel mana yang sebaiknya didahulukan.

> **JEBAKAN:** Jangan pernah "memperbaiki" pertanyaan keselamatan dengan mengajari basis pengetahuan. Bila sebuah tanda bahaya terlalu sering ditanyakan, atau dengan kata-kata yang tidak dipahami orang, ubahlah programnya dan periksa ulang sumbernya; jangan menambahkan node yang menjawab tanda bahaya dengan perawatan di rumah.

Pengujian. `HealthTest.java` dan `test_health.py` memeriksa fungsi pembantu Langkah 1 (negasi, perbandingan, fakta yang tidak diketahui), sesi mengajar (sembilan node, kondisi `quit smoking help#1`, setiap cornerstone mempertahankan sarannya), dan aplikasinya (pembungkus kata, empat tanda bahaya, sembilan node awal, dua saran untuk demam dan sakit kepala, dan kata-kata di `advice-texts.txt` untuk setiap saran yang dapat diberikan basis pengetahuan). Keduanya diakhiri dengan:

```
all tests passed
```

5.8 Memakainya Sungguh-Sungguh

Di posbindu atau puskesmas. Jalankan pemeriksaan mandiri dalam mode perawat di posbindu, bersama orang-orang yang datang, dan dalam mode biasa di ponsel atau laptop di ruang tunggu bagi yang sedang menunggu. Ajarilah hanya dari pasien yang benar-benar Anda temui, yang keadaannya Anda pahami. Setelah beberapa bulan, basis pengetahuan menyimpan pengalaman puskesmas itu sendiri: saran untuk penyakit-penyakit yang umum di kampung *ini*, dengan kata-kata yang dipahami warganya.

Periksa pertanyaan keselamatan terhadap sumber terkini. Tanda-tanda bahaya dalam bab ini mengikuti lembar fakta WHO tentang dengue [1], kampanye SeGeRa Ke RS Kementerian Kesehatan untuk stroke [6], dan panduannya tentang kapan dengue membutuhkan IGD [7]. Pedoman berubah; bacalah lagi setiap tahun, dan setiap kali Kementerian Kesehatan menerbitkan panduan baru, dan ubahlah programnya bila pedomannya berubah.

Kata-kata yang dipakai orang. Pertanyaan dalam bab ini berbahasa Inggris agar kodenya sama di kedua edisi buku ini. Di puskesmas sungguhan pertanyaannya harus dalam bahasa pasien — bahasa Indonesia, Jawa, Madura — dan dicoba pada beberapa orang sebelum dipakai. Pertanyaan yang disalahpahami menghasilkan fakta yang salah, dan fakta yang salah membuat node yang salah menyala.

Data kesehatan adalah data pribadi. Kasus-kasus dalam basis pengetahuan menyimpan gejala, bukan nama, dan itu disengaja. Menurut Undang-Undang Pelindungan Data Pribadi, data kesehatan termasuk data pribadi yang bersifat spesifik dan membutuhkan perlindungan khusus [19]. Basis pengetahuan yang dimaksudkan untuk dipakai bersama antarpuskesmas semestinya memuat pengetahuan tentang keadaan, tidak pernah tentang orang: tanpa nama, tanpa alamat, tanpa nomor telepon, dan tanpa gabungan fakta yang dapat mengenali satu orang.

Apakah pemeriksaan mandiri ini membantu? Riwayat basis pengetahuan menjawab sebagian pertanyaan ini tanpa kerja tambahan. Hitunglah, setiap bulan, berapa konsultasi yang diterima perawat (opsi 0) dan berapa yang harus ia betulkan (opsi 1 sampai 5). Pada bulan-bulan pertama pembetulan akan sering terjadi; ketika basis pengetahuan makin matang, bagian konsultasi yang diterima semestinya naik. Lonjakan pembetulan yang tiba-tiba adalah tanda yang layak dibaca — penyakit baru di kampung, perubahan panduan, pertanyaan yang disalahpahami. Bagian lain dari pertanyaan itu tidak dapat dijawab program: apakah orang yang disuruh *berangkat hari ini* benar-benar berangkat, dan apakah yang diberi tahu *cukup dirawat di rumah* benar-benar membaik? Tindak lanjut sederhana di posbindu berikutnya, menanyakan kepada beberapa orang apa yang mereka lakukan dan bagaimana keadaan mereka, adalah satu-satunya ukuran yang jujur, dan sekaligus sumber terbaik kasus baru untuk diajarkan.

Siapa yang mengajar. Hanya tenaga kesehatan yang boleh mengajari basis pengetahuan, dan setiap node mencatat siapa yang mengajarkannya. Ketika dua perawat berbeda pendapat, riwayat menunjukkan kasus dan orang di balik setiap node, dan perbedaan itu menjadi percakapan tentang kasus nyata. Bab 11 membangun di atas hal ini untuk tim klinik dengan banyak pakar, versi, dan jejak audit.

5.9 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Menaruh tanda bahaya di dalam pengetahuan yang dipelajari. Tanda bahaya yang tinggal di basis pengetahuan dapat ditimpa oleh sebuah pengecualian. *Hindari:* simpan pertanyaan keselamatan di dalam program, sebelum basis pengetahuan.

Menerima opsi apa adanya. Opsi 2 dan 3 menyalin setiap fakta berbeda dari sebuah kasus nyata, dan sebagian besar tidak relevan (sesi 3). *Hindari:* baca daftar perbedaan, dan simpan hanya fakta yang menjelaskan saran baru — atau tulis aturan yang sungguh Anda pakai.

Menegasikan padahal yang dimaksud ambang. Opsi 1 mengubah `fever days = 1` menjadi `fever days <> 1`, yang juga menangkap nol hari. *Hindari:* bila bagian A memuat angka, tulislah ambang yang Anda maksud (`fever days >= 2`).

Mengabaikan cornerstone yang terdampak. Node baru yang tampak benar bisa menjauhkan pasien lama dari saran yang mereka butuhkan (kasus 8). *Hindari:* baca setiap kasus yang terdampak, dan tambahkan kondisi yang membedakannya kecuali sarannya memang sepatutnya berubah.

Mengajar dari tebakan. Kasus yang diajarkan sebelum hasil akhir pasien diketahui bisa mengajarkan kesalahan. *Hindari:* ajarkan dari kasus yang sudah Anda ikuti sampai selesai, atau setidaknya sudah dicek bersama rekan.

Menghapus atau menyunting node lama. Ini menghancurkan jaminan bahwa kasus lama mempertahankan jawabannya, dan riwayat siapa mengajarkan apa. *Hindari:* hentikan node-nya, dan ajarkan saran yang lebih baik sebagai node baru.

Saran yang mendiagnosis. “Anda terkena demam berdarah” adalah diagnosis; “demam dua hari atau lebih perlu diperiksa di puskesmas hari ini” adalah saran tentang apa yang harus dilakukan. *Hindari:* tulis setiap konklusi sebagai tindakan dan tempat.

Melupakan berkas kata-kata. Label saran baru tanpa baris di `advice-texts.txt` dicetak sebagai label polosnya. *Hindari:* tambahkan kata-katanya sebelum mengajarkan labelnya; pengujian memeriksa bahwa setiap label punya kata-kata.

5.10 Ringkasan Bab

- Pemeriksaan kesehatan mandiri menaruh **pertanyaan keselamatan yang tetap** di depan — tanda stroke, nyeri dada atau sesak napas, tanda bahaya dengue berat, kehilangan terlalu banyak cairan — dan mengarahkan ke 119 atau IGD pada *ya* mana pun; baru setelah itu ia berkonsultasi dengan basis pengetahuan yang dapat belajar (**Gambar 5.1**).
- **Penyempurnaan kasus demi kasus**: untuk setiap kasus yang jawabannya kurang atau keliru, pakar membaca **daftar perbedaan** terhadap **kasus cornerstone** node yang menjawab — bagian A, hanya pada kasus lama; bagian B, hanya pada kasus baru — dan memilih satu dari **enam opsi** (**Tabel 5.1**, **Gambar 5.2**).
- Opsi-opsi membangun node secara mekanis, dan kasus nyata membawa banyak fakta yang tak relevan; pakar dapat **memilih kondisinya sendiri**. Dikerjakan dengan tangan, opsi-opsi itu terlalu spesifik atau terlalu umum, dan `fever days >= 2` pilihan perawat yang tepat (**Kode 5.1**, **Kode 5.2**).
- Sebelum menambahkan node, mesin mendaftar **cornerstone yang terdampak**; dalam sesi mengajar, satu kasus seperti itu membuat perawat menambahkan `cough weeks < 2`, dan sesudahnya setiap cornerstone mempertahankan sarannya (**Kode 5.3**, **Kode 5.4**).

- Node tidak pernah dihapus; saran yang sudah keliru **dihentikan**, dan saran yang lebih baik diajarkan sebagai node baru.
- **Cek kesehatan mandiri** (**Kode 5.5**, **Kode 5.6**) memberikan setiap saran yang berlaku, dengan rule di balik masing-masing, mengirim orang dengan tanda bahaya dengue langsung ke IGD, dan diajari perawat — node, opsi, dan kondisi dipilih olehnya — untuk melaporkan dugaan keracunan makanan, yang lalu ia sarankan kepada tamu berikutnya.

5.11 Soal Latihan

5.1 Kasus cornerstone `cough home care#1` adalah `cough = yes, cough weeks = 1`. Sebuah kasus baru adalah `cough = yes, cough weeks = 1, fever = yes, smoker = no`, dan saran yang tepat adalah saran baru. Tuliskan bagian A dan B daftar perbedaannya, dan node yang akan dibangun masing-masing opsi 1, 2, dan 3.

5.2 Untuk setiap keadaan, sebutkan opsi mana (0 sampai 5) yang akan Anda pakai, dan mengapa. (a) Sarannya sudah benar. (b) Orang dengan ruam dan gatal, tanpa demam, batuk, diare, atau sakit kepala, tidak mendapat saran apa pun. (c) Orang diare yang baru pulang dari daerah wabah kolera mendapat saran oralit, padahal ia harus ke puskesmas hari ini. (d) Orang demam yang digigit anjing mendapat saran demam, padahal ia harus diperiksa untuk rabies, demam atau tidak.

5.3 Pada Langkah 1, opsi 1 membangun `fever days <> 1`. Jelaskan dengan kata-kata pasien mana yang ditangkapnya padahal tidak seharusnya, dan mengapa masalah itu sulit disadari.

5.4 Andaikan pada kasus 8 sesi mengajar Ners Ratna menaruh node berhenti merokok di tingkat atas, dengan satu kondisi `smoker = yes` (opsi 5 dengan kondisi pilihan). Kasus cornerstone mana yang akan terdampak? Apakah perokok pada kasus 7 akan kehilangan pemeriksaan TBC-nya dalam mode banyak-konklusi?

5.5 Seorang rekan mengusulkan memindahkan pertanyaan keselamatan ke dalam basis pengetahuan “agar para perawat dapat memperbaikinya”. Berikan dua alasan untuk menolaknya dan satu hal yang dapat Anda lakukan sebagai gantinya agar perawat dapat mengusulkan perbaikan.

5.6 Pada sesi 1, pemeriksaan mandiri bertanya tentang batuk, diare, dan sakit kepala walaupun Pak Darmo datang karena demam. Jelaskan sebabnya, dari sudut mode banyak-konklusi, dan apa yang berubah bila aplikasi memanggil `infer` tanpa mode itu. Saran mana yang akan terlewat oleh Pak Darmo?

5.7 Dokter puskesmas meminta agar saran sakit kepala tidak lagi menyebut parasetamol. Tulislah program yang memuat basis pengetahuan awal, menghentikan `headache home care#1`, mengajarkan node tingkat atas baru dengan saran `headache rest and water`, dan menampilkan saran untuk orang yang sakit kepala dan untuk orang yang sakit kepala dengan tekanan darah 150, sebelumnya, sesudah penghentian, dan sesudah node baru.

5.8 Fungsi `wrap` aplikasi memecah baris di spasi. Apa yang dilakukannya pada satu kata yang lebih panjang daripada lebarnya, dan mengapa itu dapat diterima di sini?

5.9 Buatlah sketsa pertanyaan keselamatan tetap dan tiga kasus awal pertama untuk pemeriksaan mandiri tersendiri bagi anak di bawah lima tahun, yang diajarkan oleh bidan. Bagian `HealthCheck` mana yang akan Anda pakai ulang tanpa perubahan?

5.12 Penyelesaian Soal Latihan

Penyelesaian 5.1. Bagian A (hanya pada kasus lama) kosong: kedua fakta cornerstone ada pada kasus baru. Bagian B (hanya pada kasus baru) adalah `fever = yes, smoker = no`. Opsi 1 tidak membangun apa pun — mesin menolak, karena daftar kondisi yang kosong tidak dapat membedakan kasusnya. Opsi 2 membangun `fever = yes, smoker = no`; opsi 3, dengan bagian A yang kosong, membangun yang sama. Benar tidaknya bergantung pada saran barunya: bila saran itu tentang batuk *disertai* demam, `fever = yes` adalah alasannya dan `smoker = no` hanya gangguan, dan perawat sebaiknya hanya menyimpan yang pertama.

Penyelesaian 5.2. (a) Opsi 0: terima; pemeriksaannya dicatat. (b) Opsi 5: tidak ada bagian basis pengetahuan yang terkait dengan ruam, jadi kasusnya menjadi node tingkat atas baru — dengan kondisi yang dipilih dengan tangan (`rash = yes`), bukan semua jawaban *no*. (c) Pengecualian `ORS at home#1`, opsi 2 dengan kondisi pilihan `cholera area = yes`, ditaruh di tempat sarannya berasal. (d) Pemeriksaan rabies tidak ada hubungannya dengan demam; sebagai pengecualian node demam, ia hanya akan diberikan kepada orang tergigit yang juga demam. Tempatnya di tingkat atas (opsi 5), dengan kondisi `animal bite = yes` — dan lebih baik lagi, di antara pertanyaan keselamatan, karena gigitan hewan yang mungkin rabies membutuhkan penanganan hari itu juga.

Penyelesaian 5.3. `fever days <> 1` benar untuk setiap angka kecuali satu: untuk 0 (demam yang mulai hari ini), untuk 2, 3, 5, dan 30. Pasien yang demam sejak hari ini dikirim ke pemeriksaan dengue yang belum mereka butuhkan. Masalah ini sulit disadari karena node-nya benar untuk kasus yang membuatnya dan untuk sebagian besar kasus sesudahnya; jawaban kelirunya jarang, dan kelirunya ke arah *berhati-hati*, sehingga tidak ada yang mengeluh. Hanya dengan melihat kondisi node itu kesalahannya tampak.

Penyelesaian 5.4. Mesin memeriksa kondisi node baru terhadap setiap cornerstone: `smoker = yes` terpenuhi oleh cornerstone `TB check#1`, perokok dengan batuk tiga minggu, dan di tingkat atas tidak ada kondisi induk yang membatasinya. Jadi `TB check#1` akan terdaftar sebagai terdampak. Dalam mode banyak-konklusi, perokok pada kasus 7 *tidak* akan kehilangan pemeriksaan TBC-nya: mesin mengikuti setiap cabang, jadi ia akan mendapat pemeriksaan TBC (dari cabang batuk) sekaligus bantuan berhenti merokok (dari node tingkat atas yang baru). Itu sebenarnya saran yang masuk akal, dan Ners Ratna mungkin menerima kasus terdampak itu. Namun dalam mode satu-konklusi, urutan node tingkat atas yang akan menentukan mana yang ia dapat — satu alasan lagi untuk melihat setiap kasus terdampak alih-alih mempercayai jumlahnya saja.

Penyelesaian 5.5. Pertama, pengetahuan yang dapat belajar juga dapat mempelajari kesalahan: pengecualian yang diajarkan terburu-buru dapat mengarahkan tanda bahaya ke perawatan di rumah, dan tidak ada yang tahu sampai seorang pasien dirugikan. Kedua, pertanyaan keselamatan berasal dari panduan yang diterbitkan [1], [6], [7], dan harus berubah ketika panduannya berubah, dengan sumbernya diperiksa — proses yang berbeda dari belajar dari kasus. Sebagai gantinya, biarkan perawat *mengusulkan* perubahan: aplikasi dapat menyimpan catatan sederhana tentang pertanyaan tanda bahaya yang membingungkan orang, atau kasus ketika sebuah tanda bahaya terlewat, yang ditelaah tim bersama sumbernya sebelum program diubah.

Penyelesaian 5.6. Dengan `multiple` dinyalakan, mesin mencoba setiap node tingkat atas, dan masing-masing membutuhkan jawaban atas kondisi pertamanya sendiri: `cough`, `diarrhea`, `headache`. Tanpa itu, mesin berhenti pada cabang pertama yang menjawab — cabang demam,

yang berakhir di `dengue check#1` — dan tidak menanyakan apa-apa lagi. Pak Darmo akan kehilangan saran sakit kepala dan, yang lebih penting, pertanyaan tekanan darah yang dibuka oleh node sakit kepala.

Penyelesaian 5.7. Program penyelesaiannya adalah `StopDemo.java` dan `stop_demo.py`. Versi Python-nya:

```
"""Soal 5.7: mengistirahatkan sebuah saran tanpa menghapusnya: hentikan
node-nya, dan ajarkan saran yang lebih baik sebagai node baru."""

import pathlib
import sys

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import TOP_LEVEL, Case, Condition, KnowledgeBase, casefile


PEOPLE = [{"headache": "yes"}, {"headache": "yes", "blood pressure": "150"}]


def show(kb, when):
    for i, facts in enumerate(PEOPLE):
        got = [c.value for c in kb.infer(facts, multiple=True).conclusions()]
        print(when + ", person " + str(i + 1) + ": "
              + (", ".join(got) or "(no advice)"))


name, description, steps = casefile.read_cases("../data/health-seed-cases.json")
kb = casefile.build(KnowledgeBase(name, description), steps)
show(kb, "before")
kb.stop("headache home care#1", "Ratna")
show(kb, "stopped")
case = Case.of(["headache = yes"], ["advice = headache rest and water"])
node = kb.refine(case, TOP_LEVEL, chosen=[Condition.parse("headache = yes")],
                 author="Ratna")
print("new node: " + str(node))
show(kb, "after")
print("nodes kept: " + str(len(kb.nodes)) + ", history: "
      + ", ".join(h["action"] for h in kb.history[-2:]))
```

Program ini mencetak:

```
before, person 1: headache home care
before, person 2: blood pressure check
stopped, person 1: (no advice)
stopped, person 2: blood pressure check
new node: headache rest and water#1: headache = yes -> advice = headache rest and water
after, person 1: headache rest and water
after, person 2: blood pressure check, headache rest and water
nodes kept: 10, history: stop, add
```

Sesudah penghentian, orang yang hanya sakit kepala tidak mendapat saran — node yang dihentikan menyala, tetapi tidak memberi apa-apa — sedangkan orang dengan hasil ukur tinggi tetap mendapat pemeriksaan tekanan darah, karena pengecualian node yang dihentikan tetap bekerja. Sesudah node baru, keduanya mendapat saran baru. Basis pengetahuan tetap memiliki semua node-nya (sepuluh, dengan yang baru), dan riwayatnya berakhir dengan penghentian dan penambahan itu.

Penyelesaian 5.8. Kata yang lebih panjang daripada lebarnya menjadi baris tersendiri, lebih panjang daripada lebarnya: `wrap` tidak pernah memotong di tengah kata. Itu dapat diterima

karena teksnya kalimat berisi kata-kata biasa; "kata" panjang hanyalah beberapa label dan angka, yang lebih baik dibiarkan utuh daripada dipotong di tengah.

Penyelesaian 5.9. Pertanyaan keselamatannya mengikuti panduan khusus anak yang dipilih bidan — misalnya tanda bahaya umum yang dilatihkan kepada tenaga kesehatan untuk anak kecil, seperti anak yang tidak mau minum atau menyusu atau yang baru kejang, yang masing-masing berarti *berangkat sekarang* — dan pemeriksaan pertamanya adalah usia anak. Kasus awalnya mungkin *demam* → perawatan di rumah untuk anak; *diare* → oralit dan zinc, seperti dianjurkan WHO untuk anak [4]; dan *batuk dengan napas cepat* → puskesmas hari ini. Semua bagian `HealthCheck` kecuali konstantanya (`ADULT`, `RED_FLAGS`, `QUESTIONS`), berkas kasus awal, dan berkas kata-kata dapat dipakai ulang tanpa perubahan: perulangan keselamatan, pembungkus kata, konsultasi dalam mode banyak-konklusi, dan seluruh layar mengajar.

Daftar Pustaka

[1] World Health Organization, "Dengue and severe dengue," fact sheet, Aug. 21, 2025. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/dengue-and-severe-dengue

[2] World Health Organization, "Hypertension," fact sheet, Sep. 25, 2025. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/hypertension

[3] World Health Organization, "Tuberculosis," fact sheet, 2025. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/tuberculosis

[4] World Health Organization, "Diarrhoeal disease," fact sheet. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/diarrhoeal-disease

[5] Kementerian Kesehatan Republik Indonesia, "Public Safety Center 119," layanan gawat darurat medis. [Online]. Available: https://psc.kemkes.go.id/web/

[6] Kementerian Kesehatan Republik Indonesia, "Kenali gejala stroke dengan SeGeRa Ke RS," *Ayo Sehat*, Oct. 28, 2023. [Online]. Available: https://ayosehat.kemkes.go.id/kenali-gejala-stroke-dengan-segera-ke-rs

[7] M. D. B. Kholil, "Demam berdarah dengue: kapan harus ke IGD," Direktorat Jenderal Pelayanan Kesehatan, Kementerian Kesehatan RI, Jan. 10, 2025. [Online]. Available: https://keslan.kemkes.go.id/view_artikel/3932/demam-berdarah-dengue-kapan-harus-ke-igd

[8] Kementerian Komunikasi dan Digital Republik Indonesia, "Tentang call center 112," Layanan Panggilan Darurat 112. [Online]. Available: https://layanan112.komdigi.go.id/tentang

[9] I. Subakti and R. Hidayatullah, "Aplikasi sistem pakar untuk diagnosis awal gangguan kesehatan secara mandiri menggunakan Variable-Centered Intelligent Rule System," vol. 6, no. 1, pp. 11–16, Jan. 2007.

[10] Kementerian Kesehatan Republik Indonesia, "Ayo kenali gejala tuberkulosis yang perlu diwaspadai, mulai dari batuk hingga berat badan menurun," *Ayo Sehat*, Mar. 15, 2024. [Online]. Available: https://ayosehat.kemkes.go.id/ayo-kenali-gejala-tuberkulosis-yang-perlu-diwaspadai-mulai-dari-batuk-hingga-berat-badan-menurun

[11] P. Compton and B. H. Kang, *Ripple-Down Rules: The Alternative to Machine Learning*. Boca Raton, FL, USA: CRC Press, 2021.

[12] P. Compton, G. Edwards, B. Kang, L. Lazarus, R. Malor, T. Menzies, P. Preston, A. Srinivasan, and C. Sammut, "Ripple down rules: possibilities and limitations," in *Proc. 6th AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1991, pp. 6.1–6.20.

[13] M. M. I. Subakti, "A Variable-Centered Intelligent Rule System," M.Sc.Eng. thesis, Dept. of Computer Science and Information Engineering, Taipei, Taiwan, Jun. 2005.

[14] B. H. Kang, P. Compton, and P. Preston, "Multiple classification ripple down rules: evaluation and possibilities," in *Proc. 9th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1995, pp. 17.1–17.20.

[15] I. Subakti, "Some revisions in VCIRS and cases reconstructing perspectives," in *Proc. Information and Communication Technology Seminar (ICTS 2006)*, Surabaya, Indonesia, 2006.

[16] I. Subakti, *Variable-Centered Intelligent Rule System*, Indonesian ed. (resume). Surabaya, Indonesia: Dept. of Informatics, Institut Teknologi Sepuluh Nopember, 2006.

[17] D. Richards, "Two decades of ripple down rules research," *The Knowledge Engineering Review*, vol. 24, no. 2, pp. 159–184, 2009.

[18] Kementerian Kesehatan Republik Indonesia, "Ingin berhenti merokok? Hubungi layanan konseling bebas biaya 0-800-177-6565." [Online]. Available: https://kemkes.go.id/id/ingin-berhenti-merokok-hubungi-layanan-konseling-bebas-biaya

[19] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*. Jakarta, Indonesia, 2022.

Bab 6

Penasihat Masalah Tanaman dan Pohon Buah untuk Petani: Pembangkitan Rule

Di akhir bab ini Anda akan mampu menjelaskan apa artinya sepotong pengetahuan *dipakai*, bukan sekadar disimpan; menghitung, dengan tangan maupun dengan mesin, tiga **tingkat pemakaian** (*usage rate*) VCIRS — Variable Usage Rate (VUR), Node Usage Rate (NUR), dan Rule Usage Rate (RUR) — dari Credit, jumlah pemakaian bersama, dan derajat kedekatan setiap kondisi; menyebutkan bagaimana tingkat-tingkat itu menempatkan node-node sebuah basis pengetahuan dalam **urutan relatif**; memakai urutan itu untuk **membangkitkan** kandidat rule dan node; serta menelaah kandidat-kandidat itu seperti seorang pakar, hanya menyimpan yang benar. Anda akan menjalankan **dokter tanaman dan pohon buah**, aplikasi dalam Java dan Python yang membantu petani menghadapi masalah durian, mangga, cabai, dan padi, belajar dari penyuluh pertanian desa, menampilkan tingkat pemakaiannya, dan membiarkan penyuluh mengubah sebuah kandidat hasil bangkitan menjadi saran baru yang lebih umum, yang menjawab kebun-kebun yang tak pernah diajarkan kepadanya.

6.1 Kebun Pak Slamet

Keluarga Pak Slamet punya kebun buah dan durian di lereng di atas desa: empat puluh pohon durian yang ditanam ayahnya, sederet pohon mangga di sepanjang jalan setapak, dan di bawahnya beberapa bedeng cabai serta sepetak sawah yang digarap bersama adiknya. Dalam satu hal yang penting, inilah kebun khas Indonesia. Sensus Pertanian 2023 mencatat 28,4 juta rumah tangga usaha pertanian di Indonesia, dan 16,89 juta di antaranya — enam dari sepuluh — mengusahakan lahan kurang dari setengah hektare [1]. Kebun seperti ini tidak sanggup kehilangan satu panen pun, dan tidak sanggup membayar ahli yang selalu siap dipanggil.

Setiap musim membawa masalahnya sendiri. Sehabis hujan, bercak gelap dan basah muncul di kulit batang salah satu durian tua dan pelan-pelan meluas; penyakitnya, jamur air bernama *Phytophthora palmivora*, menyebabkan kanker batang, busuk akar, dan busuk buah, dan di sebagian Asia Tenggara ia telah menginfeksi sampai 30 persen pohon durian dan memangkas hasil seperlima atau lebih [2]. Mangganya mendapat bercak hitam cekung ketika mulai masak — antraknosa, jamur yang membusukkan buah di pohon dan sesudah panen [3] — dan sebagian ditusuk lalat buah, yang belatungnya membusukkan daging buah dari dalam [4]. Daun cabainya keriting dan menguning di tempat kutu kebul membawa virus kuning keriting, yang dapat merenggut sampai 80 persen panen [5]. Dan di sawah, wereng batang cokelat berkumpul di pangkal tanaman, kadang begitu banyak sampai sepetak-sepetak sawah mengering dan roboh, ledakan yang paling sering justru disebabkan penyemprotan insektisida berlebihan, yang membunuh laba-laba dan kumbang pemangsanya [6].

Mbak Nur adalah penyuluh pertanian desa itu. Ia tahu semua ini, dan ia melayani lima desa dengan sepeda motor. Pada hari pasar ia duduk di pos kelompok tani dan menjawab pertanyaan yang sama berulang-ulang: *bercak apa ini, apa yang harus saya lakukan dengan bercak di batang*

ini, mengapa daun cabai saya keriting? Ia ingin jawaban-jawaban itu tersedia ketika ia tidak ada. Dan ia menyadari sesuatu tentang pengetahuannya sendiri: banyak di antaranya sama sekali tidak khusus untuk satu tanaman. Bercak hitam cekung antraknosa tampak sama pada mangga, cabai, pepaya, dan buah naga, karena jenis jamur yang sama menyebabkannya. Ia mengajarkan kasus mangga dan kasus cabai secara terpisah; petani dengan pepaya berbercak tetap tidak mendapat jawaban.

Bab ini membangun dokter yang diinginkan Mbak Nur, dan mengajarkan bagian VCIRS yang menyadari persis apa yang ia sadari. VCIRS terus menghitung bagaimana pengetahuannya dipakai. Dari hitungan itu ia dapat melihat kondisi dan node mana yang paling berbobot, dan ia dapat **membangkitkan** kandidat rule dan node baru dengan menggabungkannya — kandidat yang lalu dikonfirmasi atau ditolak pakar. Begitulah dokter itu belajar, dengan persetujuan Mbak Nur, bahwa bercak hitam cekung pada buah *apa pun* membutuhkan penanganan antraknosa.

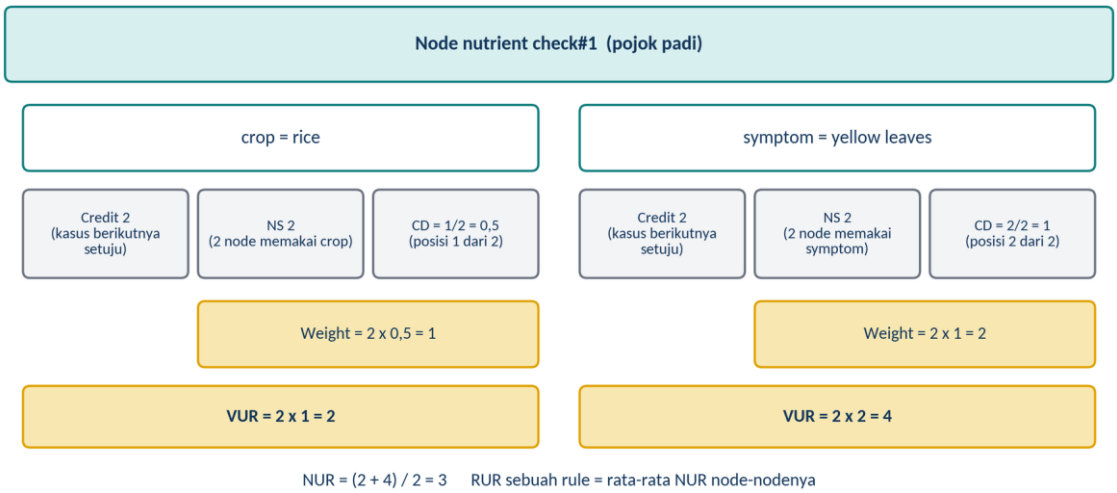
6.2 Gagasannya dengan Kata-Kata Sederhana

6.2.1 Pengetahuan yang dipakai

Basis pengetahuan yang dibangun kasus demi kasus tumbuh di tempat-tempat asal kasusnya. Sebagian kondisi muncul di banyak node; sebagian node menjadi awal banyak rule; sebagian kondisi berkali-kali dikonfirmasi oleh kasus-kasus berikutnya. Yang lain ditulis sekali dan tidak pernah disentuh lagi. Tesis yang mengusulkan VCIRS menyebut analisis atas hal ini **analisis nilai** (*value analysis*), atau *usage assignment*, dan memberinya tiga tujuan [7], [8], [9]:

1. menjadi pedoman bagi pengguna saat membangun pengetahuan tentang rule, node, atau variabel mana yang dikunjungi lebih dulu;
2. menjadi pedoman bagi pengguna dan sistem saat inferensi dengan cara yang sama, sehingga pengetahuan yang paling banyak dipakai dicoba lebih dulu;
3. menjadi pedoman bagi sistem, bersama analisis variabel di Bab 4, dalam **pembangkitan rule** (*rule generation*).

Ukurannya tiga angka: **Variable Usage Rate** (VUR) untuk setiap kondisi dalam setiap node, **Node Usage Rate** (NUR) untuk setiap node, dan **Rule Usage Rate** (RUR) untuk setiap rule. Makin besar angkanya, makin banyak potongan pengetahuan itu dipakai [7]. Ketiganya dibangun dari tiga bahan sederhana, dan kita akan menemuinya satu per satu di pojok terkecil basis pengetahuan Mbak Nur: padi. Pojok ini punya empat node: `nutrient check#1` (padi berdaun kuning), pengecualiannya `planthopper control#1` (wereng cokelat di pangkal), pengecualian node itu `planthopper emergency#1` (petak-petak mengering), dan `blast control#1` (padi dengan bercak berbentuk belah ketupat). **Gambar 6.1** memperlihatkan seluruh perhitungan untuk node pertama.



Gambar 6.1 Cara menghitung tingkat pemakaian sebuah node. Variable Usage Rate setiap kondisi adalah Credit-nya dikali Weight-nya, dan Weight adalah banyaknya node yang berbagi variabel itu dikali kedekatan kondisi ke konklusi. Tingkat pemakaian node adalah rata-rata tingkat pemakaian kondisi-kondisinya.

6.2.2 Credit: seberapa sering sebuah kondisi disetujui

Ketika Mbak Nur mengajarkan `planthopper control#1`, kasus yang ia pakai — padi, daun kuning, wereng cokelat — *menyetujui* kedua kondisi node induknya, `crop = rice` dan `symptom = yellow leaves`. VCIRS mencatat persetujuan itu: setiap kondisi induk yang disetujui kasus baru mendapat satu **Credit** lagi [7]. Setiap kondisi mulai dengan Credit 1, saat node-nya dibuat; setelah kasus wereng, kedua kondisi `nutrient check#1` ber-Credit 2. Dengan cara yang sama, ketika ia mengajarkan node darurat di bawah `planthopper control#1`, kondisi `insects = brown hoppers` mendapat Credit keduanya.

$$\text{Credit}_i = \text{banyaknya kali kondisi } i \text{ dipakai dalam Node Structure} \tag{6.1}$$

Dengan kata-kata sederhana, Persamaan (6.1) menyatakan bahwa kondisi yang dilewati banyak kasus berikutnya, dalam perjalanan menuju jawaban yang lebih spesifik, adalah kondisi yang diandalkan pengetahuan itu.

6.2.3 NS, CD, dan Weight: seberapa dibagi, seberapa dekat

Dua hal lagi membuat sebuah kondisi lebih berbobot. Yang pertama adalah berapa banyak node yang memakai variabelnya: **NS** (*number of nodes sharing*) variabel itu [7]. Di pojok padi, `crop` dipakai dua node (`nutrient check#1` dan `blast control#1`), demikian pula `symptom`; `insects` dan `patches` masing-masing dipakai satu node. Inilah analisis variabel Bab 4, diringkas menjadi satu angka per variabel.

Yang kedua adalah letak kondisi itu di dalam node-nya. Node adalah rangkaian kondisi, dan tesis berpendapat bahwa makin dekat sebuah variabel ke konklusi node-nya, makin baik — kondisi terakhir sebuah node adalah yang akhirnya menentukannya [7]. **Closeness Degree** (CD, derajat kedekatan) mengukurnya sebagai posisi kondisi, yaitu urutan variabelnya VO, dibagi jumlah seluruh kondisi TV di node itu:

$$\text{CD}_i = \frac{\text{VO}_i}{\text{TV}} \tag{6.2}$$

Di `nutrient check#1`, `crop = rice` adalah kondisi 1 dari 2, jadi CD-nya $\frac{1}{2}$; `symptom = yellow leaves` adalah kondisi 2 dari 2, jadi CD-nya 1. Node dengan satu kondisi memberinya CD 1. **Weight** (bobot) sebuah kondisi menggabungkan keduanya:

$$\text{Weight}_i = \text{NS}_i \times \text{CD}_i \tag{6.3}$$

Jadi `crop = rice` di `nutrient check#1` berbobot $2 \times 0.5 = 1$, dan `symptom = yellow leaves` berbobot $2 \times 1 = 2$.

6.2.4 VUR, NUR, dan RUR

Sekarang ketiga tingkat pemakaian menyusul dalam tiga baris. Variable Usage Rate sebuah kondisi adalah Credit-nya dikali Weight-nya:

$$\text{VUR}_i = \text{Credit}_i \times \text{Weight}_i \tag{6.4}$$

Node Usage Rate sebuah node adalah rata-rata VUR dari N kondisinya, dan Rule Usage Rate sebuah rule adalah rata-rata NUR dari N node-nya [7], [8]:

$$\text{NUR}_j = \frac{1}{N}\sum_{i=1}^{N} \text{VUR}_{ij} \tag{6.5}$$

$$\text{RUR}_k = \frac{1}{N}\sum_{j=1}^{N} \text{NUR}_{jk} \tag{6.6}$$

Untuk `nutrient check#1`: VUR `crop = rice` adalah $2 \times 1 = 2$ dan VUR `symptom = yellow leaves` adalah $2 \times 2 = 4$, jadi NUR-nya $(2 + 4)/2 = 3$. Rule `planthopper emergency#1` adalah rangkaian `nutrient check#1 -> planthopper control#1 -> planthopper emergency#1`; bila dua node lainnya ber-NUR 2 dan 1, RUR-nya $(3 + 2 + 1)/3 = 2$. Subbab 6.3 menghitung setiap angka pojok padi dengan cara ini dan mencocokkannya dengan mesin.

Layak dikatakan dengan terang apa yang *bukan* diukur tingkat-tingkat ini. Mereka tidak menghitung berapa petani yang telah dijawab sebuah node, dan mereka bukan peluang. Mereka mengukur seberapa besar *struktur* pengetahuan bersandar pada setiap potongannya: seberapa dibagi, seberapa menentukan letaknya, dan seberapa sering pengajaran berikutnya melewatinya. Itulah tepat yang dibutuhkan untuk memutuskan potongan mana yang layak digabungkan menjadi sesuatu yang baru.

> **WAWASAN KUNCI:** Tingkat pemakaian didapat cuma-cuma. Ia dihitung dari apa yang sudah disimpan Variable-Centered Rule Structure — urutan kondisi, node mana berbagi variabel mana, dan Credit yang diperoleh selama pakar mengajar — dan dihitung ulang setiap kali pengetahuannya berubah.

6.2.5 Dari pemakaian ke pembangkitan

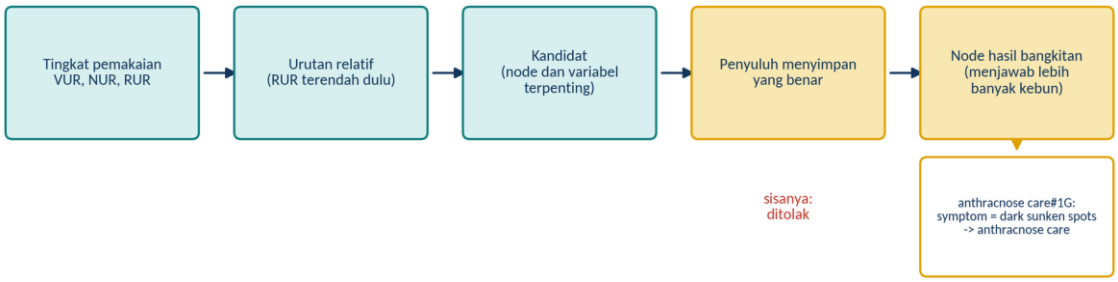
Basis pengetahuan yang dibangun dari kasus bersifat **tepat tetapi sempit**. Setiap node dibuat untuk kasus nyata dan tidak mengganggu kasus sebelumnya, sehingga jawabannya biasanya benar; tetapi ia hanya menjawab yang telah diajarkan, dan keadaan yang benar-benar baru tidak mendapat jawaban sama sekali [10]. Pepaya berbercak adalah keadaan seperti itu: Mbak Nur

mengajarkan antraknosa untuk mangga dan untuk cabai, masing-masing node dijaga oleh jenis tanamannya, dan pepaya bukan keduanya.

Pembangkitan rule adalah jawaban VCIRS atas kesempitan itu [7], [8]. Ia melihat pengetahuan yang paling penting — node yang dibagi rule terbanyak dan variabel yang dibagi node terbanyak, hasil analisis variabel Bab 4 — dan mengusulkan gabungan-gabungan baru darinya, dengan urutan yang tidak merusak urutan mana pun yang sudah ada dalam basis pengetahuan. Dua hal dibangkitkan:

- **rule hasil bangkitan** adalah rangkaian baru node-node yang sudah ada, yang berakhir pada salah satu node terpenting;
- **node hasil bangkitan** adalah rangkaian baru kondisi-kondisi yang sudah ada, yang berakhir pada salah satu variabel terpenting, dengan konklusi node asalnya.

Keduanya ditandai `G` pada namanya (`anthracnose care#1G`) dan tidak satu pun disimpan sebelum pakar mengonfirmasinya — dan ia boleh menyimpan hanya sebagian. **Gambar 6.2** memperlihatkan jalan dari tingkat pemakaian ke node baru.



Gambar 6.2 Pembangkitan rule dalam VCIRS. Tingkat pemakaian memberi urutan relatif pengetahuan; urutan itu serta node dan variabel terpenting memberi kandidat; pakar menyimpan yang benar, dan node baru menjawab kasus yang tak dapat dijawab node mana pun yang diajarkan.

6.2.6 Urutan relatif

Dari mana urutan sebuah kandidat berasal? Mesin menelusuri rule-rule mulai dari yang **RUR-nya terendah** — yang paling sedikit dipakai — dan mendaftar node-node-nya; setiap kali sebuah node dibagi dengan rule lain, ia ikut membawa node-node awal rule-rule itu, ditempatkan menurut posisinya dan, bila posisinya sama, menurut NUR yang lebih rendah [7]. Hasilnya adalah **urutan node relatif**: satu daftar berisi semua node basis pengetahuan, disusun sehingga urutan rule mana pun tidak rusak. Rule hasil bangkitan untuk sebuah node penting adalah bagian daftar ini yang datang sebelum node itu, diakhiri node itu sendiri. Penelusuran yang sama atas kondisi, mulai dari node dengan NUR terendah dan memakai VUR, memberi **urutan variabel relatif**, dan node hasil bangkitan adalah bagian daftar itu yang datang sebelum salah satu variabel terpenting [7].

Kedengarannya rumit, dan algoritma lengkapnya punya selusin langkah, tetapi tujuannya sederhana: menjajarkan semua yang diketahui basis pengetahuan, dari yang paling sedikit dipakai sampai yang paling banyak, dengan urutan yang menghormati setiap rule. Sebuah kandidat lalu hanyalah “semua yang datang sebelum butir penting ini, menurut urutan itu”.

Ambillah pojok padi dari Subbab 6.2.1 sebagai contoh kecil. Rule-nya yang RUR-nya terendah adalah `blast control#1` (1,50, seperti dihitung Subbab 6.3), jadi penelusuran dimulai di sana, dan `blast control#1` menjadi yang pertama. Yang terendah berikutnya adalah rule darurat (2,00), dan node-node-nya menyusul dengan urutannya sendiri: `nutrient check#1`, `planthopper control#1`, `planthopper emergency#1`. Node pertamanya dibagi dengan dua rule lain, tetapi

node-node awal rule-rule itu sudah ada di daftar, jadi tidak ada yang bergeser. Urutan node relatif pojok padi karenanya `blast control#1`, `nutrient check#1`, `planthopper control#1`, `planthopper emergency#1`. Node terpentingnya `nutrient check#1`, yang dibagi tiga rule, dan kandidat rule-nya adalah bagian urutan sampai node itu: `blast control#1 -> nutrient check#1`, padi berbercak belah ketupat lalu padi berdaun kuning. Kandidat node untuk node yang sama, dibangun dengan cara yang sama dari urutan variabel relatif, adalah `patches = drying, crop = rice, symptom = diamond spots on leaves, symptom = yellow leaves -> nutrient check`. Pakar yang membacanya akan menyimpan `crop = rice` dan `symptom = yellow leaves` — yang tak lain node yang sudah ada — atau `symptom = yellow leaves` saja, aturan nitrogen umum untuk setiap tanaman, yang menurut Subbab 6.4 selangkah terlalu jauh. Bahkan pada empat node, kandidat-kandidatnya memuat satu gagasan baik, sedikit pengulangan, dan sedikit omong kosong; memilahnya adalah pekerjaan pakar.

6.2.7 Pakar punya kata terakhir

Sebuah kandidat adalah usulan, bukan pengetahuan. Sebagian besar isinya biasanya keliru untuk konklusinya: pada demonstrasi Zookeeper, tesis mendapati bahwa antara 36 dan 50 persen node dalam setiap rule hasil bangkitan masuk akal, dan pengguna mencoret sisanya [7]. Itu bukan cacat. Mesin tidak dapat tahu apa yang benar secara pertanian atau kedokteran; ia hanya dapat memperhatikan apa yang dipakai bersama. Pakar membaca kandidat, menyimpan kondisi atau node yang memang termasuk, dan menolak sisanya. Ketika ia hanya menyimpan sedikit kondisi, seperti dilakukan Mbak Nur di bawah, node hasil bangkitan menjadi *lebih umum* daripada node-node asalnya, dan ia menjawab kasus yang tak dapat dijawab satu pun dari mereka.

Dalam pertanian, taruhan telaah ini nyata. Generalisasi yang terlalu luas mengarahkan petani ke obat yang salah: pohon yang akarnya tenggelam diberi pupuk, sawah berwereng disemprot insektisida yang salah, panen hilang sementara masalah sebenarnya menyebar. Sebaliknya, generalisasi yang benar membawa pengalaman penyuluh ke tanaman dan kebun yang tak sempat ia ajarkan. Sumbangan mesin adalah *menyadari* pola di antara puluhan node, yang tak dapat dilakukan orang dengan mata begitu basis pengetahuan tumbuh; sumbangan pakar adalah *menilainya*. Tidak ada yang cukup sendirian, dan VCIRS dibangun agar yang kedua selalu menyusul yang pertama.

> **DEFINISI.** *Pembangkitan rule* dalam VCIRS mengusulkan rule baru (rangkaian node yang sudah ada, berakhir pada node terpenting) dan node baru (rangkaian kondisi yang sudah ada, berakhir pada variabel terpenting), diurutkan menurut urutan relatif yang dihasilkan tingkat pemakaian. Setiap kandidat ditandai `G` dan hanya disimpan dengan bagian-bagian yang dikonfirmasi pakar.

6.3 Langkah 1: Tingkat Pemakaian dengan Tangan

Kode 6.1 menghitung setiap tingkat pemakaian pojok padi dengan tangan, mencetak setiap bahannya, lalu membangun keempat node yang sama di atas mesin dan memeriksa bahwa mesin memperoleh angka-angka yang sama.

Kode 6.1 `UsageByHand.java`: Credit, NS, CD, Weight, VUR, NUR, dan RUR dengan tangan, dicocokkan dengan vcirs-core.

```java
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;

import vcirs.Case;
import vcirs.KnowledgeBase;
import vcirs.Usage;

/**
 * Credit, NS, CD, Weight, VUR, NUR, dan RUR dihitung dengan tangan pada
 * pojok padi dokter tani, lalu dicocokkan dengan mesin vcirs-core.
 */
public class UsageByHand {
    /** Kondisi sebuah node: variabelnya dan Credit-nya. */
    record Cond(String variable, int credit) { }

    /** Sebuah node: id-nya, induknya (atau null), dan kondisinya. */
    record HandNode(String id, String parent, List<Cond> conds) { }

    // Credit 2 berarti sebuah kasus berikutnya sekali menyetujui kondisinya.
    static final List<HandNode> NODES = List.of(
            new HandNode("nutrient check#1", null,
                    List.of(new Cond("crop", 2), new Cond("symptom", 2))),
            new HandNode("planthopper control#1", "nutrient check#1",
                    List.of(new Cond("insects", 2))),
            new HandNode("planthopper emergency#1", "planthopper control#1",
                    List.of(new Cond("patches", 1))),
            new HandNode("blast control#1", null,
                    List.of(new Cond("crop", 1), new Cond("symptom", 1))));

    static final Map<String, Double> vur = new LinkedHashMap<>();
    static final Map<String, Double> nur = new LinkedHashMap<>();
    static final Map<String, Double> rur = new LinkedHashMap<>();

    static String f2(double x) {
        return String.format(Locale.ROOT, "%.2f", x);  // selalu titik
    }
```

Kode 6.1 (Lanjutan) *UsageByHand.java: Credit, NS, CD, Weight, VUR, NUR, dan RUR dengan tangan, dicocokkan dengan vcirs-core.*

```
    static HandNode find(String id) {
        for (HandNode n : NODES) {
            if (n.id().equals(id)) {
                return n;
            }
        }
        throw new IllegalArgumentException(id);
    }

    static void byHand() {
        Map<String, Integer> ns = new LinkedHashMap<>();  // NS: node berbagi
        for (HandNode n : NODES) {
            for (Cond c : n.conds()) {
                ns.merge(c.variable(), 1, Integer::sum);
            }
        }
        System.out.println("Node                    Variable  Credit  NS  CD"
                + "    Weight  VUR");
        for (HandNode n : NODES) {
            double total = 0;
            for (int i = 0; i < n.conds().size(); i++) {
                Cond c = n.conds().get(i);
                double cd = (i + 1.0) / n.conds().size();    // CD = VO / TV
                double weight = ns.get(c.variable()) * cd;   // Weight = NS x CD
                double v = c.credit() * weight;              // VUR = Credit x W
                vur.put(n.id() + "/" + i, v);
                total += v;
                System.out.println(String.format(Locale.ROOT,
                        "%-25s%-10s%-8d%-4d%-6s%-8s%s", i == 0 ? n.id() : "",
                        c.variable(), c.credit(),
                        ns.get(c.variable()), f2(cd), f2(weight), f2(v)));
            }
            nur.put(n.id(), total / n.conds().size());   // NUR = rata-rata VUR
        }
        System.out.println();
        for (HandNode n : NODES) {
            List<String> parts = new ArrayList<>();
            for (int i = 0; i < n.conds().size(); i++) {
                parts.add(f2(vur.get(n.id() + "/" + i)));
            }
            System.out.println("NUR " + n.id() + " = (" + String.join(" + ", parts)
                    + ") / " + n.conds().size() + " = " + f2(nur.get(n.id())));
        }
        for (HandNode n : NODES) {
            List<String> path = new ArrayList<>(List.of(n.id()));
            String p = n.parent();
```

Kode 6.1 (Lanjutan) *UsageByHand.java: Credit, NS, CD, Weight, VUR, NUR, dan RUR dengan tangan, dicocokkan dengan vcirs-core.*

```
            while (p != null) {                    // rule: jalur ke bawah
                path.add(0, p);
                p = find(p).parent();
            }
            double sum = 0;
            List<String> parts = new ArrayList<>();
            for (String id : path) {
                sum += nur.get(id);
                parts.add(f2(nur.get(id)));
            }
            rur.put(n.id(), sum / path.size());
            System.out.println("RUR " + n.id() + " = (" + String.join(" + ", parts)
                    + ") / " + path.size() + " = " + f2(rur.get(n.id())));
        }
    }

    static Usage byEngine() {
        KnowledgeBase kb = new KnowledgeBase("rice corner");
        String[][] cases = {
            {"nutrient check", "crop = rice", "symptom = yellow leaves"},
            {"planthopper control", "crop = rice", "symptom = yellow leaves",
                "insects = brown hoppers"},
            {"planthopper emergency", "crop = rice", "symptom = yellow leaves",
                "insects = brown hoppers", "patches = drying"},
            {"blast control", "crop = rice", "symptom = diamond spots on leaves"},
        };
        for (String[] c : cases) {
            List<String> facts = List.of(c).subList(1, c.length);
            kb.learn(Case.of(facts, List.of("advice = " + c[0])));
        }
        return kb.usage();
    }

    public static void main(String[] args) {
        byHand();
        Usage u = byEngine();
        int same = 0;
        int total = 0;
        for (Map.Entry<String, Double> e : vur.entrySet()) {
            String[] key = e.getKey().split("/");
            same += Math.abs(u.vur(key[0], Integer.parseInt(key[1]))
                    - e.getValue()) < 1e-9 ? 1 : 0;
            total++;
        }
        for (Map.Entry<String, Double> e : nur.entrySet()) {
            same += Math.abs(u.nur(e.getKey()) - e.getValue()) < 1e-9 ? 1 : 0;
```

Kode 6.1 (Lanjutan) *UsageByHand.java: Credit, NS, CD, Weight, VUR, NUR, dan RUR dengan tangan, dicocokkan dengan vcirs-core.*

```
            total++;
        }
        for (Map.Entry<String, Double> e : rur.entrySet()) {
            same += Math.abs(u.rur(e.getKey()) - e.getValue()) < 1e-9 ? 1 : 0;
            total++;
        }
        System.out.println();
        System.out.println("The engine agrees on " + same + " of " + total
                + " numbers.");
    }
}
```

`NODES` menggambarkan pojok padi seperti Node Structure tesis menggambarkannya: setiap node dengan induknya dan kondisi-kondisinya, dan setiap kondisi dengan Credit-nya. Credit-nya ditulis langsung: 2 untuk kondisi yang disetujui kasus berikutnya, 1 untuk yang lain, tepat seperti dijelaskan Subbab 6.2.2.

`byHand` mengikuti persamaan-persamaan satu per satu. Ia menghitung NS setiap variabel dengan menelusuri semua kondisi semua node. Untuk setiap kondisi ia menghitung CD sebagai posisi dibagi jumlah kondisi (Persamaan (6.2)), Weight sebagai NS kali CD (Persamaan (6.3)), dan VUR sebagai Credit kali Weight (Persamaan (6.4)), lalu mencetak satu baris tabel. NUR setiap node adalah rata-rata VUR-nya (Persamaan (6.5)). Untuk RUR, ia membangun ulang rule setiap node dengan berjalan naik melewati induk-induknya, seperti di Bab 4, dan mengambil rata-rata NUR di sepanjangnya (Persamaan (6.6)), mencetak setiap jumlahnya agar Anda dapat mengikuti hitungannya. `f2` mencetak angka dengan dua desimal; `Locale.ROOT` membuat Java menulis titik desimal di setiap komputer, bahkan yang diatur untuk bahasa Indonesia, yang bawaannya memakai koma.

`byEngine` membangun pojok padi seperti Mbak Nur akan mengajarkannya, dengan empat kasus dan `kb.learn`, dan `main` membandingkan keempat belas angka dengan `usage().vur`, `nur`, dan `rur` milik mesin. **Kode 6.2** adalah versi Python-nya.

Kode 6.2 `usage_by_hand.py`: perhitungan yang sama dalam Python.

```
"""Credit, NS, CD, Weight, VUR, NUR, dan RUR dihitung dengan tangan pada
pojok padi dokter tani, lalu dicocokkan dengan mesin vcirs-core."""

import pathlib
import sys

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import Case, KnowledgeBase

# Tiap node: id-nya, induknya, dan kondisinya sebagai (variabel, Credit).
# Credit 2 berarti sebuah kasus berikutnya sekali menyetujui kondisi itu.
NODES = [
    ("nutrient check#1", None, [("crop", 2), ("symptom", 2)]),
    ("planthopper control#1", "nutrient check#1", [("insects", 2)]),
    ("planthopper emergency#1", "planthopper control#1", [("patches", 1)]),
    ("blast control#1", None, [("crop", 1), ("symptom", 1)]),
]


def by_hand():
    ns = {}                                     # NS: node yang berbagi variabel
    for _, _, conds in NODES:
        for var, _ in conds:
            ns[var] = ns.get(var, 0) + 1
    vur, nur = {}, {}
    print("Node                    Variable  Credit  NS  CD    Weight  VUR")
    for node, _, conds in NODES:
        total = 0.0
        for i, (var, credit) in enumerate(conds):
            cd = (i + 1) / len(conds)           # CD = VO / TV
            weight = ns[var] * cd               # Weight = NS x CD
            v = credit * weight                 # VUR = Credit x Weight
            vur[(node, i)] = v
            total += v
            print(f"{node if i == 0 else '':<25}{var:<10}{credit:<8}"
                  f"{ns[var]:<4}{cd:<6.2f}{weight:<8.2f}{v:.2f}")
        nur[node] = total / len(conds)          # NUR = rata-rata VUR
    print()
    for node, _, conds in NODES:
        parts = " + ".join(f"{vur[(node, i)]:.2f}" for i in range(len(conds)))
        print(f"NUR {node} = ({parts}) / {len(conds)} = {nur[node]:.2f}")
    rur = {}
    for node, parent, _ in NODES:
        path, p = [node], parent
        while p is not None:                    # rule: jalur ke bawah
```

Kode 6.2 (Lanjutan) *usage_by_hand.py: perhitungan yang sama dalam Python.*

```
            path.insert(0, p)
            p = [n for n in NODES if n[0] == p][0][1]
        rur[node] = sum(nur[n] for n in path) / len(path)
        parts = " + ".join(f"{nur[n]:.2f}" for n in path)
        print(f"RUR {node} = ({parts}) / {len(path)} = {rur[node]:.2f}")
    return vur, nur, rur


def by_engine():
    kb = KnowledgeBase("rice corner")
    for facts, advice in [
            (["crop = rice", "symptom = yellow leaves"], "nutrient check"),
            (["crop = rice", "symptom = yellow leaves",
              "insects = brown hoppers"], "planthopper control"),
            (["crop = rice", "symptom = yellow leaves",
              "insects = brown hoppers", "patches = drying"],
             "planthopper emergency"),
            (["crop = rice", "symptom = diamond spots on leaves"],
             "blast control")]:
        kb.learn(Case.of(facts, ["advice = " + advice]))
    return kb.usage()


def main():
    vur, nur, rur = by_hand()
    u = by_engine()
    same = [abs(u.vur(n, i) - v) < 1e-9 for (n, i), v in vur.items()]
    same += [abs(u.nur(n) - v) < 1e-9 for n, v in nur.items()]
    same += [abs(u.rur(n) - v) < 1e-9 for n, v in rur.items()]
    print()
    print("The engine agrees on " + str(sum(same)) + " of " + str(len(same))
          + " numbers.")


if __name__ == "__main__":
    main()
```

Keduanya mencetak:

```
Node                    Variable  Credit  NS  CD    Weight  VUR
nutrient check#1        crop      2       2   0.50  1.00    2.00
                        symptom   2       2   1.00  2.00    4.00
planthopper control#1   insects   2       1   1.00  1.00    2.00
planthopper emergency#1 patches   1       1   1.00  1.00    1.00
blast control#1         crop      1       2   0.50  1.00    1.00
                        symptom   1       2   1.00  2.00    2.00

NUR nutrient check#1 = (2.00 + 4.00) / 2 = 3.00
NUR planthopper control#1 = (2.00) / 1 = 2.00
NUR planthopper emergency#1 = (1.00) / 1 = 1.00
NUR blast control#1 = (1.00 + 2.00) / 2 = 1.50
RUR nutrient check#1 = (3.00) / 1 = 3.00
RUR planthopper control#1 = (3.00 + 2.00) / 2 = 2.50
RUR planthopper emergency#1 = (3.00 + 2.00 + 1.00) / 3 = 2.00
RUR blast control#1 = (1.50) / 1 = 1.50

The engine agrees on 14 of 14 numbers.
```

Bacalah tabelnya dari kanan. VUR tertinggi di pojok padi adalah 4,00, untuk `symptom = yellow leaves` di `nutrient check#1`: ia dibagi dua node, berdiri di ujung node-nya, dan sebuah kasus berikutnya menyetujuinya. Yang terendah 1,00, untuk `patches = drying` dan untuk `crop = rice` di `blast control#1`: masing-masing dipakai dengan satu cara saja, dan belum ada yang

melewatinya. NUR memperlihatkan gambaran yang sama satu tingkat di atasnya: `nutrient check#1`, node tempat segala yang lain di pojok padi bergantung, punya NUR tertinggi, 3,00.

RUR menyimpan sedikit kejutan. Rule darurat punya NUR *terendah* di ujungnya (1,00), tetapi RUR-nya, 2,00, lebih tinggi daripada RUR `blast control#1` (1,50), karena ia melewati `nutrient check#1` yang banyak dipakai dalam perjalanannya. Sebuah rule dinilai dari semua pengetahuan yang dilaluinya, bukan hanya dari node terakhirnya. Dan baris terakhir adalah janji bahwa tidak ada sihir di sini: mesin sepakat pada keempat belas angka.

6.4 Langkah 2: Pembangkitan pada Dokter Tani

Sekarang seluruh basis pengetahuan pertama Mbak Nur: tiga belas kasus untuk durian, mangga, cabai, dan padi, disimpan dalam `farm-seed-cases.json` (Subbab 6.5 menampilkannya). **Kode 6.3** membangunnya, mencetak tingkat pemakaian rule-nya serta node dan variabel terpentingnya, mendaftar urutan node relatifnya, meminta kandidat rule dan node kepada mesin, dan memainkan telaah Mbak Nur: satu kandidat rule ditolak, satu kandidat node diterima dengan satu kondisi saja, dan tiga kebun dicoba sebelum dan sesudahnya.

Kode 6.3 `GenerateDemo.java`: pembangkitan rule dan node pada basis pengetahuan tani, dengan telaah penyuluh.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import vcirs.CaseFile;
import vcirs.Condition;
import vcirs.Facts;
import vcirs.GeneratedNode;
import vcirs.GeneratedRule;
import vcirs.InferenceResult;
import vcirs.KnowledgeBase;
import vcirs.Node;
import vcirs.Num;
import vcirs.Report;
import vcirs.Usage;

/**
 * Pembangkitan rule pada dokter tani: tingkat pemakaian, urutan node
 * relatif yang dihasilkannya, kandidat rule dan node yang diusulkan mesin,
 * dan telaah Mbak Nur, yang hanya menyimpan yang benar secara pertanian.
 */
public class GenerateDemo {
    static final List<Map<String, String>> FARMS = List.of(
            Map.of("crop", "mango", "symptom", "dark sunken spots"),
            Map.of("crop", "papaya", "symptom", "dark sunken spots"),
            Map.of("crop", "dragon fruit", "symptom", "dark sunken spots"));

    static void coverage(KnowledgeBase kb, String when) {
        for (Map<String, String> farm : FARMS) {
            InferenceResult result = kb.infer(Facts.map(farm));
            String answer = result.paths.isEmpty() ? "no advice"
                    : result.conclusions().get(0).value + " ("
                    + result.paths.get(0).ruleId() + ")";
            System.out.println("  " + when + ", " + farm.get("crop") + ": "
                    + answer);
        }
    }
```

Kode 6.3 (Lanjutan) *GenerateDemo.java: pembangkitan rule dan node pada basis pengetahuan tani, dengan telaah penyuluh.*

```
    public static void main(String[] args) {
        CaseFile seed = CaseFile.read("../data/farm-seed-cases.json");
        KnowledgeBase kb = seed.build(new KnowledgeBase(seed.name,
                seed.description));
        Usage usage = kb.usage();
        System.out.println("Rule usage rates, lowest first:");
        for (String line : Report.rurTable(kb, 4)) {
            System.out.println("  " + line);
        }
        System.out.println("Most important nodes: "
                + String.join(", ", usage.importantNodes()));
        System.out.println("Most important variables: "
                + String.join(", ", usage.importantVariables()));
        System.out.println("Relative node order:");
        List<String> order = kb.relativeNodeOrder().order;
        for (int i = 0; i < order.size(); i++) {
            System.out.println("  " + String.format("%2d", i + 1) + ". "
                    + order.get(i) + "  (NUR "
                    + Num.fixed(usage.nur(order.get(i)), 4) + ")");
        }

        System.out.println("Candidate rules:");
        for (GeneratedRule g : kb.generateRules()) {
            System.out.println("  " + g.id + ": " + g.nodes.size()
                    + " nodes, from " + g.nodes.get(0) + " to "
                    + g.nodes.get(g.nodes.size() - 1));
            System.out.println("  Mbak Nur keeps none of the other nodes: "
                    + "rejected.");
        }
        System.out.println("Candidate nodes:");
        List<GeneratedNode> candidates = kb.generateNodes();
        for (GeneratedNode g : candidates) {
            System.out.println("  " + g.id + ": " + g.conditions.size()
                    + " conditions, the last "
                    + g.conditions.get(g.conditions.size() - 1));
        }
        GeneratedNode chosen = candidates.get(2);
        System.out.println("Reviewing " + chosen.id + ":");
        List<String> texts = new ArrayList<>();
        for (int i = 0; i < chosen.conditions.size(); i++) {
            Condition c = chosen.conditions.get(i);
            texts.add(c.toString());
            System.out.println("  " + String.format("%2d", i + 1) + ". " + c);
        }
        coverage(kb, "before");
        String keep = "symptom = dark sunken spots";
```

Kode 6.3 (Lanjutan) *GenerateDemo.java: pembangkitan rule dan node pada basis pengetahuan tani, dengan telaah penyuluh.*

```
        int number = texts.indexOf(keep) + 1;
        Node node = kb.acceptNode(chosen, List.of(keep), "Nur");
        System.out.println("Mbak Nur keeps condition " + number + " only. "
                + "New node: " + node);
        coverage(kb, "after");
        System.out.println(kb.summary());
    }
}
```

Panggilan ke mesin hanya sedikit. `kb.usage()` memberi tingkat-tingkat pemakaian; `Report.rurTable(kb, 4)` mencetak RUR dengan empat desimal, dari yang terendah; `usage.importantNodes()` dan `usage.importantVariables()` memberi hasil analisis variabel;

`kb.relativeNodeOrder().order` adalah urutan node relatif. `kb.generateRules()` dan `kb.generateNodes()` mengembalikan kandidat-kandidatnya, dan `kb.acceptNode(candidate, keep, "Nur")` menyimpan node hasil bangkitan hanya dengan kondisi-kondisi dalam `keep`. `Num.fixed` mencetak angka dengan jumlah desimal tetap, dengan cara yang sama di Java dan Python. `coverage` menanyakan kepada basis pengetahuan tentang tiga kebun dengan bercak hitam cekung: mangga, pepaya, dan buah naga. **Kode 6.4** adalah versi Python-nya.

Kode 6.4 `generate_demo.py`: pembangkitan dan telaah yang sama dalam Python.

```
"""Pembangkitan rule pada dokter tani: tingkat pemakaian, urutan node
relatif yang dihasilkannya, kandidat rule dan node yang diusulkan mesin, dan
telaah Mbak Nur, yang hanya menyimpan yang benar secara pertanian."""

import pathlib
import sys

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import KnowledgeBase, casefile, num, report

FARMS = [{"crop": "mango", "symptom": "dark sunken spots"},
         {"crop": "papaya", "symptom": "dark sunken spots"},
         {"crop": "dragon fruit", "symptom": "dark sunken spots"}]


def coverage(kb, when):
    for farm in FARMS:
        result = kb.infer(farm)
        answer = (result.conclusions()[0].value + " (" + result.paths[0]
                  .rule_id() + ")") if result.paths else "no advice"
        print("  " + when + ", " + farm["crop"] + ": " + answer)


def main():
    name, description, steps = casefile.read_cases(
        "../data/farm-seed-cases.json")
    kb = casefile.build(KnowledgeBase(name, description), steps)
    usage = kb.usage()
    print("Rule usage rates, lowest first:")
    for line in report.rur_table(kb, 4):
        print("  " + line)
    print("Most important nodes: " + ", ".join(usage.important_nodes()))
    print("Most important variables: "
          + ", ".join(usage.important_variables()))
    print("Relative node order:")
    for i, node_id in enumerate(kb.relative_node_order().order):
        print("  " + str(i + 1).rjust(2) + ". " + node_id + "  (NUR "
              + num.fixed(usage.nur(node_id), 4) + ")")
```

Kode 6.4 (Lanjutan) *generate_demo.py: pembangkitan dan telaah yang sama dalam Python.*

```
    print("Candidate rules:")
    for g in kb.generate_rules():
        print("  " + g.id + ": " + str(len(g.nodes)) + " nodes, from "
              + g.nodes[0] + " to " + g.nodes[-1])
        print("  Mbak Nur keeps none of the other nodes: rejected.")
    print("Candidate nodes:")
    candidates = kb.generate_nodes()
    for g in candidates:
        print("  " + g.id + ": " + str(len(g.conditions))
              + " conditions, the last " + str(g.conditions[-1]))
    chosen = candidates[2]
    print("Reviewing " + chosen.id + ":")
    for i, c in enumerate(chosen.conditions):
        print("  " + str(i + 1).rjust(2) + ". " + str(c))
    coverage(kb, "before")
    keep = "symptom = dark sunken spots"
    number = [str(c) for c in chosen.conditions].index(keep) + 1
    node = kb.accept_node(chosen, [keep], "Nur")
    print("Mbak Nur keeps condition " + str(number) + " only. New node: "
          + str(node))
    coverage(kb, "after")
    print(kb.summary())

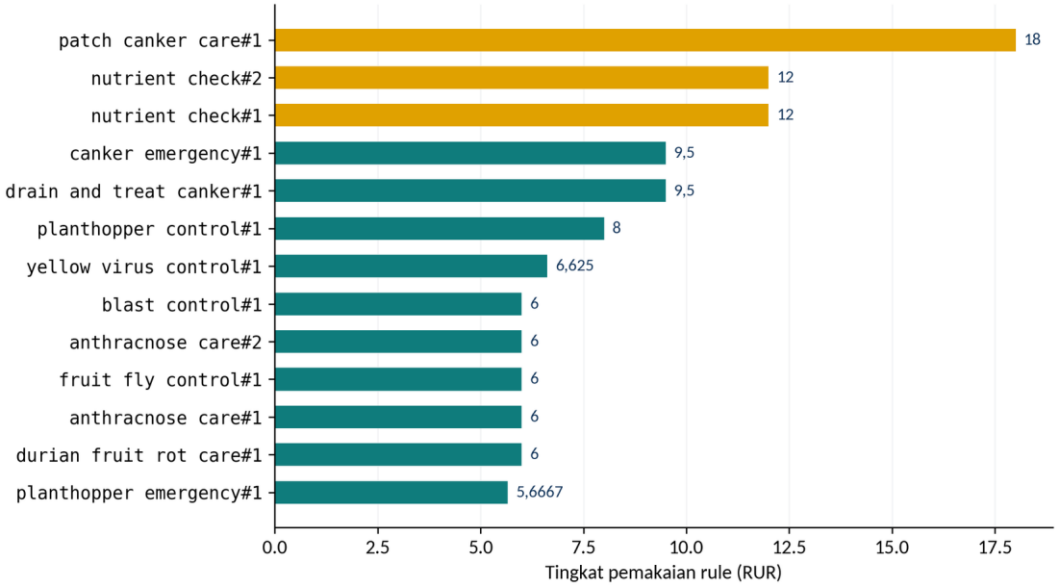

if __name__ == "__main__":
    main()
```

Keduanya mencetak:

```
Rule usage rates, lowest first:
  RuleID                    RUR
  ------------------------  -------
  planthopper emergency#1   5.6667
  durian fruit rot care#1   6.0000
  anthracnose care#1        6.0000
  fruit fly control#1       6.0000
  anthracnose care#2        6.0000
  blast control#1           6.0000
  yellow virus control#1    6.6250
  planthopper control#1     8.0000
  drain and treat canker#1  9.5000
  canker emergency#1        9.5000
  nutrient check#1          12.0000
  nutrient check#2          12.0000
  patch canker care#1       18.0000
Most important nodes: patch canker care#1, nutrient check#2
Most important variables: crop, symptom
Relative node order:
   1. nutrient check#2  (NUR 12.0000)
   2. planthopper control#1  (NUR 4.0000)
   3. planthopper emergency#1  (NUR 1.0000)
   4. blast control#1  (NUR 6.0000)
   5. anthracnose care#2  (NUR 6.0000)
   6. fruit fly control#1  (NUR 6.0000)
   7. anthracnose care#1  (NUR 6.0000)
   8. durian fruit rot care#1  (NUR 6.0000)
   9. nutrient check#1  (NUR 12.0000)
  10. yellow virus control#1  (NUR 1.2500)
  11. patch canker care#1  (NUR 18.0000)
  12. canker emergency#1  (NUR 1.0000)
  13. drain and treat canker#1  (NUR 1.0000)
Candidate rules:
  patch canker care#1G: 11 nodes, from nutrient check#2 to patch canker care#1
  Mbak Nur keeps none of the other nodes: rejected.
Candidate nodes:
  patch canker care#1G: 16 conditions, the last symptom = wet patch on bark
  durian fruit rot care#1G: 15 conditions, the last symptom = rotting fruit on tree
  anthracnose care#1G: 12 conditions, the last symptom = dark sunken spots
  fruit fly control#1G: 13 conditions, the last symptom = stung fruit
  anthracnose care#2G: 12 conditions, the last symptom = dark sunken spots
  nutrient check#1G: 10 conditions, the last symptom = yellow leaves
  nutrient check#2G: 10 conditions, the last symptom = yellow leaves
  blast control#1G: 8 conditions, the last symptom = diamond spots on leaves
Reviewing anthracnose care#1G:
   1. patches = drying
   2. patch around trunk = yes
   3. soil = waterlogged
   4. leaves curl = yes
   5. insects = whitefly
   6. insects = brown hoppers
   7. crop = rice
   8. symptom = diamond spots on leaves
   9. crop = chili
  10. symptom = yellow leaves
  11. crop = mango
  12. symptom = dark sunken spots
  before, mango: anthracnose care (anthracnose care#1)
  before, papaya: no advice
  before, dragon fruit: no advice
Mbak Nur keeps condition 12 only. New node: anthracnose care#1G: symptom = dark sunken
spots -> advice = anthracnose care
  after, mango: anthracnose care (anthracnose care#1)
  after, papaya: anthracnose care (anthracnose care#1G)
  after, dragon fruit: anthracnose care (anthracnose care#1G)
Crop and fruit-tree doctor: 8 variables, 14 nodes, 14 rules
```

Bacalah keluarannya dari atas, seperti Mbak Nur membacanya.

Tingkat pemakaian rule. **Gambar 6.3** menggambar tabel yang sama sebagai diagram. Tiga rule yang paling banyak dipakai adalah `patch canker care#1` (18,00) dan kedua rule `nutrient check` (masing-masing 12,00). Merekalah node tempat sebagian besar pengetahuan lain bergantung: `patch canker care#1` punya dua pengecualian (air menggenang di akar, dan bercak yang melingkari batang), dan `nutrient check#2` padi menuju ke node-node wereng. Kondisi-kondisinya, `crop` dan `symptom`, dipakai bersama kedelapan node tingkat atas dan mendapat Credit tambahan dari pengecualian-pengecualiannya.



Gambar 6.3 Tingkat pemakaian rule basis pengetahuan pertama Mbak Nur. Rule yang paling banyak dipakai, berwarna kuning tua, adalah rule yang node pertamanya punya pengecualian: banyak pengetahuan melewatinya.

Node dan variabel terpenting. `patch canker care#1` dan `nutrient check#2` masing-masing dibagi tiga rule — rule-nya sendiri dan rule pengecualian-pengecualiannya — lebih banyak daripada node lain, sehingga keduanya node terpenting [7]. (`nutrient check#1` punya satu pengecualian, jadi hanya dibagi dua.) Variabel terpentingnya `crop` dan `symptom`, masing-masing dipakai delapan node.

Urutan node relatif. Penelusuran dimulai dari rule yang paling sedikit dipakai, `planthopper emergency#1` (RUR 5,67), dan mendaftar node-node-nya menurut urutannya sendiri: `nutrient check#2`, `planthopper control#1`, `planthopper emergency#1`. Lalu datang lima rule satu-node ber-RUR 6,00, yang terbaru lebih dulu (aturan pemecah seri mesin), lalu rule cabai dengan pengecualiannya, dan terakhir rule kanker durian dengan kedua pengecualiannya. Urutan setiap rule tetap terjaga: `nutrient check#2` sebelum `planthopper control#1`, dan `patch canker care#1` sebelum kedua pengecualiannya.

Kandidat rule. Hanya ada satu. Untuk `nutrient check#2`, bagian urutan sebelum node itu kosong, sehingga kandidatnya adalah rule satu-node yang sudah ada, dan mesin tidak mengusulkannya. Untuk `patch canker care#1`, kandidatnya adalah sebelas node dari `nutrient check#2` sampai `patch canker care#1`: saran padi, cabai, dan mangga, lalu kanker durian. Mbak Nur tidak menyimpan satu pun dari sepuluh node lainnya — semuanya tak ada hubungannya dengan kanker di kulit batang durian — dan ia menolaknya. Inilah nasib biasa sebagian besar kandidat rule dalam basis pengetahuan seperti ini, yang setiap node-nya berakhir dengan sebuah saran. Pembangkitan rule bersinar ketika sebagian node *menyimpulkan fakta yang dipakai node*

lain, seperti Zookeeper dalam tesis, tempat satu node menyimpulkan `mammal = yes` dan node lain diawali dengannya; Soal 6.8 memperlihatkannya bekerja di sana.

Kandidat node. Ada delapan, satu untuk setiap node yang memakai `symptom`. Mengapa `symptom` dan bukan `crop`? Keduanya terpenting, tetapi di setiap node yang memakainya `crop` berdiri lebih dulu dan `symptom` sesudahnya, lebih dekat ke konklusi; tesis membuang variabel terpenting yang selalu *tertutupi* oleh variabel lain dengan cara ini, dan mesin pun begitu [7]. Setiap kandidat adalah urutan variabel relatif sampai gejala node itu sendiri, dengan konklusi node itu. Yang ketiga, `anthracnose care#1G`, berasal dari node antraknosa mangga dan berakhir dengan `symptom = dark sunken spots`.

Telaahnya. Mbak Nur membaca kedua belas kondisi `anthracnose care#1G`. Sebagian besar tidak masuk akal untuk konklusi ini: petak sawah yang mengering, kutu kebul, tanah tergenang, tanaman yang bukan mangga. Tetapi yang kedua belas persis yang ia sadari di pasar: bercak hitam cekung, pada buah apa pun, adalah antraknosa. Ia menyimpan kondisi 12 dan tidak ada yang lain. Sebelumnya, kebun mangga mendapat saran antraknosa dari `anthracnose care#1`, sedangkan kebun pepaya dan buah naga tidak mendapat apa-apa. Sesudahnya, kebun mangga tetap mendapat jawaban yang sama dari node yang sama — tidak ada yang diketahuinya yang terganggu — dan kebun pepaya dan buah naga mendapat saran antraknosa dari node baru, `anthracnose care#1G`. Basis pengetahuan punya empat belas node dan menjawab lebih banyak kebun daripada sebelumnya.

Ia tidak menerima kandidat-kandidat lain. `nutrient check#1G` menawarkan `symptom = yellow leaves` saja, yang akan menyuruh *setiap* petani dengan daun kuning memeriksa nitrogen; padahal daun kuning pada durian yang tergenang air berarti akar yang membusuk, bukan kelaparan. Tahu di mana sebuah generalisasi berhenti adalah separuh pembangkitan rule yang menjadi bagian pakar.

6.5 Dokter Tanaman dan Pohon Buah

6.5.1 Titik awal dokter

Dokter ini mulai dari tiga belas kasus awal dalam `farm-seed-cases.json`:

```
{
  "name": "Crop and fruit-tree doctor",
  "description": "Problems of durian, mango, chili and rice, taught by Mbak Nur, the
village extension worker (penyuluh).",
  "cases": [
    {"facts": ["crop = durian", "symptom = wet patch on bark"], "conclusions": ["advice =
patch canker care"], "option": -1, "author": "Nur"},
    {"facts": ["crop = durian", "symptom = wet patch on bark", "soil = waterlogged"],
"conclusions": ["advice = drain and treat canker"], "option": -1, "author": "Nur"},
    {"facts": ["crop = durian", "symptom = wet patch on bark", "patch around trunk =
yes"], "conclusions": ["advice = canker emergency"], "option": -1, "author": "Nur"},
    {"facts": ["crop = durian", "symptom = rotting fruit on tree"], "conclusions":
["advice = durian fruit rot care"], "option": -1, "author": "Nur"},
    {"facts": ["crop = mango", "symptom = dark sunken spots"], "conclusions": ["advice =
anthracnose care"], "option": -1, "author": "Nur"},
    {"facts": ["crop = mango", "symptom = stung fruit"], "conclusions": ["advice = fruit
fly control"], "option": -1, "author": "Nur"},
    {"facts": ["crop = chili", "symptom = dark sunken spots"], "conclusions": ["advice =
anthracnose care"], "option": -1, "author": "Nur"},
    {"facts": ["crop = chili", "symptom = yellow leaves"], "conclusions": ["advice =
nutrient check"], "option": -1, "author": "Nur"},
    {"facts": ["crop = chili", "symptom = yellow leaves", "leaves curl = yes", "insects =
whitefly"], "conclusions": ["advice = yellow virus control"], "option": -1, "author":
"Nur"},
    {"facts": ["crop = rice", "symptom = yellow leaves"], "conclusions": ["advice =
nutrient check"], "option": -1, "author": "Nur"},
    {"facts": ["crop = rice", "symptom = yellow leaves", "insects = brown hoppers"],
"conclusions": ["advice = planthopper control"], "option": -1, "author": "Nur"},
    {"facts": ["crop = rice", "symptom = yellow leaves", "insects = brown hoppers",
"patches = drying"], "conclusions": ["advice = planthopper emergency"], "option": -1,
"author": "Nur"},
    {"facts": ["crop = rice", "symptom = diamond spots on leaves"], "conclusions":
["advice = blast control"], "option": -1, "author": "Nur"}
  ]
}
```

Setiap kasus adalah sebuah kebun yang dikunjungi Mbak Nur, dengan tanamannya, gejala yang ia lihat, dan apa pun yang menentukan sarannya. `"option": -1` membiarkan `learn` menempatkan tiap kasus, sehingga pengecualian-pengecualiannya — tanah tergenang di bawah kanker, bercak yang melingkari batang, kutu kebul pada daun cabai yang keriting, wereng cokelat lalu petak yang mengering di sawah — tergantung di bawah node yang dihaluskannya, seperti di Bab 3 dan 4.

Seperti di Bab 5, konklusinya berupa label pendek, dan kata-kata yang dibaca petani disimpan di `advice-texts.txt`:

```
# The wording of each piece of advice, kept apart from the knowledge.
# One line per advice: label = text. Lines starting with # are notes.
patch canker care = Scrape the soft, rotten bark down to healthy wood, burn the
scrapings, and paint the wound with the fungicide your penyuluh recommends. Keep water
from standing around the trunk.
drain and treat canker = Water standing around the roots feeds this rot. Dig drains so
the soil dries after rain, then treat the patch as for canker.
canker emergency = A patch that goes most of the way around the trunk can kill the tree.
Call the penyuluh this week, and do not wait for the harvest.
durian fruit rot care = Pick up and bury rotten fruit, prune for air and light, and keep
fruit off wet soil. Ask the penyuluh about protecting the tree before the rainy season.
root rot care = Yellow, wilting leaves with water standing at the roots mean the roots
are rotting. Dig drains, keep mulch away from the trunk, and ask the penyuluh about
treatment.
anthracnose care = Dark sunken spots are anthracnose, a fungus that attacks many fruits.
Remove and burn spotted fruit and leaves, prune for air, and never keep seed from spotted
fruit. Ask the penyuluh about hot-water dipping after harvest.
fruit fly control = Collect every fallen and stung fruit and bury it at least 30 cm deep.
Hang methyl eugenol traps, and wrap young fruit in paper bags.
nutrient check = Yellow leaves often mean the plant is short of nitrogen. Ask the
penyuluh about a soil test before adding fertilizer, and give it in smaller doses.
yellow virus control = Curled yellow leaves with whiteflies are the yellow virus. Pull
out and destroy sick plants, hang yellow sticky traps, and plant rows of maize around the
field.
planthopper control = Brown planthoppers at the base of the plants: drain the field for a
few days, do not add more nitrogen, and do not spray insecticides that also kill the
spiders and beetles that eat them.
planthopper emergency = Patches drying out mean a heavy planthopper attack. Call the
penyuluh today, warn your neighbours, and plant at the same time as them next season.
blast control = Diamond-shaped spots with gray centers are blast. Split the nitrogen
fertilizer, keep the field flooded if you can, and choose a resistant variety next
season.
```

Saran-sarannya mengikuti agronomi terkini. Untuk kanker durian: kerok kulit batang yang busuk dan obati lukanya, dan terutama jangan biarkan air menggenang di akar, karena penyakit itu membutuhkannya [2]. Untuk lalat buah: sanitasi kebun dan perangkap pemusnahan jantan dari program pengendalian kawasan yang diterbitkan ACIAR pada 2024, yakni setiap buah yang jatuh dikumpulkan dan dikubur paling sedikit 30 sentimeter, dan perangkap metil eugenol di kebun [4]. Untuk virus kuning cabai: saran Kementerian Pertanian, yaitu cabut dan musnahkan tanaman sakit, perangkap kuning berperekat, dan barisan jagung di sekeliling lahan sebagai penghalang [11]. Untuk wereng: keringkan sawah, jangan menambah nitrogen, dan hindari insektisida berspektrum luas yang memicu ledakan dengan membunuh musuh alami wereng [6]; untuk blas: pupuk nitrogen yang dibagi, penggenangan bila mungkin, dan varietas tahan musim berikutnya [12]; untuk antraknosa: buang buah yang terinfeksi dan, untuk mangga, tanyakan perlakuan air panas sesudah panen, yang menurunkan penyakit itu dalam uji coba terbaru [3]. Tidak ada saran yang menyebut merek atau dosis pestisida; bila bahan kimia dibutuhkan, dokter mengarahkan petani ke penyuluh, yang tahu apa yang terdaftar dan manjur di daerahnya. (Kata-kata saran di sini dalam bahasa Inggris agar kodenya sama untuk kedua edisi; di lapangan, berkas inilah yang ditulis dalam bahasa petani.)

6.5.2 Programnya

Kode 6.5 adalah dokter itu dalam Java.

Kode 6.5 `FarmDoctor.java`: dokter tanaman dan pohon buah di atas vcirs-core, dengan tingkat pemakaian dan pembangkitan.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

import vcirs.Case;
import vcirs.CaseFile;
import vcirs.Conclusion;
import vcirs.Condition;
import vcirs.Fact;
import vcirs.Facts;
import vcirs.GeneratedNode;
import vcirs.InferenceResult;
import vcirs.KnowledgeBase;
import vcirs.LearnResult;
import vcirs.Node;
import vcirs.Question;
import vcirs.RdrPath;
import vcirs.Report;
import vcirs.Usage;
import vcirs.Variable;
import vcirs.VcirsException;

/**
 * Dokter tanaman dan pohon buah: saran untuk durian, mangga, cabai, dan
 * padi, dengan tingkat pemakaian dan pembangkitan rule VCIRS.
 * java FarmDoctor [--echo] [--tree] [--usage] [--generate] [kb.json]
 * Petani menjawab beberapa pertanyaan dan mendapat saran beserta
 * alasannya; bila sarannya keliru atau tidak ada, penyuluh mengajarkan yang
 * benar. --usage mencetak tingkat pemakaian; --generate membuat penyuluh
 * menelaah node usulan mesin dan hanya menyimpan yang benar di lapangan.
 */
public class FarmDoctor {
    static final String[][] QUESTIONS = {
        {"crop", "Which plant has the problem?", "durian", "mango", "chili",
            "rice"},
        {"symptom", "What do you see?", "wet patch on bark",
            "rotting fruit on tree", "dark sunken spots", "stung fruit",
            "yellow leaves", "diamond spots on leaves"},
        {"soil", "How is the soil after rain?", "waterlogged", "drains well"},
```

Kode 6.5 (Lanjutan) *FarmDoctor.java: dokter tanaman dan pohon buah di atas vcirs-core, dengan tingkat pemakaian dan pembangkitan.*

```
        {"patch around trunk", "Does the patch go most of the way around the "
            + "trunk?", "yes", "no"},
        {"leaves curl", "Are the leaves curling?", "yes", "no"},
        {"insects", "Which small insects do you see?", "whitefly",
            "brown hoppers", "none"},
        {"patches", "Are whole patches of the field drying out?", "drying",
            "no"},
    };

    static BufferedReader in = new BufferedReader(
            new InputStreamReader(System.in, StandardCharsets.UTF_8));
    static boolean echo = false;
    static KnowledgeBase kb;

    /** Pecah teks menjadi baris paling panjang width karakter, di spasi. */
    static List<String> wrap(String text, int width) {
        List<String> lines = new ArrayList<>();
        String line = "";
        for (String word : text.split(" ", -1)) {
            if (!line.isEmpty() && line.length() + 1 + word.length() > width) {
                lines.add(line);
                line = word;
            } else {
                line = line.isEmpty() ? word : line + " " + word;
            }
        }
        lines.add(line);
        return lines;
    }

    /** Membaca jawaban pengguna; bila echo aktif, mencetaknya juga. */
    static String read(String prompt) {
        List<String> lines = wrap(prompt.replaceAll("[ >]+$", ""), 74);
        for (int i = 0; i < lines.size() - 1; i++) {
            System.out.println(lines.get(i));
        }
        System.out.print(lines.get(lines.size() - 1) + " > ");
        String line;
        try {
            line = in.readLine();
        } catch (IOException e) {
            line = null;
        }
        String text = line == null ? "" : line.strip();
        if (echo) {
            System.out.println(text);
```

Kode 6.5 (Lanjutan) *FarmDoctor.java: dokter tanaman dan pohon buah di atas vcirs-core, dengan tingkat pemakaian dan pembangkitan.*

```
        }
        return text;
    }

    static Fact ask(Question q) {
        Variable var = kb.variables.get(q.variable);
        String text = q.variable;
        List<String> choices = q.choices;
        if (var != null && !var.question.isEmpty()) {
            text = var.question;
            choices = var.choices;
        }
        if (!choices.isEmpty()) {
            text += " (" + String.join("/", choices) + ")";
        }
        String answer = read(text + " > ");
        return answer.isEmpty() ? null : new Fact(q.variable, answer); // Enter
    }

    /** advice-texts.txt: satu "label = teks" per baris. */
    static Map<String, String> readTexts(Path path) {
        Map<String, String> texts = new LinkedHashMap<>();
        try {
            for (String line : Files.readAllLines(path, StandardCharsets.UTF_8)) {
                int at = line.indexOf('=');
                if (at >= 0 && !line.startsWith("#")) {
                    texts.put(line.substring(0, at).strip(),
                              line.substring(at + 1).strip());
                }
            }
        } catch (IOException e) {
            throw new VcirsException("cannot read " + path);
        }
        return texts;
    }
```

Kode 6.5 (Lanjutan) *FarmDoctor.java: dokter tanaman dan pohon buah di atas vcirs-core, dengan tingkat pemakaian dan pembangkitan.*

```
    /** Muat basis pengetahuan, atau bangun dari kasus-kasus awal. */
    static KnowledgeBase openKb(String path) {
        if (Files.exists(Path.of(path))) {
            return KnowledgeBase.load(path);
        }
        Path parent = Path.of(path).toAbsolutePath().getParent();
        CaseFile seed = CaseFile.read(parent.resolve("farm-seed-cases.json")
                .toString());
        KnowledgeBase k = new KnowledgeBase(seed.name, seed.description);
        for (String[] q : QUESTIONS) {
            List<String> choices = new ArrayList<>();
            for (int i = 2; i < q.length; i++) {
                choices.add(q[i]);
            }
            k.declare(q[0], q[1], choices);
        }
        seed.build(k);
        k.save(path);
        return k;
    }

    static void printUsage() {
        System.out.println("Node usage rates (NUR), lowest first:");
        for (String line : Report.nurTable(kb, 4)) {
            System.out.println("  " + line);
        }
        System.out.println("Rule usage rates (RUR), lowest first:");
        for (String line : Report.rurTable(kb, 4)) {
            System.out.println("  " + line);
        }
        Usage usage = kb.usage();
        System.out.println("Most important nodes: "
                + String.join(", ", usage.importantNodes()));
        System.out.println("Most important variables: "
                + String.join(", ", usage.importantVariables()));
    }
```

Kode 6.5 (Lanjutan) *FarmDoctor.java: dokter tanaman dan pohon buah di atas vcirs-core, dengan tingkat pemakaian dan pembangkitan.*

```
    /** Penyuluh menelaah node yang diusulkan mesin. */
    static void review(String path) {
        List<GeneratedNode> candidates = kb.generateNodes();
        for (int i = 0; i < candidates.size(); i++) {
            GeneratedNode g = candidates.get(i);
            System.out.println(String.format("%2d", i + 1) + ". " + g.id + ": "
                    + g.conditions.size() + " conditions, from " + g.source);
        }
        while (true) {
            String pick = read("Review which candidate? (number, Enter to "
                    + "finish) > ");
            if (pick.isEmpty()) {
                break;
            }
            GeneratedNode g = candidates.get(Integer.parseInt(pick) - 1);
            for (int i = 0; i < g.conditions.size(); i++) {
                System.out.println("   " + String.format("%2d", i + 1) + ". "
                        + g.conditions.get(i));
            }
            List<String> concl = new ArrayList<>();
            for (Conclusion c : g.conclusions) {
                concl.add(c.toString());
            }
            System.out.println("   -> " + String.join(", ", concl));
            String keep = read("Keep which conditions? (numbers, Enter to "
                    + "reject) > ");
            if (keep.isEmpty()) {
                System.out.println("Rejected.");
                continue;
            }
            List<String> chosen = new ArrayList<>();
            for (String k : keep.split(" ")) {
                chosen.add(g.conditions.get(Integer.parseInt(k) - 1).key());
            }
            Node node = kb.acceptNode(g, chosen, "penyuluh");
            System.out.println("Accepted: " + node);
        }
        kb.save(path);
        System.out.println("Saved " + path + ": " + kb.summary());
    }
```

Kode 6.5 (Lanjutan) *FarmDoctor.java: dokter tanaman dan pohon buah di atas vcirs-core, dengan tingkat pemakaian dan pembangkitan.*

```
    static List<Condition> factsOf(InferenceResult result) {
        List<Condition> facts = new ArrayList<>();
        for (Fact f : result.memory.facts.values()) {
            if (!f.variable.equals("advice")) {
                facts.add(new Condition(f.variable, "=", f.value));
            }
        }
        return facts;
    }

    /** Penyuluh memberi saran yang benar; learn() menaruhnya di tempatnya. */
    static void teach(InferenceResult result, String path) {
        String advice = read("Which advice is right? (label, Enter if none) > ");
        if (advice.isEmpty()) {
            System.out.println("Then there is nothing to learn this time.");
            return;
        }
        List<Condition> facts = factsOf(result);
        while (true) {
            String extra = read("Another fact that matters? "
                    + "(variable = value, Enter when done) > ");
            if (extra.isEmpty()) {
                break;
            }
            try {
                facts.add(Condition.parse(extra));
            } catch (VcirsException e) {
                System.out.println("Please write it as: variable = value");
            }
        }
        LearnResult learned;
        try {
            learned = kb.learn(new Case(facts,
                    List.of(new Conclusion("advice", advice)), 100), "penyuluh");
        } catch (VcirsException e) {
            System.out.println("Cannot learn: " + e.getMessage());
            return;
        }
        System.out.println("Learned: " + learned);
        kb.save(path);
        System.out.println("Saved " + path + ": " + kb.summary());
    }
```

Kode 6.5 (Lanjutan) *FarmDoctor.java: dokter tanaman dan pohon buah di atas vcirs-core, dengan tingkat pemakaian dan pembangkitan.*

```
    public static void main(String[] args) {
        String path = "../data/farm-kb.json";
        List<String> flags = new ArrayList<>();
        for (String a : args) {
            if (a.startsWith("--")) {
                flags.add(a);
            } else {
                path = a;
            }
        }
        echo = flags.contains("--echo");
        kb = openKb(path);
        if (flags.contains("--tree")) {
            for (String line : Report.tree(kb)) {
                System.out.println(line);
            }
            return;
        }
        if (flags.contains("--usage")) {
            printUsage();
            return;
        }
        if (flags.contains("--generate")) {
            review(path);
            return;
        }
        Map<String, String> texts = readTexts(Path.of(path).toAbsolutePath()
                .getParent().resolve("advice-texts.txt"));
        System.out.println("Crop and fruit-tree doctor: " + kb.nodes.size()
                + " pieces of advice so far.");
        System.out.println("Answer each question; press Enter if you do not "
                + "know.");
        InferenceResult result = kb.infer(Facts.none(), FarmDoctor::ask);
        if (!result.paths.isEmpty()) {
            RdrPath p = result.paths.get(0);
            String advice = p.conclusions().get(0).value;
            System.out.println("Advice (" + advice + "):");
            for (String line : wrap(texts.getOrDefault(advice, advice), 72)) {
                System.out.println("  " + line);
            }
            List<String> used = new ArrayList<>();
            for (Node n : p.nodes) {
                for (Condition c : n.conditions) {
                    used.add(c.toString());
                }
            }
```

Kode 6.5 (Lanjutan) *FarmDoctor.java: dokter tanaman dan pohon buah di atas vcirs-core, dengan tingkat pemakaian dan pembangkitan.*

```
            List<String> because = wrap("Because: " + String.join(", ", used)
                    + " (rule " + p.ruleId() + ")", 74);
            for (int i = 0; i < because.size(); i++) {
                System.out.println((i == 0 ? "" : "  ") + because.get(i));
            }
        } else {
            System.out.println("I have no advice for this yet. Ask the "
                    + "penyuluh.");
        }
        if (!result.paths.isEmpty()
                && read("Is the advice right? (yes/no) > ").equals("yes")) {
            kb.learn(new Case(factsOf(result), result.conclusions(), 100),
                    "farmer");
            kb.save(path);
            System.out.println("Good. The case is recorded as checked.");
            return;
        }
        teach(result, path);
    }
}
```

Sebagian besar isinya akan tampak akrab dari Bab 4 dan 5: `QUESTIONS` mendeklarasikan tujuh variabel dengan pertanyaan dan jawaban lazimnya; `wrap`, `read`, dan `ask` menanyakannya dengan sopan; `readTexts` dan `openKb` membaca kata-kata saran dan basis pengetahuan. Yang baru ada di tiga tempat.

`--usage` memanggil `printUsage`, yang mencetak tabel NUR dan RUR serta node dan variabel terpenting — angka-angka Subbab 6.4 untuk basis pengetahuan dalam keadaannya *sekarang*, setelah semua yang telah dipelajari dokter itu.

`--generate` memanggil `review`, layar penyuluh. Ia mendaftar setiap kandidat node yang diusulkan mesin, dengan jumlah kondisinya dan node asalnya. Penyuluh memilih satu untuk ditelaah; layar mencetak kondisi-kondisinya, bernomor, dan konklusinya; ia mengetik nomor kondisi yang disimpannya, atau menekan Enter untuk menolak. `kb.acceptNode` menyimpan kandidat yang diterima sebagai node tingkat atas baru, bertanda hasil bangkitan, dengan rule satu-node-nya sendiri, dan mencatatnya dalam riwayat.

Berkonsultasi dan mengajar sama seperti Bab 4: petani menjawab pertanyaan yang dibutuhkan node-node, dan mendapat saran, dibungkus untuk layar kecil, serta kondisi-kondisi di baliknya dengan nama rule-nya. Bila petani (atau penyuluh di sampingnya) menyatakan sarannya benar, kasusnya dicatat sebagai sudah diperiksa; bila keliru atau tidak ada, `teach` menanyakan saran yang benar dan fakta lain yang penting, lalu membiarkan `learn` menaruh pengetahuan baru itu di tempatnya.

Kode 6.6 adalah dokter yang sama dalam Python.

Kode 6.6 `farm_doctor.py`: dokter tanaman dan pohon buah dalam Python.

```
"""Dokter tanaman dan pohon buah: saran untuk durian, mangga, cabai, dan
padi, dengan tingkat pemakaian dan pembangkitan rule VCIRS.

    python farm_doctor.py [--echo] [--tree] [--usage] [--generate] [kb.json]

Petani menjawab beberapa pertanyaan dan mendapat saran beserta alasannya;
bila sarannya keliru atau tidak ada, penyuluh mengajarkan yang benar.
--usage mencetak tingkat pemakaian; --generate membuat penyuluh menelaah
node yang diusulkan mesin dan hanya menyimpan yang benar di lapangan.
"""

import os
import pathlib
import sys

# Beri tahu Python letak mesin vcirs-core (naik dua folder, lalu masuk).
sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import (Case, Condition, Conclusion, KnowledgeBase, VcirsError,
                   casefile, report)

QUESTIONS = [
    ("crop", "Which plant has the problem?", ["durian", "mango", "chili",
                                              "rice"]),
    ("symptom", "What do you see?", ["wet patch on bark",
                                     "rotting fruit on tree",
                                     "dark sunken spots", "stung fruit",
                                     "yellow leaves",
                                     "diamond spots on leaves"]),
    ("soil", "How is the soil after rain?", ["waterlogged", "drains well"]),
    ("patch around trunk", "Does the patch go most of the way around the "
     "trunk?", ["yes", "no"]),
    ("leaves curl", "Are the leaves curling?", ["yes", "no"]),
    ("insects", "Which small insects do you see?", ["whitefly",
                                                    "brown hoppers", "none"]),
    ("patches", "Are whole patches of the field drying out?",
     ["drying", "no"]),
]
```

Kode 6.6 (Lanjutan) *farm_doctor.py: dokter tanaman dan pohon buah dalam Python.*

```
def wrap(text, width):
    """Pecah teks menjadi baris paling panjang width karakter, di spasi."""
    lines, line = [], ""
    for word in text.split(" "):
        if line and len(line) + 1 + len(word) > width:
            lines.append(line)
            line = word
        else:
            line = word if not line else line + " " + word
    return lines + [line]


class Console:
    """Membaca jawaban pengguna; bila echo aktif, mencetaknya juga."""

    def __init__(self, kb, echo):
        self.kb = kb
        self.echo = echo

    def read(self, prompt):
        lines = wrap(prompt.rstrip(" >"), 74)
        for line in lines[:-1]:
            print(line)
        print(lines[-1] + " > ", end="")
        text = sys.stdin.readline().strip()
        if self.echo:
            print(text)
        return text

    def ask(self, question):
        var = self.kb.variables.get(question.variable)
        text, choices = question.variable, question.choices
        if var is not None and var.question:
            text, choices = var.question, var.choices
        if choices:
            text += " (" + "/".join(choices) + ")"
        return self.read(text + " > ") or None    # Enter = "saya tidak tahu"
```

Kode 6.6 (Lanjutan) *farm_doctor.py: dokter tanaman dan pohon buah dalam Python.*

```
def read_texts(path):
    """advice-texts.txt: satu 'label = teks' per baris."""
    texts = {}
    with open(path, encoding="utf-8") as f:
        for line in f:
            if "=" in line and not line.startswith("#"):
                label, text = line.split("=", 1)
                texts[label.strip()] = text.strip()
    return texts


def open_kb(path):
    """Muat basis pengetahuan, atau bangun dari kasus-kasus awal."""
    if os.path.exists(path):
        return KnowledgeBase.load(path)
    seed = os.path.join(os.path.dirname(path), "farm-seed-cases.json")
    name, description, steps = casefile.read_cases(seed)
    kb = KnowledgeBase(name, description)
    for var, question, choices in QUESTIONS:
        kb.declare(var, question, choices)
    casefile.build(kb, steps)
    kb.save(path)
    return kb


def print_usage(kb):
    print("Node usage rates (NUR), lowest first:")
    for line in report.nur_table(kb, 4):
        print("  " + line)
    print("Rule usage rates (RUR), lowest first:")
    for line in report.rur_table(kb, 4):
        print("  " + line)
    usage = kb.usage()
    print("Most important nodes: " + ", ".join(usage.important_nodes()))
    print("Most important variables: "
          + ", ".join(usage.important_variables()))
```

Kode 6.6 (Lanjutan) *farm_doctor.py: dokter tanaman dan pohon buah dalam Python.*

```
def review(kb, con, path):
    """Penyuluh menelaah node yang diusulkan mesin."""
    candidates = kb.generate_nodes()
    for i, g in enumerate(candidates):
        print(str(i + 1).rjust(2) + ". " + g.id + ": "
              + str(len(g.conditions)) + " conditions, from " + g.source)
    while True:
        pick = con.read("Review which candidate? (number, Enter to finish) > ")
        if not pick:
            break
        g = candidates[int(pick) - 1]
        for i, c in enumerate(g.conditions):
            print("   " + str(i + 1).rjust(2) + ". " + str(c))
        print("   -> " + ", ".join(str(c) for c in g.conclusions))
        keep = con.read("Keep which conditions? (numbers, Enter to reject) > ")
        if not keep:
            print("Rejected.")
            continue
        chosen = [g.conditions[int(k) - 1] for k in keep.split()]
        node = kb.accept_node(g, chosen, "penyuluh")
        print("Accepted: " + str(node))
    kb.save(path)
    print("Saved " + path + ": " + kb.summary())


def teach(kb, con, result, path):
    """Penyuluh memberi saran yang benar; learn() menaruhnya di tempatnya."""
    advice = con.read("Which advice is right? (label, Enter if none) > ")
    if not advice:
        print("Then there is nothing to learn this time.")
        return
    facts = [Condition(f.variable, "=", f.value)
             for f in result.memory.facts.values() if f.variable != "advice"]
    while True:
        extra = con.read("Another fact that matters? "
                         "(variable = value, Enter when done) > ")
        if not extra:
            break
        try:
            facts.append(Condition.parse(extra))
        except VcirsError:
            print("Please write it as: variable = value")
    try:
        learned = kb.learn(Case(facts, [Conclusion("advice", advice)]),
                           "penyuluh")
    except VcirsError as e:
```

Kode 6.6 (Lanjutan) *farm_doctor.py: dokter tanaman dan pohon buah dalam Python.*

```
        print("Cannot learn: " + str(e))
        return
    print("Learned: " + str(learned))
    kb.save(path)
    print("Saved " + path + ": " + kb.summary())


def main(args):
    echo = "--echo" in args
    files = [a for a in args if not a.startswith("--")]
    path = files[0] if files else "../data/farm-kb.json"
    kb = open_kb(path)
    con = Console(kb, echo)
    if "--tree" in args:
        for line in report.tree(kb):
            print(line)
        return
    if "--usage" in args:
        print_usage(kb)
        return
    if "--generate" in args:
        review(kb, con, path)
        return
    texts = read_texts(os.path.join(os.path.dirname(path),
                                    "advice-texts.txt"))
    print("Crop and fruit-tree doctor: " + str(len(kb.nodes))
          + " pieces of advice so far.")
    print("Answer each question; press Enter if you do not know.")
    result = kb.infer({}, con.ask)
    if result.paths:
        p = result.paths[0]
        advice = p.conclusions()[0].value
        print("Advice (" + advice + "):")
        for line in wrap(texts.get(advice, advice), 72):
            print("  " + line)
        because = wrap("Because: " + ", ".join(
            str(c) for n in p.nodes for c in n.conditions)
            + " (rule " + p.rule_id() + ")", 74)
        for i, line in enumerate(because):
            print(("" if i == 0 else "  ") + line)
    else:
        print("I have no advice for this yet. Ask the penyuluh.")
    if result.paths and con.read("Is the advice right? (yes/no) > ") == "yes":
        facts = [Condition(f.variable, "=", f.value)
                 for f in result.memory.facts.values()
                 if f.variable != "advice"]
```

Kode 6.6 (Lanjutan) *farm_doctor.py: dokter tanaman dan pohon buah dalam Python.*

```
        kb.learn(Case(facts, list(result.conclusions())), "farmer")
        kb.save(path)
        print("Good. The case is recorded as checked.")
        return
    teach(kb, con, result, path)


if __name__ == "__main__":
    main(sys.argv[1:])
```

6.6 Lima Kunjungan

Kompilasi dan jalankan dari `Chapter06/java`:

```
javac -d out -sourcepath ../../vcirs-core/java/src *.java
java -cp out FarmDoctor --echo < ../data/session-1.txt
```

atau, dari `Chapter06/python`, `python farm_doctor.py --echo < ../data/session-1.txt`; sesi 4 dijalankan dengan `--generate`. Jalankan kelima sesi berurutan pada basis pengetahuan yang baru, dan transkrip serta `farm-kb.json` sama di kedua bahasa, bita demi bita.

Sesi 1: mangga Pak Slamet.

```
Crop and fruit-tree doctor: 13 pieces of advice so far.
Answer each question; press Enter if you do not know.
Which plant has the problem? (durian/mango/chili/rice) > mango
What do you see? (wet patch on bark/rotting fruit on tree/dark sunken
spots/stung fruit/yellow leaves/diamond spots on leaves) > dark sunken spots
Advice (anthracnose care):
  Dark sunken spots are anthracnose, a fungus that attacks many fruits.
  Remove and burn spotted fruit and leaves, prune for air, and never keep
  seed from spotted fruit. Ask the penyuluh about hot-water dipping after
  harvest.
Because: crop = mango, symptom = dark sunken spots (rule anthracnose
  care#1)
Is the advice right? (yes/no) > yes
Good. The case is recorded as checked.
```

Dua pertanyaan. Setiap node tingkat atas diawali `crop`, sehingga pertanyaan pertama selalu jenis tanaman; node-node durian langsung gagal, dan node mangga pertama membutuhkan gejalanya. Sarannya menyebut masalahnya, mengatakan apa yang harus dilakukan, dan menunjuk penyuluh untuk satu langkah yang membutuhkannya. Baris *Because* (karena) adalah penjelasannya: kedua kondisi dan rule `anthracnose care#1`. Pak Slamet menyatakan sarannya benar, dan pemeriksaannya dicatat.

Sesi 2: sawah Bu Lastri.

```
Crop and fruit-tree doctor: 13 pieces of advice so far.
Answer each question; press Enter if you do not know.
Which plant has the problem? (durian/mango/chili/rice) > rice
What do you see? (wet patch on bark/rotting fruit on tree/dark sunken
spots/stung fruit/yellow leaves/diamond spots on leaves) > yellow leaves
Which small insects do you see? (whitefly/brown hoppers/none) > brown hoppers
Are whole patches of the field drying out? (drying/no) > drying
Advice (planthopper emergency):
  Patches drying out mean a heavy planthopper attack. Call the penyuluh
  today, warn your neighbours, and plant at the same time as them next
  season.
Because: crop = rice, symptom = yellow leaves, insects = brown hoppers,
  patches = drying (rule planthopper emergency#1)
Is the advice right? (yes/no) > yes
Good. The case is recorded as checked.
```

Empat pertanyaan, menuruni rule tiga node: daun kuning pada padi, lalu wereng cokelat di pangkal, lalu petak-petak yang mengering. Jawabannya darurat wereng: hubungi penyuluh hari ini, peringatkan tetangga. Saran untuk tidak menyemprot insektisida berspektrum luas, yang milik node di atasnya, adalah yang paling perlu didengar Bu Lastri, karena reaksi biasa terhadap wereng adalah menyemprot, dan menyemprot justru memberi makan ledakannya [6]; versi dokter berikutnya dapat menampilkan saran node-node di jalan, seperti pemeriksaan mandiri di Bab 5.

Sesi 3: durian berdaun kuning, dan dokter belajar. Pohon durian termuda Pak Slamet, yang ditanam di sudut rendah kebun, daunnya kuning dan layu.

```
Crop and fruit-tree doctor: 13 pieces of advice so far.
Answer each question; press Enter if you do not know.
Which plant has the problem? (durian/mango/chili/rice) > durian
What do you see? (wet patch on bark/rotting fruit on tree/dark sunken
spots/stung fruit/yellow leaves/diamond spots on leaves) > yellow leaves
I have no advice for this yet. Ask the penyuluh.
Which advice is right? (label, Enter if none) > root rot care
Another fact that matters? (variable = value, Enter when done) > soil = waterlogged
Another fact that matters? (variable = value, Enter when done) >
Learned: added: root rot care#1 added at the top level
Saved ../data/farm-kb.json: Crop and fruit-tree doctor: 8 variables, 14 nodes, 14 rules
```

Tidak ada node yang cocok: node-node durian tentang kulit batang dan buah, dan node-node daun kuning tentang cabai dan padi. Dokter mengatakannya dengan jujur. Mbak Nur, yang lewat dalam kelilingnya, melihat air menggenang di akar setelah hujan semalam dan mengenali busuk akar, *Phytophthora* yang sama dengan penyebab kanker batang [2]. Ia mengajarkan `root rot care` dan fakta yang penting, `soil = waterlogged`. Karena tidak ada yang menjawab, `learn` menambahkan node tingkat atas baru dengan ketiga faktanya.

Sesi 4: Mbak Nur menelaah node-node hasil bangkitan. Di akhir bulan, di pos kelompok tani, ia menjalankan telaahnya.

```
 1. patch canker care#1G: 15 conditions, from patch canker care#1
 2. durian fruit rot care#1G: 10 conditions, from durian fruit rot care#1
 3. anthracnose care#1G: 13 conditions, from anthracnose care#1
 4. fruit fly control#1G: 14 conditions, from fruit fly control#1
 5. anthracnose care#2G: 13 conditions, from anthracnose care#2
 6. nutrient check#1G: 7 conditions, from nutrient check#1
 7. nutrient check#2G: 7 conditions, from nutrient check#2
 8. blast control#1G: 11 conditions, from blast control#1
 9. root rot care#1G: 7 conditions, from root rot care#1
Review which candidate? (number, Enter to finish) > 3
   1. patches = drying
   2. patch around trunk = yes
   3. leaves curl = yes
   4. insects = whitefly
   5. soil = waterlogged
   6. crop = durian
   7. symptom = yellow leaves
   8. crop = chili
   9. crop = rice
  10. symptom = rotting fruit on tree
  11. symptom = diamond spots on leaves
  12. crop = mango
  13. symptom = dark sunken spots
  -> advice = anthracnose care
Keep which conditions? (numbers, Enter to reject) > 13
Accepted: anthracnose care#1G: symptom = dark sunken spots -> advice = anthracnose care
Review which candidate? (number, Enter to finish) >
Saved ../data/farm-kb.json: Crop and fruit-tree doctor: 8 variables, 15 nodes, 15 rules
```

Kini ada sembilan kandidat, satu lebih banyak daripada di Subbab 6.4, karena node busuk akar yang baru juga memakai `symptom`, dan urutan variabel relatif ikut berubah: kandidat antraknosa kali ini punya tiga belas kondisi, dan yang ia inginkan nomor 13. Ia menyimpannya, dan dokter menyimpan `anthracnose care#1G: symptom = dark sunken spots -> advice = anthracnose care`. Ia menolak sisanya cukup dengan tidak menelaahnya. Seluruh telaah hanya butuh tiga jawaban.

Sesi 5: pepaya Pak Harto. Pak Harto, tetangga Pak Slamet, menanam pepaya, tanaman yang tak pernah diajarkan kepada dokter itu.

```
Crop and fruit-tree doctor: 15 pieces of advice so far.
Answer each question; press Enter if you do not know.
Which plant has the problem? (durian/mango/chili/rice) > papaya
What do you see? (wet patch on bark/rotting fruit on tree/dark sunken
spots/stung fruit/yellow leaves/diamond spots on leaves) > dark sunken spots
Advice (anthracnose care):
  Dark sunken spots are anthracnose, a fungus that attacks many fruits.
  Remove and burn spotted fruit and leaves, prune for air, and never keep
  seed from spotted fruit. Ask the penyuluh about hot-water dipping after
  harvest.
Because: symptom = dark sunken spots (rule anthracnose care#1G)
Is the advice right? (yes/no) > yes
Good. The case is recorded as checked.
```

Pertanyaan pertama adalah jenis tanaman, *papaya*, dan setiap node khusus-tanaman gagal karenanya. Node hasil bangkitan hanya membutuhkan gejalanya, yang ditanyakan, dan ia menyala: saran antraknosa, karena `symptom = dark sunken spots`, dari rule `anthracnose care#1G`. Dokter menjawab kebun yang tak dapat dijawab node mana pun yang diajarkan, dengan pengetahuan yang sudah dikonfirmasi pakar.

`--tree` memperlihatkan basis pengetahuan setelah kelima kunjungan, dengan node hasil bangkitan bertanda di bagian bawah:

```
(root)
+- patch canker care#1: crop = durian, symptom = wet patch on bark -> advice = patch canker
care
|  +- drain and treat canker#1: soil = waterlogged -> advice = drain and treat canker
|  +- canker emergency#1: patch around trunk = yes -> advice = canker emergency
+- durian fruit rot care#1: crop = durian, symptom = rotting fruit on tree -> advice =
durian fruit rot care
+- anthracnose care#1: crop = mango, symptom = dark sunken spots -> advice = anthracnose
care
+- fruit fly control#1: crop = mango, symptom = stung fruit -> advice = fruit fly control
+- anthracnose care#2: crop = chili, symptom = dark sunken spots -> advice = anthracnose
care
+- nutrient check#1: crop = chili, symptom = yellow leaves -> advice = nutrient check
|  +- yellow virus control#1: leaves curl = yes, insects = whitefly -> advice = yellow
virus control
+- nutrient check#2: crop = rice, symptom = yellow leaves -> advice = nutrient check
|  +- planthopper control#1: insects = brown hoppers -> advice = planthopper control
|     +- planthopper emergency#1: patches = drying -> advice = planthopper emergency
+- blast control#1: crop = rice, symptom = diamond spots on leaves -> advice = blast
control
+- root rot care#1: crop = durian, symptom = yellow leaves, soil = waterlogged -> advice =
root rot care
+- anthracnose care#1G: symptom = dark sunken spots -> advice = anthracnose care
[generated]
```

Dan `--usage` memperlihatkan bagaimana tingkat pemakaian bergeser:

```
Node usage rates (NUR), lowest first:
  NodeID                    NumOfVariableIDs  NUR
  ------------------------  ----------------  -------
  canker emergency#1        1                 1.0000
  planthopper emergency#1   1                 1.0000
  yellow virus control#1    2                 1.2500
  drain and treat canker#1  1                 2.0000
  root rot care#1           3                 3.8889
  planthopper control#1     1                 4.0000
  durian fruit rot care#1   2                 7.2500
  anthracnose care#1        2                 7.2500
  fruit fly control#1       2                 7.2500
  anthracnose care#2        2                 7.2500
  blast control#1           2                 7.2500
  anthracnose care#1G       1                 10.0000
  nutrient check#1          2                 14.5000
  nutrient check#2          2                 14.5000
  patch canker care#1       2                 21.7500
Rule usage rates (RUR), lowest first:
  RuleID                    RUR
  ------------------------  -------
  root rot care#1           3.8889
  planthopper emergency#1   6.5000
  durian fruit rot care#1   7.2500
  anthracnose care#1        7.2500
  fruit fly control#1       7.2500
  anthracnose care#2        7.2500
  blast control#1           7.2500
  yellow virus control#1    7.8750
  planthopper control#1     9.2500
  anthracnose care#1G       10.0000
  canker emergency#1        11.3750
  drain and treat canker#1  11.8750
  nutrient check#1          14.5000
  nutrient check#2          14.5000
  patch canker care#1       21.7500
Most important nodes: patch canker care#1, nutrient check#2
Most important variables: symptom
```

Dua hal berubah sejak Subbab 6.4. Node busuk akar yang baru punya tiga kondisi, dan RUR-nya (3,89) kini yang terendah: ia pengetahuan terbaru dan paling sedikit dibagi di basis ini, dan pembangkitan berikutnya akan dimulai darinya. Dan variabel terpentingnya kini `symptom` saja: node hasil bangkitan memakai `symptom` tanpa `crop`, sehingga `symptom` dibagi satu node lebih banyak daripada `crop`. Basis pengetahuan itu sendiri telah mencatat apa yang diajarkan Mbak Nur: gejala lebih penting daripada jenis tanaman.

6.7 Membuatnya Lebih Baik

Telaah terjadwal. Tesis menyarankan bahwa penyempurnaan dapat dijalankan ketika basis pengetahuan dibuka, menurut jadwal, atau atas permintaan [7]. Untuk dokter tani, telaah bulanan di pos kelompok tani cocok dengan irama musim: cukup banyak kasus baru untuk mengubah tingkat-tingkatnya, dan tidak terlalu banyak kandidat sampai telaahnya menjadi beban.

Tunjukkan mengapa sebuah kandidat diusulkan. Layar telaah mendaftar kondisi tanpa tingkat pemakaiannya. Mencetak VUR setiap kondisi di sampingnya (dengan `usage().vur(node, i)` untuk node asalnya) membantu penyuluh melihat bagian mana dari kandidat yang paling diandalkan basis pengetahuan; Soal 6.6 memintanya.

Bangkitkan rule di tempat node saling memberi makan. Di dokter ini setiap node berakhir dengan saran, sehingga *rule* hasil bangkitan jarang berguna. Bila pengetahuannya dibagi dua

langkah — pertama masalahnya (`problem = anthracnose`), lalu apa yang dilakukan dengan memperhatikan tanaman dan musimnya — node-node mulai menyimpulkan fakta yang dipakai node lain, dan rule hasil bangkitan menjadi rantai yang layak disimpan, seperti pada Zookeeper (Soal 6.8). Bab 8 memakai rantai seperti itu untuk penalaran maju dan mundur.

Tandai saran hasil bangkitan. Pohon menampilkan `[generated]` untuk node yang berasal dari pembangkitan, dan riwayat mencatat siapa yang menerimanya. Dokter dapat pula menyampaikannya kepada petani — *saran ini mencakup banyak tanaman; bila tidak membantu, tanyakan kepada penyuluh* — agar pengetahuan umum dipakai dengan sedikit lebih hati-hati daripada pengetahuan yang diajarkan untuk satu tanaman.

Foto. Petani menggambarkan gejala dengan kata-kata, dan kata-kata beragam. Foto yang dikirim ke penyuluh bersama konsultasi membuatnya dapat memeriksa gejala sebelum mengonfirmasi saran atau mengajarkan kasus baru. Bab 13 memperlihatkan, untuk dokter Wi-Fi, cara menghubungkan dokter semacam ini dengan model bahasa yang dapat membantu mengubah kata-kata petani sendiri menjadi variabel-variabel dokter, dengan pengetahuannya sendiri tetap diperiksa manusia.

Pengujian. `FarmTest.java` dan `test_farm.py` memeriksa perhitungan pojok padi (NUR 3 untuk `nutrient check#1`, RUR 2 untuk rule darurat), pembangkitan pada basis pengetahuan awal (tiga belas node, dua node terpenting, `nutrient check#2` pertama dalam urutan relatif, satu kandidat rule dan delapan kandidat node, pepaya baru terjawab setelah `anthracnose care#1G` diterima), serta pembungkus kata dan berkas kata-kata dokter itu. Keduanya diakhiri dengan:

```
all tests passed
```

6.8 Memakainya Sungguh-Sungguh

Bersama kelompok tani. Dokter ini cocok untuk *kelompok tani*, yang bertemu teratur dengan penyuluhnya. Petani berkonsultasi di ponsel atau di laptop kelompok; penyuluh mengajarinya dalam kelilingnya dan menelaah kandidat hasil bangkitan di pertemuan kelompok, tempat petani lain dapat menyatakan apakah sebuah generalisasi cocok dengan yang mereka lihat. Setelah beberapa musim, basis pengetahuan menyimpan masalah-masalah daerah *ini*, dengan kata-kata *daerah ini*.

Pakai nama setempat. Petani tidak berkata *Phytophthora*; mereka berkata *kanker batang*, *busuk akar*, *patek*, *lalat buah*, *wereng*, *virus kuning*. Label saran dan pilihan jawaban pertanyaan semestinya nama yang dipakai petani, dan teks saran dapat menyebut nama ilmiahnya sekali, untuk penyuluh. Mesin tidak peduli apa nama nilainya, asalkan hal yang sama selalu disebut dengan cara yang sama.

Saran yang aman diikuti. Dokter tani tidak boleh menyuruh petani menyemprot bahan kimia tertentu dengan dosis tertentu; itu bergantung pada apa yang terdaftar, pada tanamannya, fasenya, dan cuacanya, dan itu tugas penyuluh. Dokter dalam bab ini memberi langkah-langkah yang selalu aman — sanitasi, drainase, perangkap, tanaman penghalang, varietas tahan, tidak menyemprot berlebihan — dan mengarahkan petani ke penyuluh untuk selebihnya. Bila pestisida dipakai, labelnya adalah hukum: pakaian pelindung, masa tunggu sebelum panen, dan jangan pernah di dekat sumber air.

Musim itu penting. Antraknosa dan *Phytophthora* adalah penyakit musim hujan; ledakan wereng mengikuti tanam yang tidak serempak dan nitrogen yang berlebihan [6], [12], [2]. Variabel

seperti `season` (wet, dry), yang ditanyakan hanya bila sebuah node membutuhkannya, membuat penyuluh dapat mengajarkan pengecualian musiman — *pada musim kemarau, bercak ini mungkin bukan antraknosa* — persis seperti Bab 5 mengajarkan pengecualian.

Berbagi pengetahuan antardesa. Dua penyuluh di kecamatan bertetangga akan mengajari dokternya dengan cara berbeda, dan keduanya mungkin benar untuk tanah dan varietasnya masing-masing. Sebelum basis pengetahuan satu dokter disalin ke desa lain, ia semestinya dibaca bersama penyuluh setempat — pohon dan Rule Structure memungkinkannya dalam satu sore — dan setiap node yang ia ragukan dihentikan di sana alih-alih dihapus, agar riwayat menyimpan alasannya. Menggabungkan dua basis pengetahuan node demi node, bersama kasus-kasus di baliknya, adalah pekerjaan tim, dan Bab 11 membangun alat-alatnya.

Tanaman lain, pakar lain. Tidak ada bagian `FarmDoctor` yang khusus untuk keempat tanaman ini kecuali pertanyaan dan berkas kasus awalnya. Koperasi kelapa, kelompok petani bawang merah, kelompok pembudidaya ikan desa — masing-masing dapat menulis variabelnya sendiri, selusin kasus awal, dan membiarkan pakarnya sendiri mengajar dan menelaah. Tingkat pemakaian dan pembangkitan bekerja dengan cara yang sama pada basis pengetahuan mana pun.

6.9 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Menerima kandidat apa adanya. Kandidat hasil bangkitan adalah daftar semua yang datang sebelum sebuah butir penting dalam urutan relatif; sebagian besar isinya keliru untuk konklusinya. *Hindari:* simpan hanya kondisi (atau node) yang benar untuk konklusinya, dan tolak kandidatnya bila tidak ada.

Menggeneralisasi terlalu jauh. `symptom = yellow leaves` saja akan mengirim setiap tanaman berdaun kuning ke pemeriksaan nitrogen, termasuk durian yang tenggelam dalam air. *Hindari:* sebelum menyimpan satu kondisi saja, pikirkan tanaman atau keadaan tempat kondisi itu keliru; bila Anda menemukannya, simpan satu kondisi lagi, atau tolak kandidatnya.

Membaca tingkat pemakaian sebagai popularitas. NUR yang tinggi tidak berarti banyak petani mendapat saran itu; artinya banyak pengetahuan melewati node itu. *Hindari:* pakai tingkat-tingkat itu untuk apa yang diukurnya — struktur pengetahuan — dan simpan hitungan konsultasi terpisah bila Anda membutuhkannya.

Membandingkan tingkat antarbasis pengetahuan. Tingkat-tingkat itu bergantung pada berapa node yang dimiliki sebuah basis pengetahuan dan bagaimana node-node itu berbagi variabel; NUR 12 di satu basis dan 12 di basis lain tidak berkata apa-apa tentang satu sama lain. *Hindari:* bandingkan tingkat hanya di dalam satu basis pengetahuan, dan hanya pada satu saat.

Lupa bahwa tingkat-tingkat itu berubah. Setiap node baru mengubah NS, dan setiap pengecualian baru menambah Credit, sehingga kandidatnya ikut berubah: kandidat antraknosa punya dua belas kondisi di Subbab 6.4 dan tiga belas pada sesi 4. *Hindari:* selalu telaah kandidat basis pengetahuan dalam keadaannya sekarang, dan baca angka di layar alih-alih mengingatnya.

Menghitung dengan tangan memakai Credit yang salah. Credit hanya diperoleh kondisi-kondisi *induk* node baru, dan hanya bila kasus barunya menyetujuinya. *Hindari:* ketika memeriksa tingkat pemakaian dengan tangan, catat pembuatan setiap node dan di bawah induk mana ia digantung.

Menyarankan bahan kimia. Dokter yang menyebut pestisida dan dosisnya dapat merugikan petani, panen, dan sungai. *Hindari:* berikan langkah-langkah yang aman, dan arahkan petani ke penyuluh untuk apa pun yang berhubungan dengan bahan kimia.

6.10 Ringkasan Bab

- **Analisis nilai** mengukur seberapa besar struktur basis pengetahuan memakai setiap potongannya. Setiap kondisi punya **Credit** (seberapa sering kasus berikutnya menyetujuinya), **NS** (berapa node berbagi variabelnya), dan **CD** (posisinya dibagi jumlah kondisi di node-nya); **Weight**-nya NS × CD dan **VUR**-nya Credit × Weight; **NUR** sebuah node adalah rata-rata VUR kondisi-kondisinya dan **RUR** sebuah rule rata-rata NUR node-node-nya (Persamaan (6.1)–Persamaan (6.6), **Gambar 6.1**).
- Dihitung dengan tangan pada pojok padi, tingkat-tingkat itu cocok dengan mesin pada keempat belas angka (**Kode 6.1**, **Kode 6.2**).
- Tingkat pemakaian memberi **urutan relatif** node dan kondisi, dari yang paling sedikit dipakai, yang menjaga urutan setiap rule yang ada. **Pembangkitan rule** mengusulkan rule dan node yang berakhir pada node dan variabel terpenting, dan pakar hanya menyimpan yang benar (**Gambar 6.2**).
- Pada basis pengetahuan tani, satu-satunya kandidat rule ditolak — pembangkitan rule membutuhkan node-node yang saling memberi makan, seperti pada Zookeeper — dan satu kondisi sebuah kandidat node, `symptom = dark sunken spots`, menjadi node hasil bangkitan yang menjawab kebun pepaya dan buah naga yang tak pernah diajarkan (**Kode 6.3**, **Kode 6.4**, **Gambar 6.3**).
- **Dokter tanaman dan pohon buah** (**Kode 6.5**, **Kode 6.6**) memberi saran untuk durian, mangga, cabai, dan padi dengan agronomi terkini yang aman, mempelajari busuk akar dari penyuluh, membiarkannya menelaah kandidat hasil bangkitan, dan menjawab petani pepaya dengan node yang ia terima; tingkat pemakaiannya menunjukkan bahwa gejala telah menjadi lebih penting daripada jenis tanaman.

6.11 Soal Latihan

6.1 Di pojok padi, andaikan Mbak Nur mengajarkan satu pengecualian lagi di bawah `nutrient check#1`: padi berdaun kuning dengan serangga *putih* di pangkalnya, yang sarannya berbeda. Credit mana yang berubah? Hitung ulang VUR dan NUR `nutrient check#1`.

6.2 Sebuah node punya tiga kondisi, dan variabel-variabelnya dibagi oleh 1, 3, dan 2 node. Semua Credit-nya 1. Hitung CD, Weight, dan VUR setiap kondisi serta NUR node itu. Kondisi mana yang paling berbobot, dan mengapa?

6.3 Jelaskan dengan kata-kata sendiri mengapa tesis memberi kondisi di dekat ujung node bobot lebih besar daripada kondisi di awalnya. Pakai `crop = rice, symptom = yellow leaves` sebagai contoh.

6.4 Di Subbab 6.4, RUR `planthopper emergency#1` (5,67) lebih rendah daripada rule-rule satu-node ber-RUR 6,00, padahal node pertamanya, `nutrient check#2`, ber-NUR 12,00. Hitunglah dari NUR ketiga node-nya, dan jelaskan hasilnya.

6.5 Mengapa mesin tidak mengusulkan kandidat rule untuk `nutrient check#2`, yang merupakan salah satu dari dua node terpenting?

6.6 Ubahlah layar telaah dokter agar mencetak, di samping setiap kondisi kandidat, VUR kondisi itu di node asalnya (atau tanda pisah bila berasal dari node lain). Panggilan mesin apa saja yang Anda perlukan?

6.7 Mbak Nur menerima `symptom = dark sunken spots` saja. Usulkan keadaan tempat node hasil bangkitan ini dapat memberi saran keliru, dan jelaskan bagaimana ia membetulkannya tanpa menghapus node itu.

6.8 Jalankan pembangkitan rule pada kasus-kasus Zookeeper dari tesis (`vcirs-core/data/zookeeper-cases.json`), cetak semua kandidat rule, terima `carnivore#1G` dengan menyimpan node `mammal#2`, `mammal#1`, dan `carnivore#1`, dan cetak aturan JIKA-MAKA yang dihasilkannya setelah transformasi basis pengetahuan. Mengapa kandidat ini layak disimpan, sedangkan kandidat dokter tani tidak?

6.9 Dokter menanyakan jenis tanaman lebih dulu karena setiap kasus awal dimulai dengannya. Setelah ada node hasil bangkitan, apakah lebih baik menanyakan gejalanya lebih dulu? Diskusikan, dengan memakai tingkat pemakaian yang dicetak setelah sesi 5 dan gagasan `questionOrder()` dari Bab 4.

6.12 Penyelesaian Soal Latihan

Penyelesaian 6.1. Kasus baru menyetujui kedua kondisi induknya, sehingga `crop = rice` dan `symptom = yellow leaves` di `nutrient check#1` masing-masing mendapat satu Credit lagi: 3. NS tidak berubah (pengecualian baru memakai `insects`, yang kini dibagi dua node, tetapi `crop` dan `symptom` tetap dipakai dua node). VUR-nya menjadi $3 \times 1 = 3$ dan $3 \times 2 = 6$, dan NUR-nya $(3 + 6)/2 = 4.5$. Kondisi pengecualian baru itu sendiri, `insects = white`, mulai dengan Credit 1; dan karena `insects` kini dibagi dua node, VUR `insects = brown hoppers` di `planthopper control#1` pun berlipat dua, menjadi $2 \times 2 = 4$.

Penyelesaian 6.2. CD-nya $\frac{1}{3}$, $\frac{2}{3}$, dan 1. Weight-nya $1 \times \frac{1}{3} \approx 0.33$, $3 \times \frac{2}{3} = 2$, dan $2 \times 1 = 2$. Dengan Credit 1, VUR sama dengan Weight, dan NUR-nya $(0.33 + 2 + 2)/3 \approx 1.44$. Kondisi kedua dan ketiga sama berbobotnya: yang kedua lebih luas dibagi, yang ketiga berdiri lebih dekat ke konklusi.

Penyelesaian 6.3. Kondisi di awal node menyempitkan bidang dengan cara yang umum; kondisi-kondisi terakhir menentukan kasusnya. `crop = rice` hanya menyatakan bahwa node ini tentang padi; banyak node padi berbagi kondisi itu. `symptom = yellow leaves` yang membuat node ini menjadi *saran ini*, bukan pengendalian blas. Derajat kedekatan menghargai kondisi yang akhirnya menentukan, yang juga kondisi yang akan disebut pakar bila ditanya "mengapa saran ini?".

Penyelesaian 6.4. Rule-nya `nutrient check#2 -> planthopper control#1 -> planthopper emergency#1`, dengan NUR 12,00, 4,00, dan 1,00. RUR-nya $(12 + 4 + 1)/3 = 17/3 \approx 5.67$. Kedua node pengecualian masing-masing punya satu kondisi yang tidak dipakai di tempat lain, dengan VUR rendah, dan keduanya menarik rata-ratanya ke bawah: sebuah rule dinilai dari semua node-nya, dan rule ini berakhir dengan pengetahuan yang sangat spesifik dan jarang dibagi.

Penyelesaian 6.5. Rule hasil bangkitan untuk sebuah node penting adalah bagian urutan node relatif yang datang sebelumnya, diakhiri node itu. `nutrient check#2` adalah node paling pertama dalam urutan itu, sehingga kandidatnya adalah rangkaian satu-node `nutrient check#2` — yang sudah menjadi rule dalam basis pengetahuan. Mesin tidak mengusulkan rule yang sudah ada [7].

Penyelesaian 6.6. Untuk setiap kondisi kandidat, carilah node basis pengetahuan yang kondisi-kondisinya memuatnya pada suatu posisi `i` — node asal kandidat, `kb.node(candidate.source)` (Python `kb.node(g.source)`), adalah tempat pertama yang wajar — dan cetak `kb.usage().vur(node.id, i)` di sampingnya, diformat dengan `Num.fixed(x, 4)`, atau tanda pisah bila kondisi itu hanya berasal dari node lain. Panggilannya `kb.node`, `node.conditions`, `Condition.same` untuk membandingkan, dan `usage().vur`. Menghitung `kb.usage()` sekali, sebelum perulangan, membuatnya cepat.

Penyelesaian 6.7. Beberapa penyakit dan luka buah juga meninggalkan tanda gelap cekung — misalnya terbakar matahari lalu busuk, atau memar karena jatuh — dan petani yang menggambarkannya sebagai "bercak hitam cekung" akan diberi tahu bahwa itu antraknosa. Mbak Nur membetulkannya dengan cara RDR: ketika kasus seperti itu datang, ia mengajarkan pengecualian di bawah `anthracnose care#1G` dengan fakta yang membedakannya (misalnya `marks only on the sunny side = yes`), dan node hasil bangkitan tetap bekerja untuk semua yang lain. Bila ia memutuskan generalisasinya keliru, ia menghentikan node itu, seperti di Bab 5; node itu tidak pernah dihapus.

Penyelesaian 6.8. Program penyelesaiannya adalah `ZooGeneration.java` dan `zoo_generation.py`. Versi Python-nya:

```
"""Soal 6.8: pembangkitan rule di tempat ia bersinar, pada Zookeeper
tesis VCIRS, yang node-nya menyimpulkan fakta yang dipakai node lain."""

import pathlib
import sys

CORE = pathlib.Path(__file__).resolve().parents[2] / "vcirs-core"
sys.path.insert(0, str(CORE / "python"))
from vcirs import KnowledgeBase, casefile

name, description, steps = casefile.read_cases(
    str(CORE / "data" / "zookeeper-cases.json"))
kb = casefile.build(KnowledgeBase(name, description), steps)
candidates = kb.generate_rules()
for g in candidates:
    print(g.id + ": " + " -> ".join(g.nodes))
carnivore = [g for g in candidates if g.id == "carnivore#1G"][0]
rule = kb.accept_rule(carnivore, ["mammal#2", "mammal#1", "carnivore#1"],
                      "zookeeper")
print("Kept: " + str(rule))
for r in kb.transform():
    if r.id == rule.id:
        print("\n".join(r.lines()))
```

Program ini mencetak:

```
carnivore#1G: bird#2 -> albatross#1 -> bird#1 -> mammal#2 -> mammal#1 -> carnivore#1
cheetah#1G: bird#2 -> albatross#1 -> bird#1 -> mammal#2 -> mammal#1 -> carnivore#1 ->
carnivore#2 -> ungulate#1 -> ungulate#2 -> ostrich#1 -> penguin#1 -> giraffe#1 -> zebra#1 -
> cheetah#1
giraffe#1G: bird#2 -> albatross#1 -> bird#1 -> mammal#2 -> mammal#1 -> carnivore#1 ->
carnivore#2 -> ungulate#1 -> ungulate#2 -> ostrich#1 -> penguin#1 -> giraffe#1
ostrich#1G: bird#2 -> albatross#1 -> bird#1 -> mammal#2 -> mammal#1 -> carnivore#1 ->
carnivore#2 -> ungulate#1 -> ungulate#2 -> ostrich#1
ungulate#1G: bird#2 -> albatross#1 -> bird#1 -> mammal#2 -> mammal#1 -> carnivore#1 ->
carnivore#2 -> ungulate#1
Kept: carnivore#1G: mammal#2 -> mammal#1 -> carnivore#1
carnivore#1G [RULE CF=50]
  IF   give milk = yes [CF=50]
  AND  has hair = yes [CF=50]
  AND  mammal = yes [CF=50]
  AND  eat meat = yes [CF=50]
  THEN carnivore = yes
```

Kelima kandidat rule-nya adalah kandidat tesis, termasuk `giraffe#1G`, yang tidak ada di tabel tesis walaupun nilainya seri dengan yang lain (catatan desain mesin menjelaskan perbedaannya). Menyimpan `mammal#2`, `mammal#1`, dan `carnivore#1` memberi rule yang, setelah transformasi, berbunyi *jika ia menyusui, berambut, mamalia, dan makan daging, maka ia karnivora* — benar, walaupun lebih spesifik daripada perlu, dan tepat rule yang disimpan tesis [7]. Rule ini layak disimpan karena node-node-nya saling memberi makan: `mammal#1` dan `mammal#2` menyimpulkan `mammal = yes`, yang dipakai `carnivore#1`. Di basis pengetahuan tani tidak ada node yang menyimpulkan sesuatu yang dipakai node lain, sehingga rangkaian node tani bukan rantai melainkan daftar saran yang tak berhubungan.

Penyelesaian 6.9. Setelah sesi 5, `symptom` adalah variabel terpenting, dibagi sepuluh node berbanding sembilan untuk `crop`, dan node hasil bangkitan hanya membutuhkan gejalanya. Menanyakan gejala lebih dulu akan membuat dokter menjawab petani pepaya dengan satu pertanyaan, dan langsung menyingkirkan sebagian besar node bagi petani mana pun. Tetapi sembilan dari sepuluh node itu tetap membutuhkan jenis tanaman juga, sehingga kebanyakan konsultasi tetap akan menanyakan keduanya. `questionOrder()` mengurutkan variabel menurut VUR tertingginya, dan setelah node hasil bangkitan `symptom` berada di urutan pertama; aplikasi yang mengikutinya menanyakan gejala lebih dulu, yang juga cara Mbak Nur sendiri memulai di pasar: *apa yang Bapak lihat?* Bab 7 menerapkan urutan pertanyaan itu.

Daftar Pustaka

[1] CNBC Indonesia, “Lahan pertanian makin sempit, petani gurem naik pesat” (Sensus Pertanian 2023, Badan Pusat Statistik), Dec. 4, 2023. [Online]. Available: https://www.cnbcindonesia.com/news/20231204161252-4-494410/lahan-pertanian-makin-sempit-petani-gurem-naik-pesat

[2] A. Singh, C. Chow, K. Nathaniel, Y. L. Vun, S. Javad, and K. Jabeen, “Management of *Phytophthora* and *Phytopythium* oomycete diseases in durian (*Durio zibethinus*),” *Crop Protection*, vol. 190, Art. no. 107086, 2025.

[3] A. Worku, A. Alemu, and B. Tsedaley, “Combining plant extracts and hot water treatments for the management of postharvest mango anthracnose (*Colletotrichum gloeosporioides*),” *Scientific Reports*, vol. 15, Art. no. 5074, 2025.

[4] S. De Faveri, S. Vijaysegaran, and J. Cheesman, *Area-Wide Management of Methyl Eugenol-Attracted Fruit Flies in Mango: A Systems Approach*, ACIAR Monograph No. 222. Canberra, Australia: Australian Centre for International Agricultural Research, 2024.

[5] Y. M. A. Sandra, "Identification of recovery traits in chili (*Capsicum* spp.): implications for breeding begomovirus-resistant varieties," *Plumula: Berkala Ilmiah Agroteknologi*, vol. 13, no. 2, pp. 63–70, Jul. 2025.

[6] Pacific Pests, Pathogens and Weeds, "Rice brown planthopper (064)," fact sheet, v10. [Online]. Available: https://apps.lucidcentral.org/pppw_v10/text/web_full/entities/rice_brown_planthopper_064.htm

[7] M. M. I. Subakti, "A Variable-Centered Intelligent Rule System," M.Sc.Eng. thesis, Dept. of Computer Science and Information Engineering, Taipei, Taiwan, Jun. 2005.

[8] I. Subakti, "A variable-centered intelligent rule system," in *Proc. 1st Annual Int. Conf. Information and Communication Technology Seminar (ICTS 2005)*, vol. 1, no. 1, Surabaya, Indonesia, Aug. 2005, pp. 167–174.

[9] I. Subakti, *Variable-Centered Intelligent Rule System*, Indonesian ed. (resume). Surabaya, Indonesia: Dept. of Informatics, Institut Teknologi Sepuluh Nopember, 2006.

[10] P. Compton and B. H. Kang, *Ripple-Down Rules: The Alternative to Machine Learning*. Boca Raton, FL, USA: CRC Press, 2021.

[11] L. Sebayang, *Teknik Pengendalian Penyakit Kuning pada Tanaman Cabai*. Medan, Indonesia: Balai Pengkajian Teknologi Pertanian Sumatera Utara, 2013; published by Direktorat Jenderal Hortikultura, Kementerian Pertanian. [Online]. Available: https://hortikultura.pertanian.go.id/wp-content/uploads/2024/11/Teknik-Pengendalian-Penyakit-Kuning-Pada-Tanaman-Cabai_watermark.pdf

[12] International Rice Research Institute, "Blast (leaf and collar)," *Rice Knowledge Bank*. [Online]. Available: http://www.knowledgebank.irri.org/training/fact-sheets/pest-management/diseases/item/blast-leaf-collar

Bab 7

Pengarah Pesan Pelanggan Toko Kecil: Statistik Pemakaian dan Kredit

Setelah menyelesaikan bab ini Anda akan mampu mengubah pesan-pesan WhatsApp sebuah toko kecil, yang ditulis dalam bahasa Indonesia, bahasa Inggris, atau campuran keduanya, menjadi segelintir **sinyal** ya-atau-tidak; mengajari basis pengetahuan VCIRS agar mengirim setiap pesan kepada orang yang tepat di toko, dan membetulkannya dengan satu kalimat bila ia keliru; menjelaskan apa yang dihitung oleh **Credit** sebuah kondisi dan bagaimana VCIRS mengubah Credit menjadi tiga tingkat pemakaian dalam tesisnya — **Variable Usage Rate** (VUR), **Node Usage Rate** (NUR), dan **Rule Usage Rate** (RUR); menghitung tingkat-tingkat itu dengan tangan dan mencocokkannya dengan mesin; serta membaca angka-angka itu seperti seorang pemilik toko membacanya: sinyal mana yang paling penting, aturan mana yang paling sering dikoreksi, pertanyaan mana yang harus didengarkan lebih dulu, dan di mana dua aturan berebut pesan yang sama. Aplikasi bab ini, **pengarah pesan** Toko Rina, ditulis dalam Java dan Python, dan keduanya mencetak keluaran yang persis sama.

7.1 Ponsel Bu Rina

Toko Rina adalah toko kelontong di sudut sebuah perumahan. Ia menjual beras, minyak goreng, telur, gula, mi instan, tabung elpiji, dan air minum galon sembilan belas liter, dan seperti kebanyakan toko kecil di Indonesia sekarang, sebagian besar pesanannya datang lewat WhatsApp. Itu tidak mengherankan: dalam laporan Digital 2026 untuk Indonesia, WhatsApp adalah platform sosial yang paling banyak dipakai di negeri ini, digunakan setiap bulan oleh sekitar sembilan dari sepuluh pengguna internet [1]. Bagi sebuah toko, artinya etalase kini punya pintu kedua, dan pintu yang ini tidak pernah tutup.

Tiga orang menjaga pintu itu. **Bu Rina**, pemiliknya, hafal harga dan stok serta menerima pesanan. Putrinya, **Dinda**, memegang uang: ia memeriksa transfer, pembayaran QRIS, dan utang kecil para tetangga yang membayar di akhir bulan. Suaminya, **Pak Darto**, mengantar barang dengan sepeda motornya. Mereka berbagi satu ponsel, nomor WhatsApp Business toko, dan pada pagi yang ramai ponsel itu menerima pesan setiap beberapa menit. Ada yang sederhana: *Bu, gula pasir 1 kg harganya berapa?* Ada yang mendesak: *Minyaknya bocor di plastik, tolong diganti*. Ada yang berbahasa Inggris, dari mahasiswa dan keluarga muda yang mengontrak di perumahan itu: *Do you deliver to Jalan Melati?* Dan ada yang sampah: *Selamat! Anda menang undian, klik www…*

Masalahnya bukan menjawab; semua anggota keluarga bisa menjawab. Masalahnya adalah **siapa** yang menjawab. Pesan pembayaran yang dibaca Pak Darto harus menunggu sampai Dinda pulang. Keluhan yang terlambat dibaca menjadi pelanggan yang marah. Pertanyaan tentang ongkos kirim yang dijawab Bu Rina tetap harus diteruskan kepada Pak Darto. WhatsApp Business menyediakan label dan balasan cepat agar obrolan tetap rapi [2], tetapi tetap harus ada orang yang membaca setiap pesan dan memilih labelnya. Bu Rina ingin ponsel itu melakukan pembacaan pertama untuknya: melihat setiap pesan, memutuskan siapa yang harus menanganinya, dan menaruhnya di tumpukan orang itu — lalu, bila ia keliru, belajar dari koreksinya, seperti dokter Wi-Fi di Bab 3 belajar dari Mas Anto.

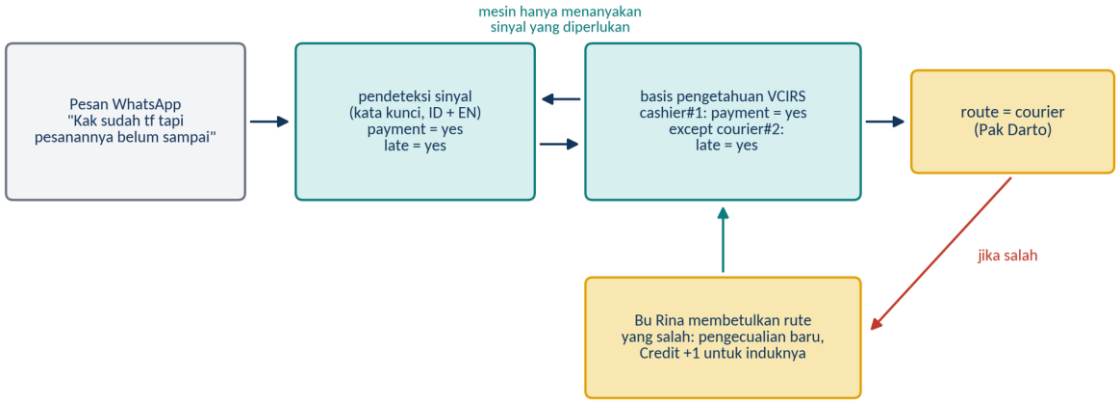
Bab ini membangun pengarah pesan itu. Namun bab ini juga menjawab pertanyaan kedua yang cepat atau lambat diajukan setiap pemilik usaha tentang sistem apa pun yang belajar: *apa yang sudah dipelajarinya, dan apa yang paling penting di dalamnya?* Setelah beberapa minggu koreksi, Bu Rina ingin tahu jenis pesan mana yang paling sering merepotkan, kata-kata mana yang paling diandalkan pengarah, dan apa yang harus didengarkan lebih dulu oleh pembantu baru ketika pelanggan mengirim pesan suara alih-alih tulisan. VCIRS menjawab pertanyaan-pertanyaan itu dengan angka-angka yang memang sudah disimpannya, sebagai hasil sampingan dari belajar: **Credit** setiap kondisi dan **tingkat pemakaian** yang dihitung darinya [3], [4]. Angka-angka itulah pokok bahasan bab ini yang sebenarnya.

7.2 Gagasannya dengan Kata-Kata Sederhana

7.2.1 Sinyal, bukan pemahaman

Program komputer yang sungguh memahami pesan berbahasa Indonesia dan Inggris tentu luar biasa, dan Bab 13 menghubungkan sebuah model bahasa lokal dengan sistem pakar. Tetapi sebuah toko tidak memerlukan pemahaman untuk mengarahkan sebagian besar pesannya. Ia cukup memperhatikan beberapa jenis kata. Pesan yang mengandung *harga*, *berapa*, atau *how much* adalah soal harga. Pesan dengan *transfer*, *tf*, *bayar*, atau *QRIS* adalah soal pembayaran. *Antar*, *kirim*, *ongkir*, dan *deliver* soal pengantaran; *rusak*, *bocor*, *kecewa*, *broken*, dan *missing* adalah keluhan.

Setiap jenis kata kita sebut **sinyal**. Pengarah ini mengenal sepuluh sinyal: `price` (harga), `stock` (stok), `order` (pesanan), `payment` (pembayaran), `delivery` (pengantaran), `complaint` (keluhan), `hours` (jam buka), `late` (pesanan yang belum sampai), `proof` (bukti pembayaran), dan `link` (alamat web, biasanya sampah). Setiap sinyal hadir atau tidak hadir dalam sebuah pesan, sehingga setiap sinyal menjadi variabel VCIRS bernilai `yes` atau `no`. Daftar kata untuk setiap sinyal disimpan dalam berkas teks kecil yang dapat diubah keluarga itu tanpa memprogram. **Gambar 7.1** memperlihatkan seluruh perjalanan satu pesan.



Gambar 7.1 Perjalanan satu pesan melalui pengarah. Pendeteksi mengubah kata menjadi sinyal ya/tidak, tetapi hanya ketika basis pengetahuan menanyakannya; basis pengetahuan memilih rute; ketika Bu Rina membetulkan rute yang salah, sebuah pengecualian baru ditambahkan dan kondisi-kondisi induknya memperoleh Credit.

Dua keputusan rancangan perlu dikatakan terang-terangan. Pertama, pendeteksi tidak mencari kesepuluh sinyal sejak awal. Mesin VCIRS hanya menanyakan sebuah sinyal ketika node yang sedang diperiksanya memerlukannya, persis seperti dokter Wi-Fi hanya menanyakan kepada pelanggan pertanyaan yang diperlukannya. Di sini "orang" yang menjawab adalah pendeteksi, dan jawabannya tidak memakan biaya apa pun, tetapi basis pengetahuan yang sama dapat dipakai

oleh pembantu manusia yang mendengarkan pesan suara lalu menjawab pertanyaannya — dan bagi dia setiap pertanyaan yang dihemat adalah waktu yang dihemat. Kedua, pendeteksi sengaja dibuat sederhana dan jujur: ia mencocokkan kata utuh dan frasa pendek, tidak lebih. Ia akan membuat kesalahan yang tidak akan dibuat manusia, dan kita akan melihatnya. Inti VCIRS adalah bahwa kesalahan seperti itu dapat dibetulkan dalam basis pengetahuan, satu pesan demi satu pesan, oleh orang yang tahu jawaban yang benar.

7.2.2 Mengarahkan adalah mengklasifikasi

Memutuskan siapa yang menangani sebuah pesan adalah masalah klasifikasi, jenis masalah yang sama dengan yang dipecahkan dokter Wi-Fi: faktanya adalah sinyal-sinyal, dan konklusinya `route = owner` (pemilik), `route = cashier` (kasir), `route = courier` (kurir), `route = owner now` (keluhan, yang ingin segera dilihat Bu Rina), `route = auto reply` (balasan cepat berisi jam buka), atau `route = ignore` (abaikan). Bila tidak ada node yang memberi rute, pesan itu diserahkan kepada Bu Rina, yang membacanya sendiri; program menyebutnya *owner reads it*.

Bu Rina memulai dengan tujuh aturan, satu untuk setiap sinyal: pertanyaan harga ke pemilik, pertanyaan stok ke pemilik, pesanan ke pemilik, pembayaran ke kasir, pertanyaan pengantaran ke kurir, keluhan segera ke pemilik, dan pertanyaan jam buka mendapat balasan otomatis. Inilah "aturan yang benar hampir sepanjang waktu" dari Bab 3, dan persis seperti di sana, aturan-aturan ini terlalu umum. *Berapa ongkir ke Perum Griya Asri?* mengandung *berapa* (harga) dan *ongkir* (pengantaran); aturan pertama mengirimnya kepada pemilik, padahal ongkos kirim adalah urusan Pak Darto. Maka Bu Rina menambahkan pengecualian dalam konteks tempat kesalahan itu terjadi: di bawah aturan harga, *bila pengantaran juga disebut, arahkan ke kurir*. Inilah Ripple Down Rules [5] sebagaimana dijalankan mesin VCIRS, dengan satu penyempurnaan kecil yang dijelaskan Subbab 7.5: Bu Rina boleh menyebut sinyal yang membuat perbedaan, dan mesin lalu membangun pengecualian dari sinyal itu saja.

7.2.3 Credit: menghitung kesepakatan

Setiap kondisi yang tersimpan dalam sebuah node VCIRS membawa sebuah bilangan bulat yang disebut **Credit** [3], [6]. Ketika sebuah node dibuat, setiap kondisinya dimulai dengan Credit 1. Sesudah itu, Credit sebuah kondisi bertambah satu setiap kali seorang pakar membuat node baru *di bawah* node-nya — sebuah pengecualian — untuk kasus yang **sepakat** dengan kondisi itu (atau yang menyatakan ulang variabelnya). Tesisnya merumuskannya dalam satu kalimat: Credit "bertambah ketika pengguna membuat node yang sepakat dengan nilai-nilai kasus lama" [3].

Pikirkan apa artinya di toko. Aturan awal `price = yes → route = owner` dimulai dengan Credit 1. Ketika Bu Rina membetulkan *Berapa ongkir…?* dengan menambahkan pengecualian di bawahnya, pesan itu memang menyebut harga, jadi ia sepakat dengan kondisi `price = yes`, dan Credit-nya menjadi 2. Ketika ia membetulkan *Harga telur kok naik terus, kecewa saya* (harga dan keluhan), Credit menjadi 3; dan seterusnya. Credit yang tinggi menyatakan dua hal sekaligus: kondisi ini **benar untuk banyak kasus yang sampai ke node ini**, dan node ini adalah **tempat sang pakar terus menyempurnakan** pengetahuannya. Kondisinya banyak dipakai; node-nya sibuk.

Perhatikan apa yang *tidak* dihitung oleh Credit. Credit tidak menghitung berapa banyak pesan yang diarahkan oleh sebuah node. Aturan yang mengarahkan seratus pesan sehari dengan sempurna tidak pernah dikoreksi, dan Credit-nya tetap 1. Credit adalah tentang **struktur** pengetahuan, yang tumbuh karena diajari, bukan tentang lalu lintas pesan. Subbab 7.2.6 kembali ke perbedaan ini, karena sebuah toko memerlukan kedua jenis hitungan.

Dua sifat Credit lainnya perlu diketahui sebelum kita memakainya. Credit **tidak pernah turun**: VCIRS tidak pernah menghapus node, sehingga sebuah koreksi, sekali dibuat, tetap tercatat dalam hitungan, bahkan bila node itu kemudian dihentikan. Dan Credit bersifat **lokal**: Credit `price = yes` di aturan harga tidak menyatakan apa pun tentang kondisi `price = yes` di node lain, yang punya Credit-nya sendiri. Bayangkan setiap kondisi punya buku catatan kecilnya sendiri, tempat sang pakar membubuhkan satu tanda setiap kali ia menyempurnakan pengetahuan tepat di bawahnya untuk kasus yang sepakat dengannya. Setelah sebulan, buku catatan yang penuh tanda memperlihatkan ke mana perhatian sang pakar tercurah.

7.2.4 Bobot, kedekatan, dan tiga tingkat pemakaian

Credit adalah bahan mentahnya. VCIRS menggabungkannya dengan dua fakta lain tentang sebuah variabel untuk mengukur seberapa banyak variabel itu dipakai; penggabungan ini disebut **analisis nilai** (*value analysis*) atau **penugasan tingkat kegunaan** (*usage assignment*) dalam tesisnya [3], [4]. Ukuran untuk satu variabel di dalam satu node adalah **Variable Usage Rate**:

$$VUR_i = Credit_i \times Weight_i \tag{7.1}$$

Bobot (*weight*) sebuah variabel dalam sebuah node terdiri atas dua bagian. Yang pertama adalah **NS**, banyaknya node yang berbagi variabel itu: variabel yang dipakai di empat node lebih penting bagi basis pengetahuan daripada variabel yang dipakai di satu node saja. Yang kedua adalah **Closeness Degree** (derajat kedekatan) CD, yang menyatakan seberapa dekat variabel itu dengan konklusi node-nya:

$$Weight_i = NS_i \times CD_i \tag{7.2}$$

$$CD_i = \frac{VO_i}{TV} \tag{7.3}$$

Di sini VO_i adalah urutan variabel i dalam node itu (1 untuk kondisi pertama, 2 untuk yang kedua, dan seterusnya) dan TV adalah banyaknya seluruh variabel dalam node itu. Dalam node dengan dua kondisi, yang pertama berkedekatan $1/2$ dan yang kedua $2/2 = 1$. Gagasannya: sebuah node adalah *rangkaian* kondisi yang menuju konklusinya, dan kondisi yang diperiksa paling akhir adalah yang akhirnya menentukan; ialah yang paling dekat dengan konklusi [3].

Pemakaian sebuah node adalah rata-rata pemakaian variabel-variabelnya, dan pemakaian sebuah rule adalah rata-rata pemakaian node-node-nya. Dengan N variabel dalam node j dan M node dalam rule k:

$$NUR_j = \frac{1}{N}\sum_{i=1}^{N} VUR_{ij} \tag{7.4}$$

$$RUR_k = \frac{1}{M}\sum_{j=1}^{M} NUR_{jk} \tag{7.5}$$

Ingat dari Bab 3 bahwa sebuah **rule**, dalam struktur VCIRS, adalah lintasan node dari tingkat puncak turun sampai sebuah node: rule milik sebuah pengecualian memuat induknya dan dirinya sendiri. Jadi RUR mengukur pemakaian satu alur penalaran utuh, dan NUR mengukur satu langkahnya.

Contoh kecil membuat rumus-rumus ini nyata. Misalkan node harga `owner#1` punya satu kondisi, `price = yes`, dengan Credit 3, dan `price` tidak muncul di node lain. Maka $NS = 1$, $CD = 1/1 = 1$, bobotnya $1 \times 1 = 1$, dan menurut Persamaan (7.1) VUR-nya $3 \times 1 = 3$. Misalkan `delivery = yes` muncul di tiga node dan salah satunya `owner#2: order = yes, delivery = yes`. Di node itu `delivery` adalah kondisi kedua dari dua, jadi $CD = 2/2 = 1$, bobotnya menurut Persamaan (7.2) $3 \times 1 = 3$, dan dengan Credit 1 VUR-nya 3. `order` adalah kondisi pertama dari dua, jadi kedekatannya menurut Persamaan (7.3) $1/2$; bila ia tidak dipakai di node lain bobotnya $1 \times 1/2 = 1/2$, dan VUR-nya $1/2$. Menurut Persamaan (7.4) NUR `owner#2` adalah $(1/2 + 3)/2 = 1.75$. Subbab 7.3 menghitung angka-angka ini dengan sebuah program, lalu membiarkan mesin menghitungnya juga. Akhirnya, Persamaan (7.5) merata-ratakan NUR di sepanjang sebuah rule: pengecualian dengan NUR 1 di bawah node dengan NUR 3 menghasilkan rule dengan RUR $(3 + 1)/2 = 2$.

> **DEFINISI.** *Credit* sebuah kondisi menghitung berapa kali seorang pakar membuat node di bawah node-nya untuk kasus yang sepakat dengannya (ditambah satu untuk pembuatannya sendiri). *Variable Usage Rate* sebuah kondisi adalah Credit-nya dikali banyaknya node yang berbagi variabelnya dikali kedekatannya dengan konklusi; *Node Usage Rate* adalah rata-rata VUR kondisi-kondisi sebuah node; *Rule Usage Rate* adalah rata-rata NUR node-node sebuah rule. Makin besar berarti makin banyak dipakai.

7.2.5 Untuk apa angka-angka itu

Tesisnya memberi analisis nilai tiga tujuan [3]. Yang pertama adalah pedoman **ketika membangun** pengetahuan: rule, node, dan variabel mana yang dikunjungi lebih dulu, karena merekalah yang paling banyak dipakai. Yang kedua adalah pedoman **ketika berkonsultasi** dengan basis pengetahuan: variabel mana yang ditanyakan lebih dulu, dan bagian pengetahuan mana yang paling menarik bagi pengguna. Yang ketiga adalah pedoman untuk **pembangkitan rule**, pokok bahasan Bab 6, tempat tingkat pemakaian dan hitungan berbagi bersama-sama menentukan rule baru mana yang layak diusulkan.

Di samping tingkat pemakaian, VCIRS menyimpan ukuran yang lebih sederhana dari **analisis variabel**: berapa node yang berbagi setiap variabel (NS) dan berapa rule yang berbagi setiap node. Variabel yang dibagi oleh node terbanyak — paling sedikit dua — adalah **variabel terpenting**, dan node yang dibagi oleh rule terbanyak adalah node terpenting [3], [4]. Mesin memberikan keduanya, ditambah **urutan pertanyaan**: variabel-variabel yang diurutkan menurut VUR terbesarnya, yang paling banyak dipakai lebih dulu.

Bagi Toko Rina, semua ini menjadi pertanyaan yang sangat praktis. Sinyal mana yang paling penting, sehingga daftar katanya layak dirawat paling cermat? Aturan mana yang paling sering dikoreksi, sehingga Bu Rina harus membacanya lebih dulu ketika meninjau pengetahuannya? Dan ketika seorang pembantu mendengarkan pesan suara, apa yang harus ia dengarkan lebih dulu?

7.2.6 Dua jenis hitungan

Sebuah toko memerlukan jenis angka kedua yang tidak disimpan VCIRS dalam strukturnya: **lalu lintas**. Berapa pesan yang diterima setiap orang pagi ini? Berapa pesan yang diarahkan setiap node? Hitungan itu berasal dari menjalankan pengarah, bukan dari mengajarinya, dan menjawab pertanyaan yang berbeda: apakah Dinda kewalahan, apakah balasan otomatis layak dipertahankan, apakah ada aturan yang tidak pernah terpakai sama sekali.

Pengarah dalam bab ini mencetak keduanya. Credit dan tingkat pemakaian memberi tahu Bu Rina tentang **pengetahuannya**: di mana ia disempurnakan, sinyal mana yang diandalkannya. Hitungan

lalu lintas memberi tahu tentang **pelanggannya**: apa yang mereka tulis. Aturan dengan pemakaian tinggi dan lalu lintas rendah adalah aturan yang halus, banyak disempurnakan untuk pesan-pesan yang jarang; aturan dengan pemakaian rendah dan lalu lintas tinggi adalah aturan sederhana yang bekerja. Keduanya baik; keduanya memerlukan perhatian yang berbeda.

7.3 Tingkat Pemakaian dengan Tangan

Sebelum mempercayai angka-angka mesin, kita menghitungnya sendiri, seperti Bab 3 membangun Ripple Down Rules dari nol. **Kode 7.1** menggambarkan lima node sebagai data biasa — identitasnya, induknya, dan kondisi-kondisinya beserta Credit — menghitung NS, CD, VUR, NUR, dan RUR dengan Persamaan (7.1) sampai Persamaan (7.5), lalu membangun lima node yang sama dengan vcirs-core dan mencetak tabel-tabel mesin.

Kode 7.1 `UsageByHand.java`: tingkat pemakaian lima node dihitung dengan tangan, lalu oleh vcirs-core.

```
import java.util.ArrayList;
import java.util.List;

import vcirs.Case;
import vcirs.KnowledgeBase;
import vcirs.Num;
import vcirs.Report;

/**
 * Credit, VUR, NUR, dan RUR dihitung dengan tangan untuk lima node, lalu
 * oleh vcirs-core untuk lima node yang sama, untuk menunjukkan kesamaannya.
 */
public class UsageByHand {
    record Cond(String variable, int credit) {
    }

    record Node(String id, String parent, List<Cond> conds) {
    }

    // id node, induk, kondisi-kondisinya menurut urutan beserta Credit-nya
    static final List<Node> NODES = List.of(
            new Node("owner#1", null, List.of(new Cond("price", 3))),
            new Node("owner#2", null, List.of(new Cond("order", 1),
                    new Cond("delivery", 1))),
            new Node("courier#1", null, List.of(new Cond("delivery", 1))),
            new Node("courier#2", "owner#1", List.of(new Cond("delivery", 1))),
            new Node("owner now#1", "owner#1",
                    List.of(new Cond("complaint", 1))));

    /** Berapa node yang memakai variabel itu (NS). */
    static int ns(String variable) {
        int n = 0;
        for (Node node : NODES) {
            if (node.conds.stream().anyMatch(c -> c.variable.equals(variable))) {
                n++;
            }
        }
        return n;
    }

    static double vur(List<Cond> conds, int i) {
        Cond c = conds.get(i);
        double cd = (double) (i + 1) / conds.size();  // Closeness Degree VO/TV
        return c.credit * ns(c.variable) * cd;         // Credit x NS x CD
    }
```

Kode 7.1 (Lanjutan) *UsageByHand.java: tingkat pemakaian lima node dihitung dengan tangan, lalu oleh vcirs-core.*

```
    static double nur(List<Cond> conds) {
        double sum = 0;
        for (int i = 0; i < conds.size(); i++) {
            sum += vur(conds, i);
        }
        return sum / conds.size();
    }

    /** Node-node dari tingkat puncak turun sampai nodeId. */
    static List<Node> ruleOf(String nodeId) {
        List<Node> path = new ArrayList<>();
        while (nodeId != null) {
            String id = nodeId;
            Node node = NODES.stream().filter(n -> n.id.equals(id))
                    .findFirst().get();
            path.add(0, node);
            nodeId = node.parent;
        }
        return path;
    }

    static String pad(String s, int width) {
        return String.format("%-" + width + "s", s);
    }

    public static void main(String[] args) {
        System.out.println("By hand:");
        for (Node node : NODES) {
            List<String> rates = new ArrayList<>();
            for (int i = 0; i < node.conds.size(); i++) {
                rates.add(Num.fmt(vur(node.conds, i), 2));
            }
            List<Node> rule = ruleOf(node.id);
            double rur = 0;
            for (Node n : rule) {
                rur += nur(n.conds);
            }
            rur /= rule.size();
            System.out.println("  " + pad(node.id, 12) + " VUR "
                    + pad(String.join(", ", rates), 10) + " NUR "
                    + pad(Num.fmt(nur(node.conds), 2), 5) + " RUR "
                    + Num.fmt(rur, 2));
        }
```

Kode 7.1 (Lanjutan) *UsageByHand.java: tingkat pemakaian lima node dihitung dengan tangan, lalu oleh vcirs-core.*

```
        KnowledgeBase kb = new KnowledgeBase("five nodes");
        kb.refine(Case.of(List.of("price = yes"), List.of("route = owner")),
                KnowledgeBase.TOP_LEVEL, null);
        kb.refine(Case.of(List.of("order = yes", "delivery = yes"),
                List.of("route = owner")), KnowledgeBase.TOP_LEVEL, null);
        kb.refine(Case.of(List.of("delivery = yes"), List.of("route = courier")),
                KnowledgeBase.TOP_LEVEL, null);
        kb.refine(Case.of(List.of("price = yes", "delivery = yes"),
                List.of("route = courier")), KnowledgeBase.ADD_NEW, "owner#1");
        kb.refine(Case.of(List.of("price = yes", "complaint = yes"),
                List.of("route = owner now")), KnowledgeBase.ADD_NEW, "owner#1");
        System.out.println("By vcirs-core:");
        List<String> lines = new ArrayList<>(Report.vurTable(kb, 2));
        lines.addAll(Report.rurTable(kb, 2));
        for (String line : lines) {
            System.out.println("  " + line);
        }
    }
}
```

Kelima node itu adalah contoh Subbab 7.2.4 yang dilengkapi. `owner#1` mengarahkan pertanyaan harga dan punya dua pengecualian, `courier#2` (pertanyaan harga yang juga menyebut pengantaran) dan `owner now#1` (pertanyaan harga dengan keluhan), sehingga kondisinya `price = yes` ber-Credit 3. `owner#2` adalah node dengan dua kondisi, `order = yes, delivery = yes`, yang memperlihatkan Closeness Degree bekerja. `courier#1` mengarahkan pertanyaan pengantaran. `ns` menghitung node yang kondisinya memakai sebuah variabel; `vur` menerapkan Persamaan (7.3), Persamaan (7.2), dan Persamaan (7.1) dalam tiga baris; `nur` merata-ratakan (Persamaan (7.4)); `ruleOf` mendaki dari sebuah node ke tingkat puncak melalui induk-induknya, dan `main` merata-ratakan NUR di sepanjang lintasan itu (Persamaan (7.5)).

Paruh kedua membangun basis pengetahuan yang sama dengan `refine` milik mesin: tiga node tingkat puncak (opsi 5) dan dua pengecualian di bawah `owner#1` (opsi 2, bagian B daftar perbedaan). Kita tidak pernah mengisi Credit sendiri; mesinlah yang menaikkan Credit `price = yes` setiap kali pengecualian dibuat di bawah `owner#1` untuk kasus yang sepakat dengannya. **Kode 7.2** adalah program yang sama dalam Python.

Kode 7.2 `usage_by_hand.py`: perhitungan yang sama dalam Python.

```
"""Credit, VUR, NUR, dan RUR dihitung dengan tangan untuk lima node, lalu
oleh vcirs-core untuk lima node yang sama, untuk menunjukkan kesamaannya."""

import pathlib
import sys

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import ADD_NEW, TOP_LEVEL, Case, KnowledgeBase, num, report

# id node, induk, [(variabel, credit)] -- kondisi menurut urutannya
NODES = [
    ("owner#1", None, [("price", 3)]),
    ("owner#2", None, [("order", 1), ("delivery", 1)]),
    ("courier#1", None, [("delivery", 1)]),
    ("courier#2", "owner#1", [("delivery", 1)]),
    ("owner now#1", "owner#1", [("complaint", 1)]),
]


def ns(variable):
    """Berapa node yang memakai variabel itu (NS)."""
    return sum(1 for _, _, conds in NODES
               if any(v == variable for v, _ in conds))


def vur(conds, i):
    variable, credit = conds[i]
    cd = (i + 1) / len(conds)             # Closeness Degree = VO / TV
    return credit * ns(variable) * cd     # Credit x Weight, Weight = NS x CD


def nur(conds):
    return sum(vur(conds, i) for i in range(len(conds))) / len(conds)


def rule_of(node_id):
    """Node-node dari tingkat puncak turun sampai node_id."""
    path = []
    while node_id is not None:
        node = next(n for n in NODES if n[0] == node_id)
        path.insert(0, node)
        node_id = node[1]
    return path
```

Kode 7.2 (Lanjutan) *usage_by_hand.py: perhitungan yang sama dalam Python.*

```
print("By hand:")
for node_id, _, conds in NODES:
    rates = [num.fmt(vur(conds, i), 2) for i in range(len(conds))]
    rule = rule_of(node_id)
    rur = sum(nur(n[2]) for n in rule) / len(rule)
    print("  " + node_id.ljust(12) + " VUR " + ", ".join(rates).ljust(10)
          + " NUR " + num.fmt(nur(conds), 2).ljust(5)
          + " RUR " + num.fmt(rur, 2))

kb = KnowledgeBase("five nodes")
kb.refine(Case.of(["price = yes"], ["route = owner"]), TOP_LEVEL)
kb.refine(Case.of(["order = yes", "delivery = yes"], ["route = owner"]),
          TOP_LEVEL)
kb.refine(Case.of(["delivery = yes"], ["route = courier"]), TOP_LEVEL)
kb.refine(Case.of(["price = yes", "delivery = yes"], ["route = courier"]),
          ADD_NEW, "owner#1")
kb.refine(Case.of(["price = yes", "complaint = yes"],
                  ["route = owner now"]), ADD_NEW, "owner#1")
print("By vcirs-core:")
for line in report.vur_table(kb, 2) + report.rur_table(kb, 2):
    print("  " + line)
```

Keduanya mencetak:

```
By hand:
  owner#1      VUR 3          NUR 3     RUR 3
  owner#2      VUR 0.5, 3     NUR 1.75  RUR 1.75
  courier#1    VUR 3          NUR 3     RUR 3
  courier#2    VUR 3          NUR 3     RUR 3
  owner now#1  VUR 1          NUR 1     RUR 2
By vcirs-core:
  VariableID  NodeID       VariableOrder  VUR
  ----------  -----------  -------------  ----
  order       owner#2      1              0.50
  complaint   owner now#1  1              1.00
  price       owner#1      1              3.00
  delivery    owner#2      2              3.00
  delivery    courier#1    1              3.00
  delivery    courier#2    1              3.00
  RuleID       RUR
  -----------  ----
  owner#2      1.75
  owner now#1  2.00
  owner#1      3.00
  courier#1    3.00
  courier#2    3.00
```

Periksa beberapa baris terhadap Subbab 7.2.4. `owner#2` punya dua VUR, 0.5 untuk `order` (Credit 1, NS 1, kedekatan 1/2) dan 3 untuk `delivery` (Credit 1, NS 3, kedekatan 1); NUR-nya adalah rata-rata keduanya, 1.75, dan karena ia node tingkat puncak, rule-nya hanya memuat dirinya sendiri, sehingga RUR-nya juga 1.75. `owner now#1` ber-NUR 1 (Credit 1, `complaint` dipakai sekali), tetapi rule-nya adalah lintasan `owner#1 → owner now#1`, jadi RUR-nya $(3 + 1)/2 = 2$. Tabel-tabel mesin memuat angka yang sama, diurutkan dari yang terkecil sampai yang terbesar seperti dalam tesisnya, dan Credit-nya berasal dari pengajaran semata: dua panggilan `refine` di bawah `owner#1` menaikkan `price = yes` dari 1 menjadi 3.

> **WAWASAN KUNCI:** Tingkat pemakaian bukan data tambahan yang harus dirawat seseorang. Ia dihitung dari struktur yang memang dibangun sang pakar — kondisi mana ada di node mana, dalam urutan apa, dan seberapa sering kasus-kasus berikutnya sepakat dengannya. Mengajari basis pengetahuan *sama dengan* mengumpulkan statistiknya.

7.4 Pendeteksi Sinyal

Pendeteksi membaca daftar kata dari `data/signals.csv`, satu kata kunci atau frasa per baris, dengan nama sinyal di depannya:

```
# signal,keyword or phrase (lower case; Indonesian and English)
price,harga
price,berapa
price,brp
price,how much
...
payment,transfer
payment,tf
payment,transferan
...
late,belum sampai
late,belum datang
late,not arrived
...
link,http
```

Berkas itu memuat 81 kata kunci dan frasa untuk sepuluh sinyal. Isinya campuran bahasa Indonesia, bahasa Inggris, dan singkatan khas obrolan (*brp* untuk *berapa*, *tf* untuk *transfer*), karena begitulah pelanggan menulis. **Kode 7.3** adalah pendeteksinya.

Kode 7.3 `Signals.java`: pendeteksi sinyal membaca daftar kata dan menemukan kata utuh serta frasa dalam sebuah pesan.

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;

/**
 * Sinyal: jenis-jenis kata yang dikandung pesan pelanggan. Sebuah sinyal
 * adalah nama seperti "price" dengan daftar kata kunci dan frasa, dalam
 * bahasa Indonesia dan Inggris, dibaca dari data/signals.csv. Pendeteksi
 * hanya menemukan kata; ia tidak memahami pesannya.
 */
public class Signals {
    private final Map<String, List<List<String>>> keywords =
            new LinkedHashMap<>();

    /** "Harganya BERAPA, kak?" -> [harganya, berapa, kak] */
    public static List<String> words(String text) {
        StringBuilder clean = new StringBuilder();
        for (char ch : text.toLowerCase(Locale.ROOT).toCharArray()) {
            boolean keep = (ch >= 'a' && ch <= 'z') || (ch >= '0' && ch <= '9');
            clean.append(keep ? ch : ' ');
        }
        String s = clean.toString().strip();
        return s.isEmpty() ? List.of() : Arrays.asList(s.split(" +"));
    }

    public Signals(String path) {
        List<String> lines;
        try {
            lines = Files.readAllLines(Path.of(path), StandardCharsets.UTF_8);
        } catch (IOException e) {
            throw new IllegalStateException("cannot read " + path);
        }
        for (String line : lines) {
            line = line.strip();
            if (line.isEmpty() || line.startsWith("#")) {
                continue;
            }
            int comma = line.indexOf(',');
            keywords.computeIfAbsent(line.substring(0, comma),
```

Kode 7.3 (Lanjutan) *Signals.java: pendeteksi sinyal membaca daftar kata dan menemukan kata utuh serta frasa dalam sebuah pesan.*

```
                    k -> new ArrayList<>()).add(words(line.substring(comma + 1)));
        }
    }

    public List<String> names() {
        return new ArrayList<>(keywords.keySet());
    }

    /** Benar bila salah satu frasa sinyal itu muncul dalam pesan. */
    public boolean found(String text, String signal) {
        List<String> tokens = words(text);
        for (List<String> phrase : keywords.getOrDefault(signal, List.of())) {
            int n = phrase.size();
            for (int i = 0; i + n <= tokens.size(); i++) {
                if (tokens.subList(i, i + n).equals(phrase)) {
                    return true;
                }
            }
        }
        return false;
    }

    public List<String> allFound(String text) {
        List<String> out = new ArrayList<>();
        for (String s : keywords.keySet()) {
            if (found(text, s)) {
                out.add(s);
            }
        }
        return out;
    }
}
```

`words` mengubah pesan menjadi kata-kata berhuruf kecil: setiap karakter yang bukan huruf a sampai z atau angka menjadi spasi, lalu hasilnya dipecah pada spasi. Jadi *Harganya BERAPA, kak?* menjadi `harganya`, `berapa`, `kak`. Konstruktor membaca berkas dan menyimpan, untuk setiap sinyal, frasa-frasanya sebagai daftar kata. `found` mencari salah satu frasa sinyal sebagai kata-kata yang berurutan dalam pesan: `how much` cocok dengan *How much is a 12 kg LPG refill?*, tetapi `stock` tidak cocok dengan *stocking*, karena hanya kata utuh yang dihitung. `allFound` mendaftar semua sinyal yang hadir, menurut urutan dalam berkas; pengarah memakainya ketika harus mengingat sebuah pesan sebagai kasus. **Kode 7.4** adalah versi Python-nya.

Kode 7.4 `signals.py`: pendeteksi yang sama dalam Python.

```
"""Sinyal: jenis-jenis kata yang dikandung pesan pelanggan.

Sebuah sinyal adalah nama seperti "price" dengan daftar kata kunci dan frasa,
dalam bahasa Indonesia dan Inggris, dibaca dari data/signals.csv. Pendeteksi
hanya menemukan kata; ia tidak memahami pesannya.
"""


def words(text):
    """'Harganya BERAPA, kak?' -> ['harganya', 'berapa', 'kak']"""
    clean = "".join(ch if "a" <= ch <= "z" or "0" <= ch <= "9" else " "
                    for ch in text.lower())
    return clean.split()


class Signals:
    def __init__(self, path):
        self.keywords = {}                      # sinyal -> daftar frasa
        with open(path, encoding="utf-8") as f:
            for line in f:
                line = line.strip()
                if not line or line.startswith("#"):
                    continue
                signal, phrase = line.split(",", 1)
                self.keywords.setdefault(signal, []).append(words(phrase))

    def names(self):
        return list(self.keywords)

    def found(self, text, signal):
        """Benar bila salah satu frasa sinyal itu muncul dalam pesan."""
        tokens = words(text)
        for phrase in self.keywords.get(signal, []):
            n = len(phrase)
            for i in range(len(tokens) - n + 1):
                if tokens[i:i + n] == phrase:
                    return True
        return False

    def all_found(self, text):
        return [s for s in self.keywords if self.found(text, s)]
```

Mencocokkan kata utuh ada harganya, dan keluarga itu akan segera merasakannya: *kurirnya* adalah kata yang berbeda dari *kurir*, dan *pesanan* berbeda dari *pesan*. Itu sebabnya berkas mencantumkan sebagian kata dalam beberapa bentuk. Tetapi ada pula untungnya: pendeteksi tidak pernah menemukan *tf* di dalam *gift* atau *ada* di dalam *adalah*, dan setiap keputusannya dapat dijelaskan dengan menunjuk sebuah kata.

7.5 Pengarah di Atas vcirs-core

Kode 7.5 memuat tiga hal yang dikerjakan aplikasi dengan basis pengetahuan: membangunnya dari aturan-aturan awal Bu Rina, mengarahkan sebuah pesan, dan belajar dari sebuah koreksi.

Kode 7.5 `Router.java`: membangun basis pengetahuan, mengarahkan pesan, dan mengajarkan koreksi.

```
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import vcirs.Case;
import vcirs.CaseFile;
import vcirs.Condition;
import vcirs.Conclusion;
import vcirs.Fact;
import vcirs.Facts;
import vcirs.InferenceResult;
import vcirs.KnowledgeBase;
import vcirs.Node;
import vcirs.Question;

/** Pengarah pesan toko di atas vcirs-core: arahkan, ajari, hitung. */
public class Router {
    public static final String NOBODY = "owner reads it";  // tak ada node pengarah
    static final String[][] QUESTIONS = {
        {"price", "Does the message ask about a price?"},
        {"stock", "Does it ask whether something is in stock?"},
        {"order", "Does the customer want to order or buy?"},
        {"payment", "Is it about a payment or a transfer?"},
        {"delivery", "Is it about delivery or a delivery fee?"},
        {"complaint", "Is the customer complaining?"},
        {"hours", "Does it ask when the shop is open?"},
        {"late", "Does the customer say an order has not arrived?"},
        {"proof", "Does it send or mention a proof of payment?"},
        {"link", "Does it contain a web link?"},
    };

    /** Basis pengetahuan dengan tujuh aturan pertama Bu Rina. */
    public static KnowledgeBase newKb(String data) {
        CaseFile seed = CaseFile.read(Path.of(data, "seed-cases.json")
                .toString());
        KnowledgeBase kb = new KnowledgeBase(seed.name, seed.description);
        for (String[] q : QUESTIONS) {
            kb.declare(q[0], q[1], List.of("yes", "no"));
        }
        return seed.build(kb);
    }
```

Kode 7.5 (Lanjutan) *Router.java: membangun basis pengetahuan, mengarahkan pesan, dan mengajarkan koreksi.*

```
    /** Penanya bagi mesin: "apakah pesan mengandung sinyal ini?" */
    static class Detector implements vcirs.Asker {
        final Signals signals;
        final String text;
        final List<String> checked = new ArrayList<>();

        Detector(Signals signals, String text) {
            this.signals = signals;
            this.text = text;
        }

        public Fact ask(Question q) {
            checked.add(q.variable);
            return new Fact(q.variable,
                    signals.found(text, q.variable) ? "yes" : "no");
        }
    }

    /** Rute, hasil mesin, dan sinyal-sinyal yang diperiksa. */
    public static class Routed {
        public String route;
        public InferenceResult result;
        public List<String> checked;
    }

    /**
     * Arahkan sebuah pesan. Dengan order, sinyal diperiksa menurut urutan
     * itu sampai satu ditemukan; mesin mengerjakan sisanya.
     */
    public static Routed route(KnowledgeBase kb, Signals signals, String text,
                               List<String> order) {
        Detector det = new Detector(signals, text);
        Map<String, Fact> facts = Facts.none();
        for (String signal : order) {
            det.checked.add(signal);
            boolean yes = signals.found(text, signal);
            facts.put(signal, new Fact(signal, yes ? "yes" : "no"));
            if (yes) {
                break;
            }
        }
        Routed r = new Routed();
        r.result = kb.infer(facts, det);
        List<Conclusion> found = r.result.conclusions();
        r.route = found.isEmpty() ? NOBODY : found.get(0).value;
        r.checked = det.checked;
```

Kode 7.5 (Lanjutan) *Router.java: membangun basis pengetahuan, mengarahkan pesan, dan mengajarkan koreksi.*

```
        return r;
    }

    public static Routed route(KnowledgeBase kb, Signals signals, String text) {
        return route(kb, signals, text, List.of());
    }

    /** Apa yang dilakukan teach() terhadap satu pesan. */
    public static class Taught {
        public String routed;
        public String fired;                    // node pengarahnya, atau null
        public Node node;                       // node baru, atau null
        public List<String> affected = List.of();
    }

    /** Arahkan satu pesan; bila rutenya salah, pelajari yang benar. */
    public static Taught teach(KnowledgeBase kb, Signals signals, String text,
                               String right, String because, String author) {
        Routed r = route(kb, signals, text);
        Taught t = new Taught();
        t.routed = r.route;
        t.fired = r.result.paths.isEmpty() ? null
                : r.result.paths.get(0).last().id;
        if (r.route.equals(right)) {
            kb.refine(new Case(), KnowledgeBase.ACCEPT, t.fired, null, author);
            return t;
        }
        List<Condition> facts = new ArrayList<>();
        for (String s : signals.allFound(text)) {
            facts.add(new Condition(s, "=", "yes"));
        }
        Case c = new Case(facts, List.of(new Conclusion("route", right)), 100);
        List<Condition> chosen = because == null ? null
                : List.of(new Condition(because, "=", "yes"));
        int option = t.fired != null ? KnowledgeBase.ADD_NEW
                : KnowledgeBase.TOP_LEVEL;
        List<Condition> conds = chosen != null ? chosen
                : t.fired != null ? kb.differences(t.fired, c).b : c.facts;
        t.affected = kb.affectedCornerstones(t.fired, conds, c.conclusions);
        t.node = kb.refine(c, option, t.fired, chosen, author);
        return t;
    }
}
```

Membangun. `newKb` membaca `seed-cases.json`, yang menyimpan tujuh aturan pertama Bu Rina sebagai tujuh kasus dengan opsi 5 (masing-masing node tingkat puncak baru), dan mendeklarasikan sepuluh sinyal beserta pertanyaan yang akan diajukan kepada pembantu manusia (`QUESTIONS`). Urutan kasus awal itu penting, seperti urutan node tingkat puncak dalam RDR selalu penting: aturan harga lebih dulu, lalu stok, pesanan, pembayaran, pengantaran, keluhan, dan jam buka. Karena tiga di antaranya menyimpulkan `route = owner`, mesin menamainya `owner#1`, `owner#2`, dan `owner#3` — satu identitas per node, dinomori menurut urutan pembuatannya.

Mengarahkan. `Detector` adalah **penanya** (*asker*) bagi mesin: ketika mesin memerlukan sinyal yang belum diketahuinya, ia memanggil `ask`, dan pendeteksi menjawab `yes` atau `no` dari kata-kata pesan sambil mencatat sinyal itu di `checked`. `route` menjalankan mesin tanpa fakta sama sekali, sehingga mesin menelusuri node tingkat puncak menurut urutannya dan menanyakan persis apa yang diperlukannya. Parameter opsional `order` memeriksa beberapa sinyal *sebelum* mesin mulai,

menurut urutan tertentu, sampai satu ditemukan; Subbab 7.9 memakainya untuk menguji urutan pertanyaan dari analisis nilai. Bila tidak ada node yang menyala, rutenya `owner reads it`.

Mengajari. `teach` lebih dulu mengarahkan pesan itu. Bila rutenya benar, ia mencatat pemeriksaan itu dengan opsi 0, *accept*, sehingga riwayat basis pengetahuan menunjukkan bahwa node itu dikonfirmasi oleh pesan sungguhan. Bila salah, ia membangun kasus dari sinyal-sinyal yang ditemukan dalam pesan (hanya yang hadir, semuanya `= yes`) dengan rute yang benar sebagai konklusinya. Bila Bu Rina menyebut sinyal yang membuat perbedaan, sinyal itu saja yang menjadi kondisi (kondisi `chosen` milik mesin); bila tidak, node baru mengambil bagian B daftar perbedaan, seperti di Bab 3 [7]. Node baru digantung di bawah node yang memberi rute salah (opsi 2), atau di tingkat puncak bila tidak ada node yang mengarahkan pesan itu (opsi 5). Sebelum mengubah apa pun, `teach` menanyakan kepada mesin **cornerstone yang terdampak** oleh node baru, yaitu kasus-kasus terdahulu yang rutenya akan berubah.

Mengapa hanya sinyal yang hadir? Karena sebuah pesan digambarkan dengan baik oleh apa yang dikandungnya, dan dengan buruk oleh daftar panjang hal yang tidak dikandungnya. Node yang dibangun dari `payment = yes, complaint = yes` terbaca seperti penalaran Bu Rina; node yang dibangun dari `payment = yes, complaint = yes, price = no, stock = no, order = no, …` hanya akan menyala untuk pesan yang menghindari tujuh sinyal, dan bukan itu yang dimaksudkannya. Kondisi `chosen` milik mesin memungkinkan aplikasi mengatakan persis apa yang dimaksud sang pakar. **Kode 7.6** adalah versi Python-nya.

Kode 7.6 `router.py`: tiga operasi yang sama dalam Python.

```
"""Pengarah pesan toko di atas vcirs-core: arahkan, ajari, hitung."""

import os
import pathlib
import sys

# Beri tahu Python letak mesin vcirs-core (naik dua folder, lalu masuk).
sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import (ACCEPT, ADD_NEW, TOP_LEVEL, Case, Condition, Conclusion,
                   KnowledgeBase, casefile)

NOBODY = "owner reads it"          # bila tak ada node yang mengarahkan
QUESTIONS = [
    ("price", "Does the message ask about a price?"),
    ("stock", "Does it ask whether something is in stock?"),
    ("order", "Does the customer want to order or buy?"),
    ("payment", "Is it about a payment or a transfer?"),
    ("delivery", "Is it about delivery or a delivery fee?"),
    ("complaint", "Is the customer complaining?"),
    ("hours", "Does it ask when the shop is open?"),
    ("late", "Does the customer say an order has not arrived?"),
    ("proof", "Does it send or mention a proof of payment?"),
    ("link", "Does it contain a web link?"),
]


def new_kb(data):
    """Basis pengetahuan dengan tujuh aturan pertama Bu Rina."""
    name, description, steps = casefile.read_cases(
        os.path.join(data, "seed-cases.json"))
    kb = KnowledgeBase(name, description)
    for var, question in QUESTIONS:
        kb.declare(var, question, ["yes", "no"])
    return casefile.build(kb, steps)


class Detector:
    """Penanya bagi mesin: 'apakah pesan mengandung sinyal ini?'"""

    def __init__(self, signals, text):
        self.signals = signals
        self.text = text
        self.checked = []
```

Kode 7.6 (Lanjutan) *router.py: tiga operasi yang sama dalam Python.*

```
    def __call__(self, question):
        self.checked.append(question.variable)
        return "yes" if self.signals.found(self.text, question.variable) \
            else "no"


def route(kb, signals, text, order=()):
    """(rute, hasil, sinyal yang diperiksa). Dengan `order`, sinyal diperiksa
    menurut urutan itu sampai satu ditemukan; mesin mengerjakan sisanya."""
    det = Detector(signals, text)
    facts = {}
    for signal in order:
        det.checked.append(signal)
        facts[signal] = "yes" if signals.found(text, signal) else "no"
        if facts[signal] == "yes":
            break
    result = kb.infer(facts, det)
    found = result.conclusions()
    return (found[0].value if found else NOBODY), result, det.checked


def teach(kb, signals, text, right, because, author):
    """Arahkan satu pesan; bila rutenya salah, pelajari yang benar. Hasilnya
    (rute, id node yang menyala atau None, node baru atau None, terdampak)."""
    routed, result, _ = route(kb, signals, text)
    fired = result.paths[0].last().id if result.paths else None
    if routed == right:
        kb.refine(Case(), ACCEPT, fired, author=author)
        return routed, fired, None, []
    case = Case([Condition(s, "=", "yes") for s in signals.all_found(text)],
                [Conclusion("route", right)])
    chosen = [Condition(because, "=", "yes")] if because else None
    option = ADD_NEW if fired else TOP_LEVEL
    conds = chosen if chosen else (kb.differences(fired, case).b if fired
                                   else case.facts)
    affected = kb.affected_cornerstones(fired, conds, case.conclusions)
    node = kb.refine(case, option, fired, chosen, author)
    return routed, fired, node, affected
```

7.6 Aplikasinya

Kode 7.7 adalah seluruh aplikasinya. Ia punya lima perintah, satu untuk setiap pekerjaan di toko.

Kode 7.7 `ShopRouter.java`: pengarah pesan Toko Rina dengan lima perintahnya.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

import vcirs.Condition;
import vcirs.Fact;
import vcirs.Facts;
import vcirs.InferenceResult;
import vcirs.KnowledgeBase;
import vcirs.Node;
import vcirs.Num;
import vcirs.Report;
import vcirs.Usage;

/**
 * Pengarah pesan Toko Rina: mengirim setiap pesan WhatsApp ke orang yang
 * tepat dan belajar dari koreksi Bu Rina.
 *   java ShopRouter teach [file ...]  bangun ulang basis pengetahuan dari
 *                                     aturan awal lalu ajari
 *   java ShopRouter stats             Credit, VUR, NUR, RUR, urutan
 *   java ShopRouter inbox             arahkan pesan-pesan hari Senin
 *   java ShopRouter route "text"      arahkan satu pesan, jelaskan
 *   java ShopRouter voice [--echo]    seseorang menjawab pertanyaannya
 */
public class ShopRouter {
    static final String DATA = "../data";
    static final String KB_FILE = DATA + "/shop-kb.json";
```

Kode 7.7 (Lanjutan) *ShopRouter.java: pengarah pesan Toko Rina dengan lima perintahnya.*

```
    /** [pesan, rute benar, sinyal atau null] dari berkas data. */
    static List<String[]> labelled(String name) {
        List<String[]> out = new ArrayList<>();
        try {
            for (String line : Files.readAllLines(Path.of(DATA, name),
                    StandardCharsets.UTF_8)) {
                if (line.isBlank() || line.startsWith("#")) {
                    continue;
                }
                String[] p = line.split("\\|");
                out.add(new String[] {p[0].strip(), p[1].strip(),
                    p.length > 2 ? p[2].strip() : null});
            }
        } catch (IOException e) {
            throw new IllegalStateException("cannot read " + name);
        }
        return out;
    }

    static String rjust(int n, int width) {
        return String.format("%" + width + "d", n);
    }

    static String ljust(String s, int width) {
        return String.format("%-" + width + "s", s);
    }

    static void teach(Signals signals, List<String> files) {
        KnowledgeBase kb = Router.newKb(DATA);
        System.out.println("Seed rules: " + kb.summary());
        List<String[]> seen = new ArrayList<>();
        int right = 0;
        for (String name : files) {
            System.out.println("Teaching " + name + ":");
            for (String[] m : labelled(name)) {
                seen.add(m);
                Router.Taught t = Router.teach(kb, signals, m[0], m[1], m[2],
                        "Rina");
                System.out.println(rjust(seen.size(), 2) + ". " + m[0]);
                List<String> found = signals.allFound(m[0]);
                System.out.println("    signals: " + (found.isEmpty()
                        ? "(none)" : String.join(", ", found)));
                String by = t.fired != null ? " (" + t.fired + ")" : "";
                if (t.node == null) {
                    right++;
                    System.out.println("    routed to " + t.routed + by
```

Kode 7.7 (Lanjutan) *ShopRouter.java: pengarah pesan Toko Rina dengan lima perintahnya.*

```
                        + ": right");
                continue;
            }
            System.out.println("    routed to " + t.routed + " by
                    + ": WRONG, Rina says " + m[1]);
            String where = t.fired != null ? "exception of " + t.fired
                    : "top level";
            System.out.println("    learned " + t.node + " (" + where
                    + ")");
            System.out.println("    earlier cases whose route would "
                    + "change: " + (t.affected.isEmpty() ? "none"
                    : String.join(", ", t.affected)));
        }
    }
    System.out.println("Right the first time: " + right + " of "
            + seen.size());
    int still = 0;
    for (String[] m : seen) {
        if (Router.route(kb, signals, m[0]).route.equals(m[1])) {
            still++;
        }
    }
    System.out.println("All taught messages again: " + still + " of "
            + seen.size() + " right");
    kb.save(KB_FILE);
    System.out.println("Saved shop-kb.json: " + kb.summary());
}

static void stats(KnowledgeBase kb) {
    System.out.println("Credits of every condition:");
    List<List<Object>> rows = new ArrayList<>();
    for (Node n : kb.nodes.values()) {
        for (Condition c : n.conditions) {
            rows.add(List.of(n.id, c.toString(), c.credit));
        }
    }
    for (String line : Report.table(List.of("NodeID", "Condition",
            "Credit"), rows)) {
        System.out.println("  " + line);
    }
    String[] titles = {"Variable Usage Rates:", "Node Usage Rates:",
        "Rule Usage Rates:"};
    List<List<String>> tables = List.of(Report.vurTable(kb, 2),
            Report.nurTable(kb, 2), Report.rurTable(kb, 2));
    for (int i = 0; i < 3; i++) {
        System.out.println(titles[i]);
```

Kode 7.7 (Lanjutan) *ShopRouter.java: pengarah pesan Toko Rina dengan lima perintahnya.*

```
            for (String line : tables.get(i)) {
                System.out.println("  " + line);
            }
        }
        Usage usage = kb.usage();
        List<String> important = new ArrayList<>();
        for (String v : usage.importantVariables()) {
            important.add(v + " (NS " + usage.ns(v) + ")");
        }
        System.out.println("Most important variables: "
                + String.join(", ", important));
        System.out.println("Question order: "
                + String.join(", ", usage.questionOrder()));
    }

    static void count(Map<String, Integer> table, String key) {
        table.merge(key, 1, Integer::sum);
    }

    static void showCounts(String title, Map<String, Integer> table) {
        System.out.println(title);
        List<String> keys = new ArrayList<>(table.keySet());
        keys.sort((a, b) -> table.get(a).equals(table.get(b))
                ? a.compareTo(b) : table.get(b) - table.get(a));
        for (String k : keys) {
            System.out.println("  " + ljust(k, 15) + table.get(k));
        }
    }

    static void inbox(KnowledgeBase kb, Signals signals) {
        Map<String, Integer> perRoute = new LinkedHashMap<>();
        Map<String, Integer> perNode = new LinkedHashMap<>();
        List<String> wrong = new ArrayList<>();
        int[] checked = {0, 0};
        List<String> order = kb.usage().questionOrder();
        List<String[]> messages = labelled("inbox.txt");
        for (int i = 0; i < messages.size(); i++) {
            String text = messages.get(i)[0];
            String good = messages.get(i)[1];
            Router.Routed r = Router.route(kb, signals, text);
            checked[0] += r.checked.size();
            Router.Routed other = Router.route(kb, signals, text, order);
            checked[1] += other.checked.size();
            boolean ok = r.route.equals(good);
            if (!ok) {
                wrong.add(String.valueOf(i + 1));
```

Kode 7.7 (Lanjutan) *ShopRouter.java: pengarah pesan Toko Rina dengan lima perintahnya.*

```
            }
            System.out.println(rjust(i + 1, 2) + ". " + (ok ? "  " : "X ")
                    + ljust(r.route, 15)
                    + text.substring(0, Math.min(48, text.length())));
            count(perRoute, r.route);
            count(perNode, r.result.paths.isEmpty() ? "(no node)"
                    : r.result.paths.get(0).last().id);
            if (!other.route.equals(r.route)) {
                System.out.println("       (important-first order would say "
                        + other.route + ")");
            }
        }
        int n = messages.size();
        System.out.println("Right: " + (n - wrong.size()) + " of " + n
                + (wrong.isEmpty() ? "" : "; wrong: " + String.join(", ",
                wrong)));
        showCounts("Messages per person:", perRoute);
        showCounts("Messages per node:", perNode);
        System.out.println("Signals checked per message: "
                + Num.fmt((double) checked[0] / n, 2) + " in the engine's "
                + "order, " + Num.fmt((double) checked[1] / n, 2)
                + " important-first");
    }

    static void route(KnowledgeBase kb, Signals signals, String text) {
        Router.Routed r = Router.route(kb, signals, text);
        System.out.println("Message: " + text);
        List<String> seen = new ArrayList<>();
        for (String v : r.checked) {
            seen.add(v + " " + r.result.memory.facts.get(v).value);
        }
        System.out.println("Signals checked: " + String.join(", ", seen));
        System.out.println("Route: " + r.route);
        if (r.route.equals(Router.NOBODY)) {
            System.out.println("Why: no node matched, so Bu Rina reads it "
                    + "herself.");
            return;
        }
        System.out.println("Why:");
        for (String line : r.result.how()) {
            System.out.println("  " + line);
        }
    }
```

Kode 7.7 (Lanjutan) *ShopRouter.java: pengarah pesan Toko Rina dengan lima perintahnya.*

```
    static void voice(KnowledgeBase kb, boolean echo) {
        BufferedReader in = new BufferedReader(
                new InputStreamReader(System.in, StandardCharsets.UTF_8));
        List<String> listen = kb.usage().questionOrder().subList(0, 3);
        System.out.println("Voice note or photo? Listen above all for: "
                + String.join(", ", listen));
        InferenceResult result = kb.infer(Facts.none(), q -> {
            System.out.print(kb.variables.get(q.variable).question
                    + " (yes/no) > ");
            String line;
            try {
                line = in.readLine();
            } catch (IOException e) {
                line = null;
            }
            String answer = line == null ? "" : line.strip();
            if (echo) {
                System.out.println(answer);
            }
            return answer.isEmpty() ? null : new Fact(q.variable, answer);
        });
        System.out.println("Route: " + (result.conclusions().isEmpty()
                ? Router.NOBODY : result.conclusions().get(0).value));
    }

    public static void main(String[] args) {
        Signals signals = new Signals(DATA + "/signals.csv");
        String command = args.length > 0 ? args[0] : "inbox";
        if (command.equals("teach")) {
            List<String> files = args.length > 1
                    ? Arrays.asList(args).subList(1, args.length)
                    : List.of("teach-week.txt");
            teach(signals, files);
            return;
        }
        KnowledgeBase kb = Files.exists(Path.of(KB_FILE))
                ? KnowledgeBase.load(KB_FILE) : Router.newKb(DATA);
        if (command.equals("stats")) {
            stats(kb);
        } else if (command.equals("route")) {
            route(kb, signals, String.join(" ",
                    Arrays.asList(args).subList(1, args.length)));
        } else if (command.equals("voice")) {
            voice(kb, Arrays.asList(args).contains("--echo"));
        } else {
            inbox(kb, signals);
```

Kode 7.7 (Lanjutan) *ShopRouter.java: pengarah pesan Toko Rina dengan lima perintahnya.*

```
        }
    }
}
```

- `teach [file …]` selalu mulai lagi dari aturan-aturan awal, lalu membaca satu atau lebih berkas pesan berlabel — pesannya, rute yang dipilih Bu Rina, dan bila ada sinyal yang membuat perbedaan, dipisahkan dengan `|` — dan mengajarkannya satu per satu sambil mencetak apa yang terjadi. Di akhir, ia mengarahkan lagi setiap pesan yang sudah diajarkan, untuk membuktikan bahwa tidak ada yang dipelajari belakangan merusak yang dipelajari lebih dulu, lalu menyimpan `shop-kb.json`. Mulai dari aturan awal setiap kali membuat hasilnya dapat diulang persis; dalam pemakaian sehari-hari Anda menyimpan berkas itu dan hanya mengajarkan pesan-pesan baru.

- `stats` mencetak Credit setiap kondisi, tiga tabel tingkat pemakaian dari tesis (VUR, NUR, dan RUR, dari yang terkecil), variabel-variabel terpenting beserta NS-nya, dan urutan pertanyaan.
- `inbox` mengarahkan berkas pesan berlabel yang *belum* diajarkan, menandai yang salah dengan `X`, menghitung pesan per orang dan per node (lalu lintas di Subbab 7.2.6), dan membandingkan banyaknya sinyal yang diperiksa menurut urutan mesin dengan banyaknya yang diperiksa bila sinyal terpenting dicari lebih dulu.
- `route "text"` mengarahkan satu pesan dan menjelaskan keputusannya dengan penjelasan How dari mesin.
- `voice` untuk pesan yang tidak dapat dibaca pendeteksi, seperti pesan suara dan foto: pembantu menjawab sendiri pertanyaan-pertanyaan mesin, setelah aplikasi mengingatkannya sinyal mana yang paling penting.

Kode 7.8 adalah versi Python-nya. Fungsi bantunya, `count` dan `show_counts`, sama dengan yang di Java, termasuk urutan pengurutannya (pesan terbanyak dulu, lalu menurut abjad), sehingga kedua versi mencetak baris-baris yang sama.

Kode 7.8 `shop_router.py`: aplikasi yang sama dalam Python.

```
"""Pengarah pesan Toko Rina: mengirim setiap pesan WhatsApp ke orang yang
tepat dan belajar dari koreksi Bu Rina.

    python shop_router.py teach [file ...]  bangun ulang basis pengetahuan
                                            dari aturan awal lalu ajari
    python shop_router.py stats             Credit, VUR, NUR, RUR, urutan
    python shop_router.py inbox             arahkan pesan hari Senin
    python shop_router.py route "text"      arahkan satu pesan, jelaskan
    python shop_router.py voice [--echo]    seseorang menjawab pertanyaannya
"""

import os
import sys

from router import NOBODY, new_kb, route, teach
from signals import Signals
from vcirs import KnowledgeBase, num, report

DATA = os.path.join(os.path.dirname(os.path.abspath(__file__)), "..", "data")
KB_FILE = os.path.join(DATA, "shop-kb.json")


def labelled(name):
    """[(pesan, rute benar, sinyal atau None)] dari berkas data."""
    out = []
    with open(os.path.join(DATA, name), encoding="utf-8") as f:
        for line in f:
            if line.strip() and not line.startswith("#"):
                parts = [p.strip() for p in line.split("|")]
                out.append((parts[0], parts[1],
                            parts[2] if len(parts) > 2 else None))
    return out


def cmd_teach(signals, files):
    kb = new_kb(DATA)
    print("Seed rules: " + kb.summary())
    seen, right = [], 0
    for name in files:
        print("Teaching " + name + ":")
        for text, good, because in labelled(name):
            seen.append((text, good))
            routed, fired, node, hit = teach(kb, signals, text, good,
                                             because, "Rina")
            print(str(len(seen)).rjust(2) + ". " + text)
            print("    signals: " + (", ".join(signals.all_found(text))
```

Kode 7.8 (Lanjutan) *shop_router.py: aplikasi yang sama dalam Python.*

```
                                    or "(none)"))
            by = " (" + fired + ")" if fired else ""
            if node is None:
                right += 1
                print("    routed to " + routed + by + ": right")
                continue
            print("    routed to " + routed + by + ": WRONG, Rina says "
                  + good)
            where = "exception of " + fired if fired else "top level"
            print("    learned " + str(node) + " (" + where + ")")
            print("    earlier cases whose route would change: "
                  + (", ".join(hit) if hit else "none"))
    print("Right the first time: " + str(right) + " of " + str(len(seen)))
    still = sum(1 for text, good in seen
                if route(kb, signals, text)[0] == good)
    print("All taught messages again: " + str(still) + " of "
          + str(len(seen)) + " right")
    kb.save(KB_FILE)
    print("Saved shop-kb.json: " + kb.summary())


def cmd_stats(kb):
    print("Credits of every condition:")
    rows = [[n.id, str(c), c.credit] for n in kb.nodes.values()
            for c in n.conditions]
    for line in report.table(["NodeID", "Condition", "Credit"], rows):
        print("  " + line)
    for title, lines in (("Variable Usage Rates:", report.vur_table(kb, 2)),
                         ("Node Usage Rates:", report.nur_table(kb, 2)),
                         ("Rule Usage Rates:", report.rur_table(kb, 2))):
        print(title)
        for line in lines:
            print("  " + line)
    usage = kb.usage()
    print("Most important variables: "
          + ", ".join(v + " (NS " + str(usage.ns(v)) + ")"
                      for v in usage.important_variables()))
    print("Question order: " + ", ".join(usage.question_order()))


def count(table, key):
    table[key] = table.get(key, 0) + 1
```

Kode 7.8 (Lanjutan) *shop_router.py: aplikasi yang sama dalam Python.*

```
def show_counts(title, table):
    print(title)
    for k in sorted(table, key=lambda k: (-table[k], k)):
        print("  " + k.ljust(15) + str(table[k]))


def cmd_inbox(kb, signals):
    per_route, per_node, wrong = {}, {}, []
    checked = [0, 0]
    order = kb.usage().question_order()
    messages = labelled("inbox.txt")
    for i, (text, good, _) in enumerate(messages):
        routed, result, asked = route(kb, signals, text)
        checked[0] += len(asked)
        other = route(kb, signals, text, order)
        checked[1] += len(other[2])
        mark = "  " if routed == good else "X "
        if routed != good:
            wrong.append(str(i + 1))
        print(str(i + 1).rjust(2) + ". " + mark + routed.ljust(15)
              + text[:48])
        count(per_route, routed)
        count(per_node, result.paths[0].last().id if result.paths
              else "(no node)")
        if other[0] != routed:
            print("       (important-first order would say " + other[0]
                  + ")")
    n = len(messages)
    print("Right: " + str(n - len(wrong)) + " of " + str(n)
          + ("; wrong: " + ", ".join(wrong) if wrong else ""))
    show_counts("Messages per person:", per_route)
    show_counts("Messages per node:", per_node)
    print("Signals checked per message: " + num.fmt(checked[0] / n, 2)
          + " in the engine's order, " + num.fmt(checked[1] / n, 2)
          + " important-first")
```

Kode 7.8 (Lanjutan) *shop_router.py: aplikasi yang sama dalam Python.*

```
def cmd_route(kb, signals, text):
    routed, result, checked = route(kb, signals, text)
    print("Message: " + text)
    print("Signals checked: " + ", ".join(
        v + " " + result.memory.facts[v].value for v in checked))
    print("Route: " + routed)
    if routed == NOBODY:
        print("Why: no node matched, so Bu Rina reads it herself.")
        return
    print("Why:")
    for line in result.how():
        print("  " + line)


def cmd_voice(kb, echo):
    def ask(question):
        print(kb.variables[question.variable].question + " (yes/no) > ",
              end="")
        answer = sys.stdin.readline().strip()
        if echo:
            print(answer)
        return answer or None

    listen = kb.usage().question_order()[:3]
    print("Voice note or photo? Listen above all for: " + ", ".join(listen))
    result = kb.infer({}, ask)
    found = result.conclusions()
    print("Route: " + (found[0].value if found else NOBODY))


def main(args):
    signals = Signals(os.path.join(DATA, "signals.csv"))
    command = args[0] if args else "inbox"
    if command == "teach":
        cmd_teach(signals, args[1:] or ["teach-week.txt"])
        return
    kb = KnowledgeBase.load(KB_FILE) if os.path.exists(KB_FILE) \
        else new_kb(DATA)
    if command == "stats":
        cmd_stats(kb)
    elif command == "route":
        cmd_route(kb, signals, " ".join(args[1:]))
    elif command == "voice":
        cmd_voice(kb, "--echo" in args)
    else:
        cmd_inbox(kb, signals)
```

Kode 7.8 (Lanjutan) *shop_router.py: aplikasi yang sama dalam Python.*

```


if __name__ == "__main__":
    main(sys.argv[1:])
```

Untuk menjalankannya, kompilasi sekali dari folder `Chapter07/java` dengan `javac -d out -sourcepath ../../vcirs-core/java/src *.java` lalu ketik `java -cp out ShopRouter teach`, atau dari `Chapter07/python`, `python shop_router.py teach`. README bab ini mendaftar setiap perintah untuk Windows, macOS, dan Linux.

7.7 Minggu Pertama Bu Rina

Selama minggu pertamanya bersama pengarah, Bu Rina mencatat lima belas pesan begitu pesan itu datang, masing-masing dengan rute yang akan ia pilih, dalam `teach-week.txt`. Ketika pengarah keliru, ia juga mencatat sinyal yang membuat perbedaan:

```
# One message per line: message | right route | the signal that made the difference
Bu, beras 5 kg masih ada? Harganya berapa sekarang? | owner
Sudah saya transfer ya bu 85rb, tolong dicek | cashier
Kak sudah tf tapi pesanannya belum sampai dari tadi siang | courier | late
Berapa ongkir ke Perum Griya Asri? | courier | delivery
...
```

`teach` memutar ulang minggunya:

```
Seed rules: Toko Rina message router: 11 variables, 7 nodes, 7 rules
Teaching teach-week.txt:
 1. Bu, beras 5 kg masih ada? Harganya berapa sekarang?
    signals: price, stock
    routed to owner (owner#1): right
 2. Sudah saya transfer ya bu 85rb, tolong dicek
    signals: payment
    routed to cashier (cashier#1): right
 3. Kak sudah tf tapi pesanannya belum sampai dari tadi siang
    signals: payment, late
    routed to cashier (cashier#1): WRONG, Rina says courier
    learned courier#2: late = yes -> route = courier (exception of cashier#1)
    earlier cases whose route would change: none
 4. Berapa ongkir ke Perum Griya Asri?
    signals: price, delivery
    routed to owner (owner#1): WRONG, Rina says courier
    learned courier#3: delivery = yes -> route = courier (exception of owner#1)
    earlier cases whose route would change: none
 5. Minyak goreng yang kemarin bocor bu, gimana ini
    signals: complaint
    routed to owner now (owner now#1): right
 6. Harga telur kok naik terus, kecewa saya
    signals: price, complaint
    routed to owner (owner#1): WRONG, Rina says owner now
    learned owner now#2: complaint = yes -> route = owner now (exception of owner#1)
    earlier cases whose route would change: none
 7. Hi, is the shop open on Sunday?
    signals: hours
    routed to auto reply (auto reply#1): right
 8. Do you deliver to Jalan Melati? How much is the delivery fee?
    signals: price, delivery
    routed to courier (courier#3): right
 9. PROMO GRATIS saldo 500rb!!! klik https://bit.ly/saldo-gratis
    signals: link
    routed to owner reads it: WRONG, Rina says ignore
    learned ignore#1: link = yes -> route = ignore (top level)
    earlier cases whose route would change: none
10. Cek harga minyak murah di sini www.promo-murah.xyz
    signals: price, link
    routed to owner (owner#1): WRONG, Rina says ignore
    learned ignore#2: link = yes -> route = ignore (exception of owner#1)
    earlier cases whose route would change: none
11. Pesan gas 3 kg 2 tabung ya bu, diantar ke rumah
    signals: order, delivery
    routed to owner (owner#3): right
12. Bu salah kirim, saya pesan gula pasir yang datang gula merah
    signals: order, delivery, complaint
    routed to owner (owner#3): WRONG, Rina says owner now
    learned owner now#3: complaint = yes -> route = owner now (exception of owner#3)
    earlier cases whose route would change: none
13. Kak tf nya kelebihan 10rb, dikembalikan atau jadi saldo aja?
    signals: payment
    routed to cashier (cashier#1): right
14. Is the 19 liter gallon available? I want 2
    signals: stock, order
    routed to owner (owner#2): right
15. The eggs I paid for yesterday were broken, 3 of them
    signals: payment, complaint
    routed to cashier (cashier#1): WRONG, Rina says owner now
    learned owner now#4: complaint = yes -> route = owner now (exception of cashier#1)
    earlier cases whose route would change: none
Right the first time: 8 of 15
All taught messages again: 15 of 15 right
Saved shop-kb.json: Toko Rina message router: 11 variables, 14 nodes, 14 rules
```

Bacalah minggu itu sebagaimana Bu Rina menjalaninya.

Pesan 1 dan 2 diarahkan dengan benar oleh aturan-aturan awal. Pesan pertama mengandung dua sinyal, `price` dan `stock`; node harga datang lebih dulu dan menjawab, dan karena keduanya menuju pemilik, tidak jadi soal yang mana.

Pesan 3 adalah pelajaran pertama. *Kak sudah tf tapi pesanannya belum sampai* mengandung `payment` dan `late`. Aturan pembayaran mengirimnya kepada Dinda, padahal pelanggan tidak menanyakan uang; ia menanyakan di mana berasnya. Bu Rina mengatakan rutenya kurir, dan sinyal yang membuat perbedaan adalah `late`. Mesin menggantung `courier#2: late = yes` di bawah `cashier#1`. Pemeriksaan cornerstone terdampak menyatakan tidak ada kasus terdahulu yang berubah rute.

Pesan 4, 6, dan 10 menyempurnakan aturan harga tiga kali: ongkos kirim (`delivery`), keluhan tentang harga (`complaint`), dan harga yang tersembunyi dalam tautan sampah (`link`). Masing-masing menjadi pengecualian `owner#1`. **Pesan 8** — *Do you deliver to Jalan Melati? How much is the delivery fee?* — lalu datang dalam bahasa Inggris dan langsung diarahkan dengan benar oleh pengecualian yang dipelajari dari pesan 4, yang berbahasa Indonesia. Pengetahuannya tentang sinyal, bukan tentang bahasa.

Pesan 9 adalah sampah, dan tidak ada node yang mengarahkannya: `link` belum pernah terlihat. Mesin menambahkan `ignore#1: link = yes` di tingkat puncak. Satu pesan kemudian, sampah yang juga menyebut harga tertangkap oleh aturan harga, dan menjadi pengecualian `ignore#2` dari `owner#1`.

Pesan 12 dan 15 adalah keluhan yang bersembunyi di balik sinyal lain: salah kirim (*salah kirim*, yang juga mengandung *pesan* dan *kirim*) dan telur pecah yang sudah dibayar. Keduanya menjadi pengecualian yang segera mengirim pesan kepada pemilik.

Di akhir, delapan dari lima belas pesan diarahkan dengan benar pada kesempatan pertama, tujuh diajarkan, dan — baris yang paling penting — kelima belasnya diarahkan dengan benar ketika dicoba lagi. Basis pengetahuan tumbuh dari 7 menjadi 14 node.

Berkas yang disimpan juga menyimpan **riwayat** minggu itu, satu catatan per langkah: tujuh catatan `add` untuk aturan-aturan awal, lalu delapan catatan `accept` (opsi 0) untuk pesan yang diarahkan dengan benar dan tujuh catatan `add` untuk koreksi-koreksinya — enam dengan opsi 2, sebagai pengecualian, dan satu dengan opsi 5, aturan sampah di tingkat puncak. Tidak ada satu pun dalam riwayat itu yang harus ditulis dengan tangan. Riwayat itu adalah buku harian minggu tersebut, dan bersama Credit ia menjawab pertanyaan yang selalu muncul pada minggu-minggu pertama setiap sistem baru: apakah ia makin baik? Di sini jawabannya terlihat dari urutan baris-barisnya. Koreksi-koreksi pertama datang bertubi-tubi; pesan-pesan berikutnya, seperti pertanyaan pengantaran berbahasa Inggris, makin sering dijawab oleh apa yang dipelajari dari pesan-pesan sebelumnya. Toko yang terus mengajar selama sebulan akan melihat bagian pesan yang diterima tumbuh minggu demi minggu, dan lonjakan koreksi yang tiba-tiba akan memberitahunya bahwa ada sesuatu yang baru sedang terjadi di antara para pelanggannya.

7.8 Membaca Angka-Angkanya

Kini pertanyaan pokok bab ini: apa yang sudah dipelajari pengarah, dan apa yang paling penting di dalamnya? `stats` dimulai dengan Credit:

```
Credits of every condition:
  NodeID        Condition        Credit
  ------------  ---------------  ------
  owner#1       price = yes      4
  owner#2       stock = yes      1
  owner#3       order = yes      2
  cashier#1     payment = yes    3
  courier#1     delivery = yes   1
  owner now#1   complaint = yes  1
  auto reply#1  hours = yes      1
  courier#2     late = yes       1
  courier#3     delivery = yes   1
  owner now#2   complaint = yes  1
  ignore#1      link = yes       1
  ignore#2      link = yes       1
  owner now#3   complaint = yes  1
  owner now#4   complaint = yes  1
```

Setiap node baru ber-Credit 1 pada kondisinya. Aturan-aturan awal yang dikoreksi punya lebih: `price = yes` ber-Credit 4 (tiga pengecualian dibuat di bawah `owner#1`), `payment = yes` ber-Credit 3 (dua pengecualian di bawah `cashier#1`), dan `order = yes` ber-Credit 2 (satu). Tanpa diminta siapa pun, Credit telah menuliskan riwayat minggu itu: aturan harga adalah aturan Toko Rina yang paling sering dikoreksi, disusul aturan pembayaran. Bila Bu Rina meninjau pengetahuannya sepuluh menit pada hari Minggu, dua node itulah yang pertama dibaca.

Berikutnya Variable Usage Rate:

```
Variable Usage Rates:
  VariableID  NodeID        VariableOrder  VUR
  ----------  ------------  -------------  ----
  stock       owner#2       1              1.00
  hours       auto reply#1  1              1.00
  late        courier#2     1              1.00
  order       owner#3       1              2.00
  delivery    courier#1     1              2.00
  delivery    courier#3     1              2.00
  link        ignore#1      1              2.00
  link        ignore#2      1              2.00
  payment     cashier#1     1              3.00
  price       owner#1       1              4.00
  complaint   owner now#1   1              4.00
  complaint   owner now#2   1              4.00
  complaint   owner now#3   1              4.00
  complaint   owner now#4   1              4.00
```

Setiap node di sini hanya punya satu kondisi, jadi setiap Closeness Degree bernilai 1 dan setiap VUR hanyalah Credit dikali NS. `complaint` ber-VUR 4 di masing-masing dari empat node-nya walaupun setiap Credit-nya 1, karena ia **dibagi** oleh empat node (`owner now#1` di tingkat puncak dan tiga pengecualian); `price` ber-VUR 4 di satu node karena Credit-nya 4. Dua jalan menuju penting itu berbeda — satu variabel dipakai di banyak tempat, yang lain disempurnakan di satu tempat — dan VUR menaruh keduanya pada skala yang sama.

Node Usage Rate mengulang VUR, karena node dengan satu kondisi ber-NUR sama dengan VUR-nya. Rule Usage Rate lebih menarik, karena ia merata-ratakan di sepanjang lintasan sebuah pengecualian:

```
Rule Usage Rates:
  RuleID        RUR
  ------------  ----
  owner#2       1.00
  auto reply#1  1.00
  owner#3       2.00
  courier#1     2.00
  courier#2     2.00
  ignore#1      2.00
  cashier#1     3.00
  courier#3     3.00
  ignore#2      3.00
  owner now#3   3.00
  owner now#4   3.50
  owner#1       4.00
  owner now#1   4.00
  owner now#2   4.00
Most important variables: complaint (NS 4)
Question order: price, complaint, payment, order, delivery, link, stock, hours, late
```

`owner now#4`, pengecualian *complaint* di bawah aturan pembayaran, ber-RUR 3.5: rata-rata `cashier#1` (NUR 3) dan dirinya sendiri (NUR 4). `courier#2`, pengecualian *late* di bawah aturan pembayaran yang sama, hanya 2: `late` dipakai di satu node, dengan Credit 1. Dua baris terakhir adalah putusan analisis variabel dan analisis nilai. `complaint` adalah variabel terpenting, dibagi oleh empat node. Urutan pertanyaan dimulai dengan `price` dan `complaint` (keduanya ber-VUR 4), lalu `payment`, `order`, dan `delivery`, dan diakhiri sinyal-sinyal yang paling jarang dipakai.

CONTOH: Membaca `stats` dalam lima langkah. (1) Cari kondisi dengan Credit terbesar: node-node itulah yang paling sering dikoreksi, jadi bacalah lebih dulu. (2) Cari variabel dengan NS terbesar di baris-baris terakhir: variabel itu dibagi banyak node, sehingga daftar katanya paling penting. (3) Di tabel VUR, carilah variabel yang VUR-nya melonjak sejak terakhir kali: ada sesuatu yang baru terjadi di sekitarnya. (4) Di tabel RUR, tingkat terbesar dimiliki alur penalaran yang paling sibuk, biasanya pengecualian di bawah aturan yang banyak disempurnakan. (5) Bandingkan dengan hitungan lalu lintas dari `inbox` sebelum memutuskan apa pun: aturan yang banyak dipakai tetapi sedikit mengarahkan pesan adalah aturan yang halus, bukan aturan yang rusak.

JEBAKAN: "Terpenting" dalam VCIRS berarti "paling banyak dipakai oleh pengetahuan", bukan "paling mendesak bagi usaha". Di Toko Rina keluhan kebetulan keduanya, tetapi sebuah sinyal dapat jarang muncul dalam pengetahuan dan tetap sangat berarti — pelanggan yang menulis bahwa tabung elpijinya berbau gas perlu segera ditangani, walaupun belum ada aturan yang menyebut *gas*. Pakailah angka-angka untuk memutuskan ke mana melihat, dan pertimbangan Anda sendiri untuk memutuskan apa yang penting.

7.9 Senin Pagi

Pada hari Senin pengarah bekerja sendiri untuk pertama kalinya, sementara Bu Rina melayani pembeli di etalase. Belakangan ia mencocokkan rute-rute pengarah dengan rute yang akan ia pilih; berkas `inbox.txt` memuat 25 pesan pagi itu beserta rute pilihan Bu Rina. `inbox` mengarahkan semuanya:

```
 1.   owner          Bu, gula pasir 1 kg harganya berapa?
 2.   cashier        Sudah tf 120rb ya kak, buktinya saya kirim
 3.   owner          Pesan beras 10 kg sama minyak 2 liter
 4. X owner          Pesanan saya belum sampai bu, sudah 2 jam
 5.   auto reply     Is the shop open today? It's a public holiday
 6.   owner          How much is a 12 kg LPG refill?
 7.   cashier        Kak, bisa bayar pakai QRIS?
 8.   owner now      Telurnya ada yang pecah 2, gimana bu?
 9.   courier        Ongkir ke Blok C berapa ya?
10.   owner          Bu, air galon masih ada? Mau ambil sore
11.   owner reads it Assalamualaikum bu, terima kasih ya kemarin
12.   ignore         Selamat! Anda menang undian, klik www.hadiah-und
13.   owner          Kak, pesen rokok 1 slop, anter ke pos ronda ya
14.   owner now      Minyaknya bocor di plastik, tolong diganti
15.   owner          Do you have Indomie goreng in stock? I need 1 bo
16.   cashier        Bu, sudah saya bayar lunas yang kemarin
17.   owner now      Harga beras naik lagi? Kecewa bu
18. X owner          Jam berapa tutup malam ini?
19.   courier        Kurirnya sudah jalan belum?
20.   owner          Bu tolong dicatat, besok saya ambil pesanan
21.   owner now      The sugar I paid for is missing from the bag
22.   owner          Ready sampo sachet renceng?
23.   cashier        Mbak, transferan saya sudah masuk?
24.   courier        Bisa kirim sekarang? Rumah saya dekat masjid
25. X owner          Bu Rina, anak saya titip uang 50rb buat bayar ut
       (important-first order would say cashier)
Right: 22 of 25; wrong: 4, 18, 25
Messages per person:
  owner          11
  cashier        4
  owner now      4
  courier        3
  auto reply     1
  ignore         1
  owner reads it 1
Messages per node:
  owner#3        5
  cashier#1      4
  owner#1        3
  owner#2        3
  courier#1      2
  owner now#1    2
  (no node)      1
  auto reply#1   1
  courier#3      1
  ignore#1       1
  owner now#2    1
  owner now#4    1
Signals checked per message: 4.72 in the engine's order, 4.28 important-first
```

Dua puluh dua dari dua puluh lima benar, dan tiga yang salah memberi pelajaran.

Pesan 4, *Pesanan saya belum sampai bu, sudah 2 jam*, mengandung `order` (*pesanan*) dan `late`. Aturan pesanan `owner#3` menjawab, dan ia tidak punya pengecualian untuk `late` — pengecualian yang dipelajari dari pesan 3 minggu lalu tinggal di bawah aturan *pembayaran*, dalam konteks tempat kesalahan itu terjadi. RDR melakukan apa yang dijanjikannya: pengetahuan yang dipelajari dalam satu konteks tidak merembes ke konteks lain. Bu Rina harus mengajarkan konteks ini juga.

Pesan 18, *Jam berapa tutup malam ini?*, mengandung *berapa*, kata harga, dan *tutup*, kata jam buka. Aturan harga datang lebih dulu dan menang. Di sini pendeteksi tidak salah: *berapa* memang muncul. Pengetahuannya yang memerlukan pengecualian.

Pesan 25, *Bu Rina, anak saya titip uang 50rb buat bayar utang*, mengandung `order` (*titip*, yang biasanya berarti "tolong sisihkan untuk saya") dan `payment` (*bayar*). Aturan pesanan menang; seharusnya kasir yang menerimanya.

Baris di bawah pesan 25 adalah percobaan analisis nilai. `inbox` mengarahkan setiap pesan dua kali: sekali menurut urutan mesin, dan sekali setelah memeriksa sinyal-sinyal menurut **urutan pertanyaan** sampai satu ditemukan — di sini `price`, lalu `complaint`, lalu `payment`, yang menemukan *bayar* dan membuat mesin mulai dari aturan pembayaran. Hanya untuk pesan 25 keduanya berbeda, dan kali ini urutan penting-dulu kebetulan benar. Itu memberi tahu Bu Rina sesuatu yang berguna: pesan itu cocok dengan **dua aturan tingkat puncak** yang rutenya berbeda, dan hanya urutan node yang memutuskan di antara keduanya. Setiap kali dua urutan itu berbeda pendapat, basis pengetahuan punya dua aturan yang berebut satu pesan, dan sebuah pengecualian perlu menyelesaikannya. Sepanjang pagi itu urutan penting-dulu memeriksa 4.28 sinyal per pesan, bukan 4.72. Mengapa mesin memerlukan hampir lima pemeriksaan untuk pesan yang hanya punya satu atau dua sinyal? Karena ia menelusuri node tingkat puncak menurut urutan Bu Rina mengajarkannya, dan setiap node yang gagal memakan satu pemeriksaan: pesan tentang jam buka ditanyai soal harga, stok, pesanan, pembayaran, pengantaran, dan keluhan sebelum aturan jam buka tercapai. Bagi pendeteksi itu tidak memakan biaya. Bagi manusia yang menjawab pertanyaan, itu melelahkan, dan karena itulah mode suara di Subbab 7.10 lebih dulu mengingatkannya pada sinyal-sinyal yang penting.

Apakah dengan demikian pengarah sebaiknya selalu memeriksa sinyal terpenting lebih dulu? Tidak secara membabi buta. Urutan penelusuran basis pengetahuan RDR adalah bagian dari pengetahuan yang diajarkan sang pakar — Bab 3 menunjukkan apa yang terjadi bila urutan aturan diubah di belakang punggung sang pakar. Pengarah tetap memakai urutan mesin untuk keputusannya, dan memakai urutan penting-dulu sebagai **pemeriksa** yang menunjuk aturan-aturan yang berebut, serta sebagai pedoman bagi manusia, dalam mode suara di Subbab 7.10.

Hitungan lalu lintas menjawab pertanyaan lain. Bu Rina sendiri menerima 11 dari 25 pesan, Dinda 4, Pak Darto 3; `owner#3`, aturan pesanan, mengarahkan 5 pesan, lebih banyak daripada node mana pun, walaupun Credit-nya hanya 2 — aturan sederhana yang bekerja. Hanya satu pesan yang tidak sampai ke node mana pun: *Assalamualaikum bu, terima kasih ya kemarin*, salam dan ucapan terima kasih, yang memang ingin dibaca Bu Rina sendiri.

Perintah `route` menjelaskan satu keputusan. Untuk pesan yang menggabungkan pembayaran dan keluhan:

```
Message: Bu, sudah tf ya, tapi telurnya pecah 2
Signals checked: price no, stock no, order no, payment yes, late no, complaint yes
Route: owner now
Why:
  owner now#4 fired because:
    payment = yes  (answered by the user, CF 100)
    complaint = yes  (answered by the user, CF 100)
    so route = owner now  [rule CF 100, conclusion CF 100]
```

Pendeteksi memeriksa enam sinyal, satu untuk setiap node yang dicoba mesin: harga, stok, dan pesanan langsung gagal; pembayaran menyalakan aturan pembayaran; pengecualian pertamanya, `late`, tidak menyala; yang kedua, `complaint`, menyala. Penjelasannya menyebut jawaban pendeteksi sebagai "answered by the user", karena bagi mesin, pendeteksi *adalah* pengguna yang menjawab pertanyaannya. Dan untuk pesan ucapan terima kasih:

```
Message: Assalamualaikum bu, terima kasih ya kemarin
Signals checked: price no, stock no, order no, payment no, delivery no, complaint no, hours
no, link no
Route: owner reads it
Why: no node matched, so Bu Rina reads it herself.
```

7.10 Membuatnya Lebih Baik

7.10.1 Mengajarkan tiga kesalahan hari Senin

Bu Rina menuliskan tiga kesalahan hari Senin dalam `teach-week2.txt`, masing-masing dengan sinyal yang membuat perbedaan, lalu mengajarkan kedua minggu itu bersama-sama dengan `teach teach-week.txt teach-week2.txt`. Baris-baris terakhir keluarannya memperlihatkan pengetahuan barunya:

```
Teaching teach-week2.txt:
16. Pesanan saya belum sampai bu, sudah 2 jam
    signals: order, late
    routed to owner (owner#3): WRONG, Rina says courier
    learned courier#4: late = yes -> route = courier (exception of owner#3)
    earlier cases whose route would change: none
17. Jam berapa tutup malam ini?
    signals: price, hours
    routed to owner (owner#1): WRONG, Rina says auto reply
    learned auto reply#2: hours = yes -> route = auto reply (exception of owner#1)
    earlier cases whose route would change: none
18. Bu Rina, anak saya titip uang 50rb buat bayar utang
    signals: order, payment
    routed to owner (owner#3): WRONG, Rina says cashier
    learned cashier#2: payment = yes -> route = cashier (exception of owner#3)
    earlier cases whose route would change: none
Right the first time: 8 of 18
All taught messages again: 18 of 18 right
Saved shop-kb.json: Toko Rina message router: 11 variables, 17 nodes, 17 rules
```

Setiap koreksi menjadi pengecualian dalam konteks yang tepat: *late* di bawah aturan pesanan, *hours* di bawah aturan harga, *payment* di bawah aturan pesanan. Baris *All taught messages again: 18 of 18 right* adalah pemeriksaan regresi: ketiga node baru tidak mengubah rute satu pun pesan terdahulu, termasuk *Pesan gas 3 kg 2 tabung ya bu, diantar ke rumah* (pesanan dan pengantaran), yang tetap menuju pemilik, karena pengecualian baru aturan pesanan memerlukan `late`, bukan `delivery`. Kotak masuk hari Senin kini berbunyi *Right: 25 of 25* — memang semestinya, karena tiga pesannya baru saja diajarkan; batu uji yang sebenarnya adalah pesan-pesan Senin berikutnya.

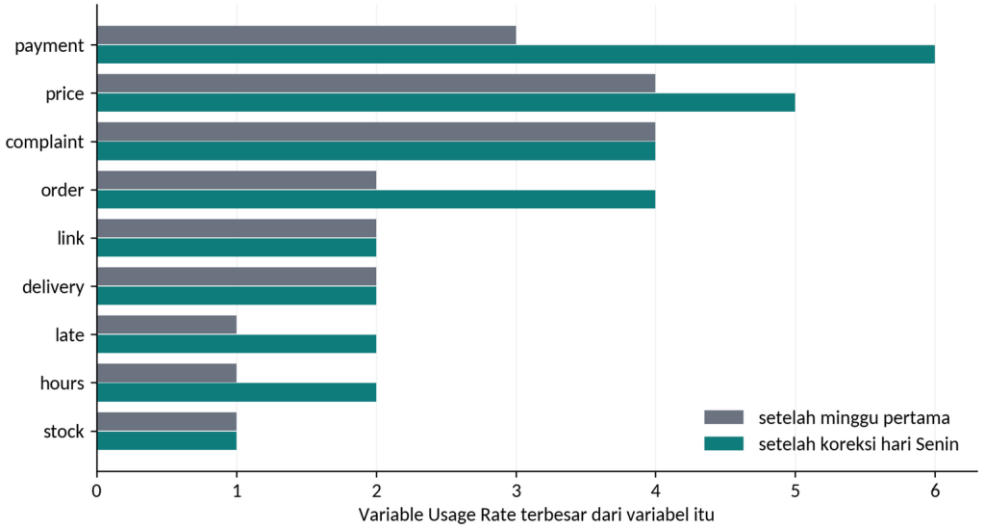
Angka-angkanya bergeser. Credit dan akhir tabel pemakaian kini berbunyi:

```
Credits of every condition:
  NodeID        Condition        Credit
  ------------  ---------------  ------
  owner#1       price = yes      5
  owner#2       stock = yes      1
  owner#3       order = yes      4
  cashier#1     payment = yes    3
  courier#1     delivery = yes   1
  owner now#1   complaint = yes  1
  auto reply#1  hours = yes      1
  courier#2     late = yes       1
  courier#3     delivery = yes   1
  owner now#2   complaint = yes  1
  ignore#1      link = yes       1
  ignore#2      link = yes       1
  owner now#3   complaint = yes  1
  owner now#4   complaint = yes  1
  courier#4     late = yes       1
  auto reply#2  hours = yes      1
  cashier#2     payment = yes    1
```

dan di bagian paling akhir:

```
Most important variables: complaint (NS 4)
Question order: payment, price, order, complaint, delivery, hours, late, link, stock
```

`payment` menjadi variabel dengan VUR terbesar, 6 di `cashier#1`: Credit-nya tetap 3, tetapi `payment` kini dibagi oleh dua node, aturan pembayaran dan pengecualian baru `cashier#2`, sehingga bobotnya berlipat dua. `order = yes` naik dari Credit 2 menjadi 4. **Gambar 7.2** membandingkan VUR terbesar setiap sinyal setelah minggu pertama dan setelah koreksi hari Senin.



Gambar 7.2 Variable Usage Rate terbesar setiap sinyal setelah minggu pertama (abu-abu) dan setelah tiga koreksi hari Senin (hijau toska). Payment, order, late, dan hours naik; yang lain tidak berubah. Sinyal `proof` tidak punya batang karena belum ada node yang memakainya.

7.10.2 Mode suara

Ketika pelanggan mengirim pesan suara, pendeteksi tidak punya apa pun untuk dibaca. Pembantu — katakanlah sepupu Dinda, yang menjaga ponsel pada hari Sabtu — mendengarkan, dan pengarah menanyakan kepadanya pertanyaan-pertanyaan yang diperlukannya. Sebelum pertanyaan pertama, pengarah mengingatkannya tiga sinyal yang paling penting, diambil dari urutan pertanyaan:

```
Voice note or photo? Listen above all for: payment, price, order
Does the message ask about a price? (yes/no) > no
Does it ask whether something is in stock? (yes/no) > no
Does the customer want to order or buy? (yes/no) > no
Is it about a payment or a transfer? (yes/no) > yes
Does the customer say an order has not arrived? (yes/no) > no
Is the customer complaining? (yes/no) > yes
Route: owner now
```

Pertanyaan-pertanyaannya datang menurut urutan mesin, urutan yang sama yang mengarahkan pesan tertulis, sehingga sebuah pesan suara mendapat persis rute yang akan didapat pesan tertulis dengan sinyal yang sama. Pengingatnya adalah analisis nilai yang dipakai sesuai maksud tesisnya, sebagai **pedoman bagi pengguna**: dengarkan terutama soal pembayaran, harga, dan pesanan, karena pengetahuan paling mengandalkan ketiganya.

7.10.3 Merawat daftar kata

Sebagian besar rute yang salah di toko sungguhan berasal dari salah satu dari dua sumber, dan ada gunanya membedakan keduanya. Entah **pendeteksi** melewatkan atau salah membaca sebuah kata — *kurirnya* belum ada di daftar, *berapa* muncul dalam pertanyaan tentang waktu — entah **pengetahuan** memberi rute yang salah untuk sinyal-sinyal yang terbaca dengan benar. Yang pertama dibetulkan di `signals.csv`, yang kedua dengan mengajar. Perintah `route` menunjukkan yang mana: bila sinyal yang diperiksa salah, betulkan kata-katanya; bila sinyalnya benar tetapi rutenya salah, ajarkan.

Tingkat pemakaian membantu di sini juga. Sinyal dengan VUR tinggi adalah sinyal yang diandalkan pengetahuan di banyak tempat, sehingga satu kata yang hilang dari daftarnya membuat banyak jenis pesan salah arah. Di Toko Rina itu berarti daftar untuk `payment`, `price`, dan `complaint` layak diperiksa cermat setiap beberapa minggu, dengan memperhatikan kebiasaan mengobrol keluarga itu sendiri (*tf*, *trf*, *transfer-an*, *sdh bayar*).

7.10.4 Pengujian

`RouterTest.java` dan `test_router.py` memeriksa pendeteksi (kata utuh, frasa, urutan sinyal yang ditemukan), minggu pertama (14 node, Credit 4 untuk `price = yes`, `complaint` sebagai variabel terpenting, dua rute), dan minggu kedua (17 node, sebuah jawaban lama bertahan, pesanan terlambat diarahkan ke kurir, `payment` pertama dalam urutan pertanyaan). Keduanya diakhiri dengan:

```
all tests passed
```

7.11 Memakainya Sungguh-Sungguh

Toko Anda sendiri. Tidak ada yang khas kelontong dalam `ShopRouter` kecuali tiga berkas data. Tuliskan sinyal-sinyal Anda dan kata-katanya di `signals.csv`, orang-orang atau tumpukan Anda sebagai rute, dan tiga sampai sepuluh aturan pertama di `seed-cases.json`. Konter pulsa mungkin mengarahkan pesan *pulsa*, *paket data*, dan *token listrik* secara berbeda; binatu mungkin punya `pickup`, `ready`, `price`, dan `complaint`; toko kue kecil `order`, `custom cake`, `pickup time`, dan `delivery`. Lalu jalankan pengarah di samping Anda selama seminggu, dan catat setiap rute yang salah beserta rute yang benar dan sinyal yang membuat perbedaan.

Angka yang sama, toko yang lain. Bayangkan binatu milik Mbak Sari setelah sebulan. `stats`-nya memperlihatkan bahwa `ready` (*sudah jadi?*, *is it done?*) ber-Credit 5 pada aturannya: ia

mengoreksinya lima kali, setiap kali untuk pesan yang juga menyebut hal lain — pesanan kilat, kemeja yang hilang, pengambilan oleh tetangga. Aturan itu adalah tempat tersibuk dalam pengetahuannya, dan membaca kelima pengecualiannya bersama-sama memberi tahu sesuatu tentang usahanya: pelanggan yang menanyakan apakah cuciannya sudah jadi sering kali punya pertanyaan kedua yang lebih penting. Ia memutuskan untuk menjawab sendiri pesan-pesan *ready* untuk sementara. Sementara itu `price` ber-Credit 1 dan mengarahkan seperempat dari seluruh pesan: aturan sederhana yang bekerja, yang dapat dibiarkan saja.

Dari ponsel ke program. Pengarah membaca teks. Cara paling sederhana untuk mengisinya adalah menyalin pesan dari WhatsApp ke berkas teks pada akhir hari, atau mengekspor obrolan (WhatsApp dapat mengekspor obrolan sebagai berkas teks) lalu membiarkan skrip kecil mengambil pesan-pesannya. Pakailah rutenya untuk memasang label WhatsApp Business dengan tangan [2]. Menghubungkan program langsung ke WhatsApp Business Platform memang mungkin, tetapi itu memerlukan akun bisnis terdaftar dan sebuah server, dan tidak diperlukan untuk mendapat manfaat bab ini: pengetahuan dan statistiknya sama saja.

Sepuluh menit setiap minggu. Sekali seminggu, jalankan `stats` dan `inbox` pada pesan-pesan beberapa hari. Bacalah node dengan Credit tertinggi lebih dulu: di sanalah pelanggan Anda terus mengejutkan aturan-aturannya. Lihat setiap pesan tempat urutan penting-dulu berbeda pendapat dengan mesin: di sana dua aturan berebut. Lihat lalu lintasnya: node yang sebulan penuh tidak mengarahkan apa pun mungkin menggambarkan keadaan yang sudah tidak terjadi lagi.

Simpan pesannya, bukan orangnya. Kasus-kasus dalam `shop-kb.json` hanya memuat sinyal — `payment = yes`, `late = yes` — tidak pernah nama, nomor, atau pesan pelanggan. Pertahankan begitu. Pesan-pesan itu sendiri adalah data pribadi pelanggan Anda dan tunduk pada undang-undang pelindungan data pribadi Indonesia [8]; simpan berkas-berkas pengajaran di komputer toko sendiri, hapus berkas lama yang tidak diperlukan lagi, dan jangan pernah memasukkan nomor telepon ke basis pengetahuan yang mungkin Anda bagikan dengan toko lain.

Tumbuh lebih jauh. Ketika basis pengetahuan sudah memiliki beberapa puluh node, pembangkitan rule di Bab 6 dapat mengusulkan aturan-aturan umum dari variabel-variabel terpenting, dan Bab 10 memperlihatkan cara memasang basis pengetahuan seperti milik pengarah ini pada halaman web kecil yang dapat dipakai seluruh keluarga dari ponsel masing-masing.

7.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Membaca Credit sebagai lalu lintas. Aturan ber-Credit 1 mungkin mengarahkan separuh dari seluruh pesan. *Hindari:* bacalah Credit dan tingkat pemakaian sebagai "seberapa banyak pengetahuan disempurnakan dan dibagi", dan pakailah hitungan lalu lintas untuk "berapa banyak pesan".

Membaca "terpenting" sebagai "paling mendesak". Tingkat pemakaian tidak tahu apa pun tentang usaha Anda. *Hindari:* tentukan sendiri mana yang mendesak; pakailah angka-angka untuk menentukan ke mana melihat lebih dulu.

Membetulkan masalah pendeteksi dengan aturan. Bila *kurirnya* tidak dikenali, mengajarkan node tidak akan menolong, karena sinyalnya tidak pernah menjadi `yes`. *Hindari:* lihat dulu sinyal-sinyal yang diperiksa oleh `route`; betulkan daftar kata bila ada kata yang terlewat.

Mengajar tanpa menyebut perbedaannya. Tanpa sinyal yang disebut, node baru mengambil semua sinyal di bagian B, dan bisa menjadi terlalu sempit untuk menyala lagi. *Hindari:* sebutkan satu sinyal yang menjelaskan rute yang berbeda setiap kali Anda mengetahuinya.

Mengubah urutan aturan awal untuk "membetulkan" kesalahan. Itu mengubah rute pesan-pesan yang tidak pernah Anda periksa, seperti daftar datar di Bab 3. *Hindari:* ajarkan pengecualian dalam konteks kesalahannya.

Mengharapkan pengetahuan berpindah antarkonteks. Pengecualian *late* di bawah aturan pembayaran tidak menolong pesan terlambat yang diarahkan aturan pesanan. *Hindari:* bila kesalahan yang sama muncul di bawah aturan lain, ajarkan di sana juga; lama-kelamaan pembangkitan rule di Bab 6 dapat menyarankan aturan umum.

Kata yang termasuk dua sinyal. *Berapa* adalah kata harga sekaligus bagian dari *jam berapa*. *Hindari:* tambahkan frasa itu ke sinyal yang lebih khusus lalu ajarkan pengecualiannya, seperti dilakukan Bu Rina, atau buang kata yang bermakna ganda dan daftarkan frasa-frasa yang lebih panjang sebagai gantinya.

Membandingkan tingkat pemakaian antartoko. VUR 6 di satu basis pengetahuan tidak berarti apa-apa di basis pengetahuan lain; angkanya bergantung pada banyaknya node masing-masing. *Hindari:* bandingkan node dan variabel di dalam satu basis pengetahuan, dan amati perubahannya dari waktu ke waktu.

Menyimpan data pribadi dalam kasus. *Hindari:* simpan hanya sinyal di basis pengetahuan.

7.13 Ringkasan Bab

- Toko kecil dapat mengarahkan sebagian besar pesan WhatsApp-nya dengan **sinyal** — jenis-jenis kata dalam bahasa Indonesia dan Inggris — tanpa memahami bahasanya; pendeteksi menemukan kata utuh dan frasa, dan menjawab pertanyaan mesin hanya ketika ditanya (**Gambar 7.1**, **Kode 7.3**, **Kode 7.4**).
- Mengarahkan adalah mengklasifikasi; tujuh aturan awal yang terlalu umum ditambah pengecualian yang diajarkan pesan demi pesan menghasilkan pengarah yang mengirim ke-25 pesan sebuah pagi kepada orang yang tepat setelah dua sesi mengajar yang singkat (**Kode 7.5**, **Kode 7.6**, **Kode 7.7**, **Kode 7.8**).
- **Credit** sebuah kondisi bertambah ketika seorang pakar membuat node di bawah node-nya untuk kasus yang sepakat dengannya; ia mencatat di mana pengetahuan disempurnakan, bukan berapa banyak pesan yang lewat.
- **VUR** adalah Credit dikali Weight, Weight adalah NS dikali Closeness Degree VO/TV; **NUR** merata-ratakan VUR dalam sebuah node dan **RUR** merata-ratakan NUR di sepanjang sebuah rule (Persamaan (7.1), Persamaan (7.2), Persamaan (7.3), Persamaan (7.4), Persamaan (7.5)); tingkat yang dihitung dengan tangan sama dengan hasil mesin (**Kode 7.1**, **Kode 7.2**).
- Analisis variabel memberi **variabel terpenting** (yang paling banyak dibagi) dan analisis nilai memberi **urutan pertanyaan**; di Toko Rina keduanya menunjuk aturan yang paling sering disempurnakan, daftar kata yang paling perlu dirawat, dan — ketika urutan penting-dulu berbeda dengan urutan mesin — aturan-aturan yang berebut pesan yang sama (**Gambar 7.2**).
- Hitungan lalu lintas (pesan per orang, per node) adalah jenis statistik kedua yang berbeda, dan sebuah toko memerlukan keduanya.

7.14 Soal Latihan

7.1 Sebuah node punya tiga kondisi: `payment = yes` (Credit 2), `late = yes` (Credit 1), dan `complaint = yes` (Credit 1). Dalam basis pengetahuan, `payment` dipakai oleh 2 node, `late` oleh 3, dan `complaint` oleh 4. Hitunglah ketiga VUR dan NUR node itu dengan tangan.

7.2 Dengan Credit yang tercetak di Subbab 7.8, jelaskan masing-masing dalam satu kalimat mengapa `owner#1` ber-Credit 4 dan mengapa `owner#2` (aturan stok) tetap ber-Credit 1, walaupun aturan stok mengarahkan tiga pesan hari Senin.

7.3 Dalam keluaran Subbab 7.8, `courier#3` ber-RUR 3 dan `courier#2` ber-RUR 2, walaupun keduanya pengecualian dengan satu kondisi dan Credit 1. Jelaskan perbedaannya.

7.4 Setelah koreksi hari Senin, `route` memberi hasil ini untuk sebuah pesan baru:

```
Message: Kak, harga telur berapa? Sekalian dianter ya
Signals checked: price yes, delivery yes
Route: courier
Why:
  courier#3 fired because:
    price = yes  (answered by the user, CF 100)
    delivery = yes  (answered by the user, CF 100)
    so route = courier  [rule CF 100, conclusion CF 100]
```

Apakah rutenya benar bagi Toko Rina? Bila Bu Rina mengatakan pemilik yang seharusnya menerimanya, dapatkah ia mengajarkannya dengan sinyal-sinyal yang dikenal pendeteksi? Apa yang akan Anda tambahkan?

7.5 Seorang mahasiswa di perumahan itu menulis *The rice has not arrived and it is already 8 pm*. Pengarah menjawab:

```
Message: The rice has not arrived and it is already 8 pm
Signals checked: price no, stock no, order no, payment no, delivery no, complaint no, hours
no, link no
Route: owner reads it
Why: no node matched, so Bu Rina reads it herself.
```

Mengapa `late` tidak pernah diperiksa, padahal *not arrived* ada di daftar katanya? Tuliskan baris yang akan Anda tambahkan ke berkas pengajaran untuk membetulkannya, dan sebutkan di mana node barunya akan ditempatkan.

7.6 Mengapa `teach` mencatat pesan yang diarahkan dengan benar dengan opsi 0, alih-alih tidak berbuat apa-apa? Sebutkan satu pertanyaan tentang basis pengetahuan yang dapat Anda jawab belakangan berkat catatan itu.

7.7 Sepupu Bu Rina membuka konter pulsa. Rancanglah lima sinyal dengan sedikitnya tiga kata kunci bahasa Indonesia dan dua bahasa Inggris masing-masing, tiga rute, dan empat aturan awal. Sinyal mana yang Anda duga akan menjadi variabel terpenting setelah sebulan, dan mengapa?

7.8 Jelaskan, dengan contoh pesan 25 di Subbab 7.9, mengapa perbedaan pendapat antara urutan mesin dan urutan penting-dulu adalah informasi yang berguna, dan mengapa pengarah tetap memutuskan menurut urutan mesin.

7.15 Penyelesaian Soal Latihan

Penyelesaian 7.1. Node itu punya $TV = 3$, jadi derajat kedekatannya $1/3$, $2/3$, dan $3/3$. `payment`: $2 \times 2 \times 1/3 = 4/3 \approx 1.33$. `late`: $1 \times 3 \times 2/3 = 2$. `complaint`: $1 \times 4 \times 1 = 4$. NUR-nya $(4/3 + 2 +$

$4)/3 = 22/9 \approx 2.44$. Perhatikan bagaimana kondisi terakhir, yang paling dekat dengan konklusi, paling berbobot.

Penyelesaian 7.2. `owner#1` dimulai dengan Credit 1 dan tiga pengecualian dibuat di bawahnya (untuk pesan 4, 6, dan 10 minggu pertama), masing-masing untuk pesan yang menyebut harga, sehingga ia memperoleh tiga Credit. `owner#2` mengarahkan pesan dengan benar tetapi tidak pernah *dikoreksi*: tidak ada node yang pernah dibuat di bawahnya, jadi Credit-nya tidak berubah. Credit menghitung penyempurnaan, bukan lalu lintas.

Penyelesaian 7.3. RUR merata-ratakan NUR semua node di lintasan rule itu. `courier#3` adalah pengecualian `owner#1`, yang NUR-nya 4, dan NUR-nya sendiri 2 (Credit 1, `delivery` dibagi dua node), jadi RUR-nya $(4 + 2)/2 = 3$. `courier#2` adalah pengecualian `cashier#1`, yang NUR-nya 3, dan NUR-nya sendiri 1 (`late` dipakai sekali), jadi RUR-nya $(3 + 1)/2 = 2$.

Penyelesaian 7.4. Pelanggan menanyakan harga *dan* meminta diantar: ia memesan telur untuk diantar. Di Toko Rina pesanan menuju pemilik, yang menyiapkannya lalu mengutus Pak Darto, jadi rutenya salah. Tetapi pesan itu punya sinyal yang persis sama dengan kasus cornerstone `courier#3` (*Berapa ongkir…?*: `price` dan `delivery`), sehingga tidak ada perbedaan di antara keduanya, dan mesin menolak belajar — daftar perbedaannya kosong. Diperlukan sinyal baru yang membedakan pesanan dari pertanyaan ongkos, misalnya kata kunci `order` seperti *sekalian* atau *mau*, atau sinyal baru `product` yang mengenali nama barang (*telur*, *beras*, *gas*). Setelah `order` terdeteksi, Bu Rina dapat mengajarkan pengecualian `order = yes → owner` di bawah `courier#3`.

Penyelesaian 7.5. `late` hanya muncul di pengecualian: `courier#2` di bawah aturan pembayaran dan `courier#4` di bawah aturan pesanan. Pesan itu tidak mengandung kata pembayaran maupun pesanan, jadi tidak satu pun induknya menyala dan pengecualian-pengecualiannya tidak pernah dicoba; tidak ada node tingkat puncak yang memakai `late`. Tambahkan ke berkas pengajaran baris `The rice has not arrived and it is already 8 pm | courier | late`. Tidak ada node yang mengarahkan pesan itu, jadi `teach` menambahkan `courier#5: late = yes → route = courier` di tingkat puncak (opsi 5), tempat setiap pesan dapat mencapainya.

Penyelesaian 7.6. Pesan yang diterima adalah sebuah verifikasi: pesan sungguhan mengonfirmasi bahwa rute node itu benar. Catatan itu memungkinkan Anda menjawab pertanyaan seperti “seberapa sering setiap node dikonfirmasi bulan ini?”, “apakah node ini pernah dikonfirmasi sejak diajarkan?”, dan “apakah bagian pesan yang diterima terus bertambah?”, yang merupakan ukuran sehari-hari apakah pengarah makin baik, seperti diuraikan Bab 3 untuk dokter Wi-Fi.

Penyelesaian 7.7. Salah satu rancangan: sinyal `pulsa` (*pulsa*, *isi*, *top up*, *credit*, *reload*), `data` (*paket*, *kuota*, *data*, *internet*, *gb*), `token` (*token*, *listrik*, *pln*, *meter*, *electricity*), `payment` (*tf*, *transfer*, *bayar*, *paid*, *qris*), dan `failed` (*belum masuk*, *gagal*, *pending*, *not received*, *failed*); rute `sell`, `cashier`, dan `check the provider`; aturan awal `pulsa → sell`, `data → sell`, `token → sell`, `payment → cashier`. Sinyal `failed` kemungkinan besar akan menjadi variabel terpenting: ia akan muncul sebagai pengecualian di bawah setiap aturan penjualan (isi ulang yang belum masuk perlu diperiksa, apa pun yang dibeli), sehingga ia dibagi oleh beberapa node.

Penyelesaian 7.8. Pesan 25 mengandung `order` dan `payment`, sehingga ia cocok dengan aturan pesanan dan aturan pembayaran, dua aturan tingkat puncak dengan rute berbeda. Menurut urutan mesin, aturan pesanan datang lebih dulu dan menang; menurut urutan penting-dulu, `payment` ditemukan lebih dulu dan aturan pembayaran yang menang. Perbedaan pendapat itu membuktikan bahwa dua aturan berebut dan hanya urutan node yang memutuskan, dan itulah persisnya keadaan ketika sang pakar perlu mengajarkan pengecualian. Pengarah tetap

memutuskan menurut urutan mesin karena urutan itu bagian dari apa yang diajarkan Bu Rina: koreksi-koreksinya terdahulu dibuat di bawah node-node yang dicapai urutan mesin, dan memutuskan menurut urutan lain akan melompati koreksi itu untuk pesan-pesan yang belum diperiksa siapa pun.

Daftar Pustaka

[1] Campaign Brief Asia, "Digital 2026: Indonesia reveals social media user identities increased 26% to 180 million," Nov. 6, 2025. [Online]. Available: https://campaignbriefasia.com/2025/11/06/digital-2026-indonesia-reveals-social-media-user-identities-increased-26-to-180-million/

[2] WhatsApp, "Staying organized as a small business using lists and quick replies on WhatsApp," WhatsApp Business resource library. [Online]. Available: https://business.whatsapp.com/resources/resource-library/staying-organized-smb-using-lists-quick-replies (accessed Sep. 2026).

[3] M. M. I. Subakti, "A Variable-Centered Intelligent Rule System," M.Sc.Eng. thesis, Dept. of Computer Science and Information Engineering, Taipei, Taiwan, Jun. 2005.

[4] I. Subakti, "A variable-centered intelligent rule system," in *Proc. 1st Annual Int. Conf. Information and Communication Technology Seminar (ICTS 2005)*, vol. 1, no. 1, Surabaya, Indonesia, Aug. 2005, pp. 167–174.

[5] P. Compton and B. H. Kang, *Ripple-Down Rules: The Alternative to Machine Learning*. Boca Raton, FL, USA: CRC Press, 2021.

[6] I. Subakti, *Variable-Centered Intelligent Rule System*, Indonesian ed. (resume). Surabaya, Indonesia: Dept. of Informatics, Institut Teknologi Sepuluh Nopember, 2006.

[7] I. Subakti, "Some revisions in VCIRS and cases reconstructing perspectives," in *Proc. Information and Communication Technology Seminar (ICTS 2006)*, Surabaya, Indonesia, 2006.

[8] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*. Jakarta, Indonesia, Oct. 17, 2022. [Online]. Available: https://peraturan.bpk.go.id/Details/229798/uu-no-27-tahun-2022

Bab 8

Pencari Kerusakan Motor untuk Pengemudi Ojek: Inferensi Maju dan Mundur dengan Penjelasan

Setelah menyelesaikan bab ini Anda akan mampu menjelaskan dua arah penalaran sebuah sistem berbasis aturan — **maju**, dari apa yang terlihat menuju apa yang mengikutinya, dan **mundur**, dari sebuah dugaan menuju pertanyaan-pertanyaan yang membenarkan atau menolaknya; membangun keduanya dari nol dalam beberapa puluh baris Java dan Python; mengubah basis pengetahuan VCIRS yang dibangun kasus demi kasus menjadi aturan JIKA-MAKA biasa; dan menjalankan **pencari kerusakan**, aplikasi untuk pengemudi ojek yang lebih dulu menyingkirkan bahaya, lalu mencari penyebab masalah motor dengan pertanyaan sesedikit mungkin, dan menjawab tiga pertanyaan yang berhak diajukan setiap pengguna sistem pakar: *apa* yang kau simpulkan, *bagaimana* kau sampai ke sana, dan *mengapa* kau menanyakan hal ini kepadaku. Aplikasi ini mengenal skuter-skuter populer di jalanan Indonesia saat ini dan memberi saran yang aman: apa yang dapat dikerjakan pengemudi sendiri, apa yang perlu dibawa ke bengkel hari ini, dan kapan harus segera berhenti berkendara.

8.1 Pagi Mas Yudi

Mas Yudi pengemudi ojek daring di Surabaya. Alat kerjanya sebuah Honda BeAT, seperti banyak sekali rekannya — irit, lincah di kemacetan, dan mudah diperbaiki. Orang Indonesia membeli 6.412.769 sepeda motor baru pada 2025, dan 91,7 persen di antaranya skuter otomatis [1]; keluarga BeAT sudah bertahun-tahun menjadi model terlaris, disusul skuter seperti Vario, Scoopy, dan PCX [2]. Ketika pengemudi ojek membandingkan motor untuk bekerja pada 2026, nama-nama yang sama muncul kembali: BeAT karena iritnya, Vario 125 karena bagasinya yang lega, Yamaha Gear 125 karena pengait dan dek datarnya, Scoopy karena kenyamanannya [3].

Bagi pengemudi seperti Mas Yudi, motor yang tidak mau hidup berarti pagi tanpa penghasilan. Pukul enam, di depan tempat kosnya, motornya tidak mau hidup. Akinya? Bensinnya? Sesuatu yang mahal? Ia bisa mendorongnya ke bengkel lalu menunggu, atau menelepon Cak Man, montir yang punya bengkel kecil di sebelah pangkalan ojek dan sudah dua puluh tahun memperbaiki skuter mereka. Cak Man selalu menanyakan beberapa pertanyaan yang sama, dalam urutan yang terasa wajar begitu Anda mendengarnya: *Klaksonnya keras atau lemah? Starternya bagaimana kalau dipencet? Jarum bensinnya menunjuk apa?* Setelah tiga atau empat jawaban ia tahu apa yang rusak, dan ia tahu apakah Mas Yudi boleh mengendarainya ke bengkel, perlu menghidupkannya dengan engkol lalu datang hari ini, atau sama sekali tidak boleh mengendarainya.

Cak Man tidak dapat menjawab telepon setiap pengemudi. Ia hanya seorang dengan dua tangan, dan pada pukul tujuh pagi, ketika separuh motor pangkalan semalaman terparkir di bawah hujan, teleponnya berdering tanpa henti. Banyak telepon itu berakhir sama: *akinya, Mas, engkol saja lalu mampir nanti siang*, atau *bensinnya habis, Mas*. Telepon-telepon seperti itulah yang dapat dijawab sebuah program, sehingga Cak Man bebas untuk skuter-skuter yang sungguh memerlukan tangannya. Bab ini menaruh pertanyaan dan pengetahuannya ke dalam program yang dapat

dijalankan setiap pengemudi di ponsel atau di laptop pangkalan. Program ini sistem pakar kecil dalam arti klasik: ia bernalar dengan aturan JIKA-MAKA, ia dapat bernalar ke dua arah, dan ia menjelaskan dirinya. Berbeda dengan sistem-sistem klasik tahun 1980-an, aturan-aturannya tidak ditulis seorang pemrogram. Aturan itu dibangun VCIRS dari kasus-kasus Cak Man, dan datang bersama aturan keselamatan yang selalu ia tekankan.

8.2 Gagasannya dengan Kata-Kata Sederhana

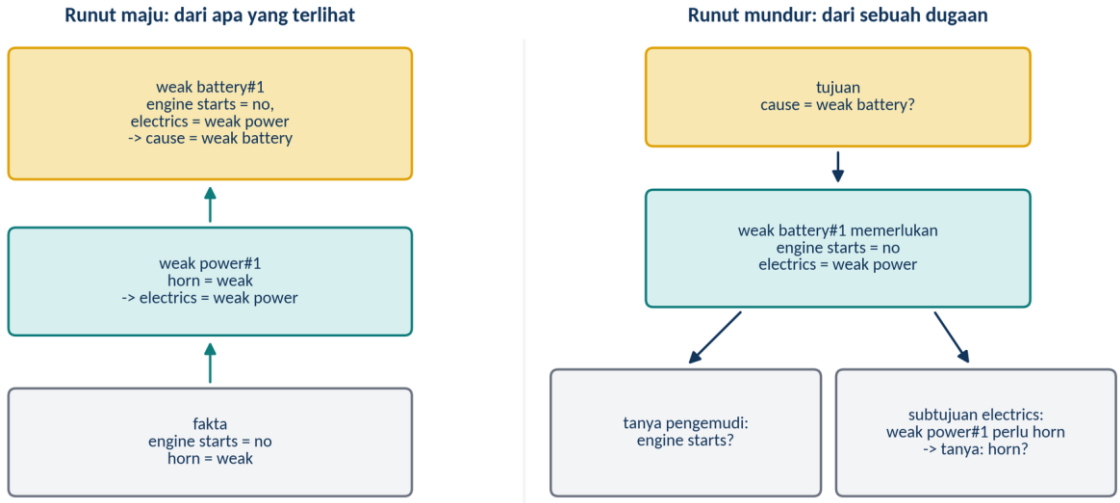
8.2.1 Dua arah penalaran

Ada dua cara alami untuk memakai sekumpulan aturan, dan keduanya setua sistem pakar itu sendiri [4], [5].

Runut maju (*forward chaining*) berangkat dari fakta. Mas Yudi menceritakan kepada program semua yang ia lihat — mesin tidak mau hidup, klakson lemah, jarum bensin menunjukkan ada bensin — dan program menyalakan setiap aturan yang semua kondisinya benar, menambahkan konklusinya ke apa yang sudah diketahui, lalu mengulanginya sampai tidak ada lagi yang baru. Ia *digerakkan data*: ia menjawab "apa yang mengikuti dari semua ini?" Inilah cara bernalar yang alami bila fakta datang lebih dulu, misalnya bila sebuah formulir sudah diisi.

Runut mundur (*backward chaining*) berangkat dari tujuan. Cak Man menduga akinya. Untuk membuktikan *cause = weak battery*, program mencari aturan-aturan yang menyimpulkannya; untuk memakai aturan seperti itu, ia harus membuktikan kondisi-kondisinya, yang masing-masing menjadi tujuan yang lebih kecil (*subtujuan*); kondisi yang tidak disimpulkan oleh aturan mana pun ditanyakan kepada pengguna. Ia *digerakkan tujuan*: ia menjawab "benarkah ini?" dan hanya menanyakan apa yang diperlukan tujuannya. Inilah cara bernalar yang alami bila bertanya itu mahal — dan bagi pengemudi yang berdiri di samping motornya di tengah hujan, setiap pertanyaan itu mahal.

Gambar 8.1 memperlihatkan keduanya pada dua aturan yang sama dari pencari kerusakan. Di kiri, fakta naik melalui aturan-aturan menuju sebuah konklusi. Di kanan, sebuah tujuan turun melalui sebuah aturan menuju satu subtujuan dan dua pertanyaan.



Gambar 8.1 Dua aturan yang sama dipakai ke dua arah. Runut maju (kiri) naik dari fakta melalui aturan-aturan menuju konklusi; runut mundur (kanan) berangkat dari tujuan, mengubah kondisi aturan menjadi subtujuan, dan hanya menanyakan kepada pengemudi apa yang tidak dapat disimpulkan aturan.

8.2.2 Aturan yang memberi makan aturan

Lihat lagi dua aturan dalam gambar itu. Yang pertama sama sekali tidak menyebut penyebab. Ia mengatakan bahwa klakson yang lemah berarti **kelistrikan** (*electrics*) dayanya lemah. Yang kedua memakai konklusi itu sebagai salah satu kondisinya: bila mesin tidak mau hidup dan kelistrikan lemah, penyebabnya aki yang lemah. Konklusi yang dipakai sebagai kondisi oleh aturan lain adalah **konklusi antara** (*intermediate conclusion*), dan dialah yang membuat perunutan lebih dari sekadar tabel pencarian.

Konklusi antara adalah cara berpikir para montir. Cak Man lebih dulu memutuskan *sistem mana* yang bermasalah — kelistrikan, bahan bakar, transmisi, pendinginan — dan baru kemudian *bagian mana*. Klakson yang lemah dan starter yang lambat sama-sama menunjuk daya listrik yang lemah; ia tidak memerlukan aturan tersendiri untuk setiap kombinasi gejala dan komponen. Pencari kerusakan memakai empat variabel antara: `electrics` (tak ada daya atau daya lemah), `transmission` (masalah CVT, transmisi otomatis skuter dengan V-belt dan koplingnya), dan `cooling` (berpendingin udara atau cairan, yang mengikuti dari model motornya). Yang keempat adalah `safety`, yang dipakai sebagai tujuan tersendiri.

8.2.3 Apa, Bagaimana, dan Mengapa

Pakar yang memberi saran tanpa alasan sulit dipercaya, apalagi sebuah program. Sejak sistem pakar pertama, sistem berbasis aturan diharapkan menjawab tiga pertanyaan selama dan sesudah inferensi [4], [6]:

- **Apa** (*What*) hasilnya? Konklusi-konklusi yang dicapai, baik akhir maupun antara.
- **Bagaimana** (*How*) hasil itu dicapai? Aturan-aturan yang menyala, dan fakta-fakta yang membuat masing-masing menyala, termasuk fakta yang sendirinya disimpulkan aturan lain.
- **Mengapa** (*Why*) pertanyaan ini diajukan? Aturan yang sedang dicoba dipakai program, kondisi yang diperlukannya, dan rantai aturan dari tujuan sampai ke pertanyaan ini.

VCIRS menyimpan catatan kejadian (*event log*) selama inferensi, dan dari catatan itulah ketiga jawaban dibaca [4]; mesin vcirs-core menyediakannya sebagai `what()`, `how()`, dan `Question.why()`. Sistem pakar kesehatan untuk diagnosis awal secara mandiri yang dibangun di atas VCIRS memakai runut mundur yang sama beserta penjelasannya bagi pasien yang menceritakan gejalanya di rumah [7]; pencari kerusakan melakukan hal yang sama untuk sepeda motor. Bagi pengemudi, *mengapa* adalah yang paling berguna dari ketiganya: ketika program bertanya "Sudah berapa km sejak V-belt diganti?", ia mungkin tidak tahu, dan mengetahui *mengapa* program bertanya memberi tahu dia apakah hal itu layak dicari tahu. Bayangkan percakapan itu dengan seorang manusia. Montir menanyakan V-belt; pengemudi bertanya "kenapa, Cak?"; montir menjawab "soalnya kalau sudah lewat dua puluh empat ribu, saya kira belt-nya aus, dan belt aus bisa putus". Jawaban itu punya tiga bagian — aturan yang sedang dicoba, kondisi yang diperlukannya, dan apa yang mengikuti bila kondisi itu terpenuhi — dan itulah persis tiga baris `Question.why()`. Bila pertanyaannya jauh di dalam rantai, program menambahkan baris keempat: rantai aturan dari tujuan sampai ke pertanyaan itu, yang sama dengan montir yang berkata "saya cek pendinginannya karena saya kira mesinnya kepanasan".

8.2.4 Seberapa yakin jawabannya?

Tidak semua aturan sama pastinya. Bahwa klakson lemah berarti daya listrik lemah hampir selalu benar; bahwa motor yang bergetar ketika mulai jalan berarti kopling CVT kotor biasanya benar, tetapi tidak selalu. Setiap node basis pengetahuan VCIRS punya **faktor kepastian** (*certainty*

factor, CF), bilangan bulat dari 0 sampai 100 yang diberikan sang pakar, dan setiap kondisi juga punya. Ketika basis pengetahuan diubah menjadi aturan, CF sebuah aturan adalah [4]:

$$CF_{\text{rule}} = \frac{\min_i CF_{\text{clause } i} \times CF_{\text{node}}}{100} \tag{8.1}$$

Ketika aturan itu menyala, kepastian konklusinya menggabungkan kepastian fakta-fakta yang dipakainya dengan kepastian aturan itu, dengan cara klasik sistem berbasis aturan [8]:

$$CF_{\text{conclusion}} = \frac{\min_i CF_{\text{fact } i} \times CF_{\text{rule}}}{100} \tag{8.2}$$

Mata rantai terlemah yang menentukan. Jawaban pengemudi bernilai CF 100. Aturan *horn = weak → electrics = weak power* ber-CF 90, sehingga menurut Persamaan (8.2) konklusi antaranya ber-CF $100 \times 90/100 = 90$. Aturan *engine starts = no, electrics = weak power → cause = weak battery* ber-CF 100 menurut Persamaan (8.1) (semua klausa dan node-nya ber-CF 100), tetapi salah satu faktanya, konklusi antara tadi, ber-CF 90, sehingga aki lemah disimpulkan dengan CF $\min(100,90) \times 100/100 = 90$. Sebuah rantai penalaran tidak pernah lebih pasti daripada langkahnya yang paling tidak pasti.

8.2.5 Dari kasus ke aturan

Di Bab 3 pengetahuan dokter Wi-Fi berupa pohon node dan pengecualian, yang dipakai dengan menelusuri pohon itu. Perunutan memerlukan hal lain: sekumpulan datar aturan JIKA-MAKA, tempat konklusi mana pun dapat menjadi kondisi aturan lain. VCIRS menjembatani keduanya dengan **transformasi basis pengetahuan** (*knowledge base transformation*) [4], [6]: setiap node menjadi satu aturan, dengan kondisi-kondisi node sebagai bagian JIKA dan konklusinya sebagai bagian MAKA; dan setiap lintasan dua node atau lebih (sebuah node dengan pengecualiannya) menjadi satu aturan lagi yang menggabungkan kondisi-kondisi di sepanjang lintasan itu. Sang pakar terus mengajarkan kasus, dengan cara RDR; mesin menyimpan aturan, dengan cara RBS. Inilah persisnya gabungan yang dirancang VCIRS: kemudahan membangun pengetahuan dari Ripple Down Rules dan kekuatan inferensi sistem berbasis aturan [4], [9].

Cak Man mengajari pencari kerusakan kasus-kasus utuh — "kalau klaksonnya lemah, kelistrikannya lemah", "kalau mesinnya tidak mau hidup dan kelistrikannya lemah, itu akinya" — dan masing-masing menjadi node tingkat puncak, sehingga setiap node menjadi tepat satu aturan. `vcirs-core` melakukan transformasi itu dengan `transform()`, dan mesin-mesin perunutan memakai aturan-aturannya.

8.2.6 Saran, bukan perbaikan

Pencari kerusakan motor harus aman sebelum pintar. Ia tidak boleh menyarankan perbaikan yang dapat mencelakakan, dan ia harus berkata jelas kapan motor tidak boleh dikendarai. Aturan-aturan Cak Man mengikuti empat tingkat kedaruratan, seperti tampak pada **Gambar 8.2**. Rem yang lembek dan bau bensin berarti *berhenti berkendara sekarang*: rem yang tidak mencengkeram dan bensin yang bocor di dekat mesin panas adalah dua kerusakan yang menyebabkan kecelakaan dan kebakaran. Aki lemah, mesin kepanasan, kampas rem aus, gangguan injeksi, atau sekring putus berarti *ke bengkel hari ini*. V-belt aus, relai starter yang rusak, atau kopling CVT yang kotor berarti *minggu ini*. Hanya tangki kosong, dan menghidupkan aki lemah dengan engkol, yang dikerjakan pengemudi sendiri. Aplikasi ini penasihat: ia menyebut

penyebab yang paling mungkin dan langkah berikut yang masuk akal, dan segala yang menyangkut membongkar mesin, rem, atau sistem bahan bakar adalah pekerjaan montir.



Gambar 8.2 Empat tingkat kedaruratan saran pencari kerusakan, yang paling mendesak di atas. Kerusakan rem dan bahan bakar menghentikan perjalanan; sebagian besar kerusakan lain dibawa ke bengkel, hari ini atau minggu ini; hanya sedikit yang dikerjakan pengemudi sendiri.

Aplikasi menegakkan tingkat teratas itu dalam strukturnya, bukan hanya dalam kata-katanya. Sebelum mencari penyebab apa pun, ia menjalankan runut mundur pada satu tujuan, *safety = stop riding now*, yang paling banyak menanyakan dua pertanyaan. Hanya bila tujuan itu tidak terbukti ia melanjutkan.

8.2.7 Arah mana, kapan?

Kedua arah itu bukan saingan; masing-masing cocok untuk keadaan berbeda, dan aplikasi yang baik memakai keduanya. Runut maju lebih tepat bila fakta datang dengan sendirinya dan tidak memakan biaya: bacaan sensor, formulir yang sudah diisi, catatan sebuah mesin. Ia menemukan *semua* yang mengikuti, termasuk hal-hal yang tidak ditanyakan siapa pun, dan itu berguna bila Anda menginginkan gambaran lengkap — montir di akhir servis yang ingin semua tanda bahaya didaftar. Runut mundur lebih tepat bila fakta harus ditanyakan kepada manusia, bila ada banyak kemungkinan pertanyaan tetapi hanya sedikit yang penting untuk satu masalah, dan bila ada dugaan yang hendak diuji. Ia berhenti begitu tujuannya terbukti, dan ia selalu dapat mengatakan mengapa ia bertanya.

Dalam praktiknya, sistem pakar mencampur keduanya. Pencari kerusakan menjalankan runut mundur dua kali — sekali untuk bahaya dan sekali untuk penyebab — dan menawarkan runut maju bagi pengemudi yang lebih suka menceritakan semuanya dulu. Pengetahuan yang sama melayani ketiga pemakaian itu tanpa diubah, karena ia disimpan sebagai aturan yang tidak tahu ke arah mana ia akan dibaca. Sifat ini patut dijaga: aturan yang ditulis sebagai “kalau klakson lemah, kelistrikan lemah” dapat dipakai maju dan mundur; aturan yang ditulis sebagai prosedur, “tanyakan dulu klaksonnya, lalu …”, hanya dapat dipakai satu arah.

8.3 Perunutan dari Nol

Sebelum memakai mesin, kita membangun kedua arah itu sendiri pada empat aturan. **Kode 8.1** adalah seluruh programnya dalam Java.

Kode 8.1 `Chain.java`: runut maju dan runut mundur dari nol, pada empat aturan pencari kerusakan.

```
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

/**
 * Runut maju dan runut mundur dari nol, pada empat aturan pencari
 * kerusakan. Sebuah aturan punya nama, kondisi, dan satu konklusi.
 */
public class Chain {
    record Rule(String name, Map<String, String> when, String var,
                String value) {
    }

    static Map<String, String> map(String... kv) {
        Map<String, String> m = new LinkedHashMap<>();
        for (int i = 0; i < kv.length; i += 2) {
            m.put(kv[i], kv[i + 1]);
        }
        return m;
    }

    static final List<Rule> RULES = List.of(
            new Rule("R1", map("horn", "weak"), "electrics", "weak power"),
            new Rule("R2", map("starter sound", "slow"), "electrics",
                    "weak power"),
            new Rule("R3", map("engine starts", "no", "electrics", "weak power"),
                    "cause", "weak battery"),
            new Rule("R4", map("engine starts", "no", "fuel gauge", "empty"),
                    "cause", "empty tank"));

    /**
     * Nyalakan setiap aturan yang semua kondisinya fakta yang diketahui,
     * berulang-ulang, sampai tidak ada lagi yang baru dapat disimpulkan.
     */
    static Map<String, String> forward(Map<String, String> given) {
        Map<String, String> facts = new LinkedHashMap<>(given);
        boolean changed = true;
        while (changed) {
            changed = false;
            for (Rule r : RULES) {
                boolean all = true;
                for (Map.Entry<String, String> c : r.when.entrySet()) {
                    all = all && c.getValue().equals(facts.get(c.getKey()));
                }
                if (!facts.containsKey(r.var) && all) {
                    facts.put(r.var, r.value);
```

Kode 8.1 (Lanjutan) *Chain.java: runut maju dan runut mundur dari nol, pada empat aturan pencari kerusakan.*

```
                    System.out.println("  " + r.name + " fires: " + r.var
                            + " = " + r.value);
                    changed = true;
                }
            }
        }
        return facts;
    }

    /**
     * Benarkah var = value? Coba aturan yang menyimpulkannya; buktikan
     * kondisinya dengan cara yang sama; tanyakan yang tak disimpulkan aturan.
     */
    static boolean backward(String var, String value, Map<String, String> facts,
                            Map<String, String> answers, int depth) {
        String pad = "  ".repeat(depth);
        if (facts.containsKey(var)) {
            return facts.get(var).equals(value);
        }
        List<Rule> rules = RULES.stream()
                .filter(r -> r.var.equals(var) && r.value.equals(value))
                .toList();
        if (rules.isEmpty()) {
            facts.put(var, answers.getOrDefault(var, "unknown"));
            System.out.println(pad + "ask " + var + ": " + facts.get(var));
            return facts.get(var).equals(value);
        }
        for (Rule r : rules) {
            System.out.println(pad + "try " + r.name + " for " + var + " = "
                    + value);
            boolean all = true;
            for (Map.Entry<String, String> c : r.when.entrySet()) {
                if (!backward(c.getKey(), c.getValue(), facts, answers,
                        depth + 1)) {
                    all = false;
                    break;
                }
            }
            if (all) {
                facts.put(var, value);
                System.out.println(pad + r.name + " proves " + var + " = "
                        + value);
                return true;
            }
        }
        return false;
```

Kode 8.1 (Lanjutan) *Chain.java: runut maju dan runut mundur dari nol, pada empat aturan pencari kerusakan.*

```
    }

    public static void main(String[] args) {
        System.out.println("Forward, from: engine starts = no, horn = weak");
        forward(map("engine starts", "no", "horn", "weak"));
        System.out.println("Backward, goal: cause = weak battery");
        Map<String, String> driver = map("engine starts", "no", "horn",
                "normal", "starter sound", "slow");
        System.out.println(backward("cause", "weak battery",
                new LinkedHashMap<>(), driver, 1) ? "  proved" : "  not proved");
    }
}
```

Sebuah `Rule` punya nama, kondisi-kondisinya (peta dari variabel ke nilai, yang semuanya harus terpenuhi), dan satu konklusi, sebuah variabel dan sebuah nilai. R1 dan R2 menyimpulkan konklusi antara `electrics = weak power` dari klakson yang lemah atau starter yang lambat; R3 menyimpulkan aki lemah; R4 tangki kosong.

Maju. `forward` menyalin fakta lalu berulang: untuk setiap aturan yang konklusinya belum diketahui dan semua kondisinya terpenuhi, ia menambahkan konklusi itu dan mencetak aturan mana yang menyala. Bila satu putaran penuh tidak menambahkan apa pun, ia berhenti. Karena konklusi baru dapat membuat aturan lain benar, perulangan berjalan sampai tidak ada perubahan: itulah *rantai*-nya.

Mundur. `backward` menjawab "apakah `var` sama dengan `value`?" Bila variabel itu sudah diketahui, ia membandingkan. Bila tidak ada aturan yang menyimpulkan tujuan itu, ia menjadi pertanyaan bagi pengemudi, yang di sini dijawab dari peta jawaban yang sudah disiapkan. Selain itu ia mencoba setiap aturan yang menyimpulkan tujuan, membuktikan kondisi-kondisinya satu demi satu dengan memanggil dirinya sendiri — setiap kondisi menjadi subtujuan, dicetak satu tingkat lebih dalam — dan berhenti pada aturan pertama yang semua kondisinya terpenuhi. **Kode 8.2** adalah versi Python-nya.

Kode 8.2 `chain.py`: kedua arah yang sama dalam Python.

```
"""Runut maju dan runut mundur dari nol, pada empat aturan pencari
kerusakan. Sebuah aturan adalah (nama, kondisi, konklusi)."""

RULES = [
    ("R1", {"horn": "weak"}, ("electrics", "weak power")),
    ("R2", {"starter sound": "slow"}, ("electrics", "weak power")),
    ("R3", {"engine starts": "no", "electrics": "weak power"},
     ("cause", "weak battery")),
    ("R4", {"engine starts": "no", "fuel gauge": "empty"},
     ("cause", "empty tank")),
]


def forward(facts):
    """Nyalakan setiap aturan yang semua kondisinya fakta yang diketahui,
    berulang-ulang, sampai tidak ada lagi yang baru dapat disimpulkan."""
    facts = dict(facts)
    changed = True
    while changed:
        changed = False
        for name, when, (var, value) in RULES:
            if var not in facts and all(facts.get(v) == x
                                        for v, x in when.items()):
                facts[var] = value
                print("  " + name + " fires: " + var + " = " + value)
                changed = True
    return facts


def backward(var, value, facts, answers, depth=1):
    """Benarkah var = value? Coba aturan yang menyimpulkannya; buktikan
    kondisinya dengan cara yang sama; tanyakan yang tak disimpulkan aturan."""
    pad = "  " * depth
    if var in facts:
        return facts[var] == value
    rules = [r for r in RULES if r[2] == (var, value)]
    if not rules:
        facts[var] = answers.get(var, "unknown")
        print(pad + "ask " + var + ": " + facts[var])
        return facts[var] == value
    for name, when, _ in rules:
        print(pad + "try " + name + " for " + var + " = " + value)
        if all(backward(v, x, facts, answers, depth + 1)
               for v, x in when.items()):
            facts[var] = value
            print(pad + name + " proves " + var + " = " + value)
```

Kode 8.2 (Lanjutan) *chain.py: kedua arah yang sama dalam Python.*

```
            return True
    return False


print("Forward, from: engine starts = no, horn = weak")
forward({"engine starts": "no", "horn": "weak"})
print("Backward, goal: cause = weak battery")
driver = {"engine starts": "no", "horn": "normal", "starter sound": "slow"}
print("  proved" if backward("cause", "weak battery", {}, driver)
      else "  not proved")
```

Keduanya mencetak:

```
Forward, from: engine starts = no, horn = weak
  R1 fires: electrics = weak power
  R3 fires: cause = weak battery
Backward, goal: cause = weak battery
  try R3 for cause = weak battery
    ask engine starts: no
    try R1 for electrics = weak power
      ask horn: normal
    try R2 for electrics = weak power
      ask starter sound: slow
    R2 proves electrics = weak power
  R3 proves cause = weak battery
  proved
```

Runut maju dari klakson yang lemah menyalakan R1 lalu R3: dua langkah naik di sepanjang rantai. Runut mundur untuk aki lemah mencoba R3, yang memerlukan `engine starts` (ditanyakan: no) dan `electrics = weak power`, sebuah subtujuan. Aturan pertama untuk subtujuan itu, R1, memerlukan klakson lemah dan gagal (klaksonnya normal); yang kedua, R2, memerlukan starter yang lambat dan berhasil. Lalu R3 terbukti. Perhatikan apa yang *tidak* ditanyakan runut mundur: jarum bensin, yang hanya diperlukan R4. Itulah seluruh keunggulannya.

Ada gunanya menelusuri satu kasus lagi dengan tangan, karena kedua arah itu dapat berbeda dalam *seberapa banyak* yang mereka pelajari. Berikan runut maju fakta `engine starts = no` dan `fuel gauge = empty` saja. Pada putaran pertama R1 dan R2 gagal (tidak ada fakta klakson maupun starter), R3 gagal (kelistrikan belum diketahui), dan R4 menyala: *empty tank*. Pada putaran kedua tidak ada yang baru. Sekarang berikan runut mundur tujuan *weak battery* untuk pengemudi yang sama, dengan klakson normal dan starter normal. Ia mencoba R3, menanyakan mesin, mencoba R1 dan R2 untuk kelistrikan, keduanya gagal, dan melaporkan *tidak terbukti* — tanpa pernah mengetahui bahwa tangkinya kosong, karena tidak ada aturan untuk tujuannya yang menyebut bensin. Runut mundur hanya selebar tujuannya. Itulah sebabnya tujuan bawaan pencari kerusakan adalah variabel `cause` tanpa nilai: “penyebab apa saja”, bukan satu penyebab tertentu.

Mesin-mesin kecil kita tidak punya tiga hal yang diperlukan aplikasi sungguhan: faktor kepastian, penjelasan yang dapat dibaca pengguna, dan urutan aturan yang mencoba yang paling pasti lebih dulu. vcirs-core menyediakan ketiganya, dengan dua arah yang sama [4].

8.4 Pengetahuan Cak Man

Pengetahuan Cak Man berupa 21 kasus dalam `data/fault-cases.json`, masing-masing dengan fakta, konklusi, CF, dan opsi 5 (node tingkat puncak baru):

```
{"facts": ["brake feel = soft"], "conclusions": ["safety = stop riding now",
  "cause = brake fluid leak or air in the brake line", "advice = do not ride;
  push it or call a bengkel that comes to you"], "cf": 100, ...}
{"facts": ["horn = weak"], "conclusions": ["electrics = weak power"], "cf": 90, ...}
{"facts": ["engine starts = no", "electrics = weak power"], "conclusions":
  ["cause = weak battery", "advice = kick-start it if it has a kick starter;
  charge or replace the battery at a bengkel today"], "cf": 100, ...}
{"facts": ["transmission = CVT problem", "km since belt change >= 24000"],
  "conclusions": ["cause = worn V-belt", "advice = replace the V-belt at a
  bengkel this week; a worn belt can snap on the road"], "cf": 90, ...}
{"facts": ["model = Yamaha NMAX"], "conclusions": ["cooling = liquid-cooled"], ...}
```

(Baris-baris di atas dipatahkan agar muat; di dalam berkas setiap kasus satu baris.) Nilai-nilai dalam basis pengetahuan ditulis dalam bahasa Inggris yang pendek, seperti di bab-bab

sebelumnya, agar Java dan Python berbagi berkas yang sama; Subbab 8.8 menunjukkan cara memberi pengemudi pertanyaan berbahasa Indonesia. **Tabel 8.1** merangkum apa yang diketahui kasus-kasus itu.

Bagaimana kasus-kasus itu dikumpulkan? Bukan dengan meminta Cak Man "menuliskan aturan-aturannya" — para pakar terkenal buruk dalam hal itu, dan usaha seperti itu biasanya menghasilkan aturan yang terlalu umum atau saling bertentangan. Sebaliknya, selama dua minggu, para pengemudi di pangkalan mencatat setiap kerusakan yang didiagnosis Cak Man: apa yang dilihat pengemudi, apa yang disimpulkan Cak Man, dan seberapa yakin ia. Setiap kasus adalah satu percakapan seperti itu, diringkas menjadi fakta-faktanya. Bila Cak Man bernalar dalam dua langkah ("klaksonnya lemah, jadi listriknya lemah; mesinnya tidak mau hidup, jadi itu akinya"), kasusnya dipecah dua, dan yang pertama menjadi konklusi antara. Inilah akuisisi pengetahuan kasus demi kasus, cara VCIRS [4], [6]: sang pakar berbicara tentang motor-motor yang nyata, tidak pernah tentang aturan secara abstrak, dan struktur pengetahuannya mengikuti dari kasus-kasus itu.

Kedua puluh satu kasus memakai opsi 5, node tingkat puncak baru, bukan diserahkan kepada `learn`. Untuk basis pengetahuan yang akan dibaca sebagai aturan, inilah pilihan yang paling jernih: setiap kasus menjadi satu node dan setiap node menjadi satu aturan, tanpa pengecualian yang maknanya bergantung pada induknya. Bila sebuah kasus bertentangan dengan kasus sebelumnya — bila Cak Man berkata "kali ini bukan akinya, karena lampunya terang" — opsi-opsi RDR dari Bab 3 tetap tersedia, dan transformasi akan mengubah pengecualian itu menjadi aturannya sendiri dengan lintasan kondisi yang lengkap.

Tabel 8.1 Apa yang diajarkan Cak Man kepada pencari kerusakan: gejala, penyebab, dan kedaruratan. Aturan tanpa CF dalam kurung ber-CF 100.

Gejala (dan konklusi antara)	Penyebab	Kedaruratan
rem terasa lembek	minyak rem bocor atau ada udara di saluran	berhenti berkendara sekarang
bau bensin yang tajam	bensin bocor	berhenti berkendara sekarang
rem berbunyi menggerus	kampas rem aus (CF 90)	bengkel hari ini
tak mau hidup; klakson mati → tak ada daya	sekring utama putus atau kabel aki kendur	bengkel hari ini bila putus lagi
tak mau hidup; klakson lemah atau starter lambat → daya lemah	aki lemah	engkol, bengkel hari ini
tak mau hidup; starter berbunyi klik; klakson normal	relai starter atau motor starter (CF 80)	engkol, minggu ini
tak mau hidup; jarum bensin kosong	tangki kosong	isi bensin
tak mau hidup; bensin ada; lampu cek mesin berkedip	gangguan injeksi atau sensor (CF 80)	bengkel hari ini
hidup; kecepatan puncak turun → masalah CVT; 24.000 km atau lebih sejak ganti belt	V-belt aus (CF 90)	minggu ini
masalah CVT; bergetar saat mulai jalan	kopling CVT kotor (CF 80)	di bengkel
lampu suhu menyala; model → berpendingin cairan	mesin kepanasan	berhenti dan dinginkan, bengkel hari ini

Aturan V-belt memakai angka. Diler Honda menganjurkan penggantian V-belt skuter otomatis pada sekitar 25.000 km [9]; Cak Man mulai mencurigainya sedikit lebih awal, pada 24.000, karena pengemudi ojek menempuh jarak jauh lebih banyak daripada kebanyakan pemilik motor dan belt yang putus membuat mereka terdampar. Kondisi `km since belt change >= 24000` dibandingkan

sebagai angka, sehingga jawaban 26000 memenuhinya dan 18000 tidak. Enam kasus model memberi tahu mesin skuter populer mana yang berpendingin udara (BeAT, Scoopy, Gear 125) dan mana yang punya radiator (Vario 125, Vario 160, NMAX); hanya mesin berpendingin cairan yang punya lampu peringatan suhu cairan pendingin dan tabung cadangan cairan pendingin untuk diperiksa. Mengapa menanyakan model dan bukan langsung "mesin Anda berpendingin cairan?" Karena setiap pengemudi tahu modelnya dan hanya sedikit yang tahu bagaimana mesinnya didinginkan. Pertanyaan yang baik menanyakan sesuatu yang dapat dilihat atau diketahui pengguna dengan pasti, dan menyerahkan kesimpulan teknisnya kepada aturan. Gagasan yang sama melandasi pertanyaan tentang kerasnya klakson, alih-alih tentang tegangan aki.

Perintah `rules` mencetak aturan-aturan yang dihasilkan transformasi dari kasus-kasus itu. Dua puluh baris pertamanya:

```
stop riding now#1 [RULE CF=100]
  IF   brake feel = soft [CF=100]
  THEN safety = stop riding now
  AND  cause = brake fluid leak or air in the brake line
  AND  advice = do not ride; push it or call a bengkel that comes to you
stop riding now#2 [RULE CF=100]
  IF   fuel smell = yes [CF=100]
  THEN safety = stop riding now
  AND  cause = fuel leak
  AND  advice = switch off, keep fire and cigarettes away, do not ride; call a bengkel
worn brake pads#1 [RULE CF=90]
  IF   brake feel = grinding [CF=100]
  THEN cause = worn brake pads
  AND  advice = ride slowly to the nearest bengkel today and replace the pads
no power#1 [RULE CF=90]
  IF   horn = none [CF=100]
  THEN electrics = no power
weak power#1 [RULE CF=90]
  IF   horn = weak [CF=100]
  THEN electrics = weak power
```

Setiap node menjadi satu aturan dengan CF-nya: `[RULE CF=90]` milik `weak power#1` berasal dari CF 90 kasusnya dan Persamaan (8.1). Nama setiap aturan adalah identitas node-nya, yang dibuat dari nilai konklusi pertama, sehingga tiga aturan kelistrikan bernama `no power#1`, `weak power#1`, dan `weak power#2`.

8.5 Pencari Kerusakan

Kode 8.3 adalah aplikasinya.

Kode 8.3 `FaultFinder.java`: pencari kerusakan dengan cek keselamatannya dan tiga cara bernalar.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import vcirs.CaseFile;
import vcirs.ChainResult;
import vcirs.Conclusion;
import vcirs.Fact;
import vcirs.KnowledgeBase;
import vcirs.Question;
import vcirs.RbsRule;
import vcirs.Report;

/**
 * Pencari kerusakan motor untuk pengemudi ojek: runut maju dan mundur
 * dengan penjelasan, di atas vcirs-core.
 *   java FaultFinder [--echo]                cek keselamatan, lalu cari
 *                                            penyebab (runut mundur)
 *   java FaultFinder diagnose [--echo]       ceritakan semuanya dulu
 *                                            (runut maju)
 *   java FaultFinder check "cause = X" [--echo]   benarkah X?
 *   java FaultFinder rules                   aturan-aturan penalarannya
 * Tambahkan nama berkas seperti fault-cases-plus.json untuk kasus lain,
 * dan --log untuk melihat setiap langkah pencarian penyebab.
 * Ketik "why" pada pertanyaan mana pun untuk tahu alasannya; tekan Enter
 * bila tidak tahu. Pencari memberi saran, bukan perbaikan: rem, bocor
 * bensin, dan apa pun yang tak dapat dinamainya dibawa ke bengkel.
 */
public class FaultFinder {
    static final String DATA = "../data";
    static final String DANGER = "safety = stop riding now";
    static final String[][] QUESTIONS = {
        {"brake feel", "How do the brakes feel when you squeeze the levers?",
            "normal", "soft", "grinding"},
        {"fuel smell", "Is there a strong smell of petrol around the motorcycle?",
            "yes", "no"},
        {"engine starts", "Does the engine start?", "yes", "no"},
        {"starter sound", "What happens when you press the starter button?",
            "none", "click", "slow", "normal"},
```

Kode 8.3 (Lanjutan) *FaultFinder.java: pencari kerusakan dengan cek keselamatannya dan tiga cara bernalar.*

```
        {"horn", "How loud is the horn?", "normal", "weak", "none"},
        {"fuel gauge", "What does the fuel gauge show?", "empty", "ok"},
        {"engine check light", "Is the engine check light blinking?",
            "yes", "no"},
        {"top speed", "Can it still reach its usual speed?", "normal", "low"},
        {"km since belt change",
            "How many km since the V-belt was changed? (a number)"},
        {"shudder", "Does it shudder when you pull away?", "yes", "no"},
        {"temperature light", "Is the temperature warning light on?",
            "yes", "no"},
        {"model", "Which model is it?", "Honda BeAT", "Honda Scoopy",
            "Yamaha Gear 125", "Honda Vario 125", "Honda Vario 160",
            "Yamaha NMAX"},
    };
    static final List<String> CHECKLIST = List.of("model", "engine starts",
            "starter sound", "horn", "fuel gauge", "engine check light",
            "top speed", "temperature light");

    static KnowledgeBase openKb(String file) {
        CaseFile cases = CaseFile.read(DATA + "/" + file);
        KnowledgeBase kb = new KnowledgeBase(cases.name, cases.description);
        for (String[] q : QUESTIONS) {
            List<String> choices = new ArrayList<>();
            for (int i = 2; i < q.length; i++) {
                choices.add(q[i]);
            }
            kb.declare(q[0], q[1], choices);
        }
        return cases.build(kb);
    }

    /**
     * Bertanya kepada pengemudi; menjawab "why" dengan alasan mesin. Ia
     * tak menanyakan yang dapat disimpulkan aturan (seperti "electrics").
     */
    static class Driver {
        final KnowledgeBase kb;
        final boolean echo;
        int count = 0;
        final Set<String> derived = new HashSet<>();
        final Set<String> skipped = new HashSet<>();  // dikosongkan di daftar
        boolean log = false;                          // tiap langkah (--log)
        final BufferedReader in = new BufferedReader(
                new InputStreamReader(System.in, StandardCharsets.UTF_8));
```

Kode 8.3 (Lanjutan) *FaultFinder.java: pencari kerusakan dengan cek keselamatannya dan tiga cara bernalar.*

```
        Driver(KnowledgeBase kb, boolean echo) {
            this.kb = kb;
            this.echo = echo;
            for (RbsRule r : kb.transform()) {
                for (Conclusion c : r.conclusions) {
                    derived.add(c.variable);
                }
            }
        }

        String read(String prompt) {
            System.out.print(prompt);
            String line;
            try {
                line = in.readLine();
            } catch (IOException e) {
                line = null;
            }
            String text = line == null ? "" : line.strip();
            if (echo) {
                System.out.println(text);
            }
            return text;
        }

        Fact ask(Question q) {
            String var = q.variable;
            if (derived.contains(var) || skipped.contains(var)) {
                return null;                // disimpulkan aturan, atau dilewati
            }
            List<String> choices = kb.variables.get(var).choices;
            String hint = choices.isEmpty() ? ""
                    : " (" + String.join("/", choices) + ")";
            String answer;
            while (true) {
                answer = read(kb.variables.get(var).question + hint + " > ");
                if (!answer.equals("why")) {
                    break;
                }
                for (String line : q.why()) {
                    System.out.println("  " + line);
                }
            }
            count++;
            return answer.isEmpty() ? null : new Fact(var, answer);
        }
```

Kode 8.3 (Lanjutan) *FaultFinder.java: pencari kerusakan dengan cek keselamatannya dan tiga cara bernalar.*

```
    }

    static void show(ChainResult result, String what) {
        for (String line : result.what()) {
            System.out.println(line);
        }
        System.out.println(what);
        for (String line : result.how()) {
            System.out.println("  " + line);
        }
    }

    /** Runut mundur pada satu tujuan yang harus disingkirkan dulu. */
    static Map<String, Fact> safetyFirst(KnowledgeBase kb, Driver driver) {
        System.out.println("Safety first.");
        ChainResult result = kb.backward(DANGER, new LinkedHashMap<>(),
                driver::ask);
        if (result.proved) {
            System.out.println("STOP: do not ride this motorcycle now.");
            show(result, "How I know:");
            return null;
        }
        return new LinkedHashMap<>(result.memory.facts);
    }

    /** Runut mundur pada "cause": hanya pertanyaan yang diperlukan. */
    static void find(KnowledgeBase kb, Driver driver) {
        Map<String, Fact> facts = safetyFirst(kb, driver);
        if (facts == null) {
            return;
        }
        System.out.println("Looking for the cause.");
        ChainResult result = kb.backward("cause", facts, driver::ask);
        if (!result.proved) {
            System.out.println("I cannot name the fault. Take it to a bengkel "
                    + "and tell the mechanic what you told me.");
        }
        show(result, "How I know:");
        System.out.println("Questions asked: " + driver.count);
        if (driver.log) {
            System.out.println("Every step:");
            for (String line : result.log.lines()) {
                System.out.println("  " + line);
            }
        }
    }
```

Kode 8.3 (Lanjutan) *FaultFinder.java: pencari kerusakan dengan cek keselamatannya dan tiga cara bernalar.*

```
    /** Runut maju: pengemudi menceritakan semuanya dulu. */
    static void diagnose(KnowledgeBase kb, Driver driver) {
        Map<String, Fact> facts = safetyFirst(kb, driver);
        if (facts == null) {
            return;
        }
        System.out.println("Describe the motorcycle (Enter to skip a line).");
        for (String var : CHECKLIST) {
            driver.count++;
            String answer = driver.read(kb.variables.get(var).question + " > ");
            if (!answer.isEmpty()) {
                facts.put(var, new Fact(var, answer));
            } else {
                driver.skipped.add(var);
            }
        }
        ChainResult result = kb.forward(facts, driver::ask);
        show(result, "How I know:");
        System.out.println("Questions asked: " + driver.count);
    }

    /** Runut mundur pada satu penyebab yang diduga montir. */
    static void check(KnowledgeBase kb, Driver driver, String goal) {
        Map<String, Fact> facts = safetyFirst(kb, driver);
        if (facts == null) {
            return;
        }
        System.out.println("Checking: " + goal);
        ChainResult result = kb.backward(goal, facts, driver::ask);
        System.out.println(result.proved ? "Proved."
                : "Not proved: look for another cause.");
        show(result, "How:");
        System.out.println("Questions asked: " + driver.count);
    }
```

Kode 8.3 (Lanjutan) *FaultFinder.java: pencari kerusakan dengan cek keselamatannya dan tiga cara bernalar.*

```
    public static void main(String[] args) {
        boolean echo = false;
        boolean log = false;
        String file = "fault-cases.json";
        List<String> words = new ArrayList<>();
        for (String a : args) {
            if (a.equals("--echo")) {
                echo = true;
            } else if (a.equals("--log")) {
                log = true;
            } else if (a.endsWith(".json")) {
                file = a;
            } else {
                words.add(a);
            }
        }
        KnowledgeBase kb = openKb(file);
        Driver driver = new Driver(kb, echo);
        driver.log = log;
        String command = words.isEmpty() ? "find" : words.get(0);
        if (command.equals("rules")) {
            for (String line : Report.transformed(kb)) {
                System.out.println(line);
            }
        } else if (command.equals("check")) {
            check(kb, driver, words.get(1));
        } else if (command.equals("diagnose")) {
            diagnose(kb, driver);
        } else {
            find(kb, driver);
        }
    }
}
```

Bacalah dari bawah. `main` membangun basis pengetahuan dari berkas kasus (nama berkas lain yang berakhiran `.json` boleh diberikan) dan menjalankan salah satu dari empat perintah.

Pengemudi. `Driver` adalah penanya yang dipanggil mesin-mesin ketika memerlukan fakta. Ia mencetak pertanyaan yang dideklarasikan beserta jawaban-jawaban yang biasa, membaca balasannya, dan menghitung pertanyaan. Bila balasannya `why`, ia mencetak alasan mesin, `Question.why()`, lalu bertanya lagi. Dua jenis pertanyaan tidak pernah diajukan kepada pengemudi: variabel yang disimpulkan oleh suatu aturan (`electrics`, `transmission`, `cooling`, `cause`, …), karena aturan dapat menyimpulkannya dan pengemudi tidak dapat diharapkan tahu apa itu “weak power”; dan baris yang ia biarkan kosong dalam daftar periksa, karena ia sudah menyatakan tidak tahu. Himpunan `derived` dihitung sekali dari aturan-aturan `kb.transform()`.

Keselamatan dulu. `safetyFirst` menjalankan `kb.backward(DANGER, …)` dengan tujuan *safety = stop riding now*. Dua aturan menyimpulkannya; runut mundur mencoba keduanya dan menanyakan rem serta bau bensin. Bila tujuan itu terbukti, aplikasi mencetak STOP beserta penjelasannya lalu selesai. Bila tidak, ia mengembalikan fakta-fakta yang sudah dipelajarinya, sehingga tidak ada yang ditanyakan dua kali.

Cari. `find`, perintah bawaan, menjalankan runut mundur pada tujuan `cause` tanpa nilai: “temukan penyebab apa saja”. Mesin mencoba setiap aturan yang menyimpulkan penyebab, yang paling pasti lebih dulu, dan berhenti pada yang pertama dapat dibuktikannya — hanya menanyakan apa yang diperlukan setiap percobaan.

Diagnosis. `diagnose` adalah runut maju. Pengemudi lebih dulu menjawab daftar periksa berisi delapan pengamatan mudah (model, apakah mesin hidup, starter, klakson, jarum bensin, lampu cek mesin, kecepatan puncak, dan lampu suhu); lalu `kb.forward` menyalakan semua yang mengikuti, dan masih dapat menanyakan fakta yang diperlukan suatu aturan, seperti jarak tempuh sejak belt diganti.

Periksa. `check "cause = worn V-belt"` adalah runut mundur pada sebuah dugaan, cara Cak Man bekerja ketika seorang pengemudi berkata “kayaknya belt-nya, Cak”.

Setiap perintah diakhiri dengan What (`result.what()`) dan How (`result.how()`) dari mesin; dengan `--log`, `find` juga mencetak setiap langkah pencariannya (Subbab 8.7.2). **Kode 8.4** adalah versi Python-nya; basis pengetahuan, pertanyaan, dan keluarannya sama.

Kode 8.4 `fault_finder.py`: aplikasi yang sama dalam Python.

```
"""Pencari kerusakan motor untuk pengemudi ojek: runut maju dan mundur
dengan penjelasan, di atas vcirs-core.

    python fault_finder.py [--echo]              cek keselamatan, lalu cari
                                                 penyebab (runut mundur)
    python fault_finder.py diagnose [--echo]     ceritakan semuanya dulu
                                                 (runut maju)
    python fault_finder.py check "cause = X" [--echo]   benarkah X?
    python fault_finder.py rules                 aturan-aturan penalarannya
    Tambahkan nama berkas seperti fault-cases-plus.json untuk kasus lain,
    dan --log untuk melihat setiap langkah pencarian penyebab.

Ketik "why" pada pertanyaan mana pun untuk tahu alasannya; tekan Enter bila
tidak tahu. Pencari memberi saran, bukan perbaikan: rem, bocor bensin, dan
apa pun yang tak dapat dinamainya dibawa ke bengkel.
"""

import os
import pathlib
import sys

# Beri tahu Python letak mesin vcirs-core (naik dua folder, lalu masuk).
sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import KnowledgeBase, casefile, report

DATA = os.path.join(os.path.dirname(os.path.abspath(__file__)), "..", "data")
DANGER = "safety = stop riding now"
QUESTIONS = [
    ("brake feel", "How do the brakes feel when you squeeze the levers?",
     ["normal", "soft", "grinding"]),
    ("fuel smell", "Is there a strong smell of petrol around the motorcycle?",
     ["yes", "no"]),
    ("engine starts", "Does the engine start?", ["yes", "no"]),
    ("starter sound", "What happens when you press the starter button?",
     ["none", "click", "slow", "normal"]),
    ("horn", "How loud is the horn?", ["normal", "weak", "none"]),
    ("fuel gauge", "What does the fuel gauge show?", ["empty", "ok"]),
    ("engine check light", "Is the engine check light blinking?",
     ["yes", "no"]),
    ("top speed", "Can it still reach its usual speed?", ["normal", "low"]),
    ("km since belt change",
     "How many km since the V-belt was changed? (a number)", []),
    ("shudder", "Does it shudder when you pull away?", ["yes", "no"]),
    ("temperature light", "Is the temperature warning light on?",
     ["yes", "no"]),
```

Kode 8.4 (Lanjutan) *fault_finder.py: aplikasi yang sama dalam Python.*

```
    ("model", "Which model is it?",
     ["Honda BeAT", "Honda Scoopy", "Yamaha Gear 125", "Honda Vario 125",
      "Honda Vario 160", "Yamaha NMAX"]),
]
CHECKLIST = ["model", "engine starts", "starter sound", "horn", "fuel gauge",
             "engine check light", "top speed", "temperature light"]


def open_kb(cases):
    name, description, steps = casefile.read_cases(os.path.join(DATA, cases))
    kb = KnowledgeBase(name, description)
    for var, question, choices in QUESTIONS:
        kb.declare(var, question, choices)
    return casefile.build(kb, steps)


class Driver:
    """Bertanya kepada pengemudi; menjawab 'why' dengan alasan mesin. Ia
    tak menanyakan yang dapat disimpulkan aturan (seperti 'electrics')."""

    def __init__(self, kb, echo):
        self.kb = kb
        self.echo = echo
        self.count = 0
        self.derived = {c.variable for r in kb.transform()
                        for c in r.conclusions}
        self.skipped = set()              # dikosongkan di daftar periksa
        self.log = False                  # cetak tiap langkah (--log)

    def read(self, prompt):
        print(prompt, end="")
        text = sys.stdin.readline().strip()
        if self.echo:
            print(text)
        return text
```

Kode 8.4 (Lanjutan) *fault_finder.py: aplikasi yang sama dalam Python.*

```
    def ask(self, question):
        var = question.variable
        if var in self.derived or var in self.skipped:
            return None                    # disimpulkan aturan, atau dilewati
        choices = self.kb.variables[var].choices
        hint = " (" + "/".join(choices) + ")" if choices else ""
        while True:
            answer = self.read(self.kb.variables[var].question + hint
                               + " > ")
            if answer != "why":
                break
            for line in question.why():
                print("  " + line)
        self.count += 1
        return answer or None


def show(result, what):
    for line in result.what():
        print(line)
    print(what)
    for line in result.how():
        print("  " + line)


def safety_first(kb, driver):
    """Runut mundur pada satu tujuan yang harus disingkirkan dulu."""
    print("Safety first.")
    result = kb.backward(DANGER, {}, driver.ask)
    if result.proved:
        print("STOP: do not ride this motorcycle now.")
        show(result, "How I know:")
        return None
    return result.memory.facts
```

Kode 8.4 (Lanjutan) *fault_finder.py: aplikasi yang sama dalam Python.*

```
def find(kb, driver):
    """Runut mundur pada 'cause': hanya pertanyaan yang diperlukan."""
    facts = safety_first(kb, driver)
    if facts is None:
        return
    print("Looking for the cause.")
    result = kb.backward("cause", facts, driver.ask)
    if not result.proved:
        print("I cannot name the fault. Take it to a bengkel and tell the "
              "mechanic what you told me.")
    show(result, "How I know:")
    print("Questions asked: " + str(driver.count))
    if driver.log:
        print("Every step:")
        for line in result.log.lines():
            print("  " + line)


def diagnose(kb, driver):
    """Runut maju: pengemudi menceritakan semuanya dulu."""
    facts = safety_first(kb, driver)
    if facts is None:
        return
    facts = dict(facts)
    print("Describe the motorcycle (Enter to skip a line).")
    for var in CHECKLIST:
        driver.count += 1
        answer = driver.read(kb.variables[var].question + " > ")
        if answer:
            facts[var] = answer
        else:
            driver.skipped.add(var)
    result = kb.forward(facts, driver.ask)
    show(result, "How I know:")
    print("Questions asked: " + str(driver.count))
```

Kode 8.4 (Lanjutan) *fault_finder.py: aplikasi yang sama dalam Python.*

```
def check(kb, driver, goal):
    """Runut mundur pada satu penyebab yang diduga montir."""
    facts = safety_first(kb, driver)
    if facts is None:
        return
    print("Checking: " + goal)
    result = kb.backward(goal, facts, driver.ask)
    print("Proved." if result.proved else "Not proved: look for another "
          "cause.")
    show(result, "How:")
    print("Questions asked: " + str(driver.count))


def main(args):
    echo = "--echo" in args
    files = [a for a in args if a.endswith(".json")]
    words = [a for a in args if a != "--echo" and a not in files]
    kb = open_kb(files[0] if files else "fault-cases.json")
    driver = Driver(kb, echo)
    driver.log = "--log" in args
    words = [w for w in words if w != "--log"]
    if words and words[0] == "rules":
        for line in report.transformed(kb):
            print(line)
    elif words and words[0] == "check":
        check(kb, driver, words[1])
    elif words and words[0] == "diagnose":
        diagnose(kb, driver)
    else:
        find(kb, driver)


if __name__ == "__main__":
    main(sys.argv[1:])
```

Folder data menyimpan jawaban sesi-sesi dalam bab ini, satu per baris, sehingga masing-masing dapat diputar ulang dengan `--echo`, persis seperti di Bab 3: dari `Chapter08/java`, kompilasi dengan `javac -d out -sourcepath ../../vcirs-core/java/src *.java` lalu jalankan, misalnya, `java -cp out FaultFinder --echo < ../data/find-beat.txt`; dari `Chapter08/python`, `python fault_finder.py --echo < ../data/find-beat.txt`.

8.6 Pagi-Pagi di Pangkalan

8.6.1 BeAT Mas Yudi tidak mau hidup

Pukul enam, di depan tempat kos. Mas Yudi menjalankan pencari kerusakan:

```
Safety first.
How do the brakes feel when you squeeze the levers? (normal/soft/grinding) > normal
Is there a strong smell of petrol around the motorcycle? (yes/no) > no
Looking for the cause.
Does the engine start? (yes/no) > no
How loud is the horn? (normal/weak/none) > weak
Result: electrics = weak power  (rule weak power#1, CF 90)
Result: cause = weak battery  (rule weak battery#1, CF 90)
Result: advice = kick-start it if it has a kick starter; charge or replace the battery at a
bengkel today  (rule weak battery#1, CF 90)
How I know:
  weak power#1 fired because:
    horn = weak  (answered by the user, CF 100)
    so electrics = weak power  [rule CF 90, conclusion CF 90]
  weak battery#1 fired because:
    engine starts = no  (answered by the user, CF 100)
    electrics = weak power  (concluded by weak power#1, CF 90)
    so cause = weak battery, advice = kick-start it if it has a kick starter; charge or
replace the battery at a bengkel today  [rule CF 100, conclusion CF 90]
Questions asked: 4
```

Empat pertanyaan. Dua yang pertama adalah cek keselamatan: remnya normal dan tidak ada bau bensin, jadi tujuan *stop riding now* tidak terbukti. Lalu runut mundur mencari penyebab. Kedua aturan keselamatan juga menyimpulkan penyebab, tetapi fakta-faktanya sudah diketahui dan salah, sehingga keduanya gagal tanpa satu pertanyaan pun. Aturan berikutnya menurut kepastian adalah *blown main fuse*, yang memerlukan `engine starts = no` (ditanyakan: no) dan `electrics = no power`, sebuah subtujuan. Untuk membuktikan apa pun tentang `electrics`, mesin mencoba aturan-aturan yang menyimpulkannya: `no power#1` memerlukan klakson yang mati sama sekali (ditanyakan: klaksonnya lemah — gagal), dan `weak power#1` memerlukan klakson lemah, yang terpenuhi, sehingga `electrics = weak power` disimpulkan dengan CF 90. Aturan sekring tetap gagal, karena kelistrikannya lemah, bukan mati. Aturan berikutnya, *weak battery*, memerlukan persis apa yang kini diketahui, dan menyala dengan CF 90, seperti diramalkan Persamaan (8.2) di Subbab 8.2.4.

Penjelasan How memperlihatkan rantainya: klakson membuat kelistrikan lemah, dan kelistrikan yang lemah dengan mesin yang mati membuat aki menjadi penyebabnya. Sarannya memberi tahu Mas Yudi apa yang dikerjakan *sekarang* (engkol, karena BeAT punya engkol) dan *hari ini* (isi ulang atau ganti aki). Pukul enam lewat seperempat ia sudah di jalan.

8.6.2 Pagi yang sama, secara maju

Bagaimana bila Mas Yudi menceritakan semuanya lebih dulu? `diagnose` melakukannya:

```
Safety first.
How do the brakes feel when you squeeze the levers? (normal/soft/grinding) > normal
Is there a strong smell of petrol around the motorcycle? (yes/no) > no
Describe the motorcycle (Enter to skip a line).
Which model is it? > Honda BeAT
Does the engine start? > no
What happens when you press the starter button? > slow
How loud is the horn? > weak
What does the fuel gauge show? > ok
Is the engine check light blinking? > no
Can it still reach its usual speed? >
Is the temperature warning light on? >
Result: cooling = air-cooled  (rule air-cooled#1, CF 100)
Result: electrics = weak power  (rule weak power#2, CF 80)
Result: cause = weak battery  (rule weak battery#1, CF 80)
Result: advice = kick-start it if it has a kick starter; charge or replace the battery at a
bengkel today  (rule weak battery#1, CF 80)
How I know:
  air-cooled#1 fired because:
    model = Honda BeAT  (given, CF 100)
    so cooling = air-cooled  [rule CF 100, conclusion CF 100]
  weak power#2 fired because:
    starter sound = slow  (given, CF 100)
    so electrics = weak power  [rule CF 80, conclusion CF 80]
  weak power#1 fired because:
    horn = weak  (given, CF 100)
    so electrics = weak power  [rule CF 90, conclusion CF 90]
  weak battery#1 fired because:
    engine starts = no  (given, CF 100)
    electrics = weak power  (concluded by weak power#2, CF 80)
    so cause = weak battery, advice = kick-start it if it has a kick starter; charge or
replace the battery at a bengkel today  [rule CF 100, conclusion CF 80]
Questions asked: 10
```

Sepuluh pertanyaan, bukan empat, dan penyebabnya sama. Runut maju tidak dapat tahu sebelumnya pengamatan mana yang penting, jadi ia mengumpulkan semuanya, lalu menurunkan semua yang tersirat — termasuk `cooling = air-cooled` dari model, yang tidak ditanyakan siapa pun dan tidak penting hari ini. Itulah watak penalaran yang digerakkan data: teliti, tetapi tidak hemat waktu pengguna. Meski begitu, runut maju punya tempatnya. Di akhir servis rutin, Cak Man dapat mengisi daftar periksa untuk setiap motor sebagai kebiasaan — model, starter, klakson, kecepatan puncak, kilometer sejak ganti belt — dan membiarkan runut maju mendaftar *setiap* peringatan yang dapat diturunkan basis pengetahuan, termasuk belt yang sudah waktunya diganti sebelum ada gejala apa pun. Di sana fakta tidak memakan biaya, karena montir memang sedang memeriksa motornya, dan kelengkapan itulah yang ia inginkan.

Dua perincian patut dibaca cermat. Pertama, urutan penjelasan How mengikuti antrean variabel mesin: model datang pertama dalam daftar periksa, sehingga aturannya menyala pertama; aturan starter `weak power#2` (CF 80) menyala sebelum aturan klakson `weak power#1` (CF 90), karena *starter sound* datang sebelum *horn* dalam daftar periksa. Kedua, `electrics` disimpulkan sekali, oleh aturan pertama yang menyala, dengan CF 80; ketika `weak power#1` menyala sesaat kemudian, mesin mencatat bahwa nilainya sudah diketahui dan tidak mengubahnya. Jadi aki lemah disimpulkan dengan CF 80 di sini, dan dengan CF 90 pada sesi mundur. Mesin mengambil konklusi pertama yang dicapainya; ia tidak menjumlahkan bukti dari dua aturan. Subbab 8.7 kembali ke hal ini.

8.6.3 Vario 160 Pak Heru kehilangan tenaga

Pak Heru mengendarai Vario 160. Seminggu ini motornya terasa loyo di tanjakan, dan kecepatan puncaknya turun. Cak Man menduga V-belt dan memeriksanya dengan `check "cause = worn V-belt"`. Pada pertanyaan tentang kilometer, Pak Heru mengetik `why`:

```
Safety first.
How do the brakes feel when you squeeze the levers? (normal/soft/grinding) > normal
Is there a strong smell of petrol around the motorcycle? (yes/no) > no
Checking: cause = worn V-belt
Does the engine start? (yes/no) > yes
Can it still reach its usual speed? (normal/low) > low
How many km since the V-belt was changed? (a number) > why
  I need to know 'km since belt change'.
  Rule worn V-belt#1 needs km since belt change >= 24000;
  if it holds, the rule concludes cause = worn V-belt, advice = replace the V-belt at a
bengkel this week; a worn belt can snap on the road.
How many km since the V-belt was changed? (a number) > 26000
Proved.
Result: transmission = CVT problem  (rule CVT problem#1, CF 90)
Result: cause = worn V-belt  (rule worn V-belt#1, CF 81)
Result: advice = replace the V-belt at a bengkel this week; a worn belt can snap on the
road  (rule worn V-belt#1, CF 81)
How:
  CVT problem#1 fired because:
    engine starts = yes  (answered by the user, CF 100)
    top speed = low  (answered by the user, CF 100)
    so transmission = CVT problem  [rule CF 90, conclusion CF 90]
  worn V-belt#1 fired because:
    transmission = CVT problem  (concluded by CVT problem#1, CF 90)
    km since belt change >= 24000  (km since belt change = 26000, answered by the user, CF
100)
    so cause = worn V-belt, advice = replace the V-belt at a bengkel this week; a worn belt
can snap on the road  [rule CF 90, conclusion CF 81]
Questions asked: 5
```

Jawaban *why* menyebut aturannya, kondisi persisnya, dan apa yang akan disimpulkan aturan itu. Pak Heru menemukan buku servis di bawah jok: belt-nya diganti 26.000 km yang lalu. Tujuan terbukti melalui konklusi antara `transmission = CVT problem`, dengan CF 81: aturan CVT ber-CF 90, aturan V-belt ber-CF 90, dan $90 \times 90/100 = 81$ menurut Persamaan (8.2). Sarannya *minggu ini*: belt yang aus dapat putus, dan pada Vario 160 yang membonceng penumpang, itu cara berhenti yang berbahaya.

Andaikan buku servis itu menyebut 18.000 km. Pemeriksaan yang sama berakhir lain:

```
Safety first.
How do the brakes feel when you squeeze the levers? (normal/soft/grinding) > normal
Is there a strong smell of petrol around the motorcycle? (yes/no) > no
Checking: cause = worn V-belt
Does the engine start? (yes/no) > yes
Can it still reach its usual speed? (normal/low) > low
How many km since the V-belt was changed? (a number) > 18000
Not proved: look for another cause.
Result: transmission = CVT problem  (rule CVT problem#1, CF 90)
How:
  CVT problem#1 fired because:
    engine starts = yes  (answered by the user, CF 100)
    top speed = low  (answered by the user, CF 100)
    so transmission = CVT problem  [rule CF 90, conclusion CF 90]
Questions asked: 5
```

Konklusi antaranya tetap berlaku — ada yang tidak beres di CVT — tetapi belt-nya terlalu muda untuk aturan V-belt, sehingga tujuan tidak terbukti, dan pencari kerusakan mengatakannya: *look*

for another cause, carilah penyebab lain. Ia tidak diam-diam berpindah ke tujuan lain; sebuah pemeriksaan menjawab pertanyaan yang diajukan kepadanya. Pak Heru kini dapat menjalankan pencari kerusakan tanpa tujuan, dan runut mundur pada `cause` akan mencoba kopling yang kotor berikutnya dan menanyakan apakah skuternya bergetar. Jawaban "tidak" dari sebuah pemeriksaan sama bergunanya dengan "ya": ia menyingkirkan komponen yang mahal dan menunjuk yang murah.

8.6.4 Lampu peringatan pada NMAX

Mbak Wulan mengendarai Yamaha NMAX untuk aplikasi pesan-antar makanan. Di tengah kemacetan siang, sebuah lampu peringatan menyala, dan ia menjalankan pencari kerusakan. Pada pertanyaan tentang model ia bertanya mengapa:

```
Safety first.
How do the brakes feel when you squeeze the levers? (normal/soft/grinding) > normal
Is there a strong smell of petrol around the motorcycle? (yes/no) > no
Looking for the cause.
Does the engine start? (yes/no) > yes
Is the temperature warning light on? (yes/no) > yes
Which model is it? (Honda BeAT/Honda Scoopy/Yamaha Gear 125/Honda Vario 125/Honda Vario
160/Yamaha NMAX) > why
  I need to know 'model'.
  Rule air-cooled#1 needs model = Honda BeAT;
  if it holds, the rule concludes cooling = air-cooled.
  Chain of rules from the goal: overheating#1 > air-cooled#1.
Which model is it? (Honda BeAT/Honda Scoopy/Yamaha Gear 125/Honda Vario 125/Honda Vario
160/Yamaha NMAX) > Yamaha NMAX
Result: cooling = liquid-cooled  (rule liquid-cooled#3, CF 100)
Result: cause = overheating  (rule overheating#1, CF 100)
Result: advice = stop in a safe place and let the engine cool; check the coolant reserve
tank only when cold; go to a bengkel today  (rule overheating#1, CF 100)
How I know:
  liquid-cooled#3 fired because:
    model = Yamaha NMAX  (answered by the user, CF 100)
    so cooling = liquid-cooled  [rule CF 100, conclusion CF 100]
  overheating#1 fired because:
    temperature light = yes  (answered by the user, CF 100)
    cooling = liquid-cooled  (concluded by liquid-cooled#3, CF 100)
    so cause = overheating, advice = stop in a safe place and let the engine cool; check
the coolant reserve tank only when cold; go to a bengkel today  [rule CF 100, conclusion CF
100]
Questions asked: 5
```

Jawaban *why* memperlihatkan **rantai aturan dari tujuan**: mesin sedang mencoba membuktikan `overheating#1`, yang perlu tahu apakah mesinnya berpendingin cairan; untuk mengetahuinya, ia sedang mencoba `air-cooled#1`, aturan pertama tentang pendinginan, yang memerlukan `model = Honda BeAT`. Jawabannya, *Yamaha NMAX*, menggagalkan ketiga aturan berpendingin udara dan membuktikan `liquid-cooled#3`. Lalu aturan kepanasan menyala. Lima pertanyaan, dan satu di antaranya — model — ditanyakan hanya karena sebuah aturan memerlukannya.

Sarannya sengaja hati-hati. Cairan pendingin mesin yang panas bertekanan, dan membuka tutup radiator mesin yang panas dapat melepuhkan kulit. Mbak Wulan diminta berhenti di tempat yang aman, membiarkan mesinnya dingin, melihat tabung cadangan cairan pendingin hanya ketika sudah dingin, dan ke bengkel hari ini.

8.6.5 Rem yang lembek

Suatu malam seorang pengemudi muda berkata tuas rem depannya masuk hampir sampai ke setang. Pencari kerusakan berhenti pada pertanyaan pertama:

```
Safety first.
How do the brakes feel when you squeeze the levers? (normal/soft/grinding) > soft
STOP: do not ride this motorcycle now.
Result: safety = stop riding now  (rule stop riding now#1, CF 100)
Result: cause = brake fluid leak or air in the brake line  (rule stop riding now#1, CF 100)
Result: advice = do not ride; push it or call a bengkel that comes to you  (rule stop
riding now#1, CF 100)
How I know:
  stop riding now#1 fired because:
    brake feel = soft  (answered by the user, CF 100)
    so safety = stop riding now, cause = brake fluid leak or air in the brake line, advice
= do not ride; push it or call a bengkel that comes to you  [rule CF 100, conclusion CF
100]
```

Tanpa diagnosis, tanpa daftar penyebab, tanpa pertanyaan tentang bensin: tujuan *stop riding now* terbukti oleh aturan pertama, dan aplikasi selesai. Inilah rancangan keselamatan Subbab 8.2.6 yang bekerja. Penjelasannya tetap mengatakan mengapa, sehingga pengemudi tahu bahwa program itu tidak sekadar rewel. Ia mendorong motornya ke tepi jalan, menelepon montir panggilan, dan malam itu tidak ada seorang pun yang celaka.

> **WAWASAN KUNCI:** Urutan tujuan adalah keputusan keselamatan. Dengan membuktikan *stop riding now* sebelum mencari penyebab apa pun, aplikasi menjamin tidak ada pengemudi dengan rem lembek yang lebih dulu diberi tahu tentang akinya. Tidak ada satu pun dalam aturan-aturan itu sendiri yang dapat menjamin hal ini; yang menjaminnya adalah struktur program di sekelilingnya.

8.6.6 Menghitung pertanyaan

Tabel 8.2 menjajarkan sesi-sesi itu. Angkanya adalah baris “Questions asked” dalam transkrip, yang sudah termasuk dua pertanyaan cek keselamatan; sesi rem lembek menanyakan satu.

Tabel 8.2 Pertanyaan yang diajukan dalam sesi-sesi bab ini, termasuk cek keselamatan.

Sesi	Mode	Pertanyaan	Hasil
BeAT Mas Yudi (8.6.1)	mundur, cari penyebab	4	aki lemah, CF 90
BeAT yang sama (8.6.2)	maju, daftar periksa dulu	10	aki lemah, CF 80
Vario 160 Pak Heru (8.6.3)	mundur, periksa V-belt	5	V-belt aus, CF 81
NMAX Mbak Wulan (8.6.4)	mundur, cari penyebab	5	kepanasan, CF 100
rem lembek (8.6.5)	mundur, tujuan keselamatan	1	berhenti berkendara
Scoopy Pak Slamet (8.7.1)	mundur, cari penyebab	8	tanpa penyebab, lalu busi aus

Runut mundur mengajukan empat atau lima pertanyaan ketika kerusakannya dikenal, dan delapan ketika ia harus menyingkirkan setiap penyebab yang dikenalnya sebelum menyerah. Runut maju mengajukan sepuluh untuk kerusakan yang ditemukan runut mundur dengan empat. Tak satu pun angka ini tetap: semuanya bergantung pada urutan aturan dan pada kerusakannya. Tetapi polanya umum, dan itulah alasan penalaran yang digerakkan tujuan menjadi baku untuk sistem konsultasi yang pertanyaannya dijawab manusia [5].

8.7 Membuatnya Lebih Baik

8.7.1 Kerusakan yang tak dapat dinamainya

Scoopy Pak Slamet tidak mau hidup. Starternya berputar normal, klaksonnya keras, bensinnya ada, dan tidak ada lampu peringatan yang menyala:

```
Safety first.
How do the brakes feel when you squeeze the levers? (normal/soft/grinding) > normal
Is there a strong smell of petrol around the motorcycle? (yes/no) > no
Looking for the cause.
Does the engine start? (yes/no) > no
How loud is the horn? (normal/weak/none) > normal
What happens when you press the starter button? (none/click/slow/normal) > normal
What does the fuel gauge show? (empty/ok) > ok
Is the temperature warning light on? (yes/no) > no
Is the engine check light blinking? (yes/no) > no
I cannot name the fault. Take it to a bengkel and tell the mechanic what you told me.
No rule fired.
How I know:
  No rule fired.
Questions asked: 8
```

Delapan pertanyaan, dan tidak ada aturan yang dapat dibuktikan. Pencari kerusakan mengatakannya dengan jujur dan memintanya membawa motor ke bengkel beserta jawaban-jawaban yang sudah ia berikan — persis yang ingin didengar seorang montir. Cak Man menemukan busi yang aus dan kotor lalu menggantinya dalam sepuluh menit. Ia kemudian mengajarkan kasus itu kepada pencari kerusakan: *mesin tidak mau hidup, starter normal, bensin ada, lampu cek mesin mati → busi aus*, dengan CF 80, karena kerusakan lain dapat tampak sama. Kasus itu menjadi satu baris tambahan dalam salinan berkas kasus, `fault-cases-plus.json`. Pengemudi berikutnya dengan gejala yang sama mendapat jawaban:

```
Safety first.
How do the brakes feel when you squeeze the levers? (normal/soft/grinding) > normal
Is there a strong smell of petrol around the motorcycle? (yes/no) > no
Looking for the cause.
Does the engine start? (yes/no) > no
How loud is the horn? (normal/weak/none) > normal
What happens when you press the starter button? (none/click/slow/normal) > normal
What does the fuel gauge show? (empty/ok) > ok
Is the temperature warning light on? (yes/no) > no
Is the engine check light blinking? (yes/no) > no
Result: cause = worn spark plug  (rule worn spark plug#1, CF 80)
Result: advice = have the spark plug checked and replaced at a bengkel; it takes ten
minutes  (rule worn spark plug#1, CF 80)
How I know:
  worn spark plug#1 fired because:
    engine starts = no  (answered by the user, CF 100)
    starter sound = normal  (answered by the user, CF 100)
    fuel gauge = ok  (answered by the user, CF 100)
    engine check light = no  (answered by the user, CF 100)
    so cause = worn spark plug, advice = have the spark plug checked and replaced at a
bengkel; it takes ten minutes  [rule CF 80, conclusion CF 80]
Questions asked: 8
```

Aturan baru itu dicoba paling akhir, karena CF-nya, 80, paling rendah; semua aturan yang lebih pasti sudah disingkirkan lebih dulu oleh delapan jawaban yang sama. Inilah putaran sehari-hari VCIRS dari Bab 3 yang diterapkan pada perunutan: ketika sistem tidak punya jawaban, sang pakar mengajarkan kasusnya, dan transformasi basis pengetahuan mengubahnya menjadi aturan yang langsung dapat dipakai runut maju maupun runut mundur.

Bagaimana Cak Man memilih CF 80? Bukan dengan mengukur sesuatu. Faktor kepastian adalah pertimbangan sang pakar, dan ada gunanya menyepakati skala sederhana di pangkalan lalu berpegang padanya: 100 untuk “selalu, saya belum pernah melihat yang lain” (tangki kosong mematikan mesin); 90 untuk “hampir selalu” (klakson lemah berarti daya lemah); 80 untuk “biasanya, tetapi kerusakan lain bisa tampak sama” (busi, relai starter, kopling yang kotor); 70 atau kurang untuk “dugaan pertama yang masuk akal”. Dengan skala bersama, dua montir yang

mengajari basis pengetahuan yang sama memberi angka yang sebanding, dan pengemudi yang membaca *CF 72* tahu artinya "cukup mungkin, layak diperiksa", bukan "pasti". Skala itu sebaiknya ditempel di dinding bengkel, di samping kartu pemeriksaan tiga menit, agar setiap orang yang mengajarkan kasus baru memakai ukuran yang sama.

8.7.2 Melihat setiap langkah

Penjelasan How memperlihatkan aturan-aturan yang menyala. Kadang seorang montir menginginkan lebih: setiap aturan yang *dicoba* mesin, termasuk yang gagal, dan tepat di mana masing-masing gagal. Itulah catatan kejadian mesin, EventLog dalam tesisnya [4], dan opsi `--log` mencetaknya setelah pencarian penyebab. Untuk pagi Mas Yudi (`--echo --log < ../data/find-beat.txt`), akhir keluarannya adalah:

```
Questions asked: 4
Every step:
   1. backward chaining for cause
   2. goal cause: rules stop riding now#1, stop riding now#2, blown main fuse or loose
battery cable#1, weak battery#1, empty tank#1, overheating#1, worn brake pads#1, worn V-
belt#1, starter relay or starter motor#1, injection or sensor fault#1, dirty CVT clutch#1
   3. try stop riding now#1 for cause
   4. stop riding now#1: brake feel = soft fails (brake feel = normal)
   5. stop riding now#1 does not fire
   6. try stop riding now#2 for cause
   7. stop riding now#2: fuel smell = yes fails (fuel smell = no)
   8. stop riding now#2 does not fire
   9. try blown main fuse or loose battery cable#1 for cause
  10. ask engine starts (for blown main fuse or loose battery cable#1: engine starts = no)
  11. engine starts = no [CF 100] (user)
  12. blown main fuse or loose battery cable#1: engine starts = no holds (engine starts =
no)
  13. subgoal electrics (needed by blown main fuse or loose battery cable#1)
  14. goal electrics: rules no power#1, weak power#1, weak power#2
  15. try no power#1 for electrics
  16. ask horn (for no power#1: horn = none)
  17. horn = weak [CF 100] (user)
  18. no power#1: horn = none fails (horn = weak)
  19. no power#1 does not fire
  20. try weak power#1 for electrics
  21. weak power#1: horn = weak holds (horn = weak)
  22. weak power#1 fires: electrics = weak power [CF 90]
  23. blown main fuse or loose battery cable#1: electrics = no power fails (electrics =
weak power)
  24. blown main fuse or loose battery cable#1 does not fire
  25. try weak battery#1 for cause
  26. weak battery#1: engine starts = no holds (engine starts = no)
  27. weak battery#1: electrics = weak power holds (electrics = weak power)
  28. weak battery#1 fires: cause = weak battery, advice = kick-start it if it has a kick
starter; charge or replace the battery at a bengkel today [CF 90]
  29. goal cause proved
```

Baris 2 adalah agenda runut mundur: sebelas aturan yang menyimpulkan penyebab, menurut urutan akan dicoba — kepastian lebih dulu, lalu urutan dalam berkas kasus. Baris 3 sampai 8 adalah kedua aturan keselamatan yang gagal pada fakta yang sudah diketahui, tanpa pertanyaan. Baris 13 adalah saat sebuah kondisi menjadi subtujuan: `electrics` diperlukan aturan sekring, sehingga mesin membuka agenda baru yang lebih kecil (baris 14) berisi tiga aturan yang menyimpulkannya. Baris 16 adalah pertanyaan, dengan alasannya di dalam kurung: *ask horn, for no power#1: horn = none*. Baris 22 adalah konklusi antara beserta CF-nya, dan baris 26 dan 27 memperlihatkan aturan aki lemah memakai ulang kedua fakta tanpa bertanya lagi. Dua puluh sembilan langkah, empat pertanyaan.

Catatan seperti inilah cara sebuah basis pengetahuan di-*debug*. Bila pencari kerusakan mengajukan pertanyaan yang tampak konyol, catatan menunjukkan aturan mana yang menanyakannya dan mengapa aturan itu sedang dicoba pada saat itu; bila ia melewatkan sebuah penyebab, catatan menunjukkan kondisi mana yang gagal. Cak Man membacanya seperti ia membaca suara mesin.

8.7.3 Menggabungkan bukti

Di Subbab 8.6.2 kelistrikan disimpulkan dengan CF 80 oleh aturan starter, dan aturan klakson, dengan CF 90, diabaikan karena nilainya sudah diketahui. Banyak sistem berbasis aturan **menggabungkan** kepastian dua aturan yang mencapai konklusi yang sama: dua bukti yang saling bebas membuat konklusi lebih pasti daripada masing-masing. Penggabungan klasik MYCIN untuk dua faktor positif, ditulis pada skala 100, adalah $CF_1 + CF_2(100 - CF_1)/100$; untuk 80 dan 90 hasilnya $80 + 90 \times 20/100 = 98$ [5]. Mesin VCIRS mengikuti tesisnya dan mempertahankan konklusi pertama. Bagi pencari kerusakan, inilah pilihan yang hati-hati — ia tidak pernah menjadi lebih yakin daripada yang dibenarkan satu aturan — dan penjelasan How menampilkan kedua aturan yang menyala, sehingga montir yang membacanya melihat bukti kedua. Soal Latihan 8.6 meminta Anda menghitung apa yang akan berubah bila bukti digabungkan.

8.7.4 Urutan aturan

Runut mundur mencoba aturan-aturan untuk sebuah tujuan menurut urutan kepastian, dan bila kepastiannya sama, menurut urutan pengajarannya. Urutan itu menentukan *pertanyaan mana yang datang lebih dulu*, bukan jawaban mana yang benar: sebuah aturan hanya terbukti bila semua kondisinya terpenuhi. Tetapi urutan itu penting bagi waktu pengemudi. Di Subbab 8.6.4 Mbak Wulan lebih dulu ditanya apakah mesinnya hidup — pertanyaan untuk aturan sekring dan aki, yang datang lebih awal dalam berkas — dan baru kemudian tentang lampu suhu dan model, karena aturan kepanasan datang sesudah keduanya. Bila sekelompok pengemudi kebanyakan membawa satu jenis kerusakan, menaruh kasus-kasus yang paling umum lebih awal dalam berkas kasus membuat pencari kerusakan rata-rata bertanya lebih sedikit. Tingkat pemakaian dari Bab 7 dapat memandu pilihan itu.

8.7.5 Pengujian

`FinderTest.java` dan `test_finder.py` memeriksa basis pengetahuan (21 node dan 21 aturan), tujuan keselamatan (terbukti untuk rem lembek, tidak terbukti untuk rem normal tanpa bau bensin), bahwa runut mundur menemukan aki lemah dengan empat pertanyaan, dan bahwa runut maju menyimpulkan V-belt aus dari 30.000 km serta kepanasan dari NMAX yang lampunya menyala. Keduanya diakhiri dengan:

```
all tests passed
```

8.8 Memakainya Sungguh-Sungguh

Di pangkalan. Pasang pencari kerusakan di laptop atau ponsel bekas di pangkalan, dengan berkas kasus yang dipercaya montir setempat. Ajarkan kasus-kasus yang berulang di daerah Anda — debu jalan musim kemarau, banjir musim hujan yang meninggalkan air di rumah CVT dan saringan udara — dan biarkan montir membaca aturan-aturannya dengan `rules` sesekali. Pengemudi yang mogok di tempat lain dapat menjalankan program yang sama di ponselnya

dengan aplikasi Python, atau cukup menelepon rekan di pangkalan yang menjalankannya untuknya.

Model Anda sendiri. Enam kasus model hanyalah permulaan. Tambahkan model yang dikendarai rekan-rekan Anda — Honda PCX 160, Yamaha Aerox, Honda Genio, skuter listrik — dengan jenis pendinginan yang benar, setelah memeriksa buku pedoman pemiliknya. Motor listrik memerlukan aturannya sendiri (daya baterai, peringatan kontroler) dan montirnya sendiri; jangan memaksakan aturan mesin bensin untuk mencakupnya.

Mengenal angka Anda. Ambang V-belt 24.000 km, gagasan bahwa klakson lemah berarti aki lemah, CF setiap aturan — semuanya berasal dari pengalaman seorang montir dan dari jadwal diler [9]. Montir Anda mungkin memakai angka lain, dan memang sebaiknya. Inti pencari kerusakan adalah bahwa angka-angkanya, sekali diajarkan, diterapkan dengan cara yang sama untuk setiap pengemudi, dan setiap jawaban dapat dilacak kembali kepadanya.

Tiga menit sebelum pencari kerusakan. Beberapa pemeriksaan lebih cepat daripada program apa pun, dan pertanyaan-pertanyaan pencari kerusakan menganggap pengemudi sudah melakukannya: apakah standar samping sudah naik (kebanyakan skuter tidak mau hidup dengan standar samping turun), apakah tuas rem ditarik ketika menekan starter (starter skuter hanya bekerja bila rem ditahan), apakah kunci sudah diputar penuh, dan apakah sakelar pemutus mesin di setang kanan berada pada posisi *run*? Tuliskan keempatnya di sebuah kartu di pangkalan. Sebagian besar telepon "motor saya tidak mau hidup" kepada montir mana pun berakhir dengan salah satunya, dan mengajari program menanyakannya kalah berguna dibanding mengajari pengemudi memeriksanya lebih dulu.

Keselamatan bukan pengaturan. Pertahankan kedua aturan berhenti berkendara di puncak basis pengetahuan dan pertahankan cek keselamatan di depan setiap diagnosis. Jangan pernah mengajari pencari kerusakan perbaikan yang berarti mengerjakan rem, sistem bahan bakar, atau apa pun yang panas. Bila ragu, sarannya adalah bengkel.

Buku catatan. Mintalah para pengemudi menyimpan transkrip setiap konsultasi — pertanyaan, jawaban, dan saran — dan menulis satu baris lagi setelah motornya diperbaiki: apa yang sebenarnya ditemukan montir. Sebulan sekali, bandingkan. Bila penyebab dari pencari kerusakan sama dengan temuan montir, pengetahuannya terkonfirmasi; bila berbeda, transkrip menunjukkan persis jawaban mana yang menyesatkan pencari kerusakan, dan Cak Man dapat mengajarkan kasus yang belum ada, seperti yang ia lakukan untuk busi Pak Slamet. Setelah beberapa bulan, buku catatan itu juga menjadi sejarah kecil motor-motor pangkalan: kerusakan mana yang umum, pada musim apa, pada model apa.

Dalam bahasa Indonesia. Pertanyaan-pertanyaan pencari kerusakan adalah teks biasa di `QUESTIONS`; pengetahuannya memakai nilai-nilai pendek berbahasa Inggris seperti `weak` dan `no power`. Untuk memberi pengemudi pertanyaan berbahasa Indonesia — *Bagaimana bunyi klakson?* — ubahlah hanya teks pertanyaannya dan pertahankan nilainya, atau terjemahkan nilai-nilai dalam berkas kasus bersama pilihan jawabannya sekaligus. Aturan-aturan tidak peduli bahasa apa yang dibaca pengemudi; yang penting jawabannya cocok dengan nilai-nilai dalam kondisi.

Mesin-mesin lain. Program yang sama, dengan kasus lain, menjadi pencari kerusakan untuk pompa air, genset, *rice cooker* di binatu, atau kotak pendingin penjual ikan. Dua arah penalaran dan tiga penjelasannya tetap sama.

8.9 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Menanyakan konklusi antara kepada pengguna. Pengemudi tidak dapat menjawab "apakah kelistrikannya berdaya lemah?" *Hindari:* biarkan aturan menyimpulkan variabel antara, dan jangan pernah menanyakan variabel yang disimpulkan suatu aturan, seperti dilakukan `Driver.ask`.

Memakai runut maju ketika bertanya itu mahal. Runut maju memerlukan fakta lebih dulu dan menanyakan banyak hal yang tidak penting. *Hindari:* pakailah runut mundur bila manusia yang harus menjawab; pakailah runut maju bila fakta datang dengan sendirinya, dari formulir atau sensor.

Menggantungkan keselamatan pada urutan aturan. Bila aturan berhenti berkendara hanyalah penyebab biasa di antara yang lain, seorang pengemudi bisa diberi tahu tentang akinya sebelum ada yang menanyakan remnya. *Hindari:* periksa tujuan bahaya lebih dulu, tersendiri.

Menyarankan perbaikan yang dapat mencelakakan. *Hindari:* saran menyebut penyebab dan langkah berikutnya; membongkar rem, saluran bensin, dan sistem pendingin yang panas adalah pekerjaan montir.

Menjawab angka dengan kata-kata. `km since belt change >= 24000` memerlukan angka; *sekitar dua puluh ribu* tidak dibandingkan sebagai angka dan kondisinya gagal. *Hindari:* pertanyaannya menyebut "(a number)"; bila pengemudi tidak tahu, menekan Enter lebih baik daripada menebak.

Kata yang berbeda untuk model yang sama. `Honda Beat` dan `Honda BeAT` adalah nilai yang berbeda. *Hindari:* pakailah pilihan yang ditampilkan pertanyaan.

Membaca CF sebagai peluang. CF 81 tidak berarti 81 persen skuter seperti itu belt-nya aus. CF adalah ukuran seberapa kuat aturan-aturan mendukung konklusinya. *Hindari:* bacalah sebagai dukungan "kuat", "cukup kuat", atau "lemah", dan biarkan montir yang memutuskan.

Melupakan keselamatan pengemudi di luar motornya. Pengemudi yang berhenti di jalan ramai berada dalam bahaya, apa pun kerusakannya. *Hindari:* saran pertama setiap konsultasi di jalan adalah mendorong motor ke tempat aman, keluar dari lajur lalu lintas, sebelum menjawab pertanyaan apa pun.

Membiarkan pencari kerusakan menggantikan servis. Pencari kerusakan yang pandai menjawab dapat menggoda pengemudi melewatkan servis rutin. *Hindari:* penggantian berkala — oli, belt, kampas rem, busi — mengikuti jadwal diler [9]; pencari kerusakan untuk hari-hari ketika sesuatu sudah telanjur rusak.

Mengajarkan penyebab yang belum terbukti. Bila busi diganti *dan* saringan bensin juga, tidak jelas mana yang memperbaikinya. *Hindari:* ajarkan hanya apa yang dibuktikan oleh perbaikan.

8.10 Ringkasan Bab

- **Runut maju** bernalar dari fakta menuju semua yang tersirat; **runut mundur** bernalar dari tujuan menuju aturan-aturan dan pertanyaan-pertanyaan yang membuktikannya, dan hanya menanyakan apa yang diperlukan tujuan itu (**Gambar 8.1**, **Kode 8.1**, **Kode 8.2**).
- **Konklusi antara** seperti `electrics = weak power` membuat aturan memberi makan aturan, seperti montir yang lebih dulu menemukan sistemnya lalu komponennya.

- Sistem berbasis aturan menjawab **Apa** (konklusi-konklusinya), **Bagaimana** (aturan dan fakta yang menghasilkannya), dan **Mengapa** (aturan dan kondisi di balik sebuah pertanyaan, serta rantai aturan dari tujuan).
- Kepastian sebuah aturan dan sebuah konklusi mengikuti Persamaan (8.1) dan Persamaan (8.2): mata rantai terlemah yang menentukan, dan sebuah rantai tidak pernah lebih pasti daripada langkahnya yang paling tidak pasti.
- VCIRS mengubah basis pengetahuan yang dibangun dari kasus menjadi aturan JIKA-MAKA melalui **transformasi basis pengetahuan**, sehingga kemudahan mengajar dari RDR dan kekuatan inferensi sistem berbasis aturan bekerja bersama (**Tabel 8.1**).
- **Pencari kerusakan** (**Kode 8.3**, **Kode 8.4**) memeriksa tujuan bahaya lebih dulu, menemukan penyebab dengan runut mundur dalam empat atau lima pertanyaan, juga dapat mendiagnosis dengan runut maju, menjelaskan setiap langkah, dan memberi saran menurut kedaruratan — berhenti sekarang, bengkel hari ini, minggu ini, atau kerjakan sendiri (**Gambar 8.2**, **Tabel 8.2**).

8.11 Soal Latihan

8.1 Dengan empat aturan dalam **Kode 8.2**, tuliskan dengan tangan jejak runut maju dari `engine starts = no`, `starter sound = slow`, `fuel gauge = empty`. Aturan mana yang menyala, dalam urutan apa, dan apa yang disimpulkan tentang penyebabnya?

8.2 Untuk empat aturan yang sama, apa yang ditanyakan runut mundur, dalam urutan apa, untuk membuktikan `cause = empty tank` bagi pengemudi yang mesinnya tidak mau hidup dan jarum bensinnya menunjuk kosong? Mengapa klakson tidak pernah ditanyakan?

8.3 Hitunglah dengan Persamaan (8.1) dan Persamaan (8.2) CF yang dipakai pencari kerusakan untuk menyimpulkan *dirty CVT clutch* bagi skuter yang hidup, kecepatan puncaknya turun, dan bergetar. Tunjukkan kedua langkahnya.

8.4 Seorang pengemudi menjalankan `check "cause = dirty CVT clutch"`. Daftarlah pertanyaan-pertanyaan yang diajukan pencari kerusakan setelah cek keselamatan, secara berurutan, bila jawabannya: mesinnya hidup, kecepatan puncaknya turun, motornya bergetar.

8.5 Di Subbab 8.6.1, aturan *blown main fuse* dicoba sebelum aturan *weak battery*. Jelaskan dari aturan-aturan dan CF-nya mengapa demikian, dan mengapa mencobanya tidak memakan pertanyaan tambahan.

8.6 Di Subbab 8.6.2, berapakah CF aki lemah bila mesin menggabungkan dua aturan kelistrikan dengan rumus MYCIN dari Subbab 8.7.3 sebelum menyalakan aturan aki?

8.7 Rekan Mas Yudi mengendarai Honda PCX 160, yang berpendingin cairan. Tuliskan kasus yang akan Anda tambahkan ke `fault-cases.json`, dan jelaskan apa yang ditanyakan pencari kerusakan kepada pengendara PCX yang lampu suhunya menyala, sebelum dan sesudah perubahan Anda.

8.8 Rancanglah dua kasus baru untuk musim hujan: skuter yang *hidup tetapi mati setelah menerobos banjir*, dan skuter yang *CVT-nya berbunyi keras setelah hujan*. Untuk masing-masing, berikan fakta, konklusi, CF, dan tingkat kedaruratan menurut **Gambar 8.2**, dan katakan apakah pencari kerusakan boleh menyarankan pengemudi terus berkendara.

8.12 Penyelesaian Soal Latihan

Penyelesaian 8.1. Putaran pertama: R1 tidak terpenuhi (tidak ada fakta klakson); R2 terpenuhi (`starter sound = slow`) dan menyala, `electrics = weak power`; R3 kini terpenuhi (`engine starts = no`, `electrics = weak power`) dan menyala, `cause = weak battery`; R4 juga akan terpenuhi, tetapi variabel konklusinya, `cause`, sudah diketahui, jadi ia tidak menyala. Putaran kedua: tidak ada yang baru; berhenti. Pencari kerusakan menyimpulkan *weak battery*, walaupun tangkinya kosong — runut maju pada mesin kecil ini mengambil penyebab pertama yang dicapainya. Pencari kerusakan yang sesungguhnya mencoba aturan menurut urutan kepastian dan tetap hanya melaporkan satu penyebab; pengemudi yang melihat jarum bensin kosong sebaiknya mengisi bensin dulu, dan itulah sebabnya Soal 8.2 penting.

Penyelesaian 8.2. Tujuan `cause = empty tank`: satu-satunya aturan adalah R4. Kondisi pertamanya, `engine starts = no`, tidak disimpulkan aturan mana pun, jadi ditanyakan (no). Kondisi keduanya, `fuel gauge = empty`, ditanyakan (empty). R4 terbukti. Klakson tidak pernah ditanyakan karena tidak ada aturan untuk tujuan itu, dan tidak ada aturan untuk kondisi-kondisinya, yang menyebut klakson: runut mundur hanya mengikuti aturan-aturan yang dapat menuju tujuannya.

Penyelesaian 8.3. Pertama, aturan CVT: klausa-klausanya ber-CF 100 dan node-nya ber-CF 90, jadi menurut Persamaan (8.1) CF-nya $100 \times 90/100 = 90$; jawaban pengemudi ber-CF 100, jadi menurut Persamaan (8.2) `transmission = CVT problem` ber-CF $100 \times 90/100 = 90$. Lalu aturan kopling: CF-nya $100 \times 80/100 = 80$; fakta-faktanya ber-CF 90 (masalah CVT) dan 100 (getaran), jadi konklusinya ber-CF $\min(90,100) \times 80/100 = 72$.

Penyelesaian 8.4. Satu-satunya aturan untuk tujuan itu adalah `dirty CVT clutch#1`, yang memerlukan `transmission = CVT problem` dan `shudder = yes`. Yang pertama adalah subtujuan: aturan `CVT problem#1` menanyakan *Does the engine start?* (yes) dan *Can it still reach its usual speed?* (low), lalu menyala. Kemudian *Does it shudder when you pull away?* (yes). Tiga pertanyaan setelah dua pertanyaan cek keselamatan, lima semuanya.

Penyelesaian 8.5. Runut mundur untuk `cause` mencoba aturan-aturan yang menyimpulkan penyebab menurut CF yang menurun, dan untuk CF yang sama menurut urutan berkas kasus. Kedua aturan keselamatan (CF 100) datang lebih dulu, lalu *blown main fuse* (CF 100), yang mendahului *weak battery* (CF 100) dalam berkas. Kondisi pertama aturan sekring, `engine starts = no`, ditanyakan, dan aturan aki lemah memerlukan fakta yang sama; kondisi keduanya, `electrics = no power`, adalah subtujuan yang aturan pertamanya menanyakan klakson — dan jawaban yang sama lalu membuktikan `electrics = weak power`, yang juga diperlukan aturan aki lemah. Kedua pertanyaan yang diajukan untuk aturan sekring akan tetap ditanyakan untuk aturan aki, jadi mencoba sekring lebih dulu tidak memakan apa pun.

Penyelesaian 8.6. Bila digabungkan, kelistrikan akan ber-CF $80 + 90 \times (100 - 80)/100 = 98$. Aturan aki ber-CF 100 dan fakta-faktanya akan ber-CF 100 (mesin) dan 98 (kelistrikan), sehingga aki lemah disimpulkan dengan CF $\min(100,98) \times 100/100 = 98$, bukan 80.

Penyelesaian 8.7. Tambahkan baris `{"facts": ["model = Honda PCX 160"], "conclusions": ["cooling = liquid-cooled"], "option": 5}` (dan tambahkan *Honda PCX 160* ke pilihan pertanyaan model). Sebelum perubahan, aturan-aturan model gagal untuk PCX, `cooling` tidak dapat dibuktikan, pencari kerusakan tidak dapat menanyakannya (karena ia disimpulkan aturan), sehingga aturan kepanasan gagal dan pencari kerusakan melanjutkan ke penyebab lain, dan berakhir dengan “I cannot name the fault” bila tidak ada yang terpenuhi. Sesudah perubahan,

jawaban *Honda PCX 160* membuktikan `cooling = liquid-cooled` dan aturan kepanasan menyala, dengan lima pertanyaan yang sama seperti untuk NMAX Mbak Wulan.

Penyelesaian 8.8. Salah satu rancangan. Banjir: fakta `engine starts = yes`, `dies after flood = yes`; konklusi `cause = water in the air filter or the exhaust`, `advice = do not keep starting it; push it out of the water and take it to a bengkel today`; CF 80; kedaruratan *bengkel hari ini*. Terus-menerus menstarter mesin yang sudah menyedot air dapat merusaknya parah, jadi sarannya tidak boleh "coba lagi". Bunyi CVT: fakta `engine starts = yes`, `CVT noise after rain = yes`; konklusi `transmission = CVT problem`, lalu kasus terpisah `transmission = CVT problem`, `CVT noise after rain = yes` → `cause = water in the CVT housing`, `advice = ride gently to a bengkel this week to have the CVT drained and cleaned`; CF 70; kedaruratan *minggu ini*. Pencari kerusakan tidak boleh menyarankan terus berkendara menerobos genangan dalam; untuk CVT, berkendara pelan ke bengkel dapat diterima hanya bila bunyinya tidak disertai hilangnya tenaga.

Daftar Pustaka

[1] F. Sandi, "Ramalan nasib penjualan motor tahun 2026 usai naik tipis di 2025," *CNBC Indonesia*, Jan. 12, 2026. [Online]. Available: https://www.cnbcindonesia.com/news/20260112131535-4-701584/ramalan-nasib-penjualan-motor-tahun-2026-usai-naik-tipis-di-2025

[2] Warta Ekonomi, "Sepeda motor terlaris di Indonesia sepanjang 2025," Dec. 29, 2025. [Online]. Available: https://wartaekonomi.co.id/read595218/sepeda-motor-terlaris-di-indonesia-sepanjang-2025

[3] A. Wibowo, "5 motor paling irit dan nyaman untuk ojek online 2026," *Garduoto*, Sep. 3, 2026. [Online]. Available: https://www.garduoto.com/142984/5-motor-paling-irit-dan-nyaman-untuk-ojek-online-2026

[4] M. M. I. Subakti, "A Variable-Centered Intelligent Rule System," M.Sc.Eng. thesis, Dept. of Computer Science and Information Engineering, Taipei, Taiwan, Jun. 2005.

[5] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Hoboken, NJ, USA: Pearson, 2020.

[6] I. Subakti, "A variable-centered intelligent rule system," in *Proc. 1st Annual Int. Conf. Information and Communication Technology Seminar (ICTS 2005)*, vol. 1, no. 1, Surabaya, Indonesia, Aug. 2005, pp. 167–174.

[7] I. Subakti and R. Hidayatullah, "Aplikasi sistem pakar untuk diagnosis awal gangguan kesehatan secara mandiri menggunakan Variable-Centered Intelligent Rule System," vol. 6, no. 1, pp. 11–16, Jan. 2007.

[8] G. R. Baur and D. V. Pigford, *Expert Systems for Business: Concepts and Applications*. Boston, MA, USA: Boyd & Fraser, 1990.

[9] Wahana Honda, "Komponen motor matic yang wajib diganti berkala," Wahana Honda blog. [Online]. Available: https://www.wahanahonda.com/blog/komponen-motor-matic-yang-wajib-diganti-berkala (accessed Sep. 2026).

Bab 9

Penasihat Kelayakan Pinjaman Koperasi: VCIRS Fuzzy

Setelah menyelesaikan bab ini Anda akan mampu menjelaskan mengapa kata-kata seperti *rendah*, *sedang*, dan *tinggi* bertepi lunak, dan bagaimana sebuah **himpunan fuzzy** (*fuzzy set*) memberi setiap angka derajat keanggotaan pada sebuah kata; menerapkan *hedge* (kata penguat dan pelunak) *very* dan *somewhat*; menghitung dengan tangan seberapa jauh rasio angsuran seorang anggota termasuk *rendah*, lalu mencocokkannya dengan kait fuzzy milik mesin; membedakan dua jenis ketidaktepatan dalam makalah VCIRS fuzzy, yaitu **kekaburan** (*fuzziness*) dan **ketidakpastian** (*uncertainty*), serta membawa ketidakpastian sebagai **bilangan fuzzy** kecil; dan menjalankan **penasihat pinjaman** sebuah koperasi simpan pinjam, yang menyiapkan rapat bulanan pengurus dengan rekomendasi untuk setiap permohonan, alasannya dalam kata-kata yang mudah dipahami, peringatan bila sebuah kasus berada di batas tipis, dan jumlah pinjaman yang akan mengubah jawabannya. Penasihat ini dibangun agar adil dan terbuka: ia menolak mempelajari apa pun dari atribut yang dilindungi seperti jenis kelamin, agama, atau suku, dan setiap jawabannya dapat dilacak ke angka-angka yang menghasilkannya.

9.1 Pengurus Koperasi Sumber Makmur

Koperasi Sumber Makmur adalah koperasi simpan pinjam di sebuah desa di selatan Malang. Empat ratus anggotanya adalah pedagang sayur, petani padi, penjahit, pemilik warung, pengemudi ojek, dan pengusaha katering. Setiap bulan setiap anggota membayar simpanan wajib yang kecil, dan dari simpanan yang terkumpul itu koperasi meminjamkan uang kepada anggotanya — hanya kepada anggotanya, sehingga ia termasuk koperasi yang oleh regulator disebut *closed loop*. Koperasi yang juga melayani masyarakat umum disebut *open loop*, dan menurut undang-undang sektor keuangan tahun 2023 (UU P2SK) koperasi seperti itu beralih ke pengawasan Otoritas Jasa Keuangan (OJK); Kementerian Koperasi menyerahkan daftar koperasi semacam itu kepada OJK pada Januari 2025 [1]. Usaha simpan pinjam oleh koperasi diatur oleh Peraturan Menteri Koperasi dan UKM Nomor 8 Tahun 2023 tentang Usaha Simpan Pinjam oleh Koperasi [2], di bawah Undang-Undang Perkoperasian tahun 1992 yang masih berlaku sementara undang-undang baru sedang disiapkan; pada Juli 2026 Menteri Koperasi menyatakan undang-undang itu ditargetkan disahkan tahun ini [3], [4]. Koperasi juga tumbuh pesat: sampai Juli 2026 telah terbentuk 83.382 koperasi dalam program Koperasi Desa Merah Putih [5], dan banyak di antaranya meminjamkan uang kepada anggotanya. Penghasilan anggota-anggotanya seberagam pekerjaan mereka: pedagang sayur mendapat sedikit setiap hari, penjahit menurut pesanan, istri pegawai setiap bulan, dan petani padi kebanyakan dua kali setahun, saat panen. Keputusan pinjaman yang baik harus melihat perbedaan-perbedaan itu, dan keputusan yang adil tidak boleh melihat hal lain selain itu.

Pada Sabtu pertama setiap bulan, pengurus berkumpul di balai desa untuk memutuskan permohonan pinjaman bulan itu. Bu Wati, bendahara, menyiapkan berkas-berkasnya. Sudah sebelas tahun ia melakukannya, dan ia tahu apa yang biasanya diputuskan pengurus: anggota yang angsurannya hanya memakan sebagian kecil penghasilannya mendapat pinjaman; anggota

yang angsurannya akan memakan separuh penghasilannya diminta meminjam lebih sedikit atau menabung dulu; anggota baru memulai dengan pinjaman kecil; anggota yang sering terlambat diajak bicara langsung. Yang ia inginkan bukan mesin yang memutuskan — di koperasi, rapat anggota dan pengurus yang memutuskan, dan memang seharusnya begitu. Ia menginginkan **pembacaan pertama yang konsisten** atas setiap berkas, lengkap dengan alasan tertulis, sehingga pengurus dapat mencurahkan waktunya untuk kasus-kasus yang sulit, sehingga dua anggota dengan keadaan yang sama mendapat jawaban yang sama, dan sehingga setiap anggota dapat diberi tahu *mengapa*.

Ada kesulitan yang dikenal siapa pun yang pernah duduk dalam rapat seperti itu. Pengetahuan pengurus ditulis dalam kata-kata: "kalau angsurannya **rendah** dibanding penghasilan", "kalau anggota itu punya simpanan yang **cukup**", "kalau ia **jarang** terlambat". Tetapi berkas-berkasnya berisi angka: penghasilan Rp 3.600.000 sebulan, angsuran Rp 1.010.000, satu kali terlambat tahun lalu. Di mana *rendah* berakhir? Bila jawabannya "pada 25 persen", maka anggota dengan 24,9 persen dan anggota dengan 25,1 persen mendapat jawaban yang berbeda, padahal tak seorang pun dapat dengan jujur mengatakan keadaan mereka berbeda. Jawaban bab ini adalah jawaban yang diusulkan makalah VCIRS fuzzy: biarkan kata-kata tetap bertepi lunak, dan biarkan mesin bernalar dengan *derajat* [6].

> **JEBAKAN:** Penasihat dalam bab ini menyiapkan keputusan; ia tidak membuatnya, dan ia bukan nasihat keuangan bagi siapa pun. Pengurus memutuskan setiap pinjaman, anggota memutuskan apakah akan meminjam, dan aturan koperasi sendiri, yang ditetapkan rapat anggotanya, selalu didahulukan.

9.2 Gagasannya dengan Kata-Kata Sederhana

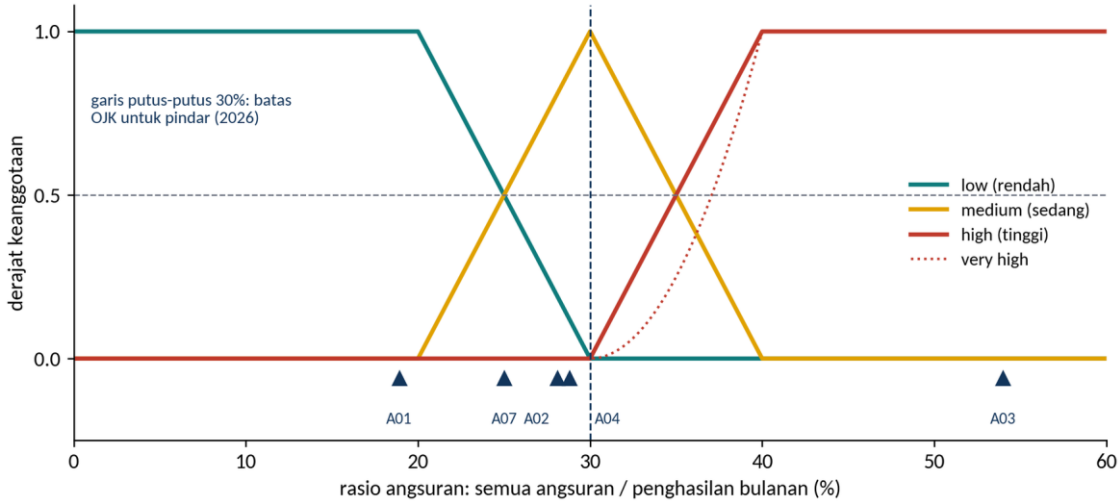
9.2.1 Kata-kata yang bertepi lunak

Himpunan klasik bertepi keras: sebuah angka termasuk di dalamnya atau tidak. **Himpunan fuzzy** dari Lotfi Zadeh [7] membolehkan sebuah angka termasuk dalam sebuah himpunan *sampai derajat tertentu* antara 0 dan 1. Untuk kata *rendah* yang dikenakan pada rasio angsuran, rasio 15 persen rendah berderajat 1, rasio 35 persen berderajat 0, dan rasio 25 persen berderajat 0,5 — setengah rendah, setengah tidak. Aturan yang mengubah angka menjadi derajat itu disebut **fungsi keanggotaan** (*membership function*) kata tersebut. Kita memakai kata-kata seperti ini sepanjang hari tanpa menyadarinya. Penjual siomay bilang harga Rp 15.000 *agak mahal* dan Rp 10.000 *murah*, dan tak seorang pun bertanya pada rupiah ke berapa yang satu berubah menjadi yang lain. Pengemudi bilang pasar *dekat* bila dua kilometer dan *jauh* bila sepuluh. Kata-kata itu berguna justru karena lunak: ia membawa pertimbangan yang tidak berubah mendadak hanya karena angkanya bergeser sedikit. Himpunan fuzzy menuliskan pertimbangan itu agar sebuah program dapat memakainya dengan cara yang sama setiap kali.

Fungsi keanggotaan paling sederhana yang berguna adalah **trapesium**, yang ditentukan empat angka $a \le b \le c \le d$: derajatnya 0 sampai a, naik lurus ke 1 di b, tetap 1 sampai c, lalu turun kembali ke 0 di d:

$$\mu(x) = \begin{cases} 1 & b \le x \le c \\ \frac{x-a}{b-a} & a < x < b \\ \frac{d-x}{d-c} & c < x < d \\ 0 & \text{otherwise} \end{cases} \tag{9.1}$$

(Baris terakhir, *otherwise*, berarti "selain itu".) Segitiga adalah trapesium dengan $b = c$, dan himpunan yang dimulai dari ujung kiri, seperti *rendah*, punya $a = b = 0$. Pengurus koperasi menyepakati tiga kata untuk rasio angsuran: *low* (rendah) adalah (0, 0, 20, 30), *medium* (sedang) adalah (20, 30, 30, 40), dan *high* (tinggi) adalah (30, 40, 100, 100). **Gambar 9.1** menggambarkannya. Antara 20 dan 30 persen sebuah rasio sebagian rendah dan sebagian sedang; antara 30 dan 40, sebagian sedang dan sebagian tinggi. Tidak ada satu titik pun tempat pendapat pengurus melompat.



Gambar 9.1 Tiga kata fuzzy untuk rasio angsuran, dengan hedge *very high* (bertitik). Garis putus-putus mendatar adalah ambang 0,5 tempat sebuah kondisi fuzzy dianggap benar; segitiga-segitiga menandai rasio lima anggota di Subbab 9.6.

Pengurus menggambarkan angka-angka lain dengan cara yang sama: *membership* (lama keanggotaan) adalah *new* (baru) (0, 0, 6, 12) atau *established* (mapan) (6, 12, 1200, 1200) bulan; *savings cover* (cakupan simpanan), yaitu simpanan anggota sebagai persentase pinjaman, adalah *small* (kecil) (0, 0, 10, 20) atau *enough* (cukup) (10, 20, 1000, 1000); *late payments* (keterlambatan) dalam dua belas bulan terakhir adalah *rare* (jarang) (0, 0, 1, 3) atau *frequent* (sering) (1, 3, 12, 12). Angka terakhir sebuah himpunan seperti 1200 cukup lebih besar daripada nilai apa pun yang mungkin muncul. Kesembilan himpunan itu tersimpan dalam berkas `fuzzy-sets.csv`, satu baris masing-masing, tempat pengurus dapat membaca dan mengubahnya. Seperti di bab-bab sebelumnya, nama variabel dan istilah dalam basis pengetahuan ditulis dalam bahasa Inggris yang pendek, agar Java dan Python berbagi berkas yang sama.

9.2.2 Sangat, agak, bukan

Orang menajamkan dan melunakkan kata-katanya dengan **hedge**: *sangat* tinggi, *agak* rendah, *bukan* rendah. Zadeh mengusulkan agar hedge diperlakukan sebagai operasi pada derajat [7]. *Very* (sangat) memusatkan kata dengan mengkuadratkan derajatnya; *somewhat* (agak) melebarkannya dengan mengambil akar kuadratnya; *not* (bukan) mengambil komplemennya:

$$\mu_{\text{very } A}(x) = \mu_A(x)^2, \qquad \mu_{\text{somewhat } A}(x) = \sqrt{\mu_A(x)}, \qquad \mu_{\text{not } A}(x) = 1 - \mu_A(x) \tag{9.2}$$

Jadi rasio yang tinggi berderajat 0,5 hanya *sangat tinggi* berderajat 0,25, dan rasio yang rendah berderajat 0,5 *agak rendah* berderajat 0,71. Mesin vcirs-core memahami *very*, *extremely* (pangkat tiga), *somewhat*, *rather*, dan *fairly* (akar kuadrat), serta *not*, yang ditulis di depan sebuah istilah dalam kondisi, seperti `repayment ratio = very high`.

Hedge juga membuat pengetahuan pengurus lebih mudah dibacakan. "Bila rasio angsurannya *agak rendah* dan anggotanya sudah mapan, sarankan" terdengar seperti kalimat yang akan diucapkan seorang pengurus, dan artinya persis seperti yang ia maksudkan: sedikit lebih murah hati di batasnya daripada *rendah* saja. Karena hedge hanyalah sebuah operasi pada derajat, ia tidak memerlukan himpunan fuzzy baru dan tidak memerlukan angka baru di `fuzzy-sets.csv`; tiga kata yang disepakati pengurus sudah membawa semua nuansanya. Dengan demikian, perbendaharaan kata pengurus dapat tumbuh tanpa menambah pekerjaan menyetel angka.

9.2.3 Kondisi fuzzy dalam VCIRS

Kait fuzzy milik mesin itu kecil, dan sudah cukup [6], [8]. Sebuah variabel boleh punya himpunan-himpunan fuzzy. Ketika kondisi seperti `repayment ratio = medium` bertemu fakta berupa *angka*, seperti `repayment ratio = 28.1`, kondisi itu tidak sekadar benar atau salah: ia benar **sampai derajat** $\mu_{\text{medium}}(28.1) = 0.81$. Kondisi itu **terpenuhi** bila derajatnya mencapai ambang basis pengetahuan, 0,5 secara bawaan. Ketika sebuah node menyala, derajat itu menjadi bagian dari kepastian konklusinya — bentuk fuzzy dari aturan kepastian klasik [9] yang dipakai di Bab 8:

$$CF_{\text{conclusion}} = \frac{\min_i(CF_{\text{fact }i} \times \mu_i) \times CF_{\text{rule}}}{100} \tag{9.3}$$

Di sini μ_i adalah derajat kondisi ke-i (1 untuk kondisi tegas). Anggota yang rasionya sedang berderajat 0,81, dengan fakta ber-CF 100 dan aturan ber-CF 100, mendapat saran aturan itu dengan CF $100 \times 0.81 \times 100/100 = 81$. Sarannya tidak hanya diberikan; ia diberikan *beserta kekuatannya*, yang menyatakan seberapa jelas angka-angka anggota itu cocok dengan kata-kata dalam aturan.

Ada dua akibat praktis. Pertama, pengetahuan pengurus dapat diajarkan dalam kata-kata, persis seperti pengurus berbicara: sebuah kasus menyebut `repayment ratio = medium`, `savings cover = enough`, dan tidak menyebut persentase apa pun. Kedua, anggota yang tepat berada di batas dua kata — rasio 25 persen, rendah berderajat 0,5 dan sedang berderajat 0,5 — memenuhi *kedua* kondisi, dan mesin dapat diminta mengikuti keduanya. Begitulah penasihat menemukan kasus di batas tipis.

Bagaimana basis pengetahuan yang sama dapat diajari dengan kata-kata tetapi ditanyai dengan angka? Karena mesin menentukan cara membandingkan dengan melihat faktanya. Ketika Bu Wati mengajarkan kasus `repayment ratio = medium`, faktanya adalah kata *medium*, dan kondisi `repayment ratio = medium` sekadar sama dengannya, seperti di semua bab sebelumnya; kasus itu dipelajari, diperiksa terhadap cornerstone-nya, dan ditempatkan dengan langkah-langkah RDR yang biasa. Ketika berkas seorang anggota diperiksa, faktanya adalah angka 28,1, dan kondisi yang sama diukur melalui himpunan fuzzy. Tidak ada yang perlu dikonversi dalam basis pengetahuan. Kata-kata itulah pengetahuannya; himpunan fuzzy adalah jembatan dari angka ke kata, dan pengurus dapat menggeser jembatan itu tanpa menyentuh pengetahuannya.

9.2.4 Dua jenis ketidaktepatan

Makalah VCIRS fuzzy menunjukkan bahwa sistem berbasis aturan menghadapi dua jenis ketidaktepatan yang berbeda, dan menanganinya dengan dua alat yang berbeda [6]. **Kekaburan** (*fuzziness*) ada dalam kata-kata: *rendah* tidak bertepi tajam. Itulah yang ditangani himpunan fuzzy dan derajat. **Ketidakpastian** (*uncertainty*) ada dalam fakta dan aturan: kita tidak yakin penghasilan anggota itu benar-benar Rp 3.600.000, dan pengurus tidak yakin aturannya selalu benar. Makalah itu memodelkan ketidakpastian dengan **bilangan fuzzy**, seperti *sekitar 0,8* atau

mendekati 1,0, dan menggabungkannya dengan aritmetika fuzzy; kepastian sebuah konklusi adalah hasil kali fuzzy antara ketidakpastian aturan dan ketidakpastian fakta.

Bilangan fuzzy segitiga (a, b, c) berarti "sekitar b, pasti tidak di bawah a dan tidak di atas c". Untuk segitiga-segitiga positif, hasil kalinya dapat didekati dengan baik dengan mengalikan sudut-sudutnya:

$$(a_1, b_1, c_1) \otimes (a_2, b_2, c_2) \approx (a_1 a_2,\ b_1 b_2,\ c_1 c_2) \tag{9.4}$$

Penasihat memakainya di satu tempat yang paling penting: seberapa pasti penghasilannya? Bila anggota menunjukkan buku penjualan, slip gaji, atau rekening koran, Bu Wati menandai penghasilannya *verified* (terverifikasi), sekitar 0,95, antara 0,85 dan 1; bila anggota hanya menyebutkannya, *stated* (dinyatakan), sekitar 0,7, antara 0,5 dan 0,9. CF saran, dibaca sebagai angka tegas (misalnya 0,81 sebagai segitiga (0,81, 0,81, 0,81)), dikalikan dengan segitiga penghasilan, dan hasilnya dicetak sebagai "how sure: about 0.57 (0.41 to 0.73)". Seorang pengurus langsung melihat apakah sebuah rekomendasi berpijak pada tanah yang kokoh.

9.2.5 Rasio angsuran dan garis 30 persen

Angka terpenting dalam setiap keputusan pinjaman adalah seberapa besar bagian penghasilan peminjam yang akan diambil angsurannya. Penasihat menyebutnya **rasio angsuran** (*repayment ratio*): semua angsuran bulanan anggota — kepada koperasi maupun ke tempat lain — termasuk angsuran pinjaman baru, dibagi penghasilan bulanannya:

$$r = \frac{E + I}{M} \times 100, \qquad I = \left\lceil \frac{A}{n} \right\rceil + A \times 1\% \tag{9.5}$$

dengan E angsuran yang sudah ada, M penghasilan bulanan, A pokok pinjaman baru, n jumlah bulan, dan I angsuran baru: pokok dibagi jumlah bulan, dibulatkan ke atas ke rupiah bulat, ditambah bunga tetap koperasi 1 persen dari pokok per bulan (bunga itu milik koperasi contoh; rapat anggota setiap koperasi menetapkan bunganya sendiri).

Di mana *rendah* seharusnya berakhir? Regulator memberi acuan yang berguna. Untuk pinjaman daring (*pindar*), OJK memperketat batas total angsuran peminjam secara bertahap, menjadi 30 persen dari penghasilan pada 2026 [10]. Aturan itu tidak mengikat koperasi *closed loop*, tetapi ia patokan yang masuk akal, dan pengurus memilih kata-katanya di sekitar patokan itu: pada 30 persen sebuah rasio sama sekali tidak lagi rendah dan sepenuhnya sedang; pegangan "jaga di bawah seperempat penghasilan" adalah titik tempat *rendah* dan *sedang* bersilangan.

Perhatikan bahwa rasio itu menghitung *semua* angsuran, bukan hanya angsuran kepada koperasi. Anggota yang sudah membayar cicilan motor kepada diler, cicilan ponsel kepada toko, dan pinjaman daring kecil mungkin tampak longgar bila hanya dilihat dari sisi koperasi, padahal dalam kenyataan napasnya sudah pendek. Bu Wati meminta setiap pemohon mendaftar semua angsurannya, dan ia mencatat apa yang dikatakan anggota meskipun tidak dapat diperiksa — baris bukti penghasilan akan memperlihatkan seberapa kokoh berkas itu.

9.2.6 Adil dan terbuka sejak rancangannya

Penasihat pinjaman menentukan siapa yang mendapat uang, dan uang mengubah hidup orang. Peraturan pelindungan konsumen sektor jasa keuangan Indonesia, POJK Nomor 22 Tahun 2023, mencantumkan perlakuan yang adil dan keterbukaan di antara prinsip yang wajib diikuti pelaku usaha jasa keuangan [11], dan koperasi yang ingin dipercaya anggotanya patut memegang

standar yang sama. Penasihat melakukan tiga hal demi keadilan dan keterbukaan, dan ketiganya dibangun ke dalam strukturnya, bukan diserahkan pada niat baik.

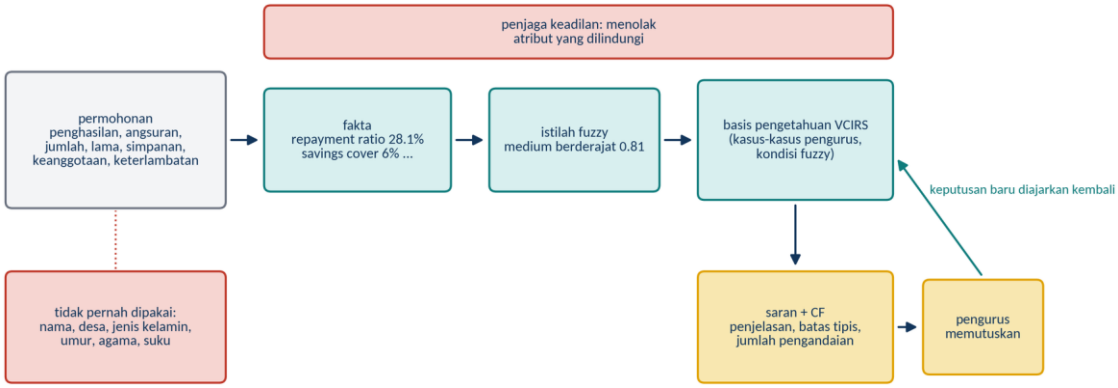
Pertama, **tidak ada atribut yang dilindungi**. Penasihat hanya melihat lima fakta, semuanya tentang kemampuan membayar: rasio angsuran, lama keanggotaan dalam bulan, cakupan simpanan, banyaknya keterlambatan, dan pola penghasilan (harian, bulanan, atau saat panen). Formulir permohonan juga memuat nama anggota, desanya, dan sebagainya; semua itu diperlihatkan kepada pengurus tetapi tidak pernah diberikan kepada mesin. Dan basis pengetahuan sendiri menolak kasus apa pun yang menyebut atribut yang dilindungi — agama, suku, jenis kelamin, status perkawinan, umur, disabilitas, pandangan politik, atau asal daerah, dalam bahasa Inggris maupun Indonesia. Koperasi yang ingin berhati-hati juga dapat mewaspadai **proksi**: variabel yang tampak tidak berbahaya tetapi mewakili atribut yang dilindungi, seperti nama dusun tempat satu kelompok tinggal. Aturannya sederhana: setiap variabel harus tentang kemampuan membayar, dan pengurus harus dapat mengatakan alasannya.

Kedua, **alasan untuk setiap jawaban**. Setiap rekomendasi datang bersama aturan yang menghasilkannya dan derajat kecocokan angka-angka anggota dengan setiap kata, sebagai penjelasan How dari mesin. Anggota yang bertanya "mengapa saya disarankan meminjam lebih sedikit?" dapat ditunjukkan: *rasio angsuran Anda 28,1 persen termasuk sedang berderajat 0,81, dan aturan pengurus untuk rasio sedang adalah menyarankan jumlah yang lebih kecil.*

Ketiga, **apa yang akan mengubah jawabannya**. Untuk setiap anggota yang rasionya tidak rendah, penasihat menghitung pinjaman terbesar, dengan langkah Rp 100.000, yang akan menurunkan rasionya menjadi *rendah* (25 persen atau kurang). "Rp 4.800.000 masih rendah untuk Anda" lebih berguna dan lebih menghormati daripada "tidak".

Ketiganya bersama-sama membuat penasihat dapat **diaudit**. Pengetahuannya adalah berkas kasus dalam kata-kata yang dapat dibaca; himpunan fuzzy-nya adalah berkas angka yang dapat dibaca; setiap perubahan tercatat dalam riwayat beserta nama orang yang mengajarkannya; dan setiap rekomendasi dapat dihasilkan ulang dari permohonan dan basis pengetahuan bulan itu. Ketika seorang anggota, auditor, atau pengawas bertanya bagaimana sebuah keputusan disiapkan, koperasi dapat memperlihatkannya, baris demi baris.

Gambar 9.2 memperlihatkan seluruh perjalanan sebuah permohonan.



Gambar 9.2 Perjalanan sebuah permohonan melalui penasihat. Hanya fakta tentang kemampuan membayar yang sampai ke mesin; penjaga keadilan menolak pengetahuan apa pun yang memakai atribut yang dilindungi; saran datang bersama CF, penjelasan, peringatan batas tipis, dan jumlah pengandaian; pengurus memutuskan, dan keputusannya diajarkan kembali.

9.2.7 Satu anggota, dari awal sampai akhir

Sebelum menulis kode, ikutilah satu berkas dengan tangan, seperti yang akan dilakukan penasihat. Mas Joko, pengemudi ojek, mengajukan Rp 6.000.000 untuk 12 bulan. Penghasilannya Rp 3.600.000 sebulan, dinyatakan tetapi tidak berdokumen; ia sudah membayar Rp 450.000 sebulan untuk motornya; simpanannya Rp 400.000; ia sudah 30 bulan menjadi anggota dan tahun lalu sekali terlambat.

Langkah 1, angka-angkanya. Menurut Persamaan (9.5) angsuran barunya $I = 500{,}000 + 60{,}000 = 560{,}000$ dan rasio angsurannya $r = 1{,}010{,}000/3{,}600{,}000 \times 100 \approx 28.1$ persen. Simpanannya mencakup $400{,}000/6{,}000{,}000$, 6 persen dari pinjaman. (Dalam rumus, pemisah ribuan ditulis dengan koma, seperti dalam keluaran program.)

Langkah 2, kata-katanya. Menurut Persamaan (9.1), rasio 28,1 termasuk *rendah* berderajat 0,19 dan *sedang* berderajat 0,81. Keanggotaan tiga puluh bulan termasuk *mapan* berderajat 1 dan *baru* berderajat 0. Cakupan simpanan 6 persen termasuk *kecil* berderajat 1 dan *cukup* berderajat 0. Satu kali terlambat termasuk *jarang* berderajat 1 dan *sering* berderajat 0.

Langkah 3, aturan-aturannya. Node *rendah* memerlukan derajat 0,5; 0,19 tidak cukup, sehingga node itu dan pengecualian-pengecualiannya tidak menyala. Node *sedang* terpenuhi dengan 0,81 dan menyala: *recommend a smaller amount*, sarankan jumlah yang lebih kecil. Pengecualiannya memerlukan simpanan *cukup*, berderajat 0: tidak menyala. Node *tinggi* memerlukan *tinggi*, berderajat 0.

Langkah 4, kekuatannya. Menurut Persamaan (9.3), $CF = 100 \times 0.81 \times 100/100 = 81$. Penghasilannya hanya dinyatakan, jadi menurut Persamaan (9.4) rekomendasinya *sekitar 0,57*, antara 0,41 dan 0,73.

Langkah 5, jalan keluarnya. Agar rasionya paling tinggi 25 persen, semua angsurannya paling banyak $0.25 \times 3{,}600{,}000 = 900{,}000$, sehingga angsuran barunya paling banyak 450,000. Angsuran kira-kira $A/12 + A/100$ dari pokok A, jadi A paling banyak sekitar $450{,}000/(1/12 + 1/100) \approx 4{,}821{,}000$; dengan langkah Rp 100.000, hasilnya Rp 4.800.000.

Lima langkah kecil, yang masing-masing dapat diperiksa pengurus dengan kalkulator. Subbab 9.6 memperlihatkan penasihat mencetak persis angka-angka ini.

9.2.8 Mengapa bukan skor kredit?

Bank dan pemberi pinjaman daring biasanya memutuskan dengan **skor kredit**: model statistik, dilatih pada ribuan peminjam terdahulu, yang mengubah permohonan menjadi sebuah angka. Bagi pemberi pinjaman besar dengan riwayat yang panjang, cara itu bekerja baik. Koperasi desa tidak punya riwayat sepanjang itu dan tidak punya ahli statistik, tetapi ia punya sesuatu yang tidak dimiliki skor: pengurus yang mengenal anggotanya dan dapat mengatakan mengapa ia memutuskan. Pendekatan VCIRS menyimpan pengetahuan dalam kata-kata pengurus sendiri dan menumbuhkannya satu keputusan demi satu keputusan, persis seperti Ripple Down Rules menumbuhkan pengetahuan dokter Wi-Fi. Setiap aturan ditaruh di sana oleh seseorang karena suatu alasan, setiap jawaban dapat dibacakan keras-keras, dan jawaban yang salah dibetulkan di tempat kesalahannya terjadi, tanpa melatih ulang apa pun.

Ada pula argumen keadilan. Model statistik dapat belajar memakai atribut yang dilindungi melalui proksi tanpa ada yang menyadarinya, karena tak seorang pun membaca bobot-bobotnya. Basis pengetahuan berisi beberapa puluh aturan yang dapat dibaca, dengan penjaga yang

menolak variabel yang dilindungi, dapat diperiksa oleh anggota koperasi mana pun dalam sepuluh menit.

9.3 Derajat Keanggotaan dengan Tangan

Seperti di bab-bab sebelumnya, kita menghitung derajat-derajat itu sendiri lebih dulu. **Kode 9.1** menuliskan trapesium Persamaan (9.1) dalam belasan baris, menerapkan hedge Persamaan (9.2) dengan tangan, dan meminta mesin menghitung derajat yang sama melalui sebuah kondisi, untuk memeriksa bahwa keduanya sama.

Kode 9.1 `FuzzyTerms.java`: derajat keanggotaan dan hedge dengan tangan, dicocokkan dengan vcirs-core.

```
import java.util.List;
import java.util.Map;

import vcirs.Condition;
import vcirs.Fact;
import vcirs.KnowledgeBase;
import vcirs.Num;

/**
 * Seberapa benar "rasio angsuran rendah"? Derajat keanggotaan dengan
 * tangan, dengan hedge "very" dan "somewhat", dicocokkan dengan vcirs-core.
 */
public class FuzzyTerms {
    static final String[] NAMES = {"low", "medium", "high"};
    static final double[][] POINTS = {{0, 0, 20, 30}, {20, 30, 30, 40},
        {30, 40, 100, 100}};

    /** Trapesium: 0 sampai a, naik ke 1 di b, 1 sampai c, 0 lagi di d. */
    static double membership(double x, double[] p) {
        double a = p[0], b = p[1], c = p[2], d = p[3];
        if (b <= x && x <= c) {
            return 1.0;
        }
        if (a < x && x < b) {
            return (x - a) / (b - a);
        }
        if (c < x && x < d) {
            return (d - x) / (d - c);
        }
        return 0.0;
    }

    static String pad(String s, int width) {
        return String.format("%-" + width + "s", s);
    }
```

Kode 9.1 (Lanjutan) *FuzzyTerms.java: derajat keanggotaan dan hedge dengan tangan, dicocokkan dengan vcirs-core.*

```
    public static void main(String[] args) {
        KnowledgeBase kb = new KnowledgeBase("terms");
        for (int i = 0; i < NAMES.length; i++) {
            kb.defineFuzzy("repayment ratio", NAMES[i], POINTS[i]);
        }
        System.out.println("ratio  low   medium  high  very high  somewhat low"
                + "  engine agrees");
        List<String> asked = List.of("low", "medium", "high", "very high",
                "somewhat low");
        for (double x : new double[] {15, 20, 25, 28.1, 30, 33.5, 35, 40, 45}) {
            double low = membership(x, POINTS[0]);
            double med = membership(x, POINTS[1]);
            double high = membership(x, POINTS[2]);
            double[] byHand = {low, med, high, high * high, Math.sqrt(low)};
            Map<String, Fact> fact = Map.of("repayment ratio",
                    new Fact("repayment ratio", Num.fmt(x, 1)));
            boolean same = true;
            for (int i = 0; i < asked.size(); i++) {
                double e = kb.evaluate(new Condition("repayment ratio", "=",
                        asked.get(i)), fact);
                same = same && Num.key(e) == Num.key(byHand[i]);
            }
            String[] cells = new String[5];
            for (int i = 0; i < 5; i++) {
                cells[i] = Num.fixed(byHand[i], 2);
            }
            System.out.println(String.format("%5s", Num.fmt(x, 1)) + "  "
                    + cells[0] + "  " + pad(cells[1], 6) + "  " + cells[2]
                    + "  " + pad(cells[3], 9) + "  " + pad(cells[4], 12) + "  "
                    + (same ? "yes" : "NO"));
        }
    }
}
```

`membership` adalah Persamaan (9.1) baris demi baris. `main` mendefinisikan ketiga kata itu bagi mesin dengan `defineFuzzy`, lalu, untuk sembilan rasio, menghitung dengan tangan derajat *low*, *medium*, *high*, *very high* (kuadrat *high*), dan *somewhat low* (akar kuadrat *low*), serta meminta mesin melakukan `evaluate` atas kondisi `repayment ratio = low`, `= medium`, `= high`, `= very high`, dan `= somewhat low` terhadap faktanya. Angka-angka dibandingkan melalui `Num.key`, cara mesin memutuskan bahwa dua bilangan *double* sama. **Kode 9.2** adalah versi Python-nya.

Kode 9.2 `fuzzy_terms.py`: tabel yang sama dalam Python.

```
"""Seberapa benar "rasio angsuran rendah"? Derajat keanggotaan dengan
tangan, dengan hedge "very" dan "somewhat", dicocokkan dengan vcirs-core."""

import math
import pathlib
import sys

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import Condition, Fact, KnowledgeBase, num

TERMS = {"low": (0, 0, 20, 30), "medium": (20, 30, 30, 40),
         "high": (30, 40, 100, 100)}


def membership(x, points):
    """Trapesium: 0 sampai a, naik ke 1 di b, 1 sampai c, 0 lagi di d."""
    a, b, c, d = points
    if b <= x <= c:
        return 1.0
    if a < x < b:
        return (x - a) / (b - a)
    if c < x < d:
        return (d - x) / (d - c)
    return 0.0


kb = KnowledgeBase("terms")
for term, points in TERMS.items():
    kb.define_fuzzy("repayment ratio", term, list(points))

print("ratio  low   medium  high  very high  somewhat low  engine agrees")
for x in [15, 20, 25, 28.1, 30, 33.5, 35, 40, 45]:
    low, med, high = (membership(x, TERMS[t]) for t in TERMS)
    by_hand = [low, med, high, high * high, math.sqrt(low)]
    fact = {"repayment ratio": Fact("repayment ratio", num.fmt(x, 1))}
    engine = [kb.evaluate(Condition("repayment ratio", "=", t), fact)
              for t in ["low", "medium", "high", "very high", "somewhat low"]]
    same = all(num.key(a) == num.key(b) for a, b in zip(by_hand, engine))
    cells = [num.fixed(v, 2) for v in by_hand]
    print(num.fmt(x, 1).rjust(5) + "  " + cells[0] + "  " + cells[1].ljust(6)
          + "  " + cells[2] + "  " + cells[3].ljust(9) + "  "
          + cells[4].ljust(12) + "  " + ("yes" if same else "NO"))
```

Keduanya mencetak:

```
ratio  low   medium  high  very high  somewhat low  engine agrees
   15  1.00  0.00    0.00  0.00       1.00          yes
   20  1.00  0.00    0.00  0.00       1.00          yes
   25  0.50  0.50    0.00  0.00       0.71          yes
 28.1  0.19  0.81    0.00  0.00       0.44          yes
   30  0.00  1.00    0.00  0.00       0.00          yes
 33.5  0.00  0.65    0.35  0.12       0.00          yes
   35  0.00  0.50    0.50  0.25       0.00          yes
   40  0.00  0.00    1.00  1.00       0.00          yes
   45  0.00  0.00    1.00  1.00       0.00          yes
```

Bacalah baris-barisnya seperti pengurus membacanya. Pada 15 dan 20 persen rasionya jelas rendah. Pada 25 ia tepat setengah rendah dan setengah sedang, dan *agak rendah* berderajat 0,71: anggota di sini berada di pagar batas, dan penasihat akan mengatakannya. Pada 28,1, rasio Mas Joko di Subbab 9.6, ia sedang berderajat 0,81 dan rendah hanya 0,19, di bawah ambang. Pada 33,5 ia sedang berderajat 0,65 dan tinggi 0,35 — dan *sangat tinggi* hanya 0,12, karena mengkuadratkan derajat yang kecil membuatnya jauh lebih kecil: *sangat* adalah kata yang

menuntut. Dari 40 ke atas ia jelas tinggi. Pada setiap baris mesin sependapat dengan hitungan tangan.

Mengapa trapesium, dan bukan kurva halus seperti lonceng? Karena pengurus dapat menetapkan trapesium dengan empat angka yang dipahaminya — "pasti rendah sampai 20, pasti tidak rendah mulai 30" — dan memeriksanya dengan pensil. Kurva halus memberi saran yang hampir sama tetapi jauh lebih sulit dijelaskan kepada anggota. Makalah VCIRS fuzzy pun membangun istilah-istilah fuzzy-nya dari bentuk-bentuk sederhana [6]. Kesederhanaan adalah bagian dari keterbukaan. Bentuk yang dapat digambar di papan tulis balai desa adalah bentuk yang dapat diperdebatkan dan disepakati bersama oleh seluruh pengurus.

9.4 Hitungan Koperasi

Kode 9.3 memuat semua yang dihitung penasihat dari sebuah permohonan, dalam rupiah bulat agar Java dan Python sepakat sampai angka terakhir.

Kode 9.3 `Loans.java`: angsuran, rasio angsuran, dan jumlah pengandaian, dalam rupiah bulat.

```java
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;

/**
 * Hitungan pinjaman koperasi, dalam rupiah bulat. Koperasi meminjamkan
 * dengan bunga tetap 1% dari pokok per bulan (rapat anggota menetapkan
 * bunganya). Angsuran adalah pokok dibagi jumlah bulan, dibulatkan ke atas,
 * ditambah bunga bulanan.
 */
public class Loans {
    static final long RATE_BP = 100;  // bunga per bulan, dalam 1/100 %

    static long installment(long amount, long months) {
        return (amount + months - 1) / months + amount * RATE_BP / 10000;
    }

    /** 100 * pembilang / penyebut, dibulatkan satu desimal, dalam persepuluhan. */
    static long tenths(long numerator, long denominator) {
        return (numerator * 1000 + denominator / 2) / denominator;
    }

    /** 253 -> "25.3" */
    static String decimal(long t) {
        return t / 10 + "." + t % 10;
    }

    /** 4800000 -> "Rp 4,800,000" */
    static String rupiah(long x) {
        return "Rp " + String.format(Locale.ROOT, "%,d", x);
    }

    static class Application {
        static final String[] FIELDS = {"id", "name", "work", "income pattern",
            "monthly income", "existing installments", "amount", "months",
            "savings", "membership months", "late payments", "income proof"};
        final Map<String, String> data = new LinkedHashMap<>();
```

Kode 9.3 (Lanjutan) *Loans.java: angsuran, rasio angsuran, dan jumlah pengandaian, dalam rupiah bulat.*

```
        Application(String line) {
            String[] parts = line.strip().split(",");
            for (int i = 0; i < FIELDS.length; i++) {
                data.put(FIELDS[i], parts[i]);
            }
        }

        long number(String field) {
            return Long.parseLong(data.get(field));
        }

        /** Rasio angsuran: semua angsuran / penghasilan, dalam 1/10 %. */
        long ratioTenths(long amount) {
            long total = number("existing installments")
                    + installment(amount, number("months"));
            return tenths(total, number("monthly income"));
        }

        long ratioTenths() {
            return ratioTenths(number("amount"));
        }

        /**
         * Yang boleh dilihat penasihat. Nama, desa, jenis kelamin, umur,
         * atau agama tidak pernah menjadi fakta.
         */
        Map<String, String> facts() {
            Map<String, String> f = new LinkedHashMap<>();
            f.put("repayment ratio", decimal(ratioTenths()));
            f.put("membership", String.valueOf(number("membership months")));
            f.put("savings cover",
                    String.valueOf(number("savings") * 100 / number("amount")));
            f.put("late payments", String.valueOf(number("late payments")));
            f.put("income pattern", data.get("income pattern"));
            return f;
        }
```

Kode 9.3 (Lanjutan) *Loans.java: angsuran, rasio angsuran, dan jumlah pengandaian, dalam rupiah bulat.*

```
        /**
         * Pokok terbesar, dengan langkah Rp 100.000, yang menjaga rasio
         * angsuran pada atau di bawah batas (null bila tidak ada).
         */
        Long largestLow(long limitTenths) {
            long amount = number("amount");
            while (amount > 0 && ratioTenths(amount) > limitTenths) {
                amount -= 100000;
            }
            return amount > 0 ? amount : null;
        }
    }

    static List<Application> readApplications(String path) {
        List<Application> out = new ArrayList<>();
        try {
            for (String line : Files.readAllLines(Path.of(path),
                    StandardCharsets.UTF_8)) {
                if (!line.isBlank() && !line.startsWith("#")) {
                    out.add(new Application(line));
                }
            }
        } catch (IOException e) {
            throw new IllegalStateException("cannot read " + path);
        }
        return out;
    }
}
```

`installment` adalah *I* dalam Persamaan (9.5): pokok dibagi jumlah bulan dan dibulatkan ke atas, ditambah 1 persen dari pokok (`RATE_BP` adalah bunga dalam seperseratus persen, sehingga koperasi dapat mengubahnya di satu tempat). `tenths` menghitung persentase yang dibulatkan satu desimal, sebagai bilangan bulat persepuluhan, hanya dengan bilangan bulat; `decimal` menuliskan 281 sebagai `28.1`, dan `rupiah` menuliskan 4800000 sebagai `Rp 4,800,000` (program memakai koma sebagai pemisah ribuan, sama di kedua bahasa).

Sebuah `Application` adalah satu baris `applications.csv`:

```
# id,name,work,income pattern,monthly income,existing installments,amount,...
A01,Bu Siti,vegetable seller,daily,4500000,300000,5000000,10,1200000,48,0,verified
A02,Mas Joko,ojek driver,daily,3600000,450000,6000000,12,400000,30,1,stated
```

Kolom-kolom selanjutnya adalah lama pinjaman dalam bulan, simpanan anggota, lama keanggotaan dalam bulan, keterlambatan dalam dua belas bulan terakhir, dan apakah penghasilannya *verified* atau *stated*. `ratioTenths` menerapkan Persamaan (9.5). `facts` adalah metode terpenting dalam bab ini dari sisi keadilan: ia membangun **satu-satunya** fakta yang pernah dilihat mesin — rasio, lama keanggotaan, cakupan simpanan sebagai persentase bulat, keterlambatan, dan pola penghasilan. `largestLow` menurunkan pokok dengan langkah Rp 100.000 sampai rasionya 25,0 persen atau kurang. Mengapa rupiah bulat dan persepuluhan persen, bukan bilangan desimal biasa? Ada dua alasan. Uang tidak boleh dibulatkan berbeda di dua tempat — angsuran yang tercetak Rp 560.000 di satu halaman dan Rp 559.999,99 di halaman lain merusak kepercayaan pada seluruh lembarnya. Dan versi Java dan Python harus mencetak hal yang sama, yang hanya terjamin bila hitungannya dilakukan dengan bilangan bulat; satu-satunya bilangan desimal dalam penasihat adalah derajat dan kepastian, yang dicetak vcirs-core dengan cara yang sama di kedua bahasa. **Kode 9.4** adalah versi Python-nya.

Kode 9.4 `loans.py`: hitungan yang sama dalam Python.

```
"""Hitungan pinjaman koperasi, dalam rupiah bulat.

Koperasi meminjamkan dengan bunga tetap 1% dari pokok per bulan (rapat
anggota menetapkan bunganya). Angsuran adalah pokok dibagi jumlah bulan,
dibulatkan ke atas, ditambah bunga bulanan.
"""

RATE_BP = 100                       # bunga per bulan, dalam 1/100 persen


def installment(amount, months):
    return (amount + months - 1) // months + amount * RATE_BP // 10000


def tenths(numerator, denominator):
    """100 * pembilang / penyebut, dibulatkan satu desimal, sebagai 1/10."""
    return (numerator * 1000 + denominator // 2) // denominator


def decimal(t):
    """253 -> '25.3'"""
    return str(t // 10) + "." + str(t % 10)


def rupiah(x):
    """4800000 -> 'Rp 4,800,000'"""
    return "Rp " + "{:,}".format(x)


class Application:
    FIELDS = ["id", "name", "work", "income pattern", "monthly income",
              "existing installments", "amount", "months", "savings",
              "membership months", "late payments", "income proof"]

    def __init__(self, line):
        self.data = dict(zip(Application.FIELDS, line.strip().split(",")))

    def number(self, field):
        return int(self.data[field])
```

Kode 9.4 (Lanjutan) *loans.py: hitungan yang sama dalam Python.*

```
    def ratio_tenths(self, amount=None):
        """Rasio angsuran: semua angsuran / penghasilan, dalam 1/10 %."""
        amount = self.number("amount") if amount is None else amount
        total = self.number("existing installments") \
            + installment(amount, self.number("months"))
        return tenths(total, self.number("monthly income"))

    def facts(self):
        """Yang boleh dilihat penasihat. Nama, desa, jenis kelamin, umur,
        atau agama tidak pernah menjadi fakta."""
        amount = self.number("amount")
        return {
            "repayment ratio": decimal(self.ratio_tenths()),
            "membership": str(self.number("membership months")),
            "savings cover": str(self.number("savings") * 100 // amount),
            "late payments": str(self.number("late payments")),
            "income pattern": self.data["income pattern"],
        }

    def largest_low(self, limit_tenths=250):
        """Pokok terbesar, dengan langkah Rp 100.000, yang menjaga rasio
        angsuran pada atau di bawah batas (None bila tidak ada)."""
        amount = self.number("amount")
        while amount > 0 and self.ratio_tenths(amount) > limit_tenths:
            amount -= 100000
        return amount if amount > 0 else None


def read_applications(path):
    with open(path, encoding="utf-8") as f:
        return [Application(line) for line in f
                if line.strip() and not line.startswith("#")]
```

Untuk Mas Joko, menurut Persamaan (9.5): $I = \lceil 6{,}000{,}000/12 \rceil + 60{,}000 = 560{,}000$; dengan $E = 450{,}000$ dan $M = 3{,}600{,}000$, $r = 1{,}010{,}000/3{,}600{,}000 \times 100 = 28.06$, yang dibulatkan menjadi 28,1 persen.

9.5 Penasihatnya

Pengetahuan pengurus berupa tujuh kasus dalam `loan-cases.json`, diajarkan Bu Wati dari keputusan-keputusan terdahulu dan dibangun dengan `learn`, putaran RDR sehari-hari dari Bab 3:

```
repayment ratio = low                          -> recommend
repayment ratio = medium                       -> recommend a smaller amount
repayment ratio = high                         -> not now: lower the amount or save first
repayment ratio = low, late payments = frequent      -> refer to the committee
repayment ratio = low, membership = new              -> start with a smaller first loan
repayment ratio = medium, savings cover = enough     -> recommend
repayment ratio = medium, savings cover = enough,
    late payments = frequent                         -> refer to the committee
```

(Ini ringkasannya; di dalam berkas setiap kasus satu baris JSON.) Tiga yang pertama menjadi node tingkat puncak: *recommend* (sarankan), *recommend a smaller amount* (sarankan jumlah lebih kecil), dan *not now* (belum sekarang: kurangi jumlahnya atau menabung dulu). Yang keempat dan kelima adalah jawaban salah dari node pertama, sehingga `learn` menggantungkannya di bawahnya sebagai pengecualian: *rendah, tetapi sering terlambat* dibawa ke pengurus (*refer to the committee*); *rendah, tetapi anggota baru* dimulai dengan pinjaman pertama yang kecil. Yang keenam adalah pengecualian node sedang — anggota dengan simpanan yang cukup untuk menutup sebagian besar pinjaman boleh meminjam jumlah penuh — dan yang

ketujuh pengecualian dari yang keenam. Semuanya dalam kata-kata; angka baru datang ketika saran diminta. **Kode 9.5** adalah penasihatnya.

Kode 9.5 `LoanAdvisor.java`: penasihat pinjaman fuzzy yang adil dan menjelaskan dirinya.

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import vcirs.Case;
import vcirs.CaseFile;
import vcirs.Condition;
import vcirs.Conclusion;
import vcirs.Facts;
import vcirs.InferenceResult;
import vcirs.KnowledgeBase;
import vcirs.Node;
import vcirs.Num;
import vcirs.RdrPath;

/**
 * Penasihat kelayakan pinjaman koperasi: VCIRS fuzzy dengan penjelasan.
 *   java LoanAdvisor advise        saran untuk setiap permohonan
 *   java LoanAdvisor teach         pelajari keputusan pengurus
 *   java LoanAdvisor fairness [cases.json]   periksa pengetahuannya
 * Penasihat menyiapkan rapat pengurus; pengurus yang memutuskan.
 */
public class LoanAdvisor {
    static final String DATA = "../data";
    static final String KB_FILE = DATA + "/loan-kb.json";
    static final List<String> PROTECTED = List.of("religion", "ethnicity",
            "gender", "marital status", "age", "disability", "political view",
            "place of origin", "agama", "suku", "jenis kelamin",
            "status perkawinan", "usia", "umur", "disabilitas",
            "pilihan politik", "asal daerah");
    // Seberapa pasti angka penghasilan, sebagai bilangan fuzzy segitiga.
    static final Map<String, double[]> PROOF = Map.of(
            "verified", new double[] {0.85, 0.95, 1.0},
            "stated", new double[] {0.5, 0.7, 0.9});
```

Kode 9.5 (Lanjutan) *LoanAdvisor.java: penasihat pinjaman fuzzy yang adil dan menjelaskan dirinya.*

```
    /** Dilempar bila pengetahuan memakai atribut yang dilindungi. */
    static class FairnessException extends RuntimeException {
        FairnessException(String message) {
            super(message);
        }
    }

    static void checkFair(String variable, String where) {
        if (PROTECTED.contains(variable.strip().toLowerCase())) {
            throw new FairnessException(where + " uses the protected attribute '"
                    + variable + "'");
        }
    }

    static List<String> lines(String name) {
        List<String> out = new ArrayList<>();
        try {
            for (String line : Files.readAllLines(Path.of(DATA, name),
                    StandardCharsets.UTF_8)) {
                if (!line.isBlank() && !line.startsWith("#")) {
                    out.add(line.strip());
                }
            }
        } catch (IOException e) {
            throw new IllegalStateException("cannot read " + name);
        }
        return out;
    }

    static KnowledgeBase buildKb(String casesFile) {
        CaseFile cases = CaseFile.read(DATA + "/" + casesFile);
        for (int i = 0; i < cases.steps.size(); i++) {
            for (Condition fact : cases.steps.get(i).aCase.facts) {
                checkFair(fact.variable, "case " + (i + 1));
            }
        }
        KnowledgeBase kb = new KnowledgeBase(cases.name, cases.description);
        for (String line : lines("fuzzy-sets.csv")) {
            String[] p = line.split(",");
            kb.defineFuzzy(p[0], p[1], new double[] {Double.parseDouble(p[2]),
                Double.parseDouble(p[3]), Double.parseDouble(p[4]),
                Double.parseDouble(p[5])});
        }
        return cases.build(kb);
    }
```

Kode 9.5 (Lanjutan) *LoanAdvisor.java: penasihat pinjaman fuzzy yang adil dan menjelaskan dirinya.*

```
    static KnowledgeBase openKb() {
        return Files.exists(Path.of(KB_FILE)) ? KnowledgeBase.load(KB_FILE)
                : buildKb("loan-cases.json");
    }

    static void adviseOne(KnowledgeBase kb, Loans.Application app) {
        Map<String, String> d = app.data;
        Map<String, String> facts = app.facts();
        System.out.println(d.get("id") + " " + d.get("name") + " ("
                + d.get("work") + "): " + Loans.rupiah(app.number("amount"))
                + " over " + d.get("months") + " months");
        System.out.println("  facts: repayment ratio "
                + facts.get("repayment ratio") + "%, membership "
                + facts.get("membership") + " months, savings cover "
                + facts.get("savings cover") + "%, late payments "
                + facts.get("late payments") + ", income "
                + facts.get("income pattern"));
        InferenceResult result = kb.infer(Facts.map(facts), null, true);
        if (result.paths.isEmpty()) {
            System.out.println("  advice: none; the committee decides");
            return;
        }
        RdrPath first = result.paths.get(0);
        String advice = first.conclusions().get(0).value;
        System.out.println("  advice: " + advice + "  (CF "
                + Num.fmt(first.cf, 2) + ")");
        System.out.println("  because:");
        for (String line : result.how()) {
            System.out.println("    " + line);
        }
        List<String> others = new ArrayList<>();
        for (RdrPath p : result.paths.subList(1, result.paths.size())) {
            if (!p.conclusions().isEmpty()
                    && !p.conclusions().get(0).value.equals(advice)) {
                others.add(p.conclusions().get(0).value);
            }
        }
        if (!others.isEmpty()) {
            System.out.println("  BORDERLINE, also possible: "
                    + String.join("; ", others));
        }
        double[] t = PROOF.get(d.get("income proof"));
        System.out.println("  how sure (income " + d.get("income proof")
                + "): about " + Num.fmt(first.cf / 100.0 * t[1], 2) + " ("
                + Num.fmt(first.cf / 100.0 * t[0], 2) + " to "
                + Num.fmt(first.cf / 100.0 * t[2], 2) + ")");
```

Kode 9.5 (Lanjutan) *LoanAdvisor.java: penasihat pinjaman fuzzy yang adil dan menjelaskan dirinya.*

```
        Long best = app.largestLow(250);
        if (best != null && best < app.number("amount")) {
            System.out.println("  " + Loans.rupiah(best) + " would bring the "
                    + "repayment ratio down to low ("
                    + Loans.decimal(app.ratioTenths(best)) + "%)");
        }
    }

    /** Istilah fuzzy yang paling memuat angka itu (yang pertama bila seri). */
    static String bestTerm(KnowledgeBase kb, String variable, String value) {
        if (!kb.fuzzySets.containsKey(variable)) {
            return value;
        }
        String best = null;
        double degree = -1;
        for (Map.Entry<String, Double> e : kb.fuzzify(variable,
                Double.parseDouble(value))) {
            if (Num.key(e.getValue()) > Num.key(degree)) {
                best = e.getKey();
                degree = e.getValue();
            }
        }
        return best;
    }

    static void teach(KnowledgeBase kb) {
        Map<String, Loans.Application> apps = new LinkedHashMap<>();
        for (Loans.Application a : Loans.readApplications(DATA
                + "/applications.csv")) {
            apps.put(a.data.get("id"), a);
        }
        for (String row : lines("decisions.csv")) {
            String[] r = row.split(",");
            Map<String, String> facts = apps.get(r[0]).facts();
            Condition chosen = Condition.parse(r[2]);
            checkFair(chosen.variable, "the decision for " + r[0]);
            List<Condition> terms = new ArrayList<>();
            for (Map.Entry<String, String> f : facts.entrySet()) {
                terms.add(new Condition(f.getKey(), "=",
                        bestTerm(kb, f.getKey(), f.getValue())));
            }
            Case c = new Case(terms, List.of(new Conclusion("advice", r[1])),
                    100);
            System.out.println(r[0] + ": " + c);
            InferenceResult result = kb.infer(Facts.map(facts));
            String fired = result.paths.isEmpty() ? null
```

Kode 9.5 (Lanjutan) *LoanAdvisor.java: penasihat pinjaman fuzzy yang adil dan menjelaskan dirinya.*

```
                    : result.paths.get(0).last().id;
            if (!result.conclusions().isEmpty()
                    && result.conclusions().get(0).value.equals(r[1])) {
                kb.refine(c, KnowledgeBase.ACCEPT, fired, null, "committee");
                System.out.println("  the advisor already agrees");
                continue;
            }
            Node node = kb.refine(c, KnowledgeBase.ADD_NEW, fired,
                    List.of(chosen), "committee");
            System.out.println("  learned " + node + " (exception of " + fired
                    + ")");
        }
        kb.save(KB_FILE);
        System.out.println("Saved loan-kb.json: " + kb.summary());
    }

    static void fairness(KnowledgeBase kb, List<Loans.Application> apps) {
        List<String> used = new ArrayList<>();
        for (String v : kb.variables.keySet()) {
            for (Node n : kb.nodes.values()) {
                if (n.variables().contains(v)) {
                    used.add(v);
                    break;
                }
            }
        }
        System.out.println("Variables the advice may depend on: "
                + String.join(", ", used));
        List<String> bad = new ArrayList<>();
        for (String v : used) {
            if (PROTECTED.contains(v.toLowerCase())) {
                bad.add(v);
            }
        }
        System.out.println("Protected attributes used: "
                + (bad.isEmpty() ? "none" : String.join(", ", bad)));
        int same = 0;
        for (Loans.Application app : apps) {
            Set<String> answers = new HashSet<>();
            for (String value : List.of("female", "male")) {
                Map<String, String> facts = new LinkedHashMap<>(app.facts());
                facts.put("gender", value);
                answers.add(kb.infer(Facts.map(facts)).conclusions().toString());
            }
            same += answers.size() == 1 ? 1 : 0;
        }
```

Kode 9.5 (Lanjutan) *LoanAdvisor.java: penasihat pinjaman fuzzy yang adil dan menjelaskan dirinya.*

```
        System.out.println("Same advice for gender = female and gender = male: "
                + same + " of " + apps.size() + " applications");
    }

    public static void main(String[] args) {
        String command = args.length > 0 ? args[0] : "advise";
        List<Loans.Application> apps = Loans.readApplications(DATA
                + "/applications.csv");
        if (command.equals("teach")) {
            teach(buildKb("loan-cases.json"));
        } else if (command.equals("fairness")) {
            KnowledgeBase kb;
            try {
                kb = args.length > 1 ? buildKb(args[1]) : openKb();
            } catch (FairnessException e) {
                System.out.println("Refused: " + e.getMessage() + ".");
                return;
            }
            fairness(kb, apps);
        } else {
            KnowledgeBase kb = openKb();
            System.out.println(kb.summary());
            for (Loans.Application app : apps) {
                adviseOne(kb, app);
            }
        }
    }
}
```

Membangun. `buildKb` membaca berkas kasus dan, sebelum apa pun, melewatkan setiap fakta dari setiap kasus melalui `checkFair`, yang melempar `FairnessException` untuk atribut yang dilindungi. Lalu ia mendefinisikan sembilan himpunan fuzzy dari `fuzzy-sets.csv` dengan `defineFuzzy` dan membangun basis pengetahuan dari kasus-kasusnya. `openKb` memakai `loan-kb.json` yang tersimpan bila pengurus sudah mengajari penasihat, dan berkas kasus bila belum.

Memberi saran. `adviseOne` mencetak permohonan dan fakta-fakta anggota lalu menjalankan `kb.infer(facts, null, true)`: inferensi RDR dengan konklusi **ganda** (*multiple*), sehingga setiap node yang menyala pada lintasan terpisah dilaporkan. Lintasan pertama memberi saran beserta CF-nya, menurut Persamaan (9.3); penjelasan How mendaftar setiap node yang menyala, dengan derajat setiap kondisi fuzzy. Bila lintasan lain memberi saran yang *berbeda*, kasus itu **batas tipis** (*borderline*). Lalu datang ketidakpastian penghasilan, menurut Persamaan (9.4), dan jumlah pengandaian.

Mengajari. `teach` membaca `decisions.csv`, keputusan pengurus yang berbeda dari saran penasihat, masing-masing dengan fakta yang membuat perbedaan. Ia mengubah angka-angka anggota menjadi kata dengan `bestTerm` — setiap angka menjadi kata yang paling memuatnya, sehingga 54,0 persen menjadi *high* — karena pengetahuan disimpan dalam kata-kata. Lalu, persis seperti di Bab 7, ia memeriksa sarannya, menerimanya (opsi 0) bila penasihat sudah sependapat, atau menggantung keputusan pengurus sebagai pengecualian dari node yang memberi saran lama, dengan fakta yang disebut sebagai satu-satunya kondisinya (opsi 2 dengan kondisi pilihan). Fakta yang disebut itu juga diperiksa keadilannya.

Memeriksa keadilan. `fairness` mendaftar variabel-variabel yang dapat memengaruhi saran, memastikan tidak ada yang dilindungi, dan menjalankan **uji kontrafaktual**: ia memberikan setiap permohonan dua kali kepada mesin, sekali dengan tambahan `gender = female` dan sekali dengan `gender = male`, lalu menghitung permohonan yang sarannya tetap sama. Karena tidak

ada node yang menyebut jenis kelamin, jawabannya harus semuanya; uji itu ada agar siapa pun dapat melihatnya, dan agar perubahan di masa depan yang merusaknya ketahuan.

Kode 9.6 adalah versi Python-nya.

Kode 9.6 `loan_advisor.py`: penasihat yang sama dalam Python.

```
"""Penasihat kelayakan pinjaman koperasi: VCIRS fuzzy dengan penjelasan.

    python loan_advisor.py advise        saran untuk setiap permohonan
    python loan_advisor.py teach         pelajari keputusan pengurus
    python loan_advisor.py fairness [cases.json]   periksa pengetahuannya

Penasihat menyiapkan rapat pengurus; pengurus yang memutuskan.
"""

import os
import pathlib
import sys

# Beri tahu Python letak mesin vcirs-core (naik dua folder, lalu masuk).
sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import (ACCEPT, ADD_NEW, Case, Condition, Conclusion,
                   KnowledgeBase, casefile, num)
from loans import decimal, read_applications, rupiah

DATA = os.path.join(os.path.dirname(os.path.abspath(__file__)), "..", "data")
KB_FILE = os.path.join(DATA, "loan-kb.json")
PROTECTED = ["religion", "ethnicity", "gender", "marital status", "age",
             "disability", "political view", "place of origin", "agama",
             "suku", "jenis kelamin", "status perkawinan", "usia", "umur",
             "disabilitas", "pilihan politik", "asal daerah"]
# Seberapa pasti angka penghasilan, sebagai bilangan fuzzy segitiga.
PROOF = {"verified": (0.85, 0.95, 1.0), "stated": (0.5, 0.7, 0.9)}


class FairnessError(Exception):
    """Dilempar bila pengetahuan memakai atribut yang dilindungi."""


def check_fair(variable, where):
    if variable.strip().lower() in PROTECTED:
        raise FairnessError(where + " uses the protected attribute '"
                            + variable + "'")
```

Kode 9.6 (Lanjutan) *loan_advisor.py: penasihat yang sama dalam Python.*

```
def build_kb(cases_file):
    name, description, steps = casefile.read_cases(
        os.path.join(DATA, cases_file))
    for i, step in enumerate(steps):
        for fact in step.case.facts:
            check_fair(fact.variable, "case " + str(i + 1))
    kb = KnowledgeBase(name, description)
    with open(os.path.join(DATA, "fuzzy-sets.csv"), encoding="utf-8") as f:
        for line in f:
            if line.strip() and not line.startswith("#"):
                var, term, a, b, c, d = line.strip().split(",")
                kb.define_fuzzy(var, term, [a, b, c, d])
    return casefile.build(kb, steps)


def open_kb():
    return KnowledgeBase.load(KB_FILE) if os.path.exists(KB_FILE) \
        else build_kb("loan-cases.json")


def times(cf, triangle):
    """Kepastian tegas dikali bilangan fuzzy segitiga."""
    return tuple(cf / 100.0 * x for x in triangle)


def advise_one(kb, app):
    d = app.data
    facts = app.facts()
    print(d["id"] + " " + d["name"] + " (" + d["work"] + "): "
          + rupiah(app.number("amount")) + " over " + d["months"]
          + " months")
    print("  facts: repayment ratio " + facts["repayment ratio"]
          + "%, membership " + facts["membership"] + " months, savings "
          + "cover " + facts["savings cover"] + "%, late payments "
          + facts["late payments"] + ", income " + facts["income pattern"])
    result = kb.infer(facts, None, True)
    if not result.paths:
        print("  advice: none; the committee decides")
        return
    first = result.paths[0]
    print("  advice: " + first.conclusions()[0].value + "  (CF "
          + num.fmt(first.cf, 2) + ")")
    print("  because:")
    for line in result.how():
        print("    " + line)
    others = [p.conclusions()[0].value for p in result.paths[1:]
```

Kode 9.6 (Lanjutan) *loan_advisor.py: penasihat yang sama dalam Python.*

```
                if p.conclusions() and p.conclusions()[0].value
                != first.conclusions()[0].value]
    if others:
        print("  BORDERLINE, also possible: " + "; ".join(others))
    a, b, c = times(first.cf, PROOF[d["income proof"]])
    print("  how sure (income " + d["income proof"] + "): about "
          + num.fmt(b, 2) + " (" + num.fmt(a, 2) + " to " + num.fmt(c, 2)
          + ")")
    best = app.largest_low()
    if best is not None and best < app.number("amount"):
        print("  " + rupiah(best) + " would bring the repayment ratio down to low ("
              + decimal(app.ratio_tenths(best)) + "%)")


def best_term(kb, variable, value):
    """Istilah fuzzy yang paling memuat angka itu (yang pertama bila seri)."""
    if variable not in kb.fuzzy_sets:
        return value
    terms = kb.fuzzify(variable, float(value))
    return max(terms, key=lambda t: num.key(t[1]))[0]


def teach(kb):
    apps = {a.data["id"]: a for a in
            read_applications(os.path.join(DATA, "applications.csv"))}
    with open(os.path.join(DATA, "decisions.csv"), encoding="utf-8") as f:
        rows = [line.strip().split(",") for line in f
                if line.strip() and not line.startswith("#")]
    for member, advice, because in rows:
        facts = apps[member].facts()
        chosen = Condition.parse(because)
        check_fair(chosen.variable, "the decision for " + member)
        case = Case([Condition(v, "=", best_term(kb, v, x))
                     for v, x in facts.items()],
                    [Conclusion("advice", advice)])
        print(member + ": " + str(case))
        result = kb.infer(facts)
        fired = result.paths[0].last().id if result.paths else None
        if result.conclusions() and result.conclusions()[0].value == advice:
            kb.refine(case, ACCEPT, fired, author="committee")
            print("  the advisor already agrees")
            continue
        node = kb.refine(case, ADD_NEW, fired, [chosen], "committee")
        print("  learned " + str(node) + " (exception of " + fired + ")")
    kb.save(KB_FILE)
    print("Saved loan-kb.json: " + kb.summary())
```

Kode 9.6 (Lanjutan) *loan_advisor.py: penasihat yang sama dalam Python.*

```
def fairness(kb, apps):
    used = [v for v in kb.variables
            if any(v in n.variables() for n in kb.nodes.values())]
    print("Variables the advice may depend on: " + ", ".join(used))
    bad = [v for v in used if v.lower() in PROTECTED]
    print("Protected attributes used: " + (", ".join(bad) or "none"))
    same = 0
    for app in apps:
        answers = set()
        for value in ["female", "male"]:
            facts = dict(app.facts())
            facts["gender"] = value
            answers.add(str(kb.infer(facts).conclusions()))
        same += 1 if len(answers) == 1 else 0
    print("Same advice for gender = female and gender = male: " + str(same)
          + " of " + str(len(apps)) + " applications")


def main(args):
    command = args[0] if args else "advise"
    apps = read_applications(os.path.join(DATA, "applications.csv"))
    if command == "teach":
        teach(build_kb("loan-cases.json"))
    elif command == "fairness":
        try:
            kb = build_kb(args[1]) if len(args) > 1 else open_kb()
        except FairnessError as e:
            print("Refused: " + str(e) + ".")
            return
        fairness(kb, apps)
    else:
        kb = open_kb()
        print(kb.summary())
        for app in apps:
            advise_one(kb, app)


if __name__ == "__main__":
    main(sys.argv[1:])
```

9.6 Rapat Pengurus

Tujuh permohonan ada di meja bulan ini. Dari folder `Chapter09/java`, `java -cp out LoanAdvisor advise` (setelah kompilasi dengan `javac -d out -sourcepath ../../vcirs-core/java/src *.java`), atau dari `Chapter09/python`, `python loan_advisor.py advise`, mencetak lembar-lembar untuk pengurus. Dua yang pertama:

```
Koperasi Sumber Makmur loan advisor: 5 variables, 7 nodes, 7 rules
A01 Bu Siti (vegetable seller): Rp 5,000,000 over 10 months
  facts: repayment ratio 18.9%, membership 48 months, savings cover 24%, late payments 0,
income daily
  advice: recommend  (CF 100)
  because:
    recommend#1 fired because:
      repayment ratio = low  (repayment ratio = 18.9, given, CF 100)
      so advice = recommend  [rule CF 100, conclusion CF 100]
  how sure (income verified): about 0.95 (0.85 to 1)
A02 Mas Joko (ojek driver): Rp 6,000,000 over 12 months
  facts: repayment ratio 28.1%, membership 30 months, savings cover 6%, late payments 1,
income daily
  advice: recommend a smaller amount  (CF 81)
  because:
    recommend a smaller amount#1 fired because:
      repayment ratio = medium  (repayment ratio = 28.1, given, CF 100, to degree 0.81)
      so advice = recommend a smaller amount  [rule CF 100, conclusion CF 81]
  how sure (income stated): about 0.57 (0.41 to 0.73)
  Rp 4,800,000 would bring the repayment ratio down to low (24.9%)
```

Bu Siti berjualan sayur di pasar pagi. Angsurannya akan mengambil 18,9 persen penghasilannya: rendah, berderajat 1. Ia sudah empat tahun menjadi anggota, simpanannya mencakup 24 persen pinjaman, dan ia tidak pernah terlambat, sehingga tidak ada pengecualian yang menyala. Sarannya *recommend*, dengan CF 100, dan penghasilannya terverifikasi dengan buku penjualannya: *sekitar 0,95*.

Mas Joko mengemudi ojek. Angsuran barunya Rp 560.000, di atas Rp 450.000 yang sudah ia bayar untuk motornya, akan mengambil 28,1 persen penghasilannya: sedang berderajat 0,81. Simpanannya hanya mencakup 6 persen — *small* — sehingga pengecualian untuk simpanan yang cukup tidak menyala. Sarannya *recommend a smaller amount*, CF 81; penghasilannya hanya dinyatakan, jadi *sekitar 0,57, antara 0,41 dan 0,73*; dan Rp 4.800.000 alih-alih Rp 6.000.000 akan menurunkan rasionya menjadi 24,9 persen, rendah. Baris terakhir itulah yang akan dikatakan Bu Wati kepadanya: bukan “tidak”, melainkan “segini, bisa”.

Lalu Pak Darmo, petani padi:

```
A03 Pak Darmo (rice farmer): Rp 8,000,000 over 8 months
  facts: repayment ratio 54.0%, membership 72 months, savings cover 18%, late payments 0,
income harvest
  advice: not now: lower the amount or save first  (CF 100)
  because:
    not now: lower the amount or save first#1 fired because:
      repayment ratio = high  (repayment ratio = 54.0, given, CF 100)
      so advice = not now: lower the amount or save first  [rule CF 100, conclusion CF 100]
  how sure (income verified): about 0.95 (0.85 to 1)
  Rp 3,700,000 would bring the repayment ratio down to low (25.0%)
```

Rasionya 54,0 persen — *tinggi*, berderajat 1 — dan sarannya *not now*. Di atas kertas itu benar: Rp 1.080.000 sebulan dari penghasilan rata-rata Rp 2.000.000 terlalu berat. Tetapi pengurus mengenal Pak Darmo, dan mereka tahu sesuatu yang tidak tampak dalam angka: penghasilan petani padi tidak datang setiap bulan. Ia datang saat panen. Subbab 9.7 kembali kepadanya. Untuk sementara, cukuplah dicatat bahwa penasihat tidak salah menghitung; ia hanya belum pernah diajari tentang penghasilan yang datang saat panen, dan karena itu belum dapat melihat apa yang dilihat pengurus.

Sisa meja itu:

```
A04 Mbak Rani (tailor): Rp 4,000,000 over 10 months
  facts: repayment ratio 28.8%, membership 36 months, savings cover 50%, late payments 0,
income monthly
  advice: recommend  (CF 88)
  because:
    recommend#2 fired because:
      repayment ratio = medium  (repayment ratio = 28.8, given, CF 100, to degree 0.88)
      savings cover = enough  (savings cover = 50, given, CF 100)
      so advice = recommend  [rule CF 100, conclusion CF 88]
  how sure (income verified): about 0.84 (0.75 to 0.88)
  Rp 2,200,000 would bring the repayment ratio down to low (24.8%)
A05 Pak Hasan (warung owner): Rp 3,000,000 over 6 months
  facts: repayment ratio 8.8%, membership 4 months, savings cover 10%, late payments 0,
income daily
  advice: start with a smaller first loan  (CF 100)
  because:
    start with a smaller first loan#1 fired because:
      repayment ratio = low  (repayment ratio = 8.8, given, CF 100)
      membership = new  (membership = 4, given, CF 100)
      so advice = start with a smaller first loan  [rule CF 100, conclusion CF 100]
  how sure (income stated): about 0.7 (0.5 to 0.9)
A06 Bu Yati (caterer): Rp 5,000,000 over 10 months
  facts: repayment ratio 18.8%, membership 40 months, savings cover 10%, late payments 4,
income monthly
  advice: refer to the committee  (CF 100)
  because:
    refer to the committee#1 fired because:
      repayment ratio = low  (repayment ratio = 18.8, given, CF 100)
      late payments = frequent  (late payments = 4, given, CF 100)
      so advice = refer to the committee  [rule CF 100, conclusion CF 100]
  how sure (income verified): about 0.95 (0.85 to 1)
```

Mbak Rani, penjahit, rasionya sedang (28,8 persen, berderajat 0,88), tetapi simpanannya mencakup separuh pinjaman, sehingga pengecualian *sedang, simpanan cukup* menyala dan sarannya *recommend*, CF 88: derajat aturannya adalah yang terkecil dari derajat kedua kondisinya, 0,88 dan 1. **Pak Hasan** membuka warung dan bergabung dengan koperasi empat bulan lalu: rasionya nyaman, 8,8 persen, tetapi *membership = new* menyalakan pengecualian *start with a smaller first loan*, mulailah dengan pinjaman pertama yang lebih kecil. **Bu Yati**, pengusaha katering, rasionya rendah tetapi empat kali terlambat dalam setahun — *frequent* — dan pengecualian itu mengirim berkasnya kepada pengurus untuk diajak bicara, bukan untuk ditolak. Perhatikan juga baris-baris pengandaian. Untuk Pak Darmo, Rp 3.700.000 akan membawa rasionya ke 25,0 persen, angka yang mungkin berguna bagi pengurus meskipun, seperti diperlihatkan Subbab 9.7, pengurus akan memutuskan lain. Untuk Mbak Rani penasihat tetap mencetak Rp 2.200.000, walaupun sarannya sudah *recommend*: baris itu menjawab pertanyaan lain — "berapa yang rendah tanpa menghitung simpanan?" — dan pengurus boleh mengabaikannya bila simpanan sudah menyelesaikan persoalannya.

Terakhir, Mas Andi:

```
A07 Mas Andi (phone-credit seller): Rp 4,000,000 over 10 months
  facts: repayment ratio 25.0%, membership 24 months, savings cover 7%, late payments 0,
income daily
  advice: recommend  (CF 50)
  because:
    recommend#1 fired because:
      repayment ratio = low  (repayment ratio = 25.0, given, CF 100, to degree 0.5)
      so advice = recommend  [rule CF 100, conclusion CF 50]
    recommend a smaller amount#1 fired because:
      repayment ratio = medium  (repayment ratio = 25.0, given, CF 100, to degree 0.5)
      so advice = recommend a smaller amount  [rule CF 100, conclusion CF 50]
  BORDERLINE, also possible: recommend a smaller amount
  how sure (income stated): about 0.35 (0.25 to 0.45)
```

Rasionya tepat 25,0 persen: rendah berderajat 0,5 dan sedang berderajat 0,5. Kedua node tingkat puncak terpenuhi tepat pada ambangnya, inferensi ganda mengikuti keduanya, dan penasihat mencetak kedua penjelasan beserta kata **BORDERLINE**. Masing-masing saran ber-CF 50, yang terlemah yang masih mungkin dimiliki konklusi fuzzy yang menyala, dan penghasilannya hanya dinyatakan, sehingga seluruh rekomendasinya *sekitar 0,35*. Inilah persisnya berkas yang perlu dibahas pengurus: angka-angka tidak dapat memutuskannya, dan penasihat mengatakannya alih-alih berpura-pura.

Apa yang diputuskan pengurus pada Sabtu itu? Pengurus menyetujui Bu Siti, Mbak Rani, dan — sebesar Rp 4.800.000 — Mas Joko, seperti disarankan penasihat. Pengurus memberi Pak Hasan pinjaman pertama Rp 1.500.000, sesuai aturan untuk anggota baru, dan mengatakan bahwa setelah setahun membayar teratur ia boleh mengajukan lebih. Pengurus mengundang Bu Yati berbincang: keterlambatannya terjadi pada bulan-bulan ketika pelanggan katering besarnya membayar terlambat, dan pengurus menyepakati pinjaman dengan angsuran pada tanggal lima setiap bulan, setelah pelanggannya membayar. Pengurus membahas Mas Andi dan menyetujui pinjamannya, karena ia sudah dua tahun menabung teratur, tetapi memintanya membawa catatan penjualan lain kali agar penghasilannya dapat ditandai *verified*. Dan sebagian besar waktu rapat dipakai untuk Pak Darmo. Enam berkas selesai dalam setengah jam karena penasihat sudah menyiapkannya dengan cara yang sama seperti pengurus akan melakukannya; waktu rapat tercurah untuk dua berkas yang memang memerlukan manusia.

> **WAWASAN KUNCI:** Kekaburan mengubah mata pisau menjadi lereng. Dengan batas tegas, Mas Andi pada 25,0 persen akan mendapat satu jawaban dan tetangganya pada 25,1 persen jawaban sebaliknya, keduanya dengan keyakinan penuh. Dengan kata-kata fuzzy, keduanya mendapat jawaban yang lemah dengan CF yang jujur, dan kasus yang duduk di batas ditandai untuk diputuskan manusia.

9.7 Mengajarkan Keputusan Pengurus

Pengurus membahas Pak Darmo. Ia sudah enam tahun menjadi anggota dan tidak pernah terlambat. Pinjamannya untuk pupuk dan benih; setelah panen empat bulan lagi ia mampu membayar sebagian besar sekaligus. Pengurus menyetujui pinjaman itu dengan **jadwal bayar saat panen** (*pinjaman yarnen*, "bayar panen"), cara yang lazim di desa-desa pertanian. Bu Wati menuliskan keputusan itu di `decisions.csv`:

```
# id,the committee's advice,the fact that made the difference
A03,consider a harvest-time schedule,income pattern = harvest
```

lalu menjalankan `teach`:

```
A03: repayment ratio = high, membership = established, savings cover = enough, late
payments = rare, income pattern = harvest -> advice = consider a harvest-time schedule
  learned consider a harvest-time schedule#1: income pattern = harvest -> advice = consider
a harvest-time schedule (exception of not now: lower the amount or save first#1)
Saved loan-kb.json: Koperasi Sumber Makmur loan advisor: 6 variables, 8 nodes, 8 rules
```

Kasusnya, dalam kata-kata, adalah apa yang dilihat pengurus: rasio tinggi, anggota mapan dengan simpanan cukup yang jarang terlambat, yang penghasilannya datang saat panen. Penasihat tadinya berkata *not now*; perbedaannya, yang disebut pengurus, adalah `income pattern = harvest`, dan node barunya adalah pengecualian node *not now* dengan tepat satu kondisi itu. Tidak ada yang lain berubah. Setelah diajari, saran untuk Pak Darmo menjadi:

```
A03 Pak Darmo (rice farmer): Rp 8,000,000 over 8 months
  facts: repayment ratio 54.0%, membership 72 months, savings cover 18%, late payments 0,
income harvest
  advice: consider a harvest-time schedule  (CF 100)
  because:
    consider a harvest-time schedule#1 fired because:
      repayment ratio = high  (repayment ratio = 54.0, given, CF 100)
      income pattern = harvest  (given, CF 100)
      so advice = consider a harvest-time schedule  [rule CF 100, conclusion CF 100]
  how sure (income verified): about 0.95 (0.85 to 1)
  Rp 3,700,000 would bring the repayment ratio down to low (25.0%)
```

Pengecualian itu menyala: rasionya tinggi (derajat 1), pola penghasilannya panen (kondisi tegas), dan sarannya mempertimbangkan jadwal bayar saat panen. Pengetahuan yang sama kini akan menyiapkan berkas petani berikutnya dengan cara yang sama — dan hanya berkas petani: pemilik warung dengan rasio tinggi tetap mendapat *not now*. Musim depan, ketika Pak Sukri, petani lain, mengajukan pinjaman untuk kebun cabainya dengan rasio 48 persen, lembarnya akan sudah berbunyi *consider a harvest-time schedule*, dan pengurus akan memulai pembahasan dari titik tempat pembahasan untuk Pak Darmo berakhir. Keenam anggota lain mendapat saran yang persis sama seperti sebelumnya, dan `advise` memastikannya baris demi baris.

Mengapa pengecualian, dan bukan aturan tingkat puncak baru "income pattern = harvest → jadwal panen"? Karena pengurus tidak mengatakan bahwa setiap petani harus mendapat jadwal panen; pengurus mengatakan bahwa bila rasio seorang petani *tinggi*, jadwal panen adalah jawabannya, bukan *not now*. Petani yang rasionya rendah meminjam dengan syarat biasa, dan basis pengetahuan tidak boleh menyarankan lain. Pengecualian menyimpan pengetahuan baru dalam konteks tempat pengurus memberikannya — pelajaran Bab 3, kini dengan kata-kata fuzzy.

Terakhir, pemeriksaan keadilan, setelah pengajaran:

```
Variables the advice may depend on: repayment ratio, membership, savings cover, late
payments, income pattern
Protected attributes used: none
Same advice for gender = female and gender = male: 7 of 7 applications
```

Saran kini dapat bergantung pada lima variabel, termasuk pola penghasilan; tidak ada yang dilindungi; dan menambahkan jenis kelamin ke dalam fakta tidak pernah mengubah saran anggota mana pun. Dan ketika seseorang mencoba mengajarkan kasus yang memakai jenis kelamin — berkas `bad-cases.json` memuat kasus seperti itu — penasihat menolak membangun basis pengetahuan sama sekali (`fairness bad-cases.json`):

```
Refused: case 1 uses the protected attribute 'gender'.
```

9.8 Membuatnya Lebih Baik

9.8.1 Menyetel kata-kata bersama pengurus

Himpunan fuzzy adalah kata-kata pengurus yang ditulis sebagai angka, dan pengurus yang seharusnya memilikinya. Cara yang baik untuk menetapkannya adalah mengambil selusin berkas lama, mengurutkannya menurut rasio, lalu meminta setiap pengurus menandai, untuk masing-masing, apakah ia akan menyebutnya rendah, sedang, atau tinggi. Bila hampir semua orang berkata *rendah*, derajatnya sebaiknya mendekati 1; bila pendapat terbelah, mendekati 0,5. Sudut-sudut trapesium mengikuti dari tanda-tanda itu. Ubahlah `fuzzy-sets.csv`, jalankan `advise` pada berkas-berkas lama, dan lihat apakah sarannya cocok dengan keputusan pengurus waktu itu. Soal Latihan 9.6 melakukannya untuk batas antara *rendah* dan *sedang*.

9.8.2 Ambang

Ambang basis pengetahuan, 0,5, menentukan kapan sebuah kondisi fuzzy dianggap benar. Pengurus yang hati-hati mungkin ingin menaikkannya menjadi 0,6 untuk aturan-aturan *recommend* — tetapi ambang itu satu angka untuk seluruh basis pengetahuan, sehingga menaikkannya juga membuat *not now* dan *refer to the committee* lebih sulit dicapai, dan itu sama sekali bukan kehati-hatian. Biasanya lebih baik mempertahankan 0,5 dan menggeser sudut-sudut kata, tempat maknanya berada. Sebuah contoh memperjelas perbedaannya. Dengan ambang 0,6, rasio Mas Andi 25,0 persen — rendah dan sedang masing-masing berderajat 0,5 — tidak akan memenuhi kata *apa pun*, dan tidak ada node yang menyala: berkas paling rumit bulan itu tidak akan mendapat saran maupun peringatan batas tipis. Menggeser sudut-sudut kata *rendah*, seperti dalam Soal Latihan 9.6, hanya mengubah makna satu kata dan tidak mengubah yang lain.

9.8.3 Hedge dalam pengetahuan

Pengurus dapat memakai hedge ketika mengajar. Kasus dengan `repayment ratio = very high` hanya menangkap rasio yang jauh di atas 40 persen dengan derajat yang kuat; kasus dengan `repayment ratio = somewhat low` murah hati di batasnya. Misalnya, pengurus yang ingin merujuk anggota yang angsurannya *sangat* tinggi kepada konselor keuangan dapat mengajarkannya sebagai pengecualian node *not now*. Hedge membuat kata-kata dalam aturan mengikuti kata-kata dalam rapat, tanpa himpunan fuzzy baru.

9.8.4 Mencatat keputusan yang berbeda

Setiap kali keputusan pengurus berbeda dari saran penasihat, `teach` mencatatnya dalam riwayat `loan-kb.json`, ditandai sebagai diajarkan oleh *committee*. Setahun sekali, hitunglah keputusan-keputusan yang berbeda itu menurut node yang disempurnakannya. Node yang berulang-ulang dikoreksi pengurus adalah aturan yang tidak lagi dipercaya pengurus, atau kata yang tepinya sudah bergeser — *rendah* pada tahun harga-harga tinggi mungkin bukan *rendah* pada tahun yang tenang. Hitungan itu, seperti Credit di Bab 7, memberi tahu pengurus ke mana harus melihat lebih dulu ketika meninjau pengetahuannya sendiri pada rapat anggota tahunan.

9.8.5 Pengujian

`LoanTest.java` dan `test_loans.py` memeriksa hitungannya (angsuran Rp 560.000, rasio Mas Joko 28,1 persen, jumlah pengandaiannya, format rupiah), saran fuzzy-nya (*recommend a smaller amount* dengan CF 81, dua lintasan untuk kasus batas tipis), dan keadilannya (berkas kasus yang

buruk ditolak, dan menambahkan jenis kelamin tidak pernah mengubah saran apa pun). Keduanya diakhiri dengan:

```
all tests passed
```

9.9 Memakainya Sungguh-Sungguh

Kata-kata koperasi Anda. Mulailah dari aturan Anda sendiri, bukan dari aturan bab ini. Tuliskan bersama pengurus kata-kata yang dipakainya ketika memutuskan — *penghasilannya cukup*, *anggotanya dikenal*, *ia punya simpanan* — dan jadikan masing-masing variabel dengan himpunan fuzzy atau nilai tegas. Pertahankan variabel-variabel itu tentang kemampuan membayar. Lalu ajarkan sepuluh atau dua puluh keputusan lama sebagai kasus, dan jalankan `advise` pada berkas bulan lalu untuk melihat apakah penasihat akan menyiapkannya seperti yang diputuskan pengurus.

Mulai dengan hati-hati. Selama tiga bulan pertama, jalankan penasihat di samping pengurus tanpa memperlihatkan sarannya selama rapat. Bu Wati mencetaknya sesudah rapat dan membandingkannya dengan keputusan pengurus. Bila sama, tidak ada yang perlu dilakukan; bila berbeda, pengurus mengajarkan kasusnya. Hanya bila penasihat sudah sependapat dengan sebagian besar keputusan — dan pengurus memahami mengapa ia berbeda pendapat untuk sisanya — lembar-lembarnya boleh diletakkan di meja sebelum pembahasan. Dengan cara itulah basis pengetahuan mendapat kepercayaan pengurus, satu rapat demi satu rapat.

Rapat anggota menetapkan aturannya. Bunga, pinjaman maksimum, simpanan yang harus dimiliki anggota sebelum meminjam, dan lama pinjaman termasuk anggaran dasar dan anggaran rumah tangga (AD/ART) koperasi dan wewenang rapat anggota, yang oleh undang-undang ditetapkan sebagai pemegang kekuasaan tertinggi dalam koperasi [3], serta peraturan yang berlaku bagi usaha simpan pinjam [2]. Penasihat membaca bunga dari satu konstanta; perbaruilah bila rapat mengubahnya, dan tambahkan setiap batas keras dalam AD/ART sebagai kondisi tegas dalam basis pengetahuan.

Data tentang orang. Berkas permohonan adalah data pribadi. Simpanlah di komputer koperasi, perlihatkan kepada setiap anggota berkasnya sendiri, dan jangan mengirimnya ke mana pun; undang-undang pelindungan data pribadi Indonesia berlaku bagi koperasi seperti bagi siapa saja [12]. Basis pengetahuannya sendiri menyimpan kasus dalam kata-kata — *repayment ratio = high* — tanpa nama, dan itu satu alasan lagi untuk mempertahankannya begitu.

Setiap keputusan milik pengurus. Cetak lembar penasihat untuk setiap berkas, biarkan pengurus menulis keputusannya di sampingnya, dan ajarkan perbedaannya sebulan sekali, seperti dilakukan Bu Wati. Dalam setahun penasihat akan mempelajari kebiasaan pengurus, dan riwayat dalam `loan-kb.json` akan memperlihatkan aturan mana yang paling sering dikoreksi pengurus — tempat untuk membicarakan apakah sebuah aturan atau sebuah kata perlu berubah.

Bendahara kedua. Bu Wati tidak akan selamanya menjadi bendahara. Penasihat, dengan kasus-kasusnya, kata-katanya, dan riwayatnya, juga merupakan cara terbaik untuk mewariskan pengalaman sebelas tahun: bendahara baru yang menyiapkan berkas dengannya memberi pembacaan pertama yang sama seperti yang akan diberikan Bu Wati, dan dapat membaca dalam basis pengetahuan *mengapa* pengurus memutuskan seperti itu. Mintalah bendahara baru menyiapkan berkas selama tiga bulan di samping Bu Wati, sambil saling mencocokkan catatan, sebelum ia mengambil alih tugas itu sepenuhnya.

Menjelaskan kepada anggota. Ketika seorang anggota bertanya mengapa, bacakan penjelasannya: kata mana yang cocok dengan angka-angkanya, sampai derajat berapa, dan aturan mana yang mengikutinya. Lalu bacakan jumlah pengandaiannya. Anggota yang memahami alasannya dapat merencanakan: menabung sedikit lagi, meminjam sedikit lebih kecil, atau kembali tiga bulan lagi sebagai anggota yang mapan.

Dalam bahasa anggota. Penasihat mencetak penjelasannya dalam kata-kata mesin (*fired because*, *to degree 0.81*). Untuk anggota, tulislah lembar pendek berbahasa Indonesia yang menerangkan arti setiap baris — *derajat 0,81 berarti angka Anda sebagian besar termasuk "sedang"* — atau terjemahkan baris-baris `adviseOne` ke dalam bahasa Indonesia; pengetahuannya sendiri tidak berubah.

Keputusan lain yang bertepi lunak. Penasihat yang sama cocok untuk keputusan apa pun yang dibuat dari angka oleh orang-orang yang berbicara dengan kata: apakah seorang siswa layak mendapat beasiswa desa (*penghasilan keluarga rendah*, *kehadiran baik*), apakah sebuah toko kecil dapat menerima kredit dari pemasok (*omzet tinggi*, *riwayat pendek*), berapa kredit pupuk yang sanggup ditanggung sebuah kelompok tani.

9.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Batas keras yang didandani sebagai himpunan fuzzy. Trapesium dengan $a = b$ dan $c = d$ adalah selang tegas, dan mengembalikan mata pisau. *Hindari:* berilah setiap kata lereng yang sungguhan, selebar keraguan pengurus.

Kata-kata yang meninggalkan celah. Bila *rendah* berakhir di 25 dan *sedang* baru mulai di 30, rasio 27 tidak termasuk kata apa pun, dan tidak ada node yang menyala. *Hindari:* biarkan kata-kata yang bertetangga saling tumpang tindih, sehingga setiap angka termasuk suatu kata berderajat 0,5 atau lebih, seperti pada **Gambar 9.1**.

Membaca CF sebagai peluang membayar kembali. CF 81 tidak berarti peluang 81 persen bahwa Mas Joko membayar kembali. CF menyatakan seberapa cocok angka-angkanya dengan kata-kata aturan. *Hindari:* bacalah sebagai kekuatan rekomendasi, dan bacalah baris ketidakpastian untuk mengetahui seberapa kokoh datanya.

Memakai atribut yang dilindungi, terang-terangan atau melalui proksi. *Hindari:* pertahankan penjaga keadilan; sebelum menambah variabel apa pun, tanyakan apakah ia tentang kemampuan membayar dan apakah ia dapat mewakili kelompok yang dilindungi.

Mengajar dengan angka alih-alih kata. Kasus dengan `repayment ratio = 54.0` hanya menyala untuk tepat 54,0. *Hindari:* ajarkan dalam kata-kata pengurus; `teach` melakukannya dengan `bestTerm`.

Mencampur komponen rasio. Melupakan angsuran yang dibayar di tempat lain — ke bank, ke pindar, ke diler motor — membuat setiap rasio tampak lebih baik daripada kenyataannya. *Hindari:* tanyakan semua angsuran, dan tandai penghasilan sebagai *stated* bila tidak dapat diperiksa.

Lupa memperbarui penghasilan. Penghasilan anggota berubah menurut musim, pekerjaan baru, atau sakit. *Hindari:* ambil penghasilan dan angsuran dari permohonan terbaru, bukan dari berkas tahun lalu, dan tandai *stated* bila tidak ada bukti baru.

Memperlakukan tanda batas tipis sebagai penolakan. *Hindari:* berkas batas tipis adalah berkas yang tidak dapat diputuskan angka; ia memerlukan pengetahuan pengurus tentang anggotanya, bukan "tidak".

Mengubah kata tanpa mengujinya. Menggeser sudut-sudut *rendah* dua poin saja mengubah saran bagi setiap anggota yang berada dekat batas. *Hindari:* jalankan `advise` pada berkas-berkas bulan lalu sebelum dan sesudah perubahan, dan biarkan pengurus melihat setiap berkas yang sarannya bergeser.

Membiarkan penasihat yang memutuskan. *Hindari:* penasihat menyiapkan; pengurus memutuskan, dan mengatakannya kepada anggota.

9.11 Ringkasan Bab

- **Himpunan fuzzy** memberi sebuah angka derajat keanggotaan pada sebuah kata; trapesium Persamaan (9.1) cukup untuk sebagian besar kata pengurus, dan hedge Persamaan (9.2) menajamkan atau melunakkannya (**Gambar 9.1**, **Kode 9.1**, **Kode 9.2**).
- Dalam VCIRS, kondisi pada variabel fuzzy terpenuhi bila derajatnya mencapai ambang, dan derajat itu masuk ke kepastian konklusi menurut Persamaan (9.3); pengetahuan diajarkan dalam kata-kata dan diterapkan pada angka.
- **Kekaburan** (kata-kata yang lunak) dan **ketidakpastian** (data dan aturan yang tidak pasti) berbeda; penasihat membawa ketidakpastian penghasilan sebagai **bilangan fuzzy** segitiga yang digabungkan menurut Persamaan (9.4).
- **Rasio angsuran** Persamaan (9.5), dihitung dalam rupiah bulat (**Kode 9.3**, **Kode 9.4**), adalah angka kunci pengurus; batas 30 persen OJK untuk pinjaman daring pada 2026 adalah patokan yang masuk akal.
- **Penasihat pinjaman** (**Kode 9.5**, **Kode 9.6**) memberi setiap permohonan saran beserta CF-nya, penjelasannya, peringatan batas tipis bila dua kata sama kuatnya, ketidakpastian penghasilan, dan jumlah pinjaman yang tergolong rendah; keputusan pengurus diajarkan kembali sebagai pengecualian (**Gambar 9.2**).
- Keadilan dan keterbukaan dibangun ke dalamnya: hanya fakta tentang kemampuan membayar yang sampai ke mesin, pengetahuan yang memakai atribut yang dilindungi ditolak, uji kontrafaktual memperlihatkan bahwa jenis kelamin tidak pernah mengubah saran, dan setiap jawaban dapat dijelaskan kepada anggota.

9.12 Soal Latihan

9.1 Dengan Persamaan (9.1) dan himpunan-himpunan pengurus, hitunglah dengan tangan derajat *low*, *medium*, dan *high* untuk rasio angsuran 22 persen dan 37 persen. Node mana dalam basis pengetahuan yang dapat menyala untuk masing-masing?

9.2 Dengan Persamaan (9.2), hitunglah *very medium*, *somewhat medium*, dan *not medium* untuk 33,5 persen. Mana yang terpenuhi pada ambang 0,5?

9.3 Bu Lastri mengajukan Rp 3.000.000 untuk 6 bulan. Penghasilannya Rp 3.000.000 sebulan dan ia sudah membayar angsuran Rp 250.000. Hitunglah angsuran dan rasio angsurannya menurut Persamaan (9.5), derajat ketiga kata, serta saran dan CF yang diberikan penasihat bila ia anggota mapan dengan simpanan cukup yang jarang terlambat.

9.4 Saran untuk Mas Joko ber-CF 81 dan penghasilannya *stated*. Hitunglah segitiga "how sure"-nya dengan Persamaan (9.4). Bagaimana hasilnya bila ia membawa rekening koran tiga bulan dan penghasilannya menjadi *verified*?

9.5 Jelaskan, dengan berkas Mas Andi, mengapa penasihat memanggil `infer` dengan konklusi ganda. Apa yang akan dicetaknya untuk Mas Andi bila tidak?

9.6 Pengurus memutuskan bahwa 25 persen masih harus "jelas rendah" dan menggeser *low* menjadi (0, 0, 22, 32) dan *medium* menjadi (22, 32, 32, 42). Hitunglah ulang derajat untuk Mas Andi dan Mas Joko. Apakah Mas Andi masih di batas tipis? Apa saran dan CF Mas Joko sekarang?

9.7 Seorang pengurus mengusulkan variabel `village` karena "anggota dari Dusun Krajan selalu membayar". Jawablah dia, dengan memakai gagasan proksi dan penjaga keadilan penasihat.

9.8 Rancanglah satu pengecualian lagi untuk basis pengetahuan: anggota yang rasio angsurannya *very high* harus dirujuk ke konselor keuangan gratis sebelum pinjaman apa pun dibahas. Tuliskan kasusnya, sebutkan di bawah node mana ia ditempatkan, dan jelaskan interaksinya dengan pengecualian jadwal panen yang diajarkan di Subbab 9.7.

9.13 Penyelesaian Soal Latihan

Penyelesaian 9.1. Pada 22 persen: *low* punya $c = 20 < 22 < d = 30$, jadi $\mu = (30 - 22)/(30 - 20) = 0.8$; *medium* punya $a = 20 < 22 < b = 30$, jadi $\mu = (22 - 20)/10 = 0.2$; *high* bernilai 0. Hanya node yang kondisi pertamanya `repayment ratio = low` yang dapat menyala (0,8 mencapai 0,5; 0,2 tidak). Pada 37 persen: *low* 0; *medium* $(40 - 37)/10 = 0.3$; *high* $(37 - 30)/10 = 0.7$. Hanya node *high* dan pengecualian-pengecualiannya yang dapat menyala.

Penyelesaian 9.2. *Medium* pada 33,5 bernilai 0,65. *Very medium* adalah $0.65^2 = 0.4225$; *somewhat medium* adalah $\sqrt{0.65} \approx 0.806$; *not medium* adalah $1 - 0.65 = 0.35$. Hanya *somewhat medium* yang terpenuhi pada 0,5.

Penyelesaian 9.3. $I = \lceil 3{,}000{,}000/6 \rceil + 30{,}000 = 530{,}000$; $r = (250{,}000 + 530{,}000)/3{,}000{,}000 \times 100 = 26.0$ persen. *Low*: $(30 - 26)/10 = 0.4$; *medium*: $(26 - 20)/10 = 0.6$; *high*: 0. Node *medium* menyala dengan derajat 0,6; pengecualiannya *savings cover = enough* juga menyala (derajat 1), dan pengecualian di bawahnya memerlukan keterlambatan yang sering, yang tidak dimilikinya. Sarannya *recommend*, dengan CF $\min(100 \times 0.6, 100 \times 1) \times 100/100 = 60$.

Penyelesaian 9.4. *Stated*: $0.81 \times (0.5, 0.7, 0.9) = (0.405, 0.567, 0.729)$, tercetak *about 0.57 (0.41 to 0.73)*. *Verified*: $0.81 \times (0.85, 0.95, 1.0) = (0.6885, 0.7695, 0.81)$, *about 0.77 (0.69 to 0.81)*. Sarannya tidak berubah; keyakinan pengurus padanya yang berubah.

Penyelesaian 9.5. Pada tepat 25,0 persen, *low* dan *medium* sama-sama terpenuhi dengan derajat 0,5, sehingga dua node tingkat puncak menyala dan memberi saran berbeda. Tanpa konklusi ganda, mesin berhenti pada lintasan pertama, node *low*, dan mencetak *recommend (CF 50)* dengan satu penjelasan tanpa peringatan. Pengurus akan melihat rekomendasi yang lemah, tetapi tidak melihat bahwa rekomendasi sebaliknya sama kuatnya. Inferensi ganda membuat keseimbangan itu terlihat.

Penyelesaian 9.6. Mas Andi, 25,0 persen: dengan sudut-sudut yang baru $22 < 25 < 32$, sehingga *low* bernilai $(32 - 25)/10 = 0.7$ dan *medium* $(25 - 22)/10 = 0.3$. Hanya *low* yang terpenuhi: ia tidak lagi di batas tipis, dan sarannya *recommend* dengan CF 70. Mas Joko, 28,1 persen: *low* $(32 - 28.1)/10 = 0.39$, *medium* $(28.1 - 22)/10 = 0.61$. Sarannya tetap *recommend a smaller amount*, kini dengan CF 61. Menggeser sudut-sudut mengubah kekuatan kedua jawaban dan menghapus

satu kasus batas tipis — itulah sebabnya perubahan seperti ini menjadi wewenang pengurus, diuji pada berkas-berkas lama.

Penyelesaian 9.7. Desa tidak berkaitan dengan kemampuan membayar; ia menggambarkan tempat tinggal seseorang, dan di banyak daerah tempat tinggal berkaitan dengan suku, agama, atau asal. Memakainya akan membuat penasihat memperlakukan dua anggota dengan penghasilan, simpanan, dan riwayat yang sama secara berbeda karena dusunnya — sebuah **proksi** bagi atribut yang dilindungi. Penjaga keadilan mendaftar atribut yang dilindungi menurut namanya dan tidak dapat mengenali setiap proksi, sehingga aturan pengurus haruslah bahwa setiap variabel tentang kemampuan membayar dan dapat dipertanggungjawabkan kepada anggota. Apa yang membuat anggota Dusun Krajan dapat diandalkan — simpanan, riwayat yang baik — sudah ada dalam basis pengetahuan sebagai fakta mereka masing-masing.

Penyelesaian 9.8. Kasusnya `repayment ratio = very high` → `advice = refer to a financial counselor first`. Ia ditempatkan di bawah node *not now* (opsi 2 dengan kondisi pilihan `repayment ratio = very high`), karena ia menyempurnakan jawaban node itu untuk kasus-kasus paling ekstrem. Node *not now* kini punya dua pengecualian, *harvest* dan *very high*. Untuk petani dengan rasio sangat tinggi keduanya menyala, dan mesin, yang mengikuti pengecualian menurut urutan pengajarannya, memberi saran jadwal panen lebih dulu; dengan konklusi ganda penasihat akan mencetak keduanya dan menandai berkas itu sebagai batas tipis. Bila pengurus ingin konselor didahulukan dalam setiap kasus, pengecualian *very high* itu sendiri perlu diberi pengecualian *harvest*, atau diajarkan sebelum pengecualian panen dalam basis pengetahuan yang dibangun ulang.

Daftar Pustaka

[1] Otoritas Jasa Keuangan, “Siaran pers bersama: Kemenkop RI serahkan daftar koperasi yang menjalankan kegiatan di sektor jasa keuangan kepada OJK,” Jan. 14, 2025. [Online]. Available: https://ojk.go.id/id/berita-dan-kegiatan/siaran-pers/Pages/Kemenkop-RI-Serahkan-Daftar-Koperasi-yang-Menjalankan-Kegiatan-di-Sektor-Jasa-Keuangan-kepada-OJK.aspx

[2] Kementerian Koperasi dan UKM, *Peraturan Menteri Koperasi dan UKM Nomor 8 Tahun 2023 tentang Usaha Simpan Pinjam oleh Koperasi*. Jakarta, Indonesia, 2023. [Online]. Available: https://peraturan.bpk.go.id/Details/260414/permenkop-ukm-no-8-tahun-2023

[3] Republik Indonesia, *Undang-Undang Nomor 25 Tahun 1992 tentang Perkoperasian*. Jakarta, Indonesia, 1992. [Online]. Available: https://peraturan.bpk.go.id/Details/46650/uu-no-25-tahun-1992

[4] ANTARA News, “Menkop: RUU Perkoperasian baru ditargetkan disahkan tahun ini,” Jul. 28, 2026. [Online]. Available: https://m.antaranews.com/amp/berita/5669913/menkop-ruu-perkoperasian-baru-ditargetkan-disahkan-tahun-ini

[5] Liputan6.com, “83 ribu Koperasi Merah Putih terbentuk, nilai transaksi tembus Rp 56,69 miliar,” Jul. 13, 2026. [Online]. Available: https://www.liputan6.com/bisnis/read/8245255/83-ribu-koperasi-merah-putih-terbentuk-nilai-transaksi-tembus-rp-5669-miliar

[6] I. Subakti and O. Wijayanto, “Penerapan konsep fuzzy dalam Variable-Centered Intelligent Rule System (studi kasus: pemilihan jurusan di Chinese University of Hongkong),” Dept. of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia.

[7] L. A. Zadeh, “Fuzzy sets,” *Information and Control*, vol. 8, no. 3, pp. 338–353, 1965.

[8] M. M. I. Subakti, “A Variable-Centered Intelligent Rule System,” M.Sc.Eng. thesis, Dept. of Computer Science and Information Engineering, Taipei, Taiwan, Jun. 2005.

[9] G. R. Baur and D. V. Pigford, *Expert Systems for Business: Concepts and Applications*. Boston, MA, USA: Boyd & Fraser, 1990.

[10] VOI, "Mulai bertahap, OJK terapkan batas rasio utang pinjol 30 persen pada 2026," Jan. 12, 2026. [Online]. Available: https://voi.id/ekonomi/550592/mulai-bertahap-ojk-terapkan-batas-rasio-utang-pinjol-30-persen-pada-2026

[11] Otoritas Jasa Keuangan, *Peraturan OJK Nomor 22 Tahun 2023 tentang Pelindungan Konsumen dan Masyarakat di Sektor Jasa Keuangan*. Jakarta, Indonesia, Dec. 2023. [Online]. Available: https://ojk.go.id/id/regulasi/Pages/Pelindungan-Konsumen-dan-Masyarakat-di-Sektor-Jasa-Keuangan.aspx

[12] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*. Jakarta, Indonesia, Oct. 17, 2022. [Online]. Available: https://peraturan.bpk.go.id/Details/229798/uu-no-27-tahun-2022

Bab 10

Penasihat Penghilang Noda sebagai Aplikasi Web dengan Basis Data

Setelah menyelesaikan bab ini Anda akan mampu menjelaskan mengapa sistem pakar yang dipakai beberapa orang sekaligus sebaiknya berupa aplikasi web dengan basis data; menyimpan basis pengetahuan VCIRS di dalam tabel-tabel basis data SQLite dan membacanya dengan SQL; mengajukan pertanyaan sistem pakar lewat web dengan *pola jalankan-ulang* (*re-run pattern*), di mana peramban mengingat jawaban-jawaban dan server tidak mengingat apa pun di antara dua panggilan; serta membangun **penasihat noda** sebuah laundry kecil dua kali — server Java di atas `HttpServer` bawaan pustaka standar dengan *driver* JDBC SQLite, dan server Python di atas Flask — yang menyajikan halaman yang sama, menjawab setiap panggilan dengan bita yang sama, dan menulis basis data yang sama. Anda akan menjalankannya di peramban, mengajarinya dua pelajaran baru, melihat apa yang dipelajarinya, dan menghitung konsultasinya dengan SQL.

10.1 Laundry Bu Rini

Laundry Bu Rini menempati sebuah ruko di jalan belakang kampus, di antara kios fotokopi dan warung makan. Di meja depan ada timbangan, setumpuk kantong plastik, dan buku pesanan; di belakang ada empat mesin cuci, dua mesin pengering, meja setrika yang panjang, dan, hampir setiap hari, Joko. Mahasiswa dari rumah-rumah kos di sekitarnya membawa pakaian kiloan dan mengambilnya dua hari kemudian, sudah dicuci, disetrika, dan dilipat. Sebagian besar pekerjaannya rutin. Bagian yang tidak rutin adalah noda.

Kemeja putih dengan noda teh di kerah. Celana jins dengan gemuk motor di lutut. Seragam sekolah dengan garis tinta pulpen yang bocor di saku. Syal sutra dengan setetes minyak goreng. Masing-masing punya satu cara penanganan yang benar dan beberapa cara yang salah, dan cara yang salah tidak selalu tanpa akibat. Air panas mengunci noda darah ke dalam katun untuk selamanya. Pemutih klorin pada noda karat justru membuatnya lebih gelap. Alkohol gosok pada sutra dapat meninggalkan bekas lingkaran yang lebih buruk daripada nodanya. Noda yang masuk ke mesin pengering sebelum ditangani biasanya menetap selamanya, dan pelanggan yang kemeja kesayangannya kembali dalam keadaan rusak tidak akan datang lagi.

Bu Rini tahu cara-cara penanganannya. Sebagian ia pelajari dari ibunya, sebagian dari label produk pembersih yang ia beli, dan sebagian besar dari dua belas tahun berurusan dengan noda. Namun ia tidak selalu ada di toko. Setiap Senin ia berbelanja ke pasar; setiap Kamis ia menjaga cabang kedua di seberang kota. Dewi, yang menerima pesanan di meja depan, dan Joko, yang mencuci, harus memutuskan sendiri. Untuk sementara mereka memakai sebuah buku catatan yang digantung di paku dekat mesin cuci. Buku itu basah, halamannya saling menempel, dan tidak ada yang dapat menemukan halaman tentang tinta ketika dibutuhkan.

Keinginan Bu Rini mudah diucapkan. Ia ingin cara-cara penanganannya tersedia di ponsel di meja depan dan di laptop tua di belakang, keduanya sekaligus, keduanya memberikan saran yang sama. Ia ingin Dewi dan Joko dapat menambahkan apa yang mereka pelajari, lengkap dengan nama mereka, sehingga pada Kamis malam ia dapat membaca apa yang dipelajari Kamis pagi. Dan ia

ingin beberapa angka di akhir pekan: berapa noda yang ditangani, seberapa sering sarannya berhasil, penanganan mana yang paling sering dipakai. Bab 3 membangun pemecah masalah yang belajar dari setiap perbaikan, tetapi program itu berjalan di satu terminal pada satu komputer dan menyimpan pengetahuannya dalam sebuah berkas. Bab ini membawa basis pengetahuan sejenis ke sebuah **aplikasi web** dengan **basis data**.

10.2 Gagasannya dengan Kata-Kata Sederhana

10.2.1 Dari satu komputer ke beberapa: aplikasi web

Aplikasi web adalah program yang terdiri atas dua bagian yang berbicara lewat jaringan. **Server** adalah program yang berjalan di satu komputer — di laundry, laptop di belakang — dan menunggu. **Klien** adalah peramban web, di laptop yang sama atau di ponsel mana pun yang tersambung ke Wi-Fi yang sama, yang menampilkan halaman dan mengirim permintaan. Ketika Dewi membuka penasihat di ponselnya, peramban meminta sebuah halaman kepada server; halaman itu berisi program kecil dalam JavaScript, dan program itu mengajukan pertanyaan kepada server seperti "apa yang harus kutanyakan berikutnya, bila jawabannya sejauh ini begini?" Server menjawab, halaman menampilkan jawabannya, dan percakapan berlanjut.

Permintaan dan jawaban berjalan dalam **HTTP**, bahasa web, dan isinya ditulis dalam **JSON**, format teks polos yang sama yang dipakai basis pengetahuan kita sejak Bab 3. Sebuah permintaan kepada server kita tampak seperti ini:

```
POST /api/consult
{"facts": {"stain": "tea"}, "unknown": []}
```

dan jawabannya seperti ini:

```
{"status": "question", "variable": "color",
 "question": "Is the item white or colored?", "choices": ["white", "colored"], ...}
```

Keuntungannya bagi laundry adalah pengetahuan tinggal di satu tempat. Tidak ada dua salinan yang lama-lama berbeda, satu di ponsel dan satu di laptop; hanya ada satu basis pengetahuan di server, dan setiap peramban melihatnya. Ketika Joko mengajari penasihat sesuatu di ruang belakang, konsultasi Dewi berikutnya di meja depan sudah memakainya.

Aplikasi web juga dapat berarti server di internet yang dapat dijangkau seluruh dunia. Bukan itu yang dibutuhkan laundry, dan hal itu membawa masalah yang tidak diinginkan laundry: kata sandi, serangan, tagihan bulanan. Server kita berjalan di komputer toko sendiri dan, kecuali diperintahkan lain, hanya menerima sambungan dari komputer itu. Dengan satu opsi, ia menerimanya dari Wi-Fi toko, dan itu cukup untuk ponsel di meja depan.

10.2.2 Mengapa basis data, dan mengapa SQLite

Bab 3 menyimpan basis pengetahuannya dalam berkas JSON, dimuat di awal dan disimpan setelah setiap pelajaran. Untuk satu orang di satu papan ketik itu cukup. Untuk sebuah toko ada tiga kelemahan.

Pertama, **dua orang sekaligus**. Bila Dewi dan Joko mengajari penasihat pada saat yang sama, dua program dapat menulis berkas yang sama pada waktu yang sama, dan satu pelajaran dapat hilang atau berkasnya rusak. Basis data dibuat persis untuk ini: setiap perubahan adalah sebuah **transaksi** yang terjadi seluruhnya atau tidak sama sekali, dan dua perubahan tidak pernah tercampur [1].

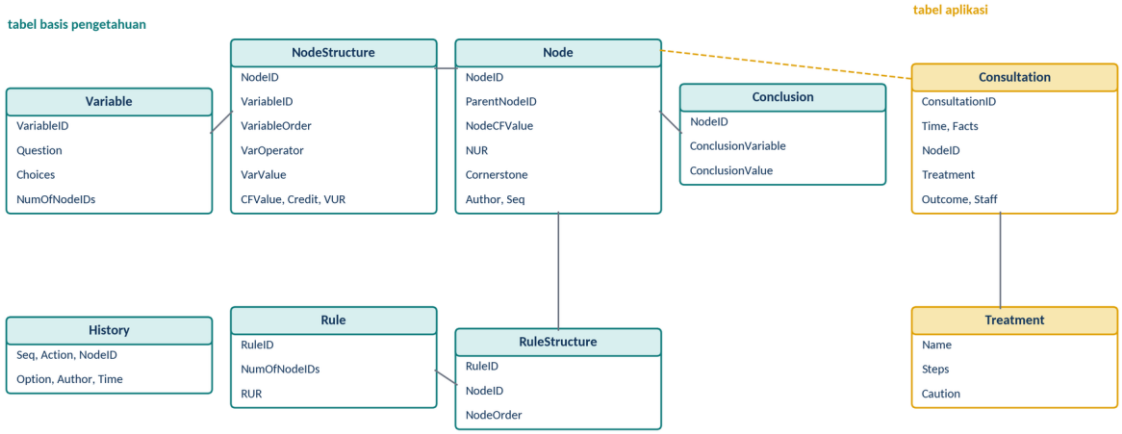
Kedua, **pertanyaan tentang catatan**. Angka-angka mingguan Bu Rini — berapa konsultasi, berapa yang berhasil, penanganan mana yang paling sering diberikan — adalah pertanyaan tentang banyak catatan sekaligus. Basis data menjawab pertanyaan seperti itu dengan beberapa kata **SQL** (*Structured Query Language*): `SELECT Treatment, COUNT(*) FROM Consultation GROUP BY Treatment`. Dengan berkas JSON kita harus menulis perulangan untuk setiap pertanyaan.

Ketiga, **catatan yang bukan pengetahuan**. Laundry juga ingin menyimpan langkah-langkah setiap penanganan dan log konsultasi. Keduanya layak berada di samping basis pengetahuan tetapi bukan bagian darinya, dan basis data menyimpannya sebagai tabel-tabel tersendiri.

SQLite adalah basis data yang kita pakai. Berbeda dari basis data yang berjalan sebagai server terpisah, SQLite adalah pustaka kecil yang membaca dan menulis satu berkas biasa — di sini `laundry.db` — sehingga tidak ada yang perlu dipasang dan tidak ada yang perlu diurus; Anda dapat menyalin berkasnya ke *flash disk* sebagai cadangan. SQLite juga mesin basis data yang paling luas terpasang di dunia: ia ada di dalam setiap ponsel Android dan iOS dan setiap peramban web utama, dan pengembangnya memperkirakan lebih dari satu triliun basis data SQLite sedang dipakai [2]. Lini rilis terkininya, per September 2026, adalah 3.53 (3.53.4 dirilis Juli 2026) [1]. Python memiliki SQLite bawaan, yaitu modul `sqlite3`; Java menjangkaunya lewat **JDBC**, cara baku Java untuk berbicara dengan basis data, dengan *driver* JDBC SQLite, sebuah berkas `.jar` tunggal (versi 3.53.2.0, dirilis Juni 2026, yang membawa SQLite 3.53.2) [3].

10.2.3 Basis pengetahuan sebagai tabel

Ketika tesis yang mendefinisikan VCIRS mengimplementasikan metodenya, basis pengetahuannya disimpan dalam basis data, waktu itu Microsoft Access, dengan satu tabel utama dan beberapa tabel pendukung [4]. Tabel utamanya adalah **NodeStructure**: satu baris untuk setiap kondisi dari setiap node, berisi variabel, posisinya di dalam node, operator, nilai, faktor kepastian, dan Credit. Nama variabel, node, dan rule disimpan di tabel **Variable**, **Node**, dan **Rule**; konklusi di **Conclusion**; dan urutan node dalam setiap rule di **RuleStructure** [4], [5]. `SqliteStore` milik vcirs-core menulis tabel-tabel yang persis sama, menambahkan tabel **History** untuk jejak audit dan tabel kecil **Meta** untuk nama dan format, serta mengisi tingkat pemakaian (VUR, NUR, RUR) agar tabel-tabelnya dapat dijelajahi. **Gambar 10.1** memperlihatkannya, bersama dua tabel milik laundry sendiri.



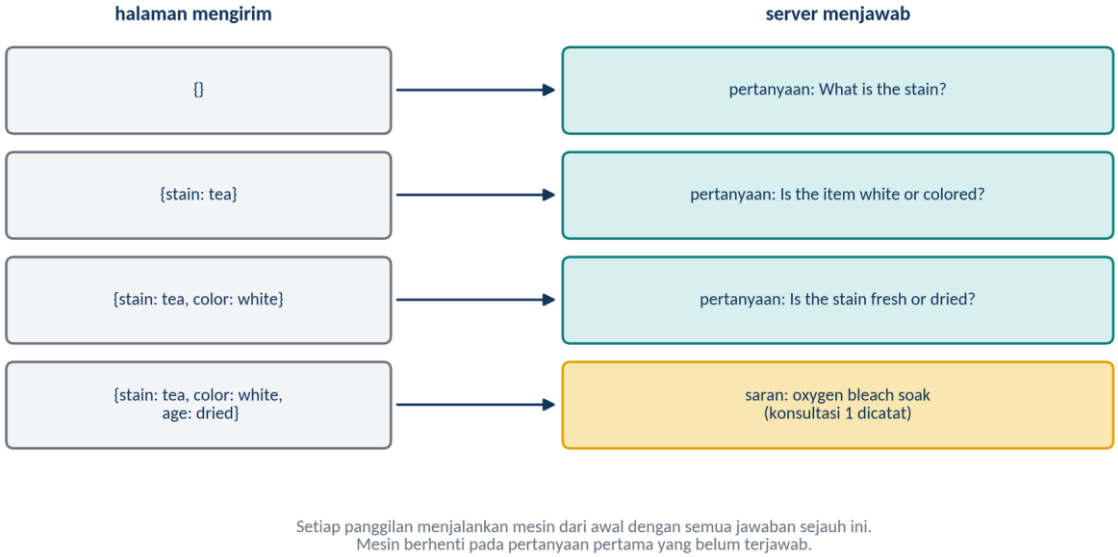
Gambar 10.1 Tabel-tabel basis data penasihat noda. Tabel berwarna hijau toska menyimpan basis pengetahuan VCIRS seperti pada Lampiran E tesis; tabel berwarna kuning milik laundry: langkah-langkah setiap penanganan dan log konsultasi.

Mengapa tabel dan bukan teks JSON dalam satu kolom? Karena tabel menjawab pertanyaan. "Node mana saja yang memakai variabel `fabric`?" adalah `SELECT NodeID FROM NodeStructure WHERE VariableID = 'fabric'`. "Rule mana yang terdiri atas lebih dari satu node?" adalah *join* antara RuleStructure dan NodeStructure. Rancangan VCIRS yang berpusat pada variabel — sebuah node adalah urutan kondisi *variabel*, dan setiap pemakaian variabel dicatat bersamanya — cocok secara alami dengan basis data relasional, dan itulah sebabnya tesis memilihnya [4].

10.2.4 Bertanya lewat web: pola jalankan-ulang

Di Bab 3 mesin mengajukan pertanyaannya lewat sebuah *asker*, fungsi yang mencetak pertanyaan dan menunggu jawaban yang diketik di terminal. Mesin berhenti di tengah pekerjaannya, menunggu, lalu melanjutkan. Server web tidak dapat bekerja seperti itu. Ketika peramban mengirim permintaan, server harus langsung menjawab, dan ia tidak dapat menahan konsultasi setengah jadi di memori sambil menunggu jawaban yang mungkin tidak pernah datang (Dewi mungkin meletakkan ponselnya untuk melayani pelanggan). Kaidah umum rancangan web adalah bahwa server tidak mengingat percakapan di antara satu permintaan dan permintaan berikutnya: setiap permintaan membawa semua yang dibutuhkan server [6].

Maka percakapannya kita balik. **Peramban** menyimpan jawaban-jawaban yang sudah diberikan. Setiap kali, ia mengirim *semuanya* ke server. Server menjalankan mesin dari awal dengan jawaban-jawaban itu dan sebuah asker jenis khusus: asker yang, alih-alih menunggu jawaban, langsung menghentikan mesin dan melaporkan pertanyaannya. Pertanyaan itulah jawaban server. Halaman menampilkannya, menambahkan jawabannya ke daftar, lalu mengirim seluruh daftar lagi. Ketika mesin sampai ke akhir tanpa membutuhkan pertanyaan, jawabannya adalah saran. **Gambar 10.2** memperlihatkan satu konsultasi utuh.



Gambar 10.2 Pola jalankan-ulang. Halaman mengirim setiap jawaban yang dimilikinya; server menjalankan mesin dari awal dan mengembalikan pertanyaan pertama yang belum dapat dijawab mesin, atau, bila tidak ada, sarannya.

Dua hal membuat pola ini berhasil. Mesinnya **deterministik**: dengan jawaban yang sama, ia menelusuri node yang sama dalam urutan yang sama dan sampai ke pertanyaan terbuka pertama yang sama, sehingga menjalankannya ulang tidak pernah mengejutkan. Dan mesinnya **murah**: basis pengetahuan berisi beberapa ratus node dievaluasi dalam sepersekian milidetik, sehingga menjalankannya lagi untuk setiap jawaban tidak memakan waktu yang dapat dirasakan pelanggan. "Saya tidak tahu" juga cocok dengan pola ini: halaman menyimpan daftar kedua,

unknown, dan asker di server menjawab “tidak diketahui” untuk variabel-variabel itu alih-alih berhenti, persis seperti menekan Enter di Bab 3.

> **DEFINISI.** Dalam *pola jalankan-ulang*, klien menyimpan setiap jawaban dalam sebuah konsultasi dan mengirim semuanya bersama setiap permintaan; server menjalankan inferensi dari awal, berhenti pada pertanyaan pertama yang belum dapat dijawabnya dan mengembalikannya, atau mengembalikan konklusi bila tidak ada pertanyaan tersisa. Server tidak menyimpan apa pun di antara permintaan-permintaan.

10.2.5 Apa yang diketahui penasihat, dan apa yang tidak boleh dilakukannya

Dua belas pelajaran pertama Bu Rini, yaitu **kasus benih** (*seed cases*) bab ini, diringkas dalam **Tabel 10.1**. Pelajaran-pelajaran itu mengikuti praktik umum penghilangan noda sebagaimana diajarkan industri produk pembersih dan layanan penyuluhan universitas [7], [8]: noda protein seperti darah dengan air *dingin*, jangan pernah air panas; noda berminyak dengan deterjen yang melarutkan lemak; tinta dengan pelarut dari balik kain; karat jangan pernah dengan pemutih klorin. Dua pelajaran merupakan pengecualian bagi pelajaran lain, dalam arti Ripple Down Rules di Bab 3: noda darah yang *sudah kering* membutuhkan rendaman deterjen enzim alih-alih bilasan dingin, dan tinta pada *sutra* dibawa ke *dry cleaner* alih-alih bertemu alkohol gosok. Penanganan kunyit (warna kuning kari dan nasi kuning) dengan sabun cuci piring dan sinar matahari adalah pengalaman Bu Rini sendiri; basis pengetahuan adalah tempat pengalaman seperti itu dituliskan.

Tabel 10.1 Dua belas kasus benih penasihat noda, beserta peringatan yang menyertai setiap penanganan.

Noda	Syarat	Penanganan	Peringatan yang ditampilkan halaman
minyak	—	dish soap and a warm wash	periksa sebelum dikeringkan
darah	—	cold rinse and soak	jangan pernah air panas
darah	kering	enzyme soak	bukan untuk sutra atau wol
teh	—	cool soak and liquid detergent	—
teh	putih, kering	oxygen bleach soak	hanya katun putih; jangan mencampur pemutih
tinta	—	rubbing alcohol from the back	uji di jahitan tersembunyi; jauhkan dari api
tinta	sutra	send to the dry cleaner	—
permen karet	—	ice and scrape	—
lilin	—	scrape and iron	setrika hangat untuk bahan sintetis
lumpur	—	brush when dry	—
kunyit	—	dish soap and sunlight	—
karat	—	rust remover	jangan pernah pemutih klorin

Noda adalah ranah yang lembut, tetapi penasihat tetap berurusan dengan bahan kimia, panas, dan barang milik orang lain, sehingga ia mengikuti tiga aturan keselamatan yang patut dinyatakan sekali. Ia tidak pernah menyarankan mencampur produk pembersih: pemutih klorin yang dicampur amonia atau asam mengeluarkan gas beracun, dan peringatan setiap penanganan pemutih menyebutkannya. Setiap penanganan dengan pelarut membawa pesan “uji di jahitan tersembunyi dahulu”, karena kain yang tidak tahan pelarut akan memperlihatkannya dalam satu menit di bagian dalam kelim. Dan setiap kali barangnya halus atau labelnya berbunyi *dry clean only*, saran yang benar adalah *dry cleaner*, bukan trik yang cerdik. Peringatan-peringatan ini tinggal di tabel Treatment, di samping langkah-langkahnya, dan halaman menampilkannya dalam kotak kuning yang sulit terlewat.

10.3 Langkah 1: Basis Pengetahuan di dalam Basis Data

Sebelum menulis kode web apa pun, mari kita masukkan basis pengetahuan ke dalam basis data dan melihat isinya. **Kode 10.1** membangun basis pengetahuan dari kasus benih, menyimpannya dengan `SqliteStore` milik vcirs-core, mengajukan empat pertanyaan kepada basis data dalam SQL, dan akhirnya memuat kembali basis pengetahuan dari basis data untuk memeriksa bahwa tidak ada yang hilang.

Kode 10.1 `db_tour.py`: basis pengetahuan disimpan di SQLite, dilihat dengan SQL, lalu dimuat kembali.

```
"""A VCIRS knowledge base in a SQLite database: save it, look inside with SQL,
and load it back.

    python db_tour.py
"""

import os
import pathlib
import sqlite3
import sys

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import KnowledgeBase, SqliteStore, casefile, report

DATA = pathlib.Path(__file__).resolve().parents[1] / "data"
DB = str(DATA / "tour.db")


def show(con, title, sql):
    """Run one SELECT and print its rows as a table."""
    cur = con.execute(sql)
    headers = [d[0] for d in cur.description]
    rows = [["(top)" if v is None else v for v in row] for row in cur]
    print(title)
    for line in report.table(headers, rows):
        print("  " + line)
    print()


if os.path.exists(DB):
    os.remove(DB)                       # start from an empty database
name, description, steps = casefile.read_cases(str(DATA
                                                    / "stain-seed-cases.json"))
kb = casefile.build(KnowledgeBase(name, description), steps)
SqliteStore(DB).save(kb)
print("Saved " + kb.summary() + " in tour.db")
print()
```

Kode 10.1 (Lanjutan) *db_tour.py: basis pengetahuan disimpan di SQLite, dilihat dengan SQL, lalu dimuat kembali.*

```
con = sqlite3.connect(DB)
show(con, "Variables, and how many nodes use each:",
     "SELECT VariableID, NumOfNodeIDs FROM Variable ORDER BY Position")
show(con, "Nodes, with their parent (an exception has one):",
     "SELECT NodeID, ParentNodeID, Author FROM Node ORDER BY Seq")
show(con, "The conditions of one node, one row each:",
     "SELECT VariableOrder, VariableID, VarOperator, VarValue, Credit "
     "FROM NodeStructure WHERE NodeID = 'oxygen bleach soak#1' "
     "ORDER BY VariableOrder")
show(con, "Rules with more than one node, and all the conditions they need:",
     "SELECT r.RuleID, COUNT(*) AS Conditions FROM RuleStructure r "
     "JOIN NodeStructure s ON s.NodeID = r.NodeID "
     "GROUP BY r.RuleID HAVING COUNT(DISTINCT r.NodeID) > 1 "
     "ORDER BY r.RuleID")
con.close()

again = SqliteStore(DB).load()
print("Loaded back: " + again.summary())
print("Same knowledge as before saving: "
      + str(again.to_json() == kb.to_json()).lower())
```

Program ini terdiri atas tiga bagian. Bagian pertama membangun basis pengetahuan persis seperti Bab 3 membangun dokter Wi-Fi: `casefile.read_cases` membaca berkas benih dan `casefile.build` mengajarkan kedua belas kasus satu per satu dengan `learn`, sehingga kedua pengecualian ditempatkan di bawah node yang dikoreksinya. Kemudian `SqliteStore(DB).save(kb)` menuliskannya ke `tour.db`, sekaligus membuat tabel-tabelnya bila belum ada.

Bagian kedua membuka basis data dengan modul `sqlite3` milik Python sendiri dan menjalankan empat pernyataan `SELECT` melalui fungsi kecil `show`, yang mencetak baris-baris hasilnya sebagai tabel dengan `report.table` dari vcirs-core. Sebuah `SELECT` menyebut kolom yang diinginkan (`SELECT VariableID, NumOfNodeIDs`), tabel asalnya (`FROM Variable`), baris mana yang dipertahankan (`WHERE NodeID = 'oxygen bleach soak#1'`), dan urutannya (`ORDER BY Position`). Pernyataan keempat adalah sebuah **join**: `FROM RuleStructure r JOIN NodeStructure s ON s.NodeID = r.NodeID` memasangkan setiap node dari setiap rule dengan setiap kondisi node itu, `GROUP BY r.RuleID` membuat satu kelompok per rule, `COUNT(*)` menghitung kondisi dalam kelompok, dan `HAVING COUNT(DISTINCT r.NodeID) > 1` hanya mempertahankan rule yang terdiri atas lebih dari satu node.

Bagian ketiga memuat kembali basis pengetahuan dengan `SqliteStore(DB).load()` dan membandingkan teks JSON kedua basis pengetahuan. Teks yang sama berarti pengetahuan yang sama: setiap variabel, node, kondisi, kasus cornerstone, dan entri riwayat selamat melewati perjalanan ke dalam tabel dan kembali. **Kode 10.2** adalah program yang sama dalam Java.

Kode 10.2 `DbTour.java`: penjelajahan basis data yang sama dalam Java, melalui JDBC.

```
import java.nio.file.Files;
import java.nio.file.Path;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

import vcirs.CaseFile;
import vcirs.KnowledgeBase;
import vcirs.Report;
import vcirs.SqliteStore;

/**
 * A VCIRS knowledge base in a SQLite database: save it, look inside with
 * SQL, and load it back.
 * java -cp "out:lib/sqlite-jdbc-3.53.2.0.jar" DbTour   (Windows: ; not :)
 */
public class DbTour {
    static final Path DATA = Path.of("..", "data");
    static final String DB = DATA.resolve("tour.db").toString();

    /** Run one SELECT and print its rows as a table. */
    static void show(Connection con, String title, String sql)
            throws SQLException {
        try (Statement st = con.createStatement();
             ResultSet rs = st.executeQuery(sql)) {
            ResultSetMetaData md = rs.getMetaData();
            List<String> headers = new ArrayList<>();
            for (int i = 1; i <= md.getColumnCount(); i++) {
                headers.add(md.getColumnLabel(i));
            }
            List<List<Object>> rows = new ArrayList<>();
            while (rs.next()) {
                List<Object> row = new ArrayList<>();
                for (int i = 1; i <= md.getColumnCount(); i++) {
                    String v = rs.getString(i);
                    row.add(v == null ? "(top)" : v);
                }
                rows.add(row);
            }
            System.out.println(title);
            for (String line : Report.table(headers, rows)) {
```

Kode 10.2 (Lanjutan) *DbTour.java: penjelajahan basis data yang sama dalam Java, melalui JDBC.*

```
                System.out.println("  " + line);
            }
            System.out.println();
        }
    }

    public static void main(String[] args) throws Exception {
        Files.deleteIfExists(Path.of(DB));   // start from an empty database
        CaseFile seed = CaseFile.read(DATA.resolve("stain-seed-cases.json")
                .toString());
        KnowledgeBase kb = seed.build(new KnowledgeBase(seed.name,
                seed.description));
        new SqliteStore(DB).save(kb);
        System.out.println("Saved " + kb.summary() + " in tour.db");
        System.out.println();

        try (Connection con = DriverManager.getConnection("jdbc:sqlite:" + DB)) {
            show(con, "Variables, and how many nodes use each:",
                    "SELECT VariableID, NumOfNodeIDs FROM Variable "
                    + "ORDER BY Position");
            show(con, "Nodes, with their parent (an exception has one):",
                    "SELECT NodeID, ParentNodeID, Author FROM Node ORDER BY Seq");
            show(con, "The conditions of one node, one row each:",
                    "SELECT VariableOrder, VariableID, VarOperator, VarValue, "
                    + "Credit FROM NodeStructure "
                    + "WHERE NodeID = 'oxygen bleach soak#1' "
                    + "ORDER BY VariableOrder");
            show(con, "Rules with more than one node, and all the conditions "
                    + "they need:",
                    "SELECT r.RuleID, COUNT(*) AS Conditions FROM RuleStructure r "
                    + "JOIN NodeStructure s ON s.NodeID = r.NodeID "
                    + "GROUP BY r.RuleID HAVING COUNT(DISTINCT r.NodeID) > 1 "
                    + "ORDER BY r.RuleID");
        }

        KnowledgeBase again = new SqliteStore(DB).load();
        System.out.println("Loaded back: " + again.summary());
        System.out.println("Same knowledge as before saving: "
                + again.toJson().equals(kb.toJson()));
    }
}
```

Di Java, basis data dijangkau melalui **JDBC**. `DriverManager.getConnection("jdbc:sqlite:" + DB)` membuka sebuah `Connection` ke berkas; teks `jdbc:sqlite:` memberi tahu JDBC agar memakai *driver* SQLite, yang harus ada di *class path* ketika program dijalankan. `createStatement()` memberikan sebuah `Statement`, `executeQuery(sql)` menjalankan `SELECT` dan mengembalikan `ResultSet`, dan `rs.next()` berpindah dari baris ke baris. `ResultSetMetaData` memberi tahu berapa kolom yang ada dan apa namanya. Bentuk `try (...)` menutup koneksi, *statement*, dan *result set* secara otomatis, bahkan bila sesuatu gagal; hal ini penting pada basis data, karena koneksi yang dibiarkan terbuka dapat terus mengunci berkasnya.

Untuk menjalankannya, kompilasi bersama mesinnya dan jalankan dengan *driver* di *class path*:

```
cd Chapter10/java
javac -d out -sourcepath ../../vcirs-core/java/src *.java
java -cp "out:lib/sqlite-jdbc-3.53.2.0.jar" DbTour
```

Di Windows pemisah *class path* adalah titik koma, `"out;lib/sqlite-jdbc-3.53.2.0.jar"`. Di Python, `python db_tour.py` (atau `python3` di macOS dan Linux). Keduanya mencetak, bita demi bita:

```
Saved Stain advisor: 5 variables, 12 nodes, 12 rules in tour.db

Variables, and how many nodes use each:
  VariableID  NumOfNodeIDs
  ----------  ------------
  stain       9
  treatment   0
  age         2
  color       1
  fabric      1

Nodes, with their parent (an exception has one):
  NodeID                           ParentNodeID                     Author
  -------------------------------  -------------------------------  ------
  dish soap and a warm wash#1      (top)                            Rini
  cold rinse and soak#1            (top)                            Rini
  enzyme soak#1                    cold rinse and soak#1            Rini
  cool soak and liquid detergent#1 (top)                            Rini
  oxygen bleach soak#1             cool soak and liquid detergent#1 Rini
  rubbing alcohol from the back#1  (top)                            Rini
  send to the dry cleaner#1        rubbing alcohol from the back#1  Rini
  ice and scrape#1                 (top)                            Rini
  scrape and iron#1                (top)                            Rini
  brush when dry#1                 (top)                            Rini
  dish soap and sunlight#1         (top)                            Rini
  rust remover#1                   (top)                            Rini

The conditions of one node, one row each:
  VariableOrder  VariableID  VarOperator  VarValue  Credit
  -------------  ----------  -----------  --------  ------
  1              color       =            white     1
  2              age         =            dried     1

Rules with more than one node, and all the conditions they need:
  RuleID                    Conditions
  ------------------------  ----------
  enzyme soak#1             2
  oxygen bleach soak#1      3
  send to the dry cleaner#1 2

Loaded back: Stain advisor: 5 variables, 12 nodes, 12 rules
Same knowledge as before saving: true
```

Bacalah keluaran ini sebagai kunjungan ke tabel-tabel tesis. Tabel **Variable** mendaftar lima variabel; `stain` dipakai oleh sembilan node, `age` oleh dua, `color` dan `fabric` masing-masing oleh satu, dan `treatment` oleh tidak satu pun — ia variabel konklusi, tidak pernah menjadi kondisi. Tabel **Node** memperlihatkan kedua belas node beserta induknya: tiga node memilikinya, yaitu pengecualian `enzyme soak#1`, `oxygen bleach soak#1`, dan `send to the dry cleaner#1`. Baris-baris **NodeStructure** milik `oxygen bleach soak#1` adalah kedua kondisinya, berurutan, masing-masing dengan Credit-nya; perhatikan bahwa node itu sendiri hanya memuat `color = white` dan `age = dried`, karena `stain = tea` milik induknya. Itulah sebabnya kueri terakhir menemukan tiga kondisi pada rule `oxygen bleach soak#1`: satu dari node induk dan dua dari pengecualiannya. Dan basis pengetahuan yang dibaca kembali dari tabel adalah pengetahuan yang sama, sampai bita terakhir JSON-nya.

> **WAWASAN KUNCI:** Basis data tidak menambahkan apa pun pada makna basis pengetahuan; ia hanya mengubah tempat pengetahuan itu tinggal dan siapa yang dapat bertanya kepadanya. Basis pengetahuan yang sama dapat disimpan sebagai JSON di Bab 3 dan sebagai tabel di sini, dan mesin memberikan jawaban yang sama dari keduanya.

10.4 Langkah 2: Advisor, Tanpa Kode Web Apa Pun

Jantung aplikasi ini adalah sebuah kelas yang tidak tahu apa-apa tentang web: **Advisor**. Ia membuka basis data, berkonsultasi dengan basis pengetahuan, belajar dari umpan balik, dan meringkas apa yang diketahuinya. Metode-metodenya menerima *map* (kamus) biasa dan mengembalikan *map* biasa, yang nanti diubah server menjadi JSON. Menjauhkan web darinya memberi dua keuntungan. Kita dapat menguji Advisor, dan memutar ulang satu hari penuh di laundry, tanpa menyalakan server. Dan kedua server bab ini, Java dan Python, berbagi satu rancangan: hanya lapisan web tipisnya yang berbeda.

Tabel 10.2 mendaftar apa yang ditawarkan Advisor, dan panggilan JSON yang melaluinya aplikasi web akan menawarkannya.

Tabel 10.2 Tiga panggilan penasihat noda dan apa yang dibawa masing-masing.

Panggilan	Yang dikirim halaman	Jawaban Advisor
`POST /api/consult`	`facts` (jawaban sejauh ini), `unknown` (variabel yang dijawab "saya tidak tahu"), `staff`	sebuah `question` (variabel, teks, pilihan, alasan), atau `advice` (penanganan, langkah, peringatan, penjelasan, nomor konsultasi), atau `none`
`POST /api/feedback`	`consultation`, `worked` (true atau false); bila false: `treatment`, `steps`, `caution`, fakta `extra`, `staff`	`accepted`, `learned` (dengan daftar perbedaan dan cornerstone yang terdampak), `closed`, atau `error`
`GET /api/kb`	tidak ada	ringkasan, pohon, jumlah konsultasi, penanganan yang paling sering diberikan, perubahan terakhir, daftar penanganan

Kode 10.3 adalah Advisor dalam Python.

Kode 10.3 `advisor.py`: logika penasihat noda, dengan pengetahuannya di SQLite.

```
"""The stain advisor: a VCIRS knowledge base kept in a SQLite database.

Advisor has no web code at all. Its three methods take and return plain
dictionaries, which the web server turns into JSON:
  consult(facts, unknown)  the next question, or the advice
  feedback(answer)         did the advice work? if not, learn the right one
  overview()               the knowledge base, the treatments, the numbers
"""

import datetime
import json
import os
import pathlib
import sqlite3
import sys

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import (Case, Conclusion, Condition, Fact, KnowledgeBase,
                   SqliteStore, VcirsError, casefile, report)

QUESTIONS = [
    ("stain", "What is the stain?",
     ["oil", "blood", "tea", "ink", "gum", "wax", "mud", "turmeric", "rust"]),
    ("age", "Is the stain fresh or dried?", ["fresh", "dried"]),
    ("color", "Is the item white or colored?", ["white", "colored"]),
    ("fabric", "What is the fabric?",
     ["cotton", "polyester", "denim", "silk", "wool"]),
]
TABLES = [
    "CREATE TABLE IF NOT EXISTS Treatment (Name TEXT PRIMARY KEY, "
    "Steps TEXT, Caution TEXT)",
    "CREATE TABLE IF NOT EXISTS Consultation (ConsultationID INTEGER "
    "PRIMARY KEY, Time TEXT, Facts TEXT, NodeID TEXT, Treatment TEXT, "
    "Outcome TEXT, Staff TEXT)",
]


class Clock:
    """The time now; or, given a start, a pretend clock that moves one
    minute each time it is read (so that a demonstration repeats exactly)."""
```

Kode 10.3 (Lanjutan) *advisor.py: logika penasihat noda, dengan pengetahuannya di SQLite.*

```
    def __init__(self, start=None):
        self.start = start and datetime.datetime.strptime(start,
                                                          "%Y-%m-%d %H:%M")
        self.ticks = 0

    def __call__(self):
        if self.start is None:
            now = datetime.datetime.now()
        else:
            now = self.start + datetime.timedelta(minutes=self.ticks)
            self.ticks += 1
        return now.strftime("%Y-%m-%d %H:%M")


class NeedAnswer(Exception):
    """Thrown by the asker to stop the engine at the first open question."""

    def __init__(self, question):
        super().__init__(question.variable)
        self.question = question


class Advisor:
    def __init__(self, db, data_dir, clock):
        self.db = db
        self.store = SqliteStore(db)
        self.clock = clock
        if not os.path.exists(db):
            self._create(data_dir)
        self.kb = self.store.load()
        self.kb.clock = clock

    def _create(self, data_dir):
        """The first run: build the knowledge from the seed cases."""
        name, description, steps = casefile.read_cases(
            os.path.join(data_dir, "stain-seed-cases.json"))
        kb = KnowledgeBase(name, description)
        for var, question, choices in QUESTIONS:
            kb.declare(var, question, choices)
        casefile.build(kb, steps)
        self.store.save(kb)
        with open(os.path.join(data_dir, "treatments.json"),
                  encoding="utf-8") as f:
            treatments = json.load(f)["treatments"]
        with self._connect() as con:
            for t in treatments:
```

Kode 10.3 (Lanjutan) *advisor.py: logika penasihat noda, dengan pengetahuannya di SQLite.*

```
                con.execute("INSERT INTO Treatment VALUES (?, ?, ?)",
                            (t["name"], t["steps"], t["caution"]))

    def _connect(self):
        con = sqlite3.connect(self.db)
        for sql in TABLES:
            con.execute(sql)
        return con

    def _record(self, facts, node_id, treatment, staff):
        with self._connect() as con:
            cur = con.execute(
                "INSERT INTO Consultation (Time, Facts, NodeID, Treatment, "
                "Outcome, Staff) VALUES (?, ?, ?, ?, 'open', ?)",
                (self.clock(), json.dumps(facts), node_id, treatment, staff))
            return cur.lastrowid

    # ----- consulting ----------------------------------------------------
    def consult(self, facts, unknown, staff=""):
        """Run the knowledge base on the answers so far. If it needs one
        more answer, return that question; otherwise record the
        consultation and return the advice (or say there is none)."""
        given = {v: Fact(v, str(x), 100, "user") for v, x in facts.items()}

        def asker(question):
            if question.variable in unknown:
                return None                    # "I do not know"
            raise NeedAnswer(question)

        try:
            result = self.kb.infer(given, asker)
        except NeedAnswer as e:
            q = e.question
            var = self.kb.variables.get(q.variable)
            return {"status": "question", "variable": q.variable,
                    "question": var.question if var and var.question
                    else q.variable + "?",
                    "choices": q.choices, "why": q.why()}
        if not result.conclusions():
            number = self._record(facts, "", "", staff)
            return {"status": "none", "consultation": number,
                    "message": "No advice for this stain yet."}
        treatment = result.conclusions()[0].value
        node = result.paths[0].last().id
        number = self._record(facts, node, treatment, staff)
        steps, caution = self._treatment(treatment)
```

Kode 10.3 (Lanjutan) *advisor.py: logika penasihat noda, dengan pengetahuannya di SQLite.*

```
        return {"status": "advice", "consultation": number,
                "treatment": treatment, "steps": steps, "caution": caution,
                "node": node, "how": result.how()}

    def _treatment(self, name):
        with self._connect() as con:
            row = con.execute("SELECT Steps, Caution FROM Treatment "
                              "WHERE Name = ?", (name,)).fetchone()
        return row if row else ("", "")

    # ----- learning ------------------------------------------------------
    def feedback(self, answer):
        """answer: consultation, worked (true/false), and when it did not
        work: treatment, steps, caution, extra ("variable = value") and
        staff. Returns what the knowledge base did."""
        number = answer.get("consultation")
        with self._connect() as con:
            row = con.execute("SELECT Facts, NodeID, Treatment, Outcome FROM "
                              "Consultation WHERE ConsultationID = ?",
                              (number,)).fetchone()
        if row is None or row[3] != "open":
            return {"status": "error",
                    "message": "no open consultation " + str(number)}
        facts = [Condition(v, "=", x) for v, x in json.loads(row[0]).items()]
        staff = answer.get("staff") or "counter"
        if answer.get("worked"):
            self.kb.learn(Case(facts, [Conclusion("treatment", row[2])]),
                          staff)
            return self._close(number, "worked", staff, {
                "status": "accepted",
                "message": "Good. The advice was right; it is recorded."})
        treatment = (answer.get("treatment") or "").strip()
        if not treatment:
            return self._close(number, "failed", staff, {
                "status": "closed",
                "message": "Nothing to learn this time."})
        try:
            for extra in answer.get("extra", []):
                facts.append(Condition.parse(extra))
            case = Case(facts, [Conclusion("treatment", treatment)])
            out = {"status": "learned"}
            if row[1]:
                d = self.kb.differences(row[1], case)
                conds = [c.negated() for c in d.a] + d.b
                out["differences"] = d.lines()
                out["affected"] = self.kb.affected_cornerstones(
```

Kode 10.3 (Lanjutan) *advisor.py: logika penasihat noda, dengan pengetahuannya di SQLite.*

```
                    row[1], conds, case.conclusions)
            learned = self.kb.learn(case, staff)
        except VcirsError as e:
            return {"status": "error", "message": str(e)}
        with self._connect() as con:
            con.execute("INSERT OR IGNORE INTO Treatment VALUES (?, ?, ?)",
                        (treatment, answer.get("steps", ""),
                         answer.get("caution", "")))
        out["result"] = str(learned)
        return self._close(number, "taught", staff, out)

    def _close(self, number, outcome, staff, out):
        with self._connect() as con:
            con.execute("UPDATE Consultation SET Outcome = ?, Staff = ? "
                        "WHERE ConsultationID = ?", (outcome, staff, number))
        self.store.save(self.kb)
        out["summary"] = self.kb.summary()
        return out

    # ----- looking at the knowledge -------------------------------------
    def overview(self):
        with self._connect() as con:
            treatments = [{"name": n, "steps": s, "caution": c}
                          for n, s, c in con.execute(
                              "SELECT Name, Steps, Caution FROM Treatment "
                              "ORDER BY Name")]
            outcomes = {o: n for o, n in con.execute(
                "SELECT Outcome, COUNT(*) FROM Consultation GROUP BY Outcome")}
            top = [{"treatment": t, "times": n} for t, n in con.execute(
                "SELECT Treatment, COUNT(*) AS n FROM Consultation "
                "WHERE Treatment <> '' GROUP BY Treatment "
                "ORDER BY n DESC, Treatment LIMIT 5")]
        counts = {"total": sum(outcomes.values())}
        for o in ("worked", "taught", "failed", "open"):
            counts[o] = outcomes.get(o, 0)
        return {"summary": self.kb.summary(), "tree": report.tree(self.kb),
                "consultations": counts, "mostGiven": top,
                "history": list(reversed(self.kb.history[-8:])),
                "treatments": treatments}
```

Mari kita baca dari atas.

Pertanyaan dan tabel. `QUESTIONS` mendeklarasikan empat variabel yang ditanyakan kepada staf, masing-masing dengan pertanyaan dan jawaban lazimnya, seperti pada dokter Wi-Fi. `TABLES` membuat dua tabel milik laundry bila belum ada. `Treatment` menyimpan langkah dan peringatan setiap penanganan menurut namanya; basis pengetahuan hanya menyimpulkan *nama*-nya (`treatment = enzyme soak`), dan halaman mencari langkah-langkahnya. Memisahkan keduanya membuat nama node tetap pendek dan memungkinkan Bu Rini memperbaiki kata-kata sebuah penanganan tanpa menyentuh pengetahuannya. `Consultation` menyimpan satu baris per konsultasi: kapan, jawaban-jawabannya (sebagai teks JSON), node yang menjawab, penanganannya, hasilnya (`open`, `worked`, `taught`, atau `failed`), dan siapa yang berada di meja depan. `ConsultationID INTEGER PRIMARY KEY` membuat SQLite memberi nomor baris dengan sendirinya: 1, 2, 3, dan seterusnya.

Jam. `Clock` memberikan waktu sebagai teks seperti `2026-09-28 08:03`. Biasanya ini waktu sungguhan. Bila diberi waktu mulai, ia menjadi jam pura-pura yang maju satu menit setiap kali dibaca; contoh-contoh dalam bab ini memakainya agar dapat diulang persis, dalam kedua bahasa, pada hari apa pun. Basis pengetahuan memakai jam yang sama untuk riwayatnya (`self.kb.clock = clock`), sehingga entri riwayat dan konsultasi saling cocok.

Membuka. Konstruktor memeriksa apakah berkas basis data sudah ada. Pertama kali, berkas itu belum ada, dan `_create` membangun basis pengetahuan dari kasus benih, mendeklarasikan pertanyaan-pertanyaannya, menyimpannya dengan `SqliteStore`, dan mengisi tabel Treatment dari `treatments.json`. Setelah itu basis pengetahuan selalu dimuat dari basis data. `_connect` membuka koneksi dan memastikan tabel-tabel laundry ada; bentuk `with self._connect() as con:` melakukan *commit* atas perubahan di akhir blok, sebagai satu transaksi.

Berkonsultasi. `consult` adalah pola jalankan-ulang dari Subbab 10.2.4. Ia mengubah jawaban-jawaban menjadi fakta bertanda jawaban `"user"`, sehingga penjelasannya berbunyi *answered by the user*, seperti di Bab 3. Asker-nya punya dua perilaku: untuk variabel dalam `unknown` ia mengembalikan `None`, "saya tidak tahu"; untuk variabel lain ia *melempar* `NeedAnswer`, sebuah pengecualian (*exception*) yang membawa pertanyaannya. Melempar pengecualian adalah cara paling sederhana untuk menghentikan mesin di tengah pekerjaannya: mesin tidak menangkapnya, sehingga ia langsung terbang keluar dari `kb.infer` ke `except NeedAnswer` milik `consult`, yang mengubah pertanyaan itu menjadi jawaban berisi variabel, teks pertanyaan, jawaban-jawaban lazim (`q.choices`, yang juga memuat setiap nilai yang pernah dilihat basis pengetahuan untuk variabel itu), dan `q.why()`, penjelasan mesin sendiri tentang mengapa ia bertanya. Bila `infer` selesai, tidak ada pertanyaan terbuka: konsultasinya dicatat, dan jawabannya berupa `none` atau saran, lengkap dengan langkah dan peringatan penanganan dari basis data serta `result.how()`, penjelasan dari Bab 3.

Belajar. `feedback` adalah paruh kedua `main` milik dokter Wi-Fi, yang bekerja pada konsultasi yang tercatat, bukan pada percakapan. Ia membaca baris konsultasinya; konsultasi yang tidak ada atau tidak lagi `open` menghasilkan galat, sehingga pelajaran yang sama tidak dapat diajarkan dua kali karena sebuah tombol ditekan dua kali. Bila sarannya berhasil, kasusnya diberikan kepada `learn`, yang mencatatnya sebagai kasus terverifikasi (opsi 0), dan konsultasinya ditutup sebagai `worked`. Bila tidak berhasil, dan staf mengetikkan apa yang berhasil sebagai gantinya, fakta-fakta tambahan dimasukkan ke dalam kasus, daftar perbedaan dan cornerstone yang terdampak dihitung terhadap node yang menjawab (untuk jawaban, agar halaman dapat menampilkannya), dan `learn` menambahkan pengetahuan baru: sebuah pengecualian bila ada node yang menjawab, sebuah node tingkat atas baru bila tidak ada. `INSERT OR IGNORE` menyimpan langkah dan peringatan penanganan baru, dan membiarkan penanganan yang sudah ada dengan nama yang sama. `_close` menandai konsultasinya, menyimpan basis pengetahuan ke basis data, dan menambahkan ringkasan satu baris pada jawabannya.

Ikhtisar. `overview` mengumpulkan semua yang ditampilkan halaman pengetahuan dalam satu jawaban. Tiga pernyataan SQL melakukan penghitungan: daftar penanganan menurut abjad; `SELECT Outcome, COUNT(*) FROM Consultation GROUP BY Outcome`, yang menghitung konsultasi menurut hasilnya dalam sekali jalan; dan lima penanganan yang paling sering diberikan, `ORDER BY n DESC, Treatment` agar nilai seri diputuskan menurut nama dan urutannya selalu sama. Riwayatnya adalah delapan entri terakhir, yang terbaru lebih dahulu.

Kode 10.4 adalah Advisor dalam Java.

Kode 10.4 `Advisor.java`: logika yang sama dalam Java, melalui JDBC.

```
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.function.Supplier;

import vcirs.Case;
import vcirs.CaseFile;
import vcirs.Conclusion;
import vcirs.Condition;
import vcirs.DifferenceList;
import vcirs.Fact;
import vcirs.InferenceResult;
import vcirs.Json;
import vcirs.KnowledgeBase;
import vcirs.LearnResult;
import vcirs.Question;
import vcirs.Report;
import vcirs.SqliteStore;
import vcirs.Variable;
import vcirs.VcirsException;

/**
 * The stain advisor: a VCIRS knowledge base kept in a SQLite database.
 * Advisor has no web code at all. Its three methods take and return plain
 * maps, which the web server turns into JSON:
 *   consult(facts, unknown)  the next question, or the advice
 *   feedback(answer)         did the advice work? if not, learn the right one
 *   overview()               the knowledge base, the treatments, the numbers
 */
public class Advisor {
    static final String[][] QUESTIONS = {
        {"stain", "What is the stain?", "oil", "blood", "tea", "ink", "gum",
            "wax", "mud", "turmeric", "rust"},
```

Kode 10.4 (Lanjutan) *Advisor.java: logika yang sama dalam Java, melalui JDBC.*

```
        {"age", "Is the stain fresh or dried?", "fresh", "dried"},
        {"color", "Is the item white or colored?", "white", "colored"},
        {"fabric", "What is the fabric?", "cotton", "polyester", "denim",
            "silk", "wool"},
    };
    static final String[] TABLES = {
        "CREATE TABLE IF NOT EXISTS Treatment (Name TEXT PRIMARY KEY, "
            + "Steps TEXT, Caution TEXT)",
        "CREATE TABLE IF NOT EXISTS Consultation (ConsultationID INTEGER "
            + "PRIMARY KEY, Time TEXT, Facts TEXT, NodeID TEXT, Treatment TEXT, "
            + "Outcome TEXT, Staff TEXT)",
    };

    /**
     * The time now; or, given a start, a pretend clock that moves one
     * minute each time it is read (so that a demonstration repeats exactly).
     */
    static class Clock implements Supplier<String> {
        static final DateTimeFormatter F =
                DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");
        final LocalDateTime start;
        int ticks = 0;

        Clock(String start) {
            this.start = start == null ? null : LocalDateTime.parse(start, F);
        }

        public String get() {
            LocalDateTime now = start == null ? LocalDateTime.now()
                    : start.plusMinutes(ticks++);
            return now.format(F);
        }
    }

    /** Thrown by the asker to stop the engine at the first open question. */
    static class NeedAnswer extends RuntimeException {
        final Question question;

        NeedAnswer(Question question) {
            super(question.variable);
            this.question = question;
        }
    }
```

Kode 10.4 (Lanjutan) *Advisor.java: logika yang sama dalam Java, melalui JDBC.*

```
    final String db;
    final SqliteStore store;
    final Supplier<String> clock;
    final KnowledgeBase kb;

    Advisor(String db, String dataDir, Supplier<String> clock) throws Exception {
        this.db = db;
        this.store = new SqliteStore(db);
        this.clock = clock;
        if (!Files.exists(Path.of(db))) {
            create(dataDir);
        }
        kb = store.load();
        kb.clock = clock;
    }

    /** The first run: build the knowledge from the seed cases. */
    @SuppressWarnings("unchecked")
    void create(String dataDir) throws Exception {
        CaseFile seed = CaseFile.read(Path.of(dataDir,
                "stain-seed-cases.json").toString());
        KnowledgeBase k = new KnowledgeBase(seed.name, seed.description);
        for (String[] q : QUESTIONS) {
            k.declare(q[0], q[1], List.of(q).subList(2, q.length));
        }
        seed.build(k);
        store.save(k);
        String text = Files.readString(Path.of(dataDir, "treatments.json"),
                StandardCharsets.UTF_8);
        Map<String, Object> file = (Map<String, Object>) Json.parse(text);
        try (Connection con = connect()) {
            for (Object o : (List<Object>) file.get("treatments")) {
                Map<String, Object> t = (Map<String, Object>) o;
                update(con, "INSERT INTO Treatment VALUES (?, ?, ?)",
                        t.get("name"), t.get("steps"), t.get("caution"));
            }
        }
    }
```

Kode 10.4 (Lanjutan) *Advisor.java: logika yang sama dalam Java, melalui JDBC.*

```
    Connection connect() throws SQLException {
        Connection con = DriverManager.getConnection("jdbc:sqlite:" + db);
        try (Statement st = con.createStatement()) {
            for (String sql : TABLES) {
                st.execute(sql);
            }
        }
        return con;
    }

    static int update(Connection con, String sql, Object... values)
            throws SQLException {
        try (PreparedStatement ps = con.prepareStatement(sql)) {
            for (int i = 0; i < values.length; i++) {
                ps.setObject(i + 1, values[i]);
            }
            return ps.executeUpdate();
        }
    }

    /** Rows of a SELECT, each a list of column values. */
    static List<List<Object>> query(Connection con, String sql,
                                    Object... values) throws SQLException {
        List<List<Object>> rows = new ArrayList<>();
        try (PreparedStatement ps = con.prepareStatement(sql)) {
            for (int i = 0; i < values.length; i++) {
                ps.setObject(i + 1, values[i]);
            }
            try (ResultSet rs = ps.executeQuery()) {
                int n = rs.getMetaData().getColumnCount();
                while (rs.next()) {
                    List<Object> row = new ArrayList<>();
                    for (int i = 1; i <= n; i++) {
                        row.add(rs.getObject(i));
                    }
                    rows.add(row);
                }
            }
        }
        return rows;
    }
```

Kode 10.4 (Lanjutan) *Advisor.java: logika yang sama dalam Java, melalui JDBC.*

```
    /** {"stain": "oil", "age": "dried"}, written as Python's json.dumps. */
    static String compact(Map<String, String> facts) {
        List<String> parts = new ArrayList<>();
        for (Map.Entry<String, String> f : facts.entrySet()) {
            parts.add(Json.quote(f.getKey()) + ": " + Json.quote(f.getValue()));
        }
        return "{" + String.join(", ", parts) + "}";
    }

    long record(Map<String, String> facts, String nodeId, String treatment,
                String staff) throws SQLException {
        try (Connection con = connect()) {
            update(con, "INSERT INTO Consultation (Time, Facts, NodeID, "
                    + "Treatment, Outcome, Staff) VALUES (?, ?, ?, ?, 'open', ?)",
                    clock.get(), compact(facts), nodeId, treatment, staff);
            return ((Number) query(con, "SELECT last_insert_rowid()")
                    .get(0).get(0)).longValue();
        }
    }

    // ----- consulting ----------------------------------------------------
    /**
     * Run the knowledge base on the answers so far. If it needs one more
     * answer, return that question; otherwise record the consultation and
     * return the advice (or say there is none).
     */
    Map<String, Object> consult(Map<String, String> facts, List<String> unknown,
                                String staff) throws SQLException {
        Map<String, Fact> given = new LinkedHashMap<>();
        for (Map.Entry<String, String> f : facts.entrySet()) {
            given.put(f.getKey(), new Fact(f.getKey(), f.getValue(), 100,
                    "user"));
        }
        Map<String, Object> out = new LinkedHashMap<>();
        InferenceResult result;
        try {
            result = kb.infer(given, q -> {
                if (unknown.contains(q.variable)) {
                    return null;                        // "I do not know"
                }
                throw new NeedAnswer(q);
            });
        } catch (NeedAnswer e) {
            Question q = e.question;
            Variable v = kb.variables.get(q.variable);
            out.put("status", "question");
```

Kode 10.4 (Lanjutan) *Advisor.java: logika yang sama dalam Java, melalui JDBC.*

```
            out.put("variable", q.variable);
            out.put("question", v != null && !v.question.isEmpty()
                    ? v.question : q.variable + "?");
            out.put("choices", q.choices);
            out.put("why", q.why());
            return out;
        }
        if (result.conclusions().isEmpty()) {
            out.put("status", "none");
            out.put("consultation", record(facts, "", "", staff));
            out.put("message", "No advice for this stain yet.");
            return out;
        }
        String treatment = result.conclusions().get(0).value;
        String node = result.paths.get(0).last().id;
        long number = record(facts, node, treatment, staff);
        List<String> t = treatment(treatment);
        out.put("status", "advice");
        out.put("consultation", number);
        out.put("treatment", treatment);
        out.put("steps", t.get(0));
        out.put("caution", t.get(1));
        out.put("node", node);
        out.put("how", result.how());
        return out;
    }

    List<String> treatment(String name) throws SQLException {
        try (Connection con = connect()) {
            List<List<Object>> rows = query(con, "SELECT Steps, Caution FROM "
                    + "Treatment WHERE Name = ?", name);
            return rows.isEmpty() ? List.of("", "")
                    : List.of((String) rows.get(0).get(0),
                              (String) rows.get(0).get(1));
        }
    }
```

Kode 10.4 (Lanjutan) *Advisor.java: logika yang sama dalam Java, melalui JDBC.*

```
    // ----- learning ------------------------------------------------------
    /**
     * answer: consultation, worked (true/false), and when it did not work:
     * treatment, steps, caution, extra ("variable = value") and staff.
     * Returns what the knowledge base did.
     */
    @SuppressWarnings("unchecked")
    Map<String, Object> feedback(Map<String, Object> answer) throws Exception {
        Object number = answer.get("consultation");
        List<List<Object>> rows;
        try (Connection con = connect()) {
            rows = query(con, "SELECT Facts, NodeID, Treatment, Outcome FROM "
                    + "Consultation WHERE ConsultationID = ?", number);
        }
        Map<String, Object> out = new LinkedHashMap<>();
        if (rows.isEmpty() || !"open".equals(rows.get(0).get(3))) {
            out.put("status", "error");
            out.put("message", "no open consultation " + number);
            return out;
        }
        List<Object> row = rows.get(0);
        List<Condition> facts = new ArrayList<>();
        Map<String, Object> stored =
                (Map<String, Object>) Json.parse((String) row.get(0));
        for (Map.Entry<String, Object> f : stored.entrySet()) {
            facts.add(new Condition(f.getKey(), "=", (String) f.getValue()));
        }
        String staff = text(answer, "staff");
        staff = staff.isEmpty() ? "counter" : staff;
        if (Boolean.TRUE.equals(answer.get("worked"))) {
            kb.learn(new Case(facts, List.of(new Conclusion("treatment",
                    (String) row.get(2))), 100), staff);
            out.put("status", "accepted");
            out.put("message", "Good. The advice was right; it is recorded.");
            return close(number, "worked", staff, out);
        }
        String treatment = text(answer, "treatment").strip();
        if (treatment.isEmpty()) {
            out.put("status", "closed");
            out.put("message", "Nothing to learn this time.");
            return close(number, "failed", staff, out);
        }
        String nodeId = (String) row.get(1);
        LearnResult learned;
        try {
            for (Object extra : (List<Object>) answer.getOrDefault("extra",
```

Kode 10.4 (Lanjutan) *Advisor.java: logika yang sama dalam Java, melalui JDBC.*

```
                    List.of())) {
                facts.add(Condition.parse((String) extra));
            }
            Case c = new Case(facts, List.of(new Conclusion("treatment",
                    treatment)), 100);
            out.put("status", "learned");
            if (!nodeId.isEmpty()) {
                DifferenceList d = kb.differences(nodeId, c);
                List<Condition> conds = new ArrayList<>();
                for (Condition x : d.a) {
                    conds.add(x.negated());
                }
                conds.addAll(d.b);
                out.put("differences", d.lines());
                out.put("affected", kb.affectedCornerstones(nodeId, conds,
                        c.conclusions));
            }
            learned = kb.learn(c, staff);
        } catch (VcirsException e) {
            out.clear();
            out.put("status", "error");
            out.put("message", e.getMessage());
            return out;
        }
        try (Connection con = connect()) {
            update(con, "INSERT OR IGNORE INTO Treatment VALUES (?, ?, ?)",
                    treatment, text(answer, "steps"), text(answer, "caution"));
        }
        out.put("result", learned.toString());
        return close(number, "taught", staff, out);
    }

    static String text(Map<String, Object> map, String key) {
        Object v = map.get(key);
        return v == null ? "" : v.toString();
    }
```

Kode 10.4 (Lanjutan) *Advisor.java: logika yang sama dalam Java, melalui JDBC.*

```
    Map<String, Object> close(Object number, String outcome, String staff,
                              Map<String, Object> out) throws SQLException {
        try (Connection con = connect()) {
            update(con, "UPDATE Consultation SET Outcome = ?, Staff = ? "
                    + "WHERE ConsultationID = ?", outcome, staff, number);
        }
        store.save(kb);
        out.put("summary", kb.summary());
        return out;
    }

    // ----- looking at the knowledge --------------------------------------
    Map<String, Object> overview() throws SQLException {
        List<Object> treatments = new ArrayList<>();
        Map<String, Long> outcomes = new LinkedHashMap<>();
        List<Object> top = new ArrayList<>();
        try (Connection con = connect()) {
            for (List<Object> r : query(con, "SELECT Name, Steps, Caution "
                    + "FROM Treatment ORDER BY Name")) {
                Map<String, Object> t = new LinkedHashMap<>();
                t.put("name", r.get(0));
                t.put("steps", r.get(1));
                t.put("caution", r.get(2));
                treatments.add(t);
            }
            for (List<Object> r : query(con, "SELECT Outcome, COUNT(*) FROM "
                    + "Consultation GROUP BY Outcome")) {
                outcomes.put((String) r.get(0), ((Number) r.get(1)).longValue());
            }
            for (List<Object> r : query(con, "SELECT Treatment, COUNT(*) AS n "
                    + "FROM Consultation WHERE Treatment <> '' "
                    + "GROUP BY Treatment ORDER BY n DESC, Treatment LIMIT 5")) {
                Map<String, Object> t = new LinkedHashMap<>();
                t.put("treatment", r.get(0));
                t.put("times", ((Number) r.get(1)).longValue());
                top.add(t);
            }
        }
        Map<String, Object> counts = new LinkedHashMap<>();
        long total = 0;
        for (long n : outcomes.values()) {
            total += n;
        }
        counts.put("total", total);
        for (String o : List.of("worked", "taught", "failed", "open")) {
            counts.put(o, outcomes.getOrDefault(o, 0L));
```

Kode 10.4 (Lanjutan) *Advisor.java: logika yang sama dalam Java, melalui JDBC.*

```
        }
        List<Object> history = new ArrayList<>(kb.history.subList(
                Math.max(0, kb.history.size() - 8), kb.history.size()));
        Collections.reverse(history);
        Map<String, Object> out = new LinkedHashMap<>();
        out.put("summary", kb.summary());
        out.put("tree", Report.tree(kb));
        out.put("consultations", counts);
        out.put("mostGiven", top);
        out.put("history", history);
        out.put("treatments", treatments);
        return out;
    }
}
```

Versi Java memiliki metode yang sama dalam urutan yang sama, dengan beberapa perbedaan yang diminta Java. JDBC tidak memiliki padanan satu baris untuk `con.execute(sql, values)` milik Python, sehingga dua pembantu kecil melakukan pekerjaannya: `update` menjalankan `INSERT` atau `UPDATE` dengan penanda tempat `?` yang diisi dari argumennya, dan `query` menjalankan `SELECT` dan mengembalikan baris-barisnya sebagai daftar. Penanda tempat itu penting: nilai yang diketik staf tidak pernah ditempelkan ke dalam teks SQL, sehingga penanganan bernama `it's` atau yang lebih aneh lagi tidak dapat merusak pernyataannya. Setelah sebuah `INSERT`, `SELECT last_insert_rowid()` memberikan nomor yang dipilih SQLite untuk baris baru. Asker-nya berupa lambda, `q -> { ... }`, dan `NeedAnswer` merupakan turunan `RuntimeException`, jenis pengecualian yang boleh dilempar lambda tanpa dideklarasikan. `compact` menulis jawaban-jawaban sebagai teks JSON persis seperti yang ditulis `json.dumps` milik Python, sehingga kedua bahasa menyimpan teks yang sama di kolom Facts dan basis data mereka identik, baris demi baris.

10.5 Langkah 3: Sehari di Laundry, Tanpa Peramban

Setelah Advisor selesai, kita dapat memutar ulang satu pagi penuh sebelum menulis kode web apa pun. Berkas `data/tour.txt` mendaftar lima belas permintaan, satu per baris, persis seperti yang akan dikirim halaman. **Kode 10.5** membaca setiap baris, memanggil Advisor sebagaimana server akan memanggilnya, lalu mencetak permintaan dan jawaban JSON-nya.

Kode 10.5 `advisor_demo.py`: kelima belas permintaan tur, diputar ulang pada Advisor.

```
"""Replay the requests of ../data/tour.txt on the Advisor, without a web
server, and print every answer as the server would send it.

    python advisor_demo.py
"""

import json
import os
import pathlib
import sys

from advisor import Advisor, Clock
from vcirs import jsonio

DATA = pathlib.Path(__file__).resolve().parents[1] / "data"
DB = str(DATA / "demo.db")

if os.path.exists(DB):
    os.remove(DB)                          # a fresh laundry every time
advisor = Advisor(DB, str(DATA), Clock("2026-09-28 08:00"))
with open(DATA / "tour.txt", encoding="utf-8") as f:
    for line in f:
        method, path, *rest = line.strip().split(" ", 2)
        body = json.loads(rest[0]) if rest else {}
        if path == "/api/consult":
            out = advisor.consult(body.get("facts", {}),
                                  body.get("unknown", []),
                                  body.get("staff", ""))
        elif path == "/api/feedback":
            out = advisor.feedback(body)
        else:
            out = advisor.overview()
        print("> " + line.strip())
        print(jsonio.dumps(out))
```

Kode 10.6 melakukan hal yang sama dalam Java; ia meminjam tiga pembantu kecil dari server di Subbab 10.6, yang mengubah badan JSON yang sudah diurai menjadi *map* dan daftar yang diharapkan Advisor.

Kode 10.6 `AdvisorDemo.java`: tur yang sama dalam Java.

```
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.List;
import java.util.Map;

import vcirs.Json;

/**
 * Replay the requests of ../data/tour.txt on the Advisor, without a web
 * server, and print every answer as the server would send it.
 * java -cp "out:lib/sqlite-jdbc-3.53.2.0.jar" AdvisorDemo  (Windows: ;)
 */
public class AdvisorDemo {
    public static void main(String[] args) throws Exception {
        Path data = Path.of("..", "data");
        String db = data.resolve("demo.db").toString();
        Files.deleteIfExists(Path.of(db));     // a fresh laundry every time
        Advisor advisor = new Advisor(db, data.toString(),
                new Advisor.Clock("2026-09-28 08:00"));
        for (String line : Files.readAllLines(data.resolve("tour.txt"),
                StandardCharsets.UTF_8)) {
            String[] part = line.strip().split(" ", 3);
            Map<String, Object> body = part.length > 2
                    ? LaundryServer.object(part[2]) : Map.of();
            Map<String, Object> out;
            if (part[1].equals("/api/consult")) {
                out = advisor.consult(LaundryServer.strings(body, "facts"),
                        LaundryServer.list(body, "unknown"),
                        Advisor.text(body, "staff"));
            } else if (part[1].equals("/api/feedback")) {
                out = advisor.feedback(body);
            } else {
                out = advisor.overview();
            }
            System.out.println("> " + line.strip());
            System.out.println(Json.write(out));
        }
    }
}
```

Kedua program mencetak 353 baris yang sama, bita demi bita, dan meninggalkan basis data yang sama. Inilah pagi itu, permintaan demi permintaan. Permintaan pertama tidak berisi jawaban sama sekali:

```
> POST /api/consult {"facts": {}, "unknown": []}
{
  "status": "question",
  "variable": "stain",
  "question": "What is the stain?",
  "choices": [
    "oil",
    "blood",
    "tea",
    "ink",
    "gum",
    "wax",
    "mud",
    "turmeric",
    "rust"
  ],
  "why": [
    "I need to know 'stain'.",
    "Rule dish soap and a warm wash#1 needs stain = oil;",
    "if it holds, the rule concludes treatment = dish soap and a warm wash."
  ]
}
```

Mesin menelusuri node tingkat atas sesuai urutan pengajarannya; node pertama, *dish soap and a warm wash*, membutuhkan `stain = oil`, nodanya belum diketahui, dan asker menghentikan mesin. Jawabannya membawa pertanyaan beserta alasan mesin menanyakannya. Dewi menjawab *oil*:

```
> POST /api/consult {"facts": {"stain": "oil"}, "unknown": []}
{
  "status": "advice",
  "consultation": 1,
  "treatment": "dish soap and a warm wash",
  "steps": "Rub a drop of dish soap into the stain with your fingers. Leave it for 10
minutes. Wash in the warmest water the care label allows.",
  "caution": "Check the stain before drying; heat sets what is left.",
  "node": "dish soap and a warm wash#1",
  "how": [
    "dish soap and a warm wash#1 fired because:",
    "  stain = oil  (answered by the user, CF 100)",
    "  so treatment = dish soap and a warm wash  [rule CF 100, conclusion CF 100]"
  ]
}
```

Kini node pertama menyala; ia tidak punya pengecualian, jadi tidak ada lagi yang perlu ditanyakan, dan saran kembali lengkap dengan langkah, peringatan, dan penjelasannya. Konsultasi ini dicatat sebagai nomor 1. Setelah dicuci, kemejanya bersih, dan Dewi mengatakannya:

```
> POST /api/feedback {"consultation": 1, "worked": true, "staff": "Dewi"}
{
  "status": "accepted",
  "message": "Good. The advice was right; it is recorded.",
  "summary": "Stain advisor: 5 variables, 12 nodes, 12 rules"
}
```

Kasusnya dicatat sebagai kasus terverifikasi. Pelanggan berikutnya membawa kemeja dengan darah di kerahnya. Dengan `stain = blood`, node *cold rinse and soak* menyala, dan pengecualiannya, *enzyme soak*, perlu tahu umur nodanya:

```
> POST /api/consult {"facts": {"stain": "blood"}, "unknown": []}
{
  "status": "question",
  "variable": "age",
  "question": "Is the stain fresh or dried?",
  "choices": [
    "fresh",
    "dried"
  ],
  "why": [
    "I need to know 'age'.",
    "Rule enzyme soak#1 needs age = dried;",
    "if it holds, the rule concludes treatment = enzyme soak."
  ]
}
> POST /api/consult {"facts": {"stain": "blood", "age": "dried"}, "unknown": []}
{
  "status": "advice",
  "consultation": 2,
  "treatment": "enzyme soak",
  "steps": "Soak for 1 to 2 hours in lukewarm water with an enzyme (bio) detergent. Rub
gently, rinse and wash as usual.",
  "caution": "Not for silk or wool: enzymes attack them.",
  "node": "enzyme soak#1",
  "how": [
    "enzyme soak#1 fired because:",
    "  stain = blood  (answered by the user, CF 100)",
    "  age = dried  (answered by the user, CF 100)",
    "  so treatment = enzyme soak  [rule CF 100, conclusion CF 100]"
  ]
}
```

Noda darah yang sudah kering mendapat rendaman enzim, dan penjelasannya memperlihatkan kedua fakta yang membuat pengecualian itu menyala. (Permintaan keenam, umpan balik bahwa sarannya berhasil, serupa dengan yang ketiga.) Lalu datang noda yang belum pernah dilihat penasihat:

```
> POST /api/consult {"facts": {"stain": "lipstick"}, "unknown": []}
{
  "status": "none",
  "consultation": 3,
  "message": "No advice for this stain yet."
}
> POST /api/feedback {"consultation": 3, "worked": false, "treatment": "makeup remover
dab", "steps": "Dab micellar water or makeup remover on the stain with cotton, from the
outside in. Rinse, rub in liquid detergent and wash.", "caution": "Test on a hidden seam
first.", "extra": [], "staff": "Dewi"}
{
  "status": "learned",
  "result": "added: makeup remover dab#1 added at the top level",
  "summary": "Stain advisor: 5 variables, 13 nodes, 13 rules"
}
```

Lipstik tidak cocok dengan node mana pun; jawabannya `none`, dengan nomor konsultasi 3. Dewi menanganinya sesuai pengetahuannya — pembersih riasan dengan kapas — dan mengajari penasihat, lengkap dengan langkah dan peringatannya. Karena tidak ada node yang menjawab, `learn` menambahkan kasusnya sebagai node tingkat atas baru, opsi 5 dari enam opsi revisi VCIRS tahun 2006 yang diperkenalkan di Bab 3 [9]. Basis pengetahuan kini memiliki 13 node.

Pelajaran paling menarik pagi itu datang dari Joko. Seorang pelanggan membawa blus sutra dengan setetes minyak goreng. Penasihat, ketika ditanya tentang noda minyak, mengatakan apa yang selalu dikatakannya:

```
> POST /api/consult {"facts": {"stain": "oil"}, "unknown": []}
{
  "status": "advice",
  "consultation": 4,
  "treatment": "dish soap and a warm wash",
  "steps": "Rub a drop of dish soap into the stain with your fingers. Leave it for 10
minutes. Wash in the warmest water the care label allows.",
  "caution": "Check the stain before drying; heat sets what is left.",
  "node": "dish soap and a warm wash#1",
  "how": [
    "dish soap and a warm wash#1 fired because:",
    "  stain = oil  (answered by the user, CF 100)",
    "  so treatment = dish soap and a warm wash  [rule CF 100, conclusion CF 100]"
  ]
}
```

Namun sutra tidak boleh digosok sabun cuci piring dan dicuci hangat; Joko menaburkan bedak talek di atas minyak untuk menyerapnya, lalu mengirim blus itu ke *dry cleaner*. Ia mengajarkan hal ini, dan menambahkan satu fakta yang membuat perbedaan, `fabric = silk`:

```
> POST /api/feedback {"consultation": 4, "worked": false, "treatment": "powder and dry
cleaner", "steps": "Sprinkle talcum powder thickly on the oil and leave it overnight. Brush
it off and send the item to the dry cleaner.", "caution": "No water and no soap on silk.",
"extra": ["fabric = silk"], "staff": "Joko"}
{
  "status": "learned",
  "differences": [
    "Differences with the cornerstone case of dish soap and a warm wash#1:",
    "  A (only in the old case): (none)",
    "  B (only in the new case): fabric = silk"
  ],
  "affected": [],
  "result": "refined: powder and dry cleaner#1 added as an exception of dish soap and a
warm wash#1",
  "summary": "Stain advisor: 5 variables, 14 nodes, 14 rules"
}
```

Inilah Ripple Down Rules yang bekerja lewat formulir web [10]. Daftar perbedaan terhadap kasus cornerstone node yang menjawab (sekadar `stain = oil`) tidak berisi apa pun di bagian A dan `fabric = silk` di bagian B; tidak ada kasus terdahulu yang jawabannya akan berubah; dan pengetahuan baru itu menjadi pengecualian bagi *dish soap and a warm wash*. Noda minyak berikutnya ditangani secara berbeda:

```
> POST /api/consult {"facts": {"stain": "oil"}, "unknown": []}
{
  "status": "question",
  "variable": "fabric",
  "question": "What is the fabric?",
  "choices": [
    "cotton",
    "polyester",
    "denim",
    "silk",
    "wool"
  ],
  "why": [
    "I need to know 'fabric'.",
    "Rule powder and dry cleaner#1 needs fabric = silk;",
    "if it holds, the rule concludes treatment = powder and dry cleaner."
  ]
}
> POST /api/consult {"facts": {"stain": "oil", "fabric": "silk"}, "unknown": []}
{
  "status": "advice",
  "consultation": 5,
  "treatment": "powder and dry cleaner",
  "steps": "Sprinkle talcum powder thickly on the oil and leave it overnight. Brush it off
and send the item to the dry cleaner.",
  "caution": "No water and no soap on silk.",
  "node": "powder and dry cleaner#1",
  "how": [
    "powder and dry cleaner#1 fired because:",
    "  stain = oil  (answered by the user, CF 100)",
    "  fabric = silk  (answered by the user, CF 100)",
    "  so treatment = powder and dry cleaner  [rule CF 100, conclusion CF 100]"
  ]
}
```

Penasihat kini menanyakan bahan kainnya — hanya untuk noda minyak, karena node baru berada di bawah node minyak — dan untuk sutra ia memberikan penanganan Joko. Perhatikan urutan pilihannya: jawaban-jawaban yang dideklarasikan, dengan `silk` di antaranya. Konsultasi terakhir pagi itu memperlihatkan "saya tidak tahu":

```
> POST /api/consult {"facts": {"stain": "ink"}, "unknown": ["fabric"]}
{
  "status": "advice",
  "consultation": 6,
  "treatment": "rubbing alcohol from the back",
  "steps": "Put a clean towel under the stain. Dab rubbing alcohol on the back of the stain
with cotton, moving to a clean part of the towel as the ink lifts. Rinse and wash.",
  "caution": "Test on a hidden seam first. Keep away from flames.",
  "node": "rubbing alcohol from the back#1",
  "how": [
    "rubbing alcohol from the back#1 fired because:",
    "  stain = ink  (answered by the user, CF 100)",
    "  so treatment = rubbing alcohol from the back  [rule CF 100, conclusion CF 100]"
  ]
}
```

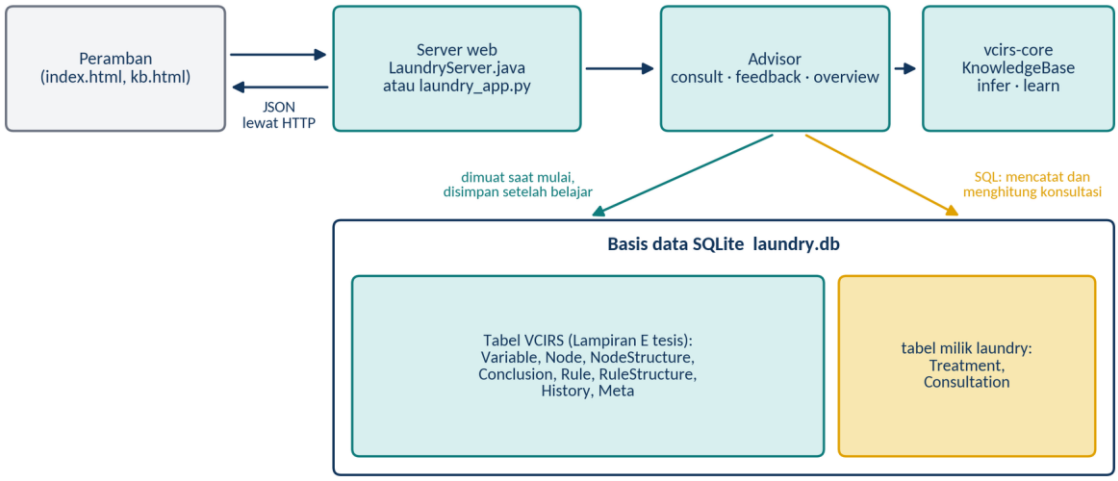
Seorang pelanggan membawa seragam dengan garis tinta dan tidak ada yang tahu apa bahannya. Halaman mengirim `fabric` di dalam `unknown`; asker menjawab "tidak diketahui" alih-alih berhenti, pengecualian sutra tidak menyala, dan penanganan tinta yang umum yang kembali. Subbab 10.9 kembali ke pertanyaan apakah perilaku ini tepat. Permintaan terakhir, `GET /api/kb`, mengembalikan ikhtisar; bagian pertamanya adalah:

```
  "summary": "Stain advisor: 5 variables, 14 nodes, 14 rules",
  "consultations": {"total": 6, "worked": 3, "taught": 2, "failed": 0, "open": 1},
  "mostGiven": [{"treatment": "dish soap and a warm wash", "times": 2}, ...]
```

Enam konsultasi: tiga langsung berhasil, dua mengajari penasihat sesuatu, dan satu — tinta tadi — masih terbuka, menunggu seragamnya keluar dari mesin cuci.

10.6 Langkah 4: Server Web

Sekarang bagian webnya. Di sini server punya dua tugas: mengirim halaman (tiga berkas: `index.html`, `kb.html`, dan `style.css`) dan menjawab ketiga panggilan JSON dengan meneruskannya kepada Advisor. **Gambar 10.3** memperlihatkan bagaimana bagian-bagiannya tersusun.



Gambar 10.3 Penasihat noda sebagai aplikasi web. Peramban dan server berbicara dalam JSON lewat HTTP; Advisor berkonsultasi dengan dan mengajari basis pengetahuan vcirs-core, yang dimuat dari dan disimpan ke tabel-tabel VCIRS, serta mencatat konsultasi di tabel milik laundry.

Kode 10.7 adalah server Java. Ia memakai `com.sun.net.httpserver.HttpServer`, server HTTP kecil yang sudah bertahun-tahun menjadi bagian JDK (dalam modul `jdk.httpserver`) dan tidak memerlukan pemasangan apa pun [11].

Kode 10.7 `LaundryServer.java`: server web dalam Java, di atas `HttpServer` bawaan JDK.

```
import java.io.IOException;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpServer;

import vcirs.Json;
import vcirs.VcirsException;

/**
 * The stain advisor as a web app: three pages and three JSON calls.
 * java -cp "out:lib/sqlite-jdbc-3.53.2.0.jar" LaundryServer
 *      [--port 4100] [--db ../data/laundry.db] [--clock "2026-09-28 08:00"]
 *      [--host 0.0.0.0]    (to let phones on the same Wi-Fi use it too)
 * Then open http://localhost:4100 in a browser.
 */
public class LaundryServer {
    static final Path WEB = Path.of("..", "web");
    static final Map<String, String> PAGES = Map.of(
            "/", "index.html", "/kb", "kb.html", "/style.css", "style.css");
    static Advisor advisor;

    @SuppressWarnings("unchecked")
    static Map<String, Object> object(String text) {
        Object o = Json.parse(text);
        if (!(o instanceof Map)) {
            throw new VcirsException("the request must be a JSON object");
        }
        return (Map<String, Object>) o;
    }
```

Kode 10.7 (Lanjutan) *LaundryServer.java: server web dalam Java, di atas HttpServer bawaan JDK.*

```
    /** body["facts"] as variable -> value text. */
    @SuppressWarnings("unchecked")
    static Map<String, String> strings(Map<String, Object> body, String key) {
        Map<String, String> out = new LinkedHashMap<>();
        Object o = body.getOrDefault(key, Map.of());
        for (Map.Entry<String, Object> e : ((Map<String, Object>) o).entrySet()) {
            out.put(e.getKey(), String.valueOf(e.getValue()));
        }
        return out;
    }

    /** body["unknown"] as a list of texts. */
    @SuppressWarnings("unchecked")
    static List<String> list(Map<String, Object> body, String key) {
        List<String> out = new ArrayList<>();
        for (Object o : (List<Object>) body.getOrDefault(key, List.of())) {
            out.add(String.valueOf(o));
        }
        return out;
    }

    static void send(HttpExchange ex, int code, String type, byte[] bytes)
            throws IOException {
        ex.getResponseHeaders().set("Content-Type", type);
        ex.sendResponseHeaders(code, bytes.length);
        try (OutputStream os = ex.getResponseBody()) {
            os.write(bytes);
        }
    }

    static void page(HttpExchange ex) throws IOException {
        String file = PAGES.get(ex.getRequestURI().getPath());
        if (file == null || !ex.getRequestMethod().equals("GET")) {
            send(ex, 404, "text/plain; charset=utf-8",
                    "Not found".getBytes(StandardCharsets.UTF_8));
            return;
        }
        String type = file.endsWith(".css") ? "text/css; charset=utf-8"
                : "text/html; charset=utf-8";
        send(ex, 200, type, Files.readAllBytes(WEB.resolve(file)));
    }
```

Kode 10.7 (Lanjutan) *LaundryServer.java: server web dalam Java, di atas HttpServer bawaan JDK.*

```
    /** Every JSON call: read the body, ask the Advisor, write the answer. */
    static void api(HttpExchange ex) throws IOException {
        String path = ex.getRequestURI().getPath();
        Map<String, Object> out;
        int code = 200;
        try {
            String text = new String(ex.getRequestBody().readAllBytes(),
                    StandardCharsets.UTF_8);
            synchronized (advisor) {                // one change at a time
                if (path.equals("/api/kb")) {
                    out = advisor.overview();
                } else if (!ex.getRequestMethod().equals("POST")) {
                    out = Map.of("status", "error", "message", "use POST");
                } else if (path.equals("/api/consult")) {
                    Map<String, Object> body = object(text);
                    out = advisor.consult(strings(body, "facts"),
                            list(body, "unknown"), Advisor.text(body, "staff"));
                } else if (path.equals("/api/feedback")) {
                    out = advisor.feedback(object(text));
                } else {
                    out = Map.of("status", "error", "message", "no such call");
                }
            }
        } catch (Exception e) {
            out = new LinkedHashMap<>();
            out.put("status", "error");
            out.put("message", String.valueOf(e.getMessage()));
        }
        if ("error".equals(out.get("status"))) {
            code = 400;
        }
        send(ex, code, "application/json",
                (Json.write(out) + "\n").getBytes(StandardCharsets.UTF_8));
    }

    public static void main(String[] args) throws Exception {
        int port = 4100;
        String db = "../data/laundry.db";
        String clock = null;
        String host = "127.0.0.1";                  // this computer only
        for (int i = 0; i + 1 < args.length; i += 2) {
            switch (args[i]) {
                case "--port" -> port = Integer.parseInt(args[i + 1]);
                case "--db" -> db = args[i + 1];
                case "--clock" -> clock = args[i + 1];
                case "--host" -> host = args[i + 1];
```

Kode 10.7 (Lanjutan) *LaundryServer.java: server web dalam Java, di atas HttpServer bawaan JDK.*

```
                default -> throw new IllegalArgumentException(args[i]);
            }
        }
        advisor = new Advisor(db, "../data", new Advisor.Clock(clock));
        HttpServer server = HttpServer.create(
                new InetSocketAddress(host, port), 0);
        server.createContext("/", LaundryServer::page);
        server.createContext("/api/", LaundryServer::api);
        server.start();
        System.out.println("Ready: " + advisor.kb.summary());
        System.out.println("Open http://localhost:" + port + " (Ctrl+C stops)");
    }
}
```

`main` membaca opsi-opsinya, membuat Advisor, dan membuat server pada sebuah **alamat** dan sebuah **port**. Alamat `127.0.0.1` berarti "hanya komputer ini": tidak ada perangkat lain yang dapat tersambung. Port adalah nomor yang membedakan program-program pada satu komputer; bab ini memakai 4100, sehingga alamat penasihat di peramban adalah `http://localhost:4100`. `createContext("/", LaundryServer::page)` mengirim setiap jalur ke metode `page`, kecuali jalur yang diawali `/api/`, yang dikirim `createContext("/api/", LaundryServer::api)` ke `api` (awalan terpanjang yang cocok yang menang).

`page` mencari jalurnya di `PAGES`, sebuah *map* dari tiga jalur yang dikenal aplikasi ke tiga berkas di folder `web`, lalu mengirim berkasnya dengan **jenis konten** (*content type*)-nya, label yang memberi tahu peramban apakah yang diterimanya halaman atau lembar gaya. Jalur lain mendapat jawaban yang dikenal setiap pengguna web, **404 Not found**. Karena hanya ketiga berkas ini yang dapat dikirim, tidak ada yang dapat meminta `../data/laundry.db` kepada server.

`api` membaca badan permintaan, menguraikannya sebagai JSON dengan `Json.parse` milik vcirs-core, memanggil Advisor, dan menulis jawabannya dengan `Json.write`, fungsi yang sama yang menyimpan basis pengetahuan dalam JSON kanonik. Seluruh panggilan berada di dalam `synchronized (advisor)`, yang hanya mengizinkan satu permintaan pada satu waktu memakai Advisor; bila Dewi dan Joko menekan tombol pada detik yang sama, yang satu menunggu beberapa milidetik. Setiap galat — badan yang bukan JSON, konsultasi yang tidak ada — menjadi jawaban JSON dengan `"status": "error"` dan kode HTTP **400** (permintaan yang buruk), sehingga halaman selalu dapat menampilkan pesan alih-alih rusak.

Kode 10.8 adalah server Python, di atas **Flask**, kerangka kerja web kecil yang biasanya dijumpai pertama kali oleh pemrogram Python. Versi terkininya 3.1.3, dirilis Februari 2026 [12]; pasanglah sekali dengan `python -m pip install -r requirements.txt` di folder `python`.

Kode 10.8 `laundry_app.py`: server web yang sama dalam Python, di atas Flask.

```
"""The stain advisor as a web app: three pages and three JSON calls.

    python laundry_app.py [--port 4100] [--db ../data/laundry.db]
                          [--clock "2026-09-28 08:00"]
                          [--host 0.0.0.0]  (phones on the same Wi-Fi too)

Then open http://localhost:4100 in a browser.
"""

import argparse
import pathlib
import threading

from flask import Flask, Response, request, send_from_directory

from advisor import Advisor, Clock
from vcirs import jsonio

HERE = pathlib.Path(__file__).resolve().parents[1]
WEB = str(HERE / "web")
app = Flask(__name__, static_folder=None)
lock = threading.Lock()                  # one change at a time
advisor = None


def reply(out):
    code = 400 if out.get("status") == "error" else 200
    return Response(jsonio.dumps(out) + "\n", status=code,
                    mimetype="application/json")


@app.get("/")
def home():
    return send_from_directory(WEB, "index.html")


@app.get("/kb")
def kb_page():
    return send_from_directory(WEB, "kb.html")


@app.get("/style.css")
def style():
    return send_from_directory(WEB, "style.css")
```

Kode 10.8 (Lanjutan) *laundry_app.py: server web yang sama dalam Python, di atas Flask.*

```
@app.get("/api/kb")
def api_kb():
    with lock:
        return reply(advisor.overview())


@app.post("/api/consult")
def api_consult():
    body = request.get_json(force=True, silent=True)
    if not isinstance(body, dict):
        return reply({"status": "error",
                      "message": "the request must be a JSON object"})
    facts = {k: str(v) for k, v in body.get("facts", {}).items()}
    unknown = [str(v) for v in body.get("unknown", [])]
    with lock:
        return reply(advisor.consult(facts, unknown,
                                     str(body.get("staff", ""))))


@app.post("/api/feedback")
def api_feedback():
    body = request.get_json(force=True, silent=True)
    if not isinstance(body, dict):
        return reply({"status": "error",
                      "message": "the request must be a JSON object"})
    with lock:
        return reply(advisor.feedback(body))


if __name__ == "__main__":
    p = argparse.ArgumentParser()
    p.add_argument("--port", type=int, default=4100)
    p.add_argument("--db", default=str(HERE / "data" / "laundry.db"))
    p.add_argument("--clock", default=None)
    p.add_argument("--host", default="127.0.0.1")   # this computer only
    args = p.parse_args()
    advisor = Advisor(args.db, str(HERE / "data"), Clock(args.clock))
    print("Ready: " + advisor.kb.summary())
    print("Open http://localhost:" + str(args.port) + " (Ctrl+C stops)")
    app.run(host=args.host, port=args.port)
```

Di Flask setiap jalur adalah sebuah fungsi dengan **dekorator** di atasnya: `@app.get("/")` berarti "panggil `home` untuk permintaan GET ke `/`", dan `@app.post("/api/consult")` melakukan hal yang sama untuk permintaan POST. `send_from_directory` mengirim berkas dari folder `web`, dan menolak nama yang akan keluar dari folder itu. `request.get_json(force=True, silent=True)` mengurai badan permintaan dan memberikan `None` alih-alih galat bila badannya bukan JSON. `reply` menulis jawaban dengan `jsonio.dumps` milik vcirs-core — bukan dengan fungsi JSON milik Flask sendiri — agar server Python mengirim bita yang persis sama dengan server Java. Kunci `threading.Lock()` melakukan apa yang dilakukan `synchronized` di Java: server pengembangan Flask menjawab permintaan dalam *thread* yang berjalan bersamaan, dan kunci itu mengizinkan satu *thread* saja pada satu waktu memakai Advisor.

Flask mencetak peringatan ketika mulai: *This is a development server. Do not use it in a production deployment.* Peringatan itu benar untuk toko di internet dengan ribuan pengunjung. Untuk laundry dengan dua staf di Wi-Fi-nya sendiri, server pengembangan sudah cukup; Bab 15 memperlihatkan cara menjalankan aplikasi seperti ini dengan server produksi ketika ia harus berhadapan dengan dunia luar.

10.7 Langkah 5: Halaman-Halamannya

Halaman-halamannya adalah berkas yang sama untuk kedua server: server Java dan Python mengirimnya tanpa mengubah satu bita pun. **Kode 10.9** adalah halaman penasihat. Isinya HTML polos dengan program JavaScript kecil, ditulis untuk peramban terkini apa pun, di ponsel maupun komputer, tanpa pustaka yang perlu diunduh.

Kode 10.9 `index.html`: halaman penasihat, yang menyimpan jawaban-jawaban dan berbicara dengan server.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Stain advisor</title>
<link rel="stylesheet" href="/style.css">
</head>
<body>
<header><h1>Stain advisor</h1>
  <nav><a href="/">Advisor</a><a href="/kb">Knowledge</a></nav></header>
<main>
  <div class="card"><label for="staff">Your name</label>
    <input id="staff" placeholder="for example Dewi"></div>
  <div class="card" id="answers"></div>
  <div class="card" id="box"></div>
</main>
<script>
// The page keeps the answers; the server keeps the knowledge. Each call
// sends every answer so far, and the server replies with the next question
// or with the advice. So the server remembers nothing between calls.
let facts = {}, unknown = [], consultation = null;
const box = document.getElementById("box");

function el(tag, text, cls) {           // a new element with safe text
  const e = document.createElement(tag);
  if (text !== undefined) e.textContent = text;
  if (cls) e.className = cls;
  return e;
}

function button(text, action, cls) {
  const b = el("button", text, cls);
  b.onclick = action;
  return b;
}

async function call(path, body) {
  const r = await fetch(path, {method: "POST", body: JSON.stringify(body),
    headers: {"Content-Type": "application/json"}});
  return r.json();
}
```

Kode 10.9 (Lanjutan) *index.html: halaman penasihat, yang menyimpan jawaban-jawaban dan berbicara dengan server.*

```
function showAnswers() {
  const a = document.getElementById("answers");
  a.replaceChildren(el("h2", "Answers so far"));
  for (const [v, x] of Object.entries(facts)) a.append(el("span",
      v + " = " + x, "chip"));
  for (const v of unknown) a.append(el("span", v + " = ?", "chip"));
  a.append(el("div"), button("Start again", start, "plain"));
}

async function next() {
  showAnswers();
  const out = await call("/api/consult", {facts: facts, unknown: unknown,
      staff: document.getElementById("staff").value});
  box.replaceChildren();
  if (out.status === "question") {
    box.append(el("h2", out.question));
    for (const c of out.choices) box.append(button(c, () => {
      facts[out.variable] = c; next(); }));
    box.append(button("I do not know", () => {
      unknown.push(out.variable); next(); }, "plain"));
    const other = el("input");
    other.placeholder = "something else? type it and press Enter";
    other.onkeydown = e => { if (e.key === "Enter" && other.value.trim()) {
      facts[out.variable] = other.value.trim(); next(); } };
    box.append(other, el("p", out.why.join(" "), "muted"));
  } else if (out.status === "advice") {
    consultation = out.consultation;
    box.append(el("h2", "Advice: " + out.treatment), el("p", out.steps));
    if (out.caution) box.append(el("div", out.caution, "caution"));
    const why = el("details");
    why.append(el("summary", "Why this advice?"),
               el("pre", out.how.join("\n")));
    box.append(why, el("p", "After the wash: did it work?"),
      button("It worked", worked, "main"),
      button("It did not work", teach));
  } else if (out.status === "none") {
    consultation = out.consultation;
    box.append(el("h2", out.message),
      el("p", "Treat it as you know best. Afterwards, teach the advisor."),
      button("Teach the advisor", teach, "main"));
  } else {
    box.append(el("p", out.message, "error"));
  }
}
```

Kode 10.9 (Lanjutan) *index.html: halaman penasihat, yang menyimpan jawaban-jawaban dan berbicara dengan server.*

```
async function worked() {
  const out = await call("/api/feedback", {consultation: consultation,
      worked: true, staff: document.getElementById("staff").value});
  done(out);
}

function field(label, id, big) {
  const f = el(big ? "textarea" : "input");
  f.id = id;
  return [el("label", label), f];
}

function teach() {
  box.replaceChildren(el("h2", "Teach the advisor"),
    ...field("What removed the stain in the end?", "treatment"),
    ...field("How to do it, step by step", "steps", true),
    ...field("A caution, if any", "caution"),
    ...field("What else was different this time? One per line, " +
             "like: fabric = silk", "extra", true),
    button("Save what I learned", send, "main"));
}

async function send() {
  const value = id => document.getElementById(id).value.trim();
  const extra = value("extra").split("\n").map(s => s.trim())
      .filter(s => s);
  const out = await call("/api/feedback", {consultation: consultation,
      worked: false, treatment: value("treatment"), steps: value("steps"),
      caution: value("caution"), extra: extra,
      staff: document.getElementById("staff").value});
  done(out);
}

function done(out) {
  box.replaceChildren();
  if (out.status === "error") {
    box.append(el("p", out.message, "error"));
    return;
  }
  box.append(el("h2", out.message || "Learned: " + out.result));
  if (out.differences) box.append(el("pre", out.differences.join("\n")));
  if (out.affected && out.affected.length) box.append(el("p",
      "Look again at: " + out.affected.join(", "), "caution"));
  box.append(el("p", out.summary, "muted"),
             button("Next item", start, "main"));
}
```

Kode 10.9 (Lanjutan) *index.html: halaman penasihat, yang menyimpan jawaban-jawaban dan berbicara dengan server.*

```
function start() {
  facts = {}; unknown = []; consultation = null;
  next();
}

start();
</script>
</body>
</html>
```

Bagian HTML-nya pendek: sebuah kepala halaman dengan dua tautan, sebuah kartu untuk nama staf, sebuah kartu untuk jawaban sejauh ini (`answers`), dan sebuah kartu (`box`) tempat pertanyaan, saran, atau formulir pengajaran muncul. JavaScript melakukan sisanya.

`facts` dan `unknown` adalah ingatan halaman tentang konsultasi, dua daftar dalam pola jalankan-ulang. `el` membuat sebuah elemen dan mengisi teksnya dengan `textContent`, tidak pernah dengan `innerHTML`, sehingga nilai yang diketik staf selalu ditampilkan sebagai teks dan tidak pernah dapat dianggap sebagai potongan halaman. `call` mengirim permintaan dengan `fetch`, fungsi bawaan peramban untuk berbicara dengan server, lalu menunggu jawaban JSON-nya (`await`).

`next` adalah seluruh perulangan konsultasi. Ia menampilkan jawaban sejauh ini, mengirimnya, lalu melihat `status` jawabannya. Untuk `question` ia menampilkan satu tombol per jawaban lazim, tombol *I do not know*, sebuah kotak untuk jawaban lain, dan alasan mesin dalam teks kecil berwarna abu-abu; menekan sebuah tombol menambahkan jawabannya ke `facts` (atau variabelnya ke `unknown`) lalu memanggil `next` lagi. Untuk `advice` ia menampilkan penanganan, langkah-langkahnya, peringatan di kotak kuning, bagian terlipat *Why this advice?* berisi penjelasannya, dan dua tombol *It worked* dan *It did not work*. Untuk `none` ia mempersilakan staf menangani noda sebaik yang mereka tahu, lalu mengajari penasihat.

`teach` menampilkan formulir pengajaran: penanganan yang berhasil, langkah-langkahnya, sebuah peringatan, dan fakta-fakta lain yang berbeda kali ini, satu per baris. `send` mengirimnya sebagai umpan balik, dan `done` menampilkan apa yang terjadi: pelajaran yang dipelajari, daftar perbedaannya, dan, bila ada kasus cornerstone terdahulu yang kini akan mendapat jawaban lain, peringatan untuk memeriksanya lagi. *Next item* memulai konsultasi baru.

Kode 10.10 adalah halaman pengetahuan, yang melakukan satu panggilan, `GET /api/kb`, dan menampilkan keempat bagiannya sebagai kartu dan tabel.

Kode 10.10 `kb.html`: halaman pengetahuan, berisi angka-angka minggu ini, pohon, perubahan terakhir, dan daftar penanganan.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Stain advisor: knowledge</title>
<link rel="stylesheet" href="/style.css">
</head>
<body>
<header><h1>Stain advisor</h1>
  <nav><a href="/">Advisor</a><a href="/kb">Knowledge</a></nav></header>
<main id="main"></main>
<script>
// One call, GET /api/kb, brings everything this page shows.
function el(tag, text, cls) {
  const e = document.createElement(tag);
  if (text !== undefined) e.textContent = text;
  if (cls) e.className = cls;
  return e;
}

function table(headers, rows) {
  const t = el("table"), head = el("tr");
  for (const h of headers) head.append(el("th", h));
  t.append(head);
  for (const r of rows) {
    const tr = el("tr");
    for (const c of r) tr.append(el("td", String(c)));
    t.append(tr);
  }
  return t;
}

function card(title, ...parts) {
  const c = el("div", undefined, "card");
  c.append(el("h2", title), ...parts);
  document.getElementById("main").append(c);
}
```

Kode 10.10 (Lanjutan) *kb.html: halaman pengetahuan, berisi angka-angka minggu ini, pohon, perubahan terakhir, dan daftar penanganan.*

```
fetch("/api/kb").then(r => r.json()).then(kb => {
  const n = kb.consultations;
  card("This week", el("p", kb.summary),
    table(["Consultations", "Worked", "Taught", "Failed", "Open"],
          [[n.total, n.worked, n.taught, n.failed, n.open]]),
    el("p", "Most given advice:", "muted"),
    table(["Treatment", "Times"],
          kb.mostGiven.map(t => [t.treatment, t.times])));
  card("The knowledge, as a tree of exceptions",
    el("pre", kb.tree.join("\n")));
  card("Latest changes",
    table(["#", "What", "Node", "Option", "Who", "When"],
          kb.history.map(h => [h.seq, h.action, h.node, h.option,
                               h.author, h.time])));
  card("Treatments",
    table(["Name", "Steps", "Caution"],
          kb.treatments.map(t => [t.name, t.steps, t.caution])));
});
</script>
</body>
</html>
```

Dan **Kode 10.11** adalah lembar gaya yang dipakai bersama oleh kedua halaman. Warna-warnanya didefinisikan sekali, di bagian atas, sebagai **properti khusus** (*custom properties*: `--navy`, `--teal`, ...); bagian lain berkas memakainya menurut nama.

Kode 10.11 `style.css`: tampilan kedua halaman.

```
/* The stain advisor's look: the same file for the Java and Python servers. */
:root { --navy: #12355B; --teal: #0F7C7C; --lteal: #D8EFEF;
        --amber: #E0A100; --lamber: #F8E7B0; --red: #C0392B;
        --grey: #6B7280; --lgrey: #F3F4F6; }
body { margin: 0; font-family: system-ui, sans-serif; color: var(--navy);
       background: var(--lgrey); }
header { background: var(--navy); color: white; padding: 12px 16px;
         display: flex; justify-content: space-between; align-items: center; }
header a { color: white; margin-left: 14px; }
main { max-width: 860px; margin: 0 auto; padding: 16px; }
.card { background: white; border-radius: 10px; padding: 16px;
        margin-bottom: 14px; border: 1px solid #dde3ea; }
h1 { font-size: 20px; margin: 0; }
h2 { font-size: 18px; margin: 0 0 10px; }
button { font-size: 16px; padding: 9px 14px; margin: 4px 6px 4px 0;
         border-radius: 8px; border: 1px solid var(--teal);
         background: var(--lteal); color: var(--navy); cursor: pointer; }
button.main { background: var(--teal); color: white; }
button.plain { background: white; border-color: var(--grey); }
input, textarea { font: inherit; padding: 8px; width: 100%;
                  box-sizing: border-box; margin: 4px 0 10px;
                  border: 1px solid #c8d0da; border-radius: 6px; }
label { font-weight: 600; }
.chip { display: inline-block; background: var(--lteal); border-radius: 12px;
        padding: 3px 10px; margin: 2px 4px 2px 0; font-size: 14px; }
.caution { background: var(--lamber); border-left: 4px solid var(--amber);
           padding: 8px 10px; margin: 10px 0; }
.error { color: var(--red); }
pre { background: var(--lgrey); padding: 10px; white-space: pre-wrap;
      font-size: 13px; }
table { border-collapse: collapse; width: 100%; font-size: 14px; }
td, th { border-bottom: 1px solid #e5e7eb; padding: 5px; text-align: left;
         vertical-align: top; }
.muted { color: var(--grey); font-size: 14px; }
```

10.8 Menjalankan Penasihat Laundry

Jalankan salah satu server. Dalam Java:

```
cd Chapter10/java
javac -d out -sourcepath ../../vcirs-core/java/src *.java
java -cp "out:lib/sqlite-jdbc-3.53.2.0.jar" LaundryServer
```

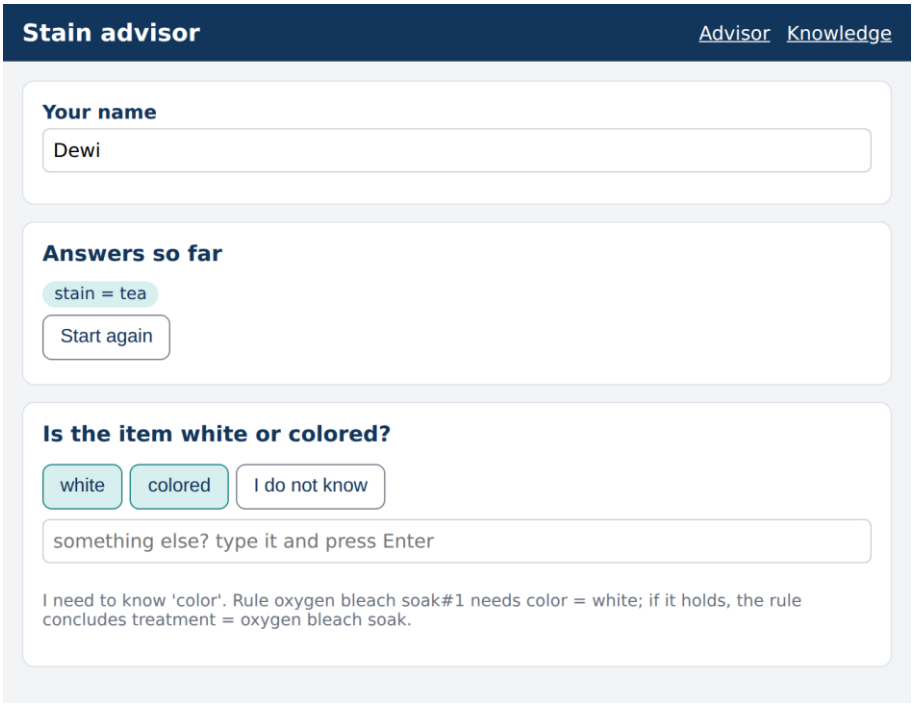
atau dalam Python:

```
cd Chapter10/python
python laundry_app.py
```

Keduanya mencetak:

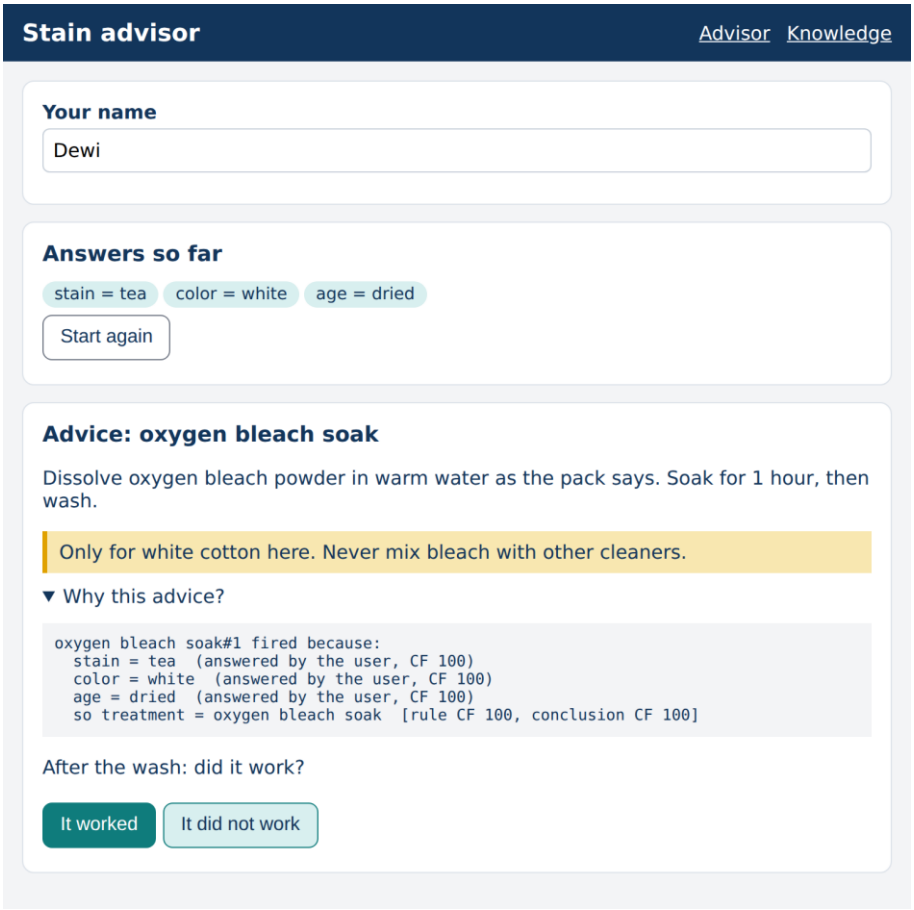
```
Ready: Stain advisor: 5 variables, 12 nodes, 12 rules
Open http://localhost:4100 (Ctrl+C stops)
```

(Flask menambahkan beberapa baris miliknya sendiri.) Saat pertama dijalankan, server membuat `data/laundry.db` dari kasus benih; setiap kali berikutnya ia memuatnya. Bukalah `http://localhost:4100` di peramban. **Gambar 10.4** memperlihatkan halaman setelah Dewi mengetikkan namanya dan memilih *tea*: jawabannya ada di daftar, dan pertanyaan berikutnya tentang warna, dengan alasan mesin di bawahnya.



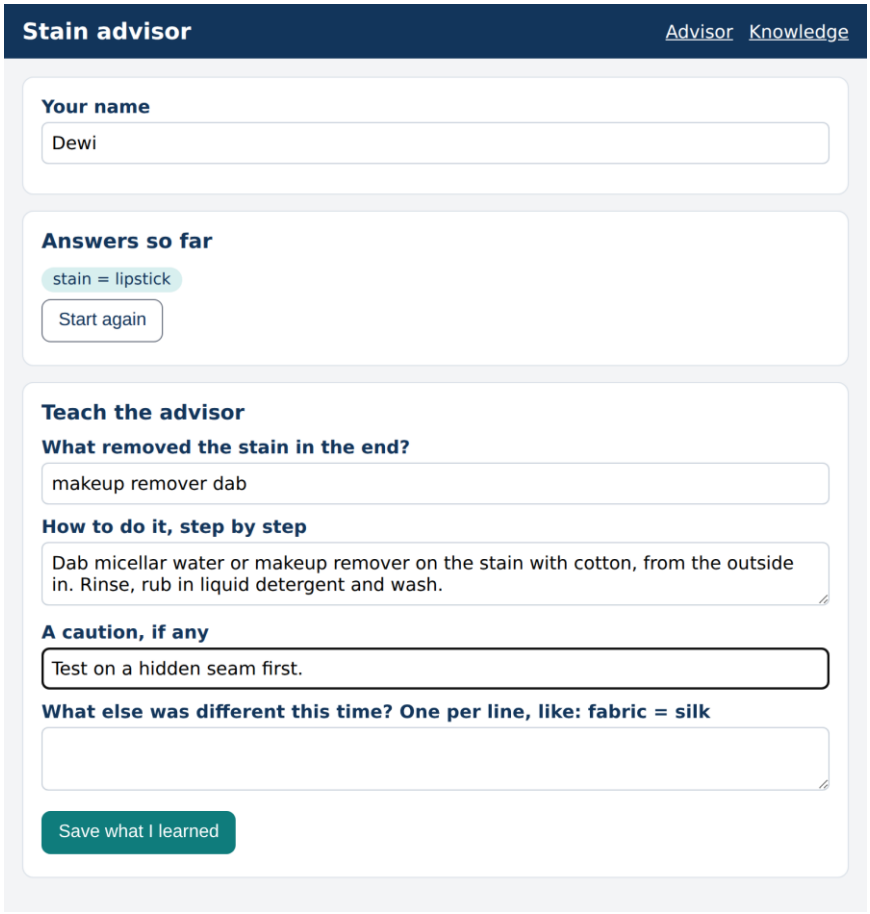
Gambar 10.4 Penasihat mengajukan pertanyaan keduanya. Jawaban sejauh ini tampil sebagai label kecil; alasan pertanyaannya berasal dari mesin.

Ia menjawab *white*, lalu *dried*. **Gambar 10.5** memperlihatkan sarannya, dengan peringatan di kotak kuning dan penjelasan yang dibuka.



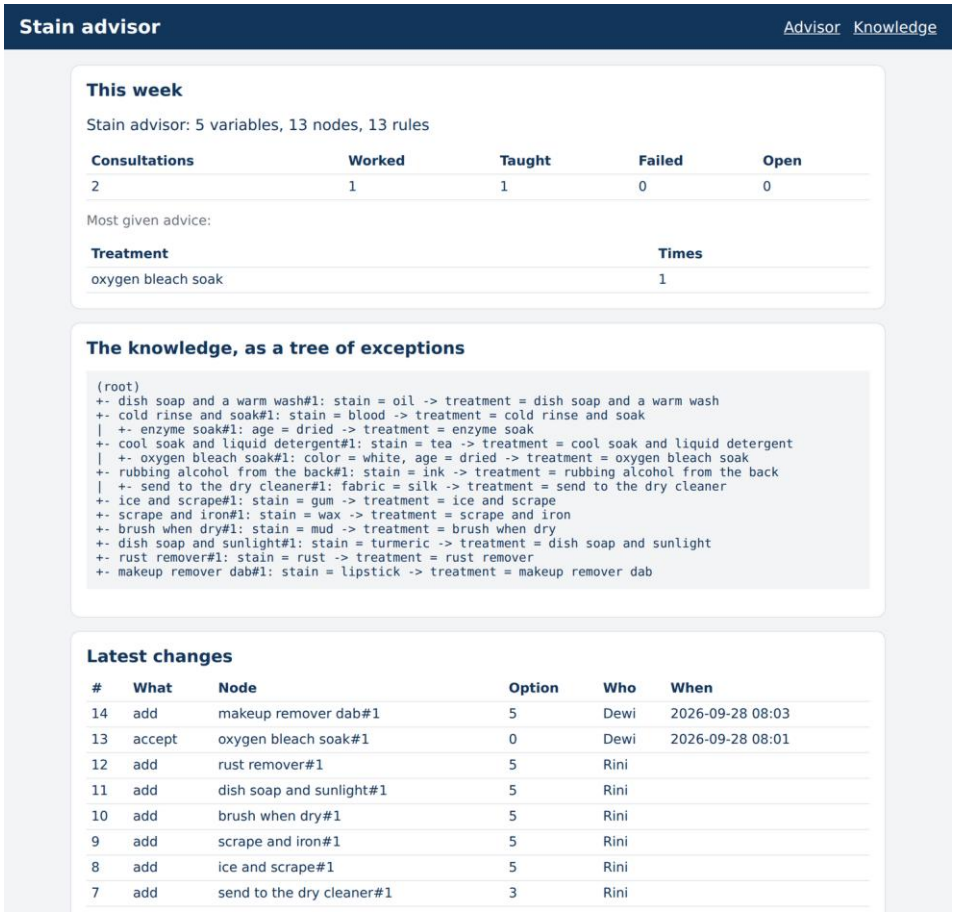
Gambar 10.5 Saran untuk noda teh yang sudah kering pada barang putih: langkah-langkah, peringatan, dan, dalam keadaan dibuka, penjelasan node yang menyala.

Kemejanya keluar dalam keadaan bersih, Dewi menekan *It worked*, lalu memulai barang berikutnya. Kali ini noda lipstik; ia mengetik *lipstick* di kotak untuk jawaban lain, penasihat tidak punya saran, dan setelah menanganinya ia mengisi formulir pengajaran pada **Gambar 10.6**.



Gambar 10.6 Mengajari penasihat penanganan noda yang belum pernah dilihatnya. Kotak fakta tambahan dibiarkan kosong karena nodanya sendirilah yang baru.

Gambar 10.7 adalah halaman pengetahuan setelah dua konsultasi ini: angka-angka hari itu, pohon node dan pengecualian dengan node lipstik yang baru di bagian bawah, dan perubahan terakhir beserta siapa yang membuatnya dan kapan. Pelajaran benih tidak punya waktu karena diajarkan sebelum laundry buka pada hari Senin.



Gambar 10.7 Halaman pengetahuan: angka-angka minggu ini, saran yang paling sering diberikan, pengetahuan sebagai pohon pengecualian, dan perubahan terakhir.

10.8.1 Berbicara dengan server tanpa peramban

Setiap panggilan penasihat juga dapat dilakukan dengan **curl**, program kecil untuk mengirim permintaan web yang sudah terpasang di Windows 10 ke atas, macOS, dan Linux. Selagi server berjalan, di terminal kedua:

```
curl -X POST http://localhost:4100/api/consult -d "{\"facts\": {\"stain\": \"blood\"}}"
```

mencetak pertanyaan tentang umur noda:

```
{
  "status": "question",
  "variable": "age",
  "question": "Is the stain fresh or dried?",
  "choices": [
    "fresh",
    "dried"
  ],
  "why": [
    "I need to know 'age'.",
    "Rule enzyme soak#1 needs age = dried;",
    "if it holds, the rule concludes treatment = enzyme soak."
  ]
}
```

(Garis miring terbalik sebelum tanda kutip bagian dalam dibutuhkan oleh Command Prompt Windows; di macOS dan Linux Anda boleh menulis `-d '{"facts": {"stain": "blood"}}'`.) `curl http://localhost:4100/api/kb` mencetak ikhtisarnya. Memutar ulang kelima belas permintaan `tour.txt` dengan cara ini terhadap server Java dan terhadap server Python, masing-masing mulai

dari basis data kosong dengan jam pura-pura yang sama, menghasilkan teks Subbab 10.5 yang persis sama dari keduanya, dan kedua berkas `laundry.db` memiliki baris-baris yang sama di setiap tabel.

10.8.2 Ponsel di meja depan

Untuk memakai penasihat dari ponsel, jalankan server dengan `--host 0.0.0.0`, yang berarti "terima juga sambungan dari jaringan":

```
java -cp "out:lib/sqlite-jdbc-3.53.2.0.jar" LaundryServer --host 0.0.0.0
python laundry_app.py --host 0.0.0.0
```

Lalu cari alamat laptop di Wi-Fi toko — `ipconfig` di Command Prompt Windows (baris *IPv4 Address*), *System Settings* → *Wi-Fi* → *Details* di macOS, `hostname -I` di Linux — misalnya `192.168.1.23`, dan buka `http://192.168.1.23:4100` di ponsel, yang harus tersambung ke Wi-Fi yang sama. Pertama kali, Windows bertanya apakah Java atau Python boleh menerima sambungan di jaringan privat; izinkan untuk jaringan privat saja. Penasihat tidak memiliki kata sandi, jadi pakailah hanya di Wi-Fi milik toko sendiri, jangan pernah di Wi-Fi publik; Soal Latihan 10.8 menambahkan PIN staf yang sederhana.

10.8.3 Pengujian

`AdvisorTest.java` dan `test_advisor.py` memeriksa Advisor tanpa server, masing-masing pada basis data baru di folder sementara: benihnya memiliki dua belas node dan pertama-tama menanyakan nodanya; sebuah pengecualian menyala; jawaban yang tidak diketahui melewatinya; noda baru dapat diajarkan, dan pelajarannya masih ada ketika Advisor dibuka lagi pada basis data yang sama; dan konsultasi yang sudah ditutup tidak dapat ditutup dua kali. Pengujian Java diakhiri dengan:

```
ok: the seed has 12 nodes
ok: the first question is the stain
ok: an exception fires
ok: an unknown answer skips the exception
ok: a new stain is taught
ok: the lesson is in the database
ok: a closed consultation stays closed
all tests passed
```

10.9 Membuatnya Lebih Baik

Pertanyaan tentang bahan untuk noda darah. Rendaman enzim membawa peringatan *not for silk or wool*, tetapi node darah kering tidak pernah menanyakan bahannya. Peringatan itu ada untuk menangkap kasus yang belum ditangani pengetahuannya. Perbaikan yang tepat adalah perbaikan yang biasa: pertama kali barang sutra dengan darah kering datang, staf menekan *It did not work* (atau, lebih baik, tidak mengikuti saran itu dan langsung mengajar) dengan `fabric = silk` dan penanganan *send to the dry cleaner*, dan sebuah pengecualian muncul di bawah *enzyme soak*. Peringatan yang terus-menerus dibutuhkan adalah tanda bahwa ada pengecualian yang belum ada.

"Saya tidak tahu" yang lebih aman. Pada konsultasi terakhir dalam tur, noda tinta pada bahan yang tidak diketahui mendapat penanganan alkohol yang umum. Untuk kebanyakan bahan hal itu tepat, dan peringatannya meminta pengujian di jahitan tersembunyi lebih dahulu. Laundry yang lebih berhati-hati mungkin lebih suka bahan yang tidak diketahui diperlakukan seperti bahan halus. Basis pengetahuan dapat mengatakannya dengan pengecualian yang dibangun dari *negasi*:

di bawah node tinta, sebuah node dengan kondisi `fabric <> cotton`... tetapi bahan yang tidak diketahui juga tidak memenuhinya, karena mesin tidak pernah membiarkan fakta yang tidak diketahui membuat sebuah kondisi benar. Penyelesaian yang jujur ada di halaman: untuk variabel yang penting bagi keselamatan, hapus tombol *I do not know*, atau tampilkan "periksa label perawatannya" sebagai gantinya. Soal Latihan 10.5 meminta Anda melakukannya dengan sebuah daftar kecil di halaman.

Label perawatan sebagai variabel. Bu Rini ingin penasihat menanyakan label perawatan hanya ketika hal itu penting. Ajarkan sebagai pengetahuan, bukan sebagai kode: dengan variabel label dideklarasikan (`label`, *What does the care label say?*, `washable`/`dry clean only`), ajarkan barang *dry clean only* yang pertama sebagai pengecualian di mana pun ia muncul. Pertanyaan-pertanyaannya lalu hanya muncul di tempat sebuah node membutuhkannya — dan itulah seluruh janji sistem pakar yang bertanya.

Foto. Foto noda sebelum dan sesudah akan membantu Bu Rini menilai pelajaran Joko pada Kamis malam. Tabel Consultation akan mendapat kolom berisi nama berkas, halaman sebuah `<input type="file" accept="image/*" capture>` yang membuka kamera ponsel, dan server panggilan keempat yang menyimpan berkasnya di sebuah folder. Tidak satu pun dari hal ini menyentuh basis pengetahuan.

Angka yang lebih baik. Halaman pengetahuan menghitung hasil; dengan satu `GROUP BY` lagi ia dapat menghitungnya per penanganan, dan porsi *taught* di antara semua konsultasi dalam seminggu menunjukkan apakah penasihat masih banyak belajar atau sudah menjadi rekan kerja yang dapat diandalkan, seperti yang disarankan Subbab 3.9 untuk dokter Wi-Fi. Bab 7 melangkah jauh lebih jauh dengan statistik pemakaian VCIRS sendiri.

Kata-kata berbahasa Indonesia di halaman. Pertanyaan-pertanyaan bab ini berbahasa Inggris karena variabelnya berbahasa Inggris; tidak ada bagian mesin yang mengharuskannya. Laundry di Surabaya akan mendeklarasikan `noda` dengan pertanyaan *Nodanya apa?* dan jawaban `minyak`, `darah`, `teh`, dan seterusnya, dan tombol serta label halaman diterjemahkan di `index.html`. Pengetahuan, basis data, dan server tetap persis seperti sekarang.

Cadangan. Seluruh pengetahuan laundry adalah satu berkas, `laundry.db`. Salinlah seminggu sekali ke *flash disk* atau penyimpanan awan, dengan tanggal di namanya. Salin ketika server berhenti, atau dengan perintah cadangan milik SQLite sendiri (`sqlite3 laundry.db ".backup laundry-2026-10-04.db"`), yang menghasilkan salinan yang konsisten bahkan ketika server sedang berjalan.

> **JEBAKAN:** Jangan membuka `laundry.db` di peramban basis data lalu menyunting tabel-tabel pengetahuan dengan tangan selagi server berjalan. Server menyimpan basis pengetahuan di memori dan menulis seluruhnya kembali setelah pelajaran berikutnya, sehingga suntingan Anda akan tergantikan tanpa pemberitahuan — dan suntingan tangan yang merusak struktur sebuah node tidak akan tertangkap oleh pemeriksaan mesin. Ubahlah pengetahuan melalui penasihat, di mana setiap perubahan adalah kasus dengan cornerstone dan nama.

10.10 Memakainya Sungguh-Sungguh

Memasangnya di toko. Laptop tua sudah cukup: server membutuhkan sangat sedikit memori, dan SQLite tidak membutuhkan memori sendiri. Pasang Java atau Python (Bab 1), salin folder `Code_new`, dan jalankan server dengan `--host 0.0.0.0` ketika toko buka. Di Windows, pintasan ke berkas `.bat` satu baris, dan di macOS atau Linux sebuah skrip kecil, memungkinkan siapa pun

menjalankannya dengan klik ganda. Taruh *bookmark* alamatnya di layar utama ponsel meja depan, dan penasihat berperilaku seperti sebuah aplikasi.

Mengisi benih pengetahuan Anda sendiri. Mulailah dengan noda-noda yang datang setiap minggu; sepuluh atau dua belas kasus, seperti pada **Tabel 10.1**, butuh kurang dari satu jam untuk ditulis bersama langkah-langkahnya. Biarkan sisanya diajarkan di meja depan. Tulis langkah-langkahnya seperti Anda mengatakannya kepada karyawan baru, dalam kalimat pendek, dan masukkan setiap peringatan ke kolom peringatan: untuk itulah kotak kuning itu ada.

Setengah jam setiap minggu. Seminggu sekali, bacalah halaman pengetahuan bersama staf. Lihat perubahan terakhir: setiap pelajaran punya nama dan waktu, dan pelajaran yang tidak diingat siapa pun layak ditanyakan. Lihat konsultasi yang masih terbuka: saran yang diberikan tetapi tidak pernah dikonfirmasi berarti staf lupa menekan tombol, atau sarannya meragukan. Dan lihat penanganan yang paling sering diberikan; bila satu penanganan diberikan untuk banyak noda yang berbeda, node-nya mungkin terlalu umum.

Toko lain yang bentuknya sama. Tidak ada bagian Advisor yang khusus tentang noda kecuali berkas benih, berkas penanganan, dan keempat pertanyaannya. Jasa cuci sepatu (bahan, noda, warna), penjahit (bahan, masalah, jenis pakaian), cuci mobil (cat, bekas, umur), daftar periksa kebersihan kantin sekolah — masing-masing adalah aplikasi yang sama dengan berkas benih yang lain. Bab 14 membangun alat yang memungkinkan siapa pun menyiapkan penasihat seperti ini tanpa menyentuh kode.

Hanya fakta yang dibutuhkan. Tabel Consultation menyimpan fakta-fakta sebuah noda dan nama depan staf, tidak lebih: tidak ada nama pelanggan, nomor telepon, atau foto orang. Di Indonesia, Undang-Undang Pelindungan Data Pribadi (UU No. 27/2022) mensyaratkan agar data pribadi hanya dikumpulkan untuk tujuan yang jelas dan hanya disimpan selama dibutuhkan [13]; cara paling sederhana untuk mematuhinya adalah tidak mengumpulkan apa yang tidak dibutuhkan penasihat. Bila laundry ingin mengaitkan konsultasi dengan pesanan, simpanlah nomor pesanannya, yang sudah tercatat di buku pesanan.

Kapan memanggil sang pakar. Penasihat memberikan cara-cara penanganan Bu Rini; ia tidak membuat Bu Rini tidak diperlukan. Sebagian barang layak dilihat Bu Rini sendiri sebelum ada yang menanganinya: apa pun yang mahal, apa pun yang berlabel *dry clean only*, apa pun yang tua atau bahannya tidak diketahui, dan noda apa pun yang sudah telanjur masuk mesin pengering. Aturan yang baik, dan layak dimasukkan ke kasus benih sebagai pengetahuan, adalah bahwa untuk barang-barang seperti ini sarannya adalah menyisihkannya untuk Bu Rini.

10.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Menyimpan percakapan di server. Server yang menyimpan "konsultasi yang sedang berjalan" di sebuah variabel akan rusak begitu dua orang berkonsultasi bersamaan, atau satu orang memuat ulang halaman. *Hindari:* simpan jawaban-jawaban di halaman dan jalankan ulang mesin dengan semuanya, seperti pada Subbab 10.2.4.

Menyusun SQL dengan menempel teks. `"... WHERE Name = '" + name + "'"` rusak pada penanganan pertama yang mengandung apostrof dan membiarkan siapa pun yang dapat mengetik di formulir mengubah SQL-nya. *Hindari:* selalu pakai penanda tempat `?`, seperti yang dilakukan kedua Advisor.

Lupa *driver*-nya. `No suitable driver found for jdbc:sqlite:...` berarti berkas `.jar` JDBC SQLite tidak ada di *class path*. *Hindari:* jalankan dengan `-cp "out:lib/sqlite-jdbc-3.53.2.0.jar"` (`;` di Windows).

Dua server pada satu port. *Address already in use* berarti program lain — sering kali server bahasa yang satunya, masih berjalan di terminal lain — memakai port 4100. *Hindari:* hentikan yang pertama dengan Ctrl+C, atau jalankan yang kedua dengan `--port 4101`.

Mengisi `innerHTML` dengan teks ketikan. Nama penanganan yang diketik dengan kurung sudut akan dianggap bagian dari halaman. *Hindari:* isi teks dengan `textContent`, seperti yang dilakukan `el`.

Menekan *It worked* sebelum dicuci. Penasihat mencatat kasus terverifikasi; bila nodanya masih ada setelah dicuci, basis pengetahuan telah mengonfirmasi sebuah kesalahan. *Hindari:* beri umpan balik hanya setelah barang keluar dari mesin; konsultasinya menunggu dengan status `open` sampai saat itu.

Mengajarkan fakta yang sama. Menambahkan `stain = oil` sebagai fakta "yang berbeda" pada noda minyak di sutra tidak mengajarkan apa pun; mesin menolak ketika tidak ada yang membedakan kasusnya dari kasus cornerstone. *Hindari:* ketikkan fakta yang benar-benar berbeda, misalnya `fabric = silk`.

Menyunting basis data dengan tangan selagi server berjalan. Lihat jebakan di Subbab 10.9. *Hindari:* ajarkan melalui halaman; untuk koreksi besar, hentikan server lebih dahulu.

Membuka server ke jaringan publik. Dengan `--host 0.0.0.0` di kafe, siapa pun di Wi-Fi kafe dapat memakai dan mengajari penasihat. *Hindari:* pakai `--host 0.0.0.0` hanya di Wi-Fi milik toko, dengan kata sandi Wi-Fi yang kuat.

Memakai kata yang berbeda untuk jawaban yang sama. `Silk`, `silk`, dan `sutra` adalah tiga nilai yang berbeda bagi mesin. *Hindari:* pilih dari tombol-tombolnya; tombol-tombol itu menampilkan setiap nilai yang sudah dikenal basis pengetahuan.

10.12 Ringkasan Bab

- Basis pengetahuan yang dipakai beberapa orang sekaligus sebaiknya berada di **server** dengan **basis data**; peramban di komputer dan ponsel toko adalah kliennya, dan mereka berbicara dengan server dalam **JSON lewat HTTP** (**Gambar 10.3**).
- **SQLite** menyimpan seluruh basis data dalam satu berkas, tidak perlu dipasang, dan merupakan mesin basis data yang paling luas terpasang; Python menjangkaunya dengan `sqlite3` dan Java dengan **JDBC** dan *driver* SQLite.
- `SqliteStore` milik vcirs-core menyimpan basis pengetahuan di tabel-tabel Lampiran E tesis — Variable, Node, NodeStructure, Conclusion, Rule, RuleStructure — ditambah History dan Meta, di samping tabel milik aplikasi sendiri; kueri **SQL** menjawab pertanyaan tentang keduanya (**Gambar 10.1**, **Kode 10.1**, **Kode 10.2**).
- **Pola jalankan-ulang** mengajukan pertanyaan sistem pakar lewat web: halaman menyimpan jawaban-jawaban, server menjalankan ulang mesin dengan semuanya dan mengembalikan pertanyaan terbuka pertama, dengan menghentikan mesin melalui pengecualian yang dilempar dari asker (**Gambar 10.2**).
- **Advisor** memuat seluruh logika tanpa kode web apa pun: `consult`, `feedback` (pembelajaran Ripple Down Rules dengan daftar perbedaan dan cornerstone yang terdampak), dan `overview` (**Tabel 10.2**, **Kode 10.3**, **Kode 10.4**); memutar ulang lima belas permintaan padanya

memperlihatkan kasus terverifikasi, sebuah pengecualian, sebuah node tingkat atas baru, dan sebuah "saya tidak tahu" (**Kode 10.5**, **Kode 10.6**).

- Dua server tipis — `HttpServer` bawaan JDK dan Flask — menyajikan halaman yang sama dan menjawab setiap panggilan dengan bita yang sama (**Kode 10.7**, **Kode 10.8**); halaman-halamannya menyimpan konsultasi dan menampilkan pertanyaan, saran, peringatan, penjelasan, dan formulir pengajaran (**Kode 10.9**, **Kode 10.10**, **Kode 10.11**, **Gambar 10.4**, **Gambar 10.5**, **Gambar 10.6**, **Gambar 10.7**).
- Keselamatan tinggal di dalam pengetahuan: peringatan di samping setiap penanganan, *dry cleaner* untuk barang yang halus, dan aturan untuk tidak pernah mencampur produk pembersih (**Tabel 10.1**).

10.13 Soal Latihan

10.1 Dengan tangan, memakai **Tabel 10.1**, daftarlah pertanyaan-pertanyaan yang diajukan penasihat, berurutan, dan saran yang diberikannya untuk: (a) darah segar pada kemeja katun; (b) teh kering pada kemeja berwarna; (c) tinta pada syal sutra; (d) tetesan lilin.

10.2 Tulislah kueri SQL yang mendaftar setiap node pengecualian beserta induknya dan jumlah kondisinya sendiri. Jalankan pada `tour.db` (dibuat oleh `db_tour.py`) dari sebuah program Python kecil, dan jelaskan hasilnya.

10.3 Dalam pola jalankan-ulang, server menjalankan mesin sekali untuk setiap jawaban. Untuk konsultasi yang berakhir setelah q pertanyaan, berapa kali mesin dijalankan seluruhnya? Mengapa jumlah itu tidak berarti bagi laundry, dan kapan ia mulai berarti?

10.4 Tabel `Consultation` mencatat node yang menjawab. Tulislah kueri yang menghitung, untuk setiap node, berapa konsultasi yang dijawabnya dan berapa di antaranya yang `worked`. Node mana yang akan Anda tinjau lebih dahulu, dan mengapa?

10.5 Ubahlah `index.html` sehingga untuk variabel-variabel dalam daftar `NO_UNKNOWN = ["fabric"]` halaman menampilkan, alih-alih tombol *I do not know*, teks *Check the care label first* dan tidak ada cara untuk melewati pertanyaannya. Mengapa perubahan ini dilakukan di halaman dan bukan di basis pengetahuan?

10.6 Seorang pelanggan membawa kemeja katun putih dengan darah kering. Joko mengikuti rendaman enzim, dan masih tersisa bekas cokelat samar; Bu Rini menyelesaikannya dengan rendaman pemutih oksigen. Bagaimana Joko seharusnya mengajarkan hal ini, dan di mana node barunya akan berada? Tuliskan permintaan umpan baliknya sebagai JSON.

10.7 Server menjawab galat dengan kode HTTP 400. Kirimkan, dengan curl, permintaan *consult* yang badannya bukan JSON, lalu permintaan *feedback* untuk konsultasi 999. Apa jawaban setiap server? Mengapa berguna bahwa keduanya menjawab dalam JSON bahkan ketika ada yang salah?

10.8 Tambahkan PIN staf: server membaca PIN dari opsi `--pin 2468`, dan setiap POST harus membawa `"pin": "2468"` di badannya, atau mendapat galat. Baris mana di `LaundryServer.java` dan `laundry_app.py` yang berubah, dan di mana halaman menyimpan PIN-nya? Mengapa ini cukup untuk Wi-Fi toko tetapi tidak untuk internet?

10.9 Hentikan server Python, jalankan server Java pada `laundry.db` yang sama, lalu lanjutkan mengajar. Mengapa ini berhasil? Apa yang akan rusak bila kedua server berjalan pada saat yang sama pada berkas yang sama, dan bagaimana SQLite melindungi berkasnya sendiri?

10.14 Penyelesaian Soal Latihan

Penyelesaian 10.1. (a) *What is the stain?* → blood; node darah menyala dan pengecualiannya membutuhkan umur noda: *Is the stain fresh or dried?* → fresh; pengecualiannya gagal, sehingga sarannya *cold rinse and soak*. Bahannya tidak pernah ditanyakan. (b) *stain* → tea; node teh menyala; pengecualiannya membutuhkan `color = white` lebih dahulu: *Is the item white or colored?* → colored; pengecualiannya langsung gagal tanpa menanyakan umur, dan sarannya *cool soak and liquid detergent*. (c) *stain* → ink; pengecualiannya menanyakan bahan → silk; saran *send to the dry cleaner*. (d) *stain* → wax; *scrape and iron*, hanya satu pertanyaan.

Penyelesaian 10.2. Program penyelesaiannya, `exceptions.py`, adalah:

```
"""Exercise 10.2: every exception node, its parent and its own conditions.

    python exceptions.py      (after python db_tour.py has made tour.db)
"""

import pathlib
import sqlite3
import sys

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import report

DB = pathlib.Path(__file__).resolve().parents[1] / "data" / "tour.db"
con = sqlite3.connect(DB)
cur = con.execute(
    "SELECT n.NodeID, n.ParentNodeID, COUNT(s.VariableID) AS Own "
    "FROM Node n JOIN NodeStructure s ON s.NodeID = n.NodeID "
    "WHERE n.ParentNodeID IS NOT NULL "
    "GROUP BY n.NodeID, n.ParentNodeID ORDER BY n.Seq")
for line in report.table([d[0] for d in cur.description], cur.fetchall()):
    print(line)
con.close()
```

Dijalankan setelah `db_tour.py`, program ini mencetak:

```
NodeID                    ParentNodeID                    Own
------------------------  ------------------------------  ---
enzyme soak#1             cold rinse and soak#1           1
oxygen bleach soak#1      cool soak and liquid detergent#1  2
send to the dry cleaner#1  rubbing alcohol from the back#1  1
```

Ketiga pengecualian dalam pengetahuan benih masing-masing mengoreksi satu node tingkat atas. `oxygen bleach soak#1` memiliki dua kondisi sendiri karena kasus teh itu membutuhkan dua fakta (putih, kering) untuk membedakannya dari kasus teh biasa; dua yang lain hanya membutuhkan satu. `COUNT(s.VariableID)` hanya menghitung baris-baris milik node itu sendiri di NodeStructure; kondisi induknya berada di baris-baris milik induk.

Penyelesaian 10.3. Setiap jawaban menambahkan satu fakta dan memicu satu kali jalan lagi, dan jalan terakhir mengembalikan saran, sehingga konsultasi dengan q pertanyaan menjalankan mesin $q + 1$ kali, dan jalan ke-k memeriksa ulang paling banyak kondisi-kondisi yang diperiksa jalan ke-$(k - 1)$ ditambah satu. Dengan empat variabel dan empat belas node, satu kali jalan hanya butuh beberapa mikrodetik; bahkan seribu node akan jauh di bawah satu milidetik, jauh lebih singkat daripada waktu yang dibutuhkan seseorang untuk menekan tombol. Jumlah itu baru berarti bagi basis pengetahuan yang kondisinya memanggil layanan luar yang lambat (daftar harga di internet, sebuah sensor), dan dalam hal itu server akan menyimpan fakta-fakta yang dicarinya, bukan percakapannya.

Penyelesaian 10.4. Satu kueri sudah cukup:

```
SELECT NodeID, COUNT(*) AS Answered,
       SUM(CASE WHEN Outcome = 'worked' THEN 1 ELSE 0 END) AS Worked
FROM Consultation WHERE NodeID <> ''
GROUP BY NodeID ORDER BY Answered DESC, NodeID
```

`SUM(CASE ... END)` menghitung baris-baris yang memenuhi sebuah syarat di dalam setiap kelompok. Tinjaulah lebih dahulu node yang sering menjawab dengan porsi *worked* yang rendah: ia sering dipakai dan sering gagal, sehingga kemungkinan ada pengecualian yang belum ada — seperti `dish soap and a warm wash#1` sebelum pelajaran sutra dari Joko. Node yang jarang menjawab dan tidak pernah gagal dapat menunggu.

Penyelesaian 10.5. Di bagian atas skrip tambahkan `const NO_UNKNOWN = ["fabric"];`, dan di `next`, ganti baris yang menambahkan tombol *I do not know* dengan: `if (NO_UNKNOWN.includes(out.variable)) box.append(el("p", "Check the care label first.", "caution")); else box.append(button("I do not know", ...));`. Perubahan ini tempatnya di halaman karena basis pengetahuan tidak tahu bahwa sebuah jawaban tidak ada; ia hanya melihat fakta-fakta yang diberikan kepadanya, dan fakta yang tidak diketahui membuat sebuah kondisi *tidak benar*, tidak pernah *berbahaya*. Menolak melanjutkan tanpa jawaban adalah keputusan tentang cara aplikasi dipakai di meja depan, dan halaman adalah tempat aturan-aturan meja depan tinggal.

Penyelesaian 10.6. Sarannya adalah *enzyme soak* dari `enzyme soak#1`, dan saran itu tidak menuntaskan pekerjaannya; penanganan yang benar untuk kasus ini adalah rendaman pemutih oksigen, dan perbedaannya dari cornerstone `enzyme soak#1` (darah, kering) adalah bahwa barangnya putih. Joko mengajarkan:

```
{"consultation": 7, "worked": false, "treatment": "oxygen bleach soak",
 "extra": ["color = white"], "staff": "Joko"}
```

(dengan nomor konsultasi yang benar). Daftar perbedaannya memuat `color = white` di bagian B, dan node baru `oxygen bleach soak#2` menjadi pengecualian `enzyme soak#1`: ia hanya akan ditanyakan untuk darah yang sudah kering. Langkah-langkah penanganannya sudah ada di tabel Treatment, sehingga Joko tidak perlu mengetiknya; `INSERT OR IGNORE` mempertahankan yang sudah ada.

Penyelesaian 10.7. `curl -X POST http://localhost:4100/api/consult -d "hello"` mengembalikan kode 400 dengan `{"status": "error", "message": ...}`; di Python pesannya *the request must be a JSON object*, di Java pesan dari pengurai JSON. `curl -X POST http://localhost:4100/api/feedback -d "{\"consultation\": 999, \"worked\": true}"` mengembalikan 400 dengan *no open consultation 999* dari keduanya. Galat dalam JSON memungkinkan halaman menampilkan pesan yang dapat dibaca (`out.message`) alih-alih gagal di tengah `r.json()`, dan memungkinkan pengguna curl serta pengujian memeriksa kolom status dengan cara yang sama untuk keberhasilan maupun kegagalan.

Penyelesaian 10.8. Di Java, tambahkan `case "--pin" -> pin = args[i + 1];` dengan medan statis `pin`, dan di awal cabang POST dalam `api`: `if (pin != null && !pin.equals(Advisor.text(body, "pin"))) out = Map.of("status", "error", "message", "wrong PIN");` (dengan mengurai badan permintaan sebelum cabangnya). Di Python, `p.add_argument("--pin")`, dan di kedua fungsi POST `if args.pin and body.get("pin") != args.pin: return reply({"status": "error", "message": "wrong PIN"})`. Halaman meminta PIN sekali, menyimpannya dalam variabel di samping `facts`, dan menambahkannya ke setiap `call`. Di Wi-Fi toko ini menghalangi pelanggan yang penasaran dan tahu alamatnya; di internet ini tidak cukup, karena PIN berjalan sebagai teks polos lewat HTTP dan dapat dibaca

siapa pun di sepanjang jalan, dan PIN empat angka dapat ditebak program dalam hitungan detik. Internet membutuhkan HTTPS, akun sungguhan, dan batas jumlah percobaan (Bab 15).

Penyelesaian 10.9. Ini berhasil karena kedua server membaca dan menulis tabel yang sama dengan makna yang sama: `SqliteStore` di Java dan di Python adalah dua belahan dari satu rancangan, dan pemeriksaan silang di Subbab 10.8.1 memperlihatkan bahwa keduanya menulis baris-baris yang identik. Bila keduanya berjalan bersamaan, masing-masing memegang salinan basis pengetahuannya sendiri di memori, dan masing-masing menulis seluruh salinannya setelah sebuah pelajaran, sehingga pelajaran yang diajarkan lewat satu server akan tertimpa oleh penyimpanan berikutnya dari server yang lain. SQLite sendiri tetap aman — ia mengunci berkasnya selama setiap transaksi, sehingga berkas tidak pernah tertulis setengah — tetapi ia tidak dapat tahu bahwa dua program memegang dua versi berbeda dari pengetahuan yang sama. Satu server untuk satu basis data: itulah aturannya.

Daftar Pustaka

[1] SQLite Consortium, “Recent SQLite news,” 2026. [Online]. Available: https://sqlite.org/news.html

[2] SQLite Consortium, “Most widely deployed and used database engine,” 2026. [Online]. Available: https://sqlite.org/mostdeployed.html

[3] Xerial, “SQLite JDBC driver, release 3.53.2.0,” Jun. 2026. [Online]. Available: https://github.com/xerial/sqlite-jdbc/releases

[4] M. M. I. Subakti, “A Variable-Centered Intelligent Rule System,” M.Sc.Eng. thesis, Dept. of Computer Science and Information Engineering, Taipei, Taiwan, Jun. 2005.

[5] I. Subakti, “A variable-centered intelligent rule system,” in *Proc. 1st Annual Int. Conf. Information and Communication Technology Seminar (ICTS 2005)*, vol. 1, no. 1, Surabaya, Indonesia, Aug. 2005, pp. 167–174.

[6] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, Univ. of California, Irvine, CA, USA, 2000.

[7] American Cleaning Institute, “Stain removal guide,” 2026. [Online]. Available: https://www.cleaninginstitute.org/cleaning-tips/clothes/stain-removal-guide

[8] Cornell Cooperative Extension of Tompkins County, “Removing stains at home.” [Online]. Available: https://ccetompkins.org/resources/stain-removal-guide

[9] I. Subakti, “Some revisions in VCIRS and cases reconstructing perspectives,” in *Proc. Information and Communication Technology Seminar (ICTS 2006)*, Surabaya, Indonesia, 2006.

[10] P. Compton and B. H. Kang, *Ripple-Down Rules: The Alternative to Machine Learning*. Boca Raton, FL, USA: CRC Press, 2021.

[11] Oracle, “HttpServer (Java SE 25 & JDK 25), module jdk.httpserver,” 2025. [Online]. Available: https://docs.oracle.com/en/java/javase/25/docs/api/jdk.httpserver/com/sun/net/httpserver/HttpServer.html

[12] Pallets Projects, “Flask 3.1.3,” Python Package Index, Feb. 2026. [Online]. Available: https://pypi.org/project/Flask/

[13] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*. Jakarta, Indonesia, 2022.

Bab 11

Basis Pengetahuan Triase Klinik untuk Tim: Banyak Pakar, Versi, dan Audit

Setelah menyelesaikan bab ini Anda akan mampu menjelaskan apa yang salah ketika beberapa pakar mengubah satu basis pengetahuan tanpa aturan, dan membangun aturan-aturan yang mencegahnya: setiap perubahan adalah sebuah **usulan** (*proposal*) dengan alasan; usulan itu dicoba pada sebuah salinan dan **diuji** terhadap setiap kasus cornerstone dan setiap *kasus emas* (*golden case*) yang telah disepakati tim; **pakar kedua** meninjaunya; ia menjadi sebuah **versi** baru yang tidak pernah diubah lagi; dan setiap tindakan ditulis ke dalam **jejak audit** yang entri-entrinya dirangkai dengan hash SHA-256, sehingga tidak ada yang dapat diam-diam menulis ulang masa lalu. Anda akan membangun basis pengetahuan tim ini dalam Java dan Python di atas vcirs-core untuk meja pendaftaran sebuah klinik kecil, tempat ia memilah pasien menurut tingkat kegawatan — RED, YELLOW, atau GREEN — tanpa pernah menyebut nama penyakit, lalu menjalankan satu minggu penuh kerja tim klinik dengannya: sebuah usulan yang ditolak, tinjauan atas usulan sendiri yang ditolak, sebuah kekeliruan, sebuah *rollback*, dan pertanyaan yang cepat atau lambat diajukan setiap tim: *siapa yang mengubah ini, dan mengapa?*

11.1 Meja Pendaftaran Klinik Sehat Bersama

Klinik Sehat Bersama adalah klinik pratama kecil di pinggir kota: seorang dokter, seorang perawat, seorang bidan, dan, di meja pendaftaran, Sari, yang mendaftar setiap pasien. Sejak pukul tujuh pagi bangku-bangku mulai penuh. Sebagian besar pasien datang dengan keluhan biasa — batuk, ruam, kontrol, demam sehari — dan dapat menunggu giliran. Beberapa tidak bisa. Seorang laki-laki yang berkata dadanya terasa sesak, bayi berumur dua bulan yang badannya panas, seorang ibu hamil yang mengalami perdarahan: mereka harus diperiksa sekarang, dan menit-menit di antara pintu depan dan dokter adalah menit-menit yang menentukan.

Sari bukan perawat, dan tidak ada yang mengharapkannya mengenali penyakit. Yang ia butuhkan lebih sederhana dan lebih penting: mengetahui, dari beberapa pertanyaan, apakah seorang pasien harus diperiksa **sekarang**, **segera**, atau **sesuai giliran**. Klinik punya poster di dekat meja yang ditulis dr. Maya bertahun-tahun lalu, dengan tanda-tanda bahaya dicetak merah. Posternya bagus, tetapi punya tiga masalah. Poster tidak dapat bertanya; Sari harus ingat untuk melihatnya. Poster tidak dapat belajar: ketika Ners Budi memeriksa pasien dengan demam empat hari dan gusi berdarah, yang oleh poster akan dimasukkan ke antrean biasa, ia memberi tahu Sari, tetapi posternya tetap sama. Dan tidak ada yang tahu bagian mana yang masih disepakati tim, karena coretan-coretan koreksi dengan pulpen tidak bernama dan tidak bertanggal.

Tim ingin punya sistem pakar di meja pendaftaran, dan setelah Bab 3 dan Bab 10 mereka tahu jenisnya: yang hanya menanyakan apa yang dibutuhkannya, menjelaskan jawabannya, dan belajar dari kasus-kasus ketika ia keliru. Namun mereka juga tahu sesuatu yang tidak perlu dipikirkan laundry. Di klinik, **tiga pakar** akan mengajari basis pengetahuan yang sama, dan pelajaran yang buruk bukan sekadar kemeja yang rusak. Mereka ingin tahu siapa yang

mengajarkan setiap potong pengetahuan dan mengapa; mereka ingin ada sepasang mata kedua pada setiap perubahan; mereka ingin yakin bahwa pelajaran baru tidak diam-diam mengubah jawaban bagi pasien-pasien yang menjadi dasar pelajaran-pelajaran lama; dan ketika sebuah perubahan ternyata keliru, mereka ingin kembali ke pengetahuan kemarin, seluruhnya dan dengan cepat, tanpa kehilangan catatan tentang apa yang terjadi. Bab ini membangun persis hal itu.

11.2 Gagasannya dengan Kata-Kata Sederhana

11.2.1 Triase: memilah menurut kegawatan, bukan menamai penyakit

Triase adalah pemilahan pasien menurut seberapa cepat mereka membutuhkan pertolongan. Instalasi gawat darurat di seluruh dunia memakai skala triase dengan tiga sampai lima tingkat. Interagency Integrated Triage Tool yang diterbitkan Organisasi Kesehatan Dunia (WHO), Komite Internasional Palang Merah (ICRC), dan Médecins Sans Frontières memakai tiga warna: **RED** (merah) untuk pasien yang membutuhkan pertolongan segera, **YELLOW** (kuning) untuk pasien yang membutuhkan pertolongan dalam waktu dekat, dan **GREEN** (hijau) untuk pasien yang aman untuk menunggu [1]. Basis pengetahuan kita memakai ketiga warna yang sama, dan **Tabel 11.1** memperlihatkan arti masing-masing di meja pendaftaran sebuah klinik kecil.

Tabel 11.1 Tiga tingkat kegawatan dalam basis pengetahuan meja pendaftaran, dan jawaban aman bawaan ketika tidak ada aturan yang cocok.

Tingkat	Artinya	Yang dilakukan meja pendaftaran
RED	harus diperiksa sekarang	Panggil perawat atau dokter ke pasien sekarang. Bila tidak ada tenaga klinis di klinik, hubungi 119.
YELLOW	harus diperiksa segera	Beri tahu perawat sekarang; pasien diperiksa sebelum antrean.
GREEN	dapat menunggu	Antrean biasa. Pasien segera kembali ke meja bila ada yang memburuk.
none	tidak ada aturan yang cocok	Minta perawat melihat pasien sekarang.

Di Indonesia, 119 adalah nomor nasional untuk kegawatdaruratan medis, yang dijawab oleh Public Safety Center dalam sistem penanggulangan gawat darurat terpadu [2]. Basis pengetahuan hanya memuat **tanda-tanda yang dapat dilihat atau ditanyakan siapa pun**: apakah pasien sadar, apakah napasnya berat, apakah ada nyeri dada, apakah satu sisi wajahnya mencong, apakah ada perdarahan yang tidak berhenti, sudah berapa hari demamnya, apakah gusinya berdarah. Ia tidak pernah menyimpulkan nama penyakit, dan tidak pernah menyuruh siapa pun minum obat apa. Ia memutuskan satu hal: siapa yang diperiksa lebih dahulu. Tenaga klinis yang memeriksa pasien memutuskan segala hal lainnya. Dan ketika basis pengetahuan tidak punya aturan untuk seorang pasien, jawabannya bukan GREEN, melainkan “minta perawat melihatnya sekarang” — jawaban bawaan paling aman yang dapat dimiliki meja pendaftaran.

Tanda-tandanya sendiri berasal dari pedoman yang terbuka untuk umum. Delapan pelajaran pertama, yang diambil dr. Maya dari poster klinik, mengikuti tanda-tanda bahaya yang dipakai dalam triase gawat darurat [1]. Pelajaran yang ditambahkan Ners Budi selama minggu itu mengikuti tanda-tanda peringatan dengue berat, yang mencakup perdarahan dari hidung atau gusi pada pasien yang demamnya sudah berlangsung beberapa hari, dan yang menuntut pertolongan segera [3].

11.2.2 Banyak pakar, satu basis pengetahuan

Ripple Down Rules diciptakan untuk jenis pengetahuan seperti ini: pengetahuan para praktisi yang menambahkan aturan ketika mereka melihat kasus yang dijawab keliru oleh sistem, tanpa perekayasa pengetahuan di antara mereka dan sistem [4]. Sistem-sistem patologi yang digambarkan di Bab 3 dirawat oleh beberapa ahli patologi sekaligus, dan tinjauan atas dua dekade penelitian RDR menganggap banyak pakar sebagai keadaan yang wajar bagi RDR, bukan pengecualian [5]. Kasus cornerstone-lah yang memungkinkannya: setiap aturan membawa kasus yang membenarkannya, sehingga seorang pakar yang tidak pernah bertemu orang yang mengajarkan sebuah aturan lama tetap dapat melihat mengapa aturan itu ada.

Namun RDR sendiri tidak mengatakan apa pun tentang *bagaimana* beberapa orang berbagi satu basis pengetahuan. Dengan sekadar berkas bersama, tiga hal menjadi salah. Pertama, **pelajaran yang hilang**: dua pakar membuka berkas yang sama, masing-masing mengajarkan sesuatu, masing-masing menyimpan, dan penyimpanan kedua menimpa yang pertama (Subbab 11.3 memperlihatkan hal itu terjadi). Kedua, **perubahan yang tidak ditinjau**: dalam RDR seorang pakar menambahkan aturan berdasarkan penilaiannya sendiri; di klinik, aturan yang mengirim pasien ke antrean biasa tidak boleh bergantung pada penilaian satu orang pada satu pagi yang sibuk. Ketiga, **tidak ada jalan kembali**: RDR tidak pernah menghapus node, dan itu melindungi kasus-kasus lama, tetapi ketika sebuah perubahan utuh ternyata keliru, tim ingin kembali ke pengetahuan sebelumnya, dan ingin tahu persis apa yang terjadi di antaranya.

VCIRS sudah menyimpan sebagian jawabannya. Basis pengetahuan dalam tesis mencatat, pada setiap node, siapa yang mengajarkannya dan kasus yang menciptakannya, dan vcirs-core menambahkan **riwayat** setiap perubahan pada basis pengetahuan [6], [7]. Bab ini membangun sisanya di sekelilingnya, dengan empat gagasan yang dipakai tim perangkat lunak setiap hari: versi, pengujian, tinjauan, dan jejak audit.

11.2.3 Versi: sebuah potret setelah setiap perubahan

Versi adalah salinan lengkap basis pengetahuan sebagaimana adanya setelah satu perubahan, disimpan dalam berkas yang tidak pernah diubah lagi: `v001.json` setelah pelajaran-pelajaran pertama, `v002.json` setelah perubahan berikutnya, dan seterusnya. Pengetahuan yang berlaku adalah versi yang paling baru. Tidak ada yang pernah disunting di tempat; sebuah perubahan selalu menghasilkan berkas baru.

Dua hal mengikutinya. Siapa pun dapat melihat pengetahuan sebagaimana adanya pada hari apa pun: “apa yang akan dikatakan meja pendaftaran kepada pasien dengan tanda-tanda ini pada hari Selasa?” punya jawaban yang pasti. Dan kembali ke belakang hanyalah menyalin: **rollback** ke versi 4 menciptakan versi baru, misalnya versi 6, yang isinya persis versi 4. Versi 5 tidak dihapus — ia tetap ada di folder dan di jejak, sebagai catatan kekeliruan — tetapi meja pendaftaran kini kembali memakai pengetahuan versi 4. Begitulah sistem kendali versi yang dipakai para pemrogram bekerja, dan itulah cara paling aman untuk membatalkan perubahan: tidak ada yang dimusnahkan.

11.2.4 Pengujian: kasus cornerstone dan kasus emas

Uji di sini adalah sebuah kasus yang jawaban benarnya kita ketahui, dijalankan melalui basis pengetahuan untuk melihat apakah basis pengetahuan masih memberikannya. Sebelum sebuah perubahan diterima, ia dicoba pada *salinan* versi yang berlaku, dan salinan itu diuji dengan dua jenis kasus.

Jenis pertama gratis: **kasus-kasus cornerstone**. Setiap node basis pengetahuan VCIRS membawa kasus yang menciptakannya, dan kasus itu harus tetap mendapat jawaban node tersebut setelah perubahan apa pun. Inilah pemeriksaan yang dilakukan RDR ketika sebuah aturan ditambahkan — "cornerstone yang terdampak" dari Bab 3 — yang diperluas ke setiap node basis pengetahuan [8], [4]. Perubahan yang akan mengubah jawaban kasus cornerstone terdahulu mana pun adalah persis jenis perubahan yang diam-diam merusak pengetahuan lama, dan perubahan itu ditolak.

Jenis kedua dipilih oleh tim: **kasus-kasus emas**. Kasus emas adalah kasus yang disepakati seluruh tim, dituliskan bersama tingkat kegawatan yang diharapkan dan alasannya: "bayi di bawah tiga bulan dengan demam: RED". Kasus emas melindungi apa yang dianggap tim paling penting, termasuk keadaan yang tidak menjadi dasar node mana pun. Ketika sesuatu salah dalam praktik, refleks yang benar adalah menuliskan kasus itu sebagai kasus emas, sehingga kekeliruan yang sama tidak dapat kembali tanpa ketahuan.

> **DEFINISI.** *Uji regresi* (*regression test*) memeriksa bahwa sebuah perubahan tidak merusak apa yang sebelumnya berjalan baik. Dalam basis pengetahuan tim, uji regresinya adalah kasus-kasus cornerstone semua node, yang harus tetap mendapat jawaban node masing-masing, dan kasus-kasus emas, yang harus tetap mendapat jawaban yang diharapkan.

11.2.5 Empat mata: tinjauan oleh pakar lain

Perubahan yang lolos pengujian belum menjadi bagian dari pengetahuan. Ia menunggu sebagai **usulan** sampai pakar kedua **meninjau**-nya dan menyetujui atau menolaknya; tidak ada yang boleh meninjau usulannya sendiri. Inilah *prinsip empat mata* yang dipakai bank dan rumah sakit untuk keputusan-keputusan penting. Pengujian menangkap perubahan yang akan merusak kasus-kasus lama; peninjau menangkap apa yang tidak dapat ditangkap pengujian: aturan yang terlalu luas, alasan yang tidak meyakinkan, fakta dari daftar perbedaan yang dipilih padahal bukan alasan sebenarnya. **Gambar 11.1** memperlihatkan perjalanan sebuah perubahan dari usulan sampai menjadi versi.



Gambar 11.1 Perjalanan sebuah perubahan. Usulan dicoba pada salinan versi yang berlaku lalu diuji; bila setiap kasus masih lolos, pakar lain meninjaunya; hanya perubahan yang disetujui yang menjadi versi baru. Setiap langkah, termasuk setiap penolakan, ditulis ke jejak audit.

Tim juga perlu memutuskan siapa boleh melakukan apa. **Tabel 11.2** adalah pengaturan di Klinik Sehat Bersama; setiap klinik dapat menentukan sendiri.

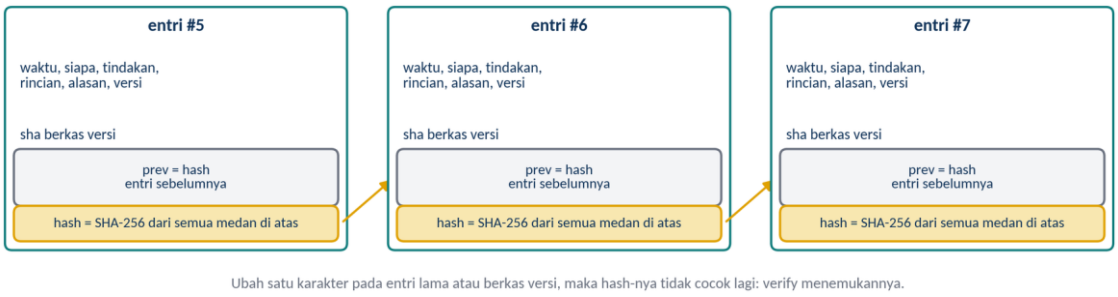
Tabel 11.2 Siapa boleh melakukan apa dalam basis pengetahuan tim klinik.

Tindakan	Siapa yang boleh
berkonsultasi dengan basis pengetahuan; melaporkan jawaban yang meragukan	semua orang, termasuk meja pendaftaran
menambahkan kasus emas; mengusulkan perubahan	dokter, perawat, bidan
meninjau usulan	dokter, perawat, bidan — tidak pernah usulannya sendiri
membuat basis pengetahuan; menghentikan node; rollback	dokter

11.2.6 Jejak audit, dan mengapa ia dirangkai

Jejak audit adalah daftar segala sesuatu yang terjadi pada basis pengetahuan: setiap pembuatan, kasus emas, laporan, usulan, tinjauan, penghentian, dan rollback, beserta waktunya, orangnya, perannya, apa yang dilakukan, alasan yang diberikan, dan versi yang dihasilkan. Jejak ini menjawab pertanyaan-pertanyaan yang penting setelah sebuah insiden: siapa yang mengubah ini, kapan, mengapa, dan siapa yang menyetujuinya?

Jejak hanya berharga bila tidak dapat ditulis ulang diam-diam. Jejak kita dilindungi oleh **hash**. Fungsi hash kriptografis seperti SHA-256 mengubah teks apa pun menjadi sidik jari sepanjang 64 karakter; mengubah satu karakter saja dalam teks mengubah sidik jarinya sama sekali, dan tidak ada yang tahu cara menemukan teks lain dengan sidik jari yang sama [9]. Setiap entri jejak menyimpan tiga sidik jari: `sha`, sidik jari berkas versi yang diciptakannya (bila ada); `prev`, sidik jari entri sebelumnya; dan `hash`, sidik jari entri itu sendiri, yang dihitung dari semua medannya yang lain. **Gambar 11.2** memperlihatkan rangkaiannya.



Gambar 11.2 Jejak audit sebagai rantai. Setiap entri memuat sidik jari berkas versi yang diciptakannya dan sidik jari entri sebelumnya, dan sidik jarinya sendiri mencakup keduanya, sehingga mengubah entri lama atau berkas versi mana pun memutus setiap mata rantai sesudahnya.

Bila seseorang kemudian menyunting sebuah entri lama — mengganti alasan, menghapus nama — sidik jari entri itu tidak lagi cocok dengan isinya. Bila ia juga menghitung ulang sidik jari itu, `prev` pada entri berikutnya tidak lagi cocok, dan seterusnya sepanjang rantai. Bila seseorang menyunting berkas versi lama, sidik jarinya tidak lagi cocok dengan `sha` di jejak. Satu perintah, `verify`, menghitung ulang semuanya dan menyebutkan di mana rantainya putus. Gagasan yang sama membuat riwayat sistem kendali versi dan buku besar publik tahan-rusak; ia tidak mencegah siapa pun mengubah berkas, tetapi ia menjamin bahwa perubahannya akan terlihat.

Jejak itu juga merupakan catatan tentang **orang**, bukan tentang pasien. Ia memuat nama anggota tim, karena memang untuk itulah ia ada, dan fakta-fakta kasus yang mereka ajarkan, yang berupa tanda-tanda seperti `fever days = 4`, tidak pernah nama, nomor, atau alamat pasien. Di Indonesia, Undang-Undang Pelindungan Data Pribadi mensyaratkan agar data pribadi hanya diproses untuk tujuan yang dinyatakan dan dilindungi [10]; data kesehatan orang yang dapat

dikenali termasuk data yang paling dilindunginya. Basis pengetahuan tentang tanda-tanda tidak membutuhkan data seperti itu sama sekali.

11.2.7 Ketika para pakar tidak sepakat

Tiga pakar tidak akan selalu sepakat, dan basis pengetahuan tim tidak boleh berpura-pura bahwa mereka selalu sepakat. Ketidaksepakatan ada dua jenis, dan keduanya ditangani secara berbeda.

Jenis pertama adalah ketidaksepakatan **tentang sebuah kasus**: fakta-faktanya sama, dan dua pakar menginginkan jawaban yang berbeda. Budi ingin anak yang demam sehari menjadi YELLOW; rapat hari Jumat menginginkan GREEN. Tidak ada basis pengetahuan yang dapat menyelesaikannya, karena ini bukan soal representasi pengetahuan melainkan soal penilaian klinis dan kapasitas klinik. Yang dapat dilakukan basis pengetahuan adalah membuat ketidaksepakatan itu terlihat dan menjaganya tetap jujur. Usulan yang bertentangan dengan kasus emas gagal dalam pengujiannya dan menyebutkan kasus mana; peninjau yang menolak usulan menuliskan alasan yang dapat dibaca semua orang; kasus emas yang tidak dipenuhi pengetahuan yang berlaku membuat setiap pengujian berikutnya gagal sampai seseorang menyelesaikannya. Ketidaksepakatan itu lalu diselesaikan di tempat yang semestinya — dalam rapat, oleh manusia — dan hasilnya dituliskan ke dalam pengetahuan sebagai usulan atau kasus emas, dengan rapat itu sebagai alasannya.

Jenis kedua adalah ketidaksepakatan **tentang alasannya**: dua pakar sepakat tentang jawabannya tetapi akan memilih kondisi yang berbeda. Di sini Ripple Down Rules bermurah hati. Dua pengecualian dengan kondisi berbeda dan konklusi yang sama dapat hidup berdampingan di bawah node yang sama, selama tidak ada yang mengubah kasus cornerstone yang lain, dan pengujian memeriksa persis hal itu. Statistik pemakaian VCIRS, yang dibangun di Bab 7, belakangan memperlihatkan variabel dan node mana yang paling sering dipakai, dan statistik itu adalah titik awal yang baik untuk percakapan tentang alasan mana yang dipercaya tim.

Aturan praktis, sampai rapat berlangsung: **jawaban yang lebih gawat yang menang**. Bila seorang pakar berkata RED dan yang lain YELLOW, meja pendaftaran memperlakukan pasien sebagai RED sampai tim memutuskan, karena mengirim perawat untuk melihat pasien yang sebenarnya dapat menunggu hanya memakan beberapa menit, sedangkan kekeliruan sebaliknya dapat memakan jauh lebih banyak.

11.3 Langkah 1: Dua Pakar, Satu Berkas

Sebelum membangun apa pun, mari kita saksikan masalah pertama dari Subbab 11.2.2 terjadi. Dalam **Kode 11.1**, dr. Maya menyimpan basis pengetahuan ke berkas bersama; lalu Ners Budi dan Bidan Ratna membuka berkas itu pada saat yang sama, masing-masing mengajarkan satu pelajaran, dan masing-masing menyimpan.

Kode 11.1 `lost_update.py`: dua pakar mengubah satu berkas bersama, dan satu pelajaran lenyap.

```
"""Two experts change one knowledge-base file at the same time, and one
lesson is lost. Why a team needs more than a shared file.

    python lost_update.py
"""

import os
import pathlib
import sys
import tempfile

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import Case, KnowledgeBase

path = os.path.join(tempfile.mkdtemp(), "shared-kb.json")
kb = KnowledgeBase("shared")
kb.learn(Case.of(["fever = yes"], ["urgency = GREEN"]), "dr. Maya")
kb.save(path)
print("Monday 08:00 the shared file has: " + ", ".join(kb.nodes))

budi = KnowledgeBase.load(path)          # both open the same file ...
ratna = KnowledgeBase.load(path)
budi.learn(Case.of(["fever = yes", "fever days = 4",
                    "bleeding gums or nose = yes"], ["urgency = RED"]),
           "Ners Budi")
ratna.learn(Case.of(["pregnant = yes", "bleeding in pregnancy = yes"],
                    ["urgency = RED"]), "Bidan Ratna")
budi.save(path)                          # ... and each saves her own copy
print("08:05 Budi saves: " + ", ".join(budi.nodes))
ratna.save(path)
print("08:06 Ratna saves: " + ", ".join(ratna.nodes))

final = KnowledgeBase.load(path)
print("Monday 09:00 the shared file has: " + ", ".join(final.nodes))
kept = [str(n) for n in final.nodes.values()]
lost = [n for n in budi.nodes.values() if str(n) not in kept]
print("Lost without a trace: " + (", ".join(
    str(n) + " by " + n.author for n in lost) or "nothing"))
```

Kode 11.2 adalah hal yang sama dalam Java.

Kode 11.2 `LostUpdate.java`: pelajaran yang hilang yang sama, dalam Java.

```
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.List;

import vcirs.Case;
import vcirs.KnowledgeBase;
import vcirs.Node;

/**
 * Two experts change one knowledge-base file at the same time, and one
 * lesson is lost. Why a team needs more than a shared file.
 */
public class LostUpdate {
    public static void main(String[] args) throws Exception {
        String path = Files.createTempDirectory("team").resolve("shared-kb.json")
                .toString();
        KnowledgeBase kb = new KnowledgeBase("shared");
        kb.learn(Case.of(List.of("fever = yes"), List.of("urgency = GREEN")),
                "dr. Maya");
        kb.save(path);
        System.out.println("Monday 08:00 the shared file has: "
                + String.join(", ", kb.nodes.keySet()));

        KnowledgeBase budi = KnowledgeBase.load(path);  // both open the file ...
        KnowledgeBase ratna = KnowledgeBase.load(path);
        budi.learn(Case.of(List.of("fever = yes", "fever days = 4",
                "bleeding gums or nose = yes"), List.of("urgency = RED")),
                "Ners Budi");
        ratna.learn(Case.of(List.of("pregnant = yes",
                "bleeding in pregnancy = yes"), List.of("urgency = RED")),
                "Bidan Ratna");
        budi.save(path);                    // ... and each saves her own copy
        System.out.println("08:05 Budi saves: "
                + String.join(", ", budi.nodes.keySet()));
        ratna.save(path);
        System.out.println("08:06 Ratna saves: "
                + String.join(", ", ratna.nodes.keySet()));
```

Kode 11.2 (Lanjutan) *LostUpdate.java: pelajaran yang hilang yang sama, dalam Java.*

```
        KnowledgeBase last = KnowledgeBase.load(path);
        System.out.println("Monday 09:00 the shared file has: "
                + String.join(", ", last.nodes.keySet()));
        List<String> kept = new ArrayList<>();
        for (Node n : last.nodes.values()) {
            kept.add(n.toString());
        }
        List<String> lost = new ArrayList<>();
        for (Node n : budi.nodes.values()) {
            if (!kept.contains(n.toString())) {
                lost.add(n + " by " + n.author);
            }
        }
        System.out.println("Lost without a trace: "
                + (lost.isEmpty() ? "nothing" : String.join(", ", lost)));
    }
}
```

Keduanya mencetak:

```
Monday 08:00 the shared file has: GREEN#1
08:05 Budi saves: GREEN#1, RED#1
08:06 Ratna saves: GREEN#1, RED#1
Monday 09:00 the shared file has: GREEN#1, RED#1
Lost without a trace: RED#1: fever days = 4, bleeding gums or nose = yes -> urgency = RED
by Ners Budi
```

Pelajaran Budi — perdarahan dengan demam yang lama berarti RED — lenyap, dan tidak ada yang mengatakannya. Setiap program memuat berkasnya, mengubah salinannya sendiri di memori, lalu menulis seluruh salinannya kembali; tulisan kedua begitu saja menggantikan tulisan pertama. Ada bahaya kedua yang lebih senyap dalam keluaran itu: kedua node baru bernama `RED#1`, karena masing-masing program memilih nama bebas berikutnya dalam salinannya sendiri. Dengan berkas bersama biasa, dua potong pengetahuan yang berbeda bahkan dapat berakhir dengan nama yang sama.

Basis data dengan transaksi, seperti di Bab 10, mencegah dua penyimpanan tercampur, tetapi dengan sendirinya tidak mencegah hal ini: masing-masing program tetap menulis kembali salinan yang dimuatnya sebelum perubahan program lain. Yang mencegahnya adalah sebuah aturan: **setiap perubahan dimulai dari versi yang berlaku dan menghasilkan versi berikutnya**. Basis pengetahuan tim di subbab berikut menerapkan setiap perubahan pada salinan segar dari versi terbaru, pada saat perubahan itu terjadi, dan mencatatnya sebagai versi $n + 1$; tidak pernah ada salinan kedua yang disunting bersamaan.

11.4 Langkah 2: Basis Pengetahuan Tim

Basis pengetahuan tim adalah sebuah folder dengan tiga jenis berkas:

```
data/clinic/
  versions/v001.json, v002.json, ...   every version, never changed
  state.json                           golden cases, reports and proposals
  audit.json                           the chained audit trail
```

dan sebuah kelas, `TeamKb`, dengan satu metode untuk setiap tindakan dalam **Tabel 11.2**. **Kode 11.3** adalah versi Python-nya.

Kode 11.3 `team_kb.py`: basis pengetahuan tim, dengan versi, pengujian, tinjauan, penghentian, rollback, dan jejak audit yang dirangkai.

```
"""A clinic's triage knowledge base for a team: many experts, versions,
tests against cornerstone and golden cases, and an audit trail.

The folder of a team knowledge base holds:
  versions/v001.json ...  every version of the knowledge, never changed
  state.json              golden cases, reports and proposals
  audit.json              who did what, when and why; each entry is chained
                          to the one before it by a SHA-256 hash
"""

import hashlib
import json
import os
import pathlib
import sys

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import (BOTH, TOP_LEVEL, Case, Conclusion, Condition,
                   KnowledgeBase, VcirsError, jsonio, report)

EXPERTS = ("doctor", "nurse", "midwife")
ACTIONS = {
    "RED": "Call the nurse or doctor to the patient now. "
           "If no clinician is in the clinic, call 119.",
    "YELLOW": "Tell the nurse now; the patient is seen before the queue.",
    "GREEN": "Normal queue. The patient comes back to the desk at once "
             "if anything gets worse.",
}
NO_MATCH = "No rule matched: ask the nurse to look at the patient now."


class TeamError(Exception):
    """A request the team knowledge base refuses, with the reason."""


def sha(text):
    return hashlib.sha256(text.encode("utf-8")).hexdigest()


def parse_list(text):
    """'a = b, c >= 3' -> [Condition, Condition]; '' -> []."""
    return [Condition.parse(p) for p in text.split(", ") if p.strip()]
```

Kode 11.3 (Lanjutan) *team_kb.py: basis pengetahuan tim, dengan versi, pengujian, tinjauan, penghentian, rollback, dan jejak audit yang dirangkai.*

```
def parse_case(text):
    """'fever = yes, age group = baby -> RED' -> (facts, 'RED')."""
    if "->" not in text:
        raise TeamError("write a case as: facts -> RED, YELLOW or GREEN")
    left, level = text.split("->")
    level = level.strip()
    if level not in ACTIONS:
        raise TeamError("the urgency must be RED, YELLOW or GREEN")
    return parse_list(left.strip()), level


def show(conds):
    return ", ".join(str(c) for c in conds)


def place(node):
    return node.parent if node.parent else "(top level)"


def answer(kb, facts):
    """The urgency the knowledge base gives, and the inference result."""
    result = kb.infer(Case(facts).known())
    got = result.conclusions()
    return (got[0].value if got else "none"), result


def teach(kb, facts, level, when, author):
    """Ripple-down teaching: None if the answer is already right; otherwise
    the new node, an exception of the node that answered (or a new
    top-level node), made of the conditions the expert chose ('when')."""
    got, result = answer(kb, facts)
    if got == level:
        return None
    case = Case(facts, [Conclusion("urgency", level)])
    fired = result.paths[0].last().id if result.paths else None
    option = BOTH if fired else TOP_LEVEL
    return kb.refine(case, option, fired, when or None, author)
```

Kode 11.3 (Lanjutan) *team_kb.py: basis pengetahuan tim, dengan versi, pengujian, tinjauan, penghentian, rollback, dan jejak audit yang dirangkai.*

```
class TeamKb:
    def __init__(self, folder, team_file):
        self.folder = folder
        with open(team_file, encoding="utf-8") as f:
            team = json.load(f)
        self.members = {m["name"]: m["role"] for m in team["members"]}
        self.clock = lambda: ""
        self.state = {"golden": [], "reports": [], "proposals": []}
        self.audit = []
        os.makedirs(os.path.join(folder, "versions"), exist_ok=True)
        if os.path.exists(self._path("audit.json")):
            with open(self._path("audit.json"), encoding="utf-8") as f:
                self.audit = json.load(f)
            with open(self._path("state.json"), encoding="utf-8") as f:
                self.state = json.load(f)

    # ----- files ------------------------------------------------------------
    def _path(self, name):
        return os.path.join(self.folder, name)

    def _write(self, name, text):
        with open(self._path(name), "w", encoding="utf-8", newline="\n") as f:
            f.write(text)

    def _read(self, name):
        with open(self._path(name), encoding="utf-8") as f:
            return f.read()

    def version(self):
        """The number of the current version (0 before the first)."""
        return self.audit[-1]["version"] if self.audit else 0

    @staticmethod
    def vname(n):
        return "versions/v%03d.json" % n

    def kb(self, n=None):
        """A fresh copy of version n (default: the current version)."""
        return KnowledgeBase.from_json(self._read(self.vname(n or
                                                             self.version())))
```

Kode 11.3 (Lanjutan) *team_kb.py: basis pengetahuan tim, dengan versi, pengujian, tinjauan, penghentian, rollback, dan jejak audit yang dirangkai.*

```
    # ----- people, and the trail they leave ------------------------------
    def role(self, who, allowed=None):
        if who not in self.members:
            raise TeamError(who + " is not a member of the team")
        role = self.members[who]
        if allowed is not None and role not in allowed:
            raise TeamError(who + " (" + role + ") may not do this")
        return role

    def _log(self, who, action, detail, reason, version, text=""):
        """Append a chained audit entry; save a new version if text."""
        if text:
            self._write(self.vname(version), text)
        entry = {"n": len(self.audit) + 1, "time": self.clock(), "who": who,
                 "role": self.members.get(who, ""), "action": action,
                 "detail": detail, "reason": reason, "version": version,
                 "sha": sha(text) if text else "",
                 "prev": self.audit[-1]["hash"] if self.audit else ""}
        entry["hash"] = sha(jsonio.dumps(entry))
        self.audit.append(entry)
        self._write("audit.json", jsonio.dumps(self.audit) + "\n")
        self._write("state.json", jsonio.dumps(self.state) + "\n")
        return entry

    # ----- tests ---------------------------------------------------------
    def run_tests(self, kb):
        """Every node's cornerstone case must still get its node's answer,
        and every golden case its expected answer. Returns (count, fails)."""
        fails, count = [], 0
        for node in kb.nodes.values():
            if node.stopped or node.cornerstone is None:
                continue
            count += 1
            want = node.conclusions[0].value
            got, _ = answer(kb, node.cornerstone.facts)
            if got != want:
                fails.append("cornerstone of " + node.id + " ("
                             + show(node.cornerstone.facts) + "): expected "
                             + want + ", got " + got)
        for g in self.state["golden"]:
            count += 1
            got, _ = answer(kb, parse_list(", ".join(g["facts"])))
            if got != g["expected"]:
                fails.append("golden " + g["id"] + " (" + ", ".join(g["facts"])
                             + "): expected " + g["expected"] + ", got " + got)
        return count, fails
```

Kode 11.3 (Lanjutan) *team_kb.py: basis pengetahuan tim, dengan versi, pengujian, tinjauan, penghentian, rollback, dan jejak audit yang dirangkai.*

```
    # ----- what the team does -----------------------------------------
    def create(self, who, seed_file, reason):
        self.role(who, ("doctor",))
        if self.audit:
            raise TeamError("the knowledge base already exists")
        with open(seed_file, encoding="utf-8") as f:
            seed = json.load(f)
        kb = KnowledgeBase(seed["name"], seed["description"])
        for var, question, choices in seed["questions"]:
            kb.declare(var, question, choices)
        for step in seed["steps"]:
            teach(kb, parse_list(", ".join(step["facts"])), step["urgency"],
                  parse_list(", ".join(step["when"])), who)
        self._log(who, "create", kb.summary(), reason, 1, kb.to_json())
        return ["v1 created: " + kb.summary()]

    def consult(self, who, facts_text):
        self.role(who)
        got, result = answer(self.kb(), parse_list(facts_text))
        out = [who + " asks about: " + facts_text]
        if got == "none":
            out.append("  urgency: none  (" + NO_MATCH + ")")
            if result.paths:
                out.append("  the last matching rule, "
                           + result.paths[0].last().id + ", is stopped")
            return out
        out.append("  urgency: " + got + "  (rule "
                   + result.paths[0].last().id + ", version v"
                   + str(self.version()) + ")")
        out.append("  what to do: " + ACTIONS[got])
        out.append("  why:")
        out += ["    " + line for line in result.how()]
        return out

    def golden(self, who, case_text, reason):
        self.role(who, EXPERTS)
        facts, level = parse_case(case_text)
        gid = "G" + str(len(self.state["golden"]) + 1)
        self.state["golden"].append({"id": gid, "facts": [str(c) for c in facts],
                                     "expected": level, "by": who,
                                     "reason": reason})
        got, _ = answer(self.kb(), facts)
        self._log(who, "golden", gid + ": " + case_text, reason,
                  self.version())
        note = "passes now" if got == level else "FAILS now: gives " + got
```

Kode 11.3 (Lanjutan) *team_kb.py: basis pengetahuan tim, dengan versi, pengujian, tinjauan, penghentian, rollback, dan jejak audit yang dirangkai.*

```
        return [gid + " added (" + note + ")"]

    def report(self, who, facts_text, note):
        self.role(who)
        rid = "R" + str(len(self.state["reports"]) + 1)
        got, _ = answer(self.kb(), parse_list(facts_text))
        self.state["reports"].append({"id": rid, "by": who, "facts": facts_text,
                                      "answer": got, "note": note})
        self._log(who, "report", rid + ": " + facts_text + " (answer "
                  + got + ")", note, self.version())
        return [rid + " recorded for the experts to read"]

    def propose(self, who, case_text, when_text, reason):
        self.role(who, EXPERTS)
        facts, level = parse_case(case_text)
        when = parse_list(when_text)
        pid = "P" + str(len(self.state["proposals"]) + 1)
        p = {"id": pid, "by": who, "case": case_text, "when": when_text,
             "reason": reason, "status": "", "base": self.version(),
             "node": "", "reviewer": ""}
        self.state["proposals"].append(p)
        kb = self.kb()
        node = teach(kb, facts, level, when, who)
        if node is None:
            p["status"] = "already known"
            out = [pid + ": the knowledge base already answers " + level]
        else:
            count, fails = self.run_tests(kb)
            p["node"] = node.id
            if fails:
                p["status"] = "rejected by tests"
                out = [pid + ": would add " + str(node) + " under "
                       + place(node)]
                out.append("  REJECTED: " + str(len(fails)) + " of "
                           + str(count) + " test cases would change:")
                out += ["    " + f for f in fails]
            else:
                p["status"] = "waiting for review"
                out = [pid + ": would add " + str(node) + " under "
                       + place(node)]
                out.append("  all " + str(count) + " test cases still pass; "
                           "waiting for another expert to review")
        self._log(who, "propose", pid + " (" + p["status"] + "): "
                  + case_text, reason, self.version())
        return out
```

Kode 11.3 (Lanjutan) *team_kb.py: basis pengetahuan tim, dengan versi, pengujian, tinjauan, penghentian, rollback, dan jejak audit yang dirangkai.*

```
    def review(self, who, pid, decision, reason):
        self.role(who, EXPERTS)
        found = [p for p in self.state["proposals"] if p["id"] == pid]
        if not found or found[0]["status"] != "waiting for review":
            raise TeamError(pid + " is not waiting for review")
        p = found[0]
        if p["by"] == who:
            raise TeamError("nobody may review their own proposal")
        p["reviewer"] = who
        if decision != "approve":
            p["status"] = "rejected by " + who
            self._log(who, "reject", pid, reason, self.version())
            return [pid + " rejected"]
        facts, level = parse_case(p["case"])
        kb = self.kb()                          # the knowledge may have moved on
        node = teach(kb, facts, level, parse_list(p["when"]), p["by"])
        count, fails = self.run_tests(kb) if node else (0, [])
        if node is None or fails:
            p["status"] = "rejected by tests"
            self._log(who, "reject", pid + " (tests)", reason, self.version())
            return [pid + " no longer passes the tests; it was not applied"]
        p["status"] = "approved"
        p["node"] = node.id
        v = self.version() + 1
        self._log(who, "approve", pid + ": " + str(node), reason, v,
                  kb.to_json())
        return [pid + " approved: v" + str(v) + " adds " + node.id
                + " (" + str(count) + " test cases pass)"]

    def stop(self, who, node_id, reason):
        self.role(who, ("doctor",))
        kb = self.kb()
        try:
            kb.stop(node_id, who)
        except VcirsError as e:
            raise TeamError(str(e))
        count, fails = self.run_tests(kb)
        if fails:
            raise TeamError("stopping " + node_id + " breaks: "
                            + "; ".join(fails))
        v = self.version() + 1
        self._log(who, "stop", node_id, reason, v, kb.to_json())
        return ["v" + str(v) + ": " + node_id + " stopped (" + str(count)
                + " test cases pass)"]
```

Kode 11.3 (Lanjutan) *team_kb.py: basis pengetahuan tim, dengan versi, pengujian, tinjauan, penghentian, rollback, dan jejak audit yang dirangkai.*

```
    def rollback(self, who, vtext, reason):
        self.role(who, ("doctor",))
        n = int(vtext.lstrip("v"))
        if n < 1 or n >= self.version():
            raise TeamError("roll back to an earlier version, v1 to v"
                            + str(self.version() - 1))
        text = self._read(self.vname(n))
        v = self.version() + 1
        self._log(who, "rollback", "back to v" + str(n), reason, v, text)
        return ["v" + str(v) + " is a copy of v" + str(n)
                + "; nothing in between is lost from the trail"]

    # ----- looking back ---------------------------------------------------
    def test(self):
        count, fails = self.run_tests(self.kb())
        if not fails:
            return ["v" + str(self.version()) + ": all " + str(count)
                    + " test cases pass"]
        return ["v" + str(self.version()) + ": " + str(len(fails))
                + " of " + str(count) + " test cases fail:"] \
            + ["  " + f for f in fails]

    def diff(self, a, b):
        old, new = self.kb(int(a.lstrip("v"))), self.kb(int(b.lstrip("v")))
        out = ["from " + a + " to " + b + ":"]
        for n in new.nodes.values():
            if n.id not in old.nodes:
                out.append("  added   " + str(n))
            elif n.stopped and not old.nodes[n.id].stopped:
                out.append("  stopped " + n.id)
        for n in old.nodes.values():
            if n.id not in new.nodes:
                out.append("  gone    " + str(n))
            elif old.nodes[n.id].stopped and not new.nodes[n.id].stopped:
                out.append("  resumed " + n.id)
        return out if len(out) > 1 else out + ["  no change"]

    def tree(self, vtext):
        return report.tree(self.kb(int(vtext.lstrip("v"))))
```

Kode 11.3 (Lanjutan) *team_kb.py: basis pengetahuan tim, dengan versi, pengujian, tinjauan, penghentian, rollback, dan jejak audit yang dirangkai.*

```
    def blame(self, node_id):
        kb = self.kb()
        if node_id not in kb.nodes:
            raise TeamError("no node " + node_id + " in v"
                            + str(self.version()))
        n = kb.nodes[node_id]
        out = [str(n), "  under: " + place(n),
               "  taught by: " + n.author,
               "  cornerstone case: " + show(n.cornerstone.facts)]
        for p in self.state["proposals"]:
            if p["node"] == node_id and p["status"] == "approved":
                out.append("  proposal " + p["id"] + ": " + p["reason"])
                out.append("  approved by " + p["reviewer"])
        for e in self.audit:
            if e["sha"] and node_id in e["detail"]:
                out.append("  in the trail: #" + str(e["n"]) + " "
                           + e["time"] + " " + e["action"] + " by "
                           + e["who"] + " -> v" + str(e["version"]))
        return out

    def log(self):
        out = []
        for e in self.audit:
            out.append("#" + str(e["n"]) + " " + e["time"] + " v"
                       + str(e["version"]) + " " + e["who"] + " ("
                       + e["role"] + ") " + e["action"] + ": "
                       + e["detail"])
            out.append("    why: " + e["reason"])
        return out

    def verify(self):
        """Recompute every hash of the trail and of every version file."""
        prev = ""
        for e in self.audit:
            body = {k: v for k, v in e.items() if k != "hash"}
            if e["prev"] != prev or sha(jsonio.dumps(body)) != e["hash"]:
                return ["the trail is broken at entry #" + str(e["n"])]
            if e["sha"] and sha(self._read(self.vname(e["version"]))) \
                    != e["sha"]:
                return ["version v" + str(e["version"]) + " was changed "
                        "after entry #" + str(e["n"])]
            prev = e["hash"]
        return ["the trail is intact: " + str(len(self.audit))
                + " entries, " + str(self.version()) + " versions, last hash "
                + prev[:12]]
```

Bacalah dalam lima bagian.

Kasus sebagai teks. Sebuah kasus ditulis sebagaimana seseorang akan mengucapkannya: `fever = yes, age group = baby -> RED`. `parse_list` mengubah `a = b, c >= 3` menjadi daftar kondisi vcirs-core, dan `parse_case` memisahkan tanda panah dan memeriksa bahwa tingkatnya RED, YELLOW, atau GREEN. Kondisi-kondisi yang dipilih pakar untuk node baru — bagian *when* sebuah usulan — memakai bentuk yang sama, sehingga seorang pakar dapat menulis `fever days >= 3` padahal kasusnya sendiri berbunyi `fever days = 3`.

Mengajar. `teach` adalah Ripple Down Rules dalam enam baris, ditulis untuk pilihan yang dibuat pakar. Ia menanyakan jawaban basis pengetahuan untuk kasus itu. Bila jawabannya sudah benar, tidak ada yang perlu diajarkan. Bila tidak, ia menemukan node yang menjawab dan menyempurnakan di bawahnya dengan opsi 3 VCIRS (sebuah pengecualian), atau, bila tidak ada node yang menjawab, menambahkan node tingkat atas baru dengan opsi 5 [7]. Bila pakar

memberikan kondisi *when*, kondisi-kondisi itulah yang menjadi kondisi node baru — pilihan dari daftar perbedaan yang oleh revisi VCIRS tahun 2006 diserahkan kepada pengguna [7]; bila tidak, mesin membangunnya dari daftar perbedaan itu sendiri. Kasusnya selalu disimpan utuh sebagai cornerstone node baru.

Jejak. `_log` adalah satu-satunya metode yang menulis. Ia menyimpan berkas versi baru lebih dahulu (bila ada), lalu menyusun entri audit: nomor entri, waktu, orang dan perannya, tindakan, rincian, alasan, versi basis pengetahuan setelah tindakan itu, `sha` (sidik jari berkas versi baru, atau kosong), dan `prev` (sidik jari entri sebelumnya). Kemudian ia menghitung `hash` dari semua medan itu, yang ditulis dalam JSON kanonik dengan `jsonio.dumps` milik vcirs-core agar Java dan Python menghitung sidik jari yang sama, menambahkan entrinya, lalu menyimpan `audit.json` dan `state.json`. Versi yang berlaku tidak lain adalah versi entri terakhir.

Pengujian. `run_tests` menjalankan setiap kasus cornerstone dari setiap node yang tidak dihentikan dan memeriksa bahwa kasus itu mendapat konklusi node tersebut, lalu menjalankan setiap kasus emas dan memeriksa tingkat yang diharapkannya. Ia mengembalikan berapa kasus yang dicobanya dan satu baris yang mudah dibaca untuk setiap kegagalan.

Tindakan. Setiap tindakan memeriksa peran orangnya lebih dahulu (`role` melempar `TeamError`, yang oleh *runner* dicetak sebagai *refused*). `propose` memuat salinan segar dari versi yang berlaku, mengajarkan kasusnya pada salinan itu, menjalankan pengujian pada salinan itu, lalu mencatat usulannya sebagai *waiting for review* atau *rejected by tests*; versi yang berlaku tidak disentuh. `review` menolak peninjau yang membuat usulan itu, dan saat menyetujui **mengajarkan kasusnya lagi pada salinan segar dari versi yang berlaku saat itu**, karena perubahan lain mungkin sudah disetujui sejak usulan dibuat; ia menjalankan pengujian lagi dan baru setelah itu menyimpan versi berikutnya. `stop` menghentikan sebuah node pada salinan, menguji, lalu menyimpan versi baru; `rollback` menyalin berkas versi lama menjadi berkas baru. `consult`, `report`, dan `golden` tidak mengubah pengetahuan, sehingga mereka mencatat entri tetapi tidak menciptakan versi. Dan empat metode melihat ke belakang: `test`, `diff` (apa yang ditambahkan, dihentikan, atau dilanjutkan di antara dua versi), `blame` (siapa yang mengajarkan sebuah node, dari kasus apa, dalam usulan mana, disetujui oleh siapa), dan `verify`, yang menelusuri rantai dan menghitung ulang setiap sidik jari. `tree` mencetak versi mana pun sebagai pohon.

Kode 11.4 adalah versi Java-nya. Ia lebih panjang, terutama karena Java menuliskan dengan lengkap *map* yang di Python ditulis sebagai kamus.

Kode 11.4 `TeamKb.java`: basis pengetahuan tim dalam Java.

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.HexFormat;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.function.Supplier;

import vcirs.Case;
import vcirs.Conclusion;
import vcirs.Condition;
import vcirs.InferenceResult;
import vcirs.Json;
import vcirs.KnowledgeBase;
import vcirs.Node;
import vcirs.Report;
import vcirs.VcirsException;

/**
 * A clinic's triage knowledge base for a team: many experts, versions,
 * tests against cornerstone and golden cases, and an audit trail.
 * The folder of a team knowledge base holds:
 *   versions/v001.json ...  every version of the knowledge, never changed
 *   state.json              golden cases, reports and proposals
 *   audit.json              who did what, when and why; each entry is chained
 *                           to the one before it by a SHA-256 hash
 */
public class TeamKb {
    static final List<String> EXPERTS = List.of("doctor", "nurse", "midwife");
    static final Map<String, String> ACTIONS = Map.of(
        "RED", "Call the nurse or doctor to the patient now. "
            + "If no clinician is in the clinic, call 119.",
        "YELLOW", "Tell the nurse now; the patient is seen before the queue.",
        "GREEN", "Normal queue. The patient comes back to the desk at once "
            + "if anything gets worse.");
    static final String NO_MATCH =
            "No rule matched: ask the nurse to look at the patient now.";
```

Kode 11.4 (Lanjutan) *TeamKb.java: basis pengetahuan tim dalam Java.*

```
    /** A request the team knowledge base refuses, with the reason. */
    static class TeamError extends RuntimeException {
        TeamError(String why) {
            super(why);
        }
    }

    static String sha(String text) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            return HexFormat.of().formatHex(
                    md.digest(text.getBytes(StandardCharsets.UTF_8)));
        } catch (NoSuchAlgorithmException e) {
            throw new IllegalStateException(e);
        }
    }

    /** "a = b, c >= 3" -> [Condition, Condition]; "" -> []. */
    static List<Condition> parseList(String text) {
        List<Condition> out = new ArrayList<>();
        for (String p : text.split(", ")) {
            if (!p.isBlank()) {
                out.add(Condition.parse(p));
            }
        }
        return out;
    }

    record Taught(List<Condition> facts, String level) {
    }

    /** "fever = yes, age group = baby -> RED" -> (facts, "RED"). */
    static Taught parseCase(String text) {
        if (!text.contains("->")) {
            throw new TeamError("write a case as: facts -> RED, YELLOW or GREEN");
        }
        String[] parts = text.split("->");
        String level = parts[1].strip();
        if (!ACTIONS.containsKey(level)) {
            throw new TeamError("the urgency must be RED, YELLOW or GREEN");
        }
        return new Taught(parseList(parts[0].strip()), level);
    }
```

Kode 11.4 (Lanjutan) *TeamKb.java: basis pengetahuan tim dalam Java.*

```
    static String show(List<Condition> conds) {
        List<String> parts = new ArrayList<>();
        for (Condition c : conds) {
            parts.add(c.toString());
        }
        return String.join(", ", parts);
    }

    static String place(Node node) {
        return node.parent != null ? node.parent : "(top level)";
    }

    static InferenceResult infer(KnowledgeBase kb, List<Condition> facts) {
        return kb.infer(new Case(facts, List.of(), 100).known());
    }

    /** The urgency the knowledge base gives, or "none". */
    static String level(InferenceResult r) {
        return r.conclusions().isEmpty() ? "none" : r.conclusions().get(0).value;
    }

    /**
     * Ripple-down teaching: null if the answer is already right; otherwise
     * the new node, an exception of the node that answered (or a new
     * top-level node), made of the conditions the expert chose ("when").
     */
    static Node teach(KnowledgeBase kb, List<Condition> facts, String level,
                      List<Condition> when, String author) {
        InferenceResult r = infer(kb, facts);
        if (level(r).equals(level)) {
            return null;
        }
        Case c = new Case(facts, List.of(new Conclusion("urgency", level)), 100);
        String fired = r.paths.isEmpty() ? null : r.paths.get(0).last().id;
        int option = fired != null ? KnowledgeBase.BOTH : KnowledgeBase.TOP_LEVEL;
        return kb.refine(c, option, fired, when.isEmpty() ? null : when, author);
    }

    final Path folder;
    final Map<String, String> members = new LinkedHashMap<>();
    Supplier<String> clock = () -> "";
    Map<String, Object> state = new LinkedHashMap<>();
    List<Object> audit = new ArrayList<>();
```

Kode 11.4 (Lanjutan) *TeamKb.java: basis pengetahuan tim dalam Java.*

```
    @SuppressWarnings("unchecked")
    TeamKb(String folder, String teamFile) throws IOException {
        this.folder = Path.of(folder);
        Map<String, Object> team = (Map<String, Object>) Json.parse(
                Files.readString(Path.of(teamFile), StandardCharsets.UTF_8));
        for (Object o : (List<Object>) team.get("members")) {
            Map<String, Object> m = (Map<String, Object>) o;
            members.put((String) m.get("name"), (String) m.get("role"));
        }
        state.put("golden", new ArrayList<Object>());
        state.put("reports", new ArrayList<Object>());
        state.put("proposals", new ArrayList<Object>());
        Files.createDirectories(this.folder.resolve("versions"));
        if (Files.exists(this.folder.resolve("audit.json"))) {
            audit = (List<Object>) Json.parse(read("audit.json"));
            state = (Map<String, Object>) Json.parse(read("state.json"));
        }
    }

    // ----- files ------------------------------------------------------------
    void write(String name, String text) {
        try {
            Files.writeString(folder.resolve(name), text, StandardCharsets.UTF_8);
        } catch (IOException e) {
            throw new TeamError("cannot write " + name + ": " + e.getMessage());
        }
    }

    String read(String name) {
        try {
            return Files.readString(folder.resolve(name), StandardCharsets.UTF_8);
        } catch (IOException e) {
            throw new TeamError("cannot read " + name + ": " + e.getMessage());
        }
    }

    @SuppressWarnings("unchecked")
    static Map<String, Object> map(Object o) {
        return (Map<String, Object>) o;
    }

    @SuppressWarnings("unchecked")
    List<Object> list(String key) {
        return (List<Object>) state.get(key);
    }
```

Kode 11.4 (Lanjutan) *TeamKb.java: basis pengetahuan tim dalam Java.*

```
    /** The number of the current version (0 before the first). */
    int version() {
        return audit.isEmpty() ? 0
                : ((Number) map(audit.get(audit.size() - 1)).get("version"))
                        .intValue();
    }

    static String vname(int n) {
        return String.format("versions/v%03d.json", n);
    }

    /** A fresh copy of version n. */
    KnowledgeBase kb(int n) {
        return KnowledgeBase.fromJson(read(vname(n)));
    }

    KnowledgeBase kb() {
        return kb(version());
    }

    // ----- people, and the trail they leave --------------------------------
    String role(String who, List<String> allowed) {
        if (!members.containsKey(who)) {
            throw new TeamError(who + " is not a member of the team");
        }
        String role = members.get(who);
        if (allowed != null && !allowed.contains(role)) {
            throw new TeamError(who + " (" + role + ") may not do this");
        }
        return role;
    }

    /** Append a chained audit entry; save a new version if text. */
    void log(String who, String action, String detail, String reason,
             int version, String text) {
        if (!text.isEmpty()) {
            write(vname(version), text);
        }
        Map<String, Object> e = new LinkedHashMap<>();
        e.put("n", audit.size() + 1);
        e.put("time", clock.get());
        e.put("who", who);
        e.put("role", members.getOrDefault(who, ""));
        e.put("action", action);
        e.put("detail", detail);
        e.put("reason", reason);
```

Kode 11.4 (Lanjutan) *TeamKb.java: basis pengetahuan tim dalam Java.*

```
        e.put("version", version);
        e.put("sha", text.isEmpty() ? "" : sha(text));
        e.put("prev", audit.isEmpty() ? ""
                : map(audit.get(audit.size() - 1)).get("hash"));
        e.put("hash", sha(Json.write(e)));
        audit.add(e);
        write("audit.json", Json.write(audit) + "\n");
        write("state.json", Json.write(state) + "\n");
    }

    // ----- tests -------------------------------------------------------
    int lastCount;                    // how many cases the last test run tried

    /**
     * Every node's cornerstone case must still get its node's answer, and
     * every golden case its expected answer. Returns the failures; the
     * number of cases tried is left in lastCount.
     */
    List<String> runTests(KnowledgeBase kb) {
        List<String> fails = new ArrayList<>();
        int count = 0;
        for (Node node : kb.nodes.values()) {
            if (node.stopped || node.cornerstone == null) {
                continue;
            }
            count++;
            String want = node.conclusions.get(0).value;
            String got = level(infer(kb, node.cornerstone.facts));
            if (!got.equals(want)) {
                fails.add("cornerstone of " + node.id + " ("
                        + show(node.cornerstone.facts) + "): expected " + want
                        + ", got " + got);
            }
        }
        for (Object o : list("golden")) {
            Map<String, Object> g = map(o);
            count++;
            String facts = String.join(", ", strings(g.get("facts")));
            String got = level(infer(kb, parseList(facts)));
            if (!got.equals(g.get("expected"))) {
                fails.add("golden " + g.get("id") + " (" + facts
                        + "): expected " + g.get("expected") + ", got " + got);
            }
        }
        lastCount = count;
        return fails;
```

Kode 11.4 (Lanjutan) *TeamKb.java: basis pengetahuan tim dalam Java.*

```
    }

    @SuppressWarnings("unchecked")
    static List<String> strings(Object o) {
        List<String> out = new ArrayList<>();
        for (Object x : (List<Object>) o) {
            out.add((String) x);
        }
        return out;
    }

    // ----- what the team does -----------------------------------------------
    @SuppressWarnings("unchecked")
    List<String> create(String who, String seedFile, String reason)
            throws IOException {
        role(who, List.of("doctor"));
        if (!audit.isEmpty()) {
            throw new TeamError("the knowledge base already exists");
        }
        Map<String, Object> seed = map(Json.parse(Files.readString(
                Path.of(seedFile), StandardCharsets.UTF_8)));
        KnowledgeBase kb = new KnowledgeBase((String) seed.get("name"),
                (String) seed.get("description"));
        for (Object o : (List<Object>) seed.get("questions")) {
            List<Object> q = (List<Object>) o;
            kb.declare((String) q.get(0), (String) q.get(1), strings(q.get(2)));
        }
        for (Object o : (List<Object>) seed.get("steps")) {
            Map<String, Object> step = map(o);
            teach(kb, parseList(String.join(", ", strings(step.get("facts")))),
                    (String) step.get("urgency"),
                    parseList(String.join(", ", strings(step.get("when")))), who);
        }
        log(who, "create", kb.summary(), reason, 1, kb.toJson());
        return List.of("v1 created: " + kb.summary());
    }
```

Kode 11.4 (Lanjutan) *TeamKb.java: basis pengetahuan tim dalam Java.*

```
    List<String> consult(String who, String factsText) {
        role(who, null);
        InferenceResult r = infer(kb(), parseList(factsText));
        String got = level(r);
        List<String> out = new ArrayList<>();
        out.add(who + " asks about: " + factsText);
        if (got.equals("none")) {
            out.add("  urgency: none  (" + NO_MATCH + ")");
            if (!r.paths.isEmpty()) {
                out.add("  the last matching rule, " + r.paths.get(0).last().id
                        + ", is stopped");
            }
            return out;
        }
        out.add("  urgency: " + got + "  (rule " + r.paths.get(0).last().id
                + ", version v" + version() + ")");
        out.add("  what to do: " + ACTIONS.get(got));
        out.add("  why:");
        for (String line : r.how()) {
            out.add("    " + line);
        }
        return out;
    }

    List<String> golden(String who, String caseText, String reason) {
        role(who, EXPERTS);
        Taught t = parseCase(caseText);
        String gid = "G" + (list("golden").size() + 1);
        List<String> facts = new ArrayList<>();
        for (Condition c : t.facts()) {
            facts.add(c.toString());
        }
        Map<String, Object> g = new LinkedHashMap<>();
        g.put("id", gid);
        g.put("facts", facts);
        g.put("expected", t.level());
        g.put("by", who);
        g.put("reason", reason);
        list("golden").add(g);
        String got = level(infer(kb(), t.facts()));
        log(who, "golden", gid + ": " + caseText, reason, version(), "");
        return List.of(gid + " added (" + (got.equals(t.level()) ? "passes now"
                : "FAILS now: gives " + got) + ")");
    }
```

Kode 11.4 (Lanjutan) *TeamKb.java: basis pengetahuan tim dalam Java.*

```
    List<String> report(String who, String factsText, String note) {
        role(who, null);
        String rid = "R" + (list("reports").size() + 1);
        String got = level(infer(kb(), parseList(factsText)));
        Map<String, Object> r = new LinkedHashMap<>();
        r.put("id", rid);
        r.put("by", who);
        r.put("facts", factsText);
        r.put("answer", got);
        r.put("note", note);
        list("reports").add(r);
        log(who, "report", rid + ": " + factsText + " (answer " + got + ")",
                note, version(), "");
        return List.of(rid + " recorded for the experts to read");
    }

    List<String> propose(String who, String caseText, String whenText,
                         String reason) {
        role(who, EXPERTS);
        Taught t = parseCase(caseText);
        List<Condition> when = parseList(whenText);
        String pid = "P" + (list("proposals").size() + 1);
        Map<String, Object> p = new LinkedHashMap<>();
        p.put("id", pid);
        p.put("by", who);
        p.put("case", caseText);
        p.put("when", whenText);
        p.put("reason", reason);
        p.put("status", "");
        p.put("base", version());
        p.put("node", "");
        p.put("reviewer", "");
        list("proposals").add(p);
        KnowledgeBase kb = kb();
        Node node = teach(kb, t.facts(), t.level(), when, who);
        List<String> out = new ArrayList<>();
        if (node == null) {
            p.put("status", "already known");
            out.add(pid + ": the knowledge base already answers " + t.level());
        } else {
            List<String> fails = runTests(kb);
            p.put("node", node.id);
            out.add(pid + ": would add " + node + " under " + place(node));
            if (!fails.isEmpty()) {
                p.put("status", "rejected by tests");
                out.add("  REJECTED: " + fails.size() + " of " + lastCount
```

Kode 11.4 (Lanjutan) *TeamKb.java: basis pengetahuan tim dalam Java.*

```
                    + " test cases would change:");
            for (String f : fails) {
                out.add("    " + f);
            }
        } else {
            p.put("status", "waiting for review");
            out.add("  all " + lastCount + " test cases still pass; "
                    + "waiting for another expert to review");
        }
    }
    log(who, "propose", pid + " (" + p.get("status") + "): " + caseText,
            reason, version(), "");
    return out;
}

List<String> review(String who, String pid, String decision,
                    String reason) {
    role(who, EXPERTS);
    Map<String, Object> p = null;
    for (Object o : list("proposals")) {
        if (map(o).get("id").equals(pid)) {
            p = map(o);
        }
    }
    if (p == null || !p.get("status").equals("waiting for review")) {
        throw new TeamError(pid + " is not waiting for review");
    }
    if (p.get("by").equals(who)) {
        throw new TeamError("nobody may review their own proposal");
    }
    p.put("reviewer", who);
    if (!decision.equals("approve")) {
        p.put("status", "rejected by " + who);
        log(who, "reject", pid, reason, version(), "");
        return List.of(pid + " rejected");
    }
    Taught t = parseCase((String) p.get("case"));
    KnowledgeBase kb = kb();              // the knowledge may have moved on
    Node node = teach(kb, t.facts(), t.level(),
            parseList((String) p.get("when")), (String) p.get("by"));
    List<String> fails = node == null ? List.of() : runTests(kb);
    if (node == null || !fails.isEmpty()) {
        p.put("status", "rejected by tests");
        log(who, "reject", pid + " (tests)", reason, version(), "");
        return List.of(pid + " no longer passes the tests; it was not "
                + "applied");
```

Kode 11.4 (Lanjutan) *TeamKb.java: basis pengetahuan tim dalam Java.*

```
        }
        p.put("status", "approved");
        p.put("node", node.id);
        int v = version() + 1;
        log(who, "approve", pid + ": " + node, reason, v, kb.toJson());
        return List.of(pid + " approved: v" + v + " adds " + node.id + " ("
                + lastCount + " test cases pass)");
    }

    List<String> stop(String who, String nodeId, String reason) {
        role(who, List.of("doctor"));
        KnowledgeBase kb = kb();
        try {
            kb.stop(nodeId, who);
        } catch (VcirsException e) {
            throw new TeamError(e.getMessage());
        }
        List<String> fails = runTests(kb);
        if (!fails.isEmpty()) {
            throw new TeamError("stopping " + nodeId + " breaks: "
                    + String.join("; ", fails));
        }
        int v = version() + 1;
        log(who, "stop", nodeId, reason, v, kb.toJson());
        return List.of("v" + v + ": " + nodeId + " stopped (" + lastCount
                + " test cases pass)");
    }

    List<String> rollback(String who, String vtext, String reason) {
        role(who, List.of("doctor"));
        int n = Integer.parseInt(vtext.replaceFirst("^v+", ""));
        if (n < 1 || n >= version()) {
            throw new TeamError("roll back to an earlier version, v1 to v"
                    + (version() - 1));
        }
        String text = read(vname(n));
        int v = version() + 1;
        log(who, "rollback", "back to v" + n, reason, v, text);
        return List.of("v" + v + " is a copy of v" + n
                + "; nothing in between is lost from the trail");
    }
```

Kode 11.4 (Lanjutan) *TeamKb.java: basis pengetahuan tim dalam Java.*

```
    // ----- looking back ----------------------------------------------------
    List<String> test() {
        List<String> fails = runTests(kb());
        List<String> out = new ArrayList<>();
        if (fails.isEmpty()) {
            out.add("v" + version() + ": all " + lastCount + " test cases pass");
            return out;
        }
        out.add("v" + version() + ": " + fails.size() + " of " + lastCount
                + " test cases fail:");
        for (String f : fails) {
            out.add("  " + f);
        }
        return out;
    }

    List<String> diff(String a, String b) {
        KnowledgeBase old = kb(Integer.parseInt(a.replaceFirst("^v+", "")));
        KnowledgeBase now = kb(Integer.parseInt(b.replaceFirst("^v+", "")));
        List<String> out = new ArrayList<>();
        out.add("from " + a + " to " + b + ":");
        for (Node n : now.nodes.values()) {
            Node o = old.nodes.get(n.id);
            if (o == null) {
                out.add("  added   " + n);
            } else if (n.stopped && !o.stopped) {
                out.add("  stopped " + n.id);
            }
        }
        for (Node o : old.nodes.values()) {
            Node n = now.nodes.get(o.id);
            if (n == null) {
                out.add("  gone    " + o);
            } else if (o.stopped && !n.stopped) {
                out.add("  resumed " + o.id);
            }
        }
        if (out.size() == 1) {
            out.add("  no change");
        }
        return out;
    }
```

Kode 11.4 (Lanjutan) *TeamKb.java: basis pengetahuan tim dalam Java.*

```
    List<String> tree(String vtext) {
        return Report.tree(kb(Integer.parseInt(vtext.replaceFirst("^v+", ""))));
    }

    List<String> blame(String nodeId) {
        KnowledgeBase kb = kb();
        if (!kb.nodes.containsKey(nodeId)) {
            throw new TeamError("no node " + nodeId + " in v" + version());
        }
        Node n = kb.nodes.get(nodeId);
        List<String> out = new ArrayList<>();
        out.add(n.toString());
        out.add("  under: " + place(n));
        out.add("  taught by: " + n.author);
        out.add("  cornerstone case: " + show(n.cornerstone.facts));
        for (Object o : list("proposals")) {
            Map<String, Object> p = map(o);
            if (p.get("node").equals(nodeId)
                    && p.get("status").equals("approved")) {
                out.add("  proposal " + p.get("id") + ": " + p.get("reason"));
                out.add("  approved by " + p.get("reviewer"));
            }
        }
        for (Object o : audit) {
            Map<String, Object> e = map(o);
            if (!e.get("sha").equals("") && ((String) e.get("detail"))
                    .contains(nodeId)) {
                out.add("  in the trail: #" + e.get("n") + " " + e.get("time")
                        + " " + e.get("action") + " by " + e.get("who")
                        + " -> v" + e.get("version"));
            }
        }
        return out;
    }

    List<String> log() {
        List<String> out = new ArrayList<>();
        for (Object o : audit) {
            Map<String, Object> e = map(o);
            out.add("#" + e.get("n") + " " + e.get("time") + " v"
                    + e.get("version") + " " + e.get("who") + " ("
                    + e.get("role") + ") " + e.get("action") + ": "
                    + e.get("detail"));
            out.add("    why: " + e.get("reason"));
        }
        return out;
```

Kode 11.4 (Lanjutan) *TeamKb.java: basis pengetahuan tim dalam Java.*

```
    }

    /** Recompute every hash of the trail and of every version file. */
    List<String> verify() {
        String prev = "";
        for (Object o : audit) {
            Map<String, Object> e = map(o);
            Map<String, Object> body = new LinkedHashMap<>(e);
            body.remove("hash");
            if (!e.get("prev").equals(prev)
                    || !sha(Json.write(body)).equals(e.get("hash"))) {
                return List.of("the trail is broken at entry #" + e.get("n"));
            }
            int v = ((Number) e.get("version")).intValue();
            if (!e.get("sha").equals("") && !sha(read(vname(v)))
                    .equals(e.get("sha"))) {
                return List.of("version v" + v + " was changed after entry #"
                        + e.get("n"));
            }
            prev = (String) e.get("hash");
        }
        return List.of("the trail is intact: " + audit.size() + " entries, "
                + version() + " versions, last hash "
                + prev.substring(0, Math.min(12, prev.length())));
    }
}
```

Tiga rincian versi Java layak disebut. Sidik jari SHA-256 berasal dari pustaka standar: `MessageDigest.getInstance("SHA-256")` menghitungnya dan `HexFormat.of().formatHex` menuliskannya sebagai 64 karakter heksadesimal, persis teks yang diberikan `hashlib.sha256(...).hexdigest()` milik Python. Keadaan (*state*) dan jejaknya disimpan sebagai objek `Map` dan `List` yang dihasilkan `Json.parse` milik vcirs-core, sehingga membaca dan menulisnya mempertahankan urutan setiap kunci, dan berkas-berkas yang ditulis kedua bahasa identik bita demi bita. Dan `Taught` adalah sebuah `record`, nilai dengan dua medan yang menyimpan fakta dan tingkat sebuah kasus yang sudah diurai.

11.5 Langkah 3: Penjalan Perintah

Tim memakai basis pengetahuan melalui perintah, satu per baris, dengan bagian-bagiannya dipisahkan oleh `|`. Perintah yang sama dapat diketik di papan ketik, satu per satu, atau disimpan dalam berkas dan dijalankan sekaligus; begitulah bab ini memutar ulang satu minggu penuh. **Kode 11.5** adalah penjalannya (*runner*).

Kode 11.5 `triage_team.py`: penjalan perintah, untuk skrip, perintah yang diketik, dan konsultasi pertanyaan demi pertanyaan.

```
"""The clinic team's triage knowledge base, driven by commands.

    python triage_team.py [--dir ../data/clinic] [--fresh] week.txt
    python triage_team.py [--dir ../data/clinic] -      (type commands)

One command per line, its parts separated by '|':
  time | 2026-10-05 07:30          pretend it is this time (for the book)
  create | who | seed.json | why   the first version (a doctor)
  consult | who | facts            the urgency for these facts
  ask | who                        the urgency, asking one question at a time
  report | who | facts | note      anyone: "this answer looked wrong"
  golden | who | facts -> LEVEL | why       an expert: a case that must hold
  propose | who | facts -> LEVEL | when | why   an expert: new knowledge
  review | who | P1 | approve or reject | why  another expert decides
  stop | who | node | why          a doctor: stop a node (a new version)
  rollback | who | v3 | why        a doctor: go back to an earlier version
  test, log, verify, diff | v1 | v2, blame | node, tree | v3
"""

import datetime
import os
import pathlib
import shutil
import sys

from team_kb import ACTIONS, NO_MATCH, TeamError, TeamKb
from vcirs import VcirsError

HERE = pathlib.Path(__file__).resolve().parents[1]


class Runner:
    def __init__(self, team, lines, echo):
        self.team = team
        self.lines = lines
        self.echo = echo
        self.now = None
        team.clock = self.time

    def time(self):
        if self.now:
            return self.now
        return datetime.datetime.now().strftime("%Y-%m-%d %H:%M")
```

Kode 11.5 (Lanjutan) *triage_team.py: penjalan perintah, untuk skrip, perintah yang diketik, dan konsultasi pertanyaan demi pertanyaan.*

```
    def read(self, prompt):
        print(prompt, end="")
        text = next(self.lines, "").strip()
        if self.echo:
            print(text)
        return text

    def ask(self, who):
        """A consultation with questions, as at the real front desk."""
        self.team.role(who)
        kb = self.team.kb()

        def asker(q):
            var = kb.variables.get(q.variable)
            text = var.question if var and var.question else q.variable
            return self.read(text + " (" + "/".join(q.choices) + ") > ") \
                or None                      # Enter = "I do not know"

        result = kb.infer({}, asker)
        got = result.conclusions()
        if not got:
            return ["urgency: none  (" + NO_MATCH + ")"]
        level = got[0].value
        return ["urgency: " + level + "  (rule " + result.paths[0].last().id
                + ")", "what to do: " + ACTIONS[level]]

    def run(self):
        t = self.team
        for line in self.lines:
            line = line.strip()
            if not line or line.startswith("#"):
                continue
            a = [p.strip() for p in line.split("|")]
            if a[0] == "time":
                self.now = a[1]
                continue
            print("> " + line)
            try:
                commands = {
                    "create": lambda: t.create(a[1], str(HERE / "data" / a[2]),
                                               a[3]),
                    "consult": lambda: t.consult(a[1], a[2]),
                    "ask": lambda: self.ask(a[1]),
                    "report": lambda: t.report(a[1], a[2], a[3]),
                    "golden": lambda: t.golden(a[1], a[2], a[3]),
                    "propose": lambda: t.propose(a[1], a[2], a[3], a[4]),
```

Kode 11.5 (Lanjutan) *triage_team.py: penjalan perintah, untuk skrip, perintah yang diketik, dan konsultasi pertanyaan demi pertanyaan.*

```
                    "review": lambda: t.review(a[1], a[2], a[3], a[4]),
                    "stop": lambda: t.stop(a[1], a[2], a[3]),
                    "rollback": lambda: t.rollback(a[1], a[2], a[3]),
                    "test": t.test, "log": t.log, "verify": t.verify,
                    "diff": lambda: t.diff(a[1], a[2]),
                    "blame": lambda: t.blame(a[1]),
                    "tree": lambda: t.tree(a[1]),
                }
                if a[0] not in commands:
                    raise TeamError("unknown command " + a[0])
                out = commands[a[0]]()
            except (TeamError, VcirsError) as e:
                out = ["refused: " + str(e)]
            except IndexError:
                out = ["refused: a part of the command is missing"]
            for text in out:
                print(text)


def main(args):
    folder = str(HERE / "data" / "clinic")
    if "--dir" in args:
        folder = args[args.index("--dir") + 1]
    if "--fresh" in args and os.path.exists(folder):
        shutil.rmtree(folder)                # start the week again
    source = args[-1] if args else "-"
    team = TeamKb(folder, str(HERE / "data" / "team.json"))
    if source == "-":
        Runner(team, iter(sys.stdin), False).run()
    else:
        with open(source, encoding="utf-8") as f:
            Runner(team, iter(f.read().splitlines()), True).run()


if __name__ == "__main__":
    main(sys.argv[1:])
```

`Runner.run` membaca setiap baris, melewati baris kosong dan komentar, memisahkan bagian-bagiannya, lalu memanggil metode `TeamKb` yang sesuai melalui tabel perintah kecil. Baris `time` menetapkan waktu pura-pura, dipakai hanya agar minggu dalam buku dapat diputar ulang dengan waktu yang sama; bila tidak ada, waktu sungguhan yang dipakai. Setiap perintah lain diulang setelah `>`, sehingga skrip yang diputar ulang terbaca seperti transkrip, dan setiap penolakan dicetak sebagai `refused:` beserta alasannya. Perintah `ask` adalah konsultasi yang sungguh-sungguh dipakai meja pendaftaran: mesin mengajukan pertanyaan-pertanyaannya satu per satu, sesuai urutan basis pengetahuan, dan jawabannya dibaca dari sumber yang sama dengan perintah — papan ketik, atau baris-baris berikutnya dari skrip. **Kode 11.6** adalah penjalan versi Java.

Kode 11.6 `TriageTeam.java`: penjalan perintah yang sama dalam Java.

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Comparator;
import java.util.Iterator;
import java.util.List;
import java.util.stream.Stream;

import vcirs.InferenceResult;
import vcirs.KnowledgeBase;
import vcirs.Variable;
import vcirs.VcirsException;

/**
 * The clinic team's triage knowledge base, driven by commands.
 * java TriageTeam [--dir ../data/clinic] [--fresh] ../data/week.txt
 * java TriageTeam [--dir ../data/clinic] -          (type commands)
 * One command per line, its parts separated by '|'; see triage_team.py
 * or the README for the list.
 */
public class TriageTeam {
    static final Path DATA = Path.of("..", "data");

    final TeamKb team;
    final Iterator<String> lines;
    final boolean echo;
    String now = null;

    TriageTeam(TeamKb team, Iterator<String> lines, boolean echo) {
        this.team = team;
        this.lines = lines;
        this.echo = echo;
        team.clock = this::time;
    }

    String time() {
        return now != null ? now : LocalDateTime.now().format(
                DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm"));
    }
```

Kode 11.6 (Lanjutan) *TriageTeam.java: penjalan perintah yang sama dalam Java.*

```
    String read(String prompt) {
        System.out.print(prompt);
        String text = lines.hasNext() ? lines.next().strip() : "";
        if (echo) {
            System.out.println(text);
        }
        return text;
    }

    /** A consultation with questions, as at the real front desk. */
    List<String> ask(String who) {
        team.role(who, null);
        KnowledgeBase kb = team.kb();
        InferenceResult r = kb.infer(new java.util.LinkedHashMap<>(), q -> {
            Variable v = kb.variables.get(q.variable);
            String text = v != null && !v.question.isEmpty() ? v.question
                    : q.variable;
            String a = read(text + " (" + String.join("/", q.choices) + ") > ");
            return a.isEmpty() ? null                // Enter = "I do not know"
                    : new vcirs.Fact(q.variable, a, 100, "user");
        });
        if (r.conclusions().isEmpty()) {
            return List.of("urgency: none  (" + TeamKb.NO_MATCH + ")");
        }
        String level = r.conclusions().get(0).value;
        return List.of("urgency: " + level + "  (rule "
                + r.paths.get(0).last().id + ")",
                "what to do: " + TeamKb.ACTIONS.get(level));
    }

    List<String> command(String[] a) throws IOException {
        switch (a[0]) {
            case "create": return team.create(a[1], DATA.resolve(a[2]).toString(),
                    a[3]);
            case "consult": return team.consult(a[1], a[2]);
            case "ask": return ask(a[1]);
            case "report": return team.report(a[1], a[2], a[3]);
            case "golden": return team.golden(a[1], a[2], a[3]);
            case "propose": return team.propose(a[1], a[2], a[3], a[4]);
            case "review": return team.review(a[1], a[2], a[3], a[4]);
            case "stop": return team.stop(a[1], a[2], a[3]);
            case "rollback": return team.rollback(a[1], a[2], a[3]);
            case "test": return team.test();
            case "log": return team.log();
            case "verify": return team.verify();
            case "diff": return team.diff(a[1], a[2]);
```

Kode 11.6 (Lanjutan) *TriageTeam.java: penjalan perintah yang sama dalam Java.*

```
            case "blame": return team.blame(a[1]);
            case "tree": return team.tree(a[1]);
            default: throw new TeamKb.TeamError("unknown command " + a[0]);
        }
    }

    void run() throws IOException {
        while (lines.hasNext()) {
            String line = lines.next().strip();
            if (line.isEmpty() || line.startsWith("#")) {
                continue;
            }
            String[] a = line.split("\\|", -1);
            for (int i = 0; i < a.length; i++) {
                a[i] = a[i].strip();
            }
            if (a[0].equals("time")) {
                now = a[1];
                continue;
            }
            System.out.println("> " + line);
            List<String> out;
            try {
                out = command(a);
            } catch (TeamKb.TeamError | VcirsException e) {
                out = List.of("refused: " + e.getMessage());
            } catch (ArrayIndexOutOfBoundsException e) {
                out = List.of("refused: a part of the command is missing");
            }
            for (String text : out) {
                System.out.println(text);
            }
        }
    }

    public static void main(String[] args) throws IOException {
        List<String> list = List.of(args);
        String folder = DATA.resolve("clinic").toString();
        if (list.contains("--dir")) {
            folder = list.get(list.indexOf("--dir") + 1);
        }
        if (list.contains("--fresh") && Files.exists(Path.of(folder))) {
            try (Stream<Path> all = Files.walk(Path.of(folder))) {
                for (Path p : all.sorted(Comparator.reverseOrder()).toList()) {
                    Files.delete(p);              // start the week again
                }
```

Kode 11.6 (Lanjutan) *TriageTeam.java: penjalan perintah yang sama dalam Java.*

```
            }
        }
        String source = list.isEmpty() ? "-" : list.get(list.size() - 1);
        TeamKb team = new TeamKb(folder, DATA.resolve("team.json").toString());
        if (source.equals("-")) {
            BufferedReader in = new BufferedReader(new InputStreamReader(
                    System.in, StandardCharsets.UTF_8));
            new TriageTeam(team, in.lines().iterator(), false).run();
        } else {
            new TriageTeam(team, Files.readAllLines(Path.of(source),
                    StandardCharsets.UTF_8).iterator(), true).run();
        }
    }
}
```

Untuk menjalankan minggu dalam buku dari awal:

```
cd Chapter11/java
javac -d out -sourcepath ../../vcirs-core/java/src *.java
java -cp out TriageTeam --fresh ../data/week.txt
```

atau `python triage_team.py --fresh ../data/week.txt` di folder `python`. `--fresh` menghapus folder `data/clinic` lebih dahulu. Keduanya mencetak 134 baris yang sama dan menulis berkas yang sama: ketujuh berkas versi, `state.json`, dan `audit.json` identik, bita demi bita, bahasa mana pun yang menjalankan minggu itu. Karena itu sebuah tim dapat memakai keduanya: laptop tua perawat dapat menjalankan Python dan komputer dokter menjalankan Java, pada folder bersama yang sama, bergantian.

11.6 Seminggu di Klinik

11.6.1 Senin: versi pertama, dan pasien yang dinilai keliru

Senin pukul 07.30 dr. Maya membuat basis pengetahuan dari poster klinik, yang ditulis sebagai delapan kasus dalam `clinic-seed.json`:

```
> create | dr. Maya | clinic-seed.json | First knowledge, from the clinic's triage poster.
v1 created: Front-desk triage: 11 variables, 8 nodes, 8 rules
```

Sebelas variabel — sepuluh pertanyaan dalam berkas benih ditambah `urgency`, variabel konklusinya — delapan node dan delapan rule. Lima node adalah tanda bahaya RED; node keenam menyatakan bahwa demam saja boleh menunggu (GREEN), dengan dua pengecualian: bayi di bawah tiga bulan adalah RED, dan demam tiga hari atau lebih adalah YELLOW. Lalu ia menulis dua kasus emas, dua keadaan yang paling ia inginkan tidak pernah dijawab keliru oleh tim:

```
> golden | dr. Maya | fever = yes, age group = baby -> RED | A baby with fever is always
seen at once.
G1 added (passes now)
> golden | dr. Maya | fever = yes, age group = adult, fever days = 1 -> GREEN | A short
fever alone waits.
G2 added (passes now)
```

Pukul 09.15 datang seorang perempuan yang sudah demam empat hari dan, katanya, gusinya berdarah. Sari berkonsultasi dengan basis pengetahuan memakai fakta-fakta yang ia punya:

```
> consult | Sari | fever = yes, age group = adult, fever days = 4, bleeding gums or nose =
yes
Sari asks about: fever = yes, age group = adult, fever days = 4, bleeding gums or nose =
yes
  urgency: YELLOW  (rule YELLOW#1, version v1)
  what to do: Tell the nurse now; the patient is seen before the queue.
  why:
    YELLOW#1 fired because:
      fever = yes  (given, CF 100)
      fever days >= 3  (fever days = 4, given, CF 100)
      so urgency = YELLOW  [rule CF 100, conclusion CF 100]
```

YELLOW: sebelum antrean, tetapi tidak saat itu juga. Ners Budi, yang diberi tahu Sari, langsung memeriksanya; perdarahan dengan demam yang sudah berlangsung beberapa hari adalah tanda peringatan, dan ia merujuknya ke rumah sakit. Sari tidak dapat mengubah pengetahuan, tetapi ia dapat melakukan hal paling berguna yang dapat dilakukan meja pendaftaran — ia **melaporkan**-nya:

```
> report | Sari | fever = yes, age group = adult, fever days = 4, bleeding gums or nose =
yes | Ners Budi saw her at once and sent her to the hospital.
R1 recorded for the experts to read
```

Pukul 13.00 Ners Budi mengubah laporan itu menjadi usulan. Ia memberikan kasusnya, jawaban yang seharusnya (RED), satu fakta yang membuat perbedaan (`bleeding gums or nose = yes`, dipilih sebagai kondisi *when*), dan alasannya:

```
> propose | Ners Budi | fever = yes, age group = adult, fever days = 4, bleeding gums or
nose = yes -> RED | bleeding gums or nose = yes | Bleeding with a fever of 3 days or more
is a warning sign (report R1).
P1: would add RED#7: bleeding gums or nose = yes -> urgency = RED under YELLOW#1
  all 11 test cases still pass; waiting for another expert to review
```

Usulan itu dicoba pada salinan versi 1: basis pengetahuan menjawab YELLOW melalui `YELLOW#1`, sehingga node baru `RED#7` akan menjadi pengecualian `YELLOW#1`, dan kesebelas kasus uji — delapan cornerstone, dua kasus emas, dan kasus milik node baru itu sendiri — masih lolos. Usulan itu menunggu tinjauan. Budi, yang terburu-buru agar selesai sebelum pasien sorenya, mencoba menyetujuinya sendiri:

```
> review | Ners Budi | P1 | approve | Looks right to me.
refused: nobody may review their own proposal
```

Pukul 16.00 dr. Maya meninjaunya:

```
> review | dr. Maya | P1 | approve | Agreed; matches the health ministry's warning signs.
P1 approved: v2 adds RED#7 (11 test cases pass)
```

Versi 2 lahir. Inilah entri jejak yang mencatatnya, sebagaimana disimpan dalam `audit.json`:

```
{
  "n": 6,
  "time": "2026-10-05 16:00",
  "who": "dr. Maya",
  "role": "doctor",
  "action": "approve",
  "detail": "P1: RED#7: bleeding gums or nose = yes -> urgency = RED",
  "reason": "Agreed; matches the health ministry's warning signs.",
  "version": 2,
  "sha": "644af82d7a449a66a003d9f996c553e43c6a21739e265ed6c762818dd2b68c6e",
  "prev": "64e64a51a9835ba79a04497d7bc9554d58b449c6f29444417e8ba0591f183c02",
  "hash": "5ec6bfda2c7068ba12364f88f96ff7ca7836eb58d5eb1da21e408fa5ba4fe6e5"
}
```

Semua yang dibutuhkan pembaca di kemudian hari ada di sana: kapan, siapa, dalam peran apa, apa yang disetujui, alasan yang diberikan, versi yang dihasilkan, sidik jari `v002.json`, dan dua mata rantainya. Pada hari Selasa Budi menambahkan pasien dari laporan R1 sebagai kasus emas, sehingga keadaan ini terlindungi untuk seterusnya:

```
> golden | Ners Budi | fever = yes, age group = adult, fever days = 4, bleeding gums or
nose = yes -> RED | The case of report R1.
G3 added (passes now)
```

11.6.2 Rabu: usulan yang ditolak pengujian

Pada hari Rabu Budi mengusulkan agar anak-anak yang demam diperiksa lebih cepat. Karena memilih kondisi *when* dengan tergesa-gesa, ia memilih `fever = yes` — yang memang benar untuk kasus itu, tetapi bukan hal yang membuatnya berbeda:

```
> propose | Ners Budi | fever = yes, age group = child, fever days = 1 -> YELLOW | fever =
yes | Children with fever should be seen sooner.
P2: would add YELLOW#2: fever = yes -> urgency = YELLOW under GREEN#1
  REJECTED: 2 of 13 test cases would change:
    cornerstone of GREEN#1 (fever = yes, age group = adult): expected GREEN, got YELLOW
    golden G2 (fever = yes, age group = adult, fever days = 1): expected GREEN, got YELLOW
```

Pengujian menjelaskan masalahnya dengan bahasa tim sendiri. Sebagai pengecualian `GREEN#1` dengan kondisi `fever = yes`, node baru itu akan menangkap *setiap* pasien yang demam, termasuk orang dewasa dalam kasus cornerstone `GREEN#1` dan orang dewasa dalam kasus emas G2, yang jawabannya akan berubah dari GREEN menjadi YELLOW. Tidak ada yang disimpan. Sepuluh menit kemudian ia mengusulkan lagi dengan kondisi yang benar:

```
> propose | Ners Budi | fever = yes, age group = child, fever days = 1 -> YELLOW | age
group = child | Children with fever should be seen sooner.
P3: would add YELLOW#2: age group = child -> urgency = YELLOW under GREEN#1
  all 13 test cases still pass; waiting for another expert to review
> review | Bidan Ratna | P3 | approve | Fine for a trial week.
P3 approved: v3 adds YELLOW#2 (13 test cases pass)
```

Bidan Ratna menyetujuinya "untuk uji coba seminggu", dan versi 3 lahir. Sore harinya ia mengusulkan pelajarannya sendiri: setiap perdarahan saat hamil adalah RED. Ia memakai variabel yang belum pernah dipakai siapa pun, `bleeding in pregnancy`, dan tidak memberikan kondisi *when*, sehingga node baru dibangun dari seluruh kasus di tingkat atas:

```
> propose | Bidan Ratna | pregnant = yes, bleeding in pregnancy = yes -> RED | | Any
bleeding in pregnancy is seen at once.
P4: would add RED#8: pregnant = yes, bleeding in pregnancy = yes -> urgency = RED under
(top level)
  all 14 test cases still pass; waiting for another expert to review
> review | dr. Maya | P4 | approve | Agreed.
P4 approved: v4 adds RED#8 (14 test cases pass)
```

11.6.3 Kamis: meja pendaftaran bekerja

Pada hari Kamis seorang ibu hamil yang mengalami perdarahan datang ke meja, dan Sari berkonsultasi dengan dua fakta yang ia punya:

```
> consult | Sari | pregnant = yes, bleeding in pregnancy = yes
Sari asks about: pregnant = yes, bleeding in pregnancy = yes
  urgency: RED  (rule RED#8, version v4)
  what to do: Call the nurse or doctor to the patient now. If no clinician is in the
clinic, call 119.
  why:
    RED#8 fired because:
      pregnant = yes  (given, CF 100)
      bleeding in pregnancy = yes  (given, CF 100)
      so urgency = RED  [rule CF 100, conclusion CF 100]
```

Namun sebagian besar waktu, Sari tidak tahu lebih dahulu fakta mana yang penting. Ia memakai `ask`, dan basis pengetahuan bertanya menurut urutannya sendiri — tanda-tanda bahaya lebih dahulu, karena node-node itulah yang diajarkan pertama:

```
> ask | Sari
Is the patient awake and answering you normally? (yes/no) > yes
Is breathing hard, noisy or very fast, or can the patient not finish a sentence? (yes/no) >
no
Does the patient have chest pain or pressure? (yes/no) > no
Is one side of the face drooping, one arm weak, or speech slurred? (yes/no) > no
Is there bleeding that does not stop when pressed? (yes/no) > no
Does the patient have a fever? (yes/no) > yes
Is the patient a baby under 3 months, a child or an adult? (baby/child/adult) > child
For how many days has the fever lasted? (1/2/3/4/5) > 2
urgency: YELLOW  (rule YELLOW#2)
what to do: Tell the nurse now; the patient is seen before the queue.
```

Delapan pertanyaan untuk seorang anak yang demam dua hari, masing-masing dibutuhkan oleh sebuah node di sepanjang jalan: kelima tanda bahaya (masing-masing dapat langsung membuat jawabannya RED), demamnya, kelompok umurnya (bayi akan RED), dan jumlah harinya (tiga atau lebih akan YELLOW). Jawabannya YELLOW, dari node yang disetujui pada hari Rabu.

11.6.4 Jumat: kekeliruan, rollback, dan pelajaran

Dalam rapat hari Jumat tim sepakat bahwa uji coba `YELLOW#2` tidak berhasil: antrean YELLOW penuh dengan anak-anak yang sebenarnya baik-baik saja. Anak yang demam tanpa tanda bahaya tidak perlu didahulukan dari semua orang. dr. Maya menghentikan node itu — atau lebih tepatnya, ia bermaksud begitu:

```
> stop | dr. Maya | YELLOW#1 | Friday meeting: a child with fever and no danger sign waits
in the normal queue.
v5: YELLOW#1 stopped (13 test cases pass)
```

Ia mengetik `YELLOW#1`, node untuk demam tiga hari atau lebih, alih-alih `YELLOW#2`. Ketiga belas kasus uji masih lolos, karena menghentikan sebuah node melewati cornerstone node itu sendiri dan tidak ada kasus emas yang mencakup demam lama tanpa perdarahan. Beberapa menit kemudian Sari mendaftar pasien persis seperti itu:

```
> consult | Sari | fever = yes, age group = adult, fever days = 4, bleeding gums or nose =
no
Sari asks about: fever = yes, age group = adult, fever days = 4, bleeding gums or nose = no
  urgency: none  (No rule matched: ask the nurse to look at the patient now.)
  the last matching rule, YELLOW#1, is stopped
```

Tidak ada yang berbahaya terjadi — jawaban “none” mengirim perawat untuk melihat pasien, seperti kata **Tabel 11.1** — tetapi jawaban itu jelas bukan yang diinginkan tim, dan pesannya menyebut node yang dihentikan. dr. Maya melakukan rollback ke pengetahuan hari Kamis, menghentikan node yang benar, dan menulis kasus emas agar kekeliruan yang sama tertangkap lain kali:

```
> rollback | dr. Maya | v4 | I stopped the wrong node; back to Thursday's knowledge.
v6 is a copy of v4; nothing in between is lost from the trail
> stop | dr. Maya | YELLOW#2 | Friday meeting: a child with fever and no danger sign waits
in the normal queue.
v7: YELLOW#2 stopped (13 test cases pass)
> golden | dr. Maya | fever = yes, age group = adult, fever days = 4, bleeding gums or nose
= no -> YELLOW | So that stopping YELLOW#1 by mistake is caught next time.
G4 added (passes now)
```

Gambar 11.3 memperlihatkan minggu itu sebagai sebaris versi.



Gambar 11.3 Tujuh versi dalam seminggu. P2 ditolak pengujian dan tidak pernah menjadi versi; versi 5 menghentikan node yang salah, dan versi 6 adalah salinan versi 4.

11.6.5 Apa yang diingat berkas-berkasnya

Setiap usulan tetap tersimpan di `state.json`, apa pun nasibnya. Inilah usulan yang ditolak pada hari Rabu:

```
{
  "id": "P2",
  "by": "Ners Budi",
  "case": "fever = yes, age group = child, fever days = 1 -> YELLOW",
  "when": "fever = yes",
  "reason": "Children with fever should be seen sooner.",
  "status": "rejected by tests",
  "base": 2,
  "node": "YELLOW#2",
  "reviewer": ""
}
```

Catatan itu menyimpan kasusnya persis seperti yang ditulis Budi, kondisi yang dipilihnya, alasannya, versi tempat usulan itu dicoba (`base`), node yang akan diciptakannya, dan nasibnya. Usulan yang ditolak pengujian bukan kegagalan yang perlu disembunyikan; ia bukti bahwa pengujiannya bekerja, dan pengingat akan gagasan yang pernah dianggap seseorang layak diusulkan. Ketika tim membicarakan lagi anak-anak yang demam bulan depan, catatan ini, bersama empat entri jejak di sekitarnya, adalah titik awal pembicaraannya.

Berkas-berkas versinya adalah berkas basis pengetahuan biasa, dengan format yang sama dengan `wifi-kb.json` di Bab 3. Berkas mana pun dapat dibuka dengan alat baris perintah vcirs-core (`python -m vcirs show versions/v004.json` di folder `python` milik mesin, atau `java -cp out vcirs.Cli show ...`), atau disalin ke program lain — penasihat noda di Bab 10 dapat memuatnya ke dalam basis datanya. Basis pengetahuan tim menambahkan aturan di sekeliling pengetahuan, tidak pernah format baru untuknya.

11.7 Menengok ke Belakang

Di akhir minggu tim menengok ke belakang dengan lima perintah yang membaca catatan. Pertama, pengujian pada versi yang berlaku:

```
> test
v7: all 14 test cases pass
```

Empat belas kasus: delapan cornerstone dari benih, `RED#7` dan `RED#8` dari usulan-usulan yang disetujui minggu ini, dan empat kasus emas; `YELLOW#2`, yang dihentikan, tidak diuji. Berikutnya, apa yang berubah antara hari Kamis dan sekarang — rollback dan penghentian kedua, dilihat dari luar, adalah satu perubahan yang bersih:

```
> diff | v4 | v7
from v4 to v7:
  stopped YELLOW#2
```

Lalu pertanyaan yang cepat atau lambat didengar setiap basis pengetahuan: *mengapa aturan ini ada di sini?*

```
> blame | RED#7
RED#7: bleeding gums or nose = yes -> urgency = RED
  under: YELLOW#1
  taught by: Ners Budi
  cornerstone case: fever = yes, age group = adult, fever days = 4, bleeding gums or nose =
yes
  proposal P1: Bleeding with a fever of 3 days or more is a warning sign (report R1).
  approved by dr. Maya
  in the trail: #6 2026-10-05 16:00 approve by dr. Maya -> v2
```

Dalam satu layar: node-nya, tempatnya (pengecualian node demam tiga hari), siapa yang mengajarkannya, kasus yang membenarkannya, usulan beserta alasannya, peninjaunya, dan entri jejak dengan waktu dan versinya. Entri rollback pukul 15.10 (#14) menciptakan versi yang juga memuat `RED#7`, tetapi rinciannya tidak menyebut node itu, sehingga `blame` hanya menampilkan entri yang memperkenalkannya. Pengetahuan sebagaimana adanya sekarang:

```
> tree | v7
(root)
+- RED#1: awake = no -> urgency = RED
+- RED#2: hard breathing = yes -> urgency = RED
+- RED#3: chest pain = yes -> urgency = RED
+- RED#4: face or arm weak = yes -> urgency = RED
+- RED#5: heavy bleeding = yes -> urgency = RED
+- GREEN#1: fever = yes -> urgency = GREEN
|  +- RED#6: age group = baby -> urgency = RED
|  +- YELLOW#1: fever days >= 3 -> urgency = YELLOW
|  |  +- RED#7: bleeding gums or nose = yes -> urgency = RED
|  +- YELLOW#2: age group = child -> urgency = YELLOW [stopped]
+- RED#8: pregnant = yes, bleeding in pregnancy = yes -> urgency = RED
```

Seluruh jejak terbaca seperti buku harian minggu itu:

```
> log
#1 2026-10-05 07:30 v1 dr. Maya (doctor) create: Front-desk triage: 11 variables, 8 nodes,
8 rules
    why: First knowledge, from the clinic's triage poster.
#2 2026-10-05 07:40 v1 dr. Maya (doctor) golden: G1: fever = yes, age group = baby -> RED
    why: A baby with fever is always seen at once.
#3 2026-10-05 07:40 v1 dr. Maya (doctor) golden: G2: fever = yes, age group = adult, fever
days = 1 -> GREEN
    why: A short fever alone waits.
#4 2026-10-05 09:20 v1 Sari (front desk) report: R1: fever = yes, age group = adult, fever
days = 4, bleeding gums or nose = yes (answer YELLOW)
    why: Ners Budi saw her at once and sent her to the hospital.
#5 2026-10-05 13:00 v1 Ners Budi (nurse) propose: P1 (waiting for review): fever = yes, age
group = adult, fever days = 4, bleeding gums or nose = yes -> RED
    why: Bleeding with a fever of 3 days or more is a warning sign (report R1).
#6 2026-10-05 16:00 v2 dr. Maya (doctor) approve: P1: RED#7: bleeding gums or nose = yes ->
urgency = RED
    why: Agreed; matches the health ministry's warning signs.
#7 2026-10-06 10:00 v2 Ners Budi (nurse) golden: G3: fever = yes, age group = adult, fever
days = 4, bleeding gums or nose = yes -> RED
    why: The case of report R1.
#8 2026-10-07 08:30 v2 Ners Budi (nurse) propose: P2 (rejected by tests): fever = yes, age
group = child, fever days = 1 -> YELLOW
    why: Children with fever should be seen sooner.
#9 2026-10-07 08:40 v2 Ners Budi (nurse) propose: P3 (waiting for review): fever = yes, age
group = child, fever days = 1 -> YELLOW
    why: Children with fever should be seen sooner.
#10 2026-10-07 11:00 v3 Bidan Ratna (midwife) approve: P3: YELLOW#2: age group = child ->
urgency = YELLOW
    why: Fine for a trial week.
#11 2026-10-07 14:00 v3 Bidan Ratna (midwife) propose: P4 (waiting for review): pregnant =
yes, bleeding in pregnancy = yes -> RED
    why: Any bleeding in pregnancy is seen at once.
#12 2026-10-08 09:00 v4 dr. Maya (doctor) approve: P4: RED#8: pregnant = yes, bleeding in
pregnancy = yes -> urgency = RED
    why: Agreed.
#13 2026-10-09 15:00 v5 dr. Maya (doctor) stop: YELLOW#1
    why: Friday meeting: a child with fever and no danger sign waits in the normal queue.
#14 2026-10-09 15:10 v6 dr. Maya (doctor) rollback: back to v4
    why: I stopped the wrong node; back to Thursday's knowledge.
#15 2026-10-09 15:12 v7 dr. Maya (doctor) stop: YELLOW#2
    why: Friday meeting: a child with fever and no danger sign waits in the normal queue.
#16 2026-10-09 15:15 v7 dr. Maya (doctor) golden: G4: fever = yes, age group = adult, fever
days = 4, bleeding gums or nose = no -> YELLOW
    why: So that stopping YELLOW#1 by mistake is caught next time.
```

Enam belas entri: konsultasi dan pertanyaan tidak ada di jejak, karena tidak mengubah apa pun; setiap perintah lain ada, termasuk usulan yang ditolak pengujian dan penghentian yang keliru. Akhirnya, rantainya diperiksa:

```
> verify
the trail is intact: 16 entries, 7 versions, last hash ab12b6ba663b
```

Untuk melihat apa yang dilindungi `verify`, salinlah foldernya, buka `audit.json` salinan itu di penyunting teks, dan ubah satu karakter pada alasan lama — misalnya titik koma dalam alasan dr. Maya di entri 6 menjadi koma. Lalu jalankan `verify` pada salinan itu:

```
> verify
the trail is broken at entry #6
```

Perubahan itu ditemukan pada entri tempat ia dibuat, dan setiap entri sesudahnya kini diragukan sampai berkas aslinya dipulihkan dari cadangan. Penolakan pun sama terlihatnya. Berikut empat perintah yang seharusnya tidak dapat dilakukan tim:

```
> review | Sari | P1 | approve | ok
refused: Sari (front desk) may not do this
> stop | Ners Budi | RED#7 | test
refused: Ners Budi (nurse) may not do this
> rollback | dr. Maya | v9 | x
refused: roll back to an earlier version, v1 to v6
> consult | Pak Joni | fever = yes
refused: Pak Joni is not a member of the team
```

11.7.1 Pengujian

`TeamKbTest.java` dan `test_team_kb.py` memutar ulang minggu itu di folder sementara dan memeriksa bahwa minggu itu berakhir pada versi 7 dengan keempat belas kasus lolos dan jejak yang utuh; bahwa perawat tidak dapat menghentikan node dan meja pendaftaran tidak dapat mengusulkan; bahwa `TeamKb` baru pada folder yang sama memuat versi yang sama, dengan `YELLOW#2` dihentikan; dan bahwa satu spasi yang ditambahkan ke `v002.json` ditemukan oleh `verify`. Pengujian Java mencetak:

```
ok: the week ends at v7
ok: all test cases pass
ok: the trail is intact
ok: only a doctor may stop a node
ok: the front desk may not propose
ok: the team knowledge base loads again
ok: tampering is found
all tests passed
```

11.8 Membuatnya Lebih Baik

Jawaban node yang dihentikan. Setelah hari Jumat, anak yang demam tanpa tanda bahaya mendapat jawaban *none*: jalur mesin berakhir pada `YELLOW#2`, yang dihentikan, dan node yang dihentikan tidak memberikan konklusi. Bagi meja pendaftaran itu aman — perawat melihatnya — tetapi itu bukan GREEN yang diinginkan rapat. Menghentikan adalah cara RDR untuk menghapus saran tanpa menghapus pengetahuan [4], [8]; itu bukan cara untuk mengganti saran. Soal Latihan 11.6 menanyakan bagaimana tim dapat memperoleh GREEN untuk anak-anak ini, dan penyelesaiannya menggabungkan alat-alat bab ini.

Usulan dari halaman web. Saat ini tim mengetik perintah. Halaman seperti di Bab 10, dengan basis pengetahuan di belakang sebuah server kecil, akan menampilkan usulan-usulan yang menunggu beserta hasil pengujiannya dan dua tombol, *approve* dan *reject*, serta kotak untuk alasannya. Seluruh logikanya ada di `TeamKb`; halaman hanya menambah kenyamanan. Dengan server, aturan "satu perubahan pada satu waktu" dari Subbab 11.3 juga mudah ditegakkan: sebuah kunci di sekeliling setiap metode yang menulis, seperti di Bab 10.

Tanda tangan, bukan hanya nama. Jejak mencatat nama yang diketik seseorang. Di klinik yang komputernya dipakai bersama, siapa pun dapat mengetik "dr. Maya". Langkah berikutnya adalah login untuk setiap anggota, dan lebih jauh lagi, **tanda tangan digital** pada setiap entri dengan kunci yang hanya dipegang orang yang menulisnya; dengan begitu jejak tidak hanya membuktikan bahwa tidak ada yang diubah, tetapi juga siapa yang menulis setiap entri.

Laporan yang menjadi kasus emas. Setiap laporan adalah kasus yang dianggap meragukan oleh meja pendaftaran. Ketika para pakar sepakat tentang jawaban benarnya, kasus itu sebaiknya menjadi kasus emas, seperti yang dilakukan Budi pada hari Selasa. Perintah yang mengubah

laporan R1 menjadi kasus emas G3 dalam satu langkah, dengan alasan diambil dari laporannya, membuat kebiasaan baik menjadi kebiasaan yang mudah.

Dua usulan sekaligus. Bila dua usulan menunggu bersamaan, yang disetujui kedua diajarkan pada versi yang sudah memuat yang pertama. `review` menanganinya dengan benar — ia mengajarkan kasusnya lagi pada versi yang berlaku dan menguji lagi — tetapi peninjau sebaiknya melihatnya. Mencetak "pengetahuan sudah berubah sejak usulan ini dibuat (v2 → v3)" saat peninjauan hanyalah perubahan dua baris.

Menghitung hari meja pendaftaran. Jejak mencatat laporan tetapi tidak mencatat konsultasi, dengan sengaja: konsultasi banyak jumlahnya dan tidak memuat apa pun yang perlu diaudit. Bila klinik ingin tahu berapa jawaban RED, YELLOW, dan GREEN yang diberikan meja setiap hari, sebuah penghitung per tingkat dan per hari sudah cukup, tanpa satu pun fakta tentang pasien mana pun.

> **JEBAKAN:** Pengujian hanya melindungi apa yang dimuatnya. Pada hari Jumat, menghentikan node yang salah lolos dari setiap pengujian karena tidak ada kasus emas yang mencakup demam lama tanpa perdarahan. Setelah setiap kejutan, tambahkan kasus yang seharusnya menangkapnya sebagai kasus emas; kumpulan uji basis pengetahuan tim harus tumbuh bersama setiap insiden, persis seperti pengetahuannya.

11.9 Memakainya Sungguh-Sungguh

Mulailah dari protokol yang sudah Anda miliki. Sebagian besar klinik punya protokol triase tertulis, sebuah poster, atau prosedur operasional baku. Tuliskan sebagai kasus benih, satu kasus per baris protokol, dengan tanda-tanda bahaya lebih dahulu; urutan node tingkat atas adalah urutan pertanyaan di meja. Mintalah dokter penanggung jawab klinik membuat versi pertama, dan tulis alasannya berupa nama dan tanggal protokol asalnya.

Sepakati kasus emas dalam rapat. Sepuluh sampai dua puluh kasus emas yang dipilih bersama lebih berharga daripada seratus yang ditulis satu orang. Masukkan kasus-kasus yang akan membuat setiap anggota malu bila dijawab keliru — bayi yang demam, nyeri dada, perdarahan saat hamil — dan kasus-kasus biasa yang harus tetap GREEN, agar antrean tidak dibanjiri YELLOW.

Jaga peran tetap jujur. Berikan meja pendaftaran hak untuk melapor, dan pakailah: laporan tidak memakan biaya dan sering menjadi tanda pertama bahwa pengetahuannya keliru. Simpan hak menghentikan dan rollback pada satu atau dua orang yang bertanggung jawab atas protokol. Dan pertahankan aturan bahwa tidak ada yang meninjau usulannya sendiri, meskipun merepotkan; aturan itulah yang menangkap kekeliruan hari Jumat yang lelah.

Menengok jejak setiap minggu. Seminggu sekali, bacalah entri-entri minggu itu bersama-sama: usulan yang disetujui, yang ditolak (usulan yang ditolak sering berarti ada perbedaan pendapat yang layak dibicarakan), laporan yang belum menjadi kasus emas, dan hasil `verify`. Hanya butuh sepuluh menit dan membuat seluruh tim tahu apa yang disuruh dilakukan meja pendaftaran.

Cadangan dan foldernya. Seluruh basis pengetahuan tim adalah satu folder berisi berkas-berkas teks kecil. Salinlah setiap hari ke tempat kedua — komputer lain, *flash disk*, penyimpanan awan — dan simpan salinan-salinan lama: bila suatu saat `verify` menemukan rantai yang putus, salinan lama itulah yang memulihkan kebenaran. Karena berkasnya berupa teks, folder bersama yang menyimpan versi berkas (seperti kebanyakan layanan penyimpanan awan) adalah rumah yang alami, asalkan hanya satu orang yang menjalankan perintah pada satu waktu.

Mengajari tim memakainya. Habiskan minggu pertama dengan basis pengetahuan di samping poster lama, bukan sebagai penggantinya. Sari berkonsultasi untuk setiap pasien dan membandingkan jawabannya dengan poster; setiap perbedaan adalah sebuah laporan atau kekeliruan dalam kasus benih, dan keduanya layak ditemukan sebelum ada yang mengandalkan program ini. Perlihatkan kepada setiap pakar hasil `blame` untuk satu node yang tidak diajarkannya: melihat kasus dan alasan di balik aturan orang lain itulah yang membuat sebuah tim memercayai basis pengetahuan bersama.

Ketika komputernya mati. Listrik padam atau laptop rusak tidak boleh membuat meja pendaftaran kehilangan aturannya. Cetaklah pohon versi yang berlaku (`tree | v7`) setelah setiap perubahan, beri tanggal, dan tempelkan di tempat poster lama. Pohonnya pendek — belasan baris untuk klinik ini — dan urutannya adalah urutan pertanyaan, sehingga halaman cetak itu menjadi poster yang setia dari pengetahuan yang berlaku, dan nomor versi di atasnya memberi tahu tim apakah poster itu masih mutakhir.

Apa yang tidak boleh terjadi pada basis pengetahuan ini. Ia tidak boleh tumbuh menjadi alat diagnosis: tidak ada node yang boleh menyimpulkan penyakit atau pengobatan, hanya tingkat kegawatan. Ia tidak boleh menggantikan tatapan tenaga klinis: setiap RED dan YELLOW mengirim seseorang ke pasien, begitu pula setiap *none*. Dan ia tidak boleh menyimpan identitas pasien: kasusnya berupa tanda-tanda, dan laporan membawa tanda-tanda dan sebuah catatan, tidak pernah nama.

Tim lain dengan bentuk yang sama. Folder yang sama berisi versi, pengujian, tinjauan, dan jejak yang dirangkai cocok untuk basis pengetahuan apa pun yang dirawat beberapa orang dan yang penting: komite pinjaman koperasi dengan pengetahuan fuzzy dari Bab 9, bengkel motor dengan beberapa mekanik (Bab 8), laundry dari Bab 10 setelah punya tiga cabang, sekolah yang para konselornya berbagi basis pengetahuan tentang siswa mana yang dipanggil lebih dahulu. Ganti tingkat-tingkatnya dan berkas benihnya; sisanya tetap.

11.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Berbagi berkas alih-alih basis pengetahuan. Dua orang yang menyimpan berkas yang sama kehilangan pelajaran tanpa jejak, seperti diperlihatkan Subbab 11.3. *Hindari:* setiap perubahan melewati satu program yang dimulai dari versi yang berlaku dan menulis versi berikutnya.

Memilih kondisi yang benar, tetapi bukan alasannya. `fever = yes` memang benar untuk anak yang dimaksud Budi, tetapi tidak membedakan anak itu dari siapa pun yang demam. *Hindari:* pilih dari daftar perbedaan fakta yang menjelaskan mengapa jawabannya berbeda; biarkan pengujian memperlihatkan apa lagi yang akan berubah.

Meninjau perubahan sendiri. Program menolaknya, tetapi kekeliruan yang sama dapat dilakukan dua orang yang saling menyetujui usulan tanpa membacanya. *Hindari:* peninjau membaca kasus, alasan, dan hasil pengujian, lalu menulis alasannya sendiri.

Percaya bahwa pengujian menangkap segalanya. Pengujian hanya menangkap perubahan pada kasus-kasus yang dimuatnya. *Hindari:* tambahkan kasus emas setelah setiap kejutan.

Menghentikan padahal maksudnya mengganti. Node yang dihentikan tidak memberikan jawaban; ia tidak memberikan jawaban induknya. *Hindari:* hentikan node bila jawaban yang benar adalah “tanyakan kepada seseorang”; bila jawaban lain yang diinginkan, ajarkan (atau lakukan rollback lalu ajarkan lagi).

Rollback terlalu jauh. Rollback mengembalikan seluruh basis pengetahuan ke versi lama, termasuk kehilangan setiap perubahan baik yang dibuat sesudahnya. *Hindari:* pakai `diff` sebelum rollback untuk melihat apa yang akan hilang; untuk membatalkan satu perubahan di antara perubahan-perubahan berikutnya, hentikan node-nya.

Menyunting berkas dengan tangan. Suntingan tangan pada berkas versi atau jejak memutus rantai, dan memang itulah tujuannya. *Hindari:* ubah pengetahuan hanya lewat perintah; bila sebuah berkas benar-benar rusak, pulihkan dari cadangan lalu jalankan `verify`.

Memasukkan pasien ke dalam jejak. Alasan seperti "Ibu Siti dari RT 05" menaruh data kesehatan pribadi dalam berkas yang dibaca seluruh tim selama bertahun-tahun. *Hindari:* tuliskan tanda-tanda dan keadaannya, jangan pernah identitasnya.

Membiarkan pengetahuan menamai penyakit. Node yang menyimpulkan sebuah penyakit mengundang meja pendaftaran untuk mendiagnosis. *Hindari:* konklusinya hanya RED, YELLOW, dan GREEN; selebihnya urusan tenaga klinis.

11.11 Ringkasan Bab

- Basis pengetahuan yang diajari beberapa pakar membutuhkan lebih dari Ripple Down Rules: tanpa aturan berbagi, pelajaran hilang ketika dua salinan disimpan (**Kode 11.1**, **Kode 11.2**), perubahan tidak ditinjau, dan tidak ada jalan kembali.
- Basis pengetahuan klinik memilah pasien menurut **kegawatan** — RED, YELLOW, GREEN — dengan "minta perawat melihat sekarang" bila tidak ada yang cocok; ia menanyakan tanda-tanda yang terlihat dan tidak pernah menamai penyakit (**Tabel 11.1**).
- Setiap perubahan adalah **usulan** yang dicoba pada salinan versi yang berlaku dan **diuji** terhadap setiap **kasus cornerstone** dan setiap **kasus emas**; usulan yang akan mengubah salah satunya ditolak. Pakar kedua **meninjau** setiap usulan yang lolos; hanya perubahan yang disetujui yang menjadi **versi** baru, yang tidak pernah disunting (**Gambar 11.1**, **Tabel 11.2**).
- **Rollback** menyalin versi lama menjadi versi baru, dan **penghentian** menghapus saran sebuah node sambil mempertahankan pengetahuannya; keduanya tidak memusnahkan apa pun (**Gambar 11.3**).
- **Jejak audit** mencatat siapa melakukan apa, kapan, mengapa, dan dengan hasil apa; setiap entri membawa sidik jari SHA-256 berkas versinya dan entri sebelumnya, sehingga `verify` menemukan setiap perubahan belakangan (**Gambar 11.2**).
- `TeamKb` (**Kode 11.3**, **Kode 11.4**) dan penjalan perintah (**Kode 11.5**, **Kode 11.6**) memutar ulang minggu klinik secara identik dalam Java dan Python: tinjauan atas usulan sendiri yang ditolak, usulan yang ditolak pengujian, tiga perubahan yang disetujui, penghentian yang keliru, rollback, dan jejak yang dapat dibaca `blame` dan `verify`.

11.12 Soal Latihan

11.1 Dengan pohon yang dicetak di Subbab 11.7, tentukan tingkat kegawatan dan pertanyaan-pertanyaan yang akan diajukan `ask`, berurutan, untuk: (a) orang dewasa yang sadar, dengan nyeri dada; (b) bayi yang demam; (c) orang dewasa yang demam lima hari tanpa perdarahan; (d) anak yang demam satu hari.

11.2 Mengapa `review` harus mengajarkan usulan itu lagi pada versi yang berlaku alih-alih menyimpan salinan yang diuji `propose`? Gambarkan urutan dua usulan di mana menyimpan salinan lama akan menghilangkan sebuah perubahan.

11.3 Dalam `run_tests`, node yang dihentikan dilewati. Jelaskan mengapa kasus cornerstone node seperti itu tidak dapat diharapkan mempertahankan jawabannya. Adakah pemeriksaan yang layak dilakukan untuk kasus cornerstone node yang dihentikan?

11.4 Ners Budi menambahkan kasus emas yang belum disepakati tim: anak yang demam satu hari harus YELLOW. Apa kata program, dan apa yang dilaporkan `test` sesudahnya? Apa yang sebaiknya dilakukan tim?

11.5 Buatlah salinan folder klinik, ubah satu karakter di `versions/v003.json`, lalu jalankan `verify`. Pesan apa yang Anda harapkan, dan mengapa pemeriksaan berkas versi penting meskipun jejaknya sendiri utuh?

11.6 Setelah hari Jumat, anak yang demam mendapat *none*. Tim menginginkan GREEN untuk anak-anak seperti itu. (a) Mengapa hal itu tidak dapat diajarkan sebagai pengecualian `YELLOW#2` dengan fakta `fever = yes, age group = child, fever days = 1`? (b) Gambarkan cara mencapai pengetahuan yang diinginkan dengan perintah-perintah bab ini, dan entri mana yang akan tampak di jejak.

11.7 Tuliskan perintah yang akan dipakai seorang bidan untuk mengusulkan bahwa ibu hamil yang demam adalah YELLOW, sebagai pengecualian `GREEN#1`. Kondisi *when* mana yang sebaiknya ia pilih, dan apa kata pengujiannya?

11.8 Tambahkan perintah `pending` yang mendaftar usulan-usulan yang menunggu tinjauan, beserta pengusulnya, kasusnya, dan alasannya. Metode apa yang Anda tambahkan ke `TeamKb`, dan baris mana di penjalan yang berubah?

11.9 Jejak membuat perubahan *terlihat*, bukan *mustahil*. Sebutkan dua cara seorang anggota yang tidak jujur tetap dapat menyalahgunakan sistem ini, dan apa yang membatasi masing-masing.

11.13 Penyelesaian Soal Latihan

Penyelesaian 11.1. (a) *awake* → yes, *hard breathing* → no, *chest pain* → yes: RED dari `RED#3`; tiga pertanyaan. (b) Kelima tanda bahaya, *fever* → yes, *age group* → baby: RED dari `RED#6`. (c) Kelima tanda bahaya, *fever* → yes, *age group* → adult, *fever days* → 5; `YELLOW#1` menyala, dan pengecualiannya `RED#7` menanyakan *bleeding gums or nose* → no: YELLOW. (d) Kelima tanda bahaya, *fever*, *age group* → child, *fever days* → 1: `RED#6` dan `YELLOW#1` gagal dan `YELLOW#2` menyala tetapi dihentikan, sehingga jawabannya *none*, "minta perawat melihat" (lihat Soal Latihan 11.6).

Penyelesaian 11.2. Misalkan P1 dan P2 sama-sama menunggu pada versi 1. P1 disetujui dan menjadi versi 2. Bila persetujuan P2 menyimpan salinan yang dibuat `propose` dari versi 1, versi 3 akan memuat node P2 tetapi tidak memuat node P1: perubahan P1 akan hilang, persis seperti di Subbab 11.3. Mengajarkan P2 lagi pada versi 2 menghasilkan versi 3 dengan kedua perubahan, dan menguji lagi memeriksa bahwa P2 tetap lolos dengan adanya P1 — keduanya dapat saling berpengaruh, misalnya bila P1 menambahkan pengecualian di bawah node yang dikoreksi P2.

Penyelesaian 11.3. Node yang dihentikan tetap menyala tetapi tidak memberikan konklusi, sehingga kasus cornerstone-nya sendiri kini mendapat *none* atau jawaban salah satu pengecualiannya; mengharapkan konklusi lamanya akan membuat setiap penghentian gagal dalam pengujiannya sendiri. Pemeriksaan yang berguna adalah kebalikannya: cornerstone node yang dihentikan *tidak* boleh mendapat konklusi yang dihentikan itu dari tempat lain, dan idealnya tim memutuskan jawaban apa yang seharusnya didapatnya lalu menuliskannya sebagai kasus emas — seperti yang dilakukan dr. Maya dengan G4.

Penyelesaian 11.4. Pada folder hari Jumat program mencetak:

```
> golden | Ners Budi | fever = yes, age group = child, fever days = 1 -> YELLOW | I still
think children should be seen sooner.
G5 added (FAILS now: gives none)
> test
v7: 1 of 15 test cases fail:
  golden G5 (fever = yes, age group = child, fever days = 1): expected YELLOW, got none
```

Kasus emas mencatat apa yang *diharapkan* tim; program memperingatkan bahwa pengetahuan yang berlaku tidak sependapat, dan sejak itu setiap `test` gagal sampai seseorang mengubah pengetahuannya atau kasus emasnya. Hal itu berguna: perbedaan pendapat kini terlihat, bernama, dan bertanggal. Namun kasus emas seharusnya disepakati tim; tim sebaiknya membicarakannya pada rapat berikutnya, lalu mengubah pengetahuannya (dengan usulan dan tinjauan) atau mencabut kasus itu. (Mencabut kasus emas bukan perintah dalam bab ini; menambahkan perintah `retire | who | G5 | why`, yang mempertahankan kasusnya tetapi menandainya sebagai dicabut di jejak, adalah perluasan yang baik.)

Penyelesaian 11.5. `verify` mencetak `version v3 was changed after entry #10`, entri yang menciptakan `v003.json` (P3 dari Budi yang disetujui Ratna). Jejaknya sendiri utuh, karena tidak ada yang menyentuh `audit.json`; tetapi pengetahuan yang akan dimuat meja pendaftaran bila seseorang melakukan rollback ke versi 3 bukan lagi pengetahuan yang dulu disetujui. Memeriksa berkas-berkas versi terhadap sidik jari di jejak itulah yang membuat versi-versinya dapat dipercaya, bukan hanya cerita tentangnya.

Penyelesaian 11.6. (a) Kasus baru itu akan memiliki fakta yang persis sama dengan kasus cornerstone `YELLOW#2`, sehingga daftar perbedaannya kosong: tidak ada fakta yang membedakan kasus baru dari kasus lama, dan mesin menolaknya (Bab 3). Kedua kasus mengatakan hal yang berbeda tentang keadaan yang sama — yang berarti perubahan pikiran, bukan pengecualian baru. (b) Cara yang bersih adalah membatalkan perubahan pikiran itu dan mengulang apa yang datang sesudahnya: `rollback | dr. Maya | v2 | ...` kembali ke pengetahuan sebelum P3 (anak YELLOW), dengan kehilangan P4; lalu Bidan Ratna mengusulkan P4 lagi, seseorang meninjaunya, dan usulan itu menjadi versi berikutnya. Anak yang demam lalu mendapat GREEN dari `GREEN#1`, seperti pada versi 2. Jejak memperlihatkan semuanya: rollback ke v2 beserta alasannya, usulan baru P5 dengan kasus yang sama dengan P4, tinjauannya, dan versi barunya. Sebelum rollback, `diff | v2 | v7` memperlihatkan persis apa yang perlu diajarkan lagi.

Penyelesaian 11.7. `propose | Bidan Ratna | fever = yes, age group = adult, pregnant = yes -> YELLOW | pregnant = yes | ...`. Basis pengetahuan menjawab GREEN melalui `GREEN#1`, daftar perbedaan terhadap cornerstone `GREEN#1` (`fever = yes, age group = adult`) hanya memuat `pregnant = yes`, dan itulah kondisi *when* yang tepat. Pengujiannya lolos: tidak ada kasus cornerstone atau kasus emas yang memuat `pregnant = yes` bersama demam, sehingga tidak ada jawaban yang berubah; usulan itu menunggu tinjauan. (Bila yang ditulisnya bayi, kasusnya sudah RED melalui `RED#6`, dan program akan menjawab “already known”.)

Penyelesaian 11.8. Di `team_kb.py`:

```
def pending(self):
    out = [p["id"] + " by " + p["by"] + ": " + p["case"] + "  (" + p["reason"]
           + ")" for p in self.state["proposals"]
           if p["status"] == "waiting for review"]
    return out or ["no proposal is waiting"]
```

dan di tabel perintah milik penjalan, `"pending": t.pending,`. Versi Java-nya adalah perulangan yang sama atas `list("proposals")` dalam metode `List<String> pending()`, ditambah satu `case`

`"pending": return team.pending();` di `command`. Tidak ada entri yang dicatat, karena tidak ada yang berubah.

Penyelesaian 11.9. Pertama, seorang anggota dapat mengetik nama anggota lain; jejak akan dengan setia mencatat sebuah kebohongan. Login untuk setiap anggota, dan idealnya tanda tangan digital, membatasi hal ini (Subbab 11.8). Kedua, anggota yang punya akses ke folder dapat mengganti seluruh folder, termasuk jejaknya, dengan folder rekaan yang rantainya konsisten di dalam dirinya sendiri. Cadangan harian yang disimpan di luar folder membatasi hal ini: rantai rekaan itu tidak akan cocok dengan sidik jari dalam salinan kemarin. Yang ketiga, lebih halus: dua anggota yang selalu saling menyetujui usulan tanpa membacanya menggagalkan aturan empat mata; pembacaan jejak mingguan, beserta alasan-alasannya, itulah yang membuat hal itu terlihat.

Daftar Pustaka

[1] World Health Organization, International Committee of the Red Cross, and Médecins Sans Frontières, “Interagency Integrated Triage Tool,” 2026. [Online]. Available: https://www.who.int/tools/triage

[2] Kementerian Kesehatan Republik Indonesia, “Akses darurat medis 119 kini bisa melalui SATUSEHAT Mobile.” [Online]. Available: https://kemkes.go.id/id/akses-darurat-medis-119-kini-bisa-melalui-satusehat-mobile

[3] Centers for Disease Control and Prevention, “Symptoms of dengue and testing,” Aug. 2025. [Online]. Available: https://www.cdc.gov/dengue/signs-symptoms/index.html

[4] P. Compton and B. H. Kang, *Ripple-Down Rules: The Alternative to Machine Learning*. Boca Raton, FL, USA: CRC Press, 2021.

[5] D. Richards, “Two decades of ripple down rules research,” *The Knowledge Engineering Review*, vol. 24, no. 2, pp. 159–184, 2009.

[6] M. M. I. Subakti, “A Variable-Centered Intelligent Rule System,” M.Sc.Eng. thesis, Dept. of Computer Science and Information Engineering, Taipei, Taiwan, Jun. 2005.

[7] I. Subakti, “Some revisions in VCIRS and cases reconstructing perspectives,” in *Proc. Information and Communication Technology Seminar (ICTS 2006)*, Surabaya, Indonesia, 2006.

[8] P. Compton, G. Edwards, B. Kang, L. Lazarus, R. Malor, T. Menzies, P. Preston, A. Srinivasan, and C. Sammut, “Ripple down rules: possibilities and limitations,” in *Proc. 6th AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1991, pp. 6.1–6.20.

[9] National Institute of Standards and Technology, *Secure Hash Standard (SHS)*, FIPS PUB 180-4. Gaithersburg, MD, USA, Aug. 2015.

[10] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*. Jakarta, Indonesia, 2022.

Bab 12

Belajar Rule dari Data: dari Pohon Keputusan ke Basis Pengetahuan VCIRS

Setelah menyelesaikan bab ini Anda akan mampu menjelaskan bagaimana sebuah program dapat mempelajari aturan dari tabel contoh; menumbuhkan **pohon keputusan** (*decision tree*) dengan metode CART, yang kita tulis sendiri dari nol dalam Java dan Python, dengan memilih setiap pertanyaan menurut **indeks Gini** yang dihitung secara tepat memakai bilangan bulat; menilai pohon hasil belajar secara jujur dengan pengujian **leave-one-out**; mengubah setiap lintasan pohon menjadi node basis pengetahuan VCIRS, lengkap dengan faktor kepastian dan kasus cornerstone yang nyata; serta membiarkan seorang pakar menyempurnakan pengetahuan hasil belajar itu kasus demi kasus, dengan setiap pelajaran diperiksa terhadap setiap contoh dalam data sebelum disimpan. Datanya adalah 101 hewan sungguhan dari sebuah kumpulan data terbuka; pakarnya seorang guru biologi yang klub pencinta alamnya bertemu hewan-hewan yang tidak pernah dilihat data itu. Di akhir bab kita memeriksa pohon kita terhadap scikit-learn, pustaka *machine learning* (pembelajaran mesin) yang paling luas dipakai di Python.

12.1 Klub Pencinta Alam Bu Lestari

Setiap Sabtu pagi Bu Lestari, guru biologi di sebuah SMP, membawa klub pencinta alamnya ke tempat yang ada hewannya: kebun binatang kota, hutan bakau di pinggir kota, kebun buah dan durian milik keluarga seorang murid, tempat kelelawar beterbangan pada sore hari dan cicak ada di setiap dinding. Anak-anak memotret apa yang mereka lihat dan, sesampainya di sekolah, mencoba menyebutkan hewan apa itu: mamalia, burung, reptil, amfibi, ikan, serangga, atau salah satu dari sekian banyak hewan tak bertulang belakang. Inilah latihan tertua dalam biologi — memilah makhluk hidup menurut apa yang dimilikinya dan apa yang dilakukannya — dan anak-anak menyukainya, asalkan ada yang dapat menjawab pertanyaan mereka: *Apakah kelelawar termasuk burung? Apakah penyu itu ikan, karena berenang dengan sirip? Mengapa paus bukan ikan?*

Bu Lestari ingin punya program kecil yang memainkan permainan ini bersama mereka: program itu bertanya tentang hewannya — apakah berbulu burung, apakah menyusui anaknya, apakah bertulang belakang — lalu menyebutkan golongannya, dan menjelaskan alasannya, seperti dokter Wi-Fi di Bab 3 menjelaskan sarannya. Ia dapat menulis sendiri aturan-aturannya, seperti aturan *Zookeeper* karya Winston ditulis dengan tangan, dan seperti tesis VCIRS memakainya untuk demonstrasi [1]. Namun ia punya sesuatu yang memudahkan pekerjaannya: sebuah tabel. Kumpulan data Zoo, yang diterbitkan di UCI Machine Learning Repository, menggambarkan 101 hewan dengan masing-masing 16 fakta — berambut, berbulu burung, bertelur, menyusui, dapat terbang, hidup di air, memburu hewan lain, bergigi, bertulang belakang, bernapas dengan udara, berbisa, bersirip, jumlah kaki, berekor, dipelihara manusia, lebih besar daripada kucing — beserta golongan masing-masing [2]. Dapatkah aturan-aturannya *dipelajari* dari tabel itu, lalu dikoreksi oleh Bu Lestari di bagian yang tidak dikatakan tabel?

Inilah pertanyaan bab ini, dan pertanyaan ini praktis, jauh melampaui urusan hewan. Toko, bengkel, laundry, dan klinik semuanya punya catatan: noda dan penanganan yang berhasil, gejala dan perbaikannya, keluhan dan bagian yang menanganinya. Di dalam catatan-catatan itu tersembunyi aturan. Pohon keputusan menemukannya dalam hitungan detik; VCIRS mengubahnya menjadi pengetahuan yang dapat dibaca, diperiksa, dan dikoreksi seorang pakar; dan koreksi itu dilakukan kasus demi kasus, persis seperti di seluruh buku ini.

12.2 Gagasannya dengan Kata-Kata Sederhana

12.2.1 Pengetahuan dari pakar, pengetahuan dari data

Ada dua cara membangun sistem berbasis aturan. Yang pertama adalah bertanya kepada pakar, seperti yang dilakukan setiap bab sejauh ini. Yang kedua adalah **pembelajaran mesin**: berikan kepada program banyak contoh yang jawabannya sudah diketahui, lalu biarkan ia menemukan aturan-aturan yang menghasilkan jawaban itu kembali. Masing-masing cara punya watak yang khas, dan tesis yang mendefinisikan VCIRS menggambarkannya dengan tepat. Pengetahuan yang dibangun dengan Ripple Down Rules cenderung *presisi*, karena setiap aturan dibuat untuk kasus nyata dan diperiksa terhadap kasus-kasus sebelumnya, tetapi ia hanya *mencakup* apa yang pernah dilihat pakarnya; metode pembelajaran mesin yang dilatih dengan banyak contoh mencakup lebih banyak keadaan, tetapi lebih sering keliru pada keadaan yang belum pernah dilihatnya [1]. Buku yang merangkum empat puluh tahun Ripple Down Rules bukan kebetulan diberi anak judul *The Alternative to Machine Learning* [3].

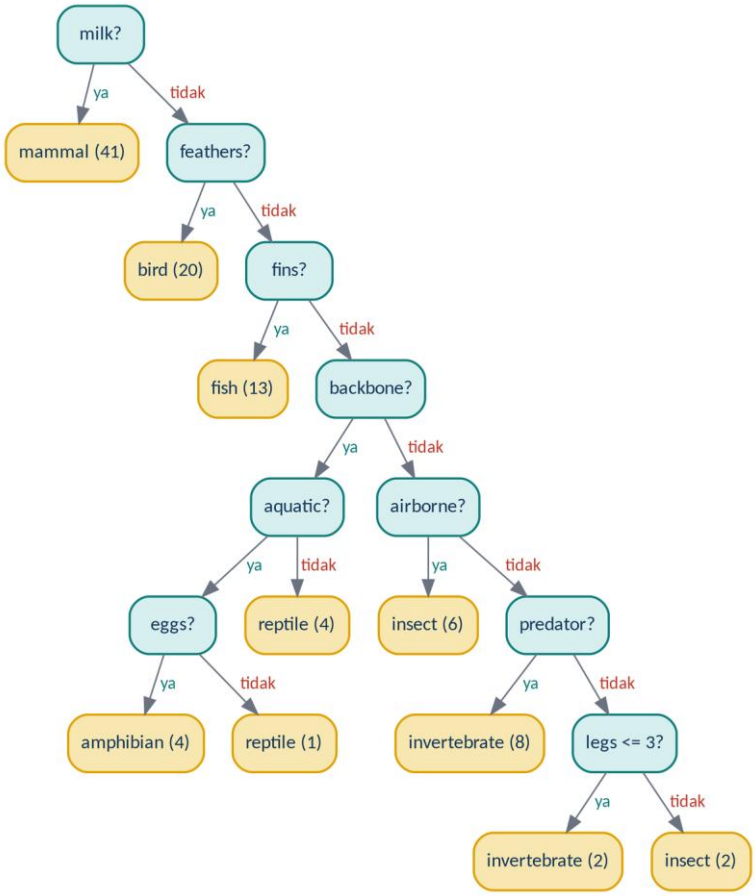
Namun keduanya bukan musuh. Tesis yang sama mengingatkan bahwa basis pengetahuan RDR pernah dipakai untuk menghasilkan data bagi pembelajaran mesin, dan induksi aturan pernah dipakai untuk menghasilkan pohon RDR [1], [4]. Bab ini melakukan yang kedua: belajar lebih dahulu dari data, untuk memperoleh cakupan yang luas dengan murah, lalu menyempurnakan bersama pakar, untuk memperoleh presisi di tempat yang penting. Pakar tidak perlu menulis aturan-aturan yang mudah — tabelnya sudah memuatnya — dan aturan hasil belajar tidak perlu dipercaya begitu saja, karena masing-masing menjadi node VCIRS dengan kasus cornerstone yang dapat diperiksa pakar, dan setiap koreksi adalah pengecualian RDR yang tidak dapat diam-diam merusak apa yang diajarkan data. **Gambar 12.1** memperlihatkan seluruh jalannya.



Gambar 12.1 Dari data ke basis pengetahuan yang sudah diperiksa. Sebuah pohon dipelajari dari tabel, setiap daunnya menjadi node VCIRS, sang guru menambahkan pengecualian, dan setiap pelajaran diperiksa terhadap semua hewan yang sudah dijawab basis pengetahuan.

12.2.2 Pohon keputusan adalah permainan tebak-tebakan

Pohon keputusan adalah permainan dua puluh pertanyaan yang dituliskan. Di puncaknya ada satu pertanyaan, misalnya *apakah ia menyusui anaknya?* Setiap jawaban mengarah ke pertanyaan lain atau ke sebuah kesimpulan. Untuk menggolongkan seekor hewan, mulailah dari puncak dan ikuti jawaban-jawabannya sampai tiba di kesimpulan, yang disebut **daun** (*leaf*). **Gambar 12.2** memperlihatkan pohon yang dipelajari program kita dari 101 hewan itu.



Gambar 12.2 Pohon keputusan yang dipelajari dari 101 hewan dalam data Zoo. Setiap pertanyaan punya cabang ya dan cabang tidak; setiap daun memberikan golongan dan, dalam kurung, berapa hewan dalam data yang sampai ke sana.

Bacalah dari puncak. Susu: ya berarti mamalia, keempat puluh satu ekor. Bila tidak, bulu burung: ya berarti burung. Bila tidak, sirip: ya berarti ikan. Bila tidak, tulang belakang memisahkan hewan bertulang belakang berdarah dingin (amfibi dan reptil, yang dibedakan oleh air dan telur) dari hewan tak bertulang belakang (serangga, yang terbang atau berkaki banyak, dan sisanya). Sepuluh daun, paling banyak tujuh pertanyaan. Seorang ahli biologi akan mengenali hampir setiap pertanyaannya; program menemukannya tanpa tahu apa pun tentang biologi, hanya dengan melihat tabel.

Perhatikan dua hal tentang pohon seperti ini. Pertama, setiap lintasan dari puncak ke daun adalah sebuah aturan: *IF milk = no AND feathers = yes THEN bird*. Pohon adalah cara ringkas menuliskan sekumpulan aturan yang tidak pernah tumpang-tindih, karena setiap hewan mengikuti tepat satu lintasan. Kedua, pohon adalah persis jenis struktur yang dianggap rapuh di Bab 3: untuk mengubah satu aturan Anda harus membentuk ulang pohonnya. Karena itulah kita tidak akan mempertahankan pohonnya. Kita akan mengubah lintasan-lintasannya menjadi node VCIRS dan membiarkan pakar menambahkan pengecualian, yang tidak pernah membentuk ulang apa pun.

12.2.3 Bagaimana pohon memilih pertanyaannya: kemurnian

Bagaimana program tahu bahwa susu harus ditanyakan lebih dahulu? Ia mencoba setiap pertanyaan yang dapat diajukannya dan mempertahankan pertanyaan yang membagi hewan-hewan menjadi kelompok-kelompok yang paling **murni**: kelompok yang sebagian besar anggotanya bergolongan sama. Metode yang kita pakai adalah **CART** (*Classification And Regression Trees*), yang diuraikan oleh Breiman, Friedman, Olshen, dan Stone [5], dan ukuran

kemurniannya adalah **indeks Gini**. Untuk sebuah kelompok yang anggotanya sebagian p_k termasuk golongan k,

$$G = 1 - \sum_k p_k^2 \tag{12.1}$$

Kelompok yang murni, seluruhnya satu golongan, memiliki $G = 0$. Kelompok yang tercampur rata antara dua golongan memiliki $G = 1 - (0.5^2 + 0.5^2) = 0.5$; dengan tujuh golongan nilainya dapat mendekati $1 - 7 \cdot (1/7)^2 = 0.857$. Anda dapat membaca G sebagai peluang: ambillah dua anggota kelompok secara acak, dengan pengembalian; G adalah peluang keduanya berasal dari golongan yang berbeda.

Sebuah pertanyaan membagi kelompok berisi n hewan menjadi kelompok *ya* berisi n_{ya} hewan dan kelompok *tidak* berisi n_{tidak}. Ketidakmurnian pertanyaan itu adalah indeks Gini kedua kelompok, masing-masing ditimbang menurut ukurannya:

$$G_{\text{split}} = \frac{n_{\text{ya}}}{n} G_{\text{ya}} + \frac{n_{\text{tidak}}}{n} G_{\text{tidak}} \tag{12.2}$$

dan CART mempertahankan pertanyaan dengan G_{split} terkecil, asalkan lebih kecil daripada ketidakmurnian kelompok sebelum dibagi. Mari kita hitung dengan tangan untuk pertanyaan pertama. Ke-101 hewan terdiri atas 41 mamalia, 20 burung, 13 ikan, 10 hewan tak bertulang belakang, 8 serangga, 5 reptil, dan 4 amfibi, sehingga menurut Persamaan (12.1)

$$G = 1 - \frac{41^2 + 20^2 + 13^2 + 10^2 + 8^2 + 5^2 + 4^2}{101^2} = 1 - \frac{2455}{10201} = 0.7593$$

Pertanyaan *milk = yes* mengirim 41 mamalia ke kelompok ya, yang murni ($G_{\text{ya}} = 0$), dan 60 hewan lainnya ke kelompok tidak, dengan jumlah per golongan 20, 13, 10, 8, 5, dan 4:

$$G_{\text{tidak}} = 1 - \frac{400 + 169 + 100 + 64 + 25 + 16}{3600} = 1 - \frac{774}{3600} = 0.785$$

sehingga menurut Persamaan (12.2), $G_{\text{split}} = \frac{41}{101} \cdot 0 + \frac{60}{101} \cdot 0.785 = 0.4663$, jauh di bawah 0.7593. **Tabel 12.1** membandingkannya dengan pertanyaan-pertanyaan terbaik berikutnya.

Tabel 12.1 Lima pertanyaan pertama terbaik untuk 101 hewan, menurut ketidakmurnian Gini tertimbang dari dua kelompok yang dihasilkannya. Susu menang.

Pertanyaan	Kelompok ya	G_{ya}	Kelompok tidak	G_{tidak}	G_{split}
milk = yes	41	0.0000	60	0.7850	0.4663
eggs = yes	59	0.7854	42	0.0918	0.4970
hair = yes	43	0.1687	58	0.7830	0.5215
feathers = yes	20	0.0000	81	0.6868	0.5508
toothed = yes	61	0.5160	40	0.6462	0.5676

Bulu burung juga menghasilkan kelompok ya yang murni sempurna, tetapi kecil (20 burung), dan menyisakan kelompok campuran yang besar; susu menghasilkan kelompok murni yang dua kali lebih besar. Telur berada tepat di belakang susu, karena hampir semua yang bertelur bukan mamalia — tetapi kelompok yang bertelur sangat bercampur. Setelah susu, CART mengulang seluruh pencarian di dalam kelompok tidak, dan seterusnya ke bawah, sampai setiap kelompok murni atau tidak ada lagi pertanyaan yang membuatnya lebih murni.

ID3 dan penerusnya, C4.5, metode pohon keputusan karya Quinlan, melakukan hal yang sama dengan ukuran kemurnian lain, **entropi**, dan memilih pertanyaan dengan *information gain* (perolehan informasi) terbesar [6], [7]; keduanya juga mengizinkan pertanyaan dengan lebih dari dua jawaban. Pada data seperti data kita, kedua ukuran itu biasanya memilih pertanyaan yang sama. Soal Latihan 12.3 meminta Anda mencobanya.

> **DEFINISI.** *Pohon keputusan* menggolongkan sebuah kasus dengan mengajukan satu pertanyaan di setiap node dan mengikuti cabang jawabannya sampai sebuah daun memberikan golongannya. *CART* menumbuhkan pohon biner dari atas ke bawah: di setiap node ia memilih, di antara semua pertanyaan ya/tidak tentang satu atribut, pertanyaan yang kedua kelompoknya memiliki ketidakmurnian Gini tertimbang terendah, dan berhenti ketika sebuah kelompok murni atau tidak ada pertanyaan yang menurunkan ketidakmurniannya.

12.2.4 Aritmetika yang tepat, agar Java dan Python sepakat

Sebuah rincian kecil tetapi penting: dua pertanyaan dapat memiliki ketidakmurnian yang *persis* sama, dan pohonnya lalu bergantung pada pertanyaan mana yang memenangkan seri itu. Bila Java dan Python menghitung ketidakmurnian dengan bilangan titik mengambang (*floating point*), perbedaan pada digit biner terakhir dapat membuat keduanya memilih secara berbeda, dan kedua program akan menumbuhkan pohon yang berbeda. Karena itu program kita tidak pernah membagi. Dengan mengalikan Persamaan (12.2) dengan n dan menuliskan indeks Gini memakai jumlah anggota, ketidakmurnian sebuah pertanyaan adalah pecahan

$$n\,G_{\text{split}} = \frac{\left(a^2 - \sum_k y_k^2\right) b + \left(b^2 - \sum_k z_k^2\right) a}{a\,b} \tag{12.3}$$

dengan a dan b ukuran kelompok ya dan tidak, serta y_k dan z_k jumlah anggota per golongan di masing-masing kelompok. Pembilang dan penyebutnya bilangan bulat, dan dua pecahan $\frac{p}{q}$ dan $\frac{r}{s}$ dibandingkan sebagai $p \cdot s < r \cdot q$, yang tepat. Seri lalu diputuskan dengan aturan tetap: atribut yang muncul lebih dahulu di tabel yang menang, dan untuk bilangan, ambang yang lebih kecil. Dengan begitu kedua bahasa menumbuhkan pohon yang sama pada data apa pun.

12.2.5 Pohon yang benar untuk semua yang pernah dilihatnya

Pohon kita menjawab ke-101 hewan dengan benar. Hal itu membuktikan lebih sedikit daripada kesannya, karena pohon itu ditumbuhkan *dari* hewan-hewan tersebut: sebuah pohon selalu dapat ditumbuhkan sampai setiap daunnya murni, dan ia lalu benar untuk datanya sendiri meskipun tidak mempelajari apa pun yang berguna. Pertanyaan yang jujur adalah bagaimana ia menjawab hewan-hewan yang **belum** pernah dilihatnya.

Untuk kumpulan data yang kecil, jawaban terbaiknya adalah pengujian **leave-one-out** (sisihkan satu). Keluarkan satu hewan, tumbuhkan pohon dari 100 hewan lainnya, lalu tanyakan kepada pohon itu tentang hewan yang disisihkan; kembalikan hewan itu, keluarkan hewan berikutnya, dan ulangi untuk ke-101 hewan. Setiap jawabannya lalu adalah jawaban tentang hewan yang

tidak menjadi bahan belajar pohon itu. Dengan cara ini pohon kita benar untuk 96 dari 101 hewan. Kelima hewan yang dijawabnya keliru sangat mendidik: kura-kura darat, ketika disisihkan, disebut serangga oleh pohon yang hanya melihat empat reptil darat lainnya; newt, amfibi berekor, disebut reptil; kalajengking, yang berkaki delapan, disebut serangga. Inilah persis "cara-cara yang tak terduga" ketika aturan hasil belajar menggeneralisasi secara berlebihan [1], dan persis kasus-kasus yang akan langsung ditangkap seorang pakar dalam sekali lihat.

12.2.6 Dari lintasan ke node VCIRS

Mengubah pohon menjadi basis pengetahuan VCIRS itu lugas, dan setiap bagian node punya sumber yang alami.

- **Kondisi** node adalah pertanyaan-pertanyaan pada lintasan dari puncak ke daun, berurutan: `milk = no, feathers = no, fins = yes`. Jawaban ya untuk `legs <= 3` memberikan kondisi `legs <= 3` dan jawaban tidak memberikan `legs > 3`; mesin membandingkan bilangan sebagai bilangan.
- **Konklusi**-nya adalah golongan daun itu: `type = fish`.
- **Faktor kepastian** node adalah porsi hewan di daun itu yang bergolongan sama dengan golongan daun, sebagai persentase bulat (Persamaan (12.4) di bawah).
- **Kasus cornerstone**-nya adalah hewan pertama dalam data yang sampai ke daun itu, lengkap dengan semua faktanya dan namanya: `animal = bass, hair = no, ….` Dengan demikian setiap node hasil belajar membawa contoh nyata, dan ketika pakar kelak mengoreksi node itu, daftar perbedaannya membandingkan kasus barunya dengan hewan nyata yang ia kenal.

Faktor kepastiannya adalah

$$CF = \text{round}\left(100 \cdot \frac{m}{n}\right) \tag{12.4}$$

dengan n hewan yang sampai ke daun itu dan m di antaranya bergolongan sama dengan daunnya. Daun yang murni memberikan CF 100; daun berisi 8 hewan, 6 di antaranya satu golongan, akan memberikan CF 75, dan mesin akan membawa ketidakpastian itu ke dalam jawaban-jawabannya seperti di setiap bab sejak Bab 3.

Setiap lintasan menjadi node **tingkat atas** (opsi 5 VCIRS), dengan pertanyaan-pertanyaan lintasannya dipilih sebagai kondisinya. Karena lintasan-lintasan sebuah pohon tidak pernah tumpang-tindih, paling banyak satu node tingkat atas yang menyala untuk hewan mana pun, dan basis pengetahuan memberikan jawaban yang persis sama dengan pohonnya. Sejak itu ia adalah basis pengetahuan VCIRS biasa: ia hanya menanyakan apa yang dibutuhkannya, menjelaskan setiap jawabannya, dan mempelajari pengecualian.

12.2.7 Apa yang tidak dapat diketahui data

Aturan hasil belajar hanya sebaik tabel asalnya, dan setiap tabel punya tiga jenis batas yang layak disebutkan sebelum kita memercayainya.

Yang pertama adalah **kosakatanya**. Data Zoo menggambarkan setiap hewan dengan 16 fakta, yang dipilih oleh orang yang membuat tabelnya. Bila fakta yang sebenarnya membedakan dua golongan tidak ada di antaranya, tidak ada metode pembelajaran yang dapat menemukannya. Reptil dan amfibi, misalnya, terutama berbeda pada kulitnya — kering dan bersisik, atau lembap dan telanjang — dan tabelnya tidak punya kolom untuk kulit. Pohon berusaha sebaik mungkin

dengan apa yang dimilikinya (air, telur), dan Subbab 12.5 akan memperlihatkan di mana hal itu tidak cukup.

Yang kedua adalah **sampelnya**. Tabel itu memuat 101 hewan yang dipilih seseorang untuk dimasukkan, dan hewan-hewan itu bukan gambaran yang adil tentang dunia hewan: 41 di antaranya mamalia, tetapi hanya 5 reptil dan 4 amfibi, sehingga aturan untuk reptil dan amfibi bersandar pada sangat sedikit contoh. Tabel itu bahkan memuat satu entri yang membuat setiap guru biologi tersenyum: sebuah baris bernama `girl` (anak perempuan), mamalia berkaki dua yang dipelihara manusia dan lebih besar daripada kucing. Entri itu tidak salah — anak perempuan memang mamalia — tetapi ia mengingatkan bahwa kumpulan data adalah buatan manusia, lengkap dengan lelucon dan celahnya.

Yang ketiga adalah **labelnya**. Golongan setiap hewan ditentukan oleh seseorang, dan golongan-golongan itu sendiri adalah sebuah pilihan. Data Zoo menyatukan kerang, kepiting, lobster, gurita, kalajengking, ubur-ubur, siput, bintang laut, dan cacing dalam satu golongan, yang di sini kita sebut *invertebrate* (tak bertulang belakang), padahal ahli biologi akan membaginya menjadi beberapa kelompok yang sangat berbeda. Pohon mempelajari golongan menurut tabel, bukan menurut biologi.

Semua ini tidak membuat tabelnya tidak berguna. Semua ini membuat pakar diperlukan. Basis pengetahuan hasil belajar adalah draf pertama, yang ditulis dengan cepat dari data; pakar membacanya, menambahkan fakta-fakta yang tidak ada di tabel, dan mengoreksi jawaban-jawaban yang tidak dapat diketahui tabel.

12.3 Langkah 1: CART dari Nol

Kode 12.1 adalah seluruh pembelajar pohon dalam Python.

Kode 12.1 `cart.py`: pohon keputusan CART dengan aritmetika Gini yang tepat dan pengujian leave-one-out.

```
"""A small CART decision tree, written from scratch.

    python cart.py [../data/zoo.csv]

CART (Classification And Regression Trees) grows a binary tree: at every
node it tries every yes/no question about one column and keeps the question
whose two groups are purest by the Gini index. It stops when a group is pure
or no question makes it purer. Numbers are compared exactly, as fractions of
whole numbers, so that Java and Python always choose the same question.
"""

import pathlib
import re
import sys

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import num

NUMBER = re.compile(r"-?[0-9]+(\.[0-9]+)?")


class Data:
    """A table read from a CSV file: a name column, attributes, a class."""

    def __init__(self, path):
        with open(path, encoding="utf-8") as f:
            lines = [ln.strip() for ln in f if ln.strip()]
        header = lines[0].split(",")
        self.attributes = header[1:-1]
        self.target = header[-1]
        self.rows = []                    # (name, {attribute: value}, class)
        for ln in lines[1:]:
            cells = ln.split(",")
            self.rows.append((cells[0], dict(zip(self.attributes, cells[1:-1])),
                              cells[-1]))
        self.numeric = [a for a in self.attributes
                        if all(NUMBER.fullmatch(r[1][a]) for r in self.rows)]


class Question:
    """'legs <= 3' or 'milk = yes': the test of a node, with its two texts."""
```

Kode 12.1 (Lanjutan) *cart.py: pohon keputusan CART dengan aritmetika Gini yang tepat dan pengujian leave-one-out.*

```
    def __init__(self, attribute, numeric, value, other):
        self.attribute = attribute
        self.numeric = numeric
        self.value = value                # a threshold, or the value tested
        self.other = other                # the other value, if only two

    def ask(self, values):
        x = values.get(self.attribute)
        if self.numeric:
            return float(x) <= float(self.value)
        return x == self.value

    def yes_text(self):
        op = " <= " if self.numeric else " = "
        return self.attribute + op + self.value

    def no_text(self):
        if self.numeric:
            return self.attribute + " > " + self.value
        if self.other is not None:
            return self.attribute + " = " + self.other
        return self.attribute + " <> " + self.value


class Node:
    def __init__(self, rows):
        self.rows = rows
        self.counts = {}
        for _, _, c in rows:
            self.counts[c] = self.counts.get(c, 0) + 1
        self.counts = dict(sorted(self.counts.items()))
        self.label = ""                   # the most frequent class; a tie
        for c, k in self.counts.items():  # goes to the first name A to Z
            if not self.label or k > self.counts[self.label]:
                self.label = c
        self.question = None
        self.yes = None
        self.no = None

    def pure(self):
        return len(self.counts) == 1
```

Kode 12.1 (Lanjutan) *cart.py: pohon keputusan CART dengan aritmetika Gini yang tepat dan pengujian leave-one-out.*

```
def impurity(counts, n):
    """n times the Gini index, times n: n*n - sum of squared counts."""
    return n * n - sum(c * c for c in counts.values())


def questions(data, rows, attribute):
    values = sorted(set(r[1][attribute] for r in rows),
                    key=float if attribute in data.numeric else str)
    if attribute in data.numeric:
        return [Question(attribute, True,
                         num.fmt((float(a) + float(b)) / 2), None)
                for a, b in zip(values, values[1:])]
    if len(values) == 2:
        return [Question(attribute, False, values[1], values[0])]
    return [Question(attribute, False, v, None) for v in values]


def best_question(data, node):
    """The question whose two groups have the lowest total Gini impurity,
    or None when no question makes the node purer. Scores are fractions
    num/den compared by cross-multiplying, so ties are exact."""
    n = len(node.rows)
    best, best_num, best_den = None, 0, 1
    for attribute in data.attributes:
        for q in questions(data, node.rows, attribute):
            yes = Node([r for r in node.rows if q.ask(r[1])])
            no = Node([r for r in node.rows if not q.ask(r[1])])
            a, b = len(yes.rows), len(no.rows)
            if a == 0 or b == 0:
                continue
            score = impurity(yes.counts, a) * b + impurity(no.counts, b) * a
            if best is None or score * best_den < best_num * a * b:
                best, best_num, best_den = q, score, a * b
    if best is None or best_num * n >= impurity(node.counts, n) * best_den:
        return None
    return best
```

Kode 12.1 (Lanjutan) *cart.py: pohon keputusan CART dengan aritmetika Gini yang tepat dan pengujian leave-one-out.*

```
def grow(data, rows):
    node = Node(rows)
    if node.pure():
        return node
    q = best_question(data, node)
    if q is None:
        return node
    node.question = q
    node.yes = grow(data, [r for r in rows if q.ask(r[1])])
    node.no = grow(data, [r for r in rows if not q.ask(r[1])])
    return node


def predict(node, values):
    while node.question is not None:
        node = node.yes if node.question.ask(values) else node.no
    return node.label


def leaf_text(node):
    wrong = len(node.rows) - node.counts[node.label]
    return node.label + " (" + str(len(node.rows)) + (
        "/" + str(wrong) if wrong else "") + ")"


def show(node, indent=""):
    """The tree as lines: 'milk = yes: mammal (41)'."""
    out = []
    for text, child in ((node.question.yes_text(), node.yes),
                        (node.question.no_text(), node.no)):
        if child.question is None:
            out.append(indent + text + ": " + leaf_text(child))
        else:
            out.append(indent + text + ":")
            out += show(child, indent + "|  ")
    return out


def paths(node, conds=()):
    """Every leaf with the conditions on the way to it (root first)."""
    if node.question is None:
        return [(list(conds), node)]
    q = node.question
    return (paths(node.yes, conds + (q.yes_text(),))
            + paths(node.no, conds + (q.no_text(),)))
```

Kode 12.1 (Lanjutan) *cart.py: pohon keputusan CART dengan aritmetika Gini yang tepat dan pengujian leave-one-out.*

```
def depth(node):
    if node.question is None:
        return 0
    return 1 + max(depth(node.yes), depth(node.no))


def main(args):
    path = args[0] if args else str(pathlib.Path(__file__).resolve()
                                    .parents[1] / "data" / "zoo.csv")
    data = Data(path)
    print(str(len(data.rows)) + " rows, " + str(len(data.attributes))
          + " attributes (numeric: " + (", ".join(data.numeric) or "none")
          + ")")
    tree = grow(data, data.rows)
    print()
    for line in show(tree):
        print(line)
    right = sum(predict(tree, r[1]) == r[2] for r in data.rows)
    print()
    print("leaves: " + str(len(paths(tree))) + ", depth: " + str(depth(tree))
          + ", right on the rows it learned from: " + str(right) + "/"
          + str(len(data.rows)))
    print()
    print("Leave-one-out: learn from all rows but one, then ask about it.")
    wrong = []
    for i, row in enumerate(data.rows):
        t = grow(data, data.rows[:i] + data.rows[i + 1:])
        got = predict(t, row[1])
        if got != row[2]:
            wrong.append(row[0] + ": " + row[2] + ", tree says " + got)
    print("right: " + str(len(data.rows) - len(wrong)) + "/"
          + str(len(data.rows)))
    for w in wrong:
        print("  " + w)


if __name__ == "__main__":
    main(sys.argv[1:])
```

Bacalah dalam lima bagian.

Datanya. `Data` membaca berkas CSV yang kolom pertamanya berupa nama, kolom terakhirnya golongan, dan kolom-kolom di antaranya atribut. Ia menyimpan setiap baris sebagai tiga serangkai: nama, kamus nilai-nilai atribut, dan golongannya. Sebuah atribut diperlakukan sebagai bilangan bila setiap nilai di kolomnya tampak seperti bilangan; dalam data Zoo itulah `legs`.

Pertanyaan. `Question` adalah satu uji ya/tidak: `legs <= 3` untuk bilangan, `milk = yes` untuk kata. `questions` mendaftar setiap pertanyaan yang layak diajukan tentang satu atribut dalam sekelompok baris. Untuk bilangan ia mencoba setiap ambang di tengah-tengah dua nilai bertetangga yang muncul (untuk kaki 0, 2, 4, 5, 6, dan 8: 1, 3, 4.5, 5.5, dan 7). Untuk kata dengan dua nilai, seperti `yes` dan `no`, satu pertanyaan sudah cukup, karena bertanya tentang `no` hanya akan menukar kedua kelompoknya; untuk kata dengan lebih banyak nilai ia mencoba setiap nilai melawan semua nilai lainnya, `stain = oil` melawan `stain <> oil`. `yes_text` dan `no_text` menuliskan kedua jawabannya sebagai kondisi VCIRS.

Node dan kemurnian. `Node` menyimpan baris-barisnya dan jumlah setiap golongan, menurut abjad; `label`-nya adalah golongan yang paling banyak, dan bila seri, nama yang pertama menurut

abjad yang menang. `impurity` menghitung $n^2 - \sum c^2$, yaitu n^2 kali indeks Gini pada Persamaan (12.1).

Memilih pertanyaan. `best_question` adalah Persamaan (12.3) yang dipekerjakan. Untuk setiap atribut dan setiap pertanyaan ia menyusun kelompok ya dan tidak, melewati pertanyaan yang membuat salah satu kelompok kosong, dan menghitung skornya sebagai pembilang Persamaan (12.3) dengan penyebut $a \cdot b$. Pertanyaan baru hanya menggantikan yang terbaik sejauh ini bila pecahannya benar-benar lebih kecil, `score * best_den < best_num * a * b`, sehingga yang pertama di antara beberapa pertanyaan yang sama baiknya yang dipertahankan. Di akhir, pertanyaan terbaik hanya dikembalikan bila benar-benar lebih murni daripada node itu sendiri; bila tidak, node itu menjadi daun.

Menumbuhkan, meramalkan, dan menampilkan. `grow` adalah rekursi dalam enam baris: buat node-nya; bila murni, atau tidak ada pertanyaan yang membantu, ia menjadi daun; bila tidak, bagi baris-barisnya dengan pertanyaan terbaik dan tumbuhkan setiap kelompok. `predict` berjalan dari puncak, mengikuti jawaban-jawabannya. `show` mencetak pohonnya dengan satu baris per cabang, `paths` mendaftar setiap daun beserta kondisi-kondisi di sepanjang jalan ke sana, dan `main` mencetak pohonnya, ukurannya, jawabannya untuk datanya sendiri, dan jawaban leave-one-out-nya. **Kode 12.2** adalah pembelajar yang sama dalam Java.

Kode 12.2 `Cart.java`: pembelajar CART yang sama dalam Java.

```java
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.TreeMap;
import java.util.TreeSet;

import vcirs.Num;

/**
 * A small CART decision tree, written from scratch.
 * java Cart [../data/zoo.csv]
 * CART grows a binary tree: at every node it tries every yes/no question
 * about one column and keeps the question whose two groups are purest by
 * the Gini index. It stops when a group is pure or no question makes it
 * purer. Numbers are compared exactly, as fractions of whole numbers, so
 * that Java and Python always choose the same question.
 */
public class Cart {
    static final String NUMBER = "-?[0-9]+(\\.[0-9]+)?";

    /** One row: the animal's name, its attribute values, its class. */
    record Row(String name, Map<String, String> values, String cls) {
    }

    /** A table read from a CSV file: a name column, attributes, a class. */
    static class Data {
        final List<String> attributes = new ArrayList<>();
        final String target;
        final List<Row> rows = new ArrayList<>();
        final List<String> numeric = new ArrayList<>();
```

Kode 12.2 (Lanjutan) *Cart.java: pembelajar CART yang sama dalam Java.*

```
        Data(String path) throws IOException {
            List<String> lines = new ArrayList<>();
            for (String ln : Files.readAllLines(Path.of(path),
                    StandardCharsets.UTF_8)) {
                if (!ln.strip().isEmpty()) {
                    lines.add(ln.strip());
                }
            }
            String[] header = lines.get(0).split(",");
            for (int i = 1; i < header.length - 1; i++) {
                attributes.add(header[i]);
            }
            target = header[header.length - 1];
            for (String ln : lines.subList(1, lines.size())) {
                String[] cells = ln.split(",");
                Map<String, String> values = new LinkedHashMap<>();
                for (int i = 0; i < attributes.size(); i++) {
                    values.put(attributes.get(i), cells[i + 1]);
                }
                rows.add(new Row(cells[0], values, cells[cells.length - 1]));
            }
            for (String a : attributes) {
                boolean all = true;
                for (Row r : rows) {
                    all &= r.values().get(a).matches(NUMBER);
                }
                if (all) {
                    numeric.add(a);
                }
            }
        }
    }

    /** "legs <= 3" or "milk = yes": the test of a node, with its texts. */
    static class Question {
        final String attribute;
        final boolean numeric;
        final String value;             // a threshold, or the value tested
        final String other;             // the other value, if only two
```

Kode 12.2 (Lanjutan) *Cart.java: pembelajar CART yang sama dalam Java.*

```
        Question(String attribute, boolean numeric, String value, String other) {
            this.attribute = attribute;
            this.numeric = numeric;
            this.value = value;
            this.other = other;
        }

        boolean ask(Map<String, String> values) {
            String x = values.get(attribute);
            if (numeric) {
                return Double.parseDouble(x) <= Double.parseDouble(value);
            }
            return value.equals(x);
        }

        String yesText() {
            return attribute + (numeric ? " <= " : " = ") + value;
        }

        String noText() {
            if (numeric) {
                return attribute + " > " + value;
            }
            if (other != null) {
                return attribute + " = " + other;
            }
            return attribute + " <> " + value;
        }
    }

    static class Node {
        final List<Row> rows;
        final Map<String, Integer> counts = new TreeMap<>();
        String label = "";              // the most frequent class; a tie
        Question question;              // goes to the first name A to Z
        Node yes;
        Node no;
```

Kode 12.2 (Lanjutan) *Cart.java: pembelajar CART yang sama dalam Java.*

```
        Node(List<Row> rows) {
            this.rows = rows;
            for (Row r : rows) {
                counts.merge(r.cls(), 1, Integer::sum);
            }
            for (Map.Entry<String, Integer> e : counts.entrySet()) {
                if (label.isEmpty() || e.getValue() > counts.get(label)) {
                    label = e.getKey();
                }
            }
        }

        boolean pure() {
            return counts.size() == 1;
        }
    }

    /** n times the Gini index, times n: n*n - sum of squared counts. */
    static long impurity(Map<String, Integer> counts, long n) {
        long s = 0;
        for (int c : counts.values()) {
            s += (long) c * c;
        }
        return n * n - s;
    }

    static List<Question> questions(Data data, List<Row> rows, String a) {
        boolean num = data.numeric.contains(a);
        Comparator<String> order = num
                ? Comparator.comparingDouble(Double::parseDouble)
                : Comparator.naturalOrder();
        TreeSet<String> set = new TreeSet<>(order);
        for (Row r : rows) {
            set.add(r.values().get(a));
        }
        List<String> values = new ArrayList<>(set);
        List<Question> out = new ArrayList<>();
        if (num) {
            for (int i = 0; i + 1 < values.size(); i++) {
                double t = (Double.parseDouble(values.get(i))
                        + Double.parseDouble(values.get(i + 1))) / 2;
                out.add(new Question(a, true, Num.fmt(t), null));
            }
        } else if (values.size() == 2) {
            out.add(new Question(a, false, values.get(1), values.get(0)));
        } else {
```

Kode 12.2 (Lanjutan) *Cart.java: pembelajar CART yang sama dalam Java.*

```
            for (String v : values) {
                out.add(new Question(a, false, v, null));
            }
        }
        return out;
    }

    static List<Row> side(List<Row> rows, Question q, boolean yes) {
        List<Row> out = new ArrayList<>();
        for (Row r : rows) {
            if (q.ask(r.values()) == yes) {
                out.add(r);
            }
        }
        return out;
    }

    /**
     * The question whose two groups have the lowest total Gini impurity, or
     * null when no question makes the node purer. Scores are fractions
     * num/den compared by cross-multiplying, so ties are exact.
     */
    static Question bestQuestion(Data data, Node node) {
        long n = node.rows.size();
        Question best = null;
        long bestNum = 0, bestDen = 1;
        for (String a : data.attributes) {
            for (Question q : questions(data, node.rows, a)) {
                Node yes = new Node(side(node.rows, q, true));
                Node no = new Node(side(node.rows, q, false));
                long ya = yes.rows.size(), nb = no.rows.size();
                if (ya == 0 || nb == 0) {
                    continue;
                }
                long score = impurity(yes.counts, ya) * nb
                        + impurity(no.counts, nb) * ya;
                if (best == null || score * bestDen < bestNum * ya * nb) {
                    best = q;
                    bestNum = score;
                    bestDen = ya * nb;
                }
            }
        }
        if (best == null || bestNum * n >= impurity(node.counts, n) * bestDen) {
            return null;
        }
```

Kode 12.2 (Lanjutan) *Cart.java: pembelajar CART yang sama dalam Java.*

```
        return best;
    }

    static Node grow(Data data, List<Row> rows) {
        Node node = new Node(rows);
        if (node.pure()) {
            return node;
        }
        Question q = bestQuestion(data, node);
        if (q == null) {
            return node;
        }
        node.question = q;
        node.yes = grow(data, side(rows, q, true));
        node.no = grow(data, side(rows, q, false));
        return node;
    }

    static String predict(Node node, Map<String, String> values) {
        while (node.question != null) {
            node = node.question.ask(values) ? node.yes : node.no;
        }
        return node.label;
    }

    static String leafText(Node node) {
        int wrong = node.rows.size() - node.counts.get(node.label);
        return node.label + " (" + node.rows.size()
                + (wrong > 0 ? "/" + wrong : "") + ")";
    }

    /** The tree as lines: "milk = yes: mammal (41)". */
    static List<String> show(Node node, String indent) {
        List<String> out = new ArrayList<>();
        String[] texts = {node.question.yesText(), node.question.noText()};
        Node[] kids = {node.yes, node.no};
        for (int i = 0; i < 2; i++) {
            if (kids[i].question == null) {
                out.add(indent + texts[i] + ": " + leafText(kids[i]));
            } else {
                out.add(indent + texts[i] + ":");
                out.addAll(show(kids[i], indent + "|   "));
            }
        }
        return out;
    }
```

Kode 12.2 (Lanjutan) *Cart.java: pembelajar CART yang sama dalam Java.*

```
    /** A leaf with the conditions on the way to it (root first). */
    record Branch(List<String> conditions, Node leaf) {
    }

    static List<Branch> paths(Node node, List<String> conds) {
        List<Branch> out = new ArrayList<>();
        if (node.question == null) {
            out.add(new Branch(conds, node));
            return out;
        }
        List<String> yes = new ArrayList<>(conds);
        yes.add(node.question.yesText());
        List<String> no = new ArrayList<>(conds);
        no.add(node.question.noText());
        out.addAll(paths(node.yes, yes));
        out.addAll(paths(node.no, no));
        return out;
    }

    static int depth(Node node) {
        return node.question == null ? 0
                : 1 + Math.max(depth(node.yes), depth(node.no));
    }

    public static void main(String[] args) throws IOException {
        String path = args.length > 0 ? args[0] : "../data/zoo.csv";
        Data data = new Data(path);
        System.out.println(data.rows.size() + " rows, " + data.attributes.size()
                + " attributes (numeric: " + (data.numeric.isEmpty() ? "none"
                : String.join(", ", data.numeric)) + ")");
        Node tree = grow(data, data.rows);
        System.out.println();
        for (String line : show(tree, "")) {
            System.out.println(line);
        }
        int right = 0;
        for (Row r : data.rows) {
            right += predict(tree, r.values()).equals(r.cls()) ? 1 : 0;
        }
        System.out.println();
        System.out.println("leaves: " + paths(tree, List.of()).size()
                + ", depth: " + depth(tree)
                + ", right on the rows it learned from: " + right + "/"
                + data.rows.size());
        System.out.println();
```

Kode 12.2 (Lanjutan) *Cart.java: pembelajar CART yang sama dalam Java.*

```
        System.out.println("Leave-one-out: learn from all rows but one, then "
                + "ask about it.");
        List<String> wrong = new ArrayList<>();
        for (int i = 0; i < data.rows.size(); i++) {
            List<Row> rest = new ArrayList<>(data.rows);
            Row row = rest.remove(i);
            String got = predict(grow(data, rest), row.values());
            if (!got.equals(row.cls())) {
                wrong.add(row.name() + ": " + row.cls() + ", tree says " + got);
            }
        }
        System.out.println("right: " + (data.rows.size() - wrong.size()) + "/"
                + data.rows.size());
        for (String w : wrong) {
            System.out.println("  " + w);
        }
    }
}
```

Java menambahkan beberapa hal yang di Python tersedia dengan sendirinya. Jumlah per golongan disimpan dalam `TreeMap`, yang menjaga kunci-kuncinya urut abjad; nilai-nilai sebuah atribut diurutkan dalam `TreeSet` dengan pembanding yang membandingkan bilangan sebagai bilangan dan kata sebagai kata; dan seluruh aritmetika ketidakmurnian memakai `long`, bilangan bulat 64 bit, sehingga hasil-hasil kali dalam Persamaan (12.3) tidak dapat meluap untuk tabel apa pun yang muat di memori. `Row` dan `Branch` adalah *record*. Ambang-ambangnya ditulis dengan `Num.fmt` milik vcirs-core, sehingga `4.5` dicetak sebagai `4.5` dan `3.0` sebagai `3` di kedua bahasa. Jalankan dengan `java -cp out Cart` di folder `java` (setelah dikompilasi bersama mesinnya, seperti di setiap bab) atau `python cart.py` di folder `python`. Keduanya mencetak:

```
101 rows, 16 attributes (numeric: legs)

milk = yes: mammal (41)
milk = no:
|  feathers = yes: bird (20)
|  feathers = no:
|  |  fins = yes: fish (13)
|  |  fins = no:
|  |  |  backbone = yes:
|  |  |  |  aquatic = yes:
|  |  |  |  |  eggs = yes: amphibian (4)
|  |  |  |  |  eggs = no: reptile (1)
|  |  |  |  aquatic = no: reptile (4)
|  |  |  backbone = no:
|  |  |  |  airborne = yes: insect (6)
|  |  |  |  airborne = no:
|  |  |  |  |  predator = yes: invertebrate (8)
|  |  |  |  |  predator = no:
|  |  |  |  |  |  legs <= 3: invertebrate (2)
|  |  |  |  |  |  legs > 3: insect (2)

leaves: 10, depth: 7, right on the rows it learned from: 101/101

Leave-one-out: learn from all rows but one, then ask about it.
right: 96/101
  ladybird: insect, tree says invertebrate
  newt: amphibian, tree says reptile
  scorpion: invertebrate, tree says insect
  seasnake: reptile, tree says amphibian
  tortoise: reptile, tree says insect
```

Pohon yang tercetak adalah **Gambar 12.2** dalam bentuk teks: setiap baris yang menjorok adalah sebuah cabang, `(41)` menghitung hewan yang sampai ke sebuah daun, dan daun yang tidak murni akan menampilkan kekeliruannya setelah garis miring, seperti `(8/2)`. Pengujian leave-one-out memakan waktu sedikit lebih lama daripada bagian lain, karena ia menumbuhkan 101 pohon; di laptop biasa kedua bahasa selesai jauh di bawah satu detik.

12.4 Langkah 2: Dari Pohon ke Basis Pengetahuan

Kode 12.3 mengubah pohon hasil belajar menjadi basis pengetahuan VCIRS, mengikuti Subbab 12.2.6 baris demi baris.

Kode 12.3 `tree_to_kb.py`: setiap daun pohon menjadi node tingkat atas, dengan lintasannya sebagai kondisi dan hewan pertamanya sebagai cornerstone.

```
"""From a decision tree to a VCIRS knowledge base.

    python tree_to_kb.py

Every leaf of the tree becomes a top-level node: its conditions are the
questions on the path from the root, its conclusion is the leaf's class, its
certainty factor is the share of the leaf's rows that have that class, and
its cornerstone case is the first animal of the data that reached the leaf.
The knowledge base is saved as ../data/zoo-kb.json.
"""

import pathlib
import sys

import cart
from vcirs import (TOP_LEVEL, Case, Conclusion, Condition, KnowledgeBase,
                   report)

DATA = pathlib.Path(__file__).resolve().parents[1] / "data"
QUESTIONS = {
    "hair": "Does it have hair or fur?", "feathers": "Does it have feathers?",
    "eggs": "Does it lay eggs?", "milk": "Does it feed its young with milk?",
    "airborne": "Can it fly?", "aquatic": "Does it live in water?",
    "predator": "Does it hunt other animals?", "toothed": "Does it have teeth?",
    "backbone": "Does it have a backbone?", "breathes": "Does it breathe air?",
    "venomous": "Is it venomous?", "fins": "Does it have fins?",
    "legs": "How many legs does it have?", "tail": "Does it have a tail?",
    "domestic": "Is it kept by people?", "catsize": "Is it bigger than a cat?",
}


def facts_of(values):
    return [Condition(a, "=", v) for a, v in values.items()]


def tree_to_kb(data, tree, name):
    kb = KnowledgeBase(name, "Learned by CART from " + str(len(data.rows))
                       + " examples, then converted leaf by leaf.")
    for a in data.attributes:
        choices = sorted(set(r[1][a] for r in data.rows),
                         key=float if a in data.numeric else str)
        kb.declare(a, QUESTIONS.get(a, ""), choices)
    for conds, leaf in cart.paths(tree):
        first = leaf.rows[0]                    # the cornerstone animal
        n = len(leaf.rows)                      # the share, rounded half up
        cf = (200 * leaf.counts[leaf.label] + n) // (2 * n)
```

Kode 12.3 (Lanjutan) *tree_to_kb.py: setiap daun pohon menjadi node tingkat atas, dengan lintasannya sebagai kondisi dan hewan pertamanya sebagai cornerstone.*

```
        case = Case([Condition("animal", "=", first[0])] + facts_of(first[1]),
                    [Conclusion(data.target, leaf.label)], cf)
        kb.refine(case, TOP_LEVEL, chosen=[Condition.parse(c) for c in conds],
                  author="CART")
    return kb


def answer(kb, values):
    got = kb.infer(values).conclusions()
    return got[0].value if got else "none"


def main():
    data = cart.Data(str(DATA / "zoo.csv"))
    tree = cart.grow(data, data.rows)
    kb = tree_to_kb(data, tree, "Animal classes")
    print(kb.summary())
    print()
    for line in report.tree(kb):
        print(line)
    same = sum(answer(kb, r[1]) == cart.predict(tree, r[1])
               for r in data.rows)
    right = sum(answer(kb, r[1]) == r[2] for r in data.rows)
    print()
    print("the knowledge base agrees with the tree on " + str(same) + "/"
          + str(len(data.rows)) + " animals, and is right on " + str(right))
    print("the cornerstone of each node:")
    for node in kb.nodes.values():
        print("  " + node.id + ": " + node.cornerstone.facts[0].value)
    kb.save(str(DATA / "zoo-kb.json"))
    print("saved ../data/zoo-kb.json")


if __name__ == "__main__":
    main()
```

`tree_to_kb` lebih dahulu mendeklarasikan setiap atribut sebagai variabel, dengan pertanyaan dalam kata-kata sederhana (dari `QUESTIONS`) dan nilai-nilai yang muncul di data sebagai jawaban lazimnya, sehingga basis pengetahuan dapat bertanya kepada seorang anak *Does it feed its young with milk? (no/yes)*. Lalu, untuk setiap lintasan, ia menyusun kasus cornerstone — nama hewan pertama di daun itu, sebagai fakta `animal = …`, diikuti semua faktanya — dengan golongan daun sebagai konklusi dan faktor kepastian menurut Persamaan (12.4), yang dihitung dengan bilangan bulat sebagai $\lfloor (200m + n)/2n \rfloor$, yang membulatkan setengah ke atas di kedua bahasa. Akhirnya `kb.refine(case, TOP_LEVEL, chosen=…)` menciptakan node-nya, dengan pertanyaan-pertanyaan lintasan sebagai kondisi yang dipilih, persis seperti seorang pakar memilih kondisi dari sebuah kasus. `answer` bertanya kepada basis pengetahuan tentang satu hewan. **Kode 12.4** adalah versi Java-nya.

Kode 12.4 `TreeToKb.java`: pengubahan yang sama dalam Java.

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.Map;
import java.util.TreeSet;

import vcirs.Case;
import vcirs.Conclusion;
import vcirs.Condition;
import vcirs.Facts;
import vcirs.KnowledgeBase;
import vcirs.Node;
import vcirs.Report;

/**
 * From a decision tree to a VCIRS knowledge base.
 * java TreeToKb
 * Every leaf of the tree becomes a top-level node: its conditions are the
 * questions on the path from the root, its conclusion is the leaf's class,
 * its certainty factor is the share of the leaf's rows that have that class,
 * and its cornerstone case is the first animal of the data that reached the
 * leaf. The knowledge base is saved as ../data/zoo-kb.json.
 */
public class TreeToKb {
    static final Map<String, String> QUESTIONS = Map.ofEntries(
        Map.entry("hair", "Does it have hair or fur?"),
        Map.entry("feathers", "Does it have feathers?"),
        Map.entry("eggs", "Does it lay eggs?"),
        Map.entry("milk", "Does it feed its young with milk?"),
        Map.entry("airborne", "Can it fly?"),
        Map.entry("aquatic", "Does it live in water?"),
        Map.entry("predator", "Does it hunt other animals?"),
        Map.entry("toothed", "Does it have teeth?"),
        Map.entry("backbone", "Does it have a backbone?"),
        Map.entry("breathes", "Does it breathe air?"),
        Map.entry("venomous", "Is it venomous?"),
        Map.entry("fins", "Does it have fins?"),
        Map.entry("legs", "How many legs does it have?"),
        Map.entry("tail", "Does it have a tail?"),
        Map.entry("domestic", "Is it kept by people?"),
        Map.entry("catsize", "Is it bigger than a cat?"));
```

Kode 12.4 (Lanjutan) *TreeToKb.java: pengubahan yang sama dalam Java.*

```
    static List<Condition> factsOf(Map<String, String> values) {
        List<Condition> out = new ArrayList<>();
        for (Map.Entry<String, String> e : values.entrySet()) {
            out.add(new Condition(e.getKey(), "=", e.getValue()));
        }
        return out;
    }

    static KnowledgeBase treeToKb(Cart.Data data, Cart.Node tree, String name) {
        KnowledgeBase kb = new KnowledgeBase(name, "Learned by CART from "
                + data.rows.size() + " examples, then converted leaf by leaf.");
        for (String a : data.attributes) {
            TreeSet<String> set = new TreeSet<>(data.numeric.contains(a)
                    ? Comparator.comparingDouble(Double::parseDouble)
                    : Comparator.<String>naturalOrder());
            for (Cart.Row r : data.rows) {
                set.add(r.values().get(a));
            }
            kb.declare(a, QUESTIONS.getOrDefault(a, ""), new ArrayList<>(set));
        }
        for (Cart.Branch b : Cart.paths(tree, List.of())) {
            Cart.Node leaf = b.leaf();
            Cart.Row first = leaf.rows.get(0);            // the cornerstone animal
            int n = leaf.rows.size();                      // the share, rounded
            int cf = (200 * leaf.counts.get(leaf.label) + n) / (2 * n); // half up
            List<Condition> facts = new ArrayList<>();
            facts.add(new Condition("animal", "=", first.name()));
            facts.addAll(factsOf(first.values()));
            Case c = new Case(facts, List.of(new Conclusion(data.target,
                    leaf.label)), cf);
            List<Condition> chosen = new ArrayList<>();
            for (String text : b.conditions()) {
                chosen.add(Condition.parse(text));
            }
            kb.refine(c, KnowledgeBase.TOP_LEVEL, null, chosen, "CART");
        }
        return kb;
    }

    static String answer(KnowledgeBase kb, Map<String, String> values) {
        List<Conclusion> got = kb.infer(Facts.map(values)).conclusions();
        return got.isEmpty() ? "none" : got.get(0).value;
    }
```

Kode 12.4 (Lanjutan) *TreeToKb.java: pengubahan yang sama dalam Java.*

```
    public static void main(String[] args) throws Exception {
        Cart.Data data = new Cart.Data("../data/zoo.csv");
        Cart.Node tree = Cart.grow(data, data.rows);
        KnowledgeBase kb = treeToKb(data, tree, "Animal classes");
        System.out.println(kb.summary());
        System.out.println();
        for (String line : Report.tree(kb)) {
            System.out.println(line);
        }
        int same = 0, right = 0;
        for (Cart.Row r : data.rows) {
            String got = answer(kb, r.values());
            same += got.equals(Cart.predict(tree, r.values())) ? 1 : 0;
            right += got.equals(r.cls()) ? 1 : 0;
        }
        System.out.println();
        System.out.println("the knowledge base agrees with the tree on " + same
                + "/" + data.rows.size() + " animals, and is right on " + right);
        System.out.println("the cornerstone of each node:");
        for (Node node : kb.nodes.values()) {
            System.out.println("  " + node.id + ": "
                    + node.cornerstone.facts.get(0).value);
        }
        kb.save("../data/zoo-kb.json");
        System.out.println("saved ../data/zoo-kb.json");
    }
}
```

Keduanya mencetak, dan keduanya menulis `zoo-kb.json` yang sama, bita demi bita:

```
Animal classes: 17 variables, 10 nodes, 10 rules

(root)
+- mammal#1: milk = yes -> type = mammal
+- bird#1: milk = no, feathers = yes -> type = bird
+- fish#1: milk = no, feathers = no, fins = yes -> type = fish
+- amphibian#1: milk = no, feathers = no, fins = no, backbone = yes, aquatic = yes, eggs =
yes -> type = amphibian
+- reptile#1: milk = no, feathers = no, fins = no, backbone = yes, aquatic = yes, eggs = no
-> type = reptile
+- reptile#2: milk = no, feathers = no, fins = no, backbone = yes, aquatic = no -> type =
reptile
+- insect#1: milk = no, feathers = no, fins = no, backbone = no, airborne = yes -> type =
insect
+- invertebrate#1: milk = no, feathers = no, fins = no, backbone = no, airborne = no,
predator = yes -> type = invertebrate
+- invertebrate#2: milk = no, feathers = no, fins = no, backbone = no, airborne = no,
predator = no, legs <= 3 -> type = invertebrate
+- insect#2: milk = no, feathers = no, fins = no, backbone = no, airborne = no, predator =
no, legs > 3 -> type = insect

the knowledge base agrees with the tree on 101/101 animals, and is right on 101
the cornerstone of each node:
  mammal#1: aardvark
  bird#1: chicken
  fish#1: bass
  amphibian#1: frog
  reptile#1: seasnake
  reptile#2: pitviper
  insect#1: gnat
  invertebrate#1: clam
  invertebrate#2: slug
  insect#2: flea
saved ../data/zoo-kb.json
```

Kesepuluh node adalah kesepuluh lintasan pada **Gambar 12.2**, masing-masing dengan daftar kondisinya yang lengkap; `report.tree` menampilkan semuanya di tingkat atas, karena pengetahuan hasil belajar belum punya pengecualian. Basis pengetahuan sepakat dengan pohonnya untuk setiap hewan, sebagaimana mestinya. Dan setiap node punya hewan cornerstone: aardvark mewakili semua mamalia (ia mamalia pertama menurut abjad), katak mewakili amfibi, ular laut mewakili satu-satunya reptil yang hidup di air dan tidak bertelur. Tujuh belas variabel: keenam belas fakta ditambah konklusinya, `type`.

Kini basis pengetahuan dapat melakukan apa yang tidak dapat dilakukan pohon. Ia hanya menanyakan apa yang dibutuhkannya: untuk seekor tokek ia akan bertanya tentang susu, bulu burung, sirip, tulang belakang, dan air, lalu berhenti. Ia menjelaskan: *reptile#2 menyala karena milk = no, feathers = no, fins = no, backbone = yes, aquatic = no*. Dan ia dapat diajari.

12.5 Langkah 3: Pelajaran dari Sang Guru

12.5.1 Enam hewan yang tidak pernah dilihat data

Bu Lestari mencoba basis pengetahuan pada enam hewan dari Sabtu-Sabtu klubnya, yang tidak satu pun termasuk ke-101 hewan tadi, dalam tujuh pelajaran: komodo dari kebun binatang, kasuari, penyu hijau di pusat konservasi bakau, biawak di sungai, ikan glodok (*mudskipper*) di dataran lumpur, dan kunang-kunang di kebun buah menjelang malam. Untuk setiap hewan ia menuliskan ke-16 faktanya, golongannya, dan — bila basis pengetahuan keliru — kondisi yang ia pilih dari daftar perbedaan. Itulah baris-baris dalam `data/lessons.txt`; berikut dua di antaranya, dipersingkat:

```
green sea turtle | hair = no, feathers = no, ..., fins = yes, legs = 4, ... -> reptile |
breathes = yes
mudskipper | hair = no, feathers = no, ..., fins = yes, legs = 0, ... -> fish | legs = 0
```

Biawak muncul dua kali, karena kondisi pertama yang dipilih Bu Lestari ternyata kurang tepat.

12.5.2 Programnya

Kode 12.5 menerapkan pelajaran-pelajaran itu. Inilah perulangan Ripple Down Rules dari Bab 3 dengan satu tambahan yang dimungkinkan oleh data: sebelum sebuah pelajaran disimpan, node barunya dicoba pada salinan basis pengetahuan dan diperiksa terhadap **setiap hewan yang jawabannya sudah diketahui** — ke-101 hewan dalam data dan pelajaran-pelajaran sebelumnya. Kasus-kasus cornerstone melindungi sepuluh hewan; data melindungi seratus satu.

Kode 12.5 `refine_zoo.py`: pelajaran-pelajaran sang guru, masing-masing dicoba pada salinan dan diperiksa terhadap setiap hewan yang sudah diketahui sebelum disimpan.

```
"""The biology teacher refines the learned knowledge base, case by case.

    python refine_zoo.py        (after python tree_to_kb.py)

For each lesson of ../data/lessons.txt: what the knowledge base answers now;
if it is wrong, a new node with the condition the teacher chose, as an
exception of the node that answered. The new node is tried on a copy first
and checked against all the animals of the data and every earlier lesson;
it is kept only if none of their answers change.
"""

import pathlib

import cart
from tree_to_kb import answer
from vcirs import (BOTH, TOP_LEVEL, Case, Conclusion, Condition,
                   KnowledgeBase, ScriptedAsker, report)

DATA = pathlib.Path(__file__).resolve().parents[1] / "data"


def main():
    data = cart.Data(str(DATA / "zoo.csv"))
    kb = KnowledgeBase.load(str(DATA / "zoo-kb.json"))
    checks = [(name, values, cls) for name, values, cls in data.rows]
    with open(DATA / "lessons.txt", encoding="utf-8") as f:
        lessons = [ln.strip() for ln in f
                   if ln.strip() and not ln.startswith("#")]
    for line in lessons:
        name, taught, when = [p.strip() for p in line.split("|")]
        left, cls = [p.strip() for p in taught.split("->")]
        facts = [Condition.parse(p) for p in left.split(", ")]
        values = {f.variable: f.value for f in facts}
        case = Case([Condition("animal", "=", name)] + facts,
                    [Conclusion("type", cls)])
        result = kb.infer(values)
        got = answer(kb, values)
        fired = result.paths[0].last().id if result.paths else None
        print(name + ": the knowledge base says " + got
              + (" (" + fired + ")" if fired else "") + "; the teacher says "
              + cls)
        if got == cls:
            kb.learn(case, "teacher")         # a checked case (option 0)
            checks.append((name, values, cls))
            print("  right; recorded as a checked case")
            continue
```

Kode 12.5 (Lanjutan) *refine_zoo.py: pelajaran-pelajaran sang guru, masing-masing dicoba pada salinan dan diperiksa terhadap setiap hewan yang sudah diketahui sebelum disimpan.*

```
        if fired:
            for text in kb.differences(fired, case).lines()[1:]:
                print("   " + text)
        chosen = [Condition.parse(p) for p in when.split(", ") if p]
        copy = KnowledgeBase.from_json(kb.to_json())
        node = copy.refine(case, BOTH if fired else TOP_LEVEL, fired,
                           chosen or None, "teacher")
        wrong = [n for n, v, c in checks if answer(copy, v) != c]
        print("  the teacher chooses: " + (when or "(the whole list)"))
        if wrong:
            print("  NOT taught: " + str(len(wrong)) + " checked animal(s) "
                  "would change: " + ", ".join(wrong))
            continue
        kb = copy
        checks.append((name, values, cls))
        print("  taught " + str(node) + ", under " + str(node.parent)
              + "; all " + str(len(checks) - 1) + " earlier checks unchanged")
    print()
    print(kb.summary())
    for line in report.tree(kb):
        print(line)
    right = sum(answer(kb, v) == c for _, v, c in checks)
    print("right on " + str(right) + "/" + str(len(checks)) + " checked animals")
    kb.save(str(DATA / "zoo-kb-refined.json"))
    print("saved ../data/zoo-kb-refined.json")
    print()
    asker = ScriptedAsker({"milk": "no", "feathers": "no", "fins": "no",
                           "backbone": "yes", "aquatic": "no"})
    result = kb.infer({}, asker)
    print("A gecko, question by question: "
          + ", ".join(q.variable for q in asker.questions))
    for line in result.what():
        print(line)


if __name__ == "__main__":
    main()
```

Untuk setiap pelajaran program mencetak jawaban basis pengetahuan dan node yang memberikannya. Bila jawabannya benar, kasusnya diberikan kepada `learn`, yang mencatatnya sebagai kasus yang sudah diperiksa (opsi 0), dan hewan itu masuk ke daftar pemeriksaan. Bila tidak, program mencetak daftar perbedaan antara pelajaran itu dan cornerstone node yang menjawab, membuat salinan basis pengetahuan dengan `from_json(to_json())`, menambahkan node baru ke salinan itu sebagai pengecualian node yang menjawab, dengan kondisi pilihan sang guru — opsi 3 dengan kondisi pilihan, seperti di Bab 11 — lalu bertanya kepada salinan itu tentang setiap hewan yang sudah diperiksa. Bila ada jawaban yang akan berubah, pelajarannya ditolak dan salinannya dibuang; bila tidak, salinan itu menjadi basis pengetahuannya. Di akhir program menyimpan `zoo-kb-refined.json` dan memainkan satu putaran permainan untuk seekor tokek dengan `ScriptedAsker`, yang menjawab pertanyaan-pertanyaan mesin dari sebuah tabel dan mengingat apa saja yang ditanyakan kepadanya. **Kode 12.6** adalah versi Java-nya.

Kode 12.6 `RefineZoo.java`: pelajaran-pelajaran sang guru dalam Java.

```
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

import vcirs.Case;
import vcirs.Conclusion;
import vcirs.Condition;
import vcirs.Facts;
import vcirs.InferenceResult;
import vcirs.KnowledgeBase;
import vcirs.Node;
import vcirs.Question;
import vcirs.Report;
import vcirs.ScriptedAsker;

/**
 * The biology teacher refines the learned knowledge base, case by case.
 * java RefineZoo        (after java TreeToKb)
 * For each lesson of ../data/lessons.txt: what the knowledge base answers
 * now; if it is wrong, a new node with the condition the teacher chose, as
 * an exception of the node that answered. The new node is tried on a copy
 * first and checked against all the animals of the data and every earlier
 * lesson; it is kept only if none of their answers change.
 */
public class RefineZoo {
    public static void main(String[] args) throws Exception {
        Cart.Data data = new Cart.Data("../data/zoo.csv");
        KnowledgeBase kb = KnowledgeBase.load("../data/zoo-kb.json");
        List<Cart.Row> checks = new ArrayList<>(data.rows);
        List<String> lessons = new ArrayList<>();
        for (String ln : Files.readAllLines(Path.of("../data/lessons.txt"),
                StandardCharsets.UTF_8)) {
            if (!ln.strip().isEmpty() && !ln.startsWith("#")) {
                lessons.add(ln.strip());
            }
        }
        for (String line : lessons) {
            String[] p = line.split("\\|", -1);
            String name = p[0].strip();
            String[] taught = p[1].split("->");
            String cls = taught[1].strip();
            String when = p[2].strip();
```

Kode 12.6 (Lanjutan) *RefineZoo.java: pelajaran-pelajaran sang guru dalam Java.*

```
        List<Condition> facts = new ArrayList<>();
        Map<String, String> values = new LinkedHashMap<>();
        for (String f : taught[0].strip().split(", ")) {
            Condition c = Condition.parse(f);
            facts.add(c);
            values.put(c.variable, c.value);
        }
        List<Condition> all = new ArrayList<>();
        all.add(new Condition("animal", "=", name));
        all.addAll(facts);
        Case c = new Case(all, List.of(new Conclusion("type", cls)), 100);
        InferenceResult result = kb.infer(Facts.map(values));
        String got = TreeToKb.answer(kb, values);
        String fired = result.paths.isEmpty() ? null
                : result.paths.get(0).last().id;
        System.out.println(name + ": the knowledge base says " + got
                + (fired != null ? " (" + fired + ")" : "")
                + "; the teacher says " + cls);
        if (got.equals(cls)) {
            kb.learn(c, "teacher");            // a checked case (option 0)
            checks.add(new Cart.Row(name, values, cls));
            System.out.println("  right; recorded as a checked case");
            continue;
        }
        if (fired != null) {
            List<String> lines = kb.differences(fired, c).lines();
            for (String text : lines.subList(1, lines.size())) {
                System.out.println("  " + text);
            }
        }
        List<Condition> chosen = new ArrayList<>();
        for (String part : when.split(", ")) {
            if (!part.isEmpty()) {
                chosen.add(Condition.parse(part));
            }
        }
        KnowledgeBase copy = KnowledgeBase.fromJson(kb.toJson());
        Node node = copy.refine(c, fired != null ? KnowledgeBase.BOTH
                : KnowledgeBase.TOP_LEVEL, fired,
                chosen.isEmpty() ? null : chosen, "teacher");
        List<String> wrong = new ArrayList<>();
        for (Cart.Row r : checks) {
            if (!TreeToKb.answer(copy, r.values()).equals(r.cls())) {
                wrong.add(r.name());
            }
        }
```

Kode 12.6 (Lanjutan) *RefineZoo.java: pelajaran-pelajaran sang guru dalam Java.*

```
            System.out.println("  the teacher chooses: "
                    + (when.isEmpty() ? "(the whole list)" : when));
            if (!wrong.isEmpty()) {
                System.out.println("  NOT taught: " + wrong.size()
                        + " checked animal(s) would change: "
                        + String.join(", ", wrong));
                continue;
            }
            kb = copy;
            checks.add(new Cart.Row(name, values, cls));
            System.out.println("  taught " + node + ", under " + node.parent
                    + "; all " + (checks.size() - 1)
                    + " earlier checks unchanged");
        }
        System.out.println();
        System.out.println(kb.summary());
        for (String line : Report.tree(kb)) {
            System.out.println(line);
        }
        int right = 0;
        for (Cart.Row r : checks) {
            right += TreeToKb.answer(kb, r.values()).equals(r.cls()) ? 1 : 0;
        }
        System.out.println("right on " + right + "/" + checks.size()
                + " checked animals");
        kb.save("../data/zoo-kb-refined.json");
        System.out.println("saved ../data/zoo-kb-refined.json");
        System.out.println();
        ScriptedAsker asker = new ScriptedAsker().answer("milk", "no")
                .answer("feathers", "no").answer("fins", "no")
                .answer("backbone", "yes").answer("aquatic", "no");
        InferenceResult r = kb.infer(Facts.none(), asker);
        List<String> asked = new ArrayList<>();
        for (Question q : asker.questions) {
            asked.add(q.variable);
        }
        System.out.println("A gecko, question by question: "
                + String.join(", ", asked));
        for (String line : r.what()) {
            System.out.println(line);
        }
    }
}
```

12.5.3 Apa yang terjadi

Kedua bahasa mencetak baris-baris yang sama dan menulis `zoo-kb-refined.json` yang sama:

```
komodo dragon: the knowledge base says reptile (reptile#2); the teacher says reptile
  right; recorded as a checked case
cassowary: the knowledge base says bird (bird#1); the teacher says bird
  right; recorded as a checked case
green sea turtle: the knowledge base says fish (fish#1); the teacher says reptile
    A (only in the old case): animal = bass, predator = yes, toothed = yes, breathes = no,
legs = 0, catsize = no
    B (only in the new case): animal = green sea turtle, predator = no, toothed = no,
breathes = yes, legs = 4, catsize = yes
  the teacher chooses: breathes = yes
  taught reptile#3: breathes = yes -> type = reptile, under fish#1; all 103 earlier checks
unchanged
water monitor: the knowledge base says amphibian (amphibian#1); the teacher says reptile
    A (only in the old case): animal = frog, tail = no, catsize = no
    B (only in the new case): animal = water monitor, tail = yes, catsize = yes
  the teacher chooses: tail = yes
  NOT taught: 1 checked animal(s) would change: newt
water monitor: the knowledge base says amphibian (amphibian#1); the teacher says reptile
    A (only in the old case): animal = frog, tail = no, catsize = no
    B (only in the new case): animal = water monitor, tail = yes, catsize = yes, skin =
scales
  the teacher chooses: skin = scales
  taught reptile#4: skin = scales -> type = reptile, under amphibian#1; all 104 earlier
checks unchanged
mudskipper: the knowledge base says reptile (reptile#3); the teacher says fish
    A (only in the old case): animal = green sea turtle, predator = no, toothed = no, legs
= 4, catsize = yes
    B (only in the new case): animal = mudskipper, predator = yes, toothed = yes, legs = 0,
catsize = no
  the teacher chooses: legs = 0
  taught fish#2: legs = 0 -> type = fish, under reptile#3; all 105 earlier checks unchanged
firefly: the knowledge base says insect (insect#1); the teacher says insect
  right; recorded as a checked case

Animal classes: 18 variables, 13 nodes, 13 rules
(root)
+- mammal#1: milk = yes -> type = mammal
+- bird#1: milk = no, feathers = yes -> type = bird
+- fish#1: milk = no, feathers = no, fins = yes -> type = fish
|  +- reptile#3: breathes = yes -> type = reptile
|     +- fish#2: legs = 0 -> type = fish
+- amphibian#1: milk = no, feathers = no, fins = no, backbone = yes, aquatic = yes, eggs =
yes -> type = amphibian
|  +- reptile#4: skin = scales -> type = reptile
+- reptile#1: milk = no, feathers = no, fins = no, backbone = yes, aquatic = yes, eggs = no
-> type = reptile
+- reptile#2: milk = no, feathers = no, fins = no, backbone = yes, aquatic = no -> type =
reptile
+- insect#1: milk = no, feathers = no, fins = no, backbone = no, airborne = yes -> type =
insect
+- invertebrate#1: milk = no, feathers = no, fins = no, backbone = no, airborne = no,
predator = yes -> type = invertebrate
+- invertebrate#2: milk = no, feathers = no, fins = no, backbone = no, airborne = no,
predator = no, legs <= 3 -> type = invertebrate
+- insect#2: milk = no, feathers = no, fins = no, backbone = no, airborne = no, predator =
no, legs > 3 -> type = insect
right on 107/107 checked animals
saved ../data/zoo-kb-refined.json

A gecko, question by question: milk, feathers, fins, backbone, aquatic
Conclusion: type = reptile  (rule reptile#2, CF 100)
```

Ikutilah pelajaran-pelajarannya satu per satu.

Komodo dan kasuari dijawab dengan benar oleh pengetahuan hasil belajar: kadal besar yang berbisa adalah hewan darat bertulang belakang tanpa susu, bulu burung, atau sirip, dan burung

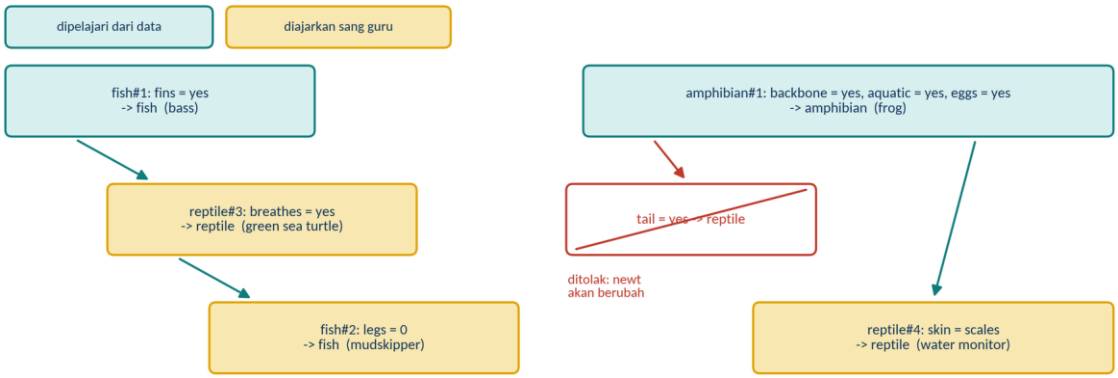
yang tidak dapat terbang tetap berbulu. Data tidak pernah melihat keduanya, tetapi aturannya menggeneralisasi dengan baik di sini. Keduanya dicatat sebagai kasus yang sudah diperiksa, dan sejak itu keduanya terlindungi seperti ke-101 hewan lainnya.

Penyu hijau disebut ikan: ia tidak menyusui dan tidak berbulu burung, dan berenang dengan sirip, yang oleh Bu Lestari dicatat sebagai `fins = yes`. Daftar perbedaan terhadap cornerstone `fish#1`, ikan bass, memperlihatkan apa yang membedakan keduanya: penyu tidak berburu, tidak bergigi, *bernapas dengan udara*, berkaki empat, dan lebih besar daripada kucing. Ia memilih satu fakta yang akan disebut seorang ahli biologi sebagai alasannya: `breathes = yes`. Node baru `reptile#3` menjadi pengecualian `fish#1`, dan ke-103 pemeriksaan sebelumnya tetap mendapat jawaban yang sama — tidak satu pun dari 13 ikan dalam data bernapas dengan udara.

Biawak disebut amfibi: seperti katak ia hidup di air, bertelur, dan bertulang belakang. Perbedaannya dari katak singkat saja: berekor, dan berukuran lebih besar. Bu Lestari memilih `tail = yes` — dan program menolaknya: **newt**, seekor amfibi dalam data, juga berekor dan hidup di air, dan dengan pengecualian ini ia akan menjadi reptil. Inilah saat data membuktikan nilainya. Cornerstone `amphibian#1` adalah katak, yang tidak berekor; pemeriksaan RDR murni yang hanya memakai cornerstone akan menerima pelajaran itu dan diam-diam merusak jawaban untuk newt. Data lebih tahu.

Biawak sekali lagi. Yang benar-benar membedakan reptil dari amfibi sama sekali tidak ada di antara ke-16 fakta: reptil berkulit kering dan bersisik, amfibi berkulit lembap tanpa sisik. Bu Lestari menambahkan fakta baru, `skin = scales`, yang tidak dimiliki satu pun dari ke-101 hewan, dan memilihnya sebagai kondisi. Tidak ada hal lain yang dapat berubah, karena tidak ada hewan yang sudah diperiksa yang memiliki fakta itu, dan `reptile#4` ditambahkan di bawah `amphibian#1`. Kini basis pengetahuan memiliki variabel yang tidak pernah dimiliki data — sang pakar telah memperluas kosakatanya, hal yang tidak mungkin dilakukan oleh pembelajaran dari tabel.

Ikan glodok menguji pelajaran tentang penyu. Ia ikan yang menghabiskan sebagian besar hidupnya di luar air dan bernapas dengan udara melalui kulit dan mulutnya, sehingga node baru `reptile#3` menyebutnya reptil. Perbedaannya dari cornerstone penyu jelas: ia berburu, bergigi, kecil, dan **tidak berkaki**. Bu Lestari memilih `legs = 0`, dan `fish#2` menjadi pengecualian `reptile#3`, yang sendirinya merupakan pengecualian `fish#1`: pengetahuan yang beriak ke bawah (*ripple down*) sampai tiga tingkat, setiap tingkat dibenarkan oleh hewan nyata. **Gambar 12.3** memperlihatkan dua cabang yang diubah sang guru.



Gambar 12.3 Pengecualian-pengecualian dari sang guru. Di bawah node ikan hasil belajar, pelajaran tentang penyu dan, di bawahnya, tentang ikan glodok; di bawah node amfibi hasil belajar, pelajaran tentang ekor yang ditolak dan pelajaran tentang sisik yang disimpan. Setiap node menyebut hewan cornerstone-nya.

Kunang-kunang adalah serangga yang terbang, dan node hasil belajar untuk hewan terbang tanpa tulang belakang menjawabnya dengan benar.

Pada akhirnya basis pengetahuan memiliki 13 node — sepuluh dipelajari dari data, tiga diajarkan sang guru — dan menjawab ke-107 hewan yang sudah diperiksa dengan benar. Baris-baris terakhir memperlihatkannya memainkan permainan: untuk seekor tokek ia bertanya tentang susu, bulu burung, sirip, tulang belakang, dan air, lima pertanyaan, lalu menyimpulkan *reptile* dari node hasil belajar `reptile#2`.

12.5.4 Permainan di klub

Pada Sabtu berikutnya klub memainkan permainan itu dengan basis pengetahuan yang sudah disempurnakan. Seorang anak yang memotret tokek di dinding kebun buah menjawab lima pertanyaan — tidak menyusui, tidak berbulu burung, tidak bersirip, bertulang belakang, tidak hidup di air — dan program berkata *reptile*, beserta alasannya: node yang menyala, dengan kelima fakta yang membuatnya menyala. Penjelasan itulah bagian yang paling disukai anak-anak, karena penjelasan mengubah jawaban menjadi sebuah argumen kecil yang dapat mereka periksa dengan foto mereka sendiri. Ketika programnya keliru, atau tidak berkata apa-apa, hasilnya justru lebih baik: sang anak dan gurunya melihat daftar perbedaan bersama-sama, dan sang anak mengusulkan fakta yang seharusnya menjadi pembedanya. Bu Lestari memutuskan apakah fakta itu tepat, dan pemeriksaan terhadap semua hewan yang sudah diketahui memberi tahu keduanya, dalam satu baris, apakah pelajaran itu akan merusak sesuatu.

Tiga kebiasaan membuat permainan ini berjalan di kelas. Sang guru menyimpan berkas pelajaran dan menambahkan setiap pelajaran sebagai satu baris, dengan tanggalnya dalam komentar, sehingga kelas dapat melihat pengetahuannya tumbuh sepanjang semester. Fakta-fakta baru, seperti `skin = scales`, ditulis seperti anak-anak mengucapkannya, dan pertanyaan untuk setiap fakta baru ditambahkan ke daftar pertanyaan agar program menanyakannya dengan kata-kata, bukan dengan nama variabel. Dan sebulan sekali kelas menjalankan ulang seluruh rantainya — pohon dari data, lalu semua pelajaran — untuk melihat bahwa setiap pelajaran masih berlaku; hanya butuh sedetik, dan itulah pelajaran terbaik tentang apa artinya *mengetahui* sesuatu.

12.6 Langkah 4: Pemeriksaan terhadap scikit-learn

Kita menulis pembelajar pohon sendiri untuk memahaminya. Dalam praktik, banyak orang memakai **scikit-learn**, pustaka pembelajaran mesin yang biasanya dijumpai pertama kali oleh pemrogram Python [8]; versi terkininya, per September 2026, adalah 1.9.1, dirilis 10 September 2026 [9]. `DecisionTreeClassifier` miliknya mengimplementasikan CART versi yang dioptimalkan, sehingga ia menjadi pemeriksa yang alami bagi kode kita. **Kode 12.7** memberinya ke-101 hewan yang sama, dengan *yes* ditulis 1 dan *no* ditulis 0, lalu membandingkan kedua pohon dan jawaban leave-one-out keduanya. Program ini hanya ditulis dalam Python, karena pemeriksaan ini tentang kode Python dan Java kita yang menghasilkan pohon yang sama; setelah pohonnya diketahui, Java tidak punya tambahan apa pun.

Kode 12.7 `sklearn_check.py`: pohon kita dibandingkan dengan `DecisionTreeClassifier` milik scikit-learn.

```
"""A check of our CART against scikit-learn's DecisionTreeClassifier.

    python -m pip install scikit-learn     (once; Python only)
    python sklearn_check.py

Both learn from the same 101 animals, with yes = 1 and no = 0. We compare
the trees' sizes, the rows they learned from, and leave-one-out answers.
"""

import pathlib

import sklearn
from sklearn.model_selection import LeaveOneOut, cross_val_predict
from sklearn.tree import DecisionTreeClassifier, export_text

import cart

DATA = pathlib.Path(__file__).resolve().parents[1] / "data"


def main():
    data = cart.Data(str(DATA / "zoo.csv"))
    X = [[float(v[a]) if a in data.numeric else float(v[a] == "yes")
          for a in data.attributes] for _, v, _ in data.rows]
    y = [c for _, _, c in data.rows]
    print("scikit-learn " + sklearn.__version__)

    ours = cart.grow(data, data.rows)
    theirs = DecisionTreeClassifier(criterion="gini", random_state=0)
    theirs.fit(X, y)
    print("our tree:          " + str(len(cart.paths(ours))) + " leaves, depth "
          + str(cart.depth(ours)))
    print("scikit-learn tree: " + str(theirs.get_n_leaves()) + " leaves, depth "
          + str(theirs.get_depth()))
    print("scikit-learn, right on the rows it learned from: "
          + str(sum(p == c for p, c in zip(theirs.predict(X), y))) + "/101")
    print()
    print(export_text(theirs, feature_names=data.attributes, decimals=1))
```

Kode 12.7 (Lanjutan) *sklearn_check.py: pohon kita dibandingkan dengan DecisionTreeClassifier milik scikit-learn.*

```
    loo = cross_val_predict(DecisionTreeClassifier(criterion="gini",
                                                   random_state=0),
                            X, y, cv=LeaveOneOut())
    mine = []
    for i, row in enumerate(data.rows):
        t = cart.grow(data, data.rows[:i] + data.rows[i + 1:])
        mine.append(cart.predict(t, row[1]))
    print("leave-one-out, ours:         " + str(sum(
        p == c for p, c in zip(mine, y))) + "/101")
    print("leave-one-out, scikit-learn: " + str(sum(
        p == c for p, c in zip(loo, y))) + "/101")
    print("the two agree on " + str(sum(a == b for a, b in zip(mine, loo)))
          + "/101 animals")
    for (name, _, c), a, b in zip(data.rows, mine, loo):
        if a != c or b != c:
            print("  " + name + " (" + c + "): ours " + a + ", scikit-learn "
                  + b)


if __name__ == "__main__":
    main()
```

Pasanglah scikit-learn sekali dengan `python -m pip install -r requirements.txt` di folder `python`, lalu jalankan `python sklearn_check.py`:

```
scikit-learn 1.9.1
our tree:          10 leaves, depth 7
scikit-learn tree: 10 leaves, depth 7
scikit-learn, right on the rows it learned from: 101/101

|--- milk <= 0.5
|   |--- feathers <= 0.5
|   |   |--- fins <= 0.5
|   |   |   |--- backbone <= 0.5
|   |   |   |   |--- airborne <= 0.5
|   |   |   |   |   |--- predator <= 0.5
|   |   |   |   |   |   |--- legs <= 3.0
|   |   |   |   |   |   |   |--- class: invertebrate
|   |   |   |   |   |   |--- legs >  3.0
|   |   |   |   |   |   |   |--- class: insect
|   |   |   |   |   |--- predator >  0.5
|   |   |   |   |   |   |--- class: invertebrate
|   |   |   |   |--- airborne >  0.5
|   |   |   |   |   |--- class: insect
|   |   |   |--- backbone >  0.5
|   |   |   |   |--- aquatic <= 0.5
|   |   |   |   |   |--- class: reptile
|   |   |   |   |--- aquatic >  0.5
|   |   |   |   |   |--- legs <= 2.0
|   |   |   |   |   |   |--- class: reptile
|   |   |   |   |   |--- legs >  2.0
|   |   |   |   |   |   |--- class: amphibian
|   |   |--- fins >  0.5
|   |   |   |--- class: fish
|   |--- feathers >  0.5
|   |   |--- class: bird
|--- milk >  0.5
|   |--- class: mammal

leave-one-out, ours:         96/101
leave-one-out, scikit-learn: 95/101
the two agree on 100/101 animals
  ladybird (insect): ours invertebrate, scikit-learn invertebrate
  newt (amphibian): ours reptile, scikit-learn reptile
  scorpion (invertebrate): ours insect, scikit-learn insect
  seasnake (reptile): ours amphibian, scikit-learn amphibian
  tortoise (reptile): ours insect, scikit-learn insect
  tuatara (reptile): ours reptile, scikit-learn amphibian
```

Kedua pohon berukuran sama dan berbentuk sama. scikit-learn menuliskan setiap pertanyaan ya/tidak sebagai `feature <= 0.5` (*no* bernilai 0, jadi "<= 0.5" berarti *no*) dan mendaftar cabang *no* lebih dahulu; bila dibaca dengan cara itu, pertanyaan-pertanyaannya sama dengan pertanyaan kita: susu, bulu burung, sirip, tulang belakang, lalu air dan terbang, lalu berburu dan kaki. Satu-satunya perbedaan adalah satu pertanyaan di bagian bawah. Untuk kelima hewan bertulang belakang berdarah dingin yang hidup di air, scikit-learn memisahkan satu-satunya reptil dengan `legs <= 2`, dan kita dengan `eggs = yes`. Kedua pertanyaan membagi kelompok itu dengan sempurna, sehingga ketidakmurniannya persis sama. Kita memutuskan seri itu dengan mengambil atribut yang muncul lebih dahulu di tabel (`eggs` kolom ketiga, `legs` kolom ketiga belas); scikit-learn mencoba atribut-atribut dalam urutan acak yang ditetapkan oleh `random_state`-nya dan mempertahankan yang pertama di antara yang sama baiknya. Tidak ada pilihan yang salah. Data tidak mengatakan mana yang lebih baik; seorang ahli biologi akan berkata bahwa telur adalah alasan yang lebih baik daripada kaki.

Jawaban leave-one-out keduanya sepakat untuk 100 dari 101 hewan. Satu-satunya perbedaan adalah tuatara, reptil dari Selandia Baru, yang tetap disebut reptil oleh pohon kita ketika belum pernah melihatnya dan disebut amfibi oleh scikit-learn — seri yang sama lagi, diputuskan ke arah sebaliknya. Kedua pembelajar keliru pada lima hewan yang sama, dan scikit-learn juga pada tuatara; hal itu menenangkan: program kecil kita berperilaku seperti pustaka itu, dan apa yang terlewat oleh keduanya adalah persis alasan kita membutuhkan pakar. (scikit-learn 1.8.0, rilis sebelum 1.9, mencetak hasil yang sama.)

12.6.1 Pengujian

`CartTest.java` dan `test_cart.py` memeriksa bahwa pertanyaan pertama adalah tentang susu dan bahwa pohonnya berdaun sepuluh dengan kedalaman 7; bahwa pada tabel empat baris, pertanyaan yang memisahkan golongan-golongannya dengan sempurna yang dipilih; bahwa basis pengetahuan hasil pengubahan menjawab setiap hewan persis seperti pohonnya; dan, dengan menjalankan pengubahan dan pelajaran-pelajarannya, bahwa pelajaran tentang ekor ditolak dan ke-107 hewan yang sudah diperiksa dijawab benar di akhir. Pengujian Java mencetak:

```
ok: the first question is about milk
ok: 10 leaves, depth 7
ok: the purest question wins: a, not b
ok: the knowledge base answers like the tree
ok: the tail lesson is refused
ok: all 107 checked animals are right
all tests passed
```

12.7 Membuatnya Lebih Baik

Entropi sebagai ganti Gini. Mengganti `impurity` dengan entropi dari ID3 dan C4.5 adalah perubahan kecil di kedua bahasa, dan Soal Latihan 12.3 melakukannya. Entropi melibatkan logaritma, sehingga aritmetika tepat dari Persamaan (12.3) hilang; agar Java dan Python tetap identik, bandingkan skornya melalui `Num.key` milik vcirs-core, yang menganggap sama bilangan-bilangan yang hanya berbeda pada digit-digit terakhir, dan putuskan seri menurut kolom seperti sebelumnya.

Pohon yang lebih kecil. Pohon yang ditumbuhkan sampai setiap daunnya murni dapat mempelajari kebetulan-kebetulan dalam data: daun berisi satu hewan adalah aturan dengan satu contoh. Obat yang lazim adalah jumlah baris minimum per daun dan kedalaman maksimum; daun-daunnya lalu tetap tidak murni, dan Persamaan (12.4) memberi node-node itu faktor kepastian di bawah 100, yang dilaporkan mesin bersama setiap jawaban. Dengan data Zoo tidak banyak yang diperoleh, tetapi dengan catatan yang berderau — penanganan yang salah dicatat, gejala yang salah dilaporkan — daun berisi 30 catatan dengan CF 90 lebih dapat dipercaya daripada tiga daun berisi 10 catatan dengan CF 100.

Hasil belajar dan hasil ajaran, berdampingan. Setiap node mencatat siapa pembuatnya: `CART` untuk yang dipelajari, `teacher` untuk pelajaran. Ketika datanya bertambah — klub tahun depan menambahkan lima puluh hewan — pohonnya dapat dipelajari ulang, dan pelajaran-pelajaran sang guru diputar ulang pada basis pengetahuan yang baru, masing-masing diperiksa seperti sebelumnya. Menyimpan pelajaran-pelajaran itu sebagai berkas, seperti `lessons.txt`, itulah yang membuat hal ini murah: pengetahuan sang pakar juga berupa data.

Pembangkitan rule dan pemakaian. Basis pengetahuan hasil belajar adalah basis pengetahuan VCIRS biasa, sehingga tingkat pemakaian dari Bab 7 dan pembangkitan rule dari Bab 6 berlaku

untuknya. Kondisi-kondisi awal yang panjang pada setiap node — `milk = no, feathers = no, fins = no` — dipakai bersama oleh banyak node, dan itulah persis yang ditemukan analisis variabel VCIRS dan yang diringkas oleh pembangkitan node-nya.

Pertanyaan dalam urutan yang lebih baik. Basis pengetahuan bertanya menurut urutan node-nya, yaitu urutan lintasan-lintasan pohon. Untuk permainan bersama anak-anak, mengajukan pertanyaan yang paling informatif lebih dahulu — yaitu pertanyaan pertama pohon itu sendiri, susu — sudah tepat. Untuk data lain, urutan pertanyaan dari Bab 7 dapat berbuat lebih baik.

> **JEBAKAN:** Pohon mempelajari apa pun yang ada di dalam data, termasuk kekeliruan dan biasnya. Bila catatannya dikumpulkan dengan ceroboh, atau hanya pada satu musim, atau hanya dari sebagian pelanggan, aturannya akan keliru dengan cara yang sama, dan tampak sama meyakinkannya. Selalu mintalah seorang pakar membaca aturan hasil belajar sebelum ada yang mengandalkannya, dan periksalah pada kasus-kasus yang tidak ada di dalam data.

12.8 Memakainya Sungguh-Sungguh

Catatan Anda sebagai tabel. `cart.py` dan `Cart.java` membaca berkas CSV apa pun yang kolom pertamanya berupa nama atau nomor, kolom terakhirnya jawaban, dan kolom-kolom lainnya fakta. Konsultasi laundry dari Bab 10, yang diekspor dengan satu kueri SQL, adalah tabel seperti itu; begitu pula catatan perbaikan sebuah bengkel, keluhan sebuah toko, masalah tanaman sebuah kebun. Untuk melihat apa yang terjadi dengan tabel kecil, `data/stain-records.csv` memuat dua puluh catatan noda rekaan seperti yang disimpan laundry di Bab 10. `python cart.py ../data/stain-records.csv` mencetak:

```
20 rows, 4 attributes (numeric: none)

fabric = silk: dry cleaner (4)
fabric <> silk:
|  stain = oil: dish soap (3)
|  stain <> oil:
|  |  stain = tea:
|  |  |  color = white:
|  |  |  |  age = fresh: cool soak (1)
|  |  |  |  age = dried: oxygen bleach (1)
|  |  |  color = colored: cool soak (2)
|  |  stain <> tea:
|  |  |  stain = ink: alcohol (2)
|  |  |  stain <> ink:
|  |  |  |  stain = mud: brush (2)
|  |  |  |  stain <> mud:
|  |  |  |  |  age = fresh: cold rinse (2)
|  |  |  |  |  age = dried:
|  |  |  |  |  |  stain = wax: scrape and iron (1)
|  |  |  |  |  |  stain = blood: enzyme soak (2)

leaves: 10, depth: 7, right on the rows it learned from: 20/20

Leave-one-out: learn from all rows but one, then ask about it.
right: 13/20
  5: cold rinse, tree says enzyme soak
  9: oxygen bleach, tree says cool soak
  11: cool soak, tree says oxygen bleach
  12: cool soak, tree says oxygen bleach
  13: alcohol, tree says cold rinse
  14: alcohol, tree says enzyme soak
  18: scrape and iron, tree says alcohol
```

Pertanyaan pertama yang ditemukannya adalah pertanyaan yang akan diajukan Bu Rini lebih dahulu: *apakah bahannya sutra?* — setiap barang sutra dikirim ke *dry cleaner*. Namun dengan dua puluh catatan pohon juga mempelajari kebetulan: di bagian bawah ia memisahkan lilin dari darah hanya karena keduanya adalah dua noda terakhir yang tersisa, dan skor leave-one-out-nya hanya 13 dari 20. Dua puluh catatan cukup untuk menyarankan aturan kepada pakar, tidak untuk menggantikannya. Data Zoo, dengan 101 contoh bersih dari tujuh golongan yang terpisah dengan baik, mencapai 96 dari 101; catatan sungguhan sebuah usaha kecil biasanya berada di antaranya.

Berapa contoh yang dibutuhkan? Tidak ada angka pasti, tetapi dua patokan membantu. Setiap golongan sebaiknya punya paling sedikit segelintir contoh — sepuluh itu nyaman, tiga itu rapuh — karena daun yang dibangun dari satu atau dua contoh hanyalah tebakan. Dan skor leave-one-out memberi tahu Anda, lebih baik daripada patokan apa pun, apakah contohnya sudah cukup: bila menyisihkan satu catatan sering mengubah jawaban untuk catatan itu, tabelnya terlalu kecil untuk pertanyaan-pertanyaannya, dan aturan hasil belajar sebaiknya diperlakukan sebagai saran bagi pakar, bukan sebagai pengetahuan. Data Zoo, dengan 41 mamalia dan 4 amfibi, memperlihatkan kedua sisi dalam satu tabel: aturan mamalia kokoh, dan aturan amfibi-lah yang harus diperbaiki sang guru.

Pilih fakta yang dapat dijawab orang. Sebuah aturan hanya berguna bila seseorang di meja pelayanan dapat menjawab pertanyaan-pertanyaannya. Kolom yang baru diketahui setelah pekerjaan selesai (berapa lama perbaikannya) tidak boleh dipakai sebagai fakta untuk meramalkan pekerjaan itu: pohon akan menemukannya, dan aturannya tidak berguna sebelum pekerjaan dimulai. Buanglah kolom-kolom seperti itu sebelum belajar.

Nilai dengan leave-one-out, putuskan bersama pakar. Leave-one-out memberikan perkiraan yang jujur tentang seberapa sering aturan hasil belajar akan benar pada kasus-kasus baru; daftar kekeliruannya adalah daftar pertanyaan terbaik untuk pertemuan pertama dengan pakar. Ubahlah pohonnya, biarkan pakar membaca setiap node beserta kasus cornerstone-nya, dan ajarkan koreksi-koreksinya sebagai pelajaran, masing-masing diperiksa terhadap semua catatan.

Jauhkan orang dari tabel. Pembelajaran membutuhkan fakta tentang keadaan, bukan tentang orang. Nama, nomor telepon, dan alamat pelanggan tidak termasuk kolom-kolomnya, dan kolom seperti “pelanggan” akan membuat pohon mempelajari aturan tentang individu, yang tidak berguna dan, menurut Undang-Undang Pelindungan Data Pribadi di Indonesia, merupakan pemakaian data pribadi yang tujuannya tidak pernah disampaikan kepada siapa pun [10].

Bukan untuk diagnosis. Mempelajari aturan dari rekam kesehatan adalah bidang serius dengan metode, kendali, dan peraturannya sendiri. Alat-alat bab ini untuk memilah, menggolongkan, dan memberi saran di tempat kekeliruan murah dan terlihat — golongan seekor hewan, penanganan sebuah noda, bagian yang menangani sebuah keluhan — dan untuk menyarankan aturan kepada pakar, tidak pernah untuk menggantikan penilaian mereka di tempat kekeliruan dapat mencelakai seseorang.

12.9 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Menilai pohon dengan datanya sendiri. 101 dari 101 untuk hewan-hewan yang menjadi bahan belajarnya tidak mengatakan apa pun tentang hewan-hewan baru. *Hindari:* pakai leave-one-out, atau sisihkan sebagian kasus yang tidak pernah dilihat pohon.

Memercayai aturan karena program yang menemukannya. Pohon tidak tahu biologi; ia tahu tabel. *Hindari:* mintalah pakar membaca setiap node hasil belajar sebelum dipakai, dimulai dari kekeliruan leave-one-out.

Memeriksa koreksi hanya terhadap cornerstone. Pelajaran pertama tentang biawak lolos pemeriksaan cornerstone dan akan merusak jawaban untuk newt. *Hindari:* bila Anda punya data, periksalah setiap koreksi terhadap seluruhnya.

Memilih kondisi yang benar, tetapi bukan alasannya. `tail = yes` memang benar untuk biawak, tetapi alasan ia reptil adalah kulitnya yang bersisik. *Hindari:* pilih dari daftar perbedaan fakta yang menjelaskan golongannya; tambahkan fakta baru bila alasannya tidak ada di antara kolom-kolomnya.

Membiarkan sebuah kolom membocorkan jawaban. Kolom yang dicatat setelah kejadian, atau yang sebenarnya adalah jawabannya dengan nama lain, menghasilkan pohon sempurna yang tidak berguna dalam praktik. *Hindari:* pertahankan hanya fakta yang sudah diketahui ketika pertanyaannya diajukan.

Ejaan yang berbeda di dalam data. `yes`, `Yes`, dan `1` adalah tiga nilai bagi pembelajar, dan tiga cabang bagi pohon. *Hindari:* bersihkan tabelnya lebih dahulu; salinan Zoo kita menuliskan setiap nilai seperti itu sebagai `yes` atau `no`.

Seri dengan bilangan titik mengambang. Dua program yang menghitung ketidakmurnian dengan bilangan titik mengambang dapat memutuskan seri secara berbeda dan menumbuhkan pohon yang berbeda. *Hindari:* hitunglah secara tepat dengan bilangan bulat, seperti Persamaan (12.3), atau bandingkan dengan toleransi dan putuskan seri dengan aturan tetap.

Belajar sekali lalu melupakan pelajarannya. Mempelajari ulang pohon dari data baru sambil membuang koreksi-koreksi sang pakar berarti kehilangan pengetahuan yang paling berharga. *Hindari:* simpan pelajaran-pelajaran itu sebagai berkas dan putar ulang setelah setiap pembelajaran ulang.

12.10 Ringkasan Bab

- Pengetahuan dapat **dipelajari dari data** lalu **disempurnakan oleh pakar**: pembelajaran memberikan cakupan yang luas dengan murah, pakar memberikan presisi di tempat yang penting (**Gambar 12.1**).
- **Pohon keputusan** mengajukan satu pertanyaan per node; **CART** memilih setiap pertanyaan menurut **ketidakmurnian Gini** tertimbang terendah dari dua kelompok yang dihasilkannya (Persamaan (12.1), Persamaan (12.2), **Tabel 12.1**); membandingkan ketidakmurnian sebagai pecahan bilangan bulat (Persamaan (12.3)) membuat Java dan Python menumbuhkan pohon yang sama.
- CART kita (**Kode 12.1**, **Kode 12.2**) mempelajari pohon berdaun sepuluh dari 101 hewan dalam data Zoo (**Gambar 12.2**); pohon itu benar untuk semuanya dan untuk 96 dari 101 menurut **leave-one-out**, pengujian yang jujur.
- Setiap lintasan pohon menjadi **node VCIRS** tingkat atas dengan lintasannya sebagai kondisi, golongannya sebagai konklusi, **faktor kepastian** dari kemurnian daunnya (Persamaan (12.4)), dan hewan pertama di daun itu sebagai **kasus cornerstone** (**Kode 12.3**, **Kode 12.4**).
- Pelajaran-pelajaran sang guru (**Kode 12.5**, **Kode 12.6**) adalah pengecualian RDR yang diperiksa terhadap **setiap hewan yang sudah diketahui**: pemeriksaan itu menolak sebuah pelajaran

yang akan diterima bila hanya memakai cornerstone, dan sang guru menambahkan fakta yang tidak pernah dimiliki data (**Gambar 12.3**).

- scikit-learn 1.9.1 menumbuhkan pohon dengan ukuran dan bentuk yang sama; satu-satunya perbedaan adalah sebuah seri yang diputuskan ke arah lain, dan jawaban leave-one-out keduanya sepakat untuk 100 dari 101 hewan (**Kode 12.7**).

12.11 Soal Latihan

12.1 Dengan Persamaan (12.1) dan Persamaan (12.2), hitunglah dengan tangan ketidakmurnian Gini tertimbang pertanyaan *feathers = yes* untuk ke-101 hewan, lalu cocokkan dengan **Tabel 12.1**.

12.2 Dalam kelompok 60 hewan yang tidak menyusui, pertanyaan mana yang dipilih CART, dan mengapa bukan *eggs = yes*? Hitunglah ketidakmurnian *feathers = yes* dan *eggs = yes* dalam kelompok itu.

12.3 Ganti indeks Gini dengan entropi, $H = -\sum_k p_k \log_2 p_k$, di `cart.py`. Apakah pohon untuk data Zoo berubah? Apa yang harus Anda lakukan agar versi Java menumbuhkan pohon yang sama?

12.4 Dengan **Gambar 12.2**, golongkan dengan tangan: (a) lumba-lumba (menyusui); (b) penguin; (c) kalajengking (tidak bertulang belakang, tidak terbang, berburu, berkaki delapan); (d) siput (tidak bertulang belakang, tidak terbang, tidak berburu, tidak berkaki). Mana di antaranya yang termasuk data Zoo itu sendiri?

12.5 Sebuah daun memuat 9 hewan: 7 serangga dan 2 hewan tak bertulang belakang. Golongan dan faktor kepastian apa yang diperoleh node hasil pengubahannya menurut Persamaan (12.4)? Apa yang dilaporkan mesin untuk hewan yang sampai ke sana?

12.6 Jelaskan mengapa pelajaran tentang ekor untuk biawak lolos pemeriksaan cornerstone tetapi tidak lolos pemeriksaan data. Apa artinya bagi pemeriksaan koreksi dalam basis pengetahuan yang dibangun tanpa data, seperti di Bab 3 sampai 11?

12.7 Kelelawar buah (`fruitbat` dalam data: menyusui, terbang) dan platipus (menyusui, bertelur, berparuh bebek) sama-sama ada di data Zoo. Node mana yang menjawab masing-masing, dan seperti apa daftar perbedaannya bila sang guru mencoba mengajarkan bahwa platipus itu istimewa?

12.8 Jalankan `cart.py` pada `stain-records.csv` dan bacalah kekeliruan leave-one-out-nya. Mana yang akan dikoreksi pakar sebagai pengecualian, dan mana yang menunjukkan bahwa tabelnya memang terlalu kecil?

12.9 Usulkan dua cara memutuskan seri di bagian bawah pohon (*eggs* atau *legs*) yang lebih baik daripada "kolom pertama yang menang", dan sebutkan apa yang dibutuhkan masing-masing.

12.12 Penyelesaian Soal Latihan

Penyelesaian 12.1. *feathers = yes* mengirim 20 burung ke kelompok ya ($G_{\text{ya}} = 0$). Kelompok tidak berisi 81 hewan: 41 mamalia, 13 ikan, 10 hewan tak bertulang belakang, 8 serangga, 5 reptil, 4 amfibi, sehingga $\sum c^2 = 1681 + 169 + 100 + 64 + 25 + 16 = 2055$ dan $G_{\text{tidak}} = 1 - 2055/6561 = 0.6868$. Menurut Persamaan (12.2), $G_{\text{split}} = \frac{81}{101} \cdot 0.6868 = 0.5508$, seperti pada **Tabel 12.1**: lebih buruk daripada susu, karena kelompok murninya lebih kecil dan kelompok campurannya tetap memuat 41 mamalia.

Penyelesaian 12.2. Ke-60 hewan itu terdiri atas 20 burung, 13 ikan, 10 hewan tak bertulang belakang, 8 serangga, 5 reptil, dan 4 amfibi, $G = 1 - 774/3600 = 0.785$. *feathers = yes* menghasilkan 20 burung (murni) dan 40 hewan lain dengan jumlah per golongan 13, 10, 8, 5, 4: $\sum c^2 = 374$, $G_{\text{tidak}} = 1 - 374/1600 = 0.766$, sehingga $G_{\text{split}} = \frac{40}{60} \cdot 0.766 = 0.511$. *eggs = yes*: 58 dari 60 hewan bertelur, sehingga pertanyaan itu hampir tidak memisahkan apa pun — kelompok ya-nya ($G_{\text{ya}} = 0.7782$) hampir seluruh node, dan kelompok tidak-nya terdiri atas satu reptil dan satu hewan tak bertulang belakang ($G_{\text{tidak}} = 0.5$), sehingga $G_{\text{split}} = \frac{58}{60} \cdot 0.7782 + \frac{2}{60} \cdot 0.5 = 0.769$, hampir tidak lebih baik daripada 0.785. Bulu burung menang, seperti yang diperlihatkan pohonnya.

Penyelesaian 12.3. Sebagai ganti `impurity`, hitunglah entropi tertimbang kedua kelompok, $\frac{a}{n}H_{\text{ya}} + \frac{b}{n}H_{\text{tidak}}$, dengan `math.log2` (Java: `Math.log(x) / Math.log(2)`). Pada data Zoo pertanyaan-pertanyaan yang terpilih tetap sama, karena kelompok-kelompok murni yang disukai Gini juga yang entropinya terendah. Agar Java dan Python sepakat, bandingkan skornya dengan `num.key` / `Num.key` (kunci yang sama berarti seri) dan pertahankan aturan bahwa atribut pertama yang memenangkan seri; pecahan-pecahan tepat dari Persamaan (12.3) tidak lagi tersedia, karena logaritma bukan pecahan.

Penyelesaian 12.4. (a) Lumba-lumba: milk = yes → mamalia (data Zoo memuat lumba-lumba). (b) Penguin: milk no, feathers yes → burung (ada di data). (c) Kalajengking: milk no, feathers no, fins no, backbone no, airborne no, predator yes → hewan tak bertulang belakang (ada di data; jawaban ini benar, sedangkan pohon leave-one-out, yang belum melihatnya, berkata serangga). (d) Siput: … predator no, legs 0 ≤ 3 → hewan tak bertulang belakang (ada di data, dan menjadi cornerstone `invertebrate#2`).

Penyelesaian 12.5. Golongannya adalah mayoritas, *insect*, dan $CF = \text{round}(100 \cdot 7/9) = \text{round}(77.8) = 78$. CF node itu menjadi CF rule menurut persamaan (10) tesis, seperti di setiap bab: hewan yang sampai ke sana dilaporkan sebagai `type = insect` dengan CF 78 (dikalikan kepastian fakta-faktanya, 100 bila pengguna yakin), yang memberi tahu pengguna bahwa kira-kira satu dari lima hewan dalam keadaan ini bukan serangga.

Penyelesaian 12.6. Pemeriksaan cornerstone hanya bertanya tentang kasus-kasus cornerstone dari node-node yang ada. Cornerstone `amphibian#1` adalah katak, yang tidak berekor, sehingga kondisi baru `tail = yes` salah untuknya dan pemeriksaan itu lolos. Newt sampai ke node yang sama tetapi bukan cornerstone; hanya pemeriksaan terhadap semua kasus yang sudah diketahui yang melihatnya. Dalam basis pengetahuan yang dibangun dengan tangan, cornerstone-lah kasus-kasus yang diketahui, sehingga pemeriksaannya lengkap untuk apa yang pernah dilihat sistem; tetapi bijaksana, seperti yang dilakukan Bab 11 dengan kasus emas, untuk menyimpan lebih banyak kasus yang sudah diperiksa daripada cornerstone, karena setiap kasus yang pernah dijawab benar oleh basis pengetahuan dan dikonfirmasi seseorang adalah uji yang gratis.

Penyelesaian 12.7. Keduanya sampai ke `mammal#1` melalui `milk = yes`, pertanyaan pertama; tidak ada hal lain yang ditanyakan. Untuk mengajarkan bahwa platipus itu istimewa (misalnya golongan *monotreme*), daftar perbedaan terhadap aardvark, cornerstone `mammal#1`, akan memperlihatkan di bagian A `eggs = no, toothed = yes, …` dan di bagian B `eggs = yes, toothed = no, aquatic = yes, …`; sang guru akan memilih `eggs = yes`, dan pemeriksaannya akan mengonfirmasi bahwa tidak ada mamalia lain dalam data yang bertelur — platipus satu-satunya.

Penyelesaian 12.8. Kekeliruan untuk catatan 13 dan 14 (tinta disebut *cold rinse* dan *enzyme soak*) dan 18 (lilin disebut *alcohol*) terjadi karena, ketika satu dari hanya dua catatan tinta atau satu-satunya catatan lilin disisihkan, pohon belum cukup melihat noda itu; catatan 5, darah segar, adalah cerita yang sama dengan satu dari hanya dua catatan darah segar. Ini tabel yang terlalu kecil, bukan aturan yang perlu dikoreksi. Kekeliruan untuk catatan 9, 11, dan 12 (teh) memperlihatkan pembedaan nyata yang tidak dapat diselesaikan dengan tiga contoh: putih dan kering mendapat pemutih oksigen, sisanya rendaman air sejuk. Seorang pakar akan menuliskannya sebagai aturan dan pengecualian — persis node teh dan pengecualiannya di kasus benih Bab 10.

Penyelesaian 12.9. Pertama, tanyakan kepada pakar: perlihatkan kedua pertanyaan dan kelompok-kelompoknya, lalu biarkan ia memilih; ini hanya membutuhkan sedikit waktunya dan memberi pohon alasan dari sang pakar. Kedua, pilih pertanyaan yang juga bekerja paling baik pada kasus di luar kelompok itu: pilih yang skor leave-one-out-nya lebih baik pada seluruh data, yang membutuhkan lebih banyak komputasi. (Cara ketiga yang murah: utamakan atribut yang paling mudah diamati — apakah seekor hewan bertelur lebih sulit dilihat di tepi sungai daripada kakinya — yang membutuhkan biaya untuk setiap kolom.)

Daftar Pustaka

[1] M. M. I. Subakti, “A Variable-Centered Intelligent Rule System,” M.Sc.Eng. thesis, Dept. of Computer Science and Information Engineering, Taipei, Taiwan, Jun. 2005.

[2] R. Forsyth, “Zoo,” UCI Machine Learning Repository, 1990, doi: 10.24432/C5R59V. Licensed under CC BY 4.0; copy used from F. Leisch and E. Dimitriadou, *mlbench: Machine Learning Benchmark Problems*, R package, CRAN.

[3] P. Compton and B. H. Kang, *Ripple-Down Rules: The Alternative to Machine Learning*. Boca Raton, FL, USA: CRC Press, 2021.

[4] B. R. Gaines and P. J. Compton, “Induction of ripple down rules,” in *Proc. 5th Australian Conf. Artificial Intelligence*, Hobart, Australia, 1992.

[5] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Belmont, CA, USA: Wadsworth, 1984.

[6] J. R. Quinlan, “Induction of decision trees,” *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[7] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA, USA: Morgan Kaufmann, 1993.

[8] F. Pedregosa *et al.*, “Scikit-learn: machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[9] scikit-learn developers, “scikit-learn 1.9.1,” Python Package Index, Sep. 2026. [Online]. Available: https://pypi.org/project/scikit-learn/

[10] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*. Jakarta, Indonesia, 2022.

Bab 13

Berbicara dengan Sistem Pakar Anda: Model Bahasa Lokal dengan Pengetahuan yang Diperiksa Manusia

Setelah menyelesaikan bab ini Anda akan mampu menjelaskan dengan kata-kata sederhana apa keunggulan sebuah *large language model* (model bahasa besar) dan mengapa model itu tidak boleh menjadi pihak yang memutuskan; menjalankan model terbuka yang kecil di komputer Anda sendiri dengan Ollama dan bercakap dengannya melalui API HTTP-nya dari Java dan dari Python; meminta model menjawab dalam bentuk JSON yang sudah ditentukan; membangun **antarmuka obrolan** untuk dokter Wi-Fi dari Bab 3, tempat model hanya *memahami* kata-kata pelanggan dan *merangkai* jawaban, sementara basis pengetahuan VCIRS yang diperiksa manusia mengambil setiap keputusan; menulis sebuah **penjaga** (*guard*) yang menolak setiap jawaban yang tidak didukung basis pengetahuan dan menggantinya dengan jawaban yang aman; serta menjalankan semuanya tanpa internet dengan sebuah server pengganti kecil, sehingga setiap percakapan dalam bab ini dapat diulang di komputer mana pun, dengan atau tanpa model.

13.1 Pesan-Pesan Pukul Sembilan Malam

Dokter Wi-Fi dari Bab 3 tinggal di komputer di konter Mas Anto. Ia mengajukan satu pertanyaan setiap kali, menjelaskan setiap saran, dan belajar dari setiap perbaikan yang belum diketahui basis pengetahuannya. Para pelanggan menyukainya. Namun pada malam hari, ketika konter sudah tutup, ponsel Mas Anto terus bergetar. Orang mengirim pesan sebagaimana orang biasa menulis pesan: *"mas, internet di rumah mati, lampu router kedip-kedip"*, *"hp saya ga bisa internet padahal wifi nyala"*, *"my phone has no internet, the wifi symbol is off and the little plane is on"*. Tidak ada yang menulis `wifi icon = off`. Tidak ada yang mau menjawab tujuh pertanyaan berturut-turut.

Mas Anto sudah melihat program-program obrolan yang ramai dibicarakan orang. Program-program itu menjawab apa saja, dalam bahasa apa saja, dengan kalimat yang indah. Keponakannya, Fajar, mengusulkan agar salah satunya saja yang menjawab pelanggan. Mas Anto mencobanya suatu malam dan cemas dengan apa yang ia lihat. Ia mengetik keluhan yang sama dua kali dan mendapat dua daftar saran yang berbeda. Salah satunya menyuruh pelanggan mengembalikan ponsel ke setelan pabrik, yang menghapus foto cucu-cucu seorang nenek dan sama sekali tidak menolong router yang lampunya merah. Tidak satu pun jawaban menyebut hal yang, menurut basis pengetahuannya sendiri, menjadi penyebab di lingkungannya: bila lampu router merah, masalahnya ada pada kabel penyedia layanan, dan satu-satunya saran yang berguna adalah menghubungi penyedia layanan.

Jadi Mas Anto menginginkan dua hal yang tampaknya saling tarik. Ia ingin dokternya memahami pesan yang ditulis dengan kata-kata biasa, seperti program-program obrolan itu. Dan ia ingin *sarannya* tetap secermat sekarang: hanya yang sudah ia periksa pada pelanggan sungguhan, dijelaskan, dan tidak pernah dikarang. Bab ini menunjukkan bahwa kedua keinginan itu sama sekali tidak bertentangan, asalkan setiap bagian mengerjakan tugas yang memang dikuasainya.

Model bahasa membaca dan menulis. Basis pengetahuan memutuskan. Dan sepotong kode kecil yang ketat — sang penjaga — berdiri di antara model dan pelanggan.

Kita akan menjalankan model di komputer konter itu sendiri, bukan di layanan di internet, karena tiga alasan yang penting bagi toko kecil: pesan pelanggan tidak pernah keluar dari toko, tidak ada tagihan per pesan, dan dokter tetap bekerja ketika internet — hal yang justru dikeluhkan pelanggan — sedang mati.

13.2 Gagasannya dengan Kata-Kata Sederhana

13.2.1 Apa itu model bahasa, dan apa yang bukan

Large language model (model bahasa besar) adalah program yang telah membaca teks dalam jumlah luar biasa banyak dan, dari situ, belajar melanjutkan teks. Berikan awal sebuah kalimat, dan ia menebak kata mana yang kemungkinan besar muncul berikutnya, lalu kata sesudahnya, dan seterusnya. Bila dilatih dengan teks yang cukup lalu diajari mengikuti perintah, model seperti ini dapat meringkas, menerjemahkan, menulis ulang, dan menjawab pertanyaan dalam kalimat yang lancar. Kelancaran itu nyata, dan untuk satu tugas justru itulah yang dibutuhkan Mas Anto: memahami apa maksud pelanggan dengan “lampu router kedip-kedip” dan menulis jawaban yang ramah.

Namun model itu tidak *mengetahui* apa pun seperti basis pengetahuan Mas Anto mengetahuinya. Ia tidak punya daftar kasus yang sudah diperiksa, tidak punya kasus *cornerstone* di balik setiap saran, dan tidak punya cara untuk berkata “masalah ini belum pernah saya lihat”. Ketika tidak tahu jawabannya, ia tetap menghasilkan kalimat yang paling tampak meyakinkan, dan kalimat yang tampak meyakinkan bisa saja keliru. Para peneliti menyebutnya **halusinasi** (*hallucination*): teks yang lancar dan yakin tetapi tidak didukung oleh fakta yang diberikan kepadanya maupun oleh kenyataan [1]. Sebuah survei di bidang ini menggambarkannya sebagai masalah umum semua pembangkit teks semacam itu, bukan kebetulan langka pada satu model [1]. Dua sifat lain membuat model tidak cocok menjadi pihak yang memutuskan di konter. Jawabannya dapat berubah dari satu kali jalan ke kali berikutnya, sehingga dua pelanggan dengan masalah yang sama bisa diberi saran berbeda. Dan ia tidak dapat menjelaskan *mengapa* dalam arti Bab 3: ia dapat menulis alasan yang masuk akal, tetapi alasan itu bukan jalan yang menghasilkan jawabannya.

Basis pengetahuan VCIRS berwatak sebaliknya. Ia hanya tahu apa yang diajarkan kepadanya, kasus demi kasus, oleh orang yang melihat kasus itu terselesaikan [2], [3]. Ia memberi jawaban yang sama untuk fakta yang sama setiap kali. Ia menjelaskan setiap jawaban dengan fakta-fakta yang membuat sebuah node menyala. Dan bila tidak ada node yang menyala, ia mengatakannya. Yang tidak dapat ia lakukan adalah membaca pesan yang ditulis dengan kata-kata biasa.

> **WAWASAN KUNCI:** Jangan meminta satu komponen mengerjakan kedua tugas. Biarkan model bahasa mengerjakan keunggulannya — membaca dan menulis kalimat — dan biarkan basis pengetahuan mengerjakan keunggulannya — memutuskan, dari pengetahuan yang sudah diperiksa, dengan penjelasan.

13.2.2 Dua tugas untuk model, satu untuk basis pengetahuan

Gambar 13.1 memperlihatkan keseluruhan rancangan. Gambar ini layak dibaca pelan-pelan, karena setiap subbab berikutnya membangun salah satu kotaknya.



Gambar 13.1 Antarmuka obrolan. Model bahasa (kuning) hanya mengubah kata menjadi fakta dan saran menjadi kalimat; semua keluarannya diperiksa. Basis pengetahuan VCIRS (biru tua) memutuskan dan menanyakan fakta yang masih dibutuhkannya. Penjaga hanya meloloskan jawaban yang memberikan saran basis pengetahuan dan tidak ada yang lain.

1. **Memahami.** Pesan pelanggan dikirim ke model bersama sebuah perintah: *ubah pesan ini menjadi fakta, hanya dengan variabel-variabel dan nilai-nilai ini*. Model membalas dengan sepotong kecil JSON, misalnya `{"facts": [{"variable": "airplane mode", "value": "on"}]}`.
2. **Menyaring.** Program tidak memercayai JSON itu. Sebuah fakta hanya disimpan bila variabelnya termasuk yang ditanyakan basis pengetahuan dan nilainya termasuk jawaban yang dikenal basis pengetahuan. Selebihnya ditampilkan lalu dibuang.
3. **Memutuskan.** Fakta-fakta diserahkan ke `kb.infer`, persis seperti di Bab 3. Bila sebuah node membutuhkan fakta yang tidak ada dalam pesan, mesin menanyakannya, dan pelanggan boleh menjawab dengan kata apa saja — model membantu memahami jawaban itu juga, melalui penyaring yang sama.
4. **Merangkai kalimat.** Ketika basis pengetahuan memberi saran, program mengirim saran itu beserta alasannya (kondisi-kondisi node yang menyala) kembali ke model dengan perintah kedua: *tulis dua atau tiga kalimat ramah yang memberikan persis saran ini*.
5. **Menjaga.** Sebelum sebuah jawaban sampai ke pelanggan, program memeriksanya: apakah jawaban itu memuat saran basis pengetahuan, kata demi kata? Apakah ia tidak menyebut perbaikan *lain* yang dikenal basis pengetahuan? Apakah ia tidak memuat kata-kata yang dilarang Mas Anto, seperti “factory”? Bila salah satu pemeriksaan gagal, jawaban itu dibuang dan program menulis sendiri jawaban sederhana dari sebuah templat.

Bila basis pengetahuan tidak punya saran, langkah 4 tidak pernah terjadi. Program mengatakan, dengan kalimat yang tetap, bahwa belum ada saran yang sudah diperiksa dan bahwa Mas Anto akan melihatnya sendiri. Model tidak pernah diminta mengisi celah itu, karena mengisi celah dengan kalimat yang tampak meyakinkan justru hal yang tidak boleh ia lakukan di sini.

Perhatikan di mana kata-kata model dapat salah dan apa yang terjadi kemudian. Fakta yang salah dari langkah 1 akan dibuang oleh penyaring atau, bila kebetulan merupakan fakta yang sah, ditampilkan kepada pelanggan sebagai `[understood] ...`, sehingga kesalahannya terlihat dan dapat dikoreksi pelanggan. Kalimat yang salah dari langkah 4 dihentikan oleh penjaga. Dalam keadaan apa pun, model tidak dapat menambahkan saran yang belum pernah diperiksa Mas Anto.

Pembagian kerja ini berkerabat dekat dengan sebuah teknik populer bernama *retrieval-augmented generation* (pembangkitan yang diperkaya penelusuran), tempat model diberi

dokumen-dokumen yang relevan untuk dibaca sebelum menjawab, agar jawabannya dapat bersandar pada dokumen itu [4]. Rancangan kita melangkah satu langkah lebih jauh ke arah yang sama: model tidak diberi dokumen untuk diringkas, melainkan diberi *keputusannya sendiri*, dan sebuah program memeriksa bahwa keputusan itu selamat setelah dirangkai ulang. Untuk konter tempat jawaban yang salah membuang satu pagi seorang pelanggan, langkah tambahan itu sepadan.

13.2.3 Model di komputer Anda sendiri: Ollama

Ollama adalah program gratis yang mengunduh model bahasa terbuka dan menjalankannya di komputer Anda sendiri, di Windows, macOS, dan Linux. Begitu berjalan, ia mendengarkan di alamat `http://localhost:11434` dan menjawab permintaan melalui HTTP, protokol yang sama dengan yang dipakai peramban web; program apa pun yang dapat mengirim permintaan HTTP dapat bercakap dengannya, termasuk program Java atau Python yang hanya memakai pustaka standarnya [5]. Per September 2026, rilis terbarunya adalah Ollama v0.34.3, yang terbit pada 19 September 2026 [6]. Ollama berubah dengan cepat, tetapi bagian yang dipakai bab ini — *endpoint* obrolan `POST /api/chat` — sudah lama tidak berubah bentuk, dan kode bab ini hanya bergantung padanya.

Model yang mana? Untuk komputer konter, modelnya harus kecil: beberapa gigabita untuk diunduh dan ukuran yang berjalan cukup lancar pada prosesor laptop biasa. **Tabel 13.1** mendaftar tiga keluarga model terbuka kecil yang tersedia di pustaka Ollama pada September 2026, beserta ukuran unduhan versi bawaan setiap tag.

Tabel 13.1 Model terbuka kecil di pustaka Ollama, September 2026, beserta ukuran unduhan versi bawaan setiap tag.

Tag	Unduhan	Catatan
`qwen3.5:0.8b`	1,0 GB	yang terkecil; untuk laptop lama [7]
`qwen3.5:2b`	2,7 GB	bawaan bab ini; dilatih untuk 201 bahasa dan dialek [7]
`qwen3.5:4b`	3,4 GB	pemahaman lebih baik, lebih lambat [7]
`llama3.2:1b`	1,3 GB	delapan bahasa resmi, bahasa Indonesia tidak termasuk [8]
`llama3.2:3b`	2,0 GB	keluarga yang sama, lebih besar [8]
`gemma4:e2b`	7,2 GB	model “edge” multimodal; unduhannya besar [9]

Kita memakai `qwen3.5:2b` sebagai bawaan karena dua alasan: ukurannya cocok untuk komputer konter, dan halaman modelnya menyebut dukungan untuk 201 bahasa dan dialek [7], hal yang penting ketika separuh pelanggan Mas Anto menulis dalam bahasa Indonesia dan sebagian dalam bahasa Jawa. Halaman Llama 3.2 menyebut delapan bahasa resmi, dan bahasa Indonesia tidak termasuk di dalamnya [8]; model itu mungkin masih memahami bahasa Indonesia yang sederhana, tetapi Anda akan bergantung pada untung-untungan. Semua ini tidak permanen: model kecil baru muncul setiap beberapa bulan. Karena tugas model hanyalah membaca dan menulis, dan karena setiap jawabannya diperiksa, Anda dapat mengganti model dengan satu opsi (`--model`) tanpa mengubah satu pun aturan dalam basis pengetahuan.

13.2.4 Meminta JSON, bukan prosa

Pada langkah 1 kita membutuhkan fakta, bukan paragraf yang ramah. Model yang diminta dengan sopan untuk menjawab dalam JSON biasanya menurut, tetapi “biasanya” tidak cukup untuk program yang harus membaca jawabannya. Ollama memecahkan hal ini dengan **keluaran terstruktur** (*structured outputs*): permintaan boleh membawa kolom `format` yang berisi JSON

Schema, yaitu gambaran bentuk persis yang harus dimiliki jawaban, dan Ollama membatasi model sehingga keluarannya sesuai dengan bentuk itu [5], [10]. Skema kita berbunyi: jawabannya adalah objek dengan satu kolom, `facts`, berupa daftar objek yang masing-masing punya `variable` dan `value`, keduanya string; dan `variable` harus salah satu variabel basis pengetahuan.

Skema menetapkan *bentuk*; ia tidak membuat *isinya* benar. Model tetap dapat mengisi `blinking` sebagai nilai `router lights`, nilai yang belum pernah dipakai basis pengetahuan Mas Anto, atau menyatakan fakta yang tidak pernah dikatakan pelanggan. Itulah sebabnya langkah 2, penyaring, adalah potongan kode tersendiri yang tidak memercayai apa pun.

13.2.5 Penjaga

> **DEFINISI.** *Penjaga* (*guard*) adalah pemeriksaan, ditulis sebagai kode biasa, yang harus dilewati setiap keluaran model bahasa sebelum dipakai. Di sini penjaga hanya menerima jawaban bila (1) jawaban itu memuat saran basis pengetahuan kata demi kata, (2) tidak memuat perbaikan lain yang dikenal basis pengetahuan, (3) tidak memuat frasa yang dilarang sang pakar, dan (4) pendek. Bila tidak, jawaban itu diganti dengan jawaban yang disusun dari templat.

Setiap aturan penjaga punya alasan. Aturan pertama memastikan pelanggan benar-benar mendengar sarannya; jawaban yang berbunyi "switch off the plane mode" mungkin bermakna sama, tetapi program tidak dapat membuktikannya, jadi ia tidak mengambil risiko. Aturan kedua menghentikan kegagalan paling umum dari model yang ingin membantu: menambahkan "Anda juga bisa me-restart router" pada saran yang sebenarnya tentang mode pesawat. Aturan ketiga adalah daftar milik Mas Anto sendiri tentang hal-hal yang tidak boleh disarankan di konter — pengembalian ke setelan pabrik, membongkar ponsel, apa pun tentang baterai — yang disimpan dalam berkas teks kecil yang dapat ia sunting. Aturan keempat mencegah model menenggelamkan sarannya dalam satu halaman teks.

Penjaga ini sengaja dibuat ketat, dan keketatan ada harganya: sebagian jawaban yang baik akan ditolak dan diganti templat yang lebih polos. Itu pertukaran yang tepat untuk sebuah konter. Kalimat polos yang benar lebih baik daripada kalimat indah yang salah, dan templat selalu tersedia, sehingga pelanggan selalu mendapat jawaban.

13.3 Langkah 1: Bercakap dengan Ollama

Setiap permintaan ke *endpoint* obrolan Ollama adalah objek JSON kecil yang dikirim dengan `POST` ke `/api/chat`. Permintaan itu menyebut nama model, memberikan daftar pesan — masing-masing dengan `role` (`system` untuk perintah, `user` untuk ucapan orang) dan `content`-nya — dan boleh menambahkan opsi. Kita selalu mengirim empat kolom lagi: `"stream": false`, agar jawaban tiba utuh dan bukan kata demi kata; `"think": false`, agar model yang dapat "berpikir keras" sebelum menjawab melewatinya dan langsung menjawab; `"options": {"temperature": 0, "seed": 42}`, yang meminta kata-kata yang paling mungkin dan benih acak yang tetap, sehingga pertanyaan yang sama mendapat jawaban yang sama sesering yang dimungkinkan model; dan, bila kita menginginkan JSON, skema `format` [5]. Jawabannya adalah objek JSON lain yang kolom `message.content`-nya memuat teks model, bersama informasi seperti `eval_count`, banyaknya token (potongan kata) yang dihasilkannya.

Kode 13.1 adalah seluruh klien dalam Java.

Kode 13.1 `OllamaClient.java`: klien minimal untuk API obrolan Ollama.

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.charset.StandardCharsets;
import java.time.Duration;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

import vcirs.Json;
import vcirs.VcirsException;

/**
 * Klien yang sangat kecil untuk API obrolan Ollama: POST /api/chat. Kelas
 * yang sama bercakap dengan Ollama sungguhan di komputer ini
 * (http://localhost:11434) atau dengan server pengganti bab ini
 * (StubOllama), yang menjawab dengan cara yang sama.
 */
public class OllamaClient {
    public static final String DEFAULT_URL = "http://localhost:11434";
    public static final String DEFAULT_MODEL = "qwen3.5:2b";

    /** Model bahasa tidak dapat dijangkau atau tidak memberi jawaban. */
    public static class LlmException extends RuntimeException {
        LlmException(String message) {
            super(message);
        }
    }

    public final String url;
    public final String model;
    public Map<String, Object> last = new LinkedHashMap<>();
    // Model lokal tidak perlu proxy, meski komputernya memakai proxy.
    private final HttpClient http = HttpClient.newBuilder()
            .proxy(HttpClient.Builder.NO_PROXY)
            .connectTimeout(Duration.ofSeconds(10)).build();

    public OllamaClient(String url, String model) {
        this.url = url.replaceAll("/+$", "");
        this.model = model;
    }
```

Kode 13.1 (Lanjutan) *OllamaClient.java: klien minimal untuk API obrolan Ollama.*

```
    /**
     * Kirim satu pesan system dan satu pesan user; kembalikan teks jawaban.
     * Dengan skema (bukan null), model harus menjawab JSON berbentuk itu.
     */
    @SuppressWarnings("unchecked")
    public String chat(String system, String user, Map<String, Object> schema) {
        Map<String, Object> body = new LinkedHashMap<>();
        body.put("model", model);
        body.put("stream", false);                    // satu jawaban utuh
        body.put("think", false);                     // tanpa penalaran
        body.put("options", Map.of("temperature", 0, "seed", 42));
        body.put("messages", List.of(
                Map.of("role", "system", "content", system),
                Map.of("role", "user", "content", user)));
        if (schema != null) {
            body.put("format", schema);               // keluaran terstruktur
        }
        HttpRequest request = HttpRequest.newBuilder(URI.create(url + "/api/chat"))
                .timeout(Duration.ofSeconds(120))
                .header("Content-Type", "application/json")
                .POST(HttpRequest.BodyPublishers.ofString(Json.write(body),
                        StandardCharsets.UTF_8))
                .build();
        Object answer;
        try {
            HttpResponse<String> response = http.send(request,
                    HttpResponse.BodyHandlers.ofString(StandardCharsets.UTF_8));
            try {
                answer = Json.parse(response.body());
            } catch (VcirsException e) {
                answer = Map.of("error", "HTTP " + response.statusCode());
            }
        } catch (IOException | IllegalArgumentException e) {
            throw new LlmException("cannot reach the language model at " + url);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            throw new LlmException("interrupted");
        }
        if (answer instanceof Map && ((Map<String, Object>) answer)
                .containsKey("error")) {                  // mis. model tak ada
            throw new LlmException("the language model said: "
                    + ((Map<String, Object>) answer).get("error"));
        }
        if (!(answer instanceof Map) || !((Map<String, Object>) answer)
                .containsKey("message")) {
            throw new LlmException("the language model gave no message");
```

Kode 13.1 (Lanjutan) *OllamaClient.java: klien minimal untuk API obrolan Ollama.*

```
        }
        last = (Map<String, Object>) answer;
        Object message = last.get("message");
        Object content = message instanceof Map
                ? ((Map<String, Object>) message).get("content") : null;
        return content == null ? "" : content.toString();
    }
}
```

Badan permintaan disusun sebagai *map* dan ditulis sebagai JSON dengan `vcirs.Json`, pembaca dan pencetak JSON kecil milik vcirs-core, sehingga bab ini tidak memerlukan pustaka tambahan. `HttpClient` adalah klien HTTP pustaka standar Java. Kita membangunnya dengan `NO_PROXY` dengan sengaja: banyak kantor dan kampus mengalirkan lalu lintas web melalui *proxy*, dan *proxy* tidak dapat menjangkau `localhost` di komputer Anda; permintaan ke model di komputer ini

harus langsung menuju ke sana. `http.send` mengirim permintaan dan menunggu seluruh jawaban. Jawaban dibaca dengan `Json.parse`; bila sama sekali bukan JSON, program menganggap ia menerima `{"error": "HTTP 500"}` (atau apa pun kode statusnya), sehingga semua kegagalan menempuh jalan yang sama.

Perhatikan tiga cara metode ini dapat gagal, karena program konter harus selamat dari ketiganya. Bila tidak ada program yang mendengarkan di alamat itu — Ollama belum terpasang atau belum dijalankan — `send` melempar `IOException`, dan kita mengubahnya menjadi `LlmException` yang menyatakan model tidak dapat dijangkau. Bila Ollama menjawab tetapi dengan galat, misalnya karena nama model salah ketik atau model belum pernah diunduh, jawabannya memuat kolom `error`, dan kita meneruskan teksnya. Bila jawabannya tidak memuat `message`, kita mengatakannya. Karena itu setiap pemanggil `chat` cukup menangani tepat satu jenis eksepsi.

Kode 13.2 adalah klien yang sama dalam Python.

Kode 13.2 `ollama_client.py`: klien yang sama dalam Python, hanya pustaka standar.

```
"""Klien yang sangat kecil untuk API obrolan Ollama: POST /api/chat.

Hanya pustaka standar Python yang dipakai. Kelas yang sama bercakap dengan
Ollama sungguhan di komputer ini (http://localhost:11434) atau dengan server
pengganti bab ini (stub_ollama.py), yang menjawab dalam format yang sama.
"""

import json
import urllib.error
import urllib.request

DEFAULT_URL = "http://localhost:11434"
DEFAULT_MODEL = "qwen3.5:2b"


class LlmError(Exception):
    """Model bahasa tidak dapat dijangkau atau tidak memberi jawaban."""


class OllamaClient:
    def __init__(self, url=DEFAULT_URL, model=DEFAULT_MODEL, timeout=120):
        self.url = url.rstrip("/")
        self.model = model
        self.timeout = timeout
        self.last = {}                  # jawaban utuh terakhir, sekadar ingin tahu
        # Model lokal tidak perlu proxy, meski komputernya memakai proxy.
        self.opener = urllib.request.build_opener(
            urllib.request.ProxyHandler({}))

    def chat(self, system, user, schema=None):
        """Kirim satu pesan system dan satu pesan user; kembalikan teks
        jawaban. Dengan skema, model harus menjawab JSON berbentuk itu."""
        body = {
            "model": self.model,
            "stream": False,                # satu jawaban utuh
            "think": False,                 # tanpa penalaran panjang
            "options": {"temperature": 0, "seed": 42},
            "messages": [{"role": "system", "content": system},
                         {"role": "user", "content": user}],
        }
        if schema is not None:
            body["format"] = schema         # keluaran terstruktur
        request = urllib.request.Request(
            self.url + "/api/chat", data=json.dumps(body).encode("utf-8"),
            headers={"Content-Type": "application/json"})
        try:
```

Kode 13.2 (Lanjutan) *ollama_client.py: klien yang sama dalam Python, hanya pustaka standar.*

```
            with self.opener.open(request, timeout=self.timeout) as response:
                status, text = response.status, response.read()
        except urllib.error.HTTPError as e:      # mis. 404: model tak ada
            status, text = e.code, e.read()
        except OSError as e:
            raise LlmError("cannot reach the language model at "
                           + self.url) from e
        try:
            answer = json.loads(text.decode("utf-8"))
        except ValueError:
            answer = {"error": "HTTP " + str(status)}
        if isinstance(answer, dict) and "error" in answer:
            raise LlmError("the language model said: " + str(answer["error"]))
        if not isinstance(answer, dict) or "message" not in answer:
            raise LlmError("the language model gave no message")
        self.last = answer
        message = answer["message"]
        return str(message.get("content", "")) if isinstance(message, dict) \
            else ""
```

`urllib.request` adalah klien HTTP bawaan Python. `ProxyHandler({})` berperan seperti `NO_PROXY`. Ketika server menjawab dengan status galat seperti 404, `urllib` melempar `HTTPError`, yang membawa badan jawaban; kita membacanya seperti jawaban biasa, sehingga kolom `error` sampai ke tempat yang sama seperti di Java.

Sekarang percakapan pertama. **Kode 13.3** menanyakan satu pertanyaan kepada model dan mencetak jawabannya.

Kode 13.3 `AskOllama.java`: langkah 1, satu pertanyaan kepada model.

```java
import java.util.ArrayList;
import java.util.List;

/**
 * Langkah 1: ajukan satu pertanyaan kepada model lokal, cetak jawabannya.
 * java AskOllama [--url URL] [--model NAME] [question]
 */
public class AskOllama {
    public static void main(String[] args) {
        String url = OllamaClient.DEFAULT_URL;
        String model = OllamaClient.DEFAULT_MODEL;
        List<String> words = new ArrayList<>();
        for (int i = 0; i < args.length; i++) {
            if (args[i].equals("--url") && i + 1 < args.length) {
                url = args[++i];
            } else if (args[i].equals("--model") && i + 1 < args.length) {
                model = args[++i];
            } else {
                words.add(args[i]);
            }
        }
        String question = words.isEmpty()
                ? "My Wi-Fi symbol is on but nothing loads. What should I do?"
                : String.join(" ", words);
        OllamaClient llm = new OllamaClient(url, model);
        System.out.println("Question: " + question);
        String reply;
        try {
            reply = llm.chat("You are a helpful assistant. Answer briefly.",
                    question, null);
        } catch (OllamaClient.LlmException e) {
            System.out.println("Sorry: " + e.getMessage());
            return;
        }
        System.out.println("Model " + model + " replied:");
        System.out.println(reply);
        Object count = llm.last.get("eval_count");
        System.out.println("(" + (count == null ? "?" : count)
                + " tokens generated)");
    }
}
```

Kode 13.4 adalah versi Python-nya.

Kode 13.4 `ask_ollama.py`: langkah 1 dalam Python.

```
"""Langkah 1: ajukan satu pertanyaan kepada model lokal dan cetak jawabannya.

    python ask_ollama.py [--url URL] [--model NAME] [question]
"""

import sys

from ollama_client import DEFAULT_MODEL, DEFAULT_URL, LlmError, OllamaClient


def main(args):
    url, model, words = DEFAULT_URL, DEFAULT_MODEL, []
    i = 0
    while i < len(args):
        if args[i] == "--url" and i + 1 < len(args):
            url, i = args[i + 1], i + 2
        elif args[i] == "--model" and i + 1 < len(args):
            model, i = args[i + 1], i + 2
        else:
            words.append(args[i])
            i += 1
    question = " ".join(words) or ("My Wi-Fi symbol is on but nothing "
                                   "loads. What should I do?")
    llm = OllamaClient(url, model)
    print("Question: " + question)
    try:
        reply = llm.chat("You are a helpful assistant. Answer briefly.",
                         question)
    except LlmError as e:
        print("Sorry: " + str(e))
        return
    print("Model " + model + " replied:")
    print(reply)
    print("(" + str(llm.last.get("eval_count", "?")) + " tokens generated)")


if __name__ == "__main__":
    main(sys.argv[1:])
```

Dengan model sungguhan Anda cukup menjalankan `java -cp out AskOllama` atau `python ask_ollama.py`. Agar buku ini dapat diulang di komputer mana pun, transkrip dalam bab ini dihasilkan dengan server pengganti dari subbab berikutnya, yang mendengarkan di port 4131; perintahnya `java -cp out AskOllama --url http://localhost:4131`:

```
Question: My Wi-Fi symbol is on but nothing loads. What should I do?
Model qwen3.5:2b replied:
Here are some things to try: restart your phone, forget the network and join it again,
reset the network settings, or update the router's firmware.
(25 tokens generated)
```

Server pengganti memberi jenis jawaban yang biasanya diberikan model obrolan umum untuk pertanyaan ini: daftar langkah yang terdengar masuk akal. Tidak ada yang aneh di dalamnya, namun itu bukan yang akan dikatakan Mas Anto. Jawaban itu tidak menanyakan apakah ponsel lain bisa internetan atau apa warna lampu router; tidak tahu bahwa di lingkungan ini lampu merah berarti kabel penyedia layanan; dan salah satu sarannya, mengatur ulang setelan jaringan, menghapus setiap kata sandi Wi-Fi yang tersimpan di ponsel. Model sungguhan akan merangkainya secara berbeda setiap kali dan mungkin menambah atau mengurangi butirnya. Perilaku inilah yang menjadi dasar rancangan sepanjang sisa bab: teks model itu lancar, masuk akal, dan tidak diperiksa.

13.4 Langkah 2: Pengganti yang Berjalan tanpa Internet

Buku yang contoh-contohnya memerlukan unduhan 2,7 gigabita dan komputer yang cepat akan kehilangan banyak pembaca pada titik ini. Lagi pula, bahkan dengan model, transkripnya tidak akan pernah dapat dicetak persis, karena model merangkai jawabannya secara berbeda dari satu kali jalan ke kali berikutnya. Jadi kita menulis sebuah **pengganti** (dalam pengujian sering disebut *stub*): server mungil yang menjawab `POST /api/chat` persis dalam format Ollama, tetapi tanpa model di dalamnya. Ia tidak dapat memahami bahasa. Ia hanya mencari frasa dari sebuah buku frasa kecil, `stub-phrasebook.json`, yang beberapa entrinya seperti ini:

```
{"variable": "airplane mode", "value": "on", "words": ["plane is on", "airplane mode is
on", "flight mode is on"]},
{"variable": "other devices online", "value": "no", "words": ["none of them", "no other
phone", "the others too"]},
{"variable": "router lights", "value": "blinking", "words": ["blinking"]},
```

Entri terakhir sengaja ada di sana. Model sungguhan yang membaca “the router lights are blinking” kemungkinan besar akan melaporkan `router lights = blinking`, nilai yang tidak dikenal basis pengetahuan Mas Anto (yang dikenalnya normal, merah, dan mati). Server pengganti menirunya, agar kita dapat melihat penyaring membuangnya.

Server pengganti punya mode kedua yang sengaja dibuat buruk, `--sloppy` (ceroboh). Dalam mode itu ia berperilaku seperti model ceroboh yang ingin terlalu membantu: setelah saran yang benar, ia menambahkan “You can also restart the router, or reset the phone to factory settings.” Kita memakainya untuk melihat penjaga bekerja.

Kode 13.5 adalah server pengganti dalam Python.

Kode 13.5 `stub_ollama.py`: pengganti Ollama yang menjawab dari buku frasa.

```
"""Pengganti Ollama, agar bab ini berjalan tanpa internet dan selalu sama.
Ia menjawab POST /api/chat dalam format Ollama, tetapi tidak memahami
bahasa: ia hanya mencari frasa-frasa dari stub-phrasebook.json.

    python stub_ollama.py [--port 4131] [--sloppy]

--sloppy membuatnya seperti model ceroboh: ia menambahkan saran sendiri.
"""

import json
import os
import sys
from http.server import BaseHTTPRequestHandler, ThreadingHTTPServer

HERE = os.path.dirname(os.path.abspath(__file__))
BOOK = os.path.join(HERE, "..", "data", "stub-phrasebook.json")
FREE_REPLY = ("Here are some things to try: restart your phone, forget the "
              "network and join it again, reset the network settings, or "
              "update the router's firmware.")
SLOPPY_EXTRA = (" You can also restart the router, or reset the phone to "
                "factory settings.")


def load_phrases(path=BOOK):
    with open(path, encoding="utf-8") as f:
        return json.load(f)["phrases"]


def parse_reply(phrases, text, allowed):
    """Fakta-fakta yang frasanya muncul dalam teks, satu per variabel."""
    low = text.lower()
    facts, seen = [], set()
    for p in phrases:
        if p["variable"] in allowed and p["variable"] not in seen \
                and any(w in low for w in p["words"]):
            facts.append({"variable": p["variable"], "value": p["value"]})
            seen.add(p["variable"])
    return json.dumps({"facts": facts})
```

Kode 13.5 (Lanjutan) *stub_ollama.py: pengganti Ollama yang menjawab dari buku frasa.*

```
def phrase_reply(text, sloppy):
    """Ubah 'ADVICE: ...' dan 'REASONS: a = b; c = d' menjadi jawaban ramah."""
    advice, reasons = "", []
    for line in text.splitlines():
        if line.startswith("ADVICE: "):
            advice = line[len("ADVICE: "):]
        elif line.startswith("REASONS: "):
            reasons = [r.replace(" = ", " is ")
                       for r in line[len("REASONS: "):].split("; ")]
    if not advice:
        return FREE_REPLY
    because = reasons[-1] if len(reasons) == 1 else \
        ", ".join(reasons[:-1]) + " and " + reasons[-1]
    reply = ("Thank you for waiting. Please " + advice + ". I suggest it "
             "because " + because + ". If it does not help, come back and "
             "we will look at it together.")
    return reply + (SLOPPY_EXTRA if sloppy else "")


def answer(request, phrases, sloppy):
    """Jawaban /api/chat lengkap untuk satu permintaan (sebuah dict)."""
    user = ""
    for m in request.get("messages", []):
        if m.get("role") == "user":
            user = m.get("content", "")
    schema = request.get("format")
    if isinstance(schema, dict):
        item = schema["properties"]["facts"]["items"]["properties"]
        content = parse_reply(phrases, user, item["variable"]["enum"])
    else:
        content = phrase_reply(user, sloppy)
    return {"model": request.get("model", "stub"),
            "created_at": "2026-09-25T00:00:00Z",
            "message": {"role": "assistant", "content": content},
            "done": True, "done_reason": "stop",
            "total_duration": 0, "eval_count": len(content.split())}


def make_handler(phrases, sloppy):
    class Handler(BaseHTTPRequestHandler):
        protocol_version = "HTTP/1.1"        # koneksi tetap terbuka
```

Kode 13.5 (Lanjutan) *stub_ollama.py: pengganti Ollama yang menjawab dari buku frasa.*

```
        def do_POST(self):
            if self.path != "/api/chat":
                self.send_error(404)
                return
            size = int(self.headers.get("Content-Length", "0"))
            request = json.loads(self.rfile.read(size).decode("utf-8"))
            body = json.dumps(answer(request, phrases, sloppy)).encode("utf-8")
            self.send_response(200)
            self.send_header("Content-Type", "application/json")
            self.send_header("Content-Length", str(len(body)))
            self.end_headers()
            self.wfile.write(body)

        def log_message(self, fmt, *args):   # terminal tetap tenang
            pass

    return Handler


def main(args):
    port = int(args[args.index("--port") + 1]) if "--port" in args else 4131
    sloppy = "--sloppy" in args
    server = ThreadingHTTPServer(("127.0.0.1", port),
                                 make_handler(load_phrases(), sloppy))
    print("Stand-in model on http://localhost:" + str(port)
          + (" (sloppy)" if sloppy else "") + "; Ctrl+C stops it.", flush=True)
    try:
        server.serve_forever()
    except KeyboardInterrupt:
        pass


if __name__ == "__main__":
    main(sys.argv[1:])
```

`answer` melihat permintaan persis seperti Ollama melihatnya. Bila permintaan membawa skema `format`, itu permintaan "memahami": server pengganti mengambil variabel-variabel yang diizinkan dari `enum` dalam skema dan mengembalikan, sebagai teks JSON, satu fakta untuk setiap variabel yang frasanya muncul dalam pesan pengguna. Bila tidak, itu permintaan "merangkai": `phrase_reply` membaca baris `ADVICE:` dan `REASONS:` yang dikirim program kita dan menyusun kalimat ramah darinya — atau, bila tidak ada baris saran, mengembalikan daftar umum yang kita lihat di Subbab 13.3. Bagian HTTP-nya memakai `ThreadingHTTPServer` dari pustaka standar; `protocol_version = "HTTP/1.1"` menjaga koneksi tetap terbuka di antara permintaan, seperti yang dilakukan Ollama, dan seperti yang diharapkan klien HTTP Java.

Kode 13.6 adalah server pengganti yang sama dalam Java, dengan server web milik JDK, `com.sun.net.httpserver`.

Kode 13.6 `StubOllama.java`: server pengganti yang sama dalam Java.

```
import java.io.IOException;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Set;

import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpServer;

import vcirs.Json;

/**
 * Pengganti Ollama, agar bab ini berjalan tanpa internet dan selalu sama.
 * Ia menjawab POST /api/chat dalam format Ollama, tetapi tidak memahami
 * bahasa: ia hanya mencari frasa-frasa dari stub-phrasebook.json.
 * java StubOllama [--port 4131] [--sloppy]
 * --sloppy membuatnya seperti model ceroboh: ia menambahkan saran sendiri.
 */
public class StubOllama {
    static final String FREE_REPLY = "Here are some things to try: restart "
            + "your phone, forget the network and join it again, reset the "
            + "network settings, or update the router's firmware.";
    static final String SLOPPY_EXTRA = " You can also restart the router, or "
            + "reset the phone to factory settings.";

    @SuppressWarnings("unchecked")
    static List<Map<String, Object>> loadPhrases(String path) throws IOException {
        Map<String, Object> book = (Map<String, Object>) Json.parse(
                Files.readString(Path.of(path), StandardCharsets.UTF_8));
        return (List<Map<String, Object>>) book.get("phrases");
    }
```

Kode 13.6 (Lanjutan) *StubOllama.java: server pengganti yang sama dalam Java.*

```
    /** Fakta-fakta yang frasanya muncul dalam teks, satu per variabel. */
    @SuppressWarnings("unchecked")
    static String parseReply(List<Map<String, Object>> phrases, String text,
                             List<Object> allowed) {
        String low = text.toLowerCase(Locale.ROOT);
        List<Object> facts = new ArrayList<>();
        Set<Object> seen = new HashSet<>();
        for (Map<String, Object> p : phrases) {
            Object var = p.get("variable");
            if (!allowed.contains(var) || seen.contains(var)) {
                continue;
            }
            for (Object w : (List<Object>) p.get("words")) {
                if (low.contains((String) w)) {
                    facts.add(Map.of("variable", var, "value", p.get("value")));
                    seen.add(var);
                    break;
                }
            }
        }
        return Json.write(Map.of("facts", facts));
    }

    /** Ubah "ADVICE: ..." dan "REASONS: a = b; c = d" menjadi jawaban ramah. */
    static String phraseReply(String text, boolean sloppy) {
        String advice = "";
        List<String> reasons = new ArrayList<>();
        for (String line : text.split("\n")) {
            if (line.startsWith("ADVICE: ")) {
                advice = line.substring("ADVICE: ".length());
            } else if (line.startsWith("REASONS: ")) {
                for (String r : line.substring("REASONS: ".length())
                        .split("; ")) {
                    reasons.add(r.replace(" = ", " is "));
                }
            }
        }
        if (advice.isEmpty()) {
            return FREE_REPLY;
        }
        int n = reasons.size();
        String because = n == 1 ? reasons.get(0)
                : String.join(", ", reasons.subList(0, n - 1)) + " and "
                + reasons.get(n - 1);
        String reply = "Thank you for waiting. Please " + advice + ". I suggest "
                + "it because " + because + ". If it does not help, come back "
```

Kode 13.6 (Lanjutan) *StubOllama.java: server pengganti yang sama dalam Java.*

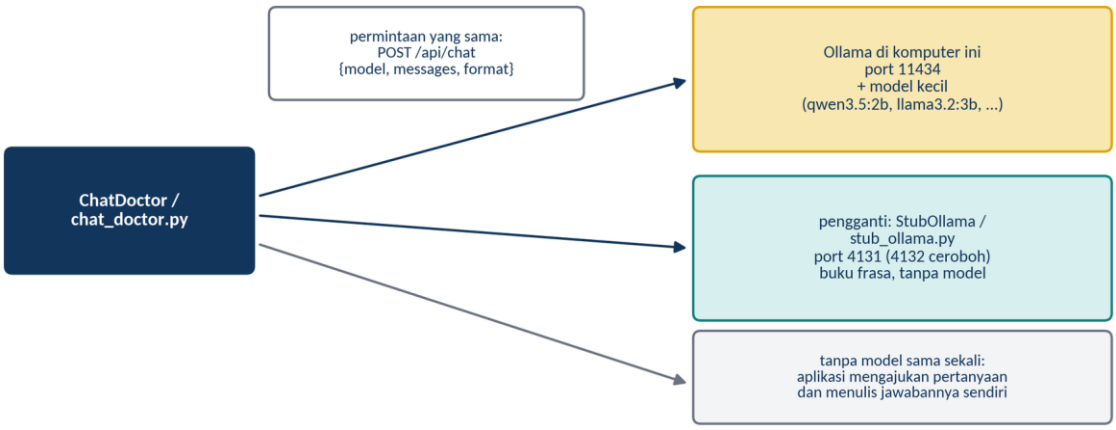
```
                + "and we will look at it together.";
        return reply + (sloppy ? SLOPPY_EXTRA : "");
    }

    /** Jawaban /api/chat lengkap untuk satu permintaan. */
    @SuppressWarnings("unchecked")
    static Map<String, Object> answer(Map<String, Object> request,
                                      List<Map<String, Object>> phrases,
                                      boolean sloppy) {
        String user = "";
        for (Object o : (List<Object>) request.get("messages")) {
            Map<String, Object> m = (Map<String, Object>) o;
            if ("user".equals(m.get("role"))) {
                user = (String) m.get("content");
            }
        }
        String content;
        if (request.get("format") instanceof Map) {
            Map<String, Object> schema =
                    (Map<String, Object>) request.get("format");
            Map<String, Object> facts = (Map<String, Object>) ((Map<String, Object>)
                    schema.get("properties")).get("facts");
            Map<String, Object> item = (Map<String, Object>) ((Map<String, Object>)
                    facts.get("items")).get("properties");
            List<Object> allowed = (List<Object>) ((Map<String, Object>)
                    item.get("variable")).get("enum");
            content = parseReply(phrases, user, allowed);
        } else {
            content = phraseReply(user, sloppy);
        }
        Map<String, Object> out = new LinkedHashMap<>();
        out.put("model", request.getOrDefault("model", "stub"));
        out.put("created_at", "2026-09-25T00:00:00Z");
        out.put("message", Map.of("role", "assistant", "content", content));
        out.put("done", true);
        out.put("done_reason", "stop");
        out.put("total_duration", 0);
        out.put("eval_count", content.trim().split("\\s+").length);
        return out;
    }
```

Kode 13.6 (Lanjutan) *StubOllama.java: server pengganti yang sama dalam Java.*

```
    @SuppressWarnings("unchecked")
    public static void main(String[] args) throws IOException {
        int port = 4131;
        boolean sloppy = false;
        for (int i = 0; i < args.length; i++) {
            if (args[i].equals("--port") && i + 1 < args.length) {
                port = Integer.parseInt(args[++i]);
            } else if (args[i].equals("--sloppy")) {
                sloppy = true;
            }
        }
        List<Map<String, Object>> phrases =
                loadPhrases("../data/stub-phrasebook.json");
        final boolean careless = sloppy;
        HttpServer server = HttpServer.create(
                new InetSocketAddress("127.0.0.1", port), 0);
        server.createContext("/api/chat", (HttpExchange ex) -> {
            byte[] body;
            int status = 200;
            if (!ex.getRequestMethod().equals("POST")) {
                status = 405;
                body = "{\"error\": \"use POST\"}".getBytes(StandardCharsets.UTF_8);
            } else {
                String text = new String(ex.getRequestBody().readAllBytes(),
                        StandardCharsets.UTF_8);
                Map<String, Object> request =
                        (Map<String, Object>) Json.parse(text);
                body = Json.write(answer(request, phrases, careless))
                        .getBytes(StandardCharsets.UTF_8);
            }
            ex.getResponseHeaders().set("Content-Type", "application/json");
            ex.sendResponseHeaders(status, body.length);
            try (OutputStream out = ex.getResponseBody()) {
                out.write(body);
            }
        });
        server.start();
        System.out.println("Stand-in model on http://localhost:" + port
                + (sloppy ? " (sloppy)" : "") + "; Ctrl+C stops it.");
    }
}
```

Kedua server pengganti menjawab dengan sama, begitu pula kedua klien. **Gambar 13.2** memperlihatkan tiga keadaan yang mungkin dialami dokter obrolan: bercakap dengan Ollama sungguhan, bercakap dengan server pengganti, atau tidak bercakap dengan siapa pun. Program tidak tahu dan tidak peduli dengan yang mana dari dua keadaan pertama ia bercakap — permintaan dan jawabannya berbentuk sama — dan ia punya rencana untuk keadaan ketiga.



Gambar 13.2 Klien yang sama bercakap dengan Ollama sungguhan beserta model kecil, dengan server pengganti tanpa internet, atau, bila keduanya tidak menjawab, dengan siapa pun tidak: saat itu ia mengajukan pertanyaannya sendiri dan menulis jawabannya sendiri.

> **PERINGATAN:** Server pengganti adalah alat uji, bukan model bahasa murah. Ia hanya mengenali frasa dalam buku frasanya, dalam bentuk yang tertulis di sana. Jangan menaruhnya di depan pelanggan sungguhan; taruhlah model sungguhan di sana, dan simpan server pengganti untuk pengujian dan peragaan tanpa model.

13.5 Langkah 3: Dokter Obrolan

Sekarang aplikasinya sendiri. Ia memakai basis pengetahuan dari Bab 3, dibangun dari enam kasus benih yang sama (`wifi-seed-cases.json`, disalin ke folder `data` bab ini), dan tujuh pertanyaan yang sama, kini disimpan dalam berkas data `wifi-questions.json`, agar program dapat menyerahkan daftar variabel beserta jawaban-jawaban biasanya kepada model. Program ini tidak pernah menyimpan basis pengetahuan: dalam bab ini dokter hanya berkonsultasi. Pengajaran tetap berada di tempat yang ditentukan Bab 3, di konter, oleh orang yang melihat kasusnya terselesaikan.

Kode 13.7 adalah dokter obrolan dalam Python.

Kode 13.7 `chat_doctor.py`: dokter obrolan. Model memahami dan merangkai kalimat; basis pengetahuan memutuskan; penjaga memeriksa.

```
"""Dokter Wi-Fi yang bisa diajak bicara. Model bahasa lokal hanya mengubah
kata-kata pelanggan menjadi fakta dan saran basis pengetahuan menjadi jawaban
yang ramah; basis pengetahuan VCIRS memutuskan, dan penjaga memeriksa.

    python chat_doctor.py [--echo] [--url URL] [--model NAME]
"""

import json
import os
import pathlib
import sys

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import KnowledgeBase, casefile                    # noqa: E402
from ollama_client import (DEFAULT_MODEL, DEFAULT_URL, LlmError,  # noqa: E402
                           OllamaClient)

DATA = os.path.join(os.path.dirname(os.path.abspath(__file__)), "..", "data")
PHRASE_RULES = (
    "You are the friendly helper at a phone counter. Write two or three "
    "short sentences to the customer. Give exactly the ADVICE below, in the "
    "same words, and explain it with the REASONS. Do not add any other "
    "advice and do not mention anything that is not in the REASONS.")


def build_kb():
    """Basis pengetahuan yang diperiksa Mas Anto, dengan pertanyaan & jawaban."""
    name, description, steps = casefile.read_cases(
        os.path.join(DATA, "wifi-seed-cases.json"))
    kb = KnowledgeBase(name, description)
    with open(os.path.join(DATA, "wifi-questions.json"),
              encoding="utf-8") as f:
        for v in json.load(f)["variables"]:
            kb.declare(v["variable"], v["question"], v["choices"])
    return casefile.build(kb, steps)


def question_variables(kb):
    """Variabel yang dapat ditanyakan kepada pelanggan: yang dipakai dalam
    kondisi node. Jawabannya ("fix") tidak pernah termasuk."""
    return [v for v in kb.variables
            if any(c.variable == v for n in kb.nodes.values()
                   for c in n.conditions)]
```

Kode 13.7 (Lanjutan) *chat_doctor.py: dokter obrolan. Model memahami dan merangkai kalimat; basis pengetahuan memutuskan; penjaga memeriksa.*

```
def read_never_say():
    with open(os.path.join(DATA, "never-say.txt"), encoding="utf-8") as f:
        return [line.strip().lower() for line in f if line.strip()]


def parse_prompt(kb, variables):
    lines = ["You turn a customer's message about a phone or Wi-Fi problem "
             "into facts.", "Use only these variables and these values:"]
    for var in variables:
        lines.append("- " + var + ": " + " / ".join(kb.known_values(var)))
    lines.append("Write a fact only if the message says it clearly. Never "
                 "guess and never give advice.")
    return "\n".join(lines)


def parse_schema(variables):
    """Bentuk JSON yang wajib dipakai model: {"facts": [...]}."""
    fact = {"type": "object",
            "properties": {"variable": {"type": "string",
                                        "enum": list(variables)},
                           "value": {"type": "string"}},
            "required": ["variable", "value"]}
    return {"type": "object",
            "properties": {"facts": {"type": "array", "items": fact}},
            "required": ["facts"]}


def understand(llm, kb, text, variables):
    """Minta fakta kepada model, lalu simpan hanya fakta yang dikenal basis
    pengetahuan. Hasil (kept, ignored): kept daftar (variabel, nilai)."""
    content = llm.chat(parse_prompt(kb, variables), text,
                       parse_schema(variables))
    try:
        data = json.loads(content)
    except ValueError:
        data = {}
    items = data.get("facts", []) if isinstance(data, dict) else []
    kept, ignored = [], []
    for item in items if isinstance(items, list) else []:
        if not isinstance(item, dict):
            continue
        var = str(item.get("variable", "")).strip()
        value = str(item.get("value", "")).strip().lower()
        said = var + " = " + value
        if var not in variables:
            ignored.append(said + " (not a question I ask)")
```

Kode 13.7 (Lanjutan) *chat_doctor.py: dokter obrolan. Model memahami dan merangkai kalimat; basis pengetahuan memutuskan; penjaga memeriksa.*

```
        elif value not in kb.known_values(var):
            ignored.append(said + " (not one of the known answers)")
        elif any(v == var for v, _ in kept):
            ignored.append(said + " (said twice)")
        else:
            kept.append((var, value))
    return kept, ignored


def check_reply(reply, advice, other_fixes, never_say):
    """Penjaga: masalah-masalah sebuah jawaban; daftar kosong berarti aman."""
    low = reply.lower()
    problems = []
    if advice.lower() not in low:
        problems.append("it does not give the advice '" + advice + "'")
    rest = low.replace(advice.lower(), " ")
    for fix in other_fixes:
        if fix.lower() in rest:
            problems.append("it adds another fix: " + fix)
    for phrase in never_say:
        if phrase in low:
            problems.append("it says something never allowed: " + phrase)
    if len(reply) > 400:
        problems.append("it is too long")
    return problems


def template_reply(advice, reasons):
    """Jawaban polos yang disusun tanpa model: selalu aman."""
    said = [r.replace(" = ", " is ") for r in reasons]
    because = said[-1] if len(said) == 1 else \
        ", ".join(said[:-1]) + " and " + said[-1]
    return "Please " + advice + ". I suggest it because " + because + "."


class Chat:
    def __init__(self, kb, llm, echo):
        self.kb = kb
        self.llm = llm                  # None bila model tak terjangkau
        self.echo = echo
```

Kode 13.7 (Lanjutan) *chat_doctor.py: dokter obrolan. Model memahami dan merangkai kalimat; basis pengetahuan memutuskan; penjaga memeriksa.*

```
    def read(self, prompt):
        print(prompt, end="")
        text = sys.stdin.readline().strip()
        if self.echo:
            print(text)
        return text

    def show(self, kept, ignored):
        if kept:
            print("[understood] " + ", ".join(v + " = " + x for v, x in kept))
        for item in ignored:
            print("[ignored] " + item)

    def ask(self, question):
        """Penanya mesin: satu pertanyaan, satu jawaban dengan kata apa saja."""
        var = question.variable
        print("Doctor > " + (self.kb.variables[var].question or var))
        text = self.read("You > ")
        if not text:
            return None                          # "saya tidak tahu"
        if text.lower() in self.kb.known_values(var):
            print("[understood] " + var + " = " + text.lower())
            return text.lower()
        if self.llm is None:
            print("[not understood] please use one of: "
                  + " / ".join(self.kb.known_values(var)))
            return None
        kept, ignored = understand(self.llm, self.kb, text, [var])
        self.show(kept, ignored)
        if not kept:
            print("[not understood] I will treat it as unknown")
            return None
        return kept[0][1]

    def run(self):
        facts = {}
        message = self.read("You > ")
        if self.llm is not None:
            try:
                kept, ignored = understand(self.llm, self.kb, message,
                                           question_variables(self.kb))
                self.show(kept, ignored)
                facts = dict(kept)
            except LlmError:
                self.llm = None
                print("[no language model] I will ask my questions instead.")
```

Kode 13.7 (Lanjutan) *chat_doctor.py: dokter obrolan. Model memahami dan merangkai kalimat; basis pengetahuan memutuskan; penjaga memeriksa.*

```
        result = self.kb.infer(facts, self.ask)
        if not result.conclusions():
            print("Doctor > I have no checked advice for this problem yet. "
                  "Mas Anto will look at it himself.")
            return
        advice = result.conclusions()[0].value
        reasons = []
        for node in result.paths[0].nodes:
            for c in node.conditions:
                if str(c) not in reasons:
                    reasons.append(str(c))
        self.reply(advice, reasons)

    def reply(self, advice, reasons):
        text = None
        if self.llm is not None:
            try:
                text = self.llm.chat(PHRASE_RULES, "ADVICE: " + advice
                                     + "\nREASONS: " + "; ".join(reasons))
            except LlmError:
                print("[no language model] I will write the reply myself.")
        if text is not None:
            others = [f for f in self.kb.known_values("fix") if f != advice]
            problems = check_reply(text, advice, others, read_never_say())
            if not problems:
                print("Doctor > " + text)
                print("[checked] the reply gives the knowledge base's advice "
                      "and nothing else")
                return
            for p in problems:
                print("[rejected] " + p)
        print("Doctor > " + template_reply(advice, reasons))
        print("[template] the reply was written without the language model")


def main(args):
    url, model = DEFAULT_URL, DEFAULT_MODEL
    if "--url" in args:
        url = args[args.index("--url") + 1]
    if "--model" in args:
        model = args[args.index("--model") + 1]
    kb = build_kb()
    print("Wi-Fi doctor chat: " + str(len(kb.nodes)) + " fixes checked by "
          "Mas Anto; model " + model + " at " + url)
    Chat(kb, OllamaClient(url, model), "--echo" in args).run()
```

Kode 13.7 (Lanjutan) *chat_doctor.py: dokter obrolan. Model memahami dan merangkai kalimat; basis pengetahuan memutuskan; penjaga memeriksa.*

```
if __name__ == "__main__":
    main(sys.argv[1:])
```

Bacalah menurut urutan **Gambar 13.1**.

Pengetahuan. `build_kb` membaca kasus benih dan pertanyaan yang dideklarasikan, persis seperti Bab 3 membangun basis pengetahuannya. `read_never_say` membaca daftar frasa terlarang milik Mas Anto:

```
factory
open the phone
battery
diagnos
guarantee
```

`diagnos` menangkap "diagnose" maupun "diagnosis": dokter boleh menyarankan, tetapi tidak pernah mengaku mendiagnosis. `guarantee` mencegah model menjanjikan bahwa sebuah perbaikan pasti berhasil.

Variabel yang mana? `question_variables` mendaftar variabel-variabel yang muncul dalam kondisi suatu node: hal-hal yang dapat ditanyakan kepada pelanggan. Variabel jawaban, `fix`, *tidak* termasuk di dalamnya, dan itu aturan keselamatan, bukan perincian. Bila model diizinkan melaporkan `fix = call the internet provider` sebagai fakta, pelanggan yang mengetik "perbaikannya adalah memberi saya ponsel baru" — atau model yang sekadar bingung — dapat memasukkan sebuah konklusi langsung ke memori kerja, dan mesin akan memperlakukannya sebagai fakta yang diberikan. Pesan yang berusaha mengarahkan model dengan cara ini disebut *prompt injection*, dan pertahanan paling sederhana adalah tidak pernah memberi model jalan untuk menyebutkan apa jawabannya.

Memahami. `parse_prompt` menyusun perintah untuk model: apa yang sedang ia kerjakan, variabel-variabel yang boleh dipakainya beserta nilai-nilai yang boleh dipakai untuk masing-masing (dari `kb.known_values`, yang mencakup pilihan yang dideklarasikan dan setiap nilai yang pernah dilihat basis pengetahuan), dan aturan "jangan menebak dan jangan memberi saran". `parse_schema` menyusun JSON Schema dari Subbab 13.2.4, dengan variabel-variabel yang diizinkan dalam sebuah `enum`. `understand` mengirim keduanya beserta teks pelanggan, lalu membaca jawabannya dengan waspada: bila bukan JSON, dianggap "tidak ada yang dipahami"; bila sebuah butir bukan objek, dilewati; variabel yang tidak ada dalam daftar diabaikan dengan alasan *not a question I ask*; nilai yang tidak dikenal basis pengetahuan diabaikan dengan alasan *not one of the known answers*; nilai kedua untuk variabel yang sama diabaikan sebagai *said twice*. Yang lolos dikembalikan sebagai daftar pasangan. Untuk pesan pertama dalam sebuah percakapan, perintah yang diterima model persis teks berikut, yang dicetak oleh `parse_prompt(kb, question_variables(kb))`:

```
You turn a customer's message about a phone or Wi-Fi problem into facts.
Use only these variables and these values:
- connection: wifi / mobile data
- wifi icon: on / off
- airplane mode: on / off
- other devices online: yes / no
- router lights: normal / red / off
- mobile data: on / off
- quota left: yes / no
Write a fact only if the message says it clearly. Never guess and never give advice.
```

Perintah ini sengaja pendek. Model kecil mengikuti perintah yang pendek dan konkret lebih baik daripada perintah yang panjang dan cerdik, dan setiap barisnya berasal dari basis pengetahuan itu sendiri: ketika Mas Anto mengajarkan node dengan variabel baru, perintah untuk percakapan berikutnya langsung menyebutnya tanpa ada yang menyunting teks perintah.

Memutuskan, dan bertanya. `Chat.run` membaca pesan pertama, memanggil `understand` dengan semua variabel pertanyaan, menampilkan apa yang dipahami dan apa yang diabaikan, lalu menyerahkan fakta-fakta itu ke `kb.infer(facts, self.ask)`. Sejak saat ini, mesinlah yang memegang kendali, persis seperti di Bab 3: ia menelusuri node-node, dan setiap kali sebuah node membutuhkan fakta yang belum dimilikinya, ia memanggil `ask`. `ask` milik kita mencetak pertanyaan yang dideklarasikan, membaca jawabannya, dan memahaminya dalam tiga langkah:

jawaban kosong berarti "saya tidak tahu"; jawaban yang persis salah satu nilai yang dikenal (seperti `red`) diterima apa adanya, tanpa merepotkan model; selain itu diserahkan ke `understand`, kali ini hanya dengan satu variabel yang sedang ditanyakan, sehingga model tidak dapat melantur ke topik lain.

Merangkai kalimat. Bila mesin menemukan saran, `run` mengumpulkan alasannya: kondisi setiap node pada jalur yang menyala, tanpa pengulangan. `reply` mengirim saran dan alasannya ke model bersama `PHRASE_RULES`, perintah untuk menulis dua atau tiga kalimat ramah yang memberikan persis saran ini. Permintaan ini tidak membawa skema: di sini kita menginginkan kalimat.

Menjaga. `check_reply` mengembalikan daftar masalah sebuah jawaban; daftar kosong berarti jawaban itu boleh dikirim. Ada satu perincian yang mudah terlewat. Sebelum mencari perbaikan *lain*, fungsi ini menghapus saran itu sendiri dari teks (`rest = low.replace(...)`). Tanpa langkah itu, saran seperti "turn on Wi-Fi" akan ditandai setiap kali basis pengetahuan juga mengenal perbaikan yang lebih panjang yang memuat kata-kata itu. Bila daftar masalahnya tidak kosong, setiap masalah dicetak sebagai `[rejected] ...` dan `template_reply` menulis jawaban polos dari saran dan alasannya.

Tanpa model. Dua kali program menangkap `LlmError`. Bila permintaan pertama gagal, program mengisi `self.llm` dengan `None` dan mengatakannya; sejak itu `ask` hanya menerima jawaban yang persis salah satu nilai yang dikenal, dan mendaftar nilai-nilai itu bila tidak memahami jawaban. Bila permintaan merangkai kalimat gagal, jawaban templat yang dipakai. Dokter tanpa model hanyalah dokter Bab 3 dengan layar yang berbeda.

Kode 13.8 adalah versi Java-nya. Strukturnya sama, metode demi metode; satu-satunya perbedaan nyata adalah Java memerlukan kelas kecil, `Understood`, di tempat Python mengembalikan dua daftar.

Kode 13.8 `ChatDoctor.java`: dokter obrolan dalam Java.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;

import vcirs.CaseFile;
import vcirs.Condition;
import vcirs.Fact;
import vcirs.Facts;
import vcirs.InferenceResult;
import vcirs.Json;
import vcirs.KnowledgeBase;
import vcirs.Node;
import vcirs.Question;
import vcirs.VcirsException;

/**
 * Dokter Wi-Fi yang bisa diajak bicara. Model bahasa lokal hanya mengubah
 * kata-kata pelanggan menjadi fakta dan saran basis pengetahuan menjadi
 * jawaban yang ramah; basis pengetahuan VCIRS memutuskan, dan penjaga
 * memeriksa setiap jawaban. java ChatDoctor [--echo] [--url URL] [--model NAME]
 */
public class ChatDoctor {
    static final String DATA = "../data/";
    static final String PHRASE_RULES = "You are the friendly helper at a phone "
            + "counter. Write two or three short sentences to the customer. "
            + "Give exactly the ADVICE below, in the same words, and explain it "
            + "with the REASONS. Do not add any other advice and do not mention "
            + "anything that is not in the REASONS.";
```

Kode 13.8 (Lanjutan) *ChatDoctor.java: dokter obrolan dalam Java.*

```
    /** Basis pengetahuan yang diperiksa Mas Anto, dengan pertanyaan & jawaban. */
    @SuppressWarnings("unchecked")
    static KnowledgeBase buildKb() throws IOException {
        CaseFile seed = CaseFile.read(DATA + "wifi-seed-cases.json");
        KnowledgeBase kb = new KnowledgeBase(seed.name, seed.description);
        Map<String, Object> q = (Map<String, Object>) Json.parse(Files.readString(
                Path.of(DATA + "wifi-questions.json"), StandardCharsets.UTF_8));
        for (Object o : (List<Object>) q.get("variables")) {
            Map<String, Object> v = (Map<String, Object>) o;
            List<String> choices = new ArrayList<>();
            for (Object c : (List<Object>) v.get("choices")) {
                choices.add((String) c);
            }
            kb.declare((String) v.get("variable"), (String) v.get("question"),
                    choices);
        }
        return seed.build(kb);
    }

    /**
     * Variabel yang dapat ditanyakan kepada pelanggan: yang dipakai dalam
     * kondisi node. Jawabannya ("fix") tidak pernah termasuk.
     */
    static List<String> questionVariables(KnowledgeBase kb) {
        List<String> out = new ArrayList<>();
        for (String v : kb.variables.keySet()) {
            for (Node n : kb.nodes.values()) {
                if (!out.contains(v) && n.conditions.stream()
                        .anyMatch(c -> c.variable.equals(v))) {
                    out.add(v);
                }
            }
        }
        return out;
    }
```

Kode 13.8 (Lanjutan) *ChatDoctor.java: dokter obrolan dalam Java.*

```
    static List<String> readNeverSay() throws IOException {
        List<String> out = new ArrayList<>();
        for (String line : Files.readAllLines(Path.of(DATA + "never-say.txt"),
                StandardCharsets.UTF_8)) {
            if (!line.strip().isEmpty()) {
                out.add(line.strip().toLowerCase(Locale.ROOT));
            }
        }
        return out;
    }

    static String parsePrompt(KnowledgeBase kb, List<String> variables) {
        List<String> lines = new ArrayList<>();
        lines.add("You turn a customer's message about a phone or Wi-Fi problem "
                + "into facts.");
        lines.add("Use only these variables and these values:");
        for (String var : variables) {
            lines.add("- " + var + ": " + String.join(" / ", kb.knownValues(var)));
        }
        lines.add("Write a fact only if the message says it clearly. Never "
                + "guess and never give advice.");
        return String.join("\n", lines);
    }

    /** Bentuk JSON yang wajib dipakai model: {"facts": [...]}. */
    static Map<String, Object> parseSchema(List<String> variables) {
        Map<String, Object> props = new LinkedHashMap<>();
        props.put("variable", Map.of("type", "string",
                "enum", new ArrayList<Object>(variables)));
        props.put("value", Map.of("type", "string"));
        Map<String, Object> fact = new LinkedHashMap<>();
        fact.put("type", "object");
        fact.put("properties", props);
        fact.put("required", List.of("variable", "value"));
        Map<String, Object> schema = new LinkedHashMap<>();
        schema.put("type", "object");
        schema.put("properties", Map.of("facts",
                Map.of("type", "array", "items", fact)));
        schema.put("required", List.of("facts"));
        return schema;
    }
```

Kode 13.8 (Lanjutan) *ChatDoctor.java: dokter obrolan dalam Java.*

```
    /** Yang dipahami model: fakta yang disimpan dan yang diabaikan (sebabnya). */
    static class Understood {
        final Map<String, String> kept = new LinkedHashMap<>();
        final List<String> ignored = new ArrayList<>();
    }

    /**
     * Minta fakta kepada model, lalu simpan hanya fakta yang dikenal basis
     * pengetahuan.
     */
    @SuppressWarnings("unchecked")
    static Understood understand(OllamaClient llm, KnowledgeBase kb, String text,
                                 List<String> variables) {
        String content = llm.chat(parsePrompt(kb, variables), text,
                parseSchema(variables));
        Object data;
        try {
            data = Json.parse(content);
        } catch (VcirsException e) {
            data = Map.of();
        }
        Object items = data instanceof Map
                ? ((Map<String, Object>) data).get("facts") : null;
        Understood u = new Understood();
        if (!(items instanceof List)) {
            return u;
        }
        for (Object o : (List<Object>) items) {
            if (!(o instanceof Map)) {
                continue;
            }
            Map<String, Object> item = (Map<String, Object>) o;
            String var = String.valueOf(item.getOrDefault("variable", "")).strip();
            String value = String.valueOf(item.getOrDefault("value", ""))
                    .strip().toLowerCase(Locale.ROOT);
            String said = var + " = " + value;
            if (!variables.contains(var)) {
                u.ignored.add(said + " (not a question I ask)");
            } else if (!kb.knownValues(var).contains(value)) {
                u.ignored.add(said + " (not one of the known answers)");
            } else if (u.kept.containsKey(var)) {
                u.ignored.add(said + " (said twice)");
            } else {
                u.kept.put(var, value);
            }
        }
```

Kode 13.8 (Lanjutan) *ChatDoctor.java: dokter obrolan dalam Java.*

```
        return u;
    }

    /** Penjaga: masalah-masalah sebuah jawaban; daftar kosong berarti aman. */
    static List<String> checkReply(String reply, String advice,
                                   List<String> otherFixes, List<String> neverSay) {
        String low = reply.toLowerCase(Locale.ROOT);
        String adv = advice.toLowerCase(Locale.ROOT);
        List<String> problems = new ArrayList<>();
        if (!low.contains(adv)) {
            problems.add("it does not give the advice '" + advice + "'");
        }
        String rest = low.replace(adv, " ");
        for (String fix : otherFixes) {
            if (rest.contains(fix.toLowerCase(Locale.ROOT))) {
                problems.add("it adds another fix: " + fix);
            }
        }
        for (String phrase : neverSay) {
            if (low.contains(phrase)) {
                problems.add("it says something never allowed: " + phrase);
            }
        }
        if (reply.length() > 400) {
            problems.add("it is too long");
        }
        return problems;
    }

    /** Jawaban polos yang disusun tanpa model: selalu aman. */
    static String templateReply(String advice, List<String> reasons) {
        List<String> said = new ArrayList<>();
        for (String r : reasons) {
            said.add(r.replace(" = ", " is "));
        }
        int n = said.size();
        String because = n == 1 ? said.get(0)
                : String.join(", ", said.subList(0, n - 1)) + " and "
                + said.get(n - 1);
        return "Please " + advice + ". I suggest it because " + because + ".";
    }
```

Kode 13.8 (Lanjutan) *ChatDoctor.java: dokter obrolan dalam Java.*

```
    final KnowledgeBase kb;
    OllamaClient llm;                    // null bila model tak terjangkau
    final boolean echo;
    final BufferedReader in = new BufferedReader(
            new InputStreamReader(System.in, StandardCharsets.UTF_8));

    ChatDoctor(KnowledgeBase kb, OllamaClient llm, boolean echo) {
        this.kb = kb;
        this.llm = llm;
        this.echo = echo;
    }

    String read(String prompt) {
        System.out.print(prompt);
        String line;
        try {
            line = in.readLine();
        } catch (IOException e) {
            line = null;
        }
        String text = line == null ? "" : line.strip();
        if (echo) {
            System.out.println(text);
        }
        return text;
    }

    void show(Understood u) {
        if (!u.kept.isEmpty()) {
            List<String> parts = new ArrayList<>();
            u.kept.forEach((v, x) -> parts.add(v + " = " + x));
            System.out.println("[understood] " + String.join(", ", parts));
        }
        for (String item : u.ignored) {
            System.out.println("[ignored] " + item);
        }
    }
```

Kode 13.8 (Lanjutan) *ChatDoctor.java: dokter obrolan dalam Java.*

```
    /** Penanya mesin: satu pertanyaan, satu jawaban dengan kata apa saja. */
    Fact ask(Question question) {
        String var = question.variable;
        String info = kb.variables.get(var).question;
        System.out.println("Doctor > " + (info.isEmpty() ? var : info));
        String text = read("You > ");
        if (text.isEmpty()) {
            return null;                              // "saya tidak tahu"
        }
        String low = text.toLowerCase(Locale.ROOT);
        if (kb.knownValues(var).contains(low)) {
            System.out.println("[understood] " + var + " = " + low);
            return new Fact(var, low);
        }
        if (llm == null) {
            System.out.println("[not understood] please use one of: "
                    + String.join(" / ", kb.knownValues(var)));
            return null;
        }
        Understood u = understand(llm, kb, text, List.of(var));
        show(u);
        if (u.kept.isEmpty()) {
            System.out.println("[not understood] I will treat it as unknown");
            return null;
        }
        return new Fact(var, u.kept.get(var));
    }

    void run() throws IOException {
        Map<String, String> facts = new LinkedHashMap<>();
        String message = read("You > ");
        if (llm != null) {
            try {
                Understood u = understand(llm, kb, message,
                        questionVariables(kb));
                show(u);
                facts.putAll(u.kept);
            } catch (OllamaClient.LlmException e) {
                llm = null;
                System.out.println("[no language model] I will ask my "
                        + "questions instead.");
            }
        }
        InferenceResult result = kb.infer(Facts.map(facts), this::ask);
        if (result.conclusions().isEmpty()) {
            System.out.println("Doctor > I have no checked advice for this "
```

Kode 13.8 (Lanjutan) *ChatDoctor.java: dokter obrolan dalam Java.*

```
                    + "problem yet. Mas Anto will look at it himself.");
            return;
        }
        String advice = result.conclusions().get(0).value;
        List<String> reasons = new ArrayList<>();
        for (Node node : result.paths.get(0).nodes) {
            for (Condition c : node.conditions) {
                if (!reasons.contains(c.toString())) {
                    reasons.add(c.toString());
                }
            }
        }
        reply(advice, reasons);
    }

    void reply(String advice, List<String> reasons) throws IOException {
        String text = null;
        if (llm != null) {
            try {
                text = llm.chat(PHRASE_RULES, "ADVICE: " + advice
                        + "\nREASONS: " + String.join("; ", reasons), null);
            } catch (OllamaClient.LlmException e) {
                System.out.println("[no language model] I will write the reply "
                        + "myself.");
            }
        }
        if (text != null) {
            List<String> others = new ArrayList<>();
            for (String f : kb.knownValues("fix")) {
                if (!f.equals(advice)) {
                    others.add(f);
                }
            }
            List<String> problems = checkReply(text, advice, others,
                    readNeverSay());
            if (problems.isEmpty()) {
                System.out.println("Doctor > " + text);
                System.out.println("[checked] the reply gives the knowledge "
                        + "base's advice and nothing else");
                return;
            }
            for (String p : problems) {
                System.out.println("[rejected] " + p);
            }
        }
        System.out.println("Doctor > " + templateReply(advice, reasons));
```

Kode 13.8 (Lanjutan) *ChatDoctor.java: dokter obrolan dalam Java.*

```
        System.out.println("[template] the reply was written without the "
                + "language model");
    }

    public static void main(String[] args) throws IOException {
        String url = OllamaClient.DEFAULT_URL;
        String model = OllamaClient.DEFAULT_MODEL;
        boolean echo = false;
        for (int i = 0; i < args.length; i++) {
            if (args[i].equals("--url") && i + 1 < args.length) {
                url = args[++i];
            } else if (args[i].equals("--model") && i + 1 < args.length) {
                model = args[++i];
            } else if (args[i].equals("--echo")) {
                echo = true;
            }
        }
        KnowledgeBase kb = buildKb();
        System.out.println("Wi-Fi doctor chat: " + kb.nodes.size() + " fixes "
                + "checked by Mas Anto; model " + model + " at " + url);
        new ChatDoctor(kb, new OllamaClient(url, model), echo).run();
    }
}
```

Untuk mengompilasi dan menjalankannya dari folder `Chapter13/java`, jalankan server pengganti di satu terminal dan dokternya di terminal lain:

```
javac -d out -sourcepath ../../vcirs-core/java/src *.java
java -cp out StubOllama
java -cp out ChatDoctor --echo --url http://localhost:4131 < ../data/chat-1.txt
```

(Python: `python stub_ollama.py`, lalu `python chat_doctor.py --echo --url http://localhost:4131 < ../data/chat-1.txt`. Di Windows PowerShell, yang tidak mengenal `<`, pakailah `Get-Content ..\data\chat-1.txt | java -cp out ChatDoctor --echo --url http://localhost:4131`.) Seperti di Bab 3, `--echo` mengulang setiap jawaban yang dibaca dari berkas, agar sesi yang diputar ulang terbaca seperti percakapan. Klien mana pun bekerja dengan server pengganti mana pun: dokter Java yang bercakap dengan server pengganti Python mencetak transkrip yang persis sama dengan dokter Python yang bercakap dengan server pengganti Java.

13.6 Lima Percakapan

Folder `data` memuat lima percakapan, `chat-1.txt` sampai `chat-5.txt`, satu baris untuk setiap hal yang diketik pelanggan.

Percakapan 1: semuanya dalam satu pesan.

```
Wi-Fi doctor chat: 6 fixes checked by Mas Anto; model qwen3.5:2b at http://localhost:4131
You > Hi, my phone has no internet. I am on the wifi, the wifi symbol is off and the little
plane is on.
[understood] connection = wifi, wifi icon = off, airplane mode = on
Doctor > Thank you for waiting. Please turn off airplane mode. I suggest it because
connection is wifi, wifi icon is off and airplane mode is on. If it does not help, come
back and we will look at it together.
[checked] the reply gives the knowledge base's advice and nothing else
```

Pesan itu memuat tiga fakta dalam kata-kata biasa — “on the wifi”, “the wifi symbol is off”, “the little plane is on” — dan ketiganya lolos dari penyaring. Dengan ketiganya, mesin tidak perlu bertanya apa-apa: node *turn on Wi-Fi* menyala, dan pengecualiannya, *turn off airplane mode*, ikut

menyala. Jawaban yang dirangkai memuat saran kata demi kata, tidak menambahkan perbaikan lain maupun kata terlarang, dan pendek, sehingga penjaga meloloskannya. Bandingkan dengan jawaban di Subbab 13.3 untuk keluhan sejenis: satu saran, yang sudah diperiksa, beserta alasannya.

Percakapan 2: pesan yang samar, dan jawaban dengan kata apa saja.

```
Wi-Fi doctor chat: 6 fixes checked by Mas Anto; model qwen3.5:2b at http://localhost:4131
You > Internet is dead at home and the router lights are blinking.
[ignored] router lights = blinking (not one of the known answers)
Doctor > Are you using Wi-Fi or mobile data?
You > the wifi
[understood] connection = wifi
Doctor > Is the Wi-Fi symbol at the top of the screen on or off?
You > on
[understood] wifi icon = on
Doctor > Do other phones get internet on the same Wi-Fi?
You > nope, none of them
[understood] other devices online = no
Doctor > Are the router's lights normal, red or off?
You > red
[understood] router lights = red
Doctor > Thank you for waiting. Please call the internet provider. I suggest it because
connection is wifi, wifi icon is on, other devices online is no and router lights is red.
If it does not help, come back and we will look at it together.
[checked] the reply gives the knowledge base's advice and nothing else
```

Inilah percakapan yang paling baik memperlihatkan rancangan ini. Pesan pertama hanya berkata sedikit: internet mati, lampu router berkedip. Satu-satunya fakta yang tampak berguna dari model, `router lights = blinking`, bukan jawaban yang dikenal basis pengetahuan, dan penyaring mengatakannya alih-alih diam-diam menebak bahwa berkedip berarti merah. Tanpa fakta, mesin mulai bertanya — dan pelanggan menjawab sebagaimana orang biasa menjawab. “the wifi” tidak persis `wifi`, jadi jawaban itu diserahkan ke model, dibatasi pada variabel `connection`, dan kembali sebagai `connection = wifi`. “on” persis nilai yang dikenal dan tidak memerlukan model. “nope, none of them” dipahami sebagai `other devices online = no`. Ketika mesin menanyakan lampu router, ia bertanya lagi alih-alih memakai “blinking”, dan “red” adalah nilai yang dikenal. Sarannya, *call the internet provider*, adalah saran yang diperiksa Mas Anto pada kasus benih keempatnya.

Perhatikan apa yang *tidak* terjadi. Model tidak pernah melihat node-node basis pengetahuan dan tidak pernah memilih pertanyaan berikutnya; mesinlah yang melakukannya, dalam urutan Bab 3. Model hanya diberi beberapa pertanyaan kecil yang sempit — fakta apa yang ada dalam kalimat ini — dan satu tugas menulis.

Percakapan 3: masalah yang tidak dikenal basis pengetahuan.

```
Wi-Fi doctor chat: 6 fixes checked by Mas Anto; model qwen3.5:2b at http://localhost:4131
You > I use mobile data, it is switched on and I still have quota, but pages will not load.
[understood] connection = mobile data, mobile data = on, quota left = yes
Doctor > I have no checked advice for this problem yet. Mas Anto will look at it himself.
```

Fakta-faktanya dipahami dengan sempurna: data seluler, menyala, kuota masih ada. Namun tidak ada node yang menyala untuk fakta-fakta itu, sehingga dokter mengatakannya, dengan kalimat yang tetap. Model tidak diminta menulis apa pun, karena tidak ada saran yang sudah diperiksa untuk dirangkainya. Inilah “belum ada saran” Bab 3 dalam bentuk obrolan, dan inilah baris terpenting bab ini. Program obrolan yang hanya bersandar pada model akan menghasilkan daftar yang meyakinkan; program ini dengan jujur memberi tahu pelanggan bahwa seseorang akan

melihatnya, dan Mas Anto kemudian dapat mengajarkan kasus itu di konter (dengan dokter Wi-Fi Bab 3, yang mempelajarinya sebagai node baru dengan opsi 5).

Percakapan 4: model yang ceroboh. Untuk percakapan ini server pengganti berjalan dalam mode cerobohnya, `StubOllama --port 4132 --sloppy`, dan dokter dijalankan dengan `--url http://localhost:4132`.

```
Wi-Fi doctor chat: 6 fixes checked by Mas Anto; model qwen3.5:2b at http://localhost:4132
You > My wifi symbol is off.
[understood] connection = wifi, wifi icon = off
Doctor > Is airplane mode (the little plane) on?
You > off
[understood] airplane mode = off
[rejected] it adds another fix: restart the router
[rejected] it says something never allowed: factory
Doctor > Please turn on Wi-Fi. I suggest it because connection is wifi and wifi icon is
off.
[template] the reply was written without the language model
```

Saran basis pengetahuan adalah *turn on Wi-Fi*, dan jawaban ceroboh itu memang memuatnya — tetapi berlanjut dengan "You can also restart the router, or reset the phone to factory settings." Penjaga menemukan dua masalah: *restart the router* adalah perbaikan lain dari basis pengetahuan, yang tidak berlaku bagi pelanggan ini, dan "factory" ada dalam daftar Mas Anto. Jawaban itu dibuang, dan pelanggan menerima jawaban templat yang polos. Tidak satu pun tambahan model sampai ke pelanggan.

Percakapan 5: tanpa model sama sekali. Dokter dijalankan dengan `--url http://localhost:4139`, tempat tidak ada yang mendengarkan — seolah-olah Ollama belum dinyalakan setelah listrik padam.

```
Wi-Fi doctor chat: 6 fixes checked by Mas Anto; model qwen3.5:2b at http://localhost:4139
You > wifi, the symbol is off, and the plane is on
[no language model] I will ask my questions instead.
Doctor > Are you using Wi-Fi or mobile data?
You > wifi
[understood] connection = wifi
Doctor > Is the Wi-Fi symbol at the top of the screen on or off?
You > off
[understood] wifi icon = off
Doctor > Is airplane mode (the little plane) on?
You > on
[understood] airplane mode = on
Doctor > Please turn off airplane mode. I suggest it because connection is wifi, wifi icon
is off and airplane mode is on.
[template] the reply was written without the language model
```

Permintaan pertama gagal, program mengatakan bahwa ia akan mengajukan pertanyaannya, dan dokter menjadi dokter tanya-jawab Bab 3. Pelanggan menjawab dengan kata-kata biasa, yang tidak memerlukan model, dan mendapat saran yang sama dengan percakapan 1, dalam kata-kata polos templat. Program konter yang berhenti bekerja ketika salah satu pembantunya tidak ada adalah program yang rapuh; program ini hanya menjadi kurang cerewet.

13.7 Menjalankannya dengan Model Sungguhan

Server pengganti membuat bab ini dapat diulang; model sungguhan membuat dokter berguna bagi pelanggan yang menulis dengan kata-kata mereka sendiri. Menyiapkannya memerlukan tiga langkah.

1. Pasang Ollama. Di **Windows** dan **macOS**, unduh penginstalnya dari situs web Ollama (ollama.com/download) dan jalankan [11]; setelah itu Ollama berjalan sendiri di latar belakang (di Windows ikonnya tampak di dekat jam, di macOS di bilah menu). Di **Linux**, situs web itu memberikan penginstal satu baris, `curl -fsSL https://ollama.com/install.sh | sh`, yang memasang Ollama sebagai layanan [11]. Periksa bahwa ia berjalan dengan membuka `http://localhost:11434` di peramban: ia menjawab "Ollama is running".

2. Unduh model, cukup sekali. Di terminal: `ollama pull qwen3.5:2b`. Perintah ini mengunduh sekitar 2,7 GB [7] dan memakan waktu selama yang dibutuhkan koneksi Anda; sesudahnya model ada di diska Anda dan bekerja tanpa internet. Cobalah secara manual dengan `ollama run qwen3.5:2b "Say hello in Indonesian"`. Pada laptop lama, pilih `qwen3.5:0.8b` (sekitar 1 GB). Model yang lebih besar memahami lebih baik tetapi menjawab lebih lambat, dan pada komputer tanpa kartu grafis perbedaannya besar; cobalah model Anda pada sepuluh pesan pelanggan sungguhan sebelum memutuskan.

3. Jalankan dokter tanpa `--url`. Dari `Chapter13/java`: `java -cp out ChatDoctor`, atau dari `Chapter13/python`: `python chat_doctor.py`. Keduanya bercakap dengan `http://localhost:11434` secara bawaan. Model lain dipilih dengan `--model`, misalnya `--model llama3.2:3b`.

Apa yang akan berbeda? Fakta yang dipahami model, pertanyaan yang diajukan mesin, dan saran yang diberikannya akan sama untuk keadaan yang sama, karena mesinlah yang memutuskan dan penyaring hanya meloloskan fakta yang dikenal. *Rangkaian kata* jawabannya akan berbeda dari satu kali jalan ke kali berikutnya dan dari satu model ke model lain, bahkan dengan *temperature* 0, dan sebagian jawaban akan ditolak penjaga, paling sering karena model mengubah kata-kata saran ("switch off flight mode" alih-alih "turn off airplane mode"). Bila itu terjadi, pelanggan menerima jawaban templat, yang selalu benar. Bila Anda melihat banyak penolakan semacam itu, obatnya bukan melonggarkan penjaga, melainkan memperbaiki perintah — misalnya dengan menambahkan "copy the ADVICE exactly, inside quotation marks" — lalu menguji lagi.

Model kecil di laptop biasa juga memerlukan waktu: beberapa detik per permintaan adalah hal biasa tanpa kartu grafis, dan satu percakapan dapat memerlukan beberapa permintaan (satu untuk setiap jawaban bebas, ditambah jawaban akhir). Itu dapat diterima untuk pesan yang datang pada malam hari; di konter yang ramai, dokter tanya-jawab Bab 3 tetap lebih cepat.

Apakah modelnya cukup baik? Ujilah seperti Mas Anto menguji pembantu baru: dengan pekerjaan sungguhan. Kumpulkan dua puluh pesan yang benar-benar dikirim pelanggan — dengan nama dan nomor dihapus — dan tuliskan, untuk setiap pesan, fakta-fakta yang akan dibaca oleh orang yang cermat. Lalu jalankan dokter pada setiap pesan dan bandingkan baris `[understood]` dengan daftar Anda. Hitung tiga hal: fakta yang ditemukan model dengan benar, fakta yang terlewat (mesin cukup menanyakannya, jadi satu fakta yang terlewat hanya berharga satu pertanyaan), dan fakta yang *salah* tetapi lolos dari penyaring (inilah yang berbahaya, karena mesin bertindak berdasarkan fakta itu). Model yang melewatkan beberapa fakta tidak apa-apa; model yang mengarang fakta yang tampak sah tidak dapat diterima, dan Anda sebaiknya mencoba model lain atau menambahkan contoh pada perintahnya. Ulangi pengujian setiap kali Anda mengganti model, perintah, atau pertanyaan, dan simpan kedua puluh pesan itu sebagai himpunan uji kecil di samping basis pengetahuan. Pekerjaan ini makan waktu setengah jam, dan menggantikan tebak-tebakan dengan bukti.

13.8 Membuatnya Lebih Baik

Bahasa Indonesia dan bahasa Jawa. Tidak ada bagian rancangan yang terikat pada bahasa Inggris. Pertanyaan dalam `wifi-questions.json` dapat ditulis dalam bahasa Indonesia, nilainya boleh tetap (nilai adalah pengenal, bukan kalimat), dan `PHRASE_RULES` dapat meminta model untuk "menulis dalam bahasa pelanggan". Karena penjaga membandingkan saran dengan jawaban kata demi kata, simpanlah nilai saran dalam bahasa jawaban, atau berikan kepada penjaga sebuah tabel kecil berisi terjemahan yang diterima untuk setiap saran; Soal 13.5 meminta Anda merancangnya.

Pemahaman yang lebih baik, tetap diperiksa. Model memahami lebih baik bila melihat contoh. Menambahkan tiga atau empat contoh pesan beserta fakta-fakta yang benar ke `parse_prompt` biasanya memperbaiki langkah pertama secara nyata, dan tidak mengurangi keselamatan sedikit pun, karena penyaring tetap memeriksa setiap fakta.

Mengonfirmasi apa yang dipahami. Dalam obrolan sungguhan, tunjukkan kepada pelanggan apa yang dipahami sebelum memutuskan: "Saya menangkap: Wi-Fi, simbolnya mati, mode pesawat menyala. Betul?" Fakta salah yang dikoreksi pelanggan jauh lebih murah daripada fakta salah yang ditindaklanjuti. Baris-baris `[understood]` dalam transkrip kita adalah bahan mentah untuk pesan itu.

Belajar dari pesan-pesan malam hari. Setiap percakapan yang berakhir dengan "belum ada saran yang sudah diperiksa" adalah kasus yang perlu dilihat Mas Anto. Tuliskan fakta-fakta yang dipahami — hanya faktanya, bukan pesan atau nomor pelanggan — ke sebuah berkas kecil, dan biarkan ia mengajarkannya di konter keesokan paginya dengan dokter Wi-Fi Bab 3. Bab 15 membangun catatan persis seperti itu untuk layanan yang sudah diterapkan, beserta aturan privasinya.

Penjaga yang bertumbuh. Ketika Mas Anto melihat jawaban yang tidak ia sukai, perbaikannya biasanya satu baris di `never-say.txt`. Rawatlah berkas itu secermat basis pengetahuan: berkas itu juga pengetahuan, tentang apa yang tidak boleh dikatakan.

Antarmuka lain. Antarmuka obrolan bercakap dengan mesin melalui antarmuka `asker` yang sama dengan konsol Bab 3 dan halaman web Bab 10 dan Bab 14. Aplikasi perpesanan, antarmuka suara, atau layar kios dapat disambungkan dengan cara yang sama, dan masing-masing mendapat saran yang sama-sama sudah diperiksa.

13.9 Memakainya Sungguh-Sungguh

Di konter ponsel. Jalankan Ollama dan dokter obrolan di komputer konter. Pesan pelanggan dapat ditempelkan ke dokter, atau — dengan sedikit kerja tambahan — diteruskan kepadanya melalui integrasi aplikasi perpesanan. Pertahankan dokter tanya-jawab Bab 3 untuk konter itu sendiri, tempat orang dapat menjawab pertanyaan secara langsung, dan pakailah dokter obrolan untuk pesan.

Di toko lain. Antarmuka obrolan sama sekali tidak bergantung pada Wi-Fi. Berikan basis pengetahuan lain — penasihat noda Bab 10, pencari kerusakan motor Bab 8 — beserta pertanyaan basis pengetahuan itu dan daftar frasa terlarang, maka ia menjadi antarmuka obrolan untuk basis pengetahuan tersebut. Perintah-perintahnya menyebut "a phone or Wi-Fi problem" dan "a phone counter"; ubahlah kedua frasa itu.

Bila taruhannya tinggi. Untuk kesehatan, uang, atau keselamatan, pembagian kerja yang sama bukan hanya nyaman, tetapi wajib. Penasihat cek kesehatan mandiri seperti yang ada di Bab 5 hanya boleh memberi saran yang sudah diperiksa tenaga kesehatan, lengkap dengan aturan rujukannya ("temui dokter atau bidan bila…") [12]; model bahasa boleh membuat pertanyaannya lebih ramah, tetapi tidak pernah boleh menambahkan saran, dan daftar kata terlarang penjaga sebaiknya memuat setiap kata yang terdengar seperti diagnosis atau resep.

Privasi. Pesan pelanggan dapat memuat nama, nomor telepon, atau alamat. Menjalankan model di komputer Anda sendiri berarti semua itu tidak pernah keluar dari toko. Jangan menyimpan pesannya; simpanlah, bila perlu, hanya fakta-fakta yang lolos dari penyaring. Bab 15 kembali ke hal ini dalam kerangka undang-undang pelindungan data pribadi Indonesia.

Kapan tidak perlu model. Bila pelanggan Anda senang memilih jawaban dari daftar — di layar kios, misalnya — Anda sama sekali tidak memerlukan model bahasa. Model hanya sepadan dengan unduhannya di tempat orang menulis secara bebas.

13.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Membiarkan model memutuskan. "Tanya saja modelnya harus bagaimana" menghasilkan saran yang lancar, berubah-ubah, dan tidak diperiksa. *Hindari:* model memahami dan merangkai kalimat; basis pengetahuan memutuskan.

Memercayai keluaran terstruktur. JSON Schema menjamin bentuk jawaban, bukan kebenarannya. *Hindari:* saring setiap fakta terhadap variabel dan nilai yang dikenal basis pengetahuan.

Membiarkan model mengisi celah. Ketika tidak ada node yang menyala, godaannya adalah meminta model memberi "jawaban umum". Justru di situlah ia berhalusinasi. *Hindari:* katakan terus terang bahwa belum ada saran yang sudah diperiksa, dan serahkan kasusnya kepada manusia.

Penjaga yang hanya memeriksa saran. Sebuah jawaban dapat memuat saran yang benar ditambah saran lain yang salah. *Hindari:* periksa juga setiap konklusi lain basis pengetahuan, dan frasa-frasa terlarang.

Menawarkan variabel jawaban kepada model. Bila `fix` ada dalam daftar variabel yang boleh dilaporkan model, sebuah pesan dapat menyelipkan konklusi sebagai fakta. *Hindari:* tawarkan hanya variabel-variabel yang muncul dalam kondisi, seperti `question_variables`.

Lupa bahwa saran bisa terkandung dalam perbaikan lain. Mencari perbaikan lain di seluruh jawaban akan menandai jawaban yang benar bila teks sebuah perbaikan memuat teks perbaikan lain. *Hindari:* hapus saran dari teks sebelum mencari, seperti yang dilakukan `check_reply`.

Tidak punya rencana bila model tidak ada. Ollama belum dinyalakan, model belum diunduh, nama model salah ketik: masing-masing membuat program yang naif berhenti mendadak. *Hindari:* tangkap galatnya di satu tempat, katakan apa yang terjadi, dan beralihlah ke pertanyaan dan jawaban templat.

Mengirim lalu lintas lokal melalui *proxy*. Pada komputer yang memakai *proxy*, permintaan ke `localhost` dapat dikirim ke *proxy* dan gagal. *Hindari:* bangun klien HTTP tanpa *proxy*, seperti kedua klien di sini.

Menguji hanya dengan model sungguhan. Jawaban yang berubah setiap kali jalan menghasilkan uji yang lulus hari ini dan gagal besok. *Hindari:* uji logika program dengan server

pengganti (atau dengan objek model palsu, seperti uji-uji bab ini), dan uji modelnya secara terpisah, dengan pesan sungguhan.

Menyimpan pesan pelanggan "untuk nanti". Pesan itu memuat data pribadi yang tidak Anda perlukan. *Hindari:* simpan hanya fakta yang dipahami, dan hanya bila Anda akan memakainya.

13.11 Ringkasan Bab

- Model bahasa melanjutkan teks dengan lancar, tetapi tidak memiliki pengetahuan yang sudah diperiksa, dapat **berhalusinasi**, dan dapat menjawab berbeda setiap kali; basis pengetahuan VCIRS memutuskan dari kasus-kasus yang sudah diperiksa, dengan cara yang sama setiap kali, beserta penjelasan [1], [2].
- Dalam antarmuka obrolan (**Gambar 13.1**) model memiliki dua tugas yang sempit — **memahami** pesan sebagai fakta dan **merangkai** saran menjadi kalimat — sedangkan keputusan ada pada basis pengetahuan. **Penyaring** hanya meloloskan variabel dan nilai yang dikenal, dan **penjaga** hanya meloloskan jawaban yang memberikan saran kata demi kata, tidak menambahkan perbaikan lain, dan tidak memakai frasa terlarang.
- **Ollama** menjalankan model terbuka di komputer Anda sendiri dan menjawab `POST /api/chat` di port 11434; rilis September 2026 adalah v0.34.3 [6]. Model kecil seperti `qwen3.5:2b` (2,7 GB) cocok untuk komputer konter (**Tabel 13.1**). Kolom `format` meminta **keluaran terstruktur** yang sesuai dengan JSON Schema [5], [10].
- Klien (**Kode 13.1**, **Kode 13.2**) dan pertanyaan pertama (**Kode 13.3**, **Kode 13.4**) hanya memakai pustaka standar. **Server pengganti** (**Kode 13.5**, **Kode 13.6**) menjawab dalam format Ollama dari sebuah buku frasa, sehingga bab ini berjalan tanpa internet dan dapat diulang (**Gambar 13.2**).
- Dokter obrolan (**Kode 13.7**, **Kode 13.8**) memahami pesan yang lengkap, menanyakan fakta yang kurang dengan jawaban berupa kata apa saja, menolak nilai yang tidak dikenal, berterus terang ketika tidak punya saran, menolak jawaban yang ceroboh, dan tetap bekerja tanpa model sama sekali.

13.12 Soal Latihan

13.1 Dengan kata-kata Anda sendiri, daftarlah tugas-tugas model bahasa dan tugas-tugas basis pengetahuan dalam **Gambar 13.1**. Untuk setiap tugas model, sebutkan potongan kode yang memeriksa hasilnya.

13.2 Dalam percakapan 2 model memahami "the router lights are blinking" sebagai `router lights = blinking`, dan penyaring mengabaikannya. Mas Anto berkata bahwa menurut pengalamannya lampu yang berkedip berarti router masih menyala ulang. Apa yang sebaiknya ia lakukan agar dokter dapat menangani lampu berkedip — mengubah penyaring, perintah, atau basis pengetahuan? Jelaskan, dan gambarkan kasus yang akan ia ajarkan.

13.3 Manakah dari jawaban berikut yang diterima penjaga untuk saran *turn off airplane mode*, dan mengapa? (a) "Please turn off airplane mode; your plane symbol is on." (b) "Switch off flight mode, then try again." (c) "Please turn off airplane mode. If that fails, turn on Wi-Fi." (d) "Please turn off airplane mode. The battery is fine."

13.4 Ubahlah `check_reply` (dalam salah satu bahasa) agar juga menolak jawaban yang memuat bilangan tiga digit atau lebih, seperti nomor telepon atau harga yang dikarang model. Tunjukkan baris-baris barunya dan sebutkan percakapan mana di Subbab 13.6 yang terpengaruh.

13.5 Rancanglah perluasan yang membuat dokter menjawab dalam bahasa Indonesia sementara penjaga tetap memeriksa sarannya. Dari mana kata-kata Indonesia untuk setiap saran berasal, bagaimana penjaga memakainya, dan seperti apa jawaban templatnya?

13.6 Dalam percakapan 5 pelanggan harus mengetik nilai yang persis. Andaikan model tidak ada tetapi pelanggan menjawab "yes it is on" untuk pertanyaan mode pesawat. Apa yang dicetak program, dan apa yang dilakukan mesin dengan jawaban itu? Apakah Anda akan mengubah perilaku ini?

13.7 Buku frasa server pengganti memetakan "the others too" ke `other devices online = no`. Jelaskan mengapa buku frasa tidak pernah dapat menggantikan model bahasa, dengan dua contoh pesan yang akan dipahami keliru oleh buku frasa.

13.8 Perkirakan berapa permintaan ke model yang diperlukan percakapan 2, dan berapa yang diperlukan percakapan 1. Bila setiap permintaan memakan empat detik di laptop konter, berapa lama setiap percakapan? Apa artinya bagi keputusan kapan memakai dokter obrolan dan kapan memakai dokter tanya-jawab?

13.13 Penyelesaian Soal Latihan

Penyelesaian 13.1. Model punya dua tugas. (1) *Memahami*: mengubah pesan atau jawaban menjadi fakta; hasilnya diperiksa oleh penyaring dalam `understand`, yang hanya menyimpan variabel dalam daftar dan nilai yang dikenal basis pengetahuan. (2) *Merangkai*: mengubah saran dan alasannya menjadi kalimat; hasilnya diperiksa oleh `check_reply`, sang penjaga. Basis pengetahuan punya tiga tugas: memutuskan node mana yang menyala (`kb.infer`), menanyakan fakta yang masih dibutuhkan (melalui *asker*), dan menjelaskan saran (kondisi-kondisi node yang menyala, yang menjadi alasan). Ada pula tugas keempat yang diam: menyatakan bahwa ia tidak punya saran, yang diubah program menjadi kalimat tetap.

Penyelesaian 13.2. Bukan penyaringnya: sudah benar mengabaikan nilai yang tidak dikenal basis pengetahuan, karena belum ada yang memeriksa artinya. Bukan pula hanya perintahnya: memberi tahu model bahwa "blinking berarti mati" akan menaruh pengetahuan yang belum diperiksa di tempat yang tidak pernah ditinjau siapa pun. Tempat yang benar adalah basis pengetahuan. Pertama, nyatakan `blinking` sebagai salah satu jawaban biasa `router lights` dalam `wifi-questions.json`, agar penyaring menerimanya dan pertanyaannya menawarkannya. Lalu, lain kali ada pelanggan dengan lampu yang berkedip, ajarkan kasusnya di konter dengan dokter Bab 3: fakta `connection = wifi`, `wifi icon = on`, `other devices online = no`, `router lights = blinking`, perbaikan *wait five minutes for the router to start*. Mesin menggantungkannya sebagai pengecualian *restart the router*, di samping pengecualian untuk lampu merah dan lampu mati, dan dokter obrolan langsung memakainya.

Penyelesaian 13.3. (a) Diterima: memuat saran kata demi kata, tanpa perbaikan lain, tanpa frasa terlarang, dan pendek. (b) Ditolak: tidak memuat "turn off airplane mode" kata demi kata, meskipun maknanya sama; pelanggan menerima jawaban templat. (c) Ditolak: menambahkan perbaikan lain, *turn on Wi-Fi*. (d) Ditolak: "battery" ada dalam daftar frasa terlarang. Pelajaran dari (b) adalah bahwa penjaga menukar sebagian jawaban yang baik dengan kepastian.

Penyelesaian 13.4. Dalam Python, tambahkan sebelum pemeriksaan panjang:

```
if re.search(r"\d{3,}", reply):
    problems.append("it contains a long number")
```

dengan `import re` di bagian atas; dalam Java, `if (reply.matches("(?s).*\\d{3,}.*")) { problems.add("it contains a long number"); }`. Tidak satu pun dari kelima percakapan terpengaruh, karena tidak ada jawaban yang memuat bilangan tiga digit atau lebih; aturan ini melindungi dari model yang mengarang nomor telepon atau harga. (Jawaban yang menyebut "20 minutes" hanya punya dua digit dan tetap lolos.)

Penyelesaian 13.5. Simpanlah tabel kecil, misalnya `advice-id.json`, yang memetakan setiap nilai saran ke rumusan Indonesianya, ditulis dan diperiksa Mas Anto: `"turn off airplane mode": "matikan mode pesawat"`. Program mengirim rumusan Indonesia itu kepada model sebagai baris `ADVICE`, dan penjaga memeriksa jawaban terhadap rumusan itu, serta terhadap rumusan Indonesia semua perbaikan lain. Jawaban templat menjadi "Silakan matikan mode pesawat. Saya menyarankannya karena …", dengan alasan ditulis sebagai "wifi icon adalah off" atau, lebih baik, dengan tabel kecil kedua berisi rumusan Indonesia untuk fakta-fakta. Basis pengetahuannya sendiri tidak berubah; hanya kata-kata di sekitarnya yang berubah.

Penyelesaian 13.6. Tanpa model, `ask` hanya menerima jawaban yang persis salah satu nilai yang dikenal. "yes it is on" bukan salah satunya, sehingga program mencetak `[not understood] please use one of: on / off` dan mengembalikan `None`. Mesin memperlakukan fakta itu sebagai tidak diketahui, pengecualian *turn off airplane mode* tidak menyala, dan pelanggan menerima *turn on Wi-Fi* — saran yang benar untuk apa yang diketahui mesin, tetapi bukan untuk pelanggan ini. Perilaku yang lebih baik adalah bertanya lagi, sampai dua kali, bila jawabannya tidak dipahami; itu hanya sebuah perulangan kecil di sekitar pembacaan dalam `ask`.

Penyelesaian 13.7. Buku frasa mencocokkan teks, bukan makna. "The others too have no internet" memuat "the others too" dan dipahami dengan benar sebagai *no*; tetapi "The others too are fine" juga memuatnya dan akan disalahpahami sebagai *no*. "My phone is not in airplane mode, the plane is off" memuat "plane is off" dan benar, tetapi "the plane is on the table" memuat "plane is on" dan salah. Model bahasa memakai seluruh kalimat untuk memutuskan makna sebuah frasa; itulah tepatnya bagian yang kita serahkan kepadanya.

Penyelesaian 13.8. Percakapan 1 memerlukan dua permintaan: satu untuk memahami pesan dan satu untuk merangkai jawaban. Percakapan 2 memerlukan empat: satu untuk pesan pertama, satu untuk "the wifi", satu untuk "nope, none of them", dan satu untuk jawaban; "on" dan "red" adalah nilai yang dikenal persis dan sama sekali tidak memerlukan permintaan. Dengan empat detik per permintaan, percakapan 1 membuat pelanggan menunggu sekitar 8 detik dan percakapan 2 sekitar 16 detik, ditambah waktu pelanggan mengetik. Untuk pesan yang datang pada malam hari itu tidak masalah. Di konter yang ramai, tempat orang dapat memilih jawaban dari daftar dalam sedetik, dokter tanya-jawab Bab 3 lebih cepat; dokter obrolan sepadan dengan waktunya di tempat orang menulis secara bebas dan tidak tahu fakta mana yang penting.

Daftar Pustaka

[1] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023.

[2] M. M. I. Subakti, "A Variable-Centered Intelligent Rule System," M.Sc.Eng. thesis, Dept. of Computer Science and Information Engineering, Taipei, Taiwan, Jun. 2005.

[3] P. Compton and B. H. Kang, *Ripple-Down Rules: The Alternative to Machine Learning*. Boca Raton, FL, USA: CRC Press, 2021.

[4] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 2020, pp. 9459–9474.

[5] Ollama, "Generate a chat message (POST /api/chat)," *Ollama API documentation*, 2026. [Online]. Available: https://docs.ollama.com/api/chat

[6] Ollama, "Releases: v0.34.3," GitHub, Sep. 19, 2026. [Online]. Available: https://github.com/ollama/ollama/releases

[7] Ollama, "qwen3.5," *Ollama model library*, 2026. [Online]. Available: https://ollama.com/library/qwen3.5

[8] Ollama, "llama3.2," *Ollama model library*, 2026. [Online]. Available: https://ollama.com/library/llama3.2

[9] Ollama, "gemma4," *Ollama model library*, 2026. [Online]. Available: https://ollama.com/library/gemma4

[10] Ollama, "Structured outputs," *Ollama blog*, Dec. 6, 2024. [Online]. Available: https://ollama.com/blog/structured-outputs

[11] Ollama, "Download Ollama," 2026. [Online]. Available: https://ollama.com/download

[12] I. Subakti and R. Hidayatullah, "Aplikasi sistem pakar untuk diagnosis awal gangguan kesehatan secara mandiri menggunakan Variable-Centered Intelligent Rule System," vol. 6, no. 1, pp. 11–16, Jan. 2007.

Bab 14

Proyek Akhir: Pembangun Sistem Pakar VCIRS untuk Siapa Saja

Setelah menyelesaikan bab ini Anda akan mampu merancang aplikasi web kecil yang memungkinkan seseorang yang belum pernah memprogram membangun sistem pakarnya sendiri: ia menulis pertanyaan-pertanyaan, mengajarkan kasus-kasus nyata satu per satu, berkonsultasi dengan pengetahuannya, membacanya sebagai pohon dan sebagai rule, melihat statistik pemakaian yang dihitung VCIRS — *Variable, Node,* dan *Rule Usage Rate* — beserta artinya bagi dia, menerima atau mengabaikan node dan rule baru yang diusulkan VCIRS, serta mengekspor atau mengimpor seluruh basis pengetahuan sebagai satu berkas JSON. Anda akan membangun aplikasi itu dua kali, dalam Java dengan server web bawaan JDK dan dalam Python dengan Flask, sedemikian rupa sehingga kedua server menampilkan halaman yang sama bita demi bita dan menulis berkas basis pengetahuan yang sama; Anda akan memisahkan apa yang *dikerjakan* aplikasi dari bagaimana ia *tampil* agar setiap bagian dapat diuji sendiri; dan Anda akan melihat pembangun yang sudah jadi dalam bahasa Inggris dan bahasa Indonesia, melalui tangkapan layar aplikasi yang sedang berjalan.

14.1 Buku Catatan Bu Wati

Bu Wati membuat kue. Tokonya ada di lantai bawah rumahnya: sebuah etalase kaca berisi bolu, *brownies*, dan kue lapis, dua oven di belakang, dan, di rak di atas tepung, sebuah buku tulis sekolah bersampul biru yang lembut. Di buku itu, selama sebelas tahun, ia mencatat setiap kue yang gagal dan apa yang ia temukan sebagai penyebabnya. *Bantat di tengah — Tini membuka oven untuk mengintip. Padat seperti bata — telur langsung dari kulkas. Kering — lupa pengatur waktu, kelebihan 15 menit.* Setiap kali ada pembantu baru, Bu Wati membacakan buku itu untuknya selama beberapa sore.

Kini dua pembantu bekerja bersamanya, Rini dan seorang gadis yang lebih muda yang baru mulai bulan ini. Ketika kue gagal pada hari Sabtu yang ramai, mereka tidak dapat berhenti untuk membaca catatan sebelas tahun, dan Bu Wati tidak selalu ada. Ia ingin apa yang diketahui seluruh buku itu tersedia bagi mereka dalam satu menit, di laptop tua dekat oven atau di ponselnya. Ia juga ingin terus menulis di dalamnya — artinya, terus menambah pengetahuan — tanpa bantuan siapa pun, karena ia tidak memprogram, dan tidak akan belajar memprogram.

Bab-bab sebelumnya dalam buku ini sudah memberinya sebagian besar yang ia butuhkan. Bab 3 memperlihatkan bagaimana pengetahuan tumbuh kasus demi kasus, dengan aman, sebagai pengecualian. Bab 4 memperlihatkan cara berpikir dengan variabel, Bab 6 bagaimana VCIRS mengusulkan rule baru dari statistik pemakaiannya, Bab 7 apa statistik itu, dan Bab 10 bagaimana menaruh sistem pakar di halaman web. Namun setiap aplikasi itu dibangun oleh seorang pemrogram untuk satu keperluan. Bab ini membangun alat yang membuat pemrogram tidak lagi diperlukan: sebuah **pembangun sistem pakar**, aplikasi web kecil tempat siapa saja dapat membangun, menumbuhkan, dan membagikan basis pengetahuan VCIRS tentang apa saja — kue, ponsel, sepeda motor, sawah — hanya dengan peramban.

14.2 Gagasannya dengan Kata-Kata Sederhana

14.2.1 Apa yang harus dapat dilakukan orang dengan pembangun

Mulailah dari orangnya, bukan dari programnya. Apa yang perlu dilakukan Bu Wati, dengan kata-katanya sendiri? Ia perlu menuliskan *pertanyaan-pertanyaan* yang ia ajukan tentang kue yang gagal, beserta jawaban-jawaban biasanya. Ia perlu *mengajarkan* sebuah kasus: "kue ini bantat, pintu oven dibuka terlalu awal, dan perbaikannya adalah jangan membuka pintu oven". Ia perlu *bertanya* — berkonsultasi dengan pengetahuannya untuk kue yang baru. Ia perlu *melihat* seperti apa basis pengetahuannya sekarang, dan *menghentikan* saran yang tidak lagi ia percayai. Ia perlu tahu *pertanyaan mana yang paling penting*, dan mendengar *usulan* mesin untuk rule yang lebih sederhana. Dan ia perlu *menyimpan* semuanya, untuk menyimpan salinan atau memberikannya kepada adiknya, yang punya toko kue di kota lain.

Masing-masing menjadi satu halaman pembangun, dan setiap halaman bersandar pada satu gagasan VCIRS yang sudah dijelaskan bab sebelumnya. **Tabel 14.1** menjajarkannya.

Tabel 14.1 Halaman-halaman pembangun, apa yang dilakukan orang di setiap halaman, dan gagasan VCIRS di baliknya.

Halaman	Yang dilakukan orang	Gagasan VCIRS di baliknya
Pertanyaan	menulis pertanyaan dan jawaban-jawaban biasanya	variabel dan nilainya (Bab 4)
Ajari	menceritakan kasus nyata dan memberi jawaban yang tepat	belajar kasus demi kasus, dengan kasus *cornerstone* dan daftar perbedaan (Bab 3)
Konsultasi	memilih jawaban yang ia ketahui dan mendapat saran	inferensi RDR dengan penjelasan (Bab 3 dan 8)
Pengetahuan	membaca node sebagai pohon, menghentikan node	Struktur Node dan Struktur Rule; menghentikan, bukan menghapus
Statistik	melihat pertanyaan dan node mana yang paling banyak dipakai	*Variable, Node,* dan *Rule Usage Rate* (Bab 7)
Usulan	menyimpan node dan rule baru yang masuk akal	pembangkitan rule dan node (Bab 6)
Beranda	mengekspor, mengimpor, memulai proyek baru	berkas JSON basis pengetahuan yang dipakai bersama Java dan Python

Dua prinsip menuntun setiap halaman. Pertama, **orang berbicara dalam kasus, bukan dalam rule**. Tidak ada satu tempat pun di pembangun tempat Bu Wati mengetik "IF … THEN …". Ia menceritakan kue yang gagal dan mengatakan apa yang memperbaikinya; mesinlah yang memutuskan di mana pengetahuan itu seharusnya berada, persis seperti di Bab 3, dan memberitahunya dalam satu kalimat apa yang telah dilakukan. Kedua, **tidak ada yang dilakukan orang yang dapat merusak apa yang sudah ia lakukan sebelumnya**. Mengajar menambahkan pengecualian yang tidak mengusik kasus lama; menghentikan node tetap menyimpan node itu beserta riwayatnya; usulan hanyalah usulan; dan setiap perubahan langsung disimpan, sehingga menutup peramban tidak menghilangkan apa pun. Inilah janji-janji yang sama yang membuat Ripple Down Rules dapat dipakai para ahli patologi yang bukan pemrogram [1], [2], kini ditepati oleh sebuah halaman web.

Pembangun yang baik untuk orang awam juga mengikuti aturan-aturan umum perangkat lunak yang mudah dipakai, yang sudah dituliskan para perancang selama bertahun-tahun: tunjukkan kepada orang di mana ia berada, pakailah bahasanya dan bukan bahasa sistem, cegah kesalahan alih-alih memarahinya, dan buat hasil setiap tindakan langsung terlihat [3]. Kita akan menjumpai

masing-masing aturan itu di halaman-halamannya, dan kita juga akan menjumpai tempat-tempat di mana pembangun, terus terang saja, masih berbicara dalam bahasa mesin.

14.2.2 Statistik pemakaian dengan kata-kata sederhana

Basis pengetahuan yang dibangun dari kasus tumbuh sesuai urutan datangnya kasus, bukan sesuai urutan pentingnya. VCIRS mengukur pentingnya dengan menghitung bagaimana pengetahuan itu *dipakai* oleh strukturnya sendiri — berapa banyak node yang memakai sebuah variabel bersama-sama, dan berapa banyak kasus kemudian yang sependapat dengan sebuah kondisi [4], [5]. Pembangun menampilkan tiga ukuran ini di halaman Statistik. Tampaknya teknis, tetapi masing-masing menjawab pertanyaan yang akan diajukan Bu Wati.

Berapa banyak node yang memakai pertanyaan ini? Hitungan itu disebut **node sharing** variabel, ditulis NS. Dalam basis pengetahuan Bu Wati, `problem` muncul di enam node dan `oven heat` di tiga: `problem` adalah pertanyaan yang paling penting, dan pembangun menanyakannya lebih dulu.

Seberapa berarti sebuah kondisi di dalam node-nya? Setiap kondisi sebuah node berdiri pada suatu posisi, yaitu **urutan variabel**-nya (VO), di antara TV kondisi node itu. Semakin belakang sebuah kondisi, semakin dekat ia dengan konklusi, dan VCIRS memberinya bobot lebih besar melalui **derajat kedekatan** CD. Setiap kondisi juga membawa **Credit**: nilainya mulai dari 1 dan bertambah 1 setiap kali kasus yang kemudian diajarkan sebagai pengecualian node ini sependapat dengan kondisi tersebut (Bab 7 menceritakan kisah lengkapnya). **Variable Usage Rate** sebuah kondisi menggabungkan ketiganya [4]:

$$\mathrm{VUR} = \mathrm{Credit} \times \mathrm{NS} \times \mathrm{CD}, \qquad \mathrm{CD} = \frac{\mathrm{VO}}{\mathrm{TV}} \tag{14.1}$$

Seberapa banyak sebuah node, dan sebuah rule, dipakai? Node Usage Rate sebuah node adalah rata-rata VUR kondisi-kondisinya, dan **Rule Usage Rate** sebuah rule — jalur node dari tingkat teratas ke bawah — adalah rata-rata NUR node-nodenya [4]:

$$\mathrm{NUR}(n) = \frac{1}{\mathrm{TV}_n} \sum_{i=1}^{\mathrm{TV}_n} \mathrm{VUR}_i\,, \qquad \mathrm{RUR}(r) = \frac{1}{k} \sum_{j=1}^{k} \mathrm{NUR}\,(n_j) \tag{14.2}$$

Mari kita hitung satu contoh dengan tangan pada basis pengetahuan Bu Wati, sebagaimana keadaannya setelah sesi di Subbab 14.4. Node *bake longer#1* punya satu kondisi, `problem = sank in the middle`, jadi TV bernilai 1 dan VO bernilai 1, sehingga CD = 1/1 = 1. Variabel `problem` dipakai oleh enam node, jadi NS = 6. Credit kondisi itu mulai dari 1 dan dinaikkan dua kali, sekali ketika *keep the door closed* diajarkan di bawah node ini dan sekali ketika *heat the oven fully first* diajarkan, karena kedua kasus itu juga memuat `problem = sank in the middle`; jadi Credit = 3. Menurut Persamaan (14.1), VUR = 3 × 6 × 1 = 18, dan karena node itu hanya punya satu kondisi, NUR-nya juga 18. Pengecualiannya, *keep the door closed#1*, punya satu kondisi, `oven door opened early = yes`: Credit 1, NS 2 (variabelnya dipakai dua node), CD 1, jadi VUR = NUR = 2. Rule yang berakhir di *keep the door closed#1* melewati kedua node itu, sehingga menurut Persamaan (14.2) RUR-nya (18 + 2) / 2 = 10.

Apa yang dipelajari Bu Wati dari angka-angka ini? Laju yang **besar** menandai pengetahuan yang menjadi pijakan banyak pengetahuan lain: node *bake longer* adalah akar sekeluarga pengecualian, dan bila node itu salah, semua pengecualiannya dicapai dengan cara yang salah. Node-node itulah

yang harus ditinjau paling cermat. Pertanyaan dengan laju besar sebaiknya ditanyakan **lebih dulu**, karena paling baik memisahkan kasus-kasus; halaman Konsultasi dan urutan pertanyaan pembangun memakai tepat hal ini. Laju yang **kecil** menandai pengetahuan yang masih muda dan sempit — sering kali sepenuhnya benar, tetapi bertumpu pada satu kasus, dan layak diawasi.

> **WAWASAN KUNCI:** Laju pemakaian tidak mengatakan apakah sebuah saran *benar*. Laju itu mengatakan seberapa besar bagian basis pengetahuan yang *bergantung* padanya. Kebenaran datang dari kasus-kasus yang diperiksa manusia; laju memberi tahu di mana ia harus melihat lebih dulu.

14.2.3 Usulan: mesin menyarankan, manusia memutuskan

Dari statistik yang sama VCIRS dapat **membangkitkan** pengetahuan baru, seperti diperlihatkan Bab 6 [4], [6]. Ia mencari node-node terpenting (yang dipakai bersama oleh paling banyak rule) dan mengusulkan rule baru yang mengunjungi node-node itu dalam urutan yang baik; dan ia mencari variabel-variabel terpenting (yang dipakai bersama oleh paling banyak node) dan mengusulkan, untuk setiap node yang memakai salah satunya, node baru yang kondisi-kondisinya diambil dari semua node dalam urutan yang baik. Usulan-usulan itu sengaja murah hati: sebuah usulan node mendaftar *banyak* kondisi, dan orangnya mencentang kondisi-kondisi yang memang saling berkaitan. VCIRS tidak pernah menambahkan butir hasil pembangkitan dengan sendirinya — tesisnya menyatakan dengan jelas bahwa pengguna hanya menyimpan butir-butir yang benar secara makna [4].

Di sinilah pembangun untuk orang awam harus paling berhati-hati dengan kata-kata. Halaman Usulan menyebutnya *usulan* dan berkata, di bagian atasnya, “simpan yang masuk akal dan abaikan sisanya”. Sebuah usulan node tampil sebagai daftar kotak centang, semuanya belum dicentang. Di Subbab 14.7 kita akan melihat Bu Wati memandangi usulan berisi delapan kondisi dan menyimpan tepat satu — `masalah = kering` — untuk membuat node yang lebih pendek dan lebih umum: *kue yang kering berarti angkat lebih awal*, apa pun yang terjadi selain itu. Itulah jenis generalisasi yang dibuat seorang pakar secara alami begitu ia melihatnya tertulis, dan tugas pembangun adalah menaruhnya di depan matanya.

14.2.4 Satu aplikasi, dua server, halaman yang sama

Gambar 14.1 memperlihatkan bagaimana pembangun disusun.



Gambar 14.1 Pembangun. Peramban bercakap dengan server web kecil (Flask di Python, HttpServer JDK di Java). Server hanya menghubungkan web dengan dua bagian yang sama sekali tidak mengenal web: halaman, yang menyusun HTML dari basis pengetahuan dan dari kata-kata di `ui-text.json`, dan pembangun, yang mengubah basis pengetahuan melalui vcirs-core dan menyimpannya setelah setiap perubahan.

Keputusan terpenting dalam gambar itu adalah pemisahan menjadi tiga bagian. **Pembangun** (`Builder.java`, `builder.py`) adalah semua yang dapat dilakukan orang dengan basis pengetahuan, ditulis sebagai metode biasa: `declare`, `teach`, `consult`, `stop`, `accept_node`, `export_json`, `import_json`. Ia tidak tahu apa-apa tentang peramban atau HTML. **Halaman** (`Pages.java`, `pages.py`) mengubah basis pengetahuan menjadi HTML; halaman tidak mengubah apa pun. **Server** (`BuilderServer.java`, `builder_app.py`) membaca permintaan web, memanggil pembangun, meminta HTML kepada halaman, lalu mengirimkannya kembali. Karena pembangun tidak mengenal web, kita dapat menguji dan memperagakannya dengan skrip biasa (Subbab 14.4) sebelum satu halaman pun ada. Karena halaman tidak mengubah apa pun, kita dapat membandingkan halaman kedua bahasa bita demi bita.

Kata-kata pada halaman tidak ada di dalam kode. Semuanya berasal dari `ui-text.json`, yang memuat setiap label dan petunjuk dalam bahasa Inggris dan bahasa Indonesia; sebuah tautan di bagian atas setiap halaman berpindah di antara keduanya. Bingkai setiap halaman — kepala halaman, navigasi, dan gaya tampilan — berasal dari `page.html`. Kedua berkas itu dipakai bersama server Java dan server Python, dan itulah salah satu alasan keduanya menghasilkan bita yang sama.

14.3 Langkah 1: Pembangun, tanpa Web

Kode 14.1 adalah pembangun dalam Python.

Kode 14.1 `builder.py`: semua yang dapat dilakukan orang dengan basis pengetahuan, tanpa web.

```
"""Pembangun sistem pakar: semua yang dapat dilakukan orang dengan basis
pengetahuan, tanpa web. Aplikasi web (builder_app.py) hanya menampilkan
formulir dan memanggil metode ini; builder_demo.py memanggilnya dari skrip.
"""

import json
import os
import pathlib
import sys

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import (Case, Conclusion, Condition, KnowledgeBase,  # noqa: E402
                   VcirsError)

DEFAULT_GOAL = "advice"


class Builder:
    def __init__(self, path):
        self.path = path
        if os.path.exists(path):
            self.kb = KnowledgeBase.load(path)
        else:
            self.kb = KnowledgeBase("My expert system", "")

    def save(self):
        self.kb.save(self.path)

    # ----- yang dibutuhkan formulir ---------------------------------
    def goal(self):
        """Variabel yang menjadi pokok jawaban: variabel node pertama."""
        for node in self.kb.nodes.values():
            return node.conclusions[0].variable
        return DEFAULT_GOAL
```

Kode 14.1 (Lanjutan) *builder.py: semua yang dapat dilakukan orang dengan basis pengetahuan, tanpa web.*

```
    def question_variables(self):
        """Variabel yang ditanyakan: yang paling banyak dipakai lebih dulu,
        lalu sisanya sesuai urutan dibuat. Variabel jawaban tidak ikut."""
        asked = set(c.variable for n in self.kb.nodes.values()
                    for c in n.conditions)
        answers = set(c.variable for n in self.kb.nodes.values()
                      for c in n.conclusions)
        order = self.kb.usage().question_order()
        rest = [v for v in self.kb.variables if v not in order]
        return [v for v in order + rest
                if v in asked or v not in answers and v != self.goal()]

    # ----- mengubah basis pengetahuan -----------------------------------
    def new_project(self, name, description):
        name = name.strip()
        if not name:
            raise VcirsError("please give the project a name")
        self.kb = KnowledgeBase(name, description.strip())
        self.save()

    def declare(self, variable, question, choices_text):
        variable = variable.strip()
        if not variable or "=" in variable or "<" in variable \
                or ">" in variable:
            raise VcirsError("a question name must not be empty or contain "
                             "= < >")
        choices = [c.strip() for c in choices_text.split(",") if c.strip()]
        self.kb.declare(variable, question.strip(), choices)
        self.save()

    def make_case(self, pairs, answer=""):
        """pairs: (variabel, nilai), nilai kosong dilewati; answer adalah
        nilai tujuan, atau 'variabel = nilai'."""
        facts = [Condition(v, "=", x.strip()) for v, x in pairs if x.strip()]
        conclusions = []
        if answer.strip():
            if "=" in answer:
                conclusions.append(Conclusion.parse(answer.strip()))
            else:
                conclusions.append(Conclusion(self.goal(), answer.strip()))
        return Case(facts, conclusions)
```

Kode 14.1 (Lanjutan) *builder.py: semua yang dapat dilakukan orang dengan basis pengetahuan, tanpa web.*

```
    def teach(self, pairs, answer, teacher):
        """Hasil: (jawaban basis pengetahuan sebelumnya, LearnResult)."""
        case = self.make_case(pairs, answer)
        if not case.facts:
            raise VcirsError("please give at least one fact")
        if not case.conclusions:
            raise VcirsError("please give the right answer")
        before = self.kb.infer(case.known()).what()[0]
        result = self.kb.learn(case, teacher.strip())
        self.save()
        return before, result

    def consult(self, pairs):
        return self.kb.infer(self.make_case(pairs).known())

    def stop(self, node_id, teacher):
        self.kb.stop(node_id, teacher.strip())
        self.save()

    def accept_rule(self, rule_id, teacher):
        for g in self.kb.generate_rules():
            if g.id == rule_id:
                rule = self.kb.accept_rule(g, g.nodes, teacher.strip())
                self.save()
                return rule
        raise VcirsError("that proposal is no longer there")

    def accept_node(self, node_id, keep, teacher):
        for g in self.kb.generate_nodes():
            if g.id == node_id:
                node = self.kb.accept_node(g, keep, teacher.strip())
                self.save()
                return node
        raise VcirsError("that proposal is no longer there")

    # ----- ekspor dan impor ---------------------------------------------
    def export_json(self):
        return self.kb.to_json()
```

Kode 14.1 (Lanjutan) *builder.py: semua yang dapat dilakukan orang dengan basis pengetahuan, tanpa web.*

```
    def import_json(self, text):
        """Berkas basis pengetahuan (vcirs-kb) atau berkas kasus ("cases")."""
        try:
            data = json.loads(text)
            if isinstance(data, dict) and "cases" in data:
                kb = cases_to_kb(data)
            else:
                kb = KnowledgeBase.from_json(text)
        except (ValueError, KeyError, TypeError, AttributeError, VcirsError):
            raise VcirsError("this is not a knowledge base or case file I "
                             "can read") from None
        self.kb = kb
        self.save()


def cases_to_kb(data):
    """Bangun basis pengetahuan dari berkas kasus, seolah diajarkan satu-satu."""
    kb = KnowledgeBase(data.get("name", "Imported cases"),
                       data.get("description", ""))
    for v in data.get("variables", []):
        kb.declare(v["variable"], v.get("question", ""), v.get("choices", []))
    for c in data["cases"]:
        case = Case.of(c["facts"], c["conclusions"], c.get("cf", 100))
        kb.learn(case, c.get("author", ""))
    return kb
```

Bacalah sebagai daftar janji kepada penggunanya.

Ia membuka apa yang sudah ada. Konstruktornya memuat berkas basis pengetahuan bila ada, dan bila tidak, memulai proyek kosong bernama “My expert system”. `save` menulis berkasnya; setiap metode yang mengubah sesuatu memanggilnya di akhir, sehingga berkas di diska selalu memuat pengetahuan terbaru.

Ia tahu pokok jawabannya. `goal` adalah variabel jawaban: variabel konklusi node pertama, atau `advice` pada proyek yang masih kosong. Di halaman Ajari orang hanya mengetik nilainya — “keep the door closed” — dan `make_case` mengubahnya menjadi `advice = keep the door closed`. Orang yang membutuhkan jenis jawaban kedua boleh mengetik `variabel = nilai`.

Ia menanyakan pertanyaan paling berguna lebih dulu. `question_variables` mendaftar variabel yang ditampilkan sebagai pertanyaan: pertama sesuai urutan dari `usage().question_order()` — variabel diperingkat menurut VUR terbaiknya, Persamaan (14.1) — lalu variabel yang sudah dideklarasikan tetapi belum dipakai node mana pun, sesuai urutan dibuatnya. Variabel yang hanya muncul sebagai jawaban tidak ikut, sehingga tujuan `advice` tidak pernah muncul sebagai pertanyaan.

Ia melindungi orang dari kesalahan sederhana. `new_project` menolak nama yang kosong. `declare` menolak nama pertanyaan yang memuat `=`, `<`, atau `>`, karena karakter-karakter itu memisahkan variabel dari nilainya dalam kondisi; jawaban-jawaban biasanya dipecah pada tanda koma dan yang kosong dibuang. `teach` menolak kasus tanpa fakta atau tanpa jawaban. Setiap penolakan adalah `VcirsError` dengan kalimat yang dapat dipahami orang; server menampilkannya di bagian atas halaman.

Ia mengajar, dan mengatakan apa yang terjadi. `teach` menyusun kasus, bertanya kepada basis pengetahuan apa yang akan ia jawab *sebelum* belajar (baris pertama `what()`), lalu memanggil `learn` — perulangan Ripple-Down-Rules sekali panggil dari Bab 3, yang menerima kasus, menambahkan pengecualian di bawah node yang menjawab salah, atau menambahkan node tingkat teratas yang baru — dan mengembalikan keduanya. Orang melihat “sebelumnya, basis

pengetahuan menjawab…" dan "yang terjadi", cara paling jujur untuk menunjukkan bahwa pengajarannya mengubah sesuatu, dan persis apa yang berubah.

Ia berkonsultasi. `consult` mengubah jawaban-jawaban yang dipilih menjadi fakta dan menjalankan `infer`. Di sini tidak ada *asker*: formulir web itu sendiri adalah daftar pertanyaan, dan pertanyaan yang dibiarkan kosong berarti tidak diketahui.

Ia menghentikan, dan menerima usulan. `stop` menghentikan sebuah node (node itu tetap disimpan, beserta riwayatnya, tetapi tidak lagi memberi saran). `accept_rule` dan `accept_node` mencari usulan dengan pengenal yang diberikan dalam daftar usulan yang baru dihitung — daftarnya bisa saja sudah berubah sejak halaman ditampilkan — lalu menerimanya dengan kondisi-kondisi yang disimpan orangnya.

Ia mengekspor dan mengimpor. `export_json` adalah JSON milik basis pengetahuan itu sendiri. `import_json` menerima dua jenis berkas. Berkas dengan daftar `cases` adalah **berkas kasus**, seperti `cake-cases.json`: `cases_to_kb` mendeklarasikan pertanyaan-pertanyaannya lalu mengajarkan kasus-kasusnya satu per satu dengan `learn`, persis seolah seseorang mengetiknya di halaman Ajari. Selain itu harus berupa berkas basis pengetahuan VCIRS. Bila teksnya bukan JSON, atau JSON dengan bentuk yang salah, orang diberi tahu, dalam satu kalimat, bahwa itu bukan berkas yang dapat dibaca pembangun — dan basis pengetahuan yang sudah ada dibiarkan utuh, karena `self.kb` baru diganti setelah yang baru berhasil dibangun.

Kode 14.2 adalah pembangun yang sama dalam Java.

Kode 14.2 `Builder.java`: pembangun yang sama dalam Java.

```
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

import vcirs.Case;
import vcirs.Conclusion;
import vcirs.Condition;
import vcirs.GeneratedNode;
import vcirs.GeneratedRule;
import vcirs.InferenceResult;
import vcirs.Json;
import vcirs.KnowledgeBase;
import vcirs.LearnResult;
import vcirs.Node;
import vcirs.Rule;
import vcirs.VcirsException;

/**
 * Pembangun sistem pakar: semua yang dapat dilakukan orang dengan basis
 * pengetahuan, tanpa web. Aplikasi web (BuilderServer) hanya menampilkan
 * formulir dan memanggil metode ini; BuilderDemo memanggilnya dari skrip.
 */
public class Builder {
    public static final String DEFAULT_GOAL = "advice";

    /** Yang dilaporkan teach(): jawaban lama dan apa yang berubah. */
    public record Taught(String before, LearnResult result) {
    }

    public final String path;
    public KnowledgeBase kb;

    public Builder(String path) {
        this.path = path;
        kb = Files.exists(Path.of(path)) ? KnowledgeBase.load(path)
                : new KnowledgeBase("My expert system", "");
    }

    public void save() {
        kb.save(path);
    }
```

Kode 14.2 (Lanjutan) *Builder.java: pembangun yang sama dalam Java.*

```
    // ----- yang dibutuhkan formulir --------------------------------------
    /** Variabel yang menjadi pokok jawaban: variabel node pertama. */
    public String goal() {
        for (Node node : kb.nodes.values()) {
            return node.conclusions.get(0).variable;
        }
        return DEFAULT_GOAL;
    }

    /**
     * Variabel yang ditanyakan: yang paling banyak dipakai lebih dulu, lalu
     * sisanya sesuai urutan dibuat. Variabel jawaban tidak ikut.
     */
    public List<String> questionVariables() {
        Set<String> asked = new HashSet<>();
        Set<String> answers = new HashSet<>();
        for (Node n : kb.nodes.values()) {
            n.conditions.forEach(c -> asked.add(c.variable));
            n.conclusions.forEach(c -> answers.add(c.variable));
        }
        List<String> all = new ArrayList<>(kb.usage().questionOrder());
        for (String v : kb.variables.keySet()) {
            if (!all.contains(v)) {
                all.add(v);
            }
        }
        List<String> out = new ArrayList<>();
        for (String v : all) {
            if (asked.contains(v) || !answers.contains(v) && !v.equals(goal())) {
                out.add(v);
            }
        }
        return out;
    }

    // ----- mengubah basis pengetahuan ------------------------------------
    public void newProject(String name, String description) {
        if (name.strip().isEmpty()) {
            throw new VcirsException("please give the project a name");
        }
        kb = new KnowledgeBase(name.strip(), description.strip());
        save();
    }
```

Kode 14.2 (Lanjutan) *Builder.java: pembangun yang sama dalam Java.*

```
    public void declare(String variable, String question, String choicesText) {
        String v = variable.strip();
        if (v.isEmpty() || v.contains("=") || v.contains("<") || v.contains(">")) {
            throw new VcirsException("a question name must not be empty or "
                    + "contain = < >");
        }
        List<String> choices = new ArrayList<>();
        for (String c : choicesText.split(",")) {
            if (!c.strip().isEmpty()) {
                choices.add(c.strip());
            }
        }
        kb.declare(v, question.strip(), choices);
        save();
    }

    /**
     * pairs: {variabel, nilai}, nilai kosong dilewati; answer adalah nilai
     * tujuan, atau "variabel = nilai".
     */
    public Case makeCase(List<String[]> pairs, String answer) {
        List<Condition> facts = new ArrayList<>();
        for (String[] p : pairs) {
            if (!p[1].strip().isEmpty()) {
                facts.add(new Condition(p[0], "=", p[1].strip()));
            }
        }
        List<Conclusion> conclusions = new ArrayList<>();
        if (!answer.strip().isEmpty()) {
            conclusions.add(answer.contains("=")
                    ? Conclusion.parse(answer.strip())
                    : new Conclusion(goal(), answer.strip()));
        }
        return new Case(facts, conclusions, 100);
    }
```

Kode 14.2 (Lanjutan) *Builder.java: pembangun yang sama dalam Java.*

```
    public Taught teach(List<String[]> pairs, String answer, String teacher) {
        Case c = makeCase(pairs, answer);
        if (c.facts.isEmpty()) {
            throw new VcirsException("please give at least one fact");
        }
        if (c.conclusions.isEmpty()) {
            throw new VcirsException("please give the right answer");
        }
        String before = kb.infer(c.known()).what().get(0);
        LearnResult result = kb.learn(c, teacher.strip());
        save();
        return new Taught(before, result);
    }

    public InferenceResult consult(List<String[]> pairs) {
        return kb.infer(makeCase(pairs, "").known());
    }

    public void stop(String nodeId, String teacher) {
        kb.stop(nodeId, teacher.strip());
        save();
    }

    public Rule acceptRule(String ruleId, String teacher) {
        for (GeneratedRule g : kb.generateRules()) {
            if (g.id.equals(ruleId)) {
                Rule rule = kb.acceptRule(g, g.nodes, teacher.strip());
                save();
                return rule;
            }
        }
        throw new VcirsException("that proposal is no longer there");
    }

    public Node acceptNode(String nodeId, List<String> keep, String teacher) {
        for (GeneratedNode g : kb.generateNodes()) {
            if (g.id.equals(nodeId)) {
                Node node = kb.acceptNode(g, keep, teacher.strip());
                save();
                return node;
            }
        }
        throw new VcirsException("that proposal is no longer there");
    }
```

Kode 14.2 (Lanjutan) *Builder.java: pembangun yang sama dalam Java.*

```
    // ----- ekspor dan impor ---------------------------------------------
    public String exportJson() {
        return kb.toJson();
    }

    /** Berkas basis pengetahuan (vcirs-kb) atau berkas kasus ("cases"). */
    public void importJson(String text) {
        KnowledgeBase loaded;
        try {
            Object data = Json.parse(text);
            if (data instanceof Map && ((Map<?, ?>) data).containsKey("cases")) {
                loaded = casesToKb((Map<?, ?>) data);
            } else {
                loaded = KnowledgeBase.fromJson(text);
            }
        } catch (RuntimeException e) {
            throw new VcirsException("this is not a knowledge base or case file "
                    + "I can read");
        }
        kb = loaded;
        save();
    }

    /** Bangun basis pengetahuan dari berkas kasus, seolah diajarkan satu-satu. */
    static KnowledgeBase casesToKb(Map<?, ?> data) {
        Object name = data.get("name");
        Object description = data.get("description");
        KnowledgeBase k = new KnowledgeBase(
                name == null ? "Imported cases" : (String) name,
                description == null ? "" : (String) description);
        Object vars = data.get("variables");
        for (Object o : vars == null ? List.of() : (List<?>) vars) {
            Map<?, ?> v = (Map<?, ?>) o;
            Object q = v.get("question");
            Object ch = v.get("choices");
            k.declare((String) v.get("variable"), q == null ? "" : (String) q,
                    strings(ch == null ? List.of() : (List<?>) ch));
        }
        for (Object o : (List<?>) data.get("cases")) {
            Map<?, ?> c = (Map<?, ?>) o;
            Object cf = c.get("cf");
            Case aCase = Case.of(strings((List<?>) c.get("facts")),
                    strings((List<?>) c.get("conclusions")),
                    cf == null ? 100 : ((Number) cf).intValue(), 100);
            Object who = c.get("author");
            k.learn(aCase, who == null ? "" : (String) who);
```

Kode 14.2 (Lanjutan) *Builder.java: pembangun yang sama dalam Java.*

```
        }
        return k;
    }

    static List<String> strings(List<?> list) {
        List<String> out = new ArrayList<>();
        for (Object o : list) {
            out.add((String) o);
        }
        return out;
    }
}
```

Versi Java lebih panjang terutama karena Java tidak punya literal kamus: sepasang fakta adalah `String[]` dengan dua elemen, dan berkas kasus dibaca dengan `Json.parse` menjadi *map* dan *list*

yang harus di-*cast*. `record Taught(String before, LearnResult result)` adalah cara ringkas untuk mengembalikan dua nilai dari `teach`. `importJson` menangkap `RuntimeException`, induk bersama dari banyak cara sebuah berkas yang salah dapat gagal (`ClassCastException` ketika ada *list* di tempat yang seharusnya string, `NullPointerException` ketika sebuah kolom tidak ada, `VcirsException` dari mesin), dan mengubah semuanya menjadi kalimat yang sama.

14.4 Langkah 2: Sesi Bu Wati, sebagai Skrip

Sebelum membangun satu halaman web pun, kita memakai pembangun dari sebuah skrip yang memerankan sesi pertama Bu Wati: memulai proyek, menulis enam pertanyaan, mengajarkan sembilan kasus dari halaman-halaman pertama buku catatannya, berkonsultasi, mengajarkan kasus baru, melihat pengetahuan, statistik, dan usulan, menerima satu usulan, lalu mengekspor dan mengimpor. Kasus-kasusnya berasal dari berkas contoh bab ini, `cake-cases.json`, yang baris-baris pertamanya:

```
{
  "name": "Cake doctor",
  "description": "Why did the cake fail? What Bu Wati teaches the helpers of her cake
shop.",
  "variables": [
    {"variable": "problem", "question": "What went wrong with the cake?", "choices":
["sank in the middle", "dense and heavy", "cracked top", "raw inside", "dry", "burnt
bottom"]},
```

Kode 14.3 adalah skripnya dalam Python.

Kode 14.3 `builder_demo.py`: sesi Bu Wati, diperankan melalui pembangun.

```
"""Bu Wati membangun dokter kuenya dengan pembangun, langkah demi langkah,
seperti di aplikasi web. Mencetak apa yang dilakukan setiap langkah.

    python builder_demo.py [knowledge-base.json]
"""

import json
import os
import sys

from builder import Builder
from vcirs import VcirsError, num, report

DATA = os.path.join(os.path.dirname(os.path.abspath(__file__)), "..", "data")


def pairs(text):
    """'problem = dry; baking time = long' -> [('problem', 'dry'), ...]"""
    out = []
    for part in text.split(";"):
        var, value = part.split("=")
        out.append((var.strip(), value.strip()))
    return out


def teach(b, facts, answer, teacher):
    try:
        before, result = b.teach(pairs(facts), answer, teacher)
    except VcirsError as e:
        print("Teach " + facts + " -> " + answer + ": refused: " + str(e))
        return
    print("Teach " + facts + " -> " + answer + " (" + teacher + ")")
    print("  before: " + before.strip())
    print("  now:    " + str(result))


def consult(b, facts):
    result = b.consult(pairs(facts))
    print("Consult " + facts)
    for line in result.what() + result.how():
        print("  " + line)
```

Kode 14.3 (Lanjutan) *builder_demo.py: sesi Bu Wati, diperankan melalui pembangun.*

```
def main(args):
    path = args[0] if args else os.path.join(DATA, "cake-kb.json")
    b = Builder(path)
    with open(os.path.join(DATA, "cake-cases.json"), encoding="utf-8") as f:
        example = json.load(f)
    print("== 1. A new project")
    b.new_project(example["name"], example["description"])
    print(b.kb.summary())
    print("== 2. The questions")
    for v in example["variables"]:
        b.declare(v["variable"], v["question"], ", ".join(v["choices"]))
        print("Question " + v["variable"] + ": " + v["question"] + " ("
              + " / ".join(v["choices"]) + ")")
    print("== 3. Teaching, case by case")
    for c in example["cases"]:
        teach(b, "; ".join(c["facts"]), c["conclusions"][0].split("= ")[1],
              c["author"])
    print("== 4. Consulting")
    consult(b, "problem = sank in the middle; oven door opened early = no; "
               "oven heat = low")
    consult(b, "problem = burnt bottom")
    teach(b, "problem = burnt bottom", "bake on a higher rack", "Wati")
    teach(b, "problem = burnt bottom", "use a lighter pan", "Rini")
    teach(b, "problem = burnt bottom; pan = dark metal", "use a lighter pan",
          "Rini")
    print("== 5. The knowledge")
    for line in report.tree(b.kb):
        print(line)
    print("== 6. Usage statistics")
    u = b.kb.usage()
    print("Ask first: " + ", ".join(b.question_variables()))
    print("Most important question: " + ", ".join(u.important_variables()))
    print("Most important nodes: " + ", ".join(u.important_nodes()))
    for row in u.nur_rows():
        print("  NUR " + num.fixed(row[2], 3) + "  " + row[0])
    print("== 7. Proposals")
    for g in b.kb.generate_rules():
        print("Rule " + g.id + ": " + " -> ".join(g.nodes))
    for g in b.kb.generate_nodes():
        print("Node " + g.id + ": " + ", ".join(str(c) for c in g.conditions))
    node = b.accept_node("take it out earlier#1G", ["problem = dry"], "Wati")
    print("Accepted: " + str(node))
    consult(b, "problem = dry")
    print("== 8. Export and import")
    text = b.export_json()
    print("Exported " + str(len(text.splitlines())) + " lines of JSON")
```

Kode 14.3 (Lanjutan) *builder_demo.py: sesi Bu Wati, diperankan melalui pembangun.*

```
    b.import_json(text)
    print("Imported again: " + b.kb.summary())
    print("Same file: " + str(b.export_json() == text).lower())


if __name__ == "__main__":
    main(sys.argv[1:])
```

`pairs` mengubah teks `problem = dry; baking time = long` menjadi daftar pasangan yang diharapkan pembangun. `teach` mencetak apa yang dijawab basis pengetahuan sebelumnya dan apa yang terjadi, atau mengapa ditolak. `consult` mencetak konklusi dan penjelasannya. **Kode 14.4** adalah skrip yang sama dalam Java.

Kode 14.4 `BuilderDemo.java`: sesi yang sama dalam Java.

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import vcirs.Condition;
import vcirs.GeneratedNode;
import vcirs.GeneratedRule;
import vcirs.InferenceResult;
import vcirs.Json;
import vcirs.Node;
import vcirs.Num;
import vcirs.Report;
import vcirs.Usage;
import vcirs.VcirsException;

/**
 * Bu Wati membangun dokter kuenya dengan pembangun, langkah demi langkah,
 * seperti di aplikasi web. Mencetak apa yang dilakukan setiap langkah.
 * java BuilderDemo [knowledge-base.json]
 */
public class BuilderDemo {
    static final String DATA = "../data/";

    /** "problem = dry; baking time = long" -> {problem, dry}, ... */
    static List<String[]> pairs(String text) {
        List<String[]> out = new ArrayList<>();
        for (String part : text.split(";")) {
            String[] kv = part.split("=");
            out.add(new String[] {kv[0].strip(), kv[1].strip()});
        }
        return out;
    }
```

Kode 14.4 (Lanjutan) *BuilderDemo.java: sesi yang sama dalam Java.*

```
    static void teach(Builder b, String facts, String answer, String teacher) {
        Builder.Taught t;
        try {
            t = b.teach(pairs(facts), answer, teacher);
        } catch (VcirsException e) {
            System.out.println("Teach " + facts + " -> " + answer + ": refused: "
                    + e.getMessage());
            return;
        }
        System.out.println("Teach " + facts + " -> " + answer + " (" + teacher
                + ")");
        System.out.println("  before: " + t.before().strip());
        System.out.println("  now:    " + t.result());
    }

    static void consult(Builder b, String facts) {
        InferenceResult result = b.consult(pairs(facts));
        System.out.println("Consult " + facts);
        List<String> lines = new ArrayList<>(result.what());
        lines.addAll(result.how());
        for (String line : lines) {
            System.out.println("  " + line);
        }
    }

    @SuppressWarnings("unchecked")
    public static void main(String[] args) throws IOException {
        String path = args.length > 0 ? args[0] : DATA + "cake-kb.json";
        Builder b = new Builder(path);
        Map<String, Object> example = (Map<String, Object>) Json.parse(
                Files.readString(Path.of(DATA + "cake-cases.json"),
                        StandardCharsets.UTF_8));
        System.out.println("== 1. A new project");
        b.newProject((String) example.get("name"),
                (String) example.get("description"));
        System.out.println(b.kb.summary());
        System.out.println("== 2. The questions");
        for (Object o : (List<Object>) example.get("variables")) {
            Map<String, Object> v = (Map<String, Object>) o;
            List<String> choices = Builder.strings((List<?>) v.get("choices"));
            b.declare((String) v.get("variable"), (String) v.get("question"),
                    String.join(", ", choices));
            System.out.println("Question " + v.get("variable") + ": "
                    + v.get("question") + " (" + String.join(" / ", choices)
                    + ")");
        }
```

Kode 14.4 (Lanjutan) *BuilderDemo.java: sesi yang sama dalam Java.*

```
        System.out.println("== 3. Teaching, case by case");
        for (Object o : (List<Object>) example.get("cases")) {
            Map<String, Object> c = (Map<String, Object>) o;
            String answer = Builder.strings((List<?>) c.get("conclusions"))
                    .get(0).split("= ")[1];
            teach(b, String.join("; ", Builder.strings((List<?>) c.get("facts"))),
                    answer, (String) c.get("author"));
        }
        System.out.println("== 4. Consulting");
        consult(b, "problem = sank in the middle; oven door opened early = no; "
                + "oven heat = low");
        consult(b, "problem = burnt bottom");
        teach(b, "problem = burnt bottom", "bake on a higher rack", "Wati");
        teach(b, "problem = burnt bottom", "use a lighter pan", "Rini");
        teach(b, "problem = burnt bottom; pan = dark metal", "use a lighter pan",
                "Rini");
        System.out.println("== 5. The knowledge");
        for (String line : Report.tree(b.kb)) {
            System.out.println(line);
        }
        System.out.println("== 6. Usage statistics");
        Usage u = b.kb.usage();
        System.out.println("Ask first: "
                + String.join(", ", b.questionVariables()));
        System.out.println("Most important question: "
                + String.join(", ", u.importantVariables()));
        System.out.println("Most important nodes: "
                + String.join(", ", u.importantNodes()));
        for (Usage.RateRow row : u.nurRows()) {
            System.out.println("  NUR " + Num.fixed(row.rate, 3) + "  " + row.id);
        }
        System.out.println("== 7. Proposals");
        for (GeneratedRule g : b.kb.generateRules()) {
            System.out.println("Rule " + g.id + ": "
                    + String.join(" -> ", g.nodes));
        }
        for (GeneratedNode g : b.kb.generateNodes()) {
            List<String> conds = new ArrayList<>();
            for (Condition c : g.conditions) {
                conds.add(c.toString());
            }
            System.out.println("Node " + g.id + ": " + String.join(", ", conds));
        }
        Node node = b.acceptNode("take it out earlier#1G", List.of("problem = dry"),
                "Wati");
        System.out.println("Accepted: " + node);
```

Kode 14.4 (Lanjutan) *BuilderDemo.java: sesi yang sama dalam Java.*

```
        consult(b, "problem = dry");
        System.out.println("== 8. Export and import");
        String text = b.exportJson();
        System.out.println("Exported " + text.split("\n").length
                + " lines of JSON");
        b.importJson(text);
        System.out.println("Imported again: " + b.kb.summary());
        System.out.println("Same file: " + b.exportJson().equals(text));
    }
}
```

Jalankan dari `Chapter14/java` dengan `javac -d out -sourcepath ../../vcirs-core/java/src *.java` dan `java -cp out BuilderDemo`, atau dari `Chapter14/python` dengan `python builder_demo.py`. Keduanya mencetak teks yang sama dan menulis `cake-kb.json` yang sama,

bita demi bita. Kita membaca keluarannya dalam lima bagian. Bagian pertama membangun pengetahuan:

```
== 1. A new project
Cake doctor: 0 variables, 0 nodes, 0 rules
== 2. The questions
Question problem: What went wrong with the cake? (sank in the middle / dense and heavy /
cracked top / raw inside / dry / burnt bottom)
Question oven door opened early: Was the oven door opened in the first 20 minutes? (yes /
no)
Question oven heat: How hot was the oven? (low / normal / high)
Question eggs: Were the eggs cold or at room temperature? (cold / room temperature)
Question flour mixing: How long was the batter mixed after the flour went in? (short /
long)
Question baking time: Was the baking time short, normal or long? (short / normal / long)
== 3. Teaching, case by case
Teach problem = sank in the middle -> bake longer (Wati)
  before: No node matched the case.
  now:    added: bake longer#1 added at the top level
Teach problem = sank in the middle; oven door opened early = yes -> keep the door closed
(Wati)
  before: Conclusion: advice = bake longer  (rule bake longer#1, CF 100)
  now:    refined: keep the door closed#1 added as an exception of bake longer#1
Teach problem = dense and heavy -> beat the eggs longer (Wati)
  before: No node matched the case.
  now:    added: beat the eggs longer#1 added at the top level
Teach problem = dense and heavy; eggs = cold -> use room-temperature eggs (Wati)
  before: Conclusion: advice = beat the eggs longer  (rule beat the eggs longer#1, CF 100)
  now:    refined: use room-temperature eggs#1 added as an exception of beat the eggs
longer#1
Teach problem = dense and heavy; eggs = room temperature; flour mixing = long -> fold the
flour in gently (Rini)
  before: Conclusion: advice = beat the eggs longer  (rule beat the eggs longer#1, CF 100)
  now:    refined: fold the flour in gently#1 added as an exception of beat the eggs
longer#1
Teach problem = cracked top; oven heat = high -> lower the heat (Wati)
  before: No node matched the case.
  now:    added: lower the heat#1 added at the top level
Teach problem = raw inside; oven heat = high -> lower the heat and bake longer (Wati)
  before: No node matched the case.
  now:    added: lower the heat and bake longer#1 added at the top level
Teach problem = dry; baking time = long -> take it out earlier (Rini)
  before: No node matched the case.
  now:    added: take it out earlier#1 added at the top level
Teach problem = sank in the middle; oven door opened early = no; oven heat = low -> heat
the oven fully first (Wati)
  before: Conclusion: advice = bake longer  (rule bake longer#1, CF 100)
  now:    refined: heat the oven fully first#1 added as an exception of bake longer#1
```

Enam pertanyaan, lalu sembilan kasus, dan untuk setiap kasus satu baris “before” dan satu baris “now”. Ikutilah dua yang pertama. Kasus pertama menjumpai basis pengetahuan yang kosong — “No node matched the case” — dan menjadi node tingkat teratas *bake longer#1*. Kasus kedua punya masalah yang sama ditambah `oven door opened early = yes`; sebelumnya basis pengetahuan akan menjawab *bake longer*, yang salah untuk kue ini, sehingga `learn` menambahkan *keep the door closed#1* sebagai pengecualian *bake longer#1*. Pola yang sama berulang untuk kue yang padat: satu node umum, *beat the eggs longer*, dan dua pengecualian di bawahnya, satu untuk telur dingin dan satu, diajarkan Rini, untuk adonan yang diaduk terlalu lama setelah tepung masuk. Tidak ada yang memilih ke mana setiap potong pengetahuan pergi; kasus-kasusnyalah yang menentukan.

Bagian kedua berkonsultasi, lalu mengajarkan sesuatu yang baru:

```
== 4. Consulting
Consult problem = sank in the middle; oven door opened early = no; oven heat = low
  Conclusion: advice = heat the oven fully first  (rule heat the oven fully first#1, CF
100)
    on the way: advice = bake longer
  heat the oven fully first#1 fired because:
    problem = sank in the middle  (given, CF 100)
    oven door opened early = no  (given, CF 100)
    oven heat = low  (given, CF 100)
    so advice = heat the oven fully first  [rule CF 100, conclusion CF 100]
Consult problem = burnt bottom
  No node matched the case.
  No rule fired.
Teach problem = burnt bottom -> bake on a higher rack (Wati)
  before: No node matched the case.
  now:    added: bake on a higher rack#1 added at the top level
Teach problem = burnt bottom -> use a lighter pan: refused: no condition tells the new case
apart; add a fact that differs from the cornerstone case
Teach problem = burnt bottom; pan = dark metal -> use a lighter pan (Rini)
  before: Conclusion: advice = bake on a higher rack  (rule bake on a higher rack#1, CF
100)
  now:    refined: use a lighter pan#1 added as an exception of bake on a higher rack#1
```

Konsultasi pertama turun sampai ke pengecualian *heat the oven fully first* dan menjelaskannya dengan tiga fakta yang membuatnya menyala; `on the way` mencatat saran umum yang dilewatinya. Konsultasi kedua, bagian bawah yang gosong, tidak menemukan apa-apa. Bu Wati mengajarkan *bake on a higher rack*, dan saran itu menjadi node tingkat teratas yang baru. Kemudian Rini mencoba mengajarkan jawaban lain, *use a lighter pan*, untuk fakta yang persis sama — dan pembangun menolak: kasus baru tidak dapat dibedakan dari kasus yang menciptakan *bake on a higher rack*. Penolakan itu adalah daftar perbedaan Bab 3 yang sedang bekerja: dua jawaban berbeda untuk fakta yang identik akan membuat basis pengetahuan bertentangan dengan dirinya sendiri. Rini menambahkan fakta yang memang berbeda — kuenya dipanggang dalam loyang logam gelap — dan kasusnya diterima sebagai pengecualian. Variabel `pan` belum ada sebelumnya; mesin membuatnya saat itu juga.

Bagian ketiga mencetak pengetahuan sebagai pohon:

```
== 5. The knowledge
(root)
+- bake longer#1: problem = sank in the middle -> advice = bake longer
|  +- keep the door closed#1: oven door opened early = yes -> advice = keep the door closed
|  +- heat the oven fully first#1: oven door opened early = no, oven heat = low -> advice =
heat the oven fully first
+- beat the eggs longer#1: problem = dense and heavy -> advice = beat the eggs longer
|  +- use room-temperature eggs#1: eggs = cold -> advice = use room-temperature eggs
|  +- fold the flour in gently#1: eggs = room temperature, flour mixing = long -> advice =
fold the flour in gently
+- lower the heat#1: problem = cracked top, oven heat = high -> advice = lower the heat
+- lower the heat and bake longer#1: problem = raw inside, oven heat = high -> advice =
lower the heat and bake longer
+- take it out earlier#1: problem = dry, baking time = long -> advice = take it out earlier
+- bake on a higher rack#1: problem = burnt bottom -> advice = bake on a higher rack
   +- use a lighter pan#1: pan = dark metal -> advice = use a lighter pan
```

Bagian keempat memperlihatkan statistiknya:

```
== 6. Usage statistics
Ask first: problem, oven heat, oven door opened early, eggs, flour mixing, baking time, pan
Most important question: problem
Most important nodes: bake longer#1, beat the eggs longer#1
  NUR 1.000  fold the flour in gently#1
  NUR 1.000  use a lighter pan#1
  NUR 2.000  keep the door closed#1
  NUR 2.000  use room-temperature eggs#1
  NUR 2.000  take it out earlier#1
  NUR 2.000  heat the oven fully first#1
  NUR 3.000  lower the heat#1
  NUR 3.000  lower the heat and bake longer#1
  NUR 12.000  bake on a higher rack#1
  NUR 18.000  bake longer#1
  NUR 18.000  beat the eggs longer#1
```

"Ask first" adalah urutan pertanyaan dari Persamaan (14.1): `problem` lebih dulu, karena VUR terbaiknya paling besar, lalu `oven heat`, dan seterusnya, diakhiri `pan` yang baru. `problem` juga pertanyaan terpenting dalam arti *node sharing*, dan dua node terpenting adalah dua node umum yang punya pengecualian: *bake longer#1* dan *beat the eggs longer#1*, masing-masing dipakai bersama oleh tiga rule. Daftar NUR, dari yang terkecil, diakhiri dengan nilai-nilai yang kita hitung dengan tangan di Subbab 14.2.2: 18 untuk *bake longer#1*, 2 untuk *keep the door closed#1*.

Bagian terakhir memperlihatkan usulan, menerima satu, dan mengekspor:

```
== 7. Proposals
Rule bake longer#1G: take it out earlier#1 -> lower the heat and bake longer#1 -> lower the
heat#1 -> bake on a higher rack#1 -> use a lighter pan#1 -> beat the eggs longer#1 -> fold
the flour in gently#1 -> use room-temperature eggs#1 -> bake longer#1
Rule beat the eggs longer#1G: take it out earlier#1 -> lower the heat and bake longer#1 ->
lower the heat#1 -> bake on a higher rack#1 -> use a lighter pan#1 -> beat the eggs
longer#1
Node bake longer#1G: pan = dark metal, eggs = room temperature, flour mixing = long, eggs =
cold, oven door opened early = yes, oven door opened early = no, oven heat = low, problem =
dry, baking time = long, problem = raw inside, problem = cracked top, oven heat = high,
problem = burnt bottom, problem = dense and heavy, problem = sank in the middle
Node beat the eggs longer#1G: pan = dark metal, eggs = room temperature, flour mixing =
long, eggs = cold, oven door opened early = yes, oven door opened early = no, oven heat =
low, problem = dry, baking time = long, problem = raw inside, problem = cracked top, oven
heat = high, problem = burnt bottom, problem = dense and heavy
Node lower the heat#1G: pan = dark metal, eggs = room temperature, flour mixing = long,
eggs = cold, oven door opened early = yes, oven door opened early = no, oven heat = low,
problem = dry, baking time = long, problem = raw inside, problem = cracked top
Node lower the heat and bake longer#1G: pan = dark metal, eggs = room temperature, flour
mixing = long, eggs = cold, oven door opened early = yes, oven door opened early = no, oven
heat = low, problem = dry, baking time = long, problem = raw inside
Node take it out earlier#1G: pan = dark metal, eggs = room temperature, flour mixing =
long, eggs = cold, oven door opened early = yes, oven door opened early = no, oven heat =
low, problem = dry
Node bake on a higher rack#1G: pan = dark metal, eggs = room temperature, flour mixing =
long, eggs = cold, oven door opened early = yes, oven door opened early = no, oven heat =
low, problem = dry, baking time = long, problem = raw inside, problem = cracked top, oven
heat = high, problem = burnt bottom
Accepted: take it out earlier#1G: problem = dry -> advice = take it out earlier
Consult problem = dry
  Conclusion: advice = take it out earlier  (rule take it out earlier#1G, CF 100)
  take it out earlier#1G fired because:
    problem = dry  (given, CF 100)
    so advice = take it out earlier  [rule CF 100, conclusion CF 100]
== 8. Export and import
Exported 848 lines of JSON
Imported again: Cake doctor: 8 variables, 12 nodes, 12 rules
Same file: true
```

Kedua usulan rule dan keenam usulan node itu panjang; begitulah cara kerja pembangkitan VCIRS, dan karena itu manusialah yang harus memilih [4]. Usulan node *take it out earlier#1G* mendaftar delapan kondisi yang dikumpulkan dari seluruh basis pengetahuan, diakhiri `problem = dry`. Menyimpan hanya satu kondisi itu menghasilkan node tingkat teratas yang baru dan umum: kue yang kering berarti *take it out earlier*, bahkan ketika tidak ada yang tahu berapa lama kue itu dipanggang. Konsultasi berikutnya langsung memakainya. Terakhir, hasil ekspornya 848 baris JSON; mengimpornya lagi menghasilkan basis pengetahuan yang sama, dan mengekspornya menghasilkan berkas yang sama, bita demi bita.

14.5 Langkah 3: Halaman-Halaman

Halaman pembangun adalah HTML polos: judul, beberapa tabel, dan formulir dengan daftar pilihan, kotak teks, dan tombol. Tidak ada kerangka kerja JavaScript dan tidak ada berkas *stylesheet* yang harus diunduh; satu-satunya skrip di halaman mana pun adalah satu baris di halaman Beranda yang menyalin berkas yang dipilih ke kotak impor. Menjaga halaman sesederhana ini memberi tiga keuntungan bagi bab ini. Halaman termuat seketika bahkan di ponsel tua. Halaman bekerja di semua peramban. Dan halaman dapat ditulis dua kali, dalam dua bahasa, dengan bita yang sama.

Setiap halaman ditempatkan dalam bingkai `page.html`, yang bagian atasnya seperti ini:

```
<!DOCTYPE html>
<html lang="{{lang}}">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>{{title}}</title>
```

dan yang badannya berupa `<header><b>{{app}}</b><nav>{{nav}}</nav></header>` diikuti `<main>{{notice}}{{body}}</main>`, dengan sebuah lembar gaya pendek di antaranya yang memakai warna-warna buku ini. Halaman mengisi tempat-tempat `{{...}}` dengan penggantian teks biasa. Baris `viewport` memberi tahu ponsel untuk menampilkan halaman selebar layarnya sendiri alih-alih mengecilkan halaman desktop, sehingga pembangun dapat dipakai di ponsel Bu Wati.

Kata-katanya berasal dari `ui-text.json`, yang punya satu bagian untuk bahasa Inggris dan satu untuk bahasa Indonesia dengan kunci yang sama:

```
    "t_title": "Teach a case", "t_hint": "Describe a real case: choose the answers that
were true, leave the rest empty, and give the right answer. The builder puts the
knowledge where it belongs.",
```

dan, di bagian Indonesia, `"t_title": "Ajarkan sebuah kasus"` dengan petunjuknya sendiri. Sebuah halaman tidak pernah memuat kata yang diketik di dalam kode, kecuali nama-nama variabel dan nilai milik basis pengetahuan itu sendiri, yang memang milik penggunanya.

Kode 14.5 adalah modul halaman dalam Python.

Kode 14.5 `pages.py`: halaman-halaman HTML, disusun dari basis pengetahuan dan kata-kata di `ui-text.json`.

```
"""Halaman-halaman HTML pembangun. Setiap halaman adalah HTML polos dari
basis pengetahuan; kata-katanya dari ui-text.json (Inggris atau Indonesia).
Versi Java (Pages.java) menyusun bita yang persis sama.
"""

import json
import os

from vcirs import num, report

DATA = os.path.join(os.path.dirname(os.path.abspath(__file__)), "..", "data")
NAV = [("/", "nav_home"), ("/variables", "nav_variables"),
       ("/teach", "nav_teach"), ("/consult", "nav_consult"),
       ("/knowledge", "nav_knowledge"), ("/stats", "nav_stats"),
       ("/proposals", "nav_proposals")]


def esc(text):
    """Buat teks aman di dalam HTML."""
    return (str(text).replace("&", "&amp;").replace("<", "&lt;")
            .replace(">", "&gt;").replace('"', "&quot;")
            .replace("'", "&#39;"))


def url_encode(text):
    """Huruf, angka, dan - _ . tetap; spasi menjadi +; sisanya %XX."""
    out = []
    for byte in text.encode("utf-8"):
        ch = chr(byte)
        if ch.isascii() and (ch.isalnum() or ch in "-_."):
            out.append(ch)
        elif ch == " ":
            out.append("+")
        else:
            out.append("%%%02X" % byte)
    return "".join(out)
```

Kode 14.5 (Lanjutan) *pages.py: halaman-halaman HTML, disusun dari basis pengetahuan dan kata-kata di ui-text.json.*

```
def table(headers, rows):
    lines = ["<table>", "<tr>" + "".join("<th>" + esc(h) + "</th>"
                                         for h in headers) + "</tr>"]
    for row in rows:
        lines.append("<tr>" + "".join("<td>" + esc(c) + "</td>"
                                      for c in row) + "</tr>")
    lines.append("</table>")
    return "\n".join(lines)


class Pages:
    def __init__(self, lang="en"):
        with open(os.path.join(DATA, "ui-text.json"), encoding="utf-8") as f:
            self.texts = json.load(f)
        with open(os.path.join(DATA, "page.html"), encoding="utf-8") as f:
            self.frame = f.read()
        self.lang = lang

    def t(self, key):
        return self.texts[self.lang][key]

    def page(self, here, body, notice="", error=False):
        nav = []
        for href, key in NAV:
            mark = ' class="here"' if href == here else ""
            nav.append('<a href="' + href + '"' + mark + ">"
                       + esc(self.t(key)) + "</a>")
        other = "id" if self.lang == "en" else "en"
        nav.append('<a href="/lang?to=' + other + '">'
                   + esc(self.t("other_lang")) + "</a>")
        box = ""
        if notice:
            box = ('<p class="error">' + esc(self.t("error_prefix")) + " "
                   if error else '<p class="notice">') + esc(notice) + "</p>\n"
        title = self.t("app") + " - " + self.t(dict(NAV)[here])
        return (self.frame.replace("{{lang}}", self.lang)
                .replace("{{title}}", esc(title))
                .replace("{{app}}", esc(self.t("app")))
                .replace("{{nav}}", "\n".join(nav))
                .replace("{{notice}}", box).replace("{{body}}", body))

    def field(self, label, name, value=""):
        return ("<label>" + esc(label) + '</label>\n<input type="text" name="'
                + name + '" value="' + esc(value) + '">')
```

Kode 14.5 (Lanjutan) *pages.py: halaman-halaman HTML, disusun dari basis pengetahuan dan kata-kata di ui-text.json.*

```
    def fact_selects(self, b, chosen):
        """Satu daftar pilihan per pertanyaan; chosen memetakan nama -> nilai."""
        out = []
        for var in b.question_variables():
            question = b.kb.variables[var].question or var
            out.append("<label>" + esc(question) + "</label>")
            out.append('<select name="f:' + esc(var) + '">')
            out.append('<option value=""></option>')
            for value in b.kb.known_values(var):
                mark = " selected" if chosen.get(var) == value else ""
                out.append('<option value="' + esc(value) + '"' + mark + ">"
                           + esc(value) + "</option>")
            out.append("</select>")
        return "\n".join(out)

    # ----- halaman-halaman ------------------------------------------------
    def home(self, b):
        kb = b.kb
        return "\n".join([
            "<h1>" + esc(kb.name) + "</h1>",
            "<p>" + esc(kb.description) + "</p>",
            '<p class="answer">' + str(len(b.question_variables())) + " "
            + esc(self.t("questions")) + ", " + str(len(kb.nodes)) + " "
            + esc(self.t("nodes")) + ", " + str(len(kb.rules)) + " "
            + esc(self.t("rules")) + "</p>",
            "<h2>" + esc(self.t("export")) + "</h2>",
            '<p class="hint">' + esc(self.t("export_hint")) + "</p>",
            '<p><a href="/export">' + esc(self.t("download")) + "</a></p>",
            "<h2>" + esc(self.t("import")) + "</h2>",
            '<p class="hint">' + esc(self.t("import_hint")) + "</p>",
            '<form method="post" action="/import">',
            '<input type="file" accept=".json" onchange="var r=new '
            'FileReader();r.onload=function(){document.getElementById('
            "'json').value=r.result};r.readAsText(this.files[0])\">",
            '<textarea id="json" name="json" rows="5"></textarea>',
            "<br><button>" + esc(self.t("load")) + "</button>",
            "</form>",
            "<h2>" + esc(self.t("new_project")) + "</h2>",
            '<p class="hint">' + esc(self.t("new_project_hint")) + "</p>",
            '<form method="post" action="/project">',
            self.field(self.t("name"), "name"),
            self.field(self.t("description"), "description"),
            "<button>" + esc(self.t("create")) + "</button>",
            "</form>"])
```

Kode 14.5 (Lanjutan) *pages.py: halaman-halaman HTML, disusun dari basis pengetahuan dan kata-kata di ui-text.json.*

```
    def variables(self, b):
        u = b.kb.usage()
        rows = [[v.id, v.question, ", ".join(b.kb.known_values(v.id)),
                 u.ns(v.id)] for v in b.kb.variables.values()]
        return "\n".join([
            "<h1>" + esc(self.t("q_title")) + "</h1>",
            '<p class="hint">' + esc(self.t("q_hint")) + "</p>",
            table([self.t("q_name"), self.t("q_text"), self.t("q_choices"),
                   self.t("q_used")], rows),
            '<form method="post" action="/variables">',
            self.field(self.t("q_name"), "variable"),
            self.field(self.t("q_text"), "question"),
            self.field(self.t("q_choices"), "choices"),
            "<button>" + esc(self.t("save")) + "</button>",
            "</form>"])

    def teach(self, b, chosen):
        return "\n".join([
            "<h1>" + esc(self.t("t_title")) + "</h1>",
            '<p class="hint">' + esc(self.t("t_hint")) + "</p>",
            '<form method="post" action="/teach">',
            self.fact_selects(b, chosen),
            "<label>" + esc(self.t("t_extra")) + "</label>",
            '<textarea name="extra" rows="2"></textarea>',
            self.field(self.t("t_answer") + " (" + b.goal() + ")", "answer"),
            self.field(self.t("t_teacher"), "teacher"),
            "<button>" + esc(self.t("t_button")) + "</button>",
            "</form>"])

    def taught(self, before, result):
        lines = ["<h1>" + esc(self.t("t_title")) + "</h1>",
                 "<p>" + esc(self.t("t_before")) + ":</p>",
                 "<pre>" + esc(before.strip()) + "</pre>",
                 "<p>" + esc(self.t("t_now")) + ":</p>",
                 '<p class="answer">' + esc(str(result)) + "</p>"]
        if result.differences is not None:
            lines.append("<p>" + esc(self.t("t_diff")) + ":</p>")
            lines.append("<pre>" + esc("\n".join(result.differences.lines()))
                         + "</pre>")
        lines.append('<p><a href="/teach">' + esc(self.t("again")) + "</a></p>")
        return "\n".join(lines)
```

Kode 14.5 (Lanjutan) *pages.py: halaman-halaman HTML, disusun dari basis pengetahuan dan kata-kata di ui-text.json.*

```
    def consult(self, b, chosen, result=None):
        lines = ["<h1>" + esc(self.t("c_title")) + "</h1>"]
        if result is not None:
            if result.conclusions():
                lines.append("<h2>" + esc(self.t("c_answer")) + "</h2>")
                lines += ['<p class="answer">' + esc(c.value) + "</p>"
                          for c in result.conclusions()]
                lines.append("<p>" + esc(self.t("c_how")) + ":</p>")
                lines.append("<pre>" + esc("\n".join(result.how())) + "</pre>")
            else:
                lines.append('<p class="answer">' + esc(self.t("c_none"))
                             + "</p>")
            query = "&".join(url_encode("f:" + v) + "=" + url_encode(x)
                             for v, x in chosen.items())
            lines.append('<p><a href="/teach?' + esc(query) + '">'
                         + esc(self.t("c_wrong")) + "</a></p>")
        lines += ['<p class="hint">' + esc(self.t("c_hint")) + "</p>",
                  '<form method="post" action="/consult">',
                  self.fact_selects(b, chosen),
                  "<button>" + esc(self.t("c_button")) + "</button>",
                  "</form>"]
        return "\n".join(lines)

    def knowledge(self, b):
        lines = ["<h1>" + esc(self.t("k_title")) + "</h1>",
                 "<h2>" + esc(self.t("k_tree")) + "</h2>",
                 "<pre>" + esc("\n".join(report.tree(b.kb))) + "</pre>",
                 "<table>", "<tr><th>" + esc(self.t("k_node")) + "</th><th>"
                 + esc(self.t("k_when")) + "</th><th>" + esc(self.t("k_then"))
                 + "</th><th>" + esc(self.t("k_by")) + "</th><th></th></tr>"]
        for n in b.kb.nodes.values():
            if n.stopped:
                action = esc(self.t("k_stopped"))
            else:
                action = ('<form method="post" action="/stop">'
                          '<input type="hidden" name="node" value="'
                          + esc(n.id) + '"><button>' + esc(self.t("k_stop"))
                          + "</button></form>")
            lines.append("<tr><td>" + esc(n.id) + "</td><td>"
                         + esc(", ".join(str(c) for c in n.conditions))
                         + "</td><td>"
                         + esc(", ".join(str(c) for c in n.conclusions))
                         + "</td><td>" + esc(n.author) + "</td><td>" + action
                         + "</td></tr>")
        lines.append("</table>")
        lines.append("<h2>" + esc(self.t("k_history")) + "</h2>")
```

Kode 14.5 (Lanjutan) *pages.py: halaman-halaman HTML, disusun dari basis pengetahuan dan kata-kata di ui-text.json.*

```
        lines.append(table(["#", self.t("k_action"), self.t("k_node"),
                            self.t("k_option"), self.t("k_by")],
                           [[h["seq"], h["action"], h["node"], h["option"],
                             h["author"]] for h in b.kb.history]))
        return "\n".join(lines)

    def stats(self, b):
        u = b.kb.usage()
        none = self.t("s_none")
        return "\n".join([
            "<h1>" + esc(self.t("s_title")) + "</h1>",
            '<p class="hint">' + esc(self.t("s_hint")) + "</p>",
            "<h2>" + esc(self.t("s_order")) + "</h2>",
            table([self.t("s_order_col"), self.t("q_name"), self.t("s_ns")],
                  [[i + 1, v, u.ns(v)]
                   for i, v in enumerate(b.question_variables())]),
            "<p>" + esc(self.t("s_important")) + ": "
            + esc(", ".join(u.important_variables()) or none) + "</p>",
            "<p>" + esc(self.t("s_important_nodes")) + ": "
            + esc(", ".join(u.important_nodes()) or none) + "</p>",
            "<h2>" + esc(self.t("s_nur")) + "</h2>",
            table([self.t("k_node"), self.t("s_count"), self.t("s_rate")],
                  [[n, k, num.fixed(x, 3)]
                   for n, k, x in reversed(u.nur_rows())]),
            "<h2>" + esc(self.t("s_rur")) + "</h2>",
            table([self.t("s_rule"), self.t("s_rate")],
                  [[r, num.fixed(x, 3)] for r, x in reversed(u.rur_rows())]),
            "<h2>" + esc(self.t("s_vur")) + "</h2>",
            table([self.t("q_name"), self.t("k_node"), self.t("s_order_col"),
                   self.t("s_rate")],
                  [[v, n, o, num.fixed(x, 3)]
                   for v, n, o, x in reversed(u.vur_rows())])])

    def proposals(self, b):
        rules, nodes = b.kb.generate_rules(), b.kb.generate_nodes()
        lines = ["<h1>" + esc(self.t("p_title")) + "</h1>",
                 '<p class="hint">' + esc(self.t("p_hint")) + "</p>"]
        if not rules and not nodes:
            lines.append("<p>" + esc(self.t("p_none")) + "</p>")
        if rules:
            lines.append("<h2>" + esc(self.t("p_rules")) + "</h2>")
        for g in rules:
            lines += ['<form method="post" action="/accept-rule">',
                      '<input type="hidden" name="id" value="' + esc(g.id)
                      + '">', "<p><b>" + esc(g.id) + "</b>: "
                      + esc(" -> ".join(g.nodes)) + "</p>",
```

Kode 14.5 (Lanjutan) *pages.py: halaman-halaman HTML, disusun dari basis pengetahuan dan kata-kata di ui-text.json.*

```
                "<button>" + esc(self.t("p_accept")) + "</button>",
                "</form>"]
        if nodes:
            lines.append("<h2>" + esc(self.t("p_nodes")) + "</h2>")
        for g in nodes:
            lines += ['<form method="post" action="/accept-node">',
                      '<input type="hidden" name="id" value="' + esc(g.id)
                      + '">', "<p><b>" + esc(g.id) + "</b> &rarr; "
                      + esc(", ".join(str(c) for c in g.conclusions))
                      + "<br>" + esc(self.t("p_keep")) + ":</p>"]
            for c in g.conditions:
                lines.append('<label><input type="checkbox" name="keep" '
                             'value="' + esc(c.key()) + '"> ' + esc(c.key())
                             + "</label>")
            lines += ["<button>" + esc(self.t("p_accept")) + "</button>",
                      "</form>"]
        return "\n".join(lines)
```

Tiga fungsi kecil muncul lebih dulu, dan masing-masing ada karena suatu alasan.

`esc` membuat teks apa pun aman untuk dimasukkan ke dalam HTML dengan mengganti lima karakter yang punya makna dalam HTML — `&`, `<`, `>`, `"`, dan `'` — dengan bentuk lolosnya. Setiap potong teks yang berasal dari pengguna atau dari basis pengetahuan melewatinya. Tanpa fungsi ini, pertanyaan bernama `<b>eggs</b>` akan tampil tebal, dan nilai yang berniat jahat dapat menyisipkan skrip ke halaman setiap orang yang membuka pembangun: serangan yang disebut *cross-site scripting*, yang pertahanan bakunya adalah meloloskan setiap keluaran [7].

`url_encode` menuliskan jawaban-jawaban yang dipilih ke dalam tautan “Kurang tepat? Ajarkan jawaban yang benar untuk kasus ini” di halaman Konsultasi, sehingga halaman Ajari terbuka dengan jawaban yang sama sudah terpilih. Fungsi ini mengodekan setiap bita kecuali huruf, angka, dan `- _ .`, serta spasi sebagai `+`. Java dan Python punya pengode URL masing-masing, dan keduanya berbeda dalam hal-hal kecil (misalnya apakah `~` atau `*` dikodekan); menulis beberapa baris sendiri membuat kedua bahasa menghasilkan tautan yang sama.

`table` mengubah kepala tabel dan baris-baris menjadi tabel HTML, sambil meloloskan setiap sel.

Kelas `Pages` kemudian memegang bahasa dan teks-teks yang sudah dimuat. `page` menaruh badan halaman ke dalam bingkai: ia menyusun navigasi, menandai halaman yang sedang dibuka, menambahkan tautan ke bahasa lain, dan menampilkan pesan dari perubahan terakhir — hijau untuk “Tersimpan.”, merah dengan “Maaf:” untuk galat. `fact_selects` menyusun satu daftar pilihan per pertanyaan, dalam urutan `question_variables`, dengan pilihan pertama yang kosong yang berarti “saya tidak tahu”; jawaban yang dipilih tetap terpilih ketika halaman ditampilkan lagi.

Metode-metode halaman mengikuti **Tabel 14.1**. `home` menampilkan nama, keterangan, dan ukuran proyek, tautan ekspor, formulir impor (pemilih berkas yang satu baris skripnya menyalin teks berkas ke kotak teks, dan tombol **Muat**), serta formulir proyek baru. `variables` mendaftar pertanyaan beserta jawaban-jawaban biasanya dan berapa node yang memakai masing-masing, serta menawarkan formulir untuk pertanyaan baru. `teach` adalah formulir kasus: pertanyaan-pertanyaan, kotak untuk fakta lain yang diketik sebagai `nama = nilai`, jawaban yang tepat, dan nama pengajarnya. `taught` menampilkan apa yang terjadi, dan daftar perbedaan ketika kasusnya menjadi pengecualian. `consult` menampilkan saran dan penjelasannya di atas formulir, serta tautan yang mengirim kasusnya ke halaman Ajari. `knowledge` menampilkan pohon, tabel node dengan tombol **Hentikan** untuk setiap node yang masih aktif, dan riwayatnya. `stats`

menampilkan urutan pertanyaan dengan *node sharing*-nya, pertanyaan dan node terpenting, serta tabel NUR, RUR, dan VUR, dari yang terbesar. `proposals` menampilkan setiap usulan rule dengan tombol **Terima** dan setiap usulan node dengan kotak centang untuk setiap kondisinya.

Kode 14.6 adalah modul yang sama dalam Java.

Kode 14.6 `Pages.java`: halaman-halaman yang sama dalam Java, bita demi bita.

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Map;

import vcirs.Condition;
import vcirs.Conclusion;
import vcirs.GeneratedNode;
import vcirs.GeneratedRule;
import vcirs.InferenceResult;
import vcirs.Json;
import vcirs.LearnResult;
import vcirs.Node;
import vcirs.Num;
import vcirs.Report;
import vcirs.Usage;
import vcirs.Variable;

/**
 * Halaman-halaman HTML pembangun. Setiap halaman adalah HTML polos dari
 * basis pengetahuan; kata-katanya dari ui-text.json (Inggris atau
 * Indonesia). Versi Python (pages.py) menyusun bita yang persis sama.
 */
public class Pages {
    static final String DATA = "../data/";
    static final String[][] NAV = {{"/", "nav_home"},
        {"/variables", "nav_variables"}, {"/teach", "nav_teach"},
        {"/consult", "nav_consult"}, {"/knowledge", "nav_knowledge"},
        {"/stats", "nav_stats"}, {"/proposals", "nav_proposals"}};

    /** Buat teks aman di dalam HTML. */
    static String esc(Object text) {
        return String.valueOf(text).replace("&", "&amp;").replace("<", "&lt;")
                .replace(">", "&gt;").replace("\"", "&quot;")
                .replace("'", "&#39;");
    }
```

Kode 14.6 (Lanjutan) *Pages.java: halaman-halaman yang sama dalam Java, bita demi bita.*

```
    /** Huruf, angka, dan - _ . tetap; spasi menjadi +; sisanya %XX. */
    static String urlEncode(String text) {
        StringBuilder out = new StringBuilder();
        for (byte b : text.getBytes(StandardCharsets.UTF_8)) {
            char ch = (char) (b & 0xff);
            boolean plain = Character.isLetterOrDigit(ch)
                    || "-_.".indexOf(ch) >= 0;
            if (ch < 128 && plain) {
                out.append(ch);
            } else if (ch == ' ') {
                out.append('+');
            } else {
                out.append(String.format("%%%02X", b & 0xff));
            }
        }
        return out.toString();
    }

    static String table(List<String> headers, List<List<Object>> rows) {
        List<String> lines = new ArrayList<>();
        lines.add("<table>");
        StringBuilder head = new StringBuilder("<tr>");
        headers.forEach(h -> head.append("<th>").append(esc(h)).append("</th>"));
        lines.add(head + "</tr>");
        for (List<Object> row : rows) {
            StringBuilder r = new StringBuilder("<tr>");
            row.forEach(c -> r.append("<td>").append(esc(c)).append("</td>"));
            lines.add(r + "</tr>");
        }
        lines.add("</table>");
        return String.join("\n", lines);
    }

    final Map<String, Map<String, String>> texts;
    final String frame;
    String lang;
```

Kode 14.6 (Lanjutan) *Pages.java: halaman-halaman yang sama dalam Java, bita demi bita.*

```
    @SuppressWarnings("unchecked")
    Pages(String lang) throws IOException {
        texts = (Map<String, Map<String, String>>) (Object) Json.parse(
                Files.readString(Path.of(DATA + "ui-text.json"),
                        StandardCharsets.UTF_8));
        frame = Files.readString(Path.of(DATA + "page.html"),
                StandardCharsets.UTF_8);
        this.lang = lang;
    }

    String t(String key) {
        return texts.get(lang).get(key);
    }

    String page(String here, String body, String notice, boolean error) {
        List<String> nav = new ArrayList<>();
        String title = "";
        for (String[] item : NAV) {
            String mark = item[0].equals(here) ? " class=\"here\"" : "";
            nav.add("<a href=\"" + item[0] + "\"" + mark + ">" + esc(t(item[1]))
                    + "</a>");
            if (item[0].equals(here)) {
                title = t("app") + " - " + t(item[1]);
            }
        }
        String other = lang.equals("en") ? "id" : "en";
        nav.add("<a href=\"/lang?to=" + other + "\">" + esc(t("other_lang"))
                + "</a>");
        String box = "";
        if (!notice.isEmpty()) {
            box = (error ? "<p class=\"error\">" + esc(t("error_prefix")) + " "
                    : "<p class=\"notice\">") + esc(notice) + "</p>\n";
        }
        return frame.replace("{{lang}}", lang).replace("{{title}}", esc(title))
                .replace("{{app}}", esc(t("app")))
                .replace("{{nav}}", String.join("\n", nav))
                .replace("{{notice}}", box).replace("{{body}}", body);
    }

    String field(String label, String name, String value) {
        return "<label>" + esc(label) + "</label>\n<input type=\"text\" name=\""
                + name + "\" value=\"" + esc(value) + "\">";
    }
```

Kode 14.6 (Lanjutan) *Pages.java: halaman-halaman yang sama dalam Java, bita demi bita.*

```
    /** Satu daftar pilihan per pertanyaan; chosen memetakan nama -> nilai. */
    String factSelects(Builder b, Map<String, String> chosen) {
        List<String> out = new ArrayList<>();
        for (String var : b.questionVariables()) {
            String q = b.kb.variables.get(var).question;
            out.add("<label>" + esc(q.isEmpty() ? var : q) + "</label>");
            out.add("<select name=\"f:" + esc(var) + "\">");
            out.add("<option value=\"\"></option>");
            for (String value : b.kb.knownValues(var)) {
                String mark = value.equals(chosen.get(var)) ? " selected" : "";
                out.add("<option value=\"" + esc(value) + "\"" + mark + ">"
                        + esc(value) + "</option>");
            }
            out.add("</select>");
        }
        return String.join("\n", out);
    }

    static String joined(List<?> items, String sep) {
        List<String> parts = new ArrayList<>();
        items.forEach(x -> parts.add(String.valueOf(x)));
        return String.join(sep, parts);
    }

    // ----- halaman-halaman ------------------------------------------------
    String home(Builder b) {
        return String.join("\n", List.of(
            "<h1>" + esc(b.kb.name) + "</h1>",
            "<p>" + esc(b.kb.description) + "</p>",
            "<p class=\"answer\">" + b.questionVariables().size() + " "
            + esc(t("questions")) + ", " + b.kb.nodes.size() + " "
            + esc(t("nodes")) + ", " + b.kb.rules.size() + " "
            + esc(t("rules")) + "</p>",
            "<h2>" + esc(t("export")) + "</h2>",
            "<p class=\"hint\">" + esc(t("export_hint")) + "</p>",
            "<p><a href=\"/export\">" + esc(t("download")) + "</a></p>",
            "<h2>" + esc(t("import")) + "</h2>",
            "<p class=\"hint\">" + esc(t("import_hint")) + "</p>",
            "<form method=\"post\" action=\"/import\">",
            "<input type=\"file\" accept=\".json\" onchange=\"var r=new "
            + "FileReader();r.onload=function(){document.getElementById("
            + "'json').value=r.result};r.readAsText(this.files[0])\">",
            "<textarea id=\"json\" name=\"json\" rows=\"5\"></textarea>",
            "<br><button>" + esc(t("load")) + "</button>",
            "</form>",
            "<h2>" + esc(t("new_project")) + "</h2>",
```

Kode 14.6 (Lanjutan) *Pages.java: halaman-halaman yang sama dalam Java, bita demi bita.*

```
            "<p class=\"hint\">" + esc(t("new_project_hint")) + "</p>",
            "<form method=\"post\" action=\"/project\">",
            field(t("name"), "name", ""),
            field(t("description"), "description", ""),
            "<button>" + esc(t("create")) + "</button>",
            "</form>"));
    }

    String variables(Builder b) {
        Usage u = b.kb.usage();
        List<List<Object>> rows = new ArrayList<>();
        for (Variable v : b.kb.variables.values()) {
            rows.add(List.of(v.id, v.question,
                    String.join(", ", b.kb.knownValues(v.id)), u.ns(v.id)));
        }
        return String.join("\n", List.of(
            "<h1>" + esc(t("q_title")) + "</h1>",
            "<p class=\"hint\">" + esc(t("q_hint")) + "</p>",
            table(List.of(t("q_name"), t("q_text"), t("q_choices"), t("q_used")),
                    rows),
            "<form method=\"post\" action=\"/variables\">",
            field(t("q_name"), "variable", ""),
            field(t("q_text"), "question", ""),
            field(t("q_choices"), "choices", ""),
            "<button>" + esc(t("save")) + "</button>",
            "</form>"));
    }

    String teach(Builder b, Map<String, String> chosen) {
        return String.join("\n", List.of(
            "<h1>" + esc(t("t_title")) + "</h1>",
            "<p class=\"hint\">" + esc(t("t_hint")) + "</p>",
            "<form method=\"post\" action=\"/teach\">",
            factSelects(b, chosen),
            "<label>" + esc(t("t_extra")) + "</label>",
            "<textarea name=\"extra\" rows=\"2\"></textarea>",
            field(t("t_answer") + " (" + b.goal() + ")", "answer", ""),
            field(t("t_teacher"), "teacher", ""),
            "<button>" + esc(t("t_button")) + "</button>",
            "</form>"));
    }
```

Kode 14.6 (Lanjutan) *Pages.java: halaman-halaman yang sama dalam Java, bita demi bita.*

```
    String taught(String before, LearnResult result) {
        List<String> lines = new ArrayList<>(List.of(
            "<h1>" + esc(t("t_title")) + "</h1>",
            "<p>" + esc(t("t_before")) + ":</p>",
            "<pre>" + esc(before.strip()) + "</pre>",
            "<p>" + esc(t("t_now")) + ":</p>",
            "<p class=\"answer\">" + esc(result) + "</p>"));
        if (result.differences != null) {
            lines.add("<p>" + esc(t("t_diff")) + ":</p>");
            lines.add("<pre>" + esc(String.join("\n", result.differences.lines()))
                    + "</pre>");
        }
        lines.add("<p><a href=\"/teach\">" + esc(t("again")) + "</a></p>");
        return String.join("\n", lines);
    }

    String consult(Builder b, Map<String, String> chosen, InferenceResult result) {
        List<String> lines = new ArrayList<>();
        lines.add("<h1>" + esc(t("c_title")) + "</h1>");
        if (result != null) {
            if (!result.conclusions().isEmpty()) {
                lines.add("<h2>" + esc(t("c_answer")) + "</h2>");
                for (Conclusion c : result.conclusions()) {
                    lines.add("<p class=\"answer\">" + esc(c.value) + "</p>");
                }
                lines.add("<p>" + esc(t("c_how")) + ":</p>");
                lines.add("<pre>" + esc(String.join("\n", result.how()))
                        + "</pre>");
            } else {
                lines.add("<p class=\"answer\">" + esc(t("c_none")) + "</p>");
            }
            List<String> query = new ArrayList<>();
            chosen.forEach((v, x) -> query.add(urlEncode("f:" + v) + "="
                    + urlEncode(x)));
            lines.add("<p><a href=\"/teach?" + esc(String.join("&", query))
                    + "\">" + esc(t("c_wrong")) + "</a></p>");
        }
        lines.add("<p class=\"hint\">" + esc(t("c_hint")) + "</p>");
        lines.add("<form method=\"post\" action=\"/consult\">");
        lines.add(factSelects(b, chosen));
        lines.add("<button>" + esc(t("c_button")) + "</button>");
        lines.add("</form>");
        return String.join("\n", lines);
    }
```

Kode 14.6 (Lanjutan) *Pages.java: halaman-halaman yang sama dalam Java, bita demi bita.*

```
    String knowledge(Builder b) {
        List<String> lines = new ArrayList<>(List.of(
            "<h1>" + esc(t("k_title")) + "</h1>",
            "<h2>" + esc(t("k_tree")) + "</h2>",
            "<pre>" + esc(String.join("\n", Report.tree(b.kb))) + "</pre>",
            "<table>", "<tr><th>" + esc(t("k_node")) + "</th><th>"
            + esc(t("k_when")) + "</th><th>" + esc(t("k_then"))
            + "</th><th>" + esc(t("k_by")) + "</th><th></th></tr>"));
        for (Node n : b.kb.nodes.values()) {
            String action = n.stopped ? esc(t("k_stopped"))
                    : "<form method=\"post\" action=\"/stop\">"
                    + "<input type=\"hidden\" name=\"node\" value=\""
                    + esc(n.id) + "\"><button>" + esc(t("k_stop"))
                    + "</button></form>";
            lines.add("<tr><td>" + esc(n.id) + "</td><td>"
                    + esc(joined(n.conditions, ", ")) + "</td><td>"
                    + esc(joined(n.conclusions, ", ")) + "</td><td>"
                    + esc(n.author) + "</td><td>" + action + "</td></tr>");
        }
        lines.add("</table>");
        lines.add("<h2>" + esc(t("k_history")) + "</h2>");
        List<List<Object>> rows = new ArrayList<>();
        for (Map<String, Object> h : b.kb.history) {
            rows.add(List.of(h.get("seq"), h.get("action"), h.get("node"),
                    h.get("option"), h.get("author")));
        }
        lines.add(table(List.of("#", t("k_action"), t("k_node"), t("k_option"),
                t("k_by")), rows));
        return String.join("\n", lines);
    }

    String stats(Builder b) {
        Usage u = b.kb.usage();
        String none = t("s_none");
        List<List<Object>> order = new ArrayList<>();
        List<String> vars = b.questionVariables();
        for (int i = 0; i < vars.size(); i++) {
            order.add(List.of(i + 1, vars.get(i), u.ns(vars.get(i))));
        }
        List<List<Object>> nur = new ArrayList<>();
        for (Usage.RateRow r : u.nurRows()) {
            nur.add(List.of(r.id, r.size, Num.fixed(r.rate, 3)));
        }
        List<List<Object>> rur = new ArrayList<>();
        for (Usage.RateRow r : u.rurRows()) {
            rur.add(List.of(r.id, Num.fixed(r.rate, 3)));
```

Kode 14.6 (Lanjutan) *Pages.java: halaman-halaman yang sama dalam Java, bita demi bita.*

```
        }
        List<List<Object>> vur = new ArrayList<>();
        for (Usage.VurRow r : u.vurRows()) {
            vur.add(List.of(r.variable, r.node, r.order, Num.fixed(r.vur, 3)));
        }
        Collections.reverse(nur);
        Collections.reverse(rur);
        Collections.reverse(vur);
        String important = String.join(", ", u.importantVariables());
        String importantNodes = String.join(", ", u.importantNodes());
        return String.join("\n", List.of(
            "<h1>" + esc(t("s_title")) + "</h1>",
            "<p class=\"hint\">" + esc(t("s_hint")) + "</p>",
            "<h2>" + esc(t("s_order")) + "</h2>",
            table(List.of(t("s_order_col"), t("q_name"), t("s_ns")), order),
            "<p>" + esc(t("s_important")) + ": "
            + esc(important.isEmpty() ? none : important) + "</p>",
            "<p>" + esc(t("s_important_nodes")) + ": "
            + esc(importantNodes.isEmpty() ? none : importantNodes) + "</p>",
            "<h2>" + esc(t("s_nur")) + "</h2>",
            table(List.of(t("k_node"), t("s_count"), t("s_rate")), nur),
            "<h2>" + esc(t("s_rur")) + "</h2>",
            table(List.of(t("s_rule"), t("s_rate")), rur),
            "<h2>" + esc(t("s_vur")) + "</h2>",
            table(List.of(t("q_name"), t("k_node"), t("s_order_col"),
                    t("s_rate")), vur)));
    }

    String proposals(Builder b) {
        List<GeneratedRule> rules = b.kb.generateRules();
        List<GeneratedNode> nodes = b.kb.generateNodes();
        List<String> lines = new ArrayList<>();
        lines.add("<h1>" + esc(t("p_title")) + "</h1>");
        lines.add("<p class=\"hint\">" + esc(t("p_hint")) + "</p>");
        if (rules.isEmpty() && nodes.isEmpty()) {
            lines.add("<p>" + esc(t("p_none")) + "</p>");
        }
        if (!rules.isEmpty()) {
            lines.add("<h2>" + esc(t("p_rules")) + "</h2>");
        }
        for (GeneratedRule g : rules) {
            lines.add("<form method=\"post\" action=\"/accept-rule\">");
            lines.add("<input type=\"hidden\" name=\"id\" value=\"" + esc(g.id)
                    + "\">");
            lines.add("<p><b>" + esc(g.id) + "</b>: "
                    + esc(String.join(" -> ", g.nodes)) + "</p>");
```

Kode 14.6 (Lanjutan) *Pages.java: halaman-halaman yang sama dalam Java, bita demi bita.*

```
            lines.add("<button>" + esc(t("p_accept")) + "</button>");
            lines.add("</form>");
        }
        if (!nodes.isEmpty()) {
            lines.add("<h2>" + esc(t("p_nodes")) + "</h2>");
        }
        for (GeneratedNode g : nodes) {
            lines.add("<form method=\"post\" action=\"/accept-node\">");
            lines.add("<input type=\"hidden\" name=\"id\" value=\"" + esc(g.id)
                    + "\">");
            lines.add("<p><b>" + esc(g.id) + "</b> &rarr; "
                    + esc(joined(g.conclusions, ", ")) + "<br>"
                    + esc(t("p_keep")) + ":</p>");
            for (Condition c : g.conditions) {
                lines.add("<label><input type=\"checkbox\" name=\"keep\" value=\""
                        + esc(c.key()) + "\"> " + esc(c.key()) + "</label>");
            }
            lines.add("<button>" + esc(t("p_accept")) + "</button>");
            lines.add("</form>");
        }
        return String.join("\n", lines);
    }
}
```

Menulis dua program yang menghasilkan bita yang sama menuntut ketelitian di tempat-tempat kecil, dan halaman-halaman ini memperlihatkan sebagian besar di antaranya. Kedua bahasa menyusun setiap halaman sebagai daftar baris yang digabung dengan `\n`. Keduanya meloloskan karakter dengan lima penggantian yang sama dalam urutan yang sama (`&` lebih dulu, agar `&` dari `<` tidak diloloskan lagi). Bilangan ditulis dengan `num.fixed` / `Num.fixed`, pemformat bersama milik mesin, tidak pernah dengan pemformat bawaan bahasa. Tabel dibalik dengan `reversed(...)` di Python dan `Collections.reverse` di Java, dari daftar yang diurutkan mesin dengan cara yang sama di keduanya. Dan *map* menyimpan urutan penambahan kuncinya: `dict` Python melakukannya menurut definisi, dan di Java kita memakai `LinkedHashMap` di mana-mana.

14.6 Langkah 4: Server-Server Web

Server adalah bagian yang paling tipis. **Kode 14.7** adalah server Python dengan **Flask**, kerangka kerja web kecil yang banyak dipakai untuk Python; versi terbarunya, 3.1.3, dirilis pada Februari 2026 [8].

Kode 14.7 `builder_app.py`: server web pembangun dengan Flask.

```
"""Pembangun sistem pakar VCIRS sebagai aplikasi web kecil (Flask).

    python builder_app.py [knowledge-base.json] [--port 4141]

Lalu buka http://localhost:4141 di peramban. Semua isi halaman berasal
dari builder.py dan pages.py; berkas ini hanya menghubungkannya ke web.
"""

import os
import sys

from flask import Flask, Response, redirect, request

from builder import Builder
from pages import DATA, Pages
from vcirs import VcirsError

app = Flask(__name__)
state = {"builder": None, "pages": Pages("en"), "notice": "", "error": False}


def show(here, body):
    """Satu halaman utuh, dengan pesan dari perubahan terakhir (sekali)."""
    html = state["pages"].page(here, body, state["notice"], state["error"])
    state["notice"], state["error"] = "", False
    return Response(html, content_type="text/html; charset=utf-8")


def done(where, error=None):
    """Setelah perubahan: ingat pesan untuk pengguna, lalu lanjutkan."""
    state["notice"] = str(error) if error else state["pages"].t("saved")
    state["error"] = error is not None
    return redirect(where, code=303)


def chosen_facts(values):
    """Kolom f:nama yang diisi, sebagai dict nama -> nilai."""
    return {k[2:]: v.strip() for k, v in values.items()
            if k.startswith("f:") and v.strip()}
```

Kode 14.7 (Lanjutan) *builder_app.py: server web pembangun dengan Flask.*

```
def extra_facts(text):
    """Baris 'nama = nilai' yang diketik sendiri."""
    out = []
    for line in text.splitlines():
        if "=" in line:
            var, value = line.split("=", 1)
            if var.strip() and value.strip():
                out.append((var.strip(), value.strip()))
    return out


@app.get("/")
def home():
    return show("/", state["pages"].home(state["builder"]))


@app.get("/variables")
def variables():
    return show("/variables", state["pages"].variables(state["builder"]))


@app.get("/teach")
def teach_form():
    b = state["builder"]
    return show("/teach", state["pages"].teach(b, chosen_facts(request.args)))


@app.get("/consult")
def consult_form():
    return show("/consult", state["pages"].consult(state["builder"], {}))


@app.get("/knowledge")
def knowledge():
    return show("/knowledge", state["pages"].knowledge(state["builder"]))


@app.get("/stats")
def stats():
    return show("/stats", state["pages"].stats(state["builder"]))


@app.get("/proposals")
def proposals():
    return show("/proposals", state["pages"].proposals(state["builder"]))
```

Kode 14.7 (Lanjutan) *builder_app.py: server web pembangun dengan Flask.*

```
@app.get("/export")
def export():
    return Response(state["builder"].export_json(),
                    content_type="application/json; charset=utf-8",
                    headers={"Content-Disposition":
                             'attachment; filename="knowledge-base.json"'})


@app.get("/lang")
def language():
    if request.args.get("to") in ("en", "id"):
        state["pages"].lang = request.args["to"]
    return redirect("/", code=303)


@app.post("/project")
def project():
    try:
        state["builder"].new_project(request.form.get("name", ""),
                                     request.form.get("description", ""))
    except VcirsError as e:
        return done("/", e)
    return done("/")


@app.post("/import")
def import_json():
    try:
        state["builder"].import_json(request.form.get("json", ""))
    except VcirsError as e:
        return done("/", e)
    return done("/")


@app.post("/variables")
def declare():
    f = request.form
    try:
        state["builder"].declare(f.get("variable", ""), f.get("question", ""),
                                 f.get("choices", ""))
    except VcirsError as e:
        return done("/variables", e)
    return done("/variables")
```

Kode 14.7 (Lanjutan) *builder_app.py: server web pembangun dengan Flask.*

```
@app.post("/teach")
def teach():
    b, f = state["builder"], request.form
    pairs = list(chosen_facts(f).items()) + extra_facts(f.get("extra", ""))
    try:
        before, result = b.teach(pairs, f.get("answer", ""),
                                 f.get("teacher", ""))
    except VcirsError as e:
        state["notice"], state["error"] = str(e), True
        return show("/teach", state["pages"].teach(b, chosen_facts(f)))
    return show("/teach", state["pages"].taught(before, result))


@app.post("/consult")
def consult():
    b = state["builder"]
    chosen = chosen_facts(request.form)
    result = b.consult(list(chosen.items()))
    return show("/consult", state["pages"].consult(b, chosen, result))


@app.post("/stop")
def stop():
    try:
        state["builder"].stop(request.form.get("node", ""), "")
    except VcirsError as e:
        return done("/knowledge", e)
    return done("/knowledge")


@app.post("/accept-rule")
def accept_rule():
    try:
        state["builder"].accept_rule(request.form.get("id", ""), "")
    except VcirsError as e:
        return done("/proposals", e)
    return done("/proposals")
```

Kode 14.7 (Lanjutan) *builder_app.py: server web pembangun dengan Flask.*

```
@app.post("/accept-node")
def accept_node():
    f = request.form
    try:
        state["builder"].accept_node(f.get("id", ""), f.getlist("keep"), "")
    except VcirsError as e:
        return done("/proposals", e)
    return done("/proposals")


def main(args):
    port = int(args[args.index("--port") + 1]) if "--port" in args else 4141
    files = [a for a in args if a.endswith(".json")]
    state["builder"] = Builder(files[0] if files
                               else os.path.join(DATA, "builder-kb.json"))
    print("VCIRS Builder: open http://localhost:" + str(port)
          + " in your browser; Ctrl+C stops it.", flush=True)
    app.run(host="127.0.0.1", port=port, threaded=False)  # satu per satu


if __name__ == "__main__":
    main(sys.argv[1:])
```

Aplikasi Flask adalah sekumpulan fungsi, masing-masing ditandai dengan alamat dan metode yang dijawabnya: `@app.get("/stats")` menjawab permintaan peramban untuk halaman Statistik, `@app.post("/teach")` menjawab formulir Ajari. `request.form` memuat kolom-kolom formulir yang dikirim, `request.args` kolom-kolom *query string* sebuah tautan; `getlist("keep")` mengembalikan setiap kotak centang bernama `keep` yang dicentang.

Server menyimpan seluruh keadaannya dalam satu kamus: pembangun, halaman (dengan bahasa yang dipilih), dan sebuah *notice* — pesan yang ditampilkan sekali di halaman berikutnya. Dua fungsi pembantu mengerjakan semua komunikasi. `show` membungkus badan halaman dengan bingkai, beserta pesannya, lalu menghapus pesan itu agar hanya tampil sekali. `done` dipanggil setelah perubahan: ia mengisi pesan dengan "Tersimpan." atau dengan kalimat galatnya dan **mengalihkan** peramban, dengan status 303, ke halaman yang seharusnya dilihat berikutnya. Pola ini — *post, redirect, get* — berarti setelah menyimpan pertanyaan, peramban menampilkan halaman Pertanyaan melalui `GET` yang baru; bila Bu Wati menekan tombol muat ulang peramban, halaman ditampilkan lagi, tetapi pertanyaan tidak tersimpan dua kali. Formulir Ajari dan Konsultasi berbeda: hasilnya *adalah* sebuah halaman (apa yang terjadi, atau sarannya), jadi hasil itu langsung ditampilkan.

`chosen_facts` mengambil kolom-kolom yang namanya diawali `f:` — daftar-daftar pilihan — dan menyimpan yang tidak kosong; `extra_facts` membaca kotak fakta lain, satu `nama = nilai` per baris. `main` membaca nama berkas dan port dari baris perintah, membuka pembangun, dan menjalankan Flask di `127.0.0.1`, yang berarti *hanya komputer ini*.

Kode 14.8 adalah server Java dengan `com.sun.net.httpserver`, server web kecil yang sudah bertahun-tahun menjadi bagian JDK dan didokumentasikan bersama API JDK 25 [9].

Kode 14.8 `BuilderServer.java`: server yang sama dengan HttpServer bawaan JDK.

```
import java.io.IOException;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.net.URLDecoder;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpServer;

import vcirs.InferenceResult;
import vcirs.VcirsException;

/**
 * Pembangun sistem pakar VCIRS sebagai aplikasi web kecil (server HTTP
 * bawaan JDK). java BuilderServer [knowledge-base.json] [--port 4141]
 * Lalu buka http://localhost:4141 di peramban. Semua isi halaman berasal
 * dari Builder dan Pages; kelas ini hanya menghubungkannya ke web.
 */
public class BuilderServer {
    static Builder builder;
    static Pages pages;
    static String notice = "";
    static boolean error = false;

    /** "a=1&b=x+y" -> {a: [1], b: [x y]}, sesuai urutan. */
    static Map<String, List<String>> form(String text) {
        Map<String, List<String>> out = new LinkedHashMap<>();
        if (text == null || text.isEmpty()) {
            return out;
        }
        for (String part : text.split("&")) {
            int eq = part.indexOf('=');
            String k = eq < 0 ? part : part.substring(0, eq);
            String v = eq < 0 ? "" : part.substring(eq + 1);
            out.computeIfAbsent(URLDecoder.decode(k, StandardCharsets.UTF_8),
                    x -> new ArrayList<>())
                    .add(URLDecoder.decode(v, StandardCharsets.UTF_8));
        }
        return out;
    }
```

Kode 14.8 (Lanjutan) *BuilderServer.java: server yang sama dengan HttpServer bawaan JDK.*

```
    static String get(Map<String, List<String>> f, String key) {
        return f.containsKey(key) ? f.get(key).get(0) : "";
    }

    /** Kolom f:nama yang diisi, sebagai nama -> nilai. */
    static Map<String, String> chosenFacts(Map<String, List<String>> f) {
        Map<String, String> out = new LinkedHashMap<>();
        f.forEach((k, v) -> {
            if (k.startsWith("f:") && !v.get(0).strip().isEmpty()) {
                out.put(k.substring(2), v.get(0).strip());
            }
        });
        return out;
    }

    /** Baris "nama = nilai" yang diketik sendiri. */
    static List<String[]> extraFacts(String text) {
        List<String[]> out = new ArrayList<>();
        for (String line : text.split("\\R")) {
            int eq = line.indexOf('=');
            if (eq > 0 && !line.substring(0, eq).strip().isEmpty()
                    && !line.substring(eq + 1).strip().isEmpty()) {
                out.add(new String[] {line.substring(0, eq).strip(),
                    line.substring(eq + 1).strip()});
            }
        }
        return out;
    }

    static void send(HttpExchange ex, int status, String type, String body,
                     String extraHeader, String extraValue) throws IOException {
        byte[] bytes = body.getBytes(StandardCharsets.UTF_8);
        ex.getResponseHeaders().set("Content-Type", type);
        if (extraHeader != null) {
            ex.getResponseHeaders().set(extraHeader, extraValue);
        }
        ex.sendResponseHeaders(status, bytes.length == 0 ? -1 : bytes.length);
        try (OutputStream out = ex.getResponseBody()) {
            out.write(bytes);
        }
    }
```

Kode 14.8 (Lanjutan) *BuilderServer.java: server yang sama dengan HttpServer bawaan JDK.*

```
    /** Satu halaman utuh, dengan pesan dari perubahan terakhir (sekali). */
    static void show(HttpExchange ex, String here, String body)
            throws IOException {
        String html = pages.page(here, body, notice, error);
        notice = "";
        error = false;
        send(ex, 200, "text/html; charset=utf-8", html, null, null);
    }

    /** Setelah perubahan: ingat pesan untuk pengguna, lalu lanjutkan. */
    static void done(HttpExchange ex, String where, VcirsException e)
            throws IOException {
        notice = e == null ? pages.t("saved") : e.getMessage();
        error = e != null;
        send(ex, 303, "text/html; charset=utf-8", "", "Location", where);
    }

    static synchronized void handle(HttpExchange ex) throws IOException {
        String path = ex.getRequestURI().getPath();
        boolean post = ex.getRequestMethod().equals("POST");
        Map<String, List<String>> f = form(post
                ? new String(ex.getRequestBody().readAllBytes(),
                        StandardCharsets.UTF_8)
                : ex.getRequestURI().getRawQuery());
        Builder b = builder;
        try {
            switch ((post ? "POST " : "GET ") + path) {
                case "GET /" -> show(ex, "/", pages.home(b));
                case "GET /variables" -> show(ex, path, pages.variables(b));
                case "GET /teach" -> show(ex, path, pages.teach(b, chosenFacts(f)));
                case "GET /consult" -> show(ex, path, pages.consult(b,
                        new LinkedHashMap<>(), null));
                case "GET /knowledge" -> show(ex, path, pages.knowledge(b));
                case "GET /stats" -> show(ex, path, pages.stats(b));
                case "GET /proposals" -> show(ex, path, pages.proposals(b));
                case "GET /export" -> send(ex, 200,
                        "application/json; charset=utf-8", b.exportJson(),
                        "Content-Disposition",
                        "attachment; filename=\"knowledge-base.json\"");
                case "GET /lang" -> {
                    String to = get(f, "to");
                    if (to.equals("en") || to.equals("id")) {
                        pages.lang = to;
                    }
                    send(ex, 303, "text/html", "", "Location", "/");
                }
```

Kode 14.8 (Lanjutan) *BuilderServer.java: server yang sama dengan HttpServer bawaan JDK.*

```
                case "POST /project" -> {
                    b.newProject(get(f, "name"), get(f, "description"));
                    done(ex, "/", null);
                }
                case "POST /import" -> {
                    b.importJson(get(f, "json"));
                    done(ex, "/", null);
                }
                case "POST /variables" -> {
                    b.declare(get(f, "variable"), get(f, "question"),
                            get(f, "choices"));
                    done(ex, "/variables", null);
                }
                case "POST /teach" -> teach(ex, b, f);
                case "POST /consult" -> {
                    Map<String, String> chosen = chosenFacts(f);
                    List<String[]> pairs = new ArrayList<>();
                    chosen.forEach((k, v) -> pairs.add(new String[] {k, v}));
                    InferenceResult result = b.consult(pairs);
                    show(ex, "/consult", pages.consult(b, chosen, result));
                }
                case "POST /stop" -> {
                    b.stop(get(f, "node"), "");
                    done(ex, "/knowledge", null);
                }
                case "POST /accept-rule" -> {
                    b.acceptRule(get(f, "id"), "");
                    done(ex, "/proposals", null);
                }
                case "POST /accept-node" -> {
                    b.acceptNode(get(f, "id"),
                            f.getOrDefault("keep", List.of()), "");
                    done(ex, "/proposals", null);
                }
                default -> send(ex, 404, "text/plain; charset=utf-8",
                        "Not found", null, null);
            }
        } catch (VcirsException e) {
            String back = path.equals("/project") || path.equals("/import")
                    ? "/" : path.startsWith("/accept") ? "/proposals"
                    : path.equals("/stop") ? "/knowledge" : path;
            done(ex, back, e);
        }
    }
```

Kode 14.8 (Lanjutan) *BuilderServer.java: server yang sama dengan HttpServer bawaan JDK.*

```
    static void teach(HttpExchange ex, Builder b, Map<String, List<String>> f)
            throws IOException {
        List<String[]> pairs = new ArrayList<>();
        chosenFacts(f).forEach((k, v) -> pairs.add(new String[] {k, v}));
        pairs.addAll(extraFacts(get(f, "extra")));
        Builder.Taught t;
        try {
            t = b.teach(pairs, get(f, "answer"), get(f, "teacher"));
        } catch (VcirsException e) {
            notice = e.getMessage();
            error = true;
            show(ex, "/teach", pages.teach(b, chosenFacts(f)));
            return;
        }
        show(ex, "/teach", pages.taught(t.before(), t.result()));
    }

    public static void main(String[] args) throws IOException {
        int port = 4141;
        String path = "../data/builder-kb.json";
        for (int i = 0; i < args.length; i++) {
            if (args[i].equals("--port") && i + 1 < args.length) {
                port = Integer.parseInt(args[++i]);
            } else if (args[i].endsWith(".json")) {
                path = args[i];
            }
        }
        builder = new Builder(path);
        pages = new Pages("en");
        HttpServer server = HttpServer.create(
                new InetSocketAddress("127.0.0.1", port), 0);
        server.createContext("/", BuilderServer::handle);
        server.start();
        System.out.println("VCIRS Builder: open http://localhost:" + port
                + " in your browser; Ctrl+C stops it.");
    }
}
```

Server JDK memberi bantuan lebih sedikit daripada Flask, jadi server Java mengerjakan tiga hal sendiri. `form` mengurai teks `application/x-www-form-urlencoded` — format yang dipakai peramban untuk mengirim formulir, `a=1&b=x+y` — menjadi *map* dari nama ke daftar nilai, dengan `URLDecoder`. `send` menulis jawaban beserta jenis isi dan panjangnya. Dan `handle` mengerjakan perutean sendiri, dengan satu `switch` atas metode dan jalur, seperti `"POST /teach"`. `handle` bersifat `synchronized` agar dua permintaan yang datang bersamaan — Bu Wati di laptop dan Rini di ponselnya — tidak pernah mengubah basis pengetahuan pada saat yang sama. Server Python mendapat jaminan yang sama dari `threaded=False` dalam `app.run`, yang membuat Flask menjawab satu permintaan setiap kali; untuk toko dengan dua atau tiga orang, satu permintaan setiap kali sudah jauh lebih dari cukup cepat.

Untuk menjalankan pembangun, jalankan salah satu server:

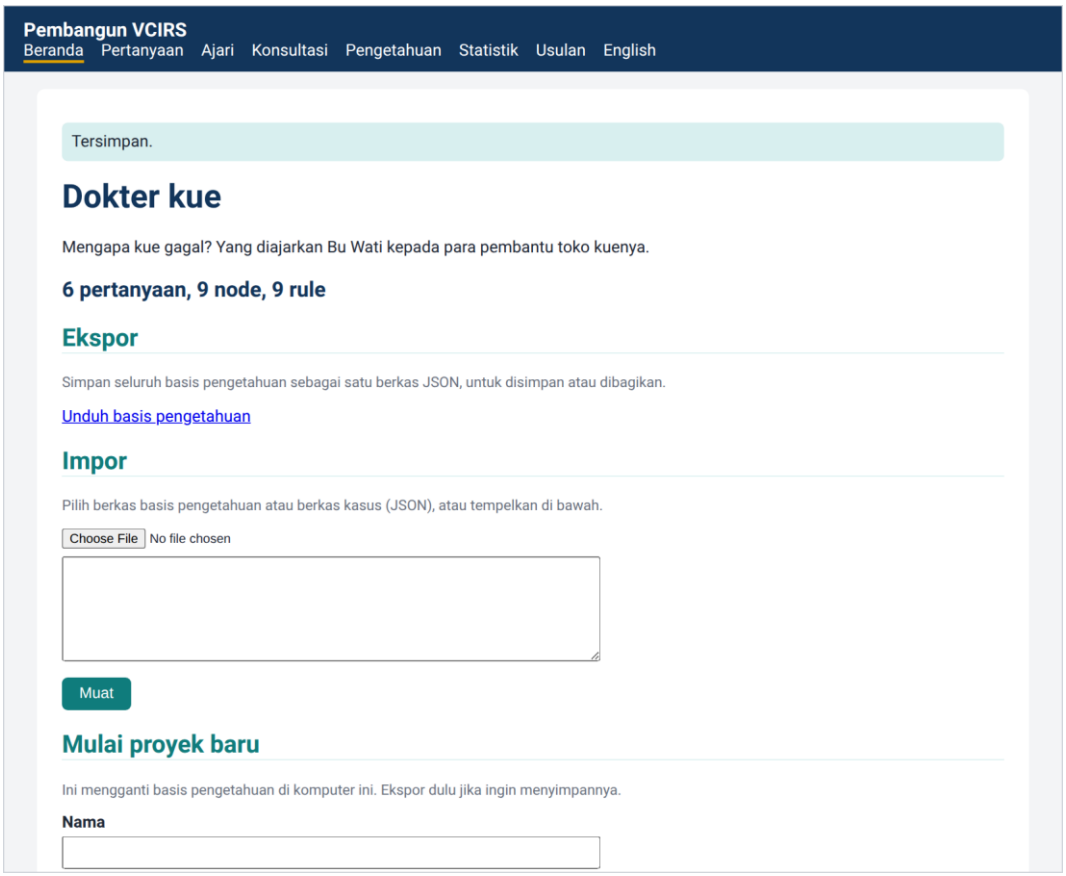
```
cd Chapter14/java
javac -d out -sourcepath ../../vcirs-core/java/src *.java
java -cp out BuilderServer
```

atau, setelah `python -m pip install -r requirements.txt` di `Chapter14/python`, `python builder_app.py`. Keduanya mencetak baris seperti `VCIRS Builder: open http://localhost:4141 in your browser; Ctrl+C stops it.` Bukalah alamat itu. Kedua server menyimpan basis pengetahuannya di `data/builder-kb.json`, sehingga Anda dapat menghentikan yang satu, menjalankan yang lain, dan melanjutkan dari tempat terakhir.

Benarkah kedua server sepakat? Untuk memeriksanya, sebuah program uji kecil (disimpan di luar folder bab) mengirim 33 permintaan yang sama ke setiap server — mengimpor kasus-kasus kue, membuka setiap halaman dalam kedua bahasa, berkonsultasi, mengajarkan tiga kasus termasuk satu yang ditolak, mendeklarasikan pertanyaan, menerima satu usulan dan satu usulan yang salah, menghentikan node, mengimpor berkas rusak, dan membuat proyek tanpa nama — lalu menyimpan setiap halamannya. Ke-33 halaman dari server Java dan ke-33 halaman dari server Python identik, bita demi bita, begitu pula kedua berkas basis pengetahuan di akhirnya.

14.7 Memakai Pembangun

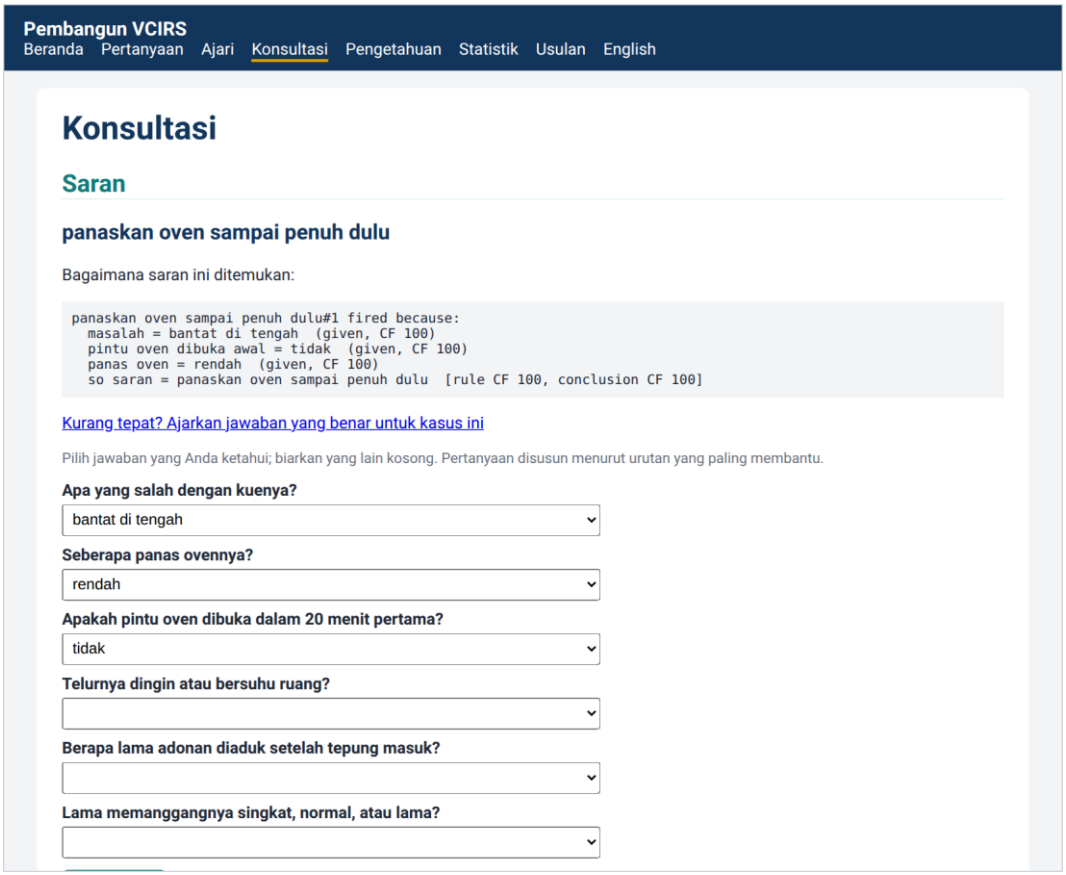
Inilah malam pertama Bu Wati dengan pembangun, dalam tangkapan layar aplikasi yang sedang berjalan, dengan tampilan berbahasa Indonesia dan basis pengetahuannya dalam bahasa Indonesia. Di halaman Beranda ia memilih `kue-cases-id.json` di bagian *Impor* — berkas tempat keponakannya mengetik halaman-halaman pertama buku catatannya — lalu menekan **Muat**.



Gambar 14.2 Halaman Beranda setelah mengimpor kasus-kasus kue: nama dan keterangan proyek, ukurannya, serta bagian ekspor, impor, dan proyek baru. “Tersimpan.” menegaskan perubahan terakhir.

Gambar 14.2 memperlihatkan hasilnya: proyek itu kini bernama *Dokter kue*, dengan 6 pertanyaan, 9 node, dan 9 rule. Tidak ada kata “node” di halaman itu yang sudah harus ia pahami; angka-angkanya ada di sana untuk saat ia membutuhkannya.

Kue Rini bantat. Ia membuka **Konsultasi**, memilih *bantat di tengah*, *rendah* untuk panas oven, *tidak* untuk pintu oven, lalu menekan **Beri saran**.



Gambar 14.3 Halaman Konsultasi: saran dalam huruf besar, bagaimana saran itu ditemukan, tautan untuk mengajarkan jawaban yang benar bila saran itu salah, dan pertanyaan-pertanyaan dalam urutan yang paling membantu.

Gambar 14.3 memperlihatkan sarannya, *panaskan oven sampai penuh dulu*, dengan penjelasan di bawahnya: node yang menyala dan tiga fakta yang membuatnya menyala. Pertanyaan-pertanyaannya tersusun menurut urutan di halaman Statistik, yang paling berguna lebih dulu, dan jawaban yang ia pilih tetap terpilih. Seandainya sarannya salah, tautan *Kurang tepat? Ajarkan jawaban yang benar untuk kasus ini* akan membuka halaman Ajari dengan jawaban yang sama sudah terpilih.

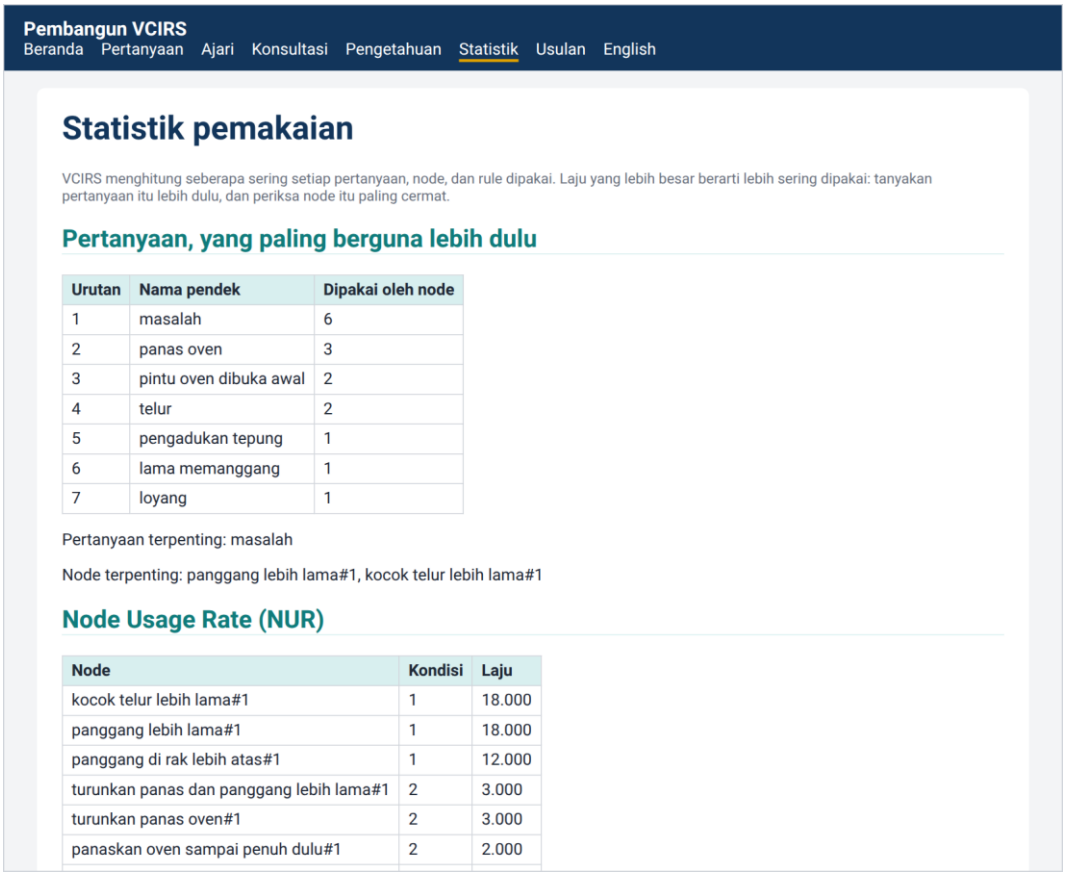
Itulah yang terjadi berikutnya, dengan kue yang gosong. Basis pengetahuan menjawab *panggang di rak lebih atas*, tetapi kali ini kuenya dipanggang dalam loyang logam gelap dan rak yang lebih atas tidak menolong; loyang yang lebih terang yang menolong. Rini mengikuti tautan itu, mengetik `loyang = logam gelap` di kotak fakta lain, *pakai loyang berwarna terang* sebagai jawaban yang tepat, dan namanya.



Gambar 14.4 Hasil mengajar: apa yang dijawab basis pengetahuan sebelumnya, apa yang terjadi, dan mengapa kasus baru berbeda dari kasus yang menciptakan node lama.

Gambar 14.4 adalah halaman terpenting pembangun. Halaman itu menyebut apa yang dijawab basis pengetahuan sebelumnya, apa yang terjadi — node baru menjadi pengecualian node lama — dan daftar perbedaan yang membenarkannya: tidak ada yang hanya ada pada kasus lama, `loyang = logam gelap` hanya ada pada kasus baru. Rini tidak perlu tahu apa itu node atau pengecualian untuk mengajarkannya; halaman itu memberitahunya, dalam kata-kata mesin (yang masih berbahasa Inggris, lihat Subbab 14.8), dan konsultasi berikutnya dengan loyang gelap akan memberinya saran.

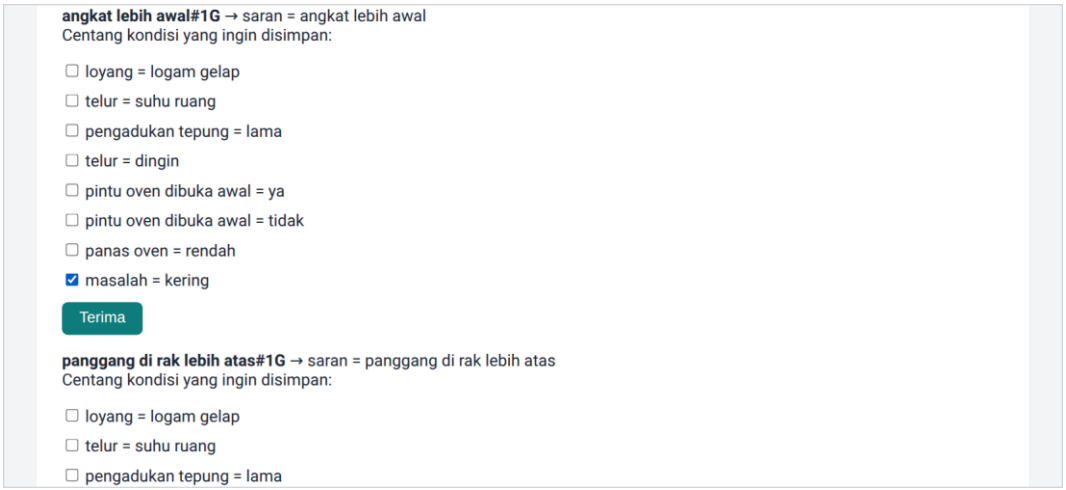
Kemudian, Bu Wati membuka **Statistik**.



Gambar 14.5 Halaman Statistik: pertanyaan dari yang paling hingga paling tidak berguna beserta banyaknya node yang memakai masing-masing, pertanyaan dan node terpenting, dan Node Usage Rate, dari yang terbesar.

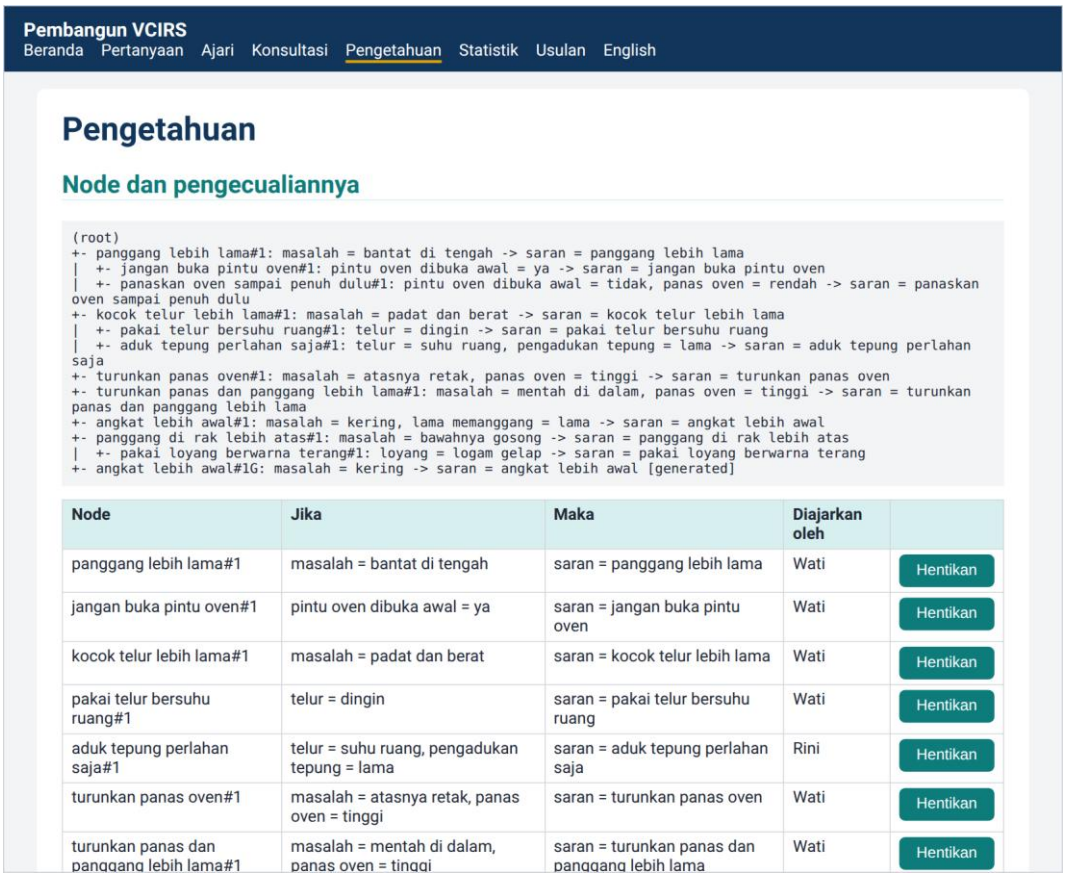
Gambar 14.5 menjawab dua pertanyaannya sekilas pandang. Pertanyaan mana yang paling penting? `masalah`, dipakai enam node, lalu `panas oven`. Potongan pengetahuan mana yang paling berbobot? *kocok telur lebih lama* dan *panggang lebih lama*, masing-masing dengan NUR 18: setiap pengecualian tentang telur dan pintu oven bergantung pada keduanya. Sisa halaman, di bawah tangkapan layar, mendaftar RUR setiap rule dan VUR setiap kondisi, untuk hari ketika ia ingin melihat perinciannya.

Di halaman **Usulan** ia menemukan daftar usulan rule dan node. Sebagian besar tampak aneh baginya, dan halaman itu berkata bahwa itu tidak apa-apa: simpan yang masuk akal. Pada usulan untuk *angkat lebih awal* ia hanya mencentang `masalah = kering`.



Gambar 14.6 Sebuah usulan node di halaman Usulan. Dari kondisi-kondisi yang dikumpulkan VCIRS dalam urutan yang baik, Bu Wati menyimpan satu, `masalah = kering`, untuk membuat node yang lebih pendek dan lebih umum.

Gambar 14.6 memperlihatkan usulan itu dengan centangnya. Setelah **Terima**, node tingkat teratas yang baru, *angkat lebih awal#1G*, menjawab setiap kue yang kering, bahkan ketika tidak ada yang ingat berapa lama kue itu dipanggang. Terakhir ia membuka **Pengetahuan**.



Gambar 14.7 Halaman Pengetahuan: pohon node dan pengecualian, dengan tanda bila sebuah node hasil pembangkitan, serta tabel berisi siapa yang mengajarkan setiap node dan tombol Hentikan.

Gambar 14.7 memperlihatkan pengetahuannya sebagai pohon: setiap node umum dengan pengecualian-pengecualian di bawahnya, pengecualian baru untuk loyang gelap, dan node hasil pembangkitan di bagian bawah, bertanda `[generated]`. Tabel di bawahnya menyebut siapa yang mengajarkan setiap node — Wati atau Rini — dan menawarkan tombol **Hentikan** untuk saran

apa pun yang tidak lagi ingin ia berikan. Lebih ke bawah lagi, riwayatnya mendaftar setiap perubahan beserta nomor opsinya, jejak audit yang sama yang dipakai Bab 11 untuk tim.

Semua yang ia lihat juga tersedia dalam bahasa Inggris. Tautan *English* di bagian atas mengganti setiap label, petunjuk, dan tombol; pertanyaan dan jawaban milik basis pengetahuan itu sendiri tetap dalam bahasa apa pun yang ia pakai untuk menulisnya. Edisi bahasa Inggris bab ini memperlihatkan pembangun yang sama dengan *Cake doctor*, diimpor dari `cake-cases.json`.

14.8 Membuatnya Lebih Baik

Beberapa proyek. Pembangun memegang satu basis pengetahuan setiap kali. Perluasan sederhana menyimpan sebuah folder berisi basis-basis pengetahuan dan daftar pilihan di halaman Beranda untuk memilih salah satunya; kelas pembangun tidak berubah, karena ia sudah menerima berkasnya sebagai parameter.

Orang dan hak akses. Siapa pun yang dapat membuka halaman dapat mengajar dan menghentikan node. Di toko keluarga itu sudah tepat. Dalam tim yang lebih besar, tambahkan proses masuk (*login*) dan izinkan hanya sebagian orang mengajar; Bab 11 memperlihatkan fitur-fitur tim — beberapa pakar, versi, dan jejak audit — yang akan dipakai pembangun semacam itu, dan nama pengajar yang sudah dicatat pembangun untuk setiap node adalah langkah pertamanya.

Pesan mesin dalam bahasa penggunanya. Label halaman berpindah ke bahasa Indonesia, tetapi kalimat-kalimat milik mesin — "refined: … added as an exception of …", "Differences with the cornerstone case of …" — tetap berbahasa Inggris, karena berasal dari vcirs-core. Pembangun yang lebih ramah akan menyusun kalimatnya sendiri dari `LearnResult` dan `DifferenceList` (semua bagiannya tersedia) dalam bahasa yang dipilih. Soal 14.5 memintanya.

Memeriksa cornerstone yang terdampak. Sebelum menerima kasus yang diajarkan, pembangun dapat menampilkan kasus-kasus *cornerstone* yang jawabannya akan berubah, dengan `affected_cornerstones`, dan meminta konfirmasi, seperti diuraikan Bab 3. Dengan beberapa puluh node hal ini tidak terlalu penting; dengan ratusan node, inilah jaring pengaman yang membuat basis pengetahuan besar tetap dapat dirawat [1].

Pertanyaan fuzzy. Pertanyaan seperti *berapa lama dipanggang?* sebenarnya berupa bilangan. Mesin memahami nilai fuzzy (Bab 9); halaman Pertanyaan dapat memungkinkan orang mendefinisikan *singkat*, *normal*, dan *lama* sebagai rentang menit, dan halaman Konsultasi dapat menerima bilangan.

Urungkan. Karena tidak ada yang pernah dihapus, "urungkan" cukup berarti "hentikan node yang baru saja saya tambahkan". Tombol yang melakukannya untuk perubahan terakhir orang yang sedang memakai akan membuat bereksperimen tidak lagi menakutkan.

14.9 Memakainya Sungguh-Sungguh

Di toko Bu Wati. Jalankan server di laptop dekat oven dan bukalah di peramban laptop itu. Untuk memakainya dari ponsel-ponsel di toko, jalankan server dengan alamat laptop di Wi-Fi toko (Flask: `app.run(host="0.0.0.0")`; Java: `new InetSocketAddress("0.0.0.0", port)`) dan buka `http://<alamat-laptop>:4141` di ponsel. Lakukan ini hanya di dalam jaringan milik toko sendiri: pembangun tidak punya kata sandi, dan siapa pun yang dapat menjangkaunya dapat mengubah

pengetahuannya. Bab 15 memperlihatkan cara menaruh basis pengetahuan yang sudah jadi di balik layanan kecil yang hanya-baca.

Memulai dari buku catatan. Cara tercepat untuk memulai adalah cara bab ini dimulai: ketik kasus-kasus dari buku catatan ke dalam berkas kasus — satu baris per kasus, dengan fakta, jawaban, dan siapa yang mengajarkannya — lalu impor. Pembangun mengajarkannya satu per satu, sehingga hasilnya sama dengan basis pengetahuan yang diketik di halaman Ajari, dan riwayatnya menunjukkan setiap kasus. Sesudah itu, ajarkanlah di etalase atau di dekat oven, ketika kasus nyata terjadi.

Membagikan. JSON hasil ekspor adalah seluruh pengetahuan: pertanyaan, node, rule, kasus *cornerstone*, dan riwayat. Bu Wati dapat mengirimkannya kepada adiknya, yang mengimpornya dan melanjutkan mengajar dengan kasus-kasusnya sendiri; basis pengetahuan kedua toko kemudian tumbuh sendiri-sendiri, masing-masing dengan riwayatnya. Berkas itu juga yang dibaca dokter Wi-Fi Bab 3, aplikasi web Bab 10, dan layanan Bab 15, karena semuanya memakai vcirs-core.

Bidang lain. Tidak ada bagian pembangun yang khusus tentang kue. Bengkel sepeda motor, kelompok tani padi, meja depan klinik kecil (dengan aturan rujukan yang aman dari Bab 5 [10]), pembantu komputer di sekolah: masing-masing mulai dengan proyek kosong, menulis pertanyaannya, lalu mengajar. Satu-satunya anggapan pembangun adalah bahwa ada seseorang yang tahu jawabannya ketika melihat sebuah kasus.

Melatih pembantu baru. Pembangun juga seorang guru. Pembantu baru dapat membuka halaman Pengetahuan dan membaca pohonnya seperti Bu Wati dulu membacakan buku catatannya: setiap saran umum dengan pengecualian-pengecualian di bawahnya, dan nama orang yang mengajarkan masing-masing. Lebih baik lagi, ia dapat memakai halaman Konsultasi sebagai permainan latihan untuk dirinya sendiri — memilih jawaban untuk kue gagal yang pernah ia lihat, menebak sarannya, lalu menekan **Beri saran** dan membaca penjelasannya. Setelah sebulan ia akan mengetahui sebagian besar yang diketahui basis pengetahuan, dan ia sudah melihat *mengapa*, fakta demi fakta. Dan ketika ia menemukan kasus yang dijawab salah oleh basis pengetahuan, ia mengajarkannya sendiri, tanda terbaik bahwa ia sudah menjadi salah satu pakarnya.

Simpan salinan. Ekspor basis pengetahuan setiap minggu dan simpan berkas-berkasnya. Ukurannya kecil, isinya pengetahuan toko, dan Bab 15 memperlihatkan cara membuat salinan itu secara otomatis dan memeriksanya.

14.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Menulis rule alih-alih mengajarkan kasus. Orang yang mengetik setiap rule yang dapat ia bayangkan ke dalam berkas kasus, tanpa kasus nyata, sedang membangun basis rule datar yang menyamar. *Hindari:* ajarkan dari kue yang benar-benar gagal; dengan begitu setiap kasus punya *cornerstone* yang benar-benar diperiksa.

Kata berbeda untuk jawaban yang sama. `dense`, `dense and heavy`, dan `heavy` adalah tiga nilai yang berbeda bagi mesin. *Hindari:* tulis jawaban-jawaban biasa di halaman Pertanyaan dan pilihlah dari daftar pilihan; ketik nilai baru hanya bila memang baru.

Menerima usulan tanpa membacanya. Usulan node dengan semua kondisinya dicentang biasanya bertentangan dengan dirinya sendiri (`problem = dry` *dan* `problem = raw inside`).

Hindari: centang hanya kondisi-kondisi yang memang saling berkaitan, dan abaikan usulan yang tidak masuk akal.

Menyusun HTML tanpa meloloskan karakter. Satu nama pertanyaan yang memuat `<` merusak halaman; yang berniat jahat menyerang setiap pengunjung. *Hindari:* lewatkan setiap potong teks melalui `esc`, seperti di setiap halaman di sini [7].

Mengubah basis pengetahuan dari dua tempat sekaligus. Dua server pada berkas yang sama, atau dua tab dari salinan lama, dapat saling menimpa perubahan. *Hindari:* jalankan satu server untuk setiap berkas basis pengetahuan, dan biarkan semua orang memakainya melalui peramban.

Menampilkan hasil POST tanpa pengalihan. Memuat ulang lalu mengulang perubahannya. *Hindari:* alihkan setelah setiap perubahan (*post, redirect, get*), seperti yang dilakukan `done`; tampilkan halaman langsung hanya bila halaman itu sendiri hasilnya, seperti untuk Ajari dan Konsultasi.

Membiarkan pembangun terjangkau dari internet. Pembangun tidak punya kata sandi. *Hindari:* dengarkan di `127.0.0.1`, atau paling jauh di jaringan milik toko sendiri.

Hanya menyimpan satu salinan. Laptop yang jatuh dari rak membawa serta catatan sebelas tahun. *Hindari:* ekspor secara teratur dan simpan salinannya di tempat lain.

14.11 Ringkasan Bab

- Pembangun memungkinkan orang yang tidak memprogram membuat, menumbuhkan, dan membagikan sistem pakar VCIRS; setiap halaman bersandar pada satu gagasan VCIRS (**Tabel 14.1**), dan orangnya berbicara dalam **kasus**, tidak pernah dalam rule.
- **Statistik pemakaian** mengatakan seberapa besar bagian basis pengetahuan yang bergantung pada setiap pertanyaan, node, dan rule: VUR = Credit × NS × CD (Persamaan (14.1)), NUR dan RUR adalah rata-rata (Persamaan (14.2)); laju yang besar menandai pengetahuan yang harus ditinjau lebih dulu dan pertanyaan yang harus ditanyakan lebih dulu.
- **Usulan** dari pembangkitan rule dan node hanyalah usulan: orangnya menyimpan kondisi-kondisi yang memang saling berkaitan (**Gambar 14.6**).
- Aplikasinya memisahkan **pembangun** (**Kode 14.1**, **Kode 14.2**), **halaman** (**Kode 14.5**, **Kode 14.6**), dan **server** (**Kode 14.7**, **Kode 14.8**) (**Gambar 14.1**); sebuah skrip memerankan satu sesi utuh tanpa web (**Kode 14.3**, **Kode 14.4**).
- Server Java (HttpServer JDK) dan server Python (Flask) menghasilkan halaman dan berkas basis pengetahuan yang sama bita demi bita, dalam bahasa Inggris dan Indonesia, dan tangkapan-tangkapan layar (**Gambar 14.2**, **Gambar 14.3**, **Gambar 14.4**, **Gambar 14.5**, **Gambar 14.7**) memperlihatkan satu malam pemakaian yang utuh.

14.12 Soal Latihan

14.1 Dengan Persamaan (14.1) dan Persamaan (14.2), hitunglah dengan tangan VUR kedua kondisi *fold the flour in gently#1* (`eggs = room temperature`, lalu `flour mixing = long`), NUR-nya, dan RUR rule yang berakhir di node itu. Bandingkan dengan keluaran di Subbab 14.4.

14.2 Rini mengajarkan `problem = burnt bottom` → *use a lighter pan* dan pembangun menolak. Jelaskan mengapa, dengan istilah kasus *cornerstone* dan daftar perbedaan. Bagaimana agar

pembangun menerima pengetahuannya, dan mengapa menghentikan node lama lebih dulu tidak menolong?

14.3 Bu Wati berpendapat *lower the heat#1* (`problem = cracked top`, `oven heat = high`) salah: menurut pengalamannya bagian atas yang retak berasal dari terlalu banyak baking powder. Halaman mana yang ia pakai, dan apa yang ia lakukan, agar saran lama tidak lagi diberikan tetapi riwayatnya tetap menunjukkannya? Apa yang ia ajarkan berikutnya?

14.4 Jelaskan mengapa pembangun menyimpan basis pengetahuan setelah setiap perubahan alih-alih menyediakan tombol Simpan. Apa kerugian tombol Simpan bagi Bu Wati pada hari Sabtu yang ramai?

14.5 Kalimat-kalimat mesin di halaman hasil Ajari berbahasa Inggris bahkan di pembangun berbahasa Indonesia. Tuliskan, dengan kata-kata atau kode, bagaimana `pages.taught` dapat menyusun kalimat Indonesia dari `result.action`, `result.node.id`, dan `result.fired`, dengan dua kunci baru di `ui-text.json`.

14.6 Seorang pembantu mengetik nama pertanyaan `oven <heat>` di halaman Pertanyaan. Apa yang dilakukan pembangun? Dan bila ia mengetik `<b>hot</b>` sebagai jawaban biasa `oven heat`, apa yang ditampilkan halaman Konsultasi, dan mengapa itu aman?

14.7 Dua usulan rule di Subbab 14.4 memuat deretan panjang node tingkat teratas. Mengapa menerimanya tidak menambahkan sesuatu yang berguna bagi basis pengetahuan Bu Wati? Kapan usulan rule menjadi berguna?

14.8 Uraikan permintaan-permintaan yang dikirim peramban ketika Bu Wati menambahkan pertanyaan di halaman Pertanyaan lalu menekan tombol muat ulang peramban. Berapa kali pertanyaan itu tersimpan, dan mengapa?

14.13 Penyelesaian Soal Latihan

Penyelesaian 14.1. Node itu punya TV = 2 kondisi. Untuk `eggs = room temperature`: VO = 1, CD = 1/2; NS(`eggs`) = 2 (node ini dan *use room-temperature eggs#1*); Credit = 1 (tidak pernah ada kasus yang diajarkan di bawah node ini). VUR = 1 × 2 × 0,5 = 1. Untuk `flour mixing = long`: VO = 2, CD = 2/2 = 1; NS = 1; Credit = 1; VUR = 1 × 1 × 1 = 1. NUR = (1 + 1)/2 = 1, yaitu baris pertama daftar NUR di Subbab 14.4 (`NUR 1.000  fold the flour in gently#1`). Rule yang berakhir di node ini melewati *beat the eggs longer#1* (NUR 18) dan node ini (NUR 1), sehingga RUR = (18 + 1)/2 = 9,5.

Penyelesaian 14.2. Node *bake on a higher rack#1* diciptakan oleh kasus dengan satu fakta `problem = burnt bottom`; itulah kasus *cornerstone*-nya. Kasus Rini punya fakta yang persis sama, sehingga daftar perbedaannya kosong di kedua bagian: tidak ada kondisi yang benar untuk kasusnya dan salah untuk kasus *cornerstone*, dan pengecualian yang dibangun dari ketiadaan akan menyala untuk setiap kue gosong, mengubah jawaban kasus lama. Karena itu pembangun menolak. Jalan keluarnya adalah menambahkan fakta yang memang berbeda pada kasusnya — `pan = dark metal`, atau fakta lain yang tidak dimiliki kasus *cornerstone* — yang memberi bagian B satu kondisi sehingga kasusnya dapat menjadi pengecualian, seperti dalam bab ini. Menghentikan *bake on a higher rack#1* lebih dulu tidak menolong: node yang dihentikan tetap menyala, hanya tidak memberi saran, sehingga kasus baru akan kembali dibandingkan dengan kasus *cornerstone* yang sama dan ditolak dengan alasan yang sama. Sebuah perbedaan harus disebutkan; itulah yang menjaga basis pengetahuan agar tidak bertentangan dengan dirinya sendiri.

Penyelesaian 14.3. Ia membuka **Pengetahuan** dan menekan **Hentikan** di samping *lower the heat#1*. Node itu tetap berada di basis pengetahuan beserta kasus *cornerstone*-nya, pengecualiannya (di sini tidak ada), dan catatan riwayatnya, tetapi tidak lagi memberi saran; riwayatnya bertambah satu baris `stop`. Kemudian, lain kali ada kue yang retak, ia mengajarkan kasus itu di halaman Ajari dengan fakta-fakta yang ia ketahui dan jawaban yang tepat, *use less baking powder*, sambil menambahkan fakta baru `baking powder = too much`. Apa yang dilakukan `learn` bergantung pada faktanya. Bila kasusnya juga memuat `oven heat = high`, node yang dihentikan tetap menyala — node yang dihentikan hanya berhenti memberi saran — sehingga kasus baru digantungkan di bawahnya sebagai pengecualian yang dibangun dari perbedaan `baking powder = too much`. Bila panas ovennya tidak disebutkan, node yang dihentikan tidak menyala, tidak ada yang menjawab, dan kasusnya menjadi node tingkat teratas yang baru. Bagaimanapun, saran lama tidak pernah diberikan lagi, dan riwayatnya menyimpan kedua langkah itu.

Penyelesaian 14.4. Orang yang tidak memprogram sama sekali tidak memikirkan "berkasnya"; ia merasa sudah memberi tahu mesin sesuatu. Bila menyimpan merupakan langkah tersendiri, hari Sabtu dengan tiga kue gagal, anak yang menangis, dan pelanggan di pintu akan berakhir dengan peramban yang tertutup dan tidak ada yang tersimpan. Menyimpan setelah setiap perubahan hanya makan beberapa milidetik dan membuat janji halaman itu benar: apa yang Anda ajarkan tersimpan. Hal itu juga membuat berkas di diska selalu sesuai dengan apa yang ditampilkan halaman, yang menjadi sandaran salinan cadangan di Bab 15.

Penyelesaian 14.5. Tambahkan dua kunci ke `ui-text.json`, misalnya `"r_refined": "{new} ditambahkan sebagai pengecualian dari {old}"` dan `"r_added": "{new} ditambahkan di tingkat teratas"` (serta versi Inggrisnya), ditambah `"r_accepted"`. Di `taught`, pilih kuncinya berdasarkan `result.action` dan ganti `{new}` dengan `result.node.id` dan `{old}` dengan `result.fired` sebelum meloloskan karakter; untuk `accepted` tidak ada node baru. Gagasan yang sama berlaku untuk daftar perbedaan: susun "Hanya pada kasus lama: …" dan "Hanya pada kasus baru: …" dari `result.differences.a` dan `.b`. Di Java penggantiannya `String.replace`, di Python `str.replace`, sehingga kedua bahasa tetap menghasilkan bita yang sama.

Penyelesaian 14.6. `oven <heat>` memuat `<`, sehingga `declare` menolaknya dengan "a question name must not be empty or contain = < >", ditampilkan dengan warna merah di bagian atas halaman Pertanyaan, dan tidak ada yang tersimpan. Jawaban `<b>hot</b>` diterima sebagai nilai (hanya nama pertanyaan yang dibatasi), dan di halaman Konsultasi nilai itu muncul di daftar pilihan sebagai teks harfiah `<b>hot</b>`, karena `esc` mengubahnya menjadi `<b>hot</b>` sebelum ditaruh di HTML. Peramban menampilkan karakter-karakternya alih-alih menafsirkannya sebagai tag, sehingga tidak ada nilai yang dapat mengubah halaman atau menjalankan skrip.

Penyelesaian 14.7. Rule VCIRS adalah jalur node, dan mesin RDR sudah menelusuri jalur-jalur yang ada: setiap node beserta leluhurnya. Usulan rule itu merangkai node-node tingkat teratas tentang masalah yang berbeda-beda (kering, mentah, retak, gosong, padat, bantat), yang tidak pernah menjadi milik satu kasus, sehingga rule yang melewati semuanya tidak akan pernah menyala untuk kue sungguhan. Usulan rule menjadi berguna pada basis pengetahuan yang node-nodenya berbagi banyak variabel dan rule-rulenya berupa jalur panjang, seperti contoh Zookeeper dalam tesis, tempat usulan rule mengunjungi node-node bersama dalam urutan yang baik dan dapat mempercepat inferensi [4]. Pada basis pengetahuan muda yang tersusun di sekitar satu pertanyaan, `problem`, usulan *node*-lah bagian yang berguna, karena orang dapat memetik generalisasi darinya.

Penyelesaian 14.8. Peramban mengirim `POST /variables` dengan kolom-kolom formulir. Server mendeklarasikan pertanyaan, menyimpan berkas, mengisi pesan "Tersimpan.", lalu menjawab `303 See Other` dengan `Location: /variables`. Peramban kemudian mengirim `GET /variables` dan menampilkan halaman beserta pesannya. Ketika Bu Wati menekan muat ulang, peramban mengulang permintaan *terakhir*, yaitu `GET`; halaman ditampilkan lagi (tanpa pesan, yang sudah tampil sekali) dan pertanyaan tersimpan tepat satu kali. Tanpa pengalihan, muat ulang akan mengulang `POST` — kebanyakan peramban terlebih dulu bertanya "kirim ulang formulir?" — dan mendeklarasikan pertanyaan itu lagi.

Daftar Pustaka

[1] P. Compton and B. H. Kang, *Ripple-Down Rules: The Alternative to Machine Learning*. Boca Raton, FL, USA: CRC Press, 2021.

[2] G. Edwards, P. Compton, R. Malor, A. Srinivasan, and L. Lazarus, "PEIRS: a pathologist-maintained expert system for the interpretation of chemical pathology reports," *Pathology*, vol. 25, no. 1, pp. 27–34, 1993.

[3] J. Nielsen, "10 usability heuristics for user interface design," Nielsen Norman Group, updated Jan. 30, 2024. [Online]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/

[4] M. M. I. Subakti, "A Variable-Centered Intelligent Rule System," M.Sc.Eng. thesis, Dept. of Computer Science and Information Engineering, Taipei, Taiwan, Jun. 2005.

[5] I. Subakti, "A variable-centered intelligent rule system," in *Proc. 1st Annual Int. Conf. Information and Communication Technology Seminar (ICTS 2005)*, vol. 1, no. 1, Surabaya, Indonesia, Aug. 2005, pp. 167–174.

[6] I. Subakti, "Some revisions in VCIRS and cases reconstructing perspectives," in *Proc. Information and Communication Technology Seminar (ICTS 2006)*, Surabaya, Indonesia, 2006.

[7] OWASP Foundation, "Cross site scripting prevention cheat sheet," *OWASP Cheat Sheet Series*, 2026. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

[8] Pallets, "Flask 3.1.3," Python Package Index, Feb. 19, 2026. [Online]. Available: https://pypi.org/project/Flask/

[9] Oracle, "Package com.sun.net.httpserver," *Java SE 25 & JDK 25 API Specification*, 2025. [Online]. Available: https://docs.oracle.com/en/java/javase/25/docs/api/jdk.httpserver/com/sun/net/httpserver/package-summary.html

[10] I. Subakti and R. Hidayatullah, "Aplikasi sistem pakar untuk diagnosis awal gangguan kesehatan secara mandiri menggunakan Variable-Centered Intelligent Rule System," vol. 6, no. 1, pp. 11–16, Jan. 2007.

Bab 15

Menerapkan, Merawat, dan Mengembangkan Sistem Pakar Anda

Setelah menyelesaikan bab ini Anda akan mampu mengubah basis pengetahuan yang sudah jadi menjadi sebuah **layanan saran** kecil yang berjalan setiap hari di komputer biasa dan menjawab ponsel, tablet, atau program apa pun di toko melalui HTTP; mengemas layanan itu beserta alat-alatnya sebagai **satu berkas** — `.jar` untuk Java atau `.pyz` untuk Python — dan memasangnya di Windows, macOS, atau Linux agar berjalan sendiri; memeriksa setiap penerapan dengan uji otomatis yang singkat; membuat **cadangan** basis pengetahuan yang dapat dipercaya, memeriksanya, dan memulihkan salah satunya; menyimpan **catatan pemakaian** yang merekam saran mana yang diberikan dan kasus mana yang tidak terjawab, tanpa merekam apa pun tentang orang-orang yang bertanya; membaca **laporan pemakaian** bulanan yang memberi tahu apa yang perlu diajarkan berikutnya dan pengetahuan mana yang tidak pernah dipakai; mematuhi undang-undang pelindungan data pribadi Indonesia, yang peraturan pelaksanaannya terbit pada Juli 2026; berpikir jernih tentang etika memberi saran lewat mesin; dan melihat ke mana melangkah selanjutnya, dengan buku ini di belakang Anda.

15.1 Pagi Ketika Laptop Tidak Mau Menyala

Selama tiga bulan dokter kue Bu Wati, yang dibangun dengan pembangun dari Bab 14, tinggal di laptop tua dekat oven. Ia bekerja dengan baik. Rini memakainya pada hari Sabtu yang ramai, pembantu baru belajar darinya, dan Bu Wati mengajarinya kasus baru hampir setiap minggu. Lalu, pada suatu Senin di bulan September, laptop itu tidak mau menyala. Layarnya tetap hitam, tukang servis berkata diskanya rusak, dan selama satu sore yang menakutkan Bu Wati mengira tiga bulan pengajarannya sudah lenyap.

Ternyata tidak, tetapi hanya karena untung: dua minggu sebelumnya keponakannya mengekspor basis pengetahuan ke ponselnya untuk ditunjukkan kepada seorang teman. Kasus-kasus dua minggu terakhir hilang untuk selamanya. Sore itu membuat Bu Wati mengajukan tiga pertanyaan yang cepat atau lambat diajukan setiap pemilik sistem pakar. *Bagaimana memastikan pengetahuannya selamat ketika sebuah komputer mati? Bagaimana membiarkan semua orang di toko memakai sarannya tanpa membiarkan semua orang mengubahnya?* Dan, setelah rasa takutnya reda, pertanyaan yang lebih pelan: *Apakah dokter kue benar-benar membantu? Saran mana yang dipakai, dan apa yang ditanyakan orang yang tidak dapat ia jawab?*

Bab ini menjawab pertanyaan-pertanyaan itu untuk dokter kue, dan melaluinya untuk setiap aplikasi dalam buku ini. Inilah bab tentang semua yang terjadi *setelah* programnya bekerja: menerapkannya, menjaganya tetap hidup, menjaga pengetahuannya tetap aman, mengamati bagaimana ia dipakai, menaati hukum tentang orang-orang yang dilayaninya, dan menumbuhkannya selama bertahun-tahun. Tidak ada yang gemerlap di sini, dan semuanya menentukan apakah sebuah sistem pakar masih menolong seseorang tiga tahun dari sekarang.

15.2 Gagasannya dengan Kata-Kata Sederhana

15.2.1 Dari program yang Anda jalankan menjadi layanan yang berjalan

Sampai sekarang, setiap aplikasi dalam buku ini adalah program yang dijalankan seseorang: dokter Wi-Fi mengajukan pertanyaannya di terminal lalu selesai; pembangun berjalan selama jendelanya terbuka. **Menerapkan** (*deploy*) sebuah aplikasi berarti menaruhnya di tempat ia akan dipakai, dalam bentuk yang terus berjalan tanpa pemrogramnya: ia menyala ketika komputer menyala, menyala ulang sendiri bila gagal, menyimpan datanya di tempat yang diketahui, dan dapat diperiksa, dicadangkan, dan diperbarui oleh orang yang bukan pemrogram.

Untuk dokter kue bentuk yang tepat adalah sebuah **layanan** kecil: program yang berjalan di latar belakang pada satu komputer di toko dan menjawab pertanyaan melalui HTTP — jenis permintaan HTTP yang sama dengan yang dipakai Bab 10, 13, dan 14. Laptop dekat oven, tablet di etalase, atau ponsel dapat meminta saran kepadanya; program lain di kemudian hari, seperti antarmuka obrolan Bab 13, juga dapat bertanya kepadanya. Layanan ini punya tiga pintu. `GET /health` menjawab "saya hidup, dan inilah basis pengetahuan yang saya pakai". `GET /questions` mendaftar pertanyaan-pertanyaan, yang paling berguna lebih dulu, beserta jawaban-jawaban biasanya, sehingga sebuah layar dapat menampilkannya. `POST /advise` menerima fakta sebagai JSON dan mengembalikan saran beserta penjelasannya.

Layanan hanya *membaca* basis pengetahuan; ia tidak pernah mengubahnya. Pengajaran tetap terjadi di pembangun, oleh orang-orang yang dipercaya Bu Wati, di laptopnya sendiri. Ketika versi baru basis pengetahuan sudah siap, berkasnya disalin ke komputer layanan, dan layanan menyadari berkas baru itu dan memuatnya sendiri. Pemisahan ini — satu tempat untuk mengubah pengetahuan, satu tempat untuk memakainya — adalah cara paling sederhana untuk membiarkan semua orang memakai saran tanpa membiarkan semua orang mengubahnya.

15.2.2 Satu berkas untuk disalin

Penerapan paling mudah bila hanya ada satu benda yang perlu disalin. Java sudah memilikinya sejak tahun-tahun pertamanya: **berkas JAR**, arsip terkompresi berisi kelas-kelas terkompilasi dengan manifes kecil yang menyebut kelas yang `main`-nya dijalankan pertama; `java -jar cake-doctor.jar` lalu menjalankan program di komputer mana pun yang punya *runtime* Java [1]. Python punya gagasan yang sama dalam **zipapp**: arsip ZIP berisi berkas-berkas Python dengan `__main__.py` di tingkat teratasnya, yang dijalankan Python secara langsung dengan `python cake-doctor.pyz`; modul `zipapp` dari pustaka standar membangunnya [2]. Kedua berkas memuat aplikasi *dan* mesin vcirs-core, sehingga komputer toko tidak membutuhkan apa-apa selain Java atau Python dan berkas basis pengetahuannya.

Kita menaruh layanan dan ketiga alatnya — pemeriksaan penerapan, laporan pemakaian, dan alat pencadangan — dalam berkas yang sama, sebagai empat **perintah**: `serve`, `check`, `report`, dan `backup`. Satu berkas, satu nama perintah, empat tugas.

15.2.3 Basis pengetahuan adalah data: cadangan yang dapat dipercaya

Pelajaran Bu Wati adalah pelajaran tertua dalam komputasi: data yang hanya ada di satu tempat sebenarnya tidak ada. Nasihat yang lazim adalah **aturan 3-2-1**: simpan tiga salinan data penting, pada dua jenis media penyimpanan yang berbeda, dengan satu salinan di tempat lain [3]. Untuk basis pengetahuan berukuran beberapa ratus kilobita, hal ini tidak memakan biaya. Salinan di komputer layanan adalah yang pertama; cadangan tiap malam di *flashdisk* atau diska kedua

adalah yang kedua; salinan yang dikirim setiap minggu ke ponsel pemilik atau ke folder awan adalah yang ketiga.

Namun cadangan hanya berguna bila dapat dipulihkan, dan cukup banyak cadangan ternyata tidak dapat dipulihkan: salinannya diambil ketika berkas baru setengah ditulis, atau *flashdisk*-nya merusaknya diam-diam. Alat pencadangan kita mengambil tiga langkah berjaga-jaga. Ia menyalin basis pengetahuan hanya setelah berhasil memuatnya, sehingga berkas yang rusak tidak pernah dicadangkan menimpa cadangan yang baik. Ia menulis **sidik jari** setiap salinan — hash SHA-256, bilangan 64 karakter yang dihitung dari bita-bita berkas dan berubah sama sekali bila satu bita saja berubah — ke dalam sebuah berkas indeks. Dan ia punya perintah `check` yang membaca ulang setiap salinan, menghitung ulang sidik jarinya, dan memuatnya, sehingga cadangan yang rusak ditemukan pada hari kerusakannya, bukan pada hari cadangan itu dibutuhkan. Terakhir, `restore` menolak salinan yang rusak, dan menyimpan berkas yang digantinya, untuk berjaga-jaga.

15.2.4 Mengamati bagaimana pengetahuan dipakai

Laju pemakaian dari Bab 14 memberi tahu Bu Wati bagaimana pengetahuan itu *tersusun*: node mana yang menjadi sandaran node lain. Laju itu tidak memberi tahu bagaimana pengetahuan itu *dipakai*: saran mana yang benar-benar diterima para pembantu pada hari Sabtu, dan pertanyaan apa yang mereka ajukan yang tidak dapat dijawab basis pengetahuan. Untuk itu, layanan menyimpan **catatan pemakaian** (*usage log*): satu baris per konsultasi, berisi waktu, fakta-fakta yang diberikan, node yang menjawab (atau tidak ada), dan sarannya.

Catatan inilah tempat perawatan dimulai. Sebulan sekali sebuah alat kecil membacanya dan menulis **laporan pemakaian** dengan empat temuan: berapa konsultasi yang terjadi setiap bulan; berapa yang mendapat jawaban; node mana yang memberi saran, dari yang paling sering dipakai, dan node mana yang tidak pernah; dan kasus tak terjawab mana yang paling sering kembali. Daftar terakhir itulah yang paling berharga: secara harfiah, itu daftar kasus yang perlu diajarkan berikutnya, diurutkan menurut seberapa besar manfaatnya. Daftar node yang tidak pernah memberi saran berada di urutan kedua: node seperti itu mungkin baik-baik saja — sebagian masalah memang jarang — atau mungkin pengetahuan yang tidak lagi cocok dengan keadaan toko, dan layak ditengok.

15.2.5 Data pribadi dan hukum

Catatan pemakaian merekam apa yang ditanyakan orang. Itu menjadikannya bagian sistem pakar yang paling langsung terkena hukum tentang data pribadi, dan bagian tempat rancangan yang ceroboh dapat menimbulkan kerugian terbesar.

Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi (UU PDP) disahkan pada 17 Oktober 2022, dan memberi waktu dua tahun kepada organisasi untuk menyesuaikan diri, sampai Oktober 2024 [4], [5]. Undang-undang ini berlaku bagi siapa pun yang memproses data pribadi — data tentang orang yang teridentifikasi atau dapat diidentifikasi — termasuk usaha kecil, dan antara lain meminta agar data pribadi dikumpulkan secara terbatas dan spesifik untuk tujuan yang sah, diproses secara akurat dan aman, tidak disimpan lebih lama dari yang diperlukan, dan agar kegagalan pelindungan data diberitahukan kepada orang-orang yang terdampak paling lambat 3 × 24 jam [4], [6]. Peraturan pelaksanaannya, **Peraturan Pemerintah Nomor 33 Tahun 2026** (PP 33/2026), ditetapkan pada 16 Juli 2026, mulai beredar di publik pada akhir Agustus, dan mulai berlaku pada 16 Januari 2027, setelah masa transisi enam bulan; per akhir Agustus 2026, lembaga pelindungan data pribadi yang akan menegakkan undang-undang itu belum terbentuk [5]. Peraturan ini memberikan perincian operasional yang

dibiarkan terbuka oleh undang-undangnya, misalnya tentang persetujuan, hak-hak subjek data, dan pejabat pelindungan data pribadi [5], [6].

Apa artinya bagi dokter kue? Cara paling sederhana untuk mematuhi hukum tentang data pribadi adalah **tidak mengumpulkan data pribadi sama sekali**, dan catatan pemakaian dirancang seperti itu. Ia merekam waktu konsultasi, fakta-fakta tentang *kue* (`problem = dry`), node yang menjawab, dan sarannya. Ia tidak merekam nama, nomor telepon, alamat IP — server web layanan bahkan disuruh untuk tidak menulis catatan akses biasa yang memuat alamat — dan tidak merekam teks bebas yang dapat memuat salah satu dari itu. Catatan itu pun hanya disimpan selama berguna: alat laporan dapat menghapus baris-baris yang lebih tua dari sekian hari, dan menjalankannya setiap bulan dengan, misalnya, 180 hari menyimpan riwayat enam bulan dan tidak lebih.

Untuk sistem pakar tentang *manusia* — cek kesehatan mandiri, penasihat pinjaman, triase klinik — fakta-faktanya sendiri dapat berupa data pribadi, dan lebih banyak ketentuan hukum yang berlaku: tujuan yang jelas, persetujuan bila perlu, pembatasan akses, pembatasan masa simpan, dan rencana bila terjadi kebocoran. Subbab 15.9 kembali ke hal ini.

> **WAWASAN KUNCI:** Rancang catatan agar menjawab pertanyaan-pertanyaan perawatan Anda — saran mana yang dipakai, apa yang tidak dapat dijawab — dan tidak lebih. Data yang tidak pernah Anda kumpulkan tidak dapat bocor, tidak memerlukan persetujuan, dan tidak memerlukan biaya untuk dilindungi.

15.3 Langkah 1: Layanan Saran

Kode 15.1 adalah layanan dalam Python, hanya dengan pustaka standar agar dapat dikemas ke dalam berkas `.pyz` tanpa ketergantungan apa pun.

Kode 15.1 `advice_service.py`: dokter kue sebagai layanan saran kecil dengan catatan pemakaian yang menjaga privasi.

```
"""Dokter kue sebagai layanan saran kecil yang siap diterapkan (pustaka standar).

    python advice_service.py [--kb FILE] [--port 4151] [--host 127.0.0.1]
                             [--logs DIR] [--replay]

GET  /health     apakah layanan hidup, dan basis pengetahuan mana yang dipakai
GET  /questions  pertanyaan, yang paling berguna lebih dulu, dengan jawabannya
POST /advise     {"facts": {"problem": "dry"}} -> saran dan alasannya
Setiap konsultasi ditulis ke DIR/usage-YYYY-MM.jsonl: waktu, fakta, dan
jawaban, tidak ada apa pun tentang orangnya. Bila berkas basis pengetahuan
berubah, layanan memuatnya ulang sendiri. --replay membolehkan klien memberi
waktu setiap konsultasi dalam header X-Time (untuk catatan lama); permintaan
dengan header X-Test dijawab tetapi tidak ditulis ke catatan.
"""

import datetime
import json
import os
import pathlib
import sys
import threading
from http.server import BaseHTTPRequestHandler, ThreadingHTTPServer

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import KnowledgeBase, VcirsError                    # noqa: E402
from vcirs.jsonio import dumps                                 # noqa: E402

ENGINE = "vcirs-core 1.0.0"


class Service:
    """Semua kecuali HTTP: basis pengetahuan, saran, dan catatan."""

    def __init__(self, kb_path, log_dir):
        self.kb_path = kb_path
        self.log_dir = log_dir
        self.lock = threading.Lock()
        self.kb = KnowledgeBase.load(kb_path)
        self.stamp = os.path.getmtime(kb_path)
        os.makedirs(log_dir, exist_ok=True)
```

Kode 15.1 (Lanjutan) *advice_service.py: dokter kue sebagai layanan saran kecil dengan catatan pemakaian yang menjaga privasi.*

```
    def fresh_kb(self):
        """Muat ulang berkas bila berubah; pertahankan yang lama bila yang
        baru tak terbaca (berkas setengah tersalin, sebuah kekeliruan)."""
        stamp = os.path.getmtime(self.kb_path)
        if stamp != self.stamp:
            try:
                self.kb = KnowledgeBase.load(self.kb_path)
                print("reloaded " + self.kb_path, file=sys.stderr)
            except (OSError, ValueError, KeyError, VcirsError):
                print("could not reload " + self.kb_path + "; keeping the "
                      "old knowledge base", file=sys.stderr)
            self.stamp = stamp
        return self.kb

    def health(self):
        kb = self.fresh_kb()
        return {"status": "ok", "name": kb.name, "nodes": len(kb.nodes),
                "engine": ENGINE}

    def questions(self):
        kb = self.fresh_kb()
        used = [c.variable for n in kb.nodes.values() for c in n.conditions]
        order = kb.usage().question_order()
        order += [v for v in kb.variables if v in used and v not in order]
        return {"questions": [{"variable": v,
                               "question": kb.variables[v].question,
                               "choices": kb.known_values(v)}
                              for v in order]}

    def advise(self, facts, time, record=True):
        kb = self.fresh_kb()
        facts = {str(k).strip(): str(v).strip() for k, v in facts.items()
                 if str(k).strip() and str(v).strip()}
        result = kb.infer(facts)
        advice = [c.value for c in result.conclusions()]
        rule = result.paths[0].rule_id() if result.paths else None
        answer = {"advice": advice, "rule": rule,
                  "how": result.how() if advice else []}
        if record:                          # pemeriksaan bukan konsultasi
            self.log({"time": time, "rule": rule or "", "advice": advice,
                      "facts": facts})
        return answer
```

Kode 15.1 (Lanjutan) *advice_service.py: dokter kue sebagai layanan saran kecil dengan catatan pemakaian yang menjaga privasi.*

```
    def log(self, record):
        """Satu baris JSON per konsultasi, satu berkas per bulan."""
        name = "usage-" + record["time"][:7] + ".jsonl"
        with self.lock, open(os.path.join(self.log_dir, name), "a",
                             encoding="utf-8", newline="\n") as f:
            f.write(json.dumps(record, ensure_ascii=False) + "\n")


def now():
    return datetime.datetime.now(datetime.timezone.utc).strftime(
        "%Y-%m-%dT%H:%M:%SZ")


def make_handler(service, replay):
    class Handler(BaseHTTPRequestHandler):
        protocol_version = "HTTP/1.1"        # koneksi tetap terbuka

        def reply(self, status, value):
            body = (dumps(value) + "\n").encode("utf-8")
            self.send_response(status)
            self.send_header("Content-Type", "application/json; charset=utf-8")
            self.send_header("Content-Length", str(len(body)))
            self.end_headers()
            self.wfile.write(body)

        def do_GET(self):
            if self.path == "/health":
                self.reply(200, service.health())
            elif self.path == "/questions":
                self.reply(200, service.questions())
            else:
                self.reply(404, {"error": "no such page"})

        def do_POST(self):
            size = int(self.headers.get("Content-Length", "0"))
            text = self.rfile.read(size).decode("utf-8", "replace")
            if self.path != "/advise":
                self.reply(404, {"error": "no such page"})
                return
            try:
                data = json.loads(text)
                facts = data["facts"]
                if not isinstance(facts, dict):
                    raise ValueError
            except (ValueError, KeyError, TypeError):
                self.reply(400, {"error": 'send {"facts": {"name": "value"}}'})
```

Kode 15.1 (Lanjutan) *advice_service.py: dokter kue sebagai layanan saran kecil dengan catatan pemakaian yang menjaga privasi.*

```
                return
            time = self.headers.get("X-Time") if replay else None
            test = self.headers.get("X-Test") is not None
            self.reply(200, service.advise(facts, time or now(), not test))

        def log_message(self, fmt, *args):   # tak ada alamat di catatan
            pass

    return Handler


def main(args):
    def option(name, default):
        return args[args.index(name) + 1] if name in args else default

    here = os.path.dirname(os.path.abspath(sys.argv[0]))
    kb_path = option("--kb", os.path.join(here, "..", "data", "cake-kb.json"))
    service = Service(kb_path, option("--logs", "logs"))
    host, port = option("--host", "127.0.0.1"), int(option("--port", "4151"))
    server = ThreadingHTTPServer((host, port),
                                 make_handler(service, "--replay" in args))
    print("Advice service for '" + service.kb.name + "' on http://" + host
          + ":" + str(port) + "; Ctrl+C stops it.", flush=True)
    try:
        server.serve_forever()
    except KeyboardInterrupt:
        pass


if __name__ == "__main__":
    main(sys.argv[1:])
```

Kelas `Service` memegang semuanya kecuali HTTP, sehingga ia tetap dapat diuji tanpa jaringan.

Ia memuat basis pengetahuan, dan memuatnya lagi ketika berubah. Konstruktornya memuat berkas dan mengingat waktu perubahannya. `fresh_kb`, yang dipanggil di awal setiap permintaan, membandingkan waktu perubahan berkas dengan yang diingatnya; bila berubah, berkas dimuat ulang. Bila berkas baru tidak dapat dibaca — mungkin masih disalin, atau seseorang menyalin berkas yang salah — layanan mempertahankan basis pengetahuan yang lama dan menulis peringatan ke keluaran galat. Toko tidak pernah mendapati layanannya berhenti menjawab karena salinan yang buruk.

Ia menjawab. `health` mengembalikan status, nama basis pengetahuan, jumlah node-nya, dan versi mesin, empat hal yang ingin Anda ketahui ketika memeriksa sebuah penerapan. `questions` mengembalikan urutan pertanyaan dari Bab 14 — variabel-variabel yang dipakai dalam kondisi, yang paling berguna lebih dulu — beserta teks pertanyaan dan jawaban-jawaban yang dikenal. `advise` membersihkan fakta-fakta yang diberikan (nama dan nilai dipangkas, yang kosong dibuang), menjalankan `infer`, dan mengembalikan saran, rule yang memberikannya, dan penjelasannya.

Ia mencatat, dengan hati-hati. `log` menambahkan satu baris JSON ke berkas yang dinamai menurut bulan konsultasinya, `usage-2026-09.jsonl`; satu berkas per bulan memudahkan penghapusan bulan-bulan lama. Sebuah kunci (*lock*) memastikan dua permintaan yang datang bersamaan tidak pernah saling menyisipkan setengah baris. Konsultasi yang permintaannya membawa header `X-Test` dijawab tetapi tidak dicatat: pemeriksaan setelah penerapan bukanlah pemakaian sungguhan, dan tidak boleh muncul dalam statistik.

Bagian HTTP-nya adalah `make_handler`. Ia memakai `ThreadingHTTPServer` dari pustaka standar dengan `protocol_version = "HTTP/1.1"`, membaca badan permintaan lebih dulu (agar jalur yang tidak dikenal tidak meninggalkan data yang belum terbaca pada koneksi yang tetap terbuka), menjawab 400 dengan penjelasan singkat bila badan permintaan bukan `{"facts": {...}}`, dan mengganti `log_message`, metode yang biasanya mencetak setiap permintaan beserta alamat IP pemanggilnya, dengan metode yang tidak mencetak apa-apa. `--replay` untuk satu tugas khusus: memasukkan konsultasi yang terjadi lebih awal, seperti catatan kertas tiga bulan di Subbab 15.7, dengan tanggal aslinya dalam header `X-Time`. Tanpa `--replay`, layanan mengabaikan `X-Time` dan memakai jamnya sendiri, sehingga tidak ada yang dapat memalsukan tanggal di catatan.

Ia hanya mendengarkan di komputer ini kecuali disuruh lain: `--host 127.0.0.1` adalah bawaannya. Untuk melayani tablet-tablet di toko, jalankan dengan `--host 0.0.0.0`, yang mendengarkan di setiap jaringan tempat komputer itu tersambung — lakukan itu hanya di Wi-Fi milik toko sendiri, tidak pernah di komputer yang tersambung langsung ke internet, karena layanan ini tidak punya kata sandi.

Kode 15.2 adalah layanan yang sama dalam Java, dengan `HttpServer` milik JDK.

Kode 15.2 `AdviceService.java`: layanan yang sama dalam Java.

```
import java.io.IOException;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardOpenOption;
import java.nio.file.attribute.FileTime;
import java.time.Instant;
import java.time.ZoneOffset;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpServer;

import vcirs.Conclusion;
import vcirs.Facts;
import vcirs.InferenceResult;
import vcirs.Json;
import vcirs.KnowledgeBase;
import vcirs.Node;
import vcirs.VcirsException;

/**
 * Dokter kue sebagai layanan saran kecil yang siap diterapkan (hanya JDK).
 * java AdviceService [--kb FILE] [--port 4151] [--host 127.0.0.1]
 *                    [--logs DIR] [--replay]
 * GET /health, GET /questions, POST /advise {"facts": {"problem": "dry"}}.
 * Setiap konsultasi ditulis ke DIR/usage-YYYY-MM.jsonl: waktu, fakta, dan
 * jawaban, tidak ada apa pun tentang orangnya. Bila berkas basis pengetahuan
 * berubah, layanan memuatnya ulang sendiri. --replay membolehkan klien memberi
 * waktu setiap konsultasi dalam header X-Time (untuk catatan lama);
 * permintaan dengan header X-Test dijawab tetapi tidak ditulis ke catatan.
 */
public class AdviceService {
    static final String ENGINE = "vcirs-core 1.0.0";
    static final DateTimeFormatter TIME = DateTimeFormatter
            .ofPattern("yyyy-MM-dd'T'HH:mm:ss'Z'").withZone(ZoneOffset.UTC);
```

Kode 15.2 (Lanjutan) *AdviceService.java: layanan yang sama dalam Java.*

```
    final Path kbPath;
    final Path logDir;
    KnowledgeBase kb;
    FileTime stamp;

    AdviceService(Path kbPath, Path logDir) throws IOException {
        this.kbPath = kbPath;
        this.logDir = logDir;
        kb = KnowledgeBase.load(kbPath.toString());
        stamp = Files.getLastModifiedTime(kbPath);
        Files.createDirectories(logDir);
    }

    /**
     * Muat ulang berkas bila berubah; pertahankan yang lama bila yang baru
     * tak terbaca (berkas setengah tersalin, sebuah kekeliruan).
     */
    synchronized KnowledgeBase freshKb() throws IOException {
        FileTime now = Files.getLastModifiedTime(kbPath);
        if (!now.equals(stamp)) {
            try {
                kb = KnowledgeBase.load(kbPath.toString());
                System.err.println("reloaded " + kbPath);
            } catch (RuntimeException e) {
                System.err.println("could not reload " + kbPath
                        + "; keeping the old knowledge base");
            }
            stamp = now;
        }
        return kb;
    }

    Map<String, Object> health() throws IOException {
        KnowledgeBase k = freshKb();
        Map<String, Object> out = new LinkedHashMap<>();
        out.put("status", "ok");
        out.put("name", k.name);
        out.put("nodes", k.nodes.size());
        out.put("engine", ENGINE);
        return out;
    }
```

Kode 15.2 (Lanjutan) *AdviceService.java: layanan yang sama dalam Java.*

```
    Map<String, Object> questions() throws IOException {
        KnowledgeBase k = freshKb();
        List<String> used = new ArrayList<>();
        for (Node n : k.nodes.values()) {
            n.conditions.forEach(c -> used.add(c.variable));
        }
        List<String> order = new ArrayList<>(k.usage().questionOrder());
        for (String v : k.variables.keySet()) {
            if (used.contains(v) && !order.contains(v)) {
                order.add(v);
            }
        }
        List<Object> list = new ArrayList<>();
        for (String v : order) {
            Map<String, Object> q = new LinkedHashMap<>();
            q.put("variable", v);
            q.put("question", k.variables.get(v).question);
            q.put("choices", new ArrayList<Object>(k.knownValues(v)));
            list.add(q);
        }
        return Map.of("questions", list);
    }

    Map<String, Object> advise(Map<?, ?> given, String time, boolean record)
            throws IOException {
        KnowledgeBase k = freshKb();
        Map<String, String> facts = new LinkedHashMap<>();
        given.forEach((name, value) -> {
            String n = String.valueOf(name).strip();
            String v = String.valueOf(value).strip();
            if (!n.isEmpty() && !v.isEmpty()) {
                facts.put(n, v);
            }
        });
        InferenceResult result = k.infer(Facts.map(facts));
        List<Object> advice = new ArrayList<>();
        for (Conclusion c : result.conclusions()) {
            advice.add(c.value);
        }
        String rule = result.paths.isEmpty() ? null
                : result.paths.get(0).ruleId();
        Map<String, Object> answer = new LinkedHashMap<>();
        answer.put("advice", advice);
        answer.put("rule", rule);
        answer.put("how", advice.isEmpty() ? List.of()
                : new ArrayList<Object>(result.how()));
```

Kode 15.2 (Lanjutan) *AdviceService.java: layanan yang sama dalam Java.*

```
        if (record) {                       // pemeriksaan bukan konsultasi
            Map<String, Object> line = new LinkedHashMap<>();
            line.put("time", time);
            line.put("rule", rule == null ? "" : rule);
            line.put("advice", advice);
            line.put("facts", facts);
            log(line);
        }
        return answer;
    }

    /** Satu baris JSON per konsultasi, satu berkas per bulan. */
    synchronized void log(Map<String, Object> line) throws IOException {
        String time = (String) line.get("time");
        Path file = logDir.resolve("usage-" + time.substring(0, 7) + ".jsonl");
        Files.writeString(file, compact(line) + "\n", StandardCharsets.UTF_8,
                StandardOpenOption.CREATE, StandardOpenOption.APPEND);
    }

    /** JSON dalam satu baris, persis seperti ditulis json.dumps Python. */
    static String compact(Object value) {
        if (value instanceof Map<?, ?> map) {
            List<String> parts = new ArrayList<>();
            map.forEach((k, v) -> parts.add(Json.quote(k.toString()) + ": "
                    + compact(v)));
            return "{" + String.join(", ", parts) + "}";
        }
        if (value instanceof List<?> list) {
            List<String> parts = new ArrayList<>();
            list.forEach(v -> parts.add(compact(v)));
            return "[" + String.join(", ", parts) + "]";
        }
        return value == null ? "null" : Json.quote(value.toString());
    }

    void reply(HttpExchange ex, int status, Object value) throws IOException {
        byte[] body = (Json.write(value) + "\n").getBytes(StandardCharsets.UTF_8);
        ex.getResponseHeaders().set("Content-Type",
                "application/json; charset=utf-8");
        ex.sendResponseHeaders(status, body.length);
        try (OutputStream out = ex.getResponseBody()) {
            out.write(body);
        }
    }
```

Kode 15.2 (Lanjutan) *AdviceService.java: layanan yang sama dalam Java.*

```
    void handle(HttpExchange ex, boolean replay) throws IOException {
        String what = ex.getRequestMethod() + " " + ex.getRequestURI().getPath();
        switch (what) {
            case "GET /health" -> reply(ex, 200, health());
            case "GET /questions" -> reply(ex, 200, questions());
            case "POST /advise" -> {
                Object facts;
                try {
                    Object data = Json.parse(new String(
                            ex.getRequestBody().readAllBytes(),
                            StandardCharsets.UTF_8));
                    facts = data instanceof Map<?, ?> m ? m.get("facts") : null;
                } catch (VcirsException e) {
                    facts = null;
                }
                if (!(facts instanceof Map<?, ?> given)) {
                    reply(ex, 400, Map.of("error",
                            "send {\"facts\": {\"name\": \"value\"}}"));
                    return;
                }
                String time = replay
                        ? ex.getRequestHeaders().getFirst("X-Time") : null;
                boolean test = ex.getRequestHeaders().containsKey("X-Test");
                reply(ex, 200, advise(given, time == null
                        ? TIME.format(Instant.now()) : time, !test));
            }
            default -> reply(ex, 404, Map.of("error", "no such page"));
        }
    }

    public static void main(String[] args) throws IOException {
        Map<String, String> opt = new LinkedHashMap<>();
        for (int i = 0; i < args.length; i++) {
            if (args[i].startsWith("--") && i + 1 < args.length
                    && !args[i + 1].startsWith("--")) {
                opt.put(args[i], args[++i]);
            } else {
                opt.put(args[i], "");
            }
        }
        AdviceService service = new AdviceService(
                Path.of(opt.getOrDefault("--kb", "../data/cake-kb.json")),
                Path.of(opt.getOrDefault("--logs", "logs")));
        String host = opt.getOrDefault("--host", "127.0.0.1");
        int port = Integer.parseInt(opt.getOrDefault("--port", "4151"));
        boolean replay = opt.containsKey("--replay");
```

Kode 15.2 (Lanjutan) *AdviceService.java: layanan yang sama dalam Java.*

```
        HttpServer server = HttpServer.create(new InetSocketAddress(host, port), 0);
        server.createContext("/", ex -> service.handle(ex, replay));
        server.start();
        System.out.println("Advice service for '" + service.kb.name + "' on http://"
                + host + ":" + port + "; Ctrl+C stops it.");
    }
}
```

Dua perincian membuat baris-baris catatan Java identik dengan yang dari Python. Badan setiap jawaban ditulis dengan `Json.write`, pencetak kanonik yang sama dengan `dumps` Python dari vcirs-core. Baris-baris catatan, yang harus muat dalam satu baris, ditulis oleh `compact`, yang meniru persis apa yang ditulis `json.dumps` Python: `{"time": "...", "rule": "...", ...}`, dengan satu spasi setelah setiap titik dua dan koma. Waktu diformat sebagai `2026-09-25T08:00:00Z` di kedua

bahasa, selalu dalam UTC, sehingga catatan dari toko di Makassar dan toko di Medan dapat dibandingkan tanpa memikirkan zona waktu.

15.4 Langkah 2: Mengemas dan Menerapkan

Satu program, empat perintah. **Kode 15.3** adalah titik masuk yang dijalankan berkas `.pyz`.

Kode 15.3 `cake_doctor.py`: satu program dengan empat perintah, titik masuk berkas `.pyz`.

```
"""Satu program dengan empat perintah, agar cukup satu berkas diterapkan.

    python cake-doctor.pyz serve  [options of advice_service]
    python cake-doctor.pyz check  [options of service_check]
    python cake-doctor.pyz report [options of usage_report]
    python cake-doctor.pyz backup [options of backup]
"""

import sys

import advice_service
import backup
import service_check
import usage_report

COMMANDS = {"serve": advice_service.main, "check": service_check.main,
            "report": usage_report.main, "backup": backup.main}


def main(args):
    if not args or args[0] not in COMMANDS:
        print("usage: cake-doctor serve|check|report|backup [options]")
        sys.exit(1)
    COMMANDS[args[0]](args[1:])


if __name__ == "__main__":
    main(sys.argv[1:])
```

Kode 15.4 adalah titik masuk Java yang disebut berkas `.jar` dalam manifesnya.

Kode 15.4 `CakeDoctor.java`: keempat perintah yang sama dalam Java.

```
import java.util.Arrays;

/**
 * Satu program dengan empat perintah, agar cukup satu berkas diterapkan.
 * java -jar cake-doctor.jar serve|check|report|backup [options]
 */
public class CakeDoctor {
    public static void main(String[] args) throws Exception {
        String[] rest = args.length > 0
                ? Arrays.copyOfRange(args, 1, args.length) : args;
        switch (args.length > 0 ? args[0] : "") {
            case "serve" -> AdviceService.main(rest);
            case "check" -> ServiceCheck.main(rest);
            case "report" -> UsageReport.main(rest);
            case "backup" -> Backup.main(rest);
            default -> {
                System.out.println("usage: cake-doctor serve|check|report|backup "
                        + "[options]");
                System.exit(1);
            }
        }
    }
}
```

Membangun `.pyz`. Kode 15.5 menyalin keempat berkas Python dan paket vcirs-core ke sebuah folder sementara, menaruh `cake_doctor.py` di sana sebagai `__main__.py`, dan meminta `zipapp` mengemas folder itu menjadi `dist/cake-doctor.pyz`, terkompresi, dengan baris pertama yang memungkinkan Linux dan macOS menjalankannya secara langsung [2].

Kode 15.5 `build_pyz.py`: mengemas layanan, alat-alatnya, dan vcirs-core menjadi satu berkas `.pyz`.

```
"""Bangun ../dist/cake-doctor.pyz: layanan, alat-alatnya, dan vcirs-core
dalam satu berkas yang dapat dijalankan Python 3.10 atau lebih baru.

    python build_pyz.py
"""

import os
import pathlib
import shutil
import tempfile
import zipapp

HERE = pathlib.Path(__file__).resolve().parent
ENGINE = HERE.parents[1] / "vcirs-core" / "python" / "vcirs"
DIST = HERE.parent / "dist"
FILES = ["advice_service.py", "backup.py", "service_check.py",
         "usage_report.py"]


def main():
    with tempfile.TemporaryDirectory() as tmp:
        root = pathlib.Path(tmp)
        for name in FILES:
            shutil.copy(HERE / name, root / name)
        shutil.copy(HERE / "cake_doctor.py", root / "__main__.py")
        shutil.copytree(ENGINE, root / "vcirs",
                        ignore=shutil.ignore_patterns("__pycache__"))
        DIST.mkdir(exist_ok=True)
        target = DIST / "cake-doctor.pyz"
        zipapp.create_archive(root, target, interpreter="/usr/bin/env python3",
                              compressed=True)
    print("built " + os.path.relpath(target, HERE.parent) + " ("
          + str(target.stat().st_size // 1024) + " KB)")


if __name__ == "__main__":
    main()
```

Menjalankan `python build_pyz.py` di `Chapter15/python` mencetak:

```
built dist/cake-doctor.pyz (34 KB)
```

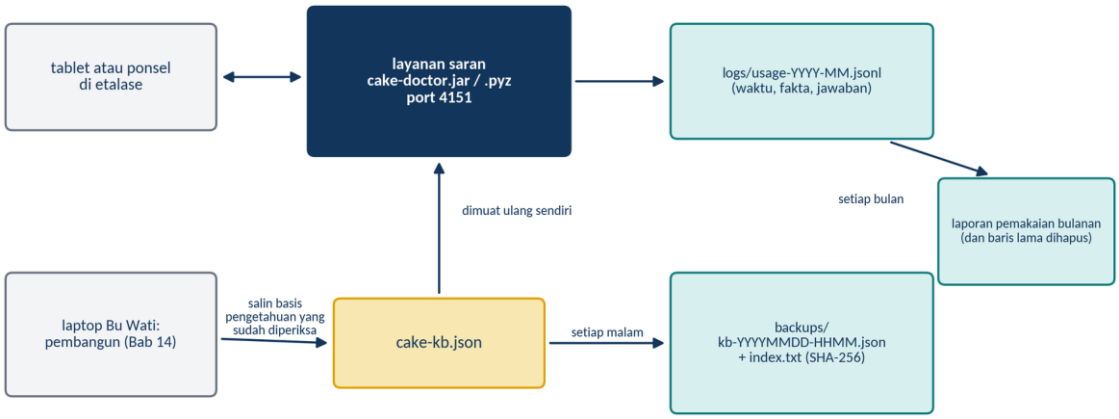
Tiga puluh empat kilobita: seluruh dokter kue, termasuk mesinnya, lebih kecil daripada satu foto kue.

Membangun `.jar`. Java tidak memerlukan skrip; alat `jar` milik JDK sudah mengerjakannya [1]. Dari `Chapter15/java`:

```
javac -d build -sourcepath ../../vcirs-core/java/src CakeDoctor.java \
      AdviceService.java ServiceCheck.java UsageReport.java Backup.java
jar --create --file ../dist/cake-doctor.jar --main-class CakeDoctor -C build .
```

`javac` mengompilasi kelima kelas dan, melalui `-sourcepath`, tepat kelas-kelas mesin yang dipakainya; `jar --create` mengemas folder hasil kompilasi dan menulis `Main-Class: CakeDoctor` ke dalam manifes. (Di Windows, tulis perintah pertama dalam satu baris tanpa `\`.) Hasilnya sekitar 94 kilobita.

Gambar 15.1 memperlihatkan seperti apa penerapan yang sudah jadi di toko Bu Wati.



Gambar 15.1 Dokter kue yang sudah diterapkan. Layanan berjalan dari satu berkas di komputer toko dan menjawab tablet atau ponsel di etalase; ia memuat ulang basis pengetahuan ketika salinan yang sudah diperiksa datang dari laptop Bu Wati, menulis catatan pemakaian, dicadangkan setiap malam, dan dilaporkan setiap bulan.

Memasangnya. Di komputer toko, buat sebuah folder — `C:\cake-doctor` di Windows, `/opt/cake-doctor` di Linux, sebuah folder di folder rumah Anda di macOS — lalu taruh di dalamnya berkas `.jar` (atau `.pyz`), folder `data` berisi `cake-kb.json`, serta folder `logs` dan `backups` yang kosong. Komputer itu membutuhkan Java 17 atau lebih baru (Bab 1 memasang JDK 25) atau Python 3.10 atau lebih baru (Bab 1 memasang Python 3.14). Ujilah dulu secara manual: `java -jar cake-doctor.jar serve --kb data/cake-kb.json --logs logs`, lalu buka `http://localhost:4151/health` di peramban.

Kemudian buatlah agar ia berjalan sendiri. Folder `deploy` bab ini memuat berkas siap pakai untuk setiap sistem.

Di **Linux**, pengelola layanannya adalah **systemd**, dan sebuah layanan dijelaskan dengan *unit file* kecil [7]:

```
# Unit systemd untuk Linux: salin ke /etc/systemd/system/cake-doctor.service,
# lalu: sudo systemctl daemon-reload && sudo systemctl enable --now cake-doctor
[Unit]
Description=Cake doctor advice service (VCIRS)
After=network.target

[Service]
User=cakedoctor
WorkingDirectory=/opt/cake-doctor
ExecStart=/usr/bin/java -jar /opt/cake-doctor/cake-doctor.jar serve \
    --kb /opt/cake-doctor/data/cake-kb.json \
    --logs /opt/cake-doctor/logs --port 4151
# Python sebagai ganti Java: baris yang sama dengan
# /usr/bin/python3 /opt/cake-doctor/cake-doctor.pyz sebagai ganti java -jar ...
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

Salin ke `/etc/systemd/system/cake-doctor.service`, buat pengguna `cakedoctor` yang memiliki folder itu (`sudo useradd --system cakedoctor`, `sudo chown -R cakedoctor /opt/cake-doctor`), lalu jalankan `sudo systemctl daemon-reload` dan `sudo systemctl enable --now cake-doctor`. `Restart=on-failure` menyalakan ulang layanan bila suatu saat berhenti mendadak; `sudo systemctl status cake-doctor` menunjukkan apakah ia berjalan, dan `journalctl -u cake-doctor` menampilkan pesan-pesannya, termasuk peringatan "could not reload". Menjalankannya sebagai pengguna tersendiri, bukan sebagai `root`, berarti kekeliruan dalam layanan pun tidak dapat menyentuh bagian lain komputer.

Di **Windows**, `start-windows.cmd` menjalankan layanan dari `C:\cake-doctor`:

```
@echo off
rem Menjalankan dokter kue di Windows. Taruh berkas ini, cake-doctor.jar (atau
rem cake-doctor.pyz), dan folder data di C:\cake-doctor. Agar ikut berjalan
rem bersama Windows, tambahkan di Task Scheduler: Create Task > Triggers:
rem At startup.
cd /d C:\cake-doctor
java -jar cake-doctor.jar serve --kb data\cake-kb.json --logs logs --port 4151
rem Python sebagai ganti Java:
rem python cake-doctor.pyz serve --kb data\cake-kb.json --logs logs --port 4151
```

Buka **Task Scheduler**, pilih *Create Task*, beri nama, tambahkan pemicu *At startup*, dan tindakan *Start a program* dengan `C:\cake-doctor\start-windows.cmd`; di bagian *General*, pilih *Run whether user is logged on or not*. Pertama kali Java membuka port jaringan, Windows Firewall mungkin meminta izin: izinkan hanya untuk jaringan *private*.

Di **macOS**, padanan *unit file* adalah *launch agent*: berkas `.plist` kecil di `~/Library/LaunchAgents` yang menyebut program beserta argumennya dan mengisi `RunAtLoad` dan `KeepAlive` dengan *true*; `launchctl load` mengaktifkannya. Soal 15.6 meminta Anda menulisnya.

Cadangan dan laporan yang terjadwal. Di Linux dan macOS, dua skrip mungil mengerjakan kedua tugas rutin. `backup.sh` membuat cadangan malam hari:

```
#!/bin/sh
# Pencadangan malam (Linux, macOS): simpan 30 salinan terbaru.
cd /opt/cake-doctor || exit 1
java -jar cake-doctor.jar backup save data/cake-kb.json backups --keep 30
```

dan `report.sh` menulis laporan bulanan ke berkas yang dinamai menurut bulannya, sambil menghapus baris-baris catatan yang lebih tua dari 180 hari:

```
#!/bin/sh
# Laporan bulanan; juga menghapus baris catatan lebih tua dari 180 hari.
cd /opt/cake-doctor || exit 1
mkdir -p reports
java -jar cake-doctor.jar report logs --kb data/cake-kb.json \
    --keep-days 180 > "reports/$(date +%Y-%m).txt"
```

Salin kedua skrip ke folder layanan, dan dua baris yang ditambahkan dengan `crontab -e` menjalankannya pada waktu yang tepat:

```
# crontab -e (Linux, macOS): pencadangan setiap malam pukul 21.30, dan
# laporan pukul 7.00 pada hari pertama setiap bulan.
30 21 * * * /opt/cake-doctor/backup.sh
0 7 1 * *   /opt/cake-doctor/report.sh
```

Lima kolom sebelum setiap perintah adalah menit, jam, tanggal, bulan, dan hari dalam minggu, dan `*` berarti “apa saja”: pencadangan berjalan setiap hari pukul 21.30, laporan pukul 7.00 pada hari pertama setiap bulan. Di Windows, `backup-windows.cmd` di folder yang sama mengerjakan pencadangan dan dijadwalkan di Task Scheduler dengan pemicu *Daily*. Ingat dua salinan lain dari aturan 3-2-1: folder cadangan sebaiknya berada di *flashdisk* atau diska kedua, dan seminggu sekali satu cadangan dibawa ke tempat lain.

15.5 Langkah 3: Memeriksa Penerapan

Setiap kali layanan dipasang, diperbarui, atau dipindahkan, periksalah. Pemeriksaan yang memakan sepuluh detik dan selalu sama menangkap kekeliruan yang mudah dibuat dan sulit disadari: basis pengetahuan yang salah tersalin, versi lama sebuah berkas, *firewall* yang menutup port, Java yang terlalu tua. **Kode 15.6** adalah pemeriksaannya.

Kode 15.6 `service_check.py`: pemeriksaan setelah setiap penerapan, dan pemutaran ulang konsultasi yang tercatat.

```
"""Periksa layanan saran yang berjalan, seperti setelah setiap penerapan.

    python service_check.py [URL] [--replay visits.txt]

Tanpa --replay ia menanyakan /health, /questions, dan dua saran, lalu
mengatakan apakah semuanya menjawab seperti diharapkan. Dengan --replay ia
mengirim setiap baris "waktu | nama = nilai; nama = nilai" berkas ke /advise.
"""

import json
import sys
import urllib.request

OPENER = urllib.request.build_opener(urllib.request.ProxyHandler({}))


def call(url, body=None, time=None):
    data = None if body is None else json.dumps(body).encode("utf-8")
    request = urllib.request.Request(url, data=data)
    request.add_header("Content-Type", "application/json")
    if time:
        request.add_header("X-Time", time)
    else:
        request.add_header("X-Test", "yes")        # pemeriksaan tak dihitung
    with OPENER.open(request, timeout=10) as response:
        return json.loads(response.read().decode("utf-8"))


def facts_of(text):
    """'problem = dry; baking time = long' -> {'problem': 'dry', ...}"""
    out = {}
    for part in text.split(";"):
        name, value = part.split("=")
        out[name.strip()] = value.strip()
    return out


def advise(url, text):
    answer = call(url + "/advise", {"facts": facts_of(text)})
    if not answer["advice"]:
        print("advise " + text + " -> (no answer yet)")
    else:
        print("advise " + text + " -> " + ", ".join(answer["advice"])
              + " (rule " + answer["rule"] + ")")
    return answer
```

Kode 15.6 (Lanjutan) *service_check.py: pemeriksaan setelah setiap penerapan, dan pemutaran ulang konsultasi yang tercatat.*

```
def main(args):
    urls = [a for a in args if a.startswith("http")]
    url = urls[0] if urls else "http://localhost:4151"
    if "--replay" in args:
        path = args[args.index("--replay") + 1]
        count = 0
        with open(path, encoding="utf-8") as f:
            for line in f:
                if "|" in line:
                    time, text = line.split("|", 1)
                    call(url + "/advise", {"facts": facts_of(text)},
                         time.strip())
                    count += 1
        print("replayed " + str(count) + " consultations to " + url)
        return
    print("Checking " + url)
    try:
        health = call(url + "/health")
    except OSError:
        print("FAIL: the service does not answer")
        sys.exit(1)
    print("health: " + health["status"] + ", " + health["name"] + ", "
          + str(health["nodes"]) + " nodes, " + health["engine"])
    names = [q["variable"] for q in call(url + "/questions")["questions"]]
    print("questions: " + ", ".join(names))
    good = advise(url, "problem = dry; baking time = long")
    none = advise(url, "problem = sticky top")
    ok = good["advice"] == ["take it out earlier"] and not none["advice"]
    print("all checks passed" if ok else "FAIL: unexpected advice")
    if not ok:
        sys.exit(1)


if __name__ == "__main__":
    main(sys.argv[1:])
```

Ia menanyakan `/health` dan `/questions`, lalu dua saran yang jawabannya sudah diketahui: kue kering yang dipanggang terlalu lama harus mendapat *take it out earlier*, dan bagian atas yang lengket, yang belum pernah diajarkan kepada basis pengetahuan, tidak boleh mendapat jawaban. Uji kedua sama pentingnya dengan yang pertama. Penerapan yang tiba-tiba memberi saran untuk bagian atas yang lengket sedang menjalankan basis pengetahuan yang salah. Setiap permintaan pemeriksaan membawa `X-Test`, sehingga tidak ada yang muncul di catatan pemakaian. Bila ada yang salah, program berkata `FAIL` dan berakhir dengan kode keluar 1, yang dapat dideteksi skrip atau tugas terjadwal.

Kode 15.7 adalah pemeriksaan yang sama dalam Java.

Kode 15.7 `ServiceCheck.java`: pemeriksaan yang sama dalam Java.

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.time.Duration;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

import vcirs.Json;

/**
 * Periksa layanan saran yang berjalan, seperti setelah setiap penerapan.
 * java ServiceCheck [URL] [--replay visits.txt]
 * Tanpa --replay ia menanyakan /health, /questions, dan dua saran, lalu
 * mengatakan apakah semuanya menjawab seperti diharapkan. Dengan --replay ia
 * mengirim setiap baris "waktu | nama = nilai; nama = nilai" ke /advise.
 */
public class ServiceCheck {
    static final HttpClient HTTP = HttpClient.newBuilder()
            .proxy(HttpClient.Builder.NO_PROXY).build();

    @SuppressWarnings("unchecked")
    static Map<String, Object> call(String url, Object body, String time)
            throws IOException, InterruptedException {
        HttpRequest.Builder r = HttpRequest.newBuilder(URI.create(url))
                .timeout(Duration.ofSeconds(10))
                .header("Content-Type", "application/json");
        if (time != null) {
            r.header("X-Time", time);
        } else {
            r.header("X-Test", "yes");              // pemeriksaan tak dihitung
        }
        r = body == null ? r.GET() : r.POST(HttpRequest.BodyPublishers
                .ofString(Json.write(body), StandardCharsets.UTF_8));
        HttpResponse<String> response = HTTP.send(r.build(),
                HttpResponse.BodyHandlers.ofString(StandardCharsets.UTF_8));
        return (Map<String, Object>) Json.parse(response.body());
    }
```

Kode 15.7 (Lanjutan) *ServiceCheck.java: pemeriksaan yang sama dalam Java.*

```
    /** "problem = dry; baking time = long" -> {problem=dry, ...} */
    static Map<String, Object> factsOf(String text) {
        Map<String, Object> out = new LinkedHashMap<>();
        for (String part : text.split(";")) {
            String[] kv = part.split("=");
            out.put(kv[0].strip(), kv[1].strip());
        }
        return out;
    }

    @SuppressWarnings("unchecked")
    static List<Object> advise(String url, String text)
            throws IOException, InterruptedException {
        Map<String, Object> answer = call(url + "/advise",
                Map.of("facts", factsOf(text)), null);
        List<Object> advice = (List<Object>) answer.get("advice");
        if (advice.isEmpty()) {
            System.out.println("advise " + text + " -> (no answer yet)");
        } else {
            System.out.println("advise " + text + " -> "
                    + String.join(", ", advice.stream().map(String::valueOf)
                            .toList()) + " (rule " + answer.get("rule") + ")");
        }
        return advice;
    }

    @SuppressWarnings("unchecked")
    public static void main(String[] args) throws Exception {
        String url = "http://localhost:4151";
        String replay = null;
        for (int i = 0; i < args.length; i++) {
            if (args[i].equals("--replay") && i + 1 < args.length) {
                replay = args[++i];
            } else if (args[i].startsWith("http")) {
                url = args[i];
            }
        }
        if (replay != null) {
            int count = 0;
            for (String line : Files.readAllLines(Path.of(replay),
                    StandardCharsets.UTF_8)) {
                int bar = line.indexOf('|');
                if (bar >= 0) {
                    call(url + "/advise", Map.of("facts",
                            factsOf(line.substring(bar + 1))),
                            line.substring(0, bar).strip());
```

Kode 15.7 (Lanjutan) *ServiceCheck.java: pemeriksaan yang sama dalam Java.*

```
                    count++;
                }
            }
            System.out.println("replayed " + count + " consultations to " + url);
            return;
        }
        System.out.println("Checking " + url);
        Map<String, Object> health;
        try {
            health = call(url + "/health", null, null);
        } catch (IOException e) {
            System.out.println("FAIL: the service does not answer");
            System.exit(1);
            return;
        }
        System.out.println("health: " + health.get("status") + ", "
                + health.get("name") + ", " + health.get("nodes") + " nodes, "
                + health.get("engine"));
        List<String> names = new ArrayList<>();
        for (Object q : (List<Object>) call(url + "/questions", null, null)
                .get("questions")) {
            names.add((String) ((Map<String, Object>) q).get("variable"));
        }
        System.out.println("questions: " + String.join(", ", names));
        List<Object> good = advise(url, "problem = dry; baking time = long");
        List<Object> none = advise(url, "problem = sticky top");
        boolean ok = good.equals(List.of("take it out earlier")) && none.isEmpty();
        System.out.println(ok ? "all checks passed" : "FAIL: unexpected advice");
        if (!ok) {
            System.exit(1);
        }
    }
}
```

Inilah penerapan di folder toko Bu Wati, dijalankan dari `.jar` dengan `java -jar cake-doctor.jar serve --kb data/cake-kb.json --logs logs --replay`, yang mencetak:

```
Advice service for 'Cake doctor' on http://127.0.0.1:4151; Ctrl+C stops it.
```

dan diperiksa dari terminal lain dengan `.pyz`, `python cake-doctor.pyz check`:

```
Checking http://localhost:4151
health: ok, Cake doctor, 12 nodes, vcirs-core 1.0.0
questions: problem, oven heat, oven door opened early, eggs, flour mixing, baking time, pan
advise problem = dry; baking time = long -> take it out earlier (rule take it out
earlier#1)
advise problem = sticky top -> (no answer yet)
all checks passed
```

Layanan Java dan pemeriksaan Python bekerja bersama tanpa penyesuaian apa pun, karena keduanya berbicara dengan HTTP dan JSON yang sama; layanan Python dengan pemeriksaan Java mencetak baris-baris yang sama. Layanan juga dapat ditanya secara manual. Di peramban, `http://localhost:4151/health` menampilkan:

```
{
  "status": "ok",
  "name": "Cake doctor",
  "nodes": 12,
  "engine": "vcirs-core 1.0.0"
}
```

dan sebuah konsultasi dapat dikirim dengan `curl`, alat HTTP baris perintah yang tersedia di Windows 10 dan 11, macOS, dan kebanyakan sistem Linux: `curl -X POST -H "X-Test: yes" -d`

`"{\"facts\": {\"problem\": \"dry\", \"baking time\": \"long\"}}"` `http://localhost:4151/advise` menjawab

```
{
  "advice": [
    "take it out earlier"
  ],
  "rule": "take it out earlier#1",
  "how": [
    "take it out earlier#1 fired because:",
    "  problem = dry  (given, CF 100)",
    "  baking time = long  (given, CF 100)",
    "  so advice = take it out earlier  [rule CF 100, conclusion CF 100]"
  ]
}
```

dengan saran, rule, dan penjelasan lengkapnya. Program apa pun dalam bahasa apa pun yang dapat mengirim HTTP — aplikasi ponsel, makro lembar kerja, antarmuka obrolan Bab 13 — dapat memakai dokter kue persis dengan cara ini.

15.6 Langkah 4: Cadangan

Kode 15.8 adalah alat pencadangannya.

Kode 15.8 `backup.py`: menyimpan, memeriksa, dan memulihkan cadangan basis pengetahuan, dengan sidik jari SHA-256.

```
"""Cadangkan basis pengetahuan, periksa cadangan, dan pulihkan salah satunya.

    python backup.py save    KB BACKUPDIR [--keep 30] [--stamp YYYYMMDD-HHMM]
    python backup.py check   BACKUPDIR
    python backup.py restore BACKUPDIR NAME KB

save menyalin berkas hanya bila termuat sebagai basis pengetahuan, menamai
salinan menurut waktu, menulis sidik jari SHA-256-nya ke BACKUPDIR/index.txt,
dan hanya menyimpan --keep salinan terbaru. check membaca ulang setiap salinan
dan membandingkannya dengan sidik jarinya. restore mengembalikan salinan
(setelah diperiksa), dan menyimpan berkas yang digantinya: KB.before-restore.
"""

import datetime
import hashlib
import os
import pathlib
import shutil
import sys

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import KnowledgeBase, VcirsError                    # noqa: E402


def fingerprint(path):
    with open(path, "rb") as f:
        return hashlib.sha256(f.read()).hexdigest()


def loads(path):
    """Basis pengetahuan dalam berkas, atau None bila tak terbaca."""
    try:
        return KnowledgeBase.load(path)
    except (OSError, ValueError, KeyError, TypeError, VcirsError):
        return None


def read_index(folder):
    path = os.path.join(folder, "index.txt")
    if not os.path.exists(path):
        return []
    with open(path, encoding="utf-8") as f:
        return [line.split() for line in f if line.strip()]
```

Kode 15.8 (Lanjutan) *backup.py: menyimpan, memeriksa, dan memulihkan cadangan basis pengetahuan, dengan sidik jari SHA-256.*

```
def write_index(folder, rows):
    with open(os.path.join(folder, "index.txt"), "w", encoding="utf-8",
              newline="\n") as f:
        f.writelines(" ".join(r) + "\n" for r in rows)


def save(kb_path, folder, keep, stamp):
    kb = loads(kb_path)
    if kb is None:
        print("NOT saved: " + kb_path + " is not a readable knowledge base")
        return False
    os.makedirs(folder, exist_ok=True)
    name = "kb-" + stamp + ".json"
    shutil.copyfile(kb_path, os.path.join(folder, name))
    rows = [r for r in read_index(folder) if r[0] != name]
    rows.append([name, fingerprint(kb_path), str(len(kb.nodes))])
    for old in rows[:-keep] if len(rows) > keep else []:
        os.remove(os.path.join(folder, old[0]))
        print("removed old backup " + old[0])
    write_index(folder, rows[-keep:])
    print("saved " + name + " (" + str(len(kb.nodes)) + " nodes, sha256 "
          + rows[-1][1][:12] + "...)")
    return True


def check(folder):
    ok = True
    for name, digest, nodes in read_index(folder):
        path = os.path.join(folder, name)
        if not os.path.exists(path):
            state = "MISSING"
        elif fingerprint(path) != digest:
            state = "DAMAGED (fingerprint differs)"
        elif loads(path) is None:
            state = "DAMAGED (cannot be read)"
        else:
            state = "ok, " + nodes + " nodes"
        ok = ok and state.startswith("ok")
        print(name + ": " + state)
    return ok
```

Kode 15.8 (Lanjutan) *backup.py: menyimpan, memeriksa, dan memulihkan cadangan basis pengetahuan, dengan sidik jari SHA-256.*

```
def restore(folder, name, kb_path):
    rows = [r for r in read_index(folder) if r[0] == name]
    path = os.path.join(folder, name)
    if not rows or not os.path.exists(path) \
            or fingerprint(path) != rows[0][1] or loads(path) is None:
        print("NOT restored: " + name + " is missing or damaged")
        return False
    if os.path.exists(kb_path):
        shutil.copyfile(kb_path, kb_path + ".before-restore")
    shutil.copyfile(path, kb_path)
    print("restored " + name + " to " + kb_path)
    return True


def main(args):
    def option(name, default):
        return args[args.index(name) + 1] if name in args else default

    words = [a for i, a in enumerate(args) if not a.startswith("--")
             and (i == 0 or not args[i - 1].startswith("--"))]
    stamp = option("--stamp", datetime.datetime.now().strftime("%Y%m%d-%H%M"))
    if len(words) >= 3 and words[0] == "save":
        good = save(words[1], words[2], int(option("--keep", "30")), stamp)
    elif len(words) >= 2 and words[0] == "check":
        good = check(words[1])
    elif len(words) >= 4 and words[0] == "restore":
        good = restore(words[1], words[2], words[3])
    else:
        print("usage: backup save KB DIR [--keep N] [--stamp S] | check DIR"
              " | restore DIR NAME KB")
        good = False
    sys.exit(0 if good else 1)


if __name__ == "__main__":
    main(sys.argv[1:])
```

`save` menolak mencadangkan berkas yang tidak termuat sebagai basis pengetahuan; menyalinnya dengan nama yang dibentuk dari tanggal dan jam, `kb-20260925-2130.json`, sehingga nama-namanya terurut menurut waktu; menambahkan nama, sidik jari, dan jumlah node ke `index.txt`; dan, bila salinannya lebih banyak dari `--keep`, menghapus yang tertua. `check` menelusuri indeks: berkas yang hilang, berkas yang sidik jarinya berbeda, dan berkas yang tidak termuat semuanya dilaporkan, dan kode keluarnya 1 bila ada yang salah. `restore` memeriksa salinan yang dipilih dengan cara yang sama sebelum memakainya, dan lebih dulu menyimpan berkas yang sekarang sebagai `…before-restore`. **Kode 15.9** adalah versi Java-nya; `MessageDigest` menghitung SHA-256 dan `HexFormat` menuliskannya sebagai digit heksadesimal.

Kode 15.9 `Backup.java`: alat pencadangan yang sama dalam Java.

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardCopyOption;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.HexFormat;
import java.util.List;

import vcirs.KnowledgeBase;

/**
 * Cadangkan basis pengetahuan, periksa cadangan, dan pulihkan salah satunya.
 * java Backup save    KB BACKUPDIR [--keep 30] [--stamp YYYYMMDD-HHMM]
 * java Backup check   BACKUPDIR
 * java Backup restore BACKUPDIR NAME KB
 * save menyalin berkas hanya bila termuat sebagai basis pengetahuan, menamai
 * salinan menurut waktu, menulis sidik jari SHA-256-nya ke
 * BACKUPDIR/index.txt, dan hanya menyimpan --keep salinan terbaru. check
 * membaca ulang setiap salinan dan membandingkannya dengan sidik jarinya.
 * restore mengembalikan salinan (setelah diperiksa), dan menyimpan berkas
 * yang digantinya sebagai KB.before-restore.
 */
public class Backup {
    static String fingerprint(Path path) throws IOException {
        try {
            MessageDigest sha = MessageDigest.getInstance("SHA-256");
            return HexFormat.of().formatHex(sha.digest(Files.readAllBytes(path)));
        } catch (NoSuchAlgorithmException e) {
            throw new IllegalStateException(e);
        }
    }

    /** Basis pengetahuan dalam berkas, atau null bila tak terbaca. */
    static KnowledgeBase loads(Path path) {
        try {
            return KnowledgeBase.load(path.toString());
        } catch (RuntimeException e) {
            return null;
        }
    }
```

Kode 15.9 (Lanjutan) *Backup.java: alat pencadangan yang sama dalam Java.*

```
    static List<String[]> readIndex(Path folder) throws IOException {
        Path path = folder.resolve("index.txt");
        List<String[]> rows = new ArrayList<>();
        if (Files.exists(path)) {
            for (String line : Files.readAllLines(path, StandardCharsets.UTF_8)) {
                if (!line.isBlank()) {
                    rows.add(line.strip().split("\\s+"));
                }
            }
        }
        return rows;
    }

    static void writeIndex(Path folder, List<String[]> rows) throws IOException {
        StringBuilder b = new StringBuilder();
        rows.forEach(r -> b.append(String.join(" ", r)).append("\n"));
        Files.writeString(folder.resolve("index.txt"), b, StandardCharsets.UTF_8);
    }

    static boolean save(Path kbPath, Path folder, int keep, String stamp)
            throws IOException {
        KnowledgeBase kb = loads(kbPath);
        if (kb == null) {
            System.out.println("NOT saved: " + kbPath
                    + " is not a readable knowledge base");
            return false;
        }
        Files.createDirectories(folder);
        String name = "kb-" + stamp + ".json";
        Files.copy(kbPath, folder.resolve(name),
                StandardCopyOption.REPLACE_EXISTING);
        List<String[]> rows = new ArrayList<>();
        for (String[] r : readIndex(folder)) {
            if (!r[0].equals(name)) {
                rows.add(r);
            }
        }
        rows.add(new String[] {name, fingerprint(kbPath),
            String.valueOf(kb.nodes.size())});
        while (rows.size() > keep) {
            Files.delete(folder.resolve(rows.get(0)[0]));
            System.out.println("removed old backup " + rows.get(0)[0]);
            rows.remove(0);
        }
        writeIndex(folder, rows);
        String[] last = rows.get(rows.size() - 1);
```

Kode 15.9 (Lanjutan) *Backup.java: alat pencadangan yang sama dalam Java.*

```
        System.out.println("saved " + name + " (" + last[2] + " nodes, sha256 "
                + last[1].substring(0, 12) + "...)");
        return true;
    }

    static boolean check(Path folder) throws IOException {
        boolean ok = true;
        for (String[] r : readIndex(folder)) {
            Path path = folder.resolve(r[0]);
            String state;
            if (!Files.exists(path)) {
                state = "MISSING";
            } else if (!fingerprint(path).equals(r[1])) {
                state = "DAMAGED (fingerprint differs)";
            } else if (loads(path) == null) {
                state = "DAMAGED (cannot be read)";
            } else {
                state = "ok, " + r[2] + " nodes";
            }
            ok = ok && state.startsWith("ok");
            System.out.println(r[0] + ": " + state);
        }
        return ok;
    }

    static boolean restore(Path folder, String name, Path kbPath)
            throws IOException {
        String[] row = null;
        for (String[] r : readIndex(folder)) {
            if (r[0].equals(name)) {
                row = r;
            }
        }
        Path path = folder.resolve(name);
        if (row == null || !Files.exists(path)
                || !fingerprint(path).equals(row[1]) || loads(path) == null) {
            System.out.println("NOT restored: " + name + " is missing or damaged");
            return false;
        }
        if (Files.exists(kbPath)) {
            Files.copy(kbPath, Path.of(kbPath + ".before-restore"),
                    StandardCopyOption.REPLACE_EXISTING);
        }
        Files.copy(path, kbPath, StandardCopyOption.REPLACE_EXISTING);
        System.out.println("restored " + name + " to " + kbPath);
        return true;
```

Kode 15.9 (Lanjutan) *Backup.java: alat pencadangan yang sama dalam Java.*

```
    }

    public static void main(String[] args) throws IOException {
        List<String> words = new ArrayList<>();
        int keep = 30;
        String stamp = LocalDateTime.now().format(
                DateTimeFormatter.ofPattern("yyyyMMdd-HHmm"));
        for (int i = 0; i < args.length; i++) {
            if (args[i].equals("--keep") && i + 1 < args.length) {
                keep = Integer.parseInt(args[++i]);
            } else if (args[i].equals("--stamp") && i + 1 < args.length) {
                stamp = args[++i];
            } else {
                words.add(args[i]);
            }
        }
        boolean good;
        String what = words.isEmpty() ? "" : words.get(0);
        if (what.equals("save") && words.size() >= 3) {
            good = save(Path.of(words.get(1)), Path.of(words.get(2)), keep, stamp);
        } else if (what.equals("check") && words.size() >= 2) {
            good = check(Path.of(words.get(1)));
        } else if (what.equals("restore") && words.size() >= 4) {
            good = restore(Path.of(words.get(1)), words.get(2),
                    Path.of(words.get(3)));
        } else {
            System.out.println("usage: backup save KB DIR [--keep N] [--stamp S]"
                    + " | check DIR | restore DIR NAME KB");
            good = false;
        }
        System.exit(good ? 0 : 1);
    }
}
```

Berikut satu minggu pencadangan yang dipadatkan menjadi beberapa menit. `--stamp` memberi setiap cadangan nama malam yang diwakilinya, dan `--keep 2` hanya menyimpan dua, agar kita dapat melihat yang tertua dihapus:

```
$ java -jar cake-doctor.jar backup save data/cake-kb.json backups --keep 2 --stamp
20260925-2130
saved kb-20260925-2130.json (12 nodes, sha256 b54cda04bbc9...)
$ java -jar cake-doctor.jar backup save data/cake-kb.json backups --keep 2 --stamp
20260926-2130
saved kb-20260926-2130.json (12 nodes, sha256 b54cda04bbc9...)
$ java -jar cake-doctor.jar backup save data/cake-kb.json backups --keep 2 --stamp
20260927-2130
removed old backup kb-20260925-2130.json
saved kb-20260927-2130.json (12 nodes, sha256 b54cda04bbc9...)
$ java -jar cake-doctor.jar backup check backups
kb-20260926-2130.json: ok, 12 nodes
kb-20260927-2130.json: ok, 12 nodes
```

Tiga kali menyimpan, yang tertua dihapus pada penyimpanan ketiga, dan pemeriksaan yang mendapati kedua salinan yang tersisa utuh. Sidik jarinya sama karena basis pengetahuan tidak berubah di antaranya; dalam minggu yang sungguhan sidik jarinya akan berbeda. Sekarang kita sengaja merusak satu cadangan dengan menambahkan satu baris ke dalamnya, seperti yang mungkin dilakukan *flashdisk* yang mulai rusak, lalu memeriksa lagi:

```
$ java -jar cake-doctor.jar backup check backups
kb-20260926-2130.json: DAMAGED (fingerprint differs)
kb-20260927-2130.json: ok, 12 nodes
$ java -jar cake-doctor.jar backup restore backups kb-20260926-2130.json data/cake-kb.json
NOT restored: kb-20260926-2130.json is missing or damaged
$ java -jar cake-doctor.jar backup restore backups kb-20260927-2130.json data/cake-kb.json
restored kb-20260927-2130.json to data/cake-kb.json
```

Pemeriksaan langsung menemukan kerusakannya. Pemulihan dari salinan yang rusak ditolak; pemulihan dari salinan yang baik berhasil, dan `cake-kb.json` yang sebelumnya disimpan sebagai `cake-kb.json.before-restore`. Pada hari Senin di Subbab 15.1, inilah perintah yang akan diketik keponakan Bu Wati di laptop baru, dan toko paling banyak kehilangan satu hari pengajaran.

15.7 Langkah 5: Memantau Pemakaian Rule

Dari Juli sampai September 2026, para pembantu Bu Wati menulis setiap konsultasi pada kartu di dekat oven — tanggal, fakta-fakta, apa kata dokter — sementara layanannya sedang disiapkan. Berkas `data/visits.txt` bab ini memuat 120 konsultasi itu, satu per baris, seperti

```
2026-07-01T04:16:00Z | problem = sank in the middle; oven door opened early = no; oven
heat = low
```

dan `python cake-doctor.pyz check --replay visits.txt`, yang dikirim ke layanan yang dijalankan dengan `--replay`, memasukkannya ke catatan dengan tanggal aslinya:

```
replayed 120 consultations to http://localhost:4151
```

Setiap konsultasi menjadi satu baris di berkas catatan bulannya; baris pertama `logs/usage-2026-07.jsonl` adalah

```
{"time": "2026-07-01T04:16:00Z", "rule": "heat the oven fully first#1", "advice": ["heat
the oven fully first"], "facts": {"problem": "sank in the middle", "oven door opened
early": "no", "oven heat": "low"}}
```

— sebuah waktu, node, saran, dan tiga fakta tentang kue. Tidak ada di dalamnya yang mengenali seseorang. **Kode 15.10** adalah alat yang membaca baris-baris ini.

Kode 15.10 `usage_report.py`: laporan pemakaian bulanan, dan penghapusan baris catatan yang lama.

```
"""Baca catatan pemakaian layanan saran dan katakan apa yang ditunjukkannya.

    python usage_report.py [LOGDIR] [--kb FILE] [--keep-days N --today DATE]

Ia menghitung konsultasi per bulan, berapa yang terjawab, node mana yang
memberi saran (dan mana yang tidak pernah), dan kasus tak terjawab mana yang
paling sering kembali: itulah kasus yang perlu diajarkan berikutnya. Dengan
--keep-days ia juga menghapus baris yang lebih tua dari N hari sebelum DATE
(bawaan: hari ini), agar catatan tidak disimpan lebih lama dari perlunya.
"""

import datetime
import json
import os
import pathlib
import sys

sys.path.insert(0, str(pathlib.Path(__file__).resolve().parents[2]
                       / "vcirs-core" / "python"))
from vcirs import KnowledgeBase, num                          # noqa: E402

MONTHS = ["January", "February", "March", "April", "May", "June", "July",
          "August", "September", "October", "November", "December"]


def read_logs(log_dir):
    """Setiap catatan dari setiap berkas usage-*.jsonl, berkas tertua dulu."""
    records = []
    for name in sorted(os.listdir(log_dir)):
        if name.startswith("usage-") and name.endswith(".jsonl"):
            with open(os.path.join(log_dir, name), encoding="utf-8") as f:
                records += [json.loads(line) for line in f if line.strip()]
    return records


def prune(log_dir, keep_days, today):
    """Hapus catatan lebih tua dari keep_days sebelum today; hasil: jumlahnya."""
    limit = (today - datetime.timedelta(days=keep_days)).isoformat()
    removed = 0
    for name in sorted(os.listdir(log_dir)):
        if not (name.startswith("usage-") and name.endswith(".jsonl")):
            continue
        path = os.path.join(log_dir, name)
        with open(path, encoding="utf-8") as f:
            lines = [line for line in f if line.strip()]
        keep = [line for line in lines if json.loads(line)["time"] >= limit]
```

Kode 15.10 (Lanjutan) *usage_report.py: laporan pemakaian bulanan, dan penghapusan baris catatan yang lama.*

```
        removed += len(lines) - len(keep)
        if not keep:
            os.remove(path)
        elif len(keep) < len(lines):
            with open(path, "w", encoding="utf-8", newline="\n") as f:
                f.writelines(keep)
    return removed


def percent(part, whole):
    return num.fixed(100.0 * part / whole, 1) + "%" if whole else "-"


def report(records, kb):
    if not records:
        return ["No consultations in the logs."]
    out = ["Consultations: " + str(len(records)) + ", from "
           + records[0]["time"][:10] + " to " + records[-1]["time"][:10]]
    months = {}
    for r in records:
        months[r["time"][:7]] = months.get(r["time"][:7], 0) + 1
    out.append("  " + ", ".join(MONTHS[int(m[5:]) - 1] + " " + m[:4] + ": "
                                + str(n) for m, n in months.items()))
    answered = [r for r in records if r["rule"]]
    out.append("Answered: " + str(len(answered)) + " ("
               + percent(len(answered), len(records)) + "), no answer: "
               + str(len(records) - len(answered)) + " ("
               + percent(len(records) - len(answered), len(records)) + ")")
    counts = {}
    for r in answered:
        counts[r["rule"]] = counts.get(r["rule"], 0) + 1
    out.append("Advice given, most used first:")
    for rule, n in sorted(counts.items(), key=lambda x: (-x[1], x[0])):
        out.append("  %4d  %-6s %s" % (n, percent(n, len(records)), rule))
    if kb is not None:
        idle = [n.id for n in kb.nodes.values()
                if n.id not in counts and not n.stopped]
        out.append("Nodes that never gave the advice: "
                   + (", ".join(idle) if idle else "none"))
    missing = {}
    for r in records:
        if not r["rule"]:
            key = "; ".join(k + " = " + v for k, v in r["facts"].items())
            missing[key] = missing.get(key, 0) + 1
    out.append("Cases to teach next (no answer), most frequent first:")
    for key, n in sorted(missing.items(), key=lambda x: (-x[1], x[0])):
```

Kode 15.10 (Lanjutan) *usage_report.py: laporan pemakaian bulanan, dan penghapusan baris catatan yang lama.*

```
        out.append("  %4d  %s" % (n, key))
    return out


def main(args):
    def option(name, default):
        return args[args.index(name) + 1] if name in args else default

    values = [args[i] for i in range(len(args)) if not args[i].startswith("--")
              and (i == 0 or not args[i - 1].startswith("--"))]
    log_dir = values[0] if values else "logs"
    kb_path = option("--kb", None)
    if "--keep-days" in args:
        today = datetime.date.fromisoformat(
            option("--today", datetime.date.today().isoformat()))
        removed = prune(log_dir, int(option("--keep-days", "0")), today)
        print("Removed " + str(removed) + " log lines older than "
              + option("--keep-days", "0") + " days before "
              + today.isoformat() + ".")
    kb = KnowledgeBase.load(kb_path) if kb_path else None
    print("Usage report for " + log_dir
          + (" and " + kb.name if kb else ""))
    for line in report(read_logs(log_dir), kb):
        print(line)


if __name__ == "__main__":
    main(sys.argv[1:])
```

`read_logs` membaca setiap berkas `usage-*.jsonl` menurut urutan nama, yang sama dengan urutan waktu. `report` menghitung konsultasi per bulan; yang terjawab dan tidak terjawab, sebagai bilangan dan persentase; seberapa sering setiap node memberi saran, dari yang paling sering; node basis pengetahuan mana yang tidak pernah memberi saran (node yang dihentikan tidak diikutkan, karena memang tidak memberi saran); dan kasus-kasus tak terjawab, dikelompokkan menurut faktanya dan diurutkan menurut seberapa sering terjadi. `prune` menghapus baris-baris yang lebih tua dari `--keep-days` hari sebelum `--today` (bawaannya hari ini yang sebenarnya), menulis ulang berkas sebuah bulan atau menghapusnya bila tidak ada isinya yang cukup baru. Persentase diformat dengan `num.fixed` milik mesin, agar Java dan Python membulatkannya dengan cara yang sama. **Kode 15.11** adalah versi Java-nya.

Kode 15.11 `UsageReport.java`: laporan yang sama dalam Java.

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Stream;

import vcirs.Json;
import vcirs.KnowledgeBase;
import vcirs.Node;
import vcirs.Num;

/**
 * Baca catatan pemakaian layanan saran dan katakan apa yang ditunjukkannya.
 * java UsageReport [LOGDIR] [--kb FILE] [--keep-days N --today DATE]
 * Ia menghitung konsultasi per bulan, berapa yang terjawab, node mana yang
 * memberi saran (dan mana yang tidak pernah), dan kasus tak terjawab mana
 * yang paling sering kembali: itulah kasus yang perlu diajarkan berikutnya.
 * Dengan --keep-days ia juga menghapus baris yang lebih tua dari N hari
 * sebelum DATE (bawaan: hari ini), agar catatan tidak disimpan terlalu lama.
 */
public class UsageReport {
    static final String[] MONTHS = {"January", "February", "March", "April",
        "May", "June", "July", "August", "September", "October", "November",
        "December"};

    static List<Path> logFiles(Path dir) throws IOException {
        try (Stream<Path> s = Files.list(dir)) {
            return s.filter(p -> p.getFileName().toString().startsWith("usage-")
                    && p.getFileName().toString().endsWith(".jsonl"))
                    .sorted().toList();
        }
    }
```

Kode 15.11 (Lanjutan) *UsageReport.java: laporan yang sama dalam Java.*

```
    /** Setiap catatan dari setiap berkas usage-*.jsonl, berkas tertua dulu. */
    @SuppressWarnings("unchecked")
    static List<Map<String, Object>> readLogs(Path dir) throws IOException {
        List<Map<String, Object>> records = new ArrayList<>();
        for (Path p : logFiles(dir)) {
            for (String line : Files.readAllLines(p, StandardCharsets.UTF_8)) {
                if (!line.isBlank()) {
                    records.add((Map<String, Object>) Json.parse(line));
                }
            }
        }
        return records;
    }

    /** Hapus catatan lebih tua dari keepDays sebelum today; hasil: jumlahnya. */
    @SuppressWarnings("unchecked")
    static int prune(Path dir, int keepDays, LocalDate today) throws IOException {
        String limit = today.minusDays(keepDays).toString();
        int removed = 0;
        for (Path p : logFiles(dir)) {
            List<String> lines = new ArrayList<>();
            for (String line : Files.readAllLines(p, StandardCharsets.UTF_8)) {
                if (!line.isBlank()) {
                    lines.add(line);
                }
            }
            List<String> keep = new ArrayList<>();
            for (String line : lines) {
                String time = (String) ((Map<String, Object>) Json.parse(line))
                        .get("time");
                if (time.compareTo(limit) >= 0) {
                    keep.add(line);
                }
            }
            removed += lines.size() - keep.size();
            if (keep.isEmpty()) {
                Files.delete(p);
            } else if (keep.size() < lines.size()) {
                Files.writeString(p, String.join("\n", keep) + "\n",
                        StandardCharsets.UTF_8);
            }
        }
        return removed;
    }
```

Kode 15.11 (Lanjutan) *UsageReport.java: laporan yang sama dalam Java.*

```
    static String percent(int part, int whole) {
        return whole == 0 ? "-" : Num.fixed(100.0 * part / whole, 1) + "%";
    }

    static void count(Map<String, Integer> counts, String key) {
        counts.merge(key, 1, Integer::sum);
    }

    /** Kunci-kunci, hitungan terbesar dulu, yang sama menurut urutan nama. */
    static List<String> byCount(Map<String, Integer> counts) {
        List<String> keys = new ArrayList<>(counts.keySet());
        keys.sort((a, b) -> counts.get(a).equals(counts.get(b))
                ? a.compareTo(b) : counts.get(b) - counts.get(a));
        return keys;
    }

    @SuppressWarnings("unchecked")
    static List<String> report(List<Map<String, Object>> records,
                               KnowledgeBase kb) {
        List<String> out = new ArrayList<>();
        if (records.isEmpty()) {
            out.add("No consultations in the logs.");
            return out;
        }
        int total = records.size();
        String first = (String) records.get(0).get("time");
        String last = (String) records.get(total - 1).get("time");
        out.add("Consultations: " + total + ", from " + first.substring(0, 10)
                + " to " + last.substring(0, 10));
        Map<String, Integer> months = new LinkedHashMap<>();
        Map<String, Integer> counts = new LinkedHashMap<>();
        Map<String, Integer> missing = new LinkedHashMap<>();
        int answered = 0;
        for (Map<String, Object> r : records) {
            count(months, ((String) r.get("time")).substring(0, 7));
            String rule = (String) r.get("rule");
            if (!rule.isEmpty()) {
                answered++;
                count(counts, rule);
            } else {
                List<String> parts = new ArrayList<>();
                ((Map<String, Object>) r.get("facts")).forEach((k, v) ->
                        parts.add(k + " = " + v));
                count(missing, String.join("; ", parts));
            }
        }
```

Kode 15.11 (Lanjutan) *UsageReport.java: laporan yang sama dalam Java.*

```
        List<String> perMonth = new ArrayList<>();
        months.forEach((m, n) -> perMonth.add(MONTHS[Integer.parseInt(
                m.substring(5)) - 1] + " " + m.substring(0, 4) + ": " + n));
        out.add("  " + String.join(", ", perMonth));
        out.add("Answered: " + answered + " (" + percent(answered, total)
                + "), no answer: " + (total - answered) + " ("
                + percent(total - answered, total) + ")");
        out.add("Advice given, most used first:");
        for (String rule : byCount(counts)) {
            out.add(String.format("  %4d  %-6s %s", counts.get(rule),
                    percent(counts.get(rule), total), rule));
        }
        if (kb != null) {
            List<String> idle = new ArrayList<>();
            for (Node n : kb.nodes.values()) {
                if (!counts.containsKey(n.id) && !n.stopped) {
                    idle.add(n.id);
                }
            }
            out.add("Nodes that never gave the advice: "
                    + (idle.isEmpty() ? "none" : String.join(", ", idle)));
        }
        out.add("Cases to teach next (no answer), most frequent first:");
        for (String key : byCount(missing)) {
            out.add(String.format("  %4d  %s", missing.get(key), key));
        }
        return out;
    }

    public static void main(String[] args) throws IOException {
        Map<String, String> opt = new LinkedHashMap<>();
        String logDir = "logs";
        for (int i = 0; i < args.length; i++) {
            if (args[i].startsWith("--") && i + 1 < args.length) {
                opt.put(args[i], args[++i]);
            } else {
                logDir = args[i];
            }
        }
        if (opt.containsKey("--keep-days")) {
            LocalDate today = LocalDate.parse(opt.getOrDefault("--today",
                    LocalDate.now().toString()));
            int removed = prune(Path.of(logDir),
                    Integer.parseInt(opt.get("--keep-days")), today);
            System.out.println("Removed " + removed + " log lines older than "
                    + opt.get("--keep-days") + " days before " + today + ".");
```

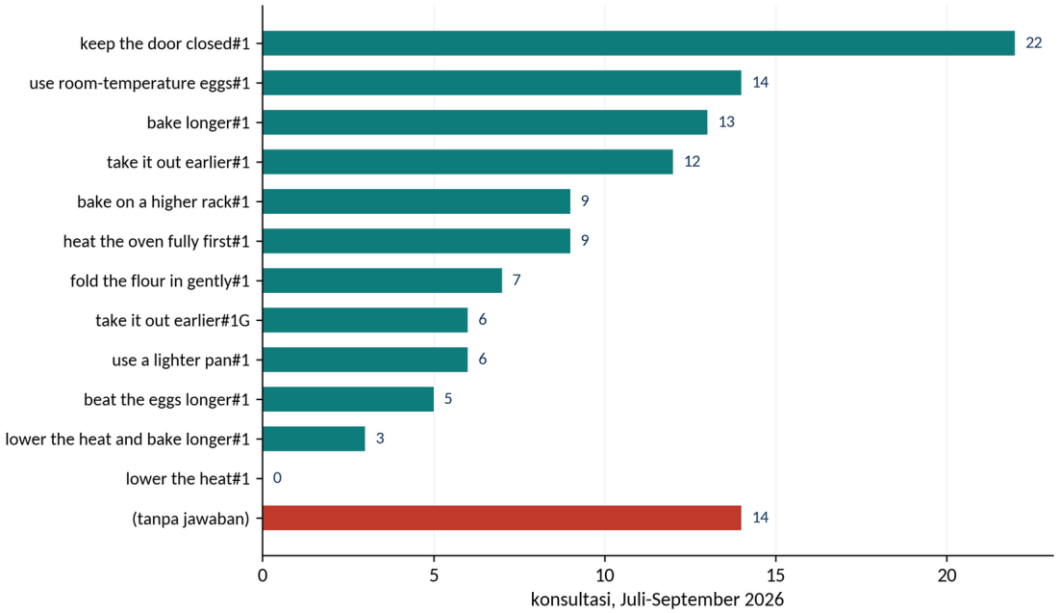
Kode 15.11 (Lanjutan) *UsageReport.java: laporan yang sama dalam Java.*

```
        }
        KnowledgeBase kb = opt.containsKey("--kb")
                ? KnowledgeBase.load(opt.get("--kb")) : null;
        System.out.println("Usage report for " + logDir
                + (kb != null ? " and " + kb.name : ""));
        for (String line : report(readLogs(Path.of(logDir)), kb)) {
            System.out.println(line);
        }
    }
}
```

`java -jar cake-doctor.jar report logs --kb data/cake-kb.json` mencetak laporan untuk ketiga bulan itu:

```
Usage report for logs and Cake doctor
Consultations: 120, from 2026-07-01 to 2026-09-30
  July 2026: 41, August 2026: 40, September 2026: 39
Answered: 106 (88.3%), no answer: 14 (11.7%)
Advice given, most used first:
    22  18.3%  keep the door closed#1
    14  11.7%  use room-temperature eggs#1
    13  10.8%  bake longer#1
    12  10.0%  take it out earlier#1
     9  7.5%   bake on a higher rack#1
     9  7.5%   heat the oven fully first#1
     7  5.8%   fold the flour in gently#1
     6  5.0%   take it out earlier#1G
     6  5.0%   use a lighter pan#1
     5  4.2%   beat the eggs longer#1
     3  2.5%   lower the heat and bake longer#1
Nodes that never gave the advice: lower the heat#1
Cases to teach next (no answer), most frequent first:
     8  problem = sticky top
     4  problem = uneven rise
     2  problem = cracked top; oven heat = normal
```

Gambar 15.2 menggambarkan angka-angka yang sama.



Gambar 15.2 Seberapa sering setiap node memberi saran dalam 120 konsultasi, Juli sampai September 2026 (dari catatan yang diputar ulang), beserta node yang tidak pernah memberi saran dan konsultasi yang tidak mendapat jawaban.

Bacalah laporan itu seperti Bu Wati membacanya, dengan secangkir teh pada tanggal satu Oktober.

Dokter dipakai, dan kebanyakan menjawab. 120 konsultasi dalam tiga bulan, sekitar sepuluh seminggu, dan 88,3% di antaranya mendapat saran. Bagian konsultasi terjawab yang bertambah dari bulan ke bulan berarti basis pengetahuan mulai menyusul keadaan toko; bagian yang tiba-tiba turun berarti ada sesuatu yang baru — tepung baru, oven baru — dan itulah saatnya memperhatikan dengan cermat.

Membuka pintu oven adalah kesalahan yang paling sering. *keep the door closed* memberi 22 dari 120 jawaban, 18,3%. Itu bukan pengetahuan yang perlu diubah Bu Wati; itu pengetahuan yang dapat ia *tindak lanjuti*: sebuah tulisan kecil di oven, *jangan dibuka dalam 20 menit pertama*, akan mencegah lebih banyak kue gagal daripada rule baru mana pun.

Node hasil pembangkitan layak dipertahankan. *take it out earlier#1G*, node umum yang diterima Bu Wati dari sebuah usulan di Bab 14, menjawab 6 konsultasi: kue kering yang tidak diketahui lama memanggangnya. Tanpa node itu, keenamnya tidak akan mendapat jawaban.

Satu node tidak pernah dipakai. *lower the heat#1* — bagian atas yang retak dengan oven yang panas — tidak pernah memberi saran. Itu saja belum membuktikan apa-apa; tetapi lihatlah daftar berikutnya.

Kasus yang perlu diajarkan berikutnya. Bagian atas yang lengket muncul 8 kali tanpa jawaban, kue yang mengembang tidak rata 4 kali, dan bagian atas yang retak dengan oven yang *normal* dua kali. Keempat belas konsultasi itu adalah hal paling berguna di seluruh catatan. Bagian atas yang lengket mudah diputuskan: lain kali terjadi, Bu Wati mencari penyebabnya dan mengajarkan kasusnya di pembangun. Bagian atas yang retak lebih menarik: kejadiannya dengan oven yang normal, sehingga node yang ada untuk oven panas tidak pernah menyala, dan kasus milik node itu sendiri tidak pernah terjadi lagi selama tiga bulan. Bersama Soal 14.3, inilah bukti yang dibutuhkan Bu Wati untuk mengajarkan penyebab yang sudah lama ia duga — terlalu banyak baking powder.

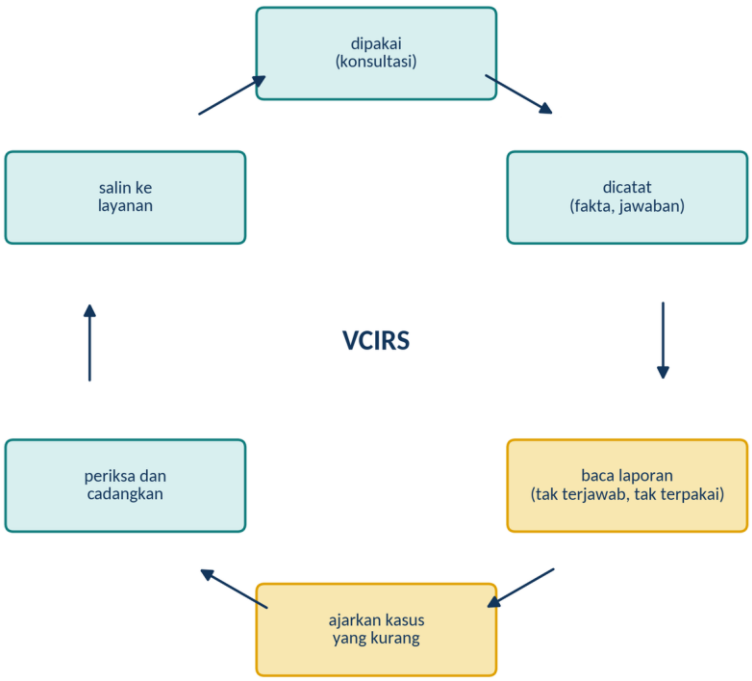
Langkah terakhir menyimpan catatan hanya selama dibutuhkan. Dengan `--keep-days 60`, seolah-olah tanggalnya 30 September, alat laporan lebih dulu menghapus baris-baris yang lebih tua:

```
Removed 41 log lines older than 60 days before 2026-09-30.
Usage report for logs
Consultations: 79, from 2026-08-01 to 2026-09-30
  August 2026: 40, September 2026: 39
Answered: 71 (89.9%), no answer: 8 (10.1%)
Advice given, most used first:
    14  17.7%  keep the door closed#1
    10  12.7%  take it out earlier#1
    10  12.7%  use room-temperature eggs#1
     9  11.4%  bake longer#1
     7  8.9%   fold the flour in gently#1
     5  6.3%   heat the oven fully first#1
     4  5.1%   bake on a higher rack#1
     4  5.1%   beat the eggs longer#1
     3  3.8%   lower the heat and bake longer#1
     3  3.8%   take it out earlier#1G
     2  2.5%   use a lighter pan#1
Cases to teach next (no answer), most frequent first:
     4  problem = sticky top
     3  problem = uneven rise
     1  problem = cracked top; oven heat = normal
```

41 baris bulan Juli sudah hilang, begitu pula berkas `usage-2026-07.jsonl`. Di toko, laporan terjadwal menyimpan 180 hari; pilih bilangannya dengan bertanya seberapa jauh ke belakang Anda benar-benar melihat.

15.8 Merawat dan Menumbuhkan Pengetahuan

Sistem pakar yang sudah diterapkan tidak pernah selesai. **Gambar 15.3** memperlihatkan siklus yang menjaganya tetap hidup.



Gambar 15.3 Siklus perawatan: layanan dipakai, setiap konsultasi dicatat, laporan bulanan menunjukkan apa yang tak terjawab dan tak terpakai, sang pakar mengajarkan kasus yang kurang di pembangun, basis pengetahuan yang baru diperiksa dan dicadangkan, dan salinan yang sudah diperiksa disalin ke layanan.

Kebiasaan bulanan. Untuk toko kecil, satu jam sebulan sudah cukup. Bacalah laporannya. Ajarkan dua atau tiga kasus tak terjawab yang paling sering ketika kasus itu terjadi lagi (atau langsung, bila Anda mengingat sebuah kasus nyata dengan baik). Tengoklah node-node yang tidak pernah memberi saran dan putuskan apakah akan dipertahankan, dan tengoklah node-node yang paling sering dipakai dan tanyakan apakah sarannya dapat ditindaklanjuti, seperti tulisan di oven. Lalu salin basis pengetahuan yang baru ke layanan, jalankan pemeriksaannya, dan buat satu cadangan secara manual di samping yang otomatis.

Satu tempat untuk mengajar. Mengajarlah di satu salinan basis pengetahuan — pembangun di laptop pemilik — lalu salin ke layanan. Jangan pernah mengajar di dua salinan sekaligus: dua basis pengetahuan yang tumbuh terpisah tidak dapat digabungkan secara otomatis, karena setiap node baru dipilih terhadap kasus-kasus *cornerstone* yang ada *di salinan itu*. Bila dua orang harus mengajar, biarkan mereka mengajar di pembangun yang sama, bergantian; riwayatnya mencatat siapa mengajarkan apa.

Versi. Setiap ekspor adalah sebuah versi. Namai salinan yang Anda kirim ke layanan menurut tanggal — alat pencadangan sudah melakukannya — dan simpan riwayat di dalam berkasnya: riwayat itu mendaftar setiap perubahan beserta opsinya dan nama orang yang membuatnya, yang memungkinkan Anda menjawab "sejak kapan dokter berkata begitu?" Bab 11 melangkah lebih jauh, dengan versi dan jejak audit untuk tim pakar.

Hentikan, jangan hapus. Ketika sebuah saran menjadi salah — oven baru, resep baru — hentikan node-nya di pembangun alih-alih mencoba menghapusnya. Kasus *cornerstone*-nya tetap ada, pengecualian-pengecualiannya tetap bekerja, riwayatnya mencatat keputusan itu, dan catatan pemakaian menunjukkan kapan node itu terakhir dipakai. VCIRS tidak pernah menghapus node, dan alasannya justru ini: pengetahuan yang dulu benar membawa bukti mengapa pengetahuan sesudahnya benar [8], [9].

Memperbarui program, bukan hanya pengetahuan. Kadang programnya sendiri yang berubah: versi baru vcirs-core, perbaikan pada layanan, Java atau Python yang lebih baru di komputer toko. Perlakukan pembaruan seperti itu dengan lebih hati-hati daripada basis pengetahuan baru, karena ia mengubah cara *setiap* jawaban dihitung. Simpan `.jar` atau `.pyz` lama di samping yang baru (`cake-doctor-2026-09.jar`, `cake-doctor-2026-12.jar`), hentikan layanan, jalankan berkas yang baru, dan jalankan pemeriksaannya. Lalu konsultasikan sepuluh kasus yang jawabannya Anda ketahui — sepuluh baris paling sering dari laporan terakhir adalah pilihan yang baik — dan bandingkan sarannya dengan saran versi lama. Bila ada yang berbeda dan Anda tidak dapat menjelaskan mengapa, hentikan versi baru dan jalankan lagi versi lama: itulah **rollback**, dan hanya makan satu menit karena berkas lamanya masih ada. Berkas basis pengetahuan tidak berubah dalam pembaruan, jadi kedua versi membaca pengetahuan yang sama.

Memeriksa secara terjadwal juga. Pemeriksaan Subbab 15.5 berguna setelah penerapan, dan sama bergunanya setiap pagi. Bila dijadwalkan pukul 7.00, sebelum toko buka, ia menjawab pertanyaan "apakah dokter hidup hari ini?" sebelum ada yang membutuhkannya. Di Linux dan macOS, baris crontab seperti `0 7 * * * cd /opt/cake-doctor && java -jar cake-doctor.jar check || echo "cake doctor is down" | mail -s "cake doctor" wati@example.com` mengirim pesan hanya bila pemeriksaan gagal (`||` menjalankan perintah kedua hanya setelah kegagalan); di Windows, tugas terjadwal dapat menjalankan pemeriksaan dan menampilkan kotak pesan bila kode keluarnya bukan nol. Layanan yang diam-diam gagal pada malam hari dan baru ditemukan oleh pembantu yang memegang kue gosong pukul sepuluh pagi adalah jalan pasti bagi sistem yang diterapkan untuk kehilangan kepercayaan orang.

Ketika basis pengetahuan menjadi besar. Beberapa ratus node mudah; beberapa ribu node menuntut kehati-hatian lebih. Laju pemakaian dari Bab 14 menunjuk node-node yang menjadi sandaran banyak pengetahuan; tinjaulah itu lebih dulu. Daftar *cornerstone* yang terdampak (Bab 3) menjadi penting ketika mengajar: dengan banyak node, pengecualian baru dapat mengubah jawaban kasus-kasus lama yang tidak Anda ingat, dan mesin dapat mendaftarnya. Pengalaman panjang dengan Ripple Down Rules di laboratorium patologi menunjukkan bahwa waktu untuk menambahkan sebuah rule tetap kecil bahkan ketika basis pengetahuan tumbuh sampai ribuan rule, karena sang pakar hanya pernah melihat satu kasus dan daftar perbedaannya [8]; pekerjaan yang bertambah adalah menjaga *bidangnya* tetap terpantau, dan untuk itulah laporan bulanan ada.

Kapan memulai basis pengetahuan kedua. Ketika pertanyaan-pertanyaan dua kelompok kasus sama sekali tidak beririsan — kue dan roti, ponsel dan laptop — dua basis pengetahuan kecil lebih mudah diajar, diperiksa, dan dipahami daripada satu yang besar. Pembangun dan layanan masing-masing menangani satu berkas; jalankan dua layanan di dua port.

15.9 Privasi, Etika, dan Keselamatan

15.9.1 Data pribadi, dalam praktik

Dokter kue tidak mengumpulkan data pribadi, dan itu keputusan rancangan, bukan kebetulan. Banyak sistem pakar tidak seberuntung itu: cek kesehatan mandiri harus tahu gejala, penasihat pinjaman harus tahu penghasilan, triase klinik harus tahu pasiennya. Untuk sistem seperti itu, UU PDP dan, mulai Januari 2027, PP 33/2026 menuntut lebih banyak [4], [5], [6]. Dengan kata-kata sederhana:

- **Ketahui tujuan Anda, dan kumpulkan hanya yang melayaninya.** Setiap pertanyaan dalam basis pengetahuan sebaiknya pertanyaan yang dibutuhkan saran. Bila saran tidak pernah bergantung pada nama seseorang, jangan tanyakan namanya. Bila sebuah variabel tidak dipakai node mana pun, hapus pertanyaannya.
- **Beri tahu orang, dan minta persetujuan bila perlu.** Sebelum konsultasi yang melibatkan data pribadi, katakan apa yang dikumpulkan, mengapa, dan untuk berapa lama; bila persetujuan menjadi dasar yang sah, mintalah dengan jelas dan catat bahwa persetujuan itu diberikan.
- **Pisahkan identitas dari fakta.** Bila Anda harus mengaitkan konsultasi dengan seseorang — berkas pasien, misalnya — simpan identitasnya di tempat terpisah dari catatan pemakaian, dan pakai bilangan acak untuk mengaitkannya. Catatan pemakaian itu sendiri tidak pernah seharusnya membutuhkan nama.
- **Simpan data hanya selama dibutuhkan**, dan hapus sesuai jadwal, seperti yang dilakukan `--keep-days`.
- **Lindungi apa yang Anda simpan.** Taruh layanan di jaringan internal, buat berkas catatan hanya dapat dibaca pengguna layanan, dan simpan cadangan di tempat yang hanya dapat dijangkau pemiliknya.
- **Bersiaplah menghadapi kebocoran.** Bila data pribadi bocor — laptop yang dicuri dengan catatan yang tidak dienkripsi — undang-undang meminta orang-orang yang terdampak diberi tahu paling lambat 3 × 24 jam [4], [6]. Rencana tertulis tentang siapa mengerjakan apa adalah bagian dari penerapan.
- **Hormati hak orang-orang yang datanya ada**: untuk melihat apa yang disimpan tentang mereka, memperbaikinya, dan meminta penghapusannya.

Organisasi yang lebih besar punya kewajiban tambahan, seperti menunjuk pejabat pelindungan data pribadi dan menilai dampak pemrosesan yang berisiko tinggi [5], [6]. Perincian peraturannya masih baru per September 2026, dan lembaga yang akan menegakkannya belum ada pada akhir Agustus [5]; bila sistem pakar Anda menangani data pribadi dalam jumlah besar, bacalah PP 33/2026 itu sendiri atau tanyakan kepada orang yang memahaminya dengan baik sebelum 16 Januari 2027.

15.9.2 Etika memberi saran lewat mesin

Sistem pakar memberi saran, dan orang bertindak berdasarkan saran. Kementerian Komunikasi dan Informatika Indonesia menetapkan nilai-nilai etika untuk kecerdasan artifisial dalam surat edaran tahun 2023 — di antaranya inklusivitas, kemanusiaan, keamanan, aksesibilitas, transparansi, kredibilitas dan akuntabilitas, pelindungan data pribadi, pembangunan dan lingkungan berkelanjutan, serta kekayaan intelektual [10] — dan rekomendasi UNESCO tentang etika kecerdasan artifisial yang berlaku sedunia meminta pengawasan manusia, transparansi dan keterjelasan, keadilan, serta proporsionalitas [11]. Sistem VCIRS memenuhi banyak di antaranya karena wataknya, dan sisanya karena cara ia dipakai.

Jelaskan setiap jawaban. Setiap aplikasi dalam buku ini menunjukkan *mengapa* ia memberi sarannya, fakta demi fakta. Pertahankan penjelasan itu terlihat bagi orang yang menerima sarannya, jangan disembunyikan di catatan. Jawaban yang dapat dijelaskan dapat dipertanyakan, dan jawaban yang dapat dipertanyakan dapat diperbaiki.

Pastikan ada manusia yang bertanggung jawab. Pengetahuannya berasal dari orang-orang yang melihat kasus-kasus terselesaikan, dan setiap node mencatat siapa yang mengajarkannya. Sistem ini tidak menggantikan pertimbangan mereka; ia menyebarkannya. Katakan hal itu kepada

orang-orang yang memakainya, dan permudah mereka menghubungi manusia ketika sarannya tidak cocok.

Katakan "saya tidak tahu". "Belum ada saran" yang jujur dari Bab 3 dan kalimat tetap dari Bab 13 adalah fitur etis, bukan kekurangan. Sistem yang selalu menjawab kadang-kadang akan menjawab salah dengan penuh keyakinan.

Kenali tempat taruhannya tinggi. Untuk kesehatan, uang, dan keselamatan, berikan hanya saran yang sudah diperiksa orang yang berwenang, sertakan aturan rujukan yang jelas — *temui dokter atau bidan bila…*, *tanyakan kepada teknisi listrik bersertifikat sebelum…* — dan jangan pernah menyajikan keluarannya sebagai diagnosis atau rekomendasi keuangan [12]. Basis pengetahuan dapat memuat pengetahuan rujukan itu sebagai node seperti yang lain, dan memang seharusnya begitu.

Waspadai ketidakadilan. Basis pengetahuan belajar dari kasus-kasus yang dilihat para pengajarnya. Bila mereka hanya melihat satu jenis pelanggan, ia mungkin kurang baik melayani yang lain. Laporan pemakaian adalah pemeriksaan pertama: banyak kasus tak terjawab dari satu kelompok orang adalah tanda bahwa pengetahuannya belum mencakup mereka.

Jangan memakai catatan untuk tujuan yang bukan tujuannya. Catatan pemakaian yang dikumpulkan untuk memperbaiki saran tidak boleh diam-diam menjadi alat untuk menilai para pembantu yang memakainya.

15.10 Ke Mana Selanjutnya

Buku ini dimulai dengan seorang pengemudi becak, seorang penjual siomay, dan seorang tukang parkir, dan dengan sebuah janji: bahwa orang yang tidak tahu apa-apa tentang pemrograman dapat membangun sistem pakar yang belajar. Lima belas bab kemudian, Anda sudah membangun belasan sistem seperti itu, dari penghemat tagihan listrik dengan sembilan aturan sampai pembangun yang memungkinkan siapa saja membuat sistem pakarnya sendiri dan layanan yang menjalankannya setiap hari. Berikut beberapa arah dari sini.

Menyelami gagasannya lebih dalam. Sumber mesin buku ini adalah Variable-Centered Intelligent Rule System, dalam tesisnya, makalah-makalah konferensinya, dan revisi-revisinya [9], [13], [14]. Akarnya ada pada Ripple Down Rules, yang buku tunggal terbaiknya adalah karya Compton dan Kang [8], dan tinjauan atas dua dasawarsa penelitian [15] menunjukkan betapa luas gagasan itu diterapkan. Membacanya sekarang, setelah membangun aplikasi-aplikasinya, Anda akan mengenali setiap konsepnya.

Memperluas mesinnya. vcirs-core cukup kecil untuk dibaca dalam satu akhir pekan, dan ditulis dua kali, sehingga setiap gagasan dapat dicoba dalam bahasa yang Anda sukai. Beberapa perluasan yang disarankan sumber-sumbernya dan belum dibangun buku ini: bilangan fuzzy untuk fakta yang tidak pasti (Bab 9 membangun nilai-nilai fuzzy; makalah VCIRS fuzzy melangkah lebih jauh [16]), konklusi majemuk di setiap aplikasi, dan basis pengetahuan yang dipakai bersama beberapa toko yang menggabungkan pengetahuan dengan saling mengajarkan kasus-kasus tak terjawab.

Menggabungkannya dengan belajar dari data. Bab 12 mengubah pohon keputusan menjadi basis pengetahuan VCIRS; Bab 13 menaruh model bahasa di depannya. Keduanya menunjukkan pola yang sama: biarkan metode statistik mengusulkan, dan biarkan pengetahuan yang sudah diperiksa memutuskan. Pola itu akan tetap berguna seiring metode statistik terus membaik.

Membagikan yang Anda bangun, dengan hati-hati. Basis pengetahuan itu kecil, mudah dibaca, dan mudah dipindahkan, sehingga mudah dibagikan: adik Bu Wati mengimpor dokter kue dan menumbuhkan versinya sendiri. Ketika Anda membagikannya, bagikan juga riwayatnya, agar orang berikutnya dapat melihat siapa mengajarkan apa dan kapan; hapus darinya apa pun yang bersifat pribadi yang mungkin termuat dalam sebuah kasus *cornerstone*; dan katakan terus terang untuk apa dan oleh siapa ia dibangun. Basis pengetahuan tentang kue di Surabaya bisa saja keliru tentang kue di Malang yang lebih tinggi letaknya, dan pemilik barunya berhak tahu asal-usulnya sebelum memercayainya.

Membangun untuk komunitas Anda sendiri. Aplikasi terpenting dari buku ini adalah aplikasi yang akan Anda bangun berikutnya — untuk toko keluarga Anda, kelompok tani di desa Anda, kantor masjid atau gereja Anda, sekolah Anda. Mulailah dengan beberapa kasus nyata, ajarkan sisanya ketika datang, simpan cadangan, baca laporannya sebulan sekali, dan pastikan ada manusia yang bertanggung jawab. Hanya itu yang dibutuhkan sebuah sistem pakar untuk terus menolong selama bertahun-tahun.

15.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Menjalankan layanan dari jendela terminal. Menutup jendela, atau keluar dari akun, menghentikannya. *Hindari:* pasang sebagai layanan (systemd, Task Scheduler, launchd) dengan kebijakan menyala ulang.

Mengajar di salinan yang sudah diterapkan. Dua salinan yang sama-sama berubah tidak dapat digabungkan. *Hindari:* mengajarlah di satu tempat, lalu salin hasil yang sudah diperiksa ke layanan.

Mencadangkan tanpa memeriksa. Cadangan yang tidak dapat dipulihkan baru ketahuan ketika dibutuhkan. *Hindari:* beri sidik jari pada setiap salinan, periksa salinan secara teratur, dan berlatihlah memulihkan.

Satu salinan di satu tempat. Kebakaran, pencurian, atau diska yang rusak membawa serta berkas asli dan cadangannya sekaligus. *Hindari:* ikuti aturan 3-2-1 [3].

Mencatat segalanya “untuk berjaga-jaga”. Nama, nomor telepon, alamat IP, dan teks bebas mengubah catatan perawatan menjadi data pribadi dengan kewajiban hukum. *Hindari:* catat hanya yang dibutuhkan pertanyaan perawatan, dan matikan catatan akses bawaan server web.

Menyimpan catatan selamanya. Catatan lama adalah beban, bukan aset. *Hindari:* hapus sesuai jadwal, seperti yang dilakukan `--keep-days`.

Menghitung uji sebagai pemakaian. Pemeriksaan setelah penerapan menggelembungkan statistik. *Hindari:* tandai permintaan uji (`X-Test`) dan jangan masukkan ke catatan.

Membuka layanan ke internet. Layanan ini tidak punya kata sandi dan tidak punya enkripsi. *Hindari:* dengarkan hanya di `127.0.0.1` atau jaringan toko; bila harus dapat dijangkau dari luar, taruh di belakang server web yang dikonfigurasi dengan benar dengan HTTPS dan autentikasi, dan mintalah bantuan orang yang berpengalaman.

Menghapus saran yang menjadi salah. Menghapus menghilangkan bukti dan merusak pengecualian. *Hindari:* hentikan node-nya; riwayatnya mencatat alasannya.

Tidak pernah membaca laporan. Catatan yang tidak dibaca siapa pun adalah biaya tanpa manfaat. *Hindari:* masukkan satu jam bulanan itu ke kalender.

15.12 Ringkasan Bab

- **Menerapkan** berarti membuat aplikasi berjalan tanpa pemrogramnya: dokter kue menjadi **layanan saran** dengan tiga pintu, `/health`, `/questions`, dan `/advise` (**Kode 15.1**, **Kode 15.2**), yang membaca basis pengetahuan, memuatnya ulang ketika salinan baru datang, dan tidak pernah mengubahnya.
- Layanan dan alat-alatnya dikemas ke dalam **satu berkas**, `.jar` atau `.pyz`, dengan empat perintah (**Kode 15.3**, **Kode 15.4**, **Kode 15.5**), dan dipasang agar berjalan sendiri dengan systemd, Task Scheduler, atau launchd (**Gambar 15.1**).
- **Pemeriksaan** singkat setelah setiap penerapan menguji kesehatan, pertanyaan, satu jawaban yang sudah diketahui, dan satu ketiadaan jawaban yang sudah diketahui (**Kode 15.6**, **Kode 15.7**).
- **Cadangan** hanya dibuat dari berkas yang dapat dibaca, diberi sidik jari SHA-256, diperiksa secara teratur, dipulihkan hanya bila utuh, dan disimpan dalam tiga salinan (**Kode 15.8**, **Kode 15.9**).
- **Catatan pemakaian** merekam waktu, fakta, node, dan saran — tidak ada apa pun tentang orang — dan **laporan** bulanan menunjukkan berapa saran yang diberikan, node mana yang tidak pernah dipakai, dan kasus tak terjawab mana yang perlu diajarkan berikutnya (**Kode 15.10**, **Kode 15.11**, **Gambar 15.2**); baris-baris lama dihapus sesuai jadwal.
- Perawatan adalah **siklus** pemakaian, pencatatan, pembacaan, pengajaran, pemeriksaan, dan penerapan (**Gambar 15.3**); saran yang menjadi salah dihentikan, tidak pernah dihapus.
- **UU PDP** (UU 27/2022) berlaku untuk data pribadi; peraturan pelaksanaannya, **PP 33/2026**, ditetapkan pada 16 Juli 2026 dan mulai berlaku pada 16 Januari 2027 [4], [5]. Kepatuhan paling sederhana adalah tidak mengumpulkan data pribadi; bila dibutuhkan, kumpulkan sedikit, simpan sebentar, lindungi, dan bersiaplah menghadapi kebocoran. Penjelasan, manusia yang bertanggung jawab, jawaban “saya tidak tahu” yang jujur, dan aturan rujukan membuat saran lewat mesin menjadi etis [10], [11].

15.13 Soal Latihan

15.1 Daftarlah ketiga pintu layanan saran dan, untuk masing-masing, satu keadaan di toko ketika pintu itu dipakai.

15.2 Bu Wati menyalin `cake-kb.json` yang baru ke folder layanan dengan Windows Explorer, dan penyalinan itu memakan dua detik. Seorang pembantu meminta saran selama dua detik itu. Apa yang terjadi, dan mengapa jawabannya tetap benar?

15.3 Dari laporan di Subbab 15.7, hitunglah dengan tangan persentase konsultasi yang dijawab *bake longer#1* dan *take it out earlier#1G*, lalu cocokkan dengan laporannya.

15.4 Para pembantu bertanya apakah catatan juga dapat merekam pembantu mana yang bertanya, “untuk melihat siapa yang perlu dilatih lagi”. Bahaslah permintaan ini berdasarkan Subbab 15.9. Bila Bu Wati setuju, bagaimana Anda melakukannya dengan data pribadi sesedikit mungkin?

15.5 Tuliskan baris-baris tugas terjadwal yang, setiap Minggu pukul 22.00, menyimpan cadangan ke *flashdisk* yang terpasang di `/media/usb` (Linux) lalu menjalankan `backup check` pada *flashdisk* itu. Terhadap apa pemeriksaan itu melindungi yang tidak dilindungi cadangan malam hari?

15.6 Tuliskan *launch agent* `.plist` untuk macOS yang menjalankan layanan dari `~/cake-doctor` ketika masuk dan menyalakannya ulang bila berhenti.

15.7 Laporan menyatakan *lower the heat#1* tidak pernah memberi saran. Berikan dua penjelasan yang berbeda, dan katakan apa yang sebaiknya dilakukan Bu Wati dalam setiap kasus.

15.8 Sebuah klinik ingin menerapkan cek kesehatan mandiri dari Bab 5 sebagai layanan seperti ini. Sebutkan empat perubahan pada catatan pemakaian dan penerapan yang dibutuhkan klinik itu dan tidak dibutuhkan toko kue.

15.14 Penyelesaian Soal Latihan

Penyelesaian 15.1. `GET /health`: setelah memasang atau memperbarui layanan, atau ketika seorang pembantu mengira layanan tidak menjawab, pemeriksaan (atau peramban) menanyakannya. `GET /questions`: layar tablet di etalase menampilkan pertanyaan-pertanyaan dalam urutan yang paling berguna, dengan jawaban-jawaban biasanya sebagai tombol. `POST /advise`: tablet mengirim jawaban yang dipilih untuk kue yang gagal dan menampilkan saran beserta penjelasannya; antarmuka obrolan Bab 13 dapat memakai pintu yang sama.

Penyelesaian 15.2. Selama penyalinan, waktu perubahan berkas berubah, sehingga `fresh_kb` mencoba memuatnya. Bila berkas masih setengah tertulis, pemuatan gagal — JSON-nya belum lengkap — dan layanan mempertahankan basis pengetahuan lama serta menulis "could not reload … keeping the old knowledge base" ke keluaran galatnya. Pembantu mendapat saran dari basis pengetahuan lama, yang masih benar sampai sesaat sebelumnya. Karena layanan mengingat waktu perubahan yang baru, ia tidak langsung mencoba lagi; kali berikutnya berkas berubah (ketika penyalinan selesai dan waktunya ditetapkan lagi), ia memuat berkas yang lengkap. Kebiasaan yang lebih aman adalah menyalin berkas dengan nama lain, seperti `cake-kb.new`, lalu menamainya ulang menjadi `cake-kb.json`: penamaan ulang mengganti berkas dalam satu langkah.

Penyelesaian 15.3. *bake longer#1* memberi saran 13 kali dari 120: 13/120 = 0,10833…, yaitu 10,8%, seperti di laporan. *take it out earlier#1G* memberi saran 6 kali: 6/120 = 0,05 = 5,0%, seperti di laporan.

Penyelesaian 15.4. Merekam siapa yang bertanya mengubah catatan pemakaian menjadi data pribadi tentang karyawan dan mengubah tujuannya dari memperbaiki saran menjadi menilai orang, hal yang diperingatkan Subbab 15.9.2. Menurut UU PDP toko itu memerlukan tujuan yang jelas, harus memberi tahu para pembantu, serta harus melindungi dan akhirnya menghapus data itu [4]. Jawaban yang lebih baik untuk kebutuhan sesungguhnya — "siapa yang perlu dilatih lagi?" — adalah membahas kasus-kasus tak terjawab dan kasus-kasus yang sering terjadi bersama para pembantu, tanpa nama. Bila Bu Wati tetap menginginkannya, rekamlah kode pendek untuk setiap pembantu (bukan nama), beri tahu para pembantu apa yang direkam dan mengapa, simpan tabel yang mengaitkan kode dengan nama hanya di laptopnya sendiri, dan hapus kode-kode itu dari catatan setelah pelatihan.

Penyelesaian 15.5. Cara paling sederhana adalah membiarkan alat pencadangan sendiri menyimpan ke *flashdisk*, sehingga *flashdisk* punya `index.txt` sendiri dengan sidik jari. Tulis skrip `weekly-usb.sh` di samping kedua skrip lainnya:

```
#!/bin/sh
cd /opt/cake-doctor || exit 1
java -jar cake-doctor.jar backup save data/cake-kb.json \
    /media/usb/backups --keep 52 &&
java -jar cake-doctor.jar backup check /media/usb/backups
```

dan tambahkan `0 22 * * 0 /opt/cake-doctor/weekly-usb.sh` dengan `crontab -e`: menit 0, jam 22, tanggal apa saja, bulan apa saja, hari ke-0 dalam minggu (Minggu). `--keep 52` menyimpan salinan mingguan selama setahun, dan `&&` menjalankan pemeriksaan hanya bila penyimpanan berhasil. Salinan mingguan di perangkat terpisah melindungi dari rusaknya diska komputer layanan itu sendiri, yang akan membawa serta cadangan-cadangan malam hari; pemeriksaan pada *flashdisk* melindungi dari *flashdisk* itu sendiri yang diam-diam merusak berkas, hal yang lazim pada *flashdisk* murah. Sebulan sekali, bawalah *flashdisk* itu pulang: itulah "satu salinan di tempat lain" dari aturan 3-2-1.

Penyelesaian 15.6. Simpan sebagai `~/Library/LaunchAgents/com.example.cakedoctor.plist`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
  "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>com.example.cakedoctor</string>
  <key>WorkingDirectory</key>
  <string>/Users/wati/cake-doctor</string>
  <key>ProgramArguments</key>
  <array>
    <string>/usr/bin/java</string>
    <string>-jar</string>
    <string>cake-doctor.jar</string>
    <string>serve</string>
    <string>--kb</string>
    <string>data/cake-kb.json</string>
    <string>--logs</string>
    <string>logs</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
  <key>KeepAlive</key>
  <true/>
</dict>
</plist>
```

lalu aktifkan dengan `launchctl load ~/Library/LaunchAgents/com.example.cakedoctor.plist`. Ganti `/Users/wati` dengan folder rumah Anda, dan `/usr/bin/java` dengan keluaran `which java` bila Java berada di tempat lain.

Penyelesaian 15.7. (1) Masalahnya jarang: bagian atas yang retak dengan oven yang *panas* memang tidak terjadi selama tiga bulan, karena ovennya diatur dengan benar. Dalam hal itu node-nya baik-baik saja; pertahankan dan lihat lagi setahun kemudian. (2) Pengetahuan node itu belum lengkap atau salah: bagian atas yang retak memang terjadi (dua kali, dengan oven yang normal), tetapi karena sebab lain. Dalam hal itu Bu Wati sebaiknya mengajarkan penyebab yang sebenarnya ketika kue retak berikutnya terjadi, dan, bila ia sampai yakin bahwa saran untuk oven panas juga salah, menghentikan node itu. Laporan tidak dapat membedakan keduanya; orang yang memahami kue dapat.

Penyelesaian 15.8. (1) Fakta-faktanya adalah data kesehatan, yang oleh UU PDP diperlakukan sebagai data pribadi yang bersifat spesifik [4]; klinik membutuhkan dasar yang sah — biasanya persetujuan yang tegas — dan harus memberi tahu pasien apa yang disimpan dan mengapa. (2) Catatan itu sendiri tidak boleh dapat dikaitkan dengan seorang pasien; bila kaitan dibutuhkan, simpan terpisah, dilindungi, dengan pengenal acak. (3) Akses ke layanan dan catatan harus dibatasi pada staf klinik, dengan layanan di jaringan internal dan berkas-berkas hanya dapat dibaca pengguna layanan; cadangan harus dienkripsi. (4) Masa simpan sebaiknya ditetapkan

kebijakan klinik dan dijaga oleh penghapusan terjadwal, dan rencana kebocoran dengan pemberitahuan 3 × 24 jam harus dituliskan. Selain itu, basis pengetahuannya harus memuat aturan rujukan dan tidak pernah menyajikan keluarannya sebagai diagnosis [12].

Daftar Pustaka

[1] Oracle, “The jar command,” *Java Development Kit Version 25 Tool Specifications*, 2025. [Online]. Available: https://docs.oracle.com/en/java/javase/25/docs/specs/man/jar.html

[2] Python Software Foundation, “zipapp — Manage executable Python zip archives,” *The Python Standard Library*, 3.14. [Online]. Available: https://docs.python.org/3/library/zipapp.html

[3] Cybersecurity and Infrastructure Security Agency (US-CERT), “Data backup options,” 2012. [Online]. Available: https://www.cisa.gov/sites/default/files/publications/data_backup_options.pdf

[4] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*, Lembaran Negara RI Tahun 2022 Nomor 196, Oct. 17, 2022. [Online]. Available: https://peraturan.bpk.go.id/Details/229798/uu-no-27-tahun-2022

[5] C. Aw, T. Darmawan, A. Rabbani, F. Seow, and C. O’Leary, “Indonesia’s Personal Data Protection Law implementing regulation arrives quietly: a first look at GR 33/2026,” Hogan Lovells, Aug. 31, 2026. [Online]. Available: https://www.hlc.com/en/publications/indonesias-personal-data-protection-law-implementing-regulation-arrives-quietly

[6] Robere & Associates, “PP 33 2026: panduan pelindungan data pribadi,” 2026. [Online]. Available: https://robere.co.id/id/pp-33-2026-pelindungan-data-pribadi/

[7] systemd project, “systemd.service — Service unit configuration,” *systemd manual pages*. [Online]. Available: https://www.freedesktop.org/software/systemd/man/latest/systemd.service.html

[8] P. Compton and B. H. Kang, *Ripple-Down Rules: The Alternative to Machine Learning*. Boca Raton, FL, USA: CRC Press, 2021.

[9] M. M. I. Subakti, “A Variable-Centered Intelligent Rule System,” M.Sc.Eng. thesis, Dept. of Computer Science and Information Engineering, Taipei, Taiwan, Jun. 2005.

[10] Kementerian Komunikasi dan Informatika Republik Indonesia, *Surat Edaran Menteri Komunikasi dan Informatika Nomor 9 Tahun 2023 tentang Etika Kecerdasan Artifisial*, Dec. 19, 2023.

[11] UNESCO, *Recommendation on the Ethics of Artificial Intelligence*. Paris, France: UNESCO, 2021.

[12] I. Subakti and R. Hidayatullah, “Aplikasi sistem pakar untuk diagnosis awal gangguan kesehatan secara mandiri menggunakan Variable-Centered Intelligent Rule System,” vol. 6, no. 1, pp. 11–16, Jan. 2007.

[13] I. Subakti, “A variable-centered intelligent rule system,” in *Proc. 1st Annual Int. Conf. Information and Communication Technology Seminar (ICTS 2005)*, vol. 1, no. 1, Surabaya, Indonesia, Aug. 2005, pp. 167–174.

[14] I. Subakti, “Some revisions in VCIRS and cases reconstructing perspectives,” in *Proc. Information and Communication Technology Seminar (ICTS 2006)*, Surabaya, Indonesia, 2006.

[15] D. Richards, “Two decades of ripple down rules research,” *The Knowledge Engineering Review*, vol. 24, no. 2, pp. 159–184, 2009.

[16] I. Subakti and O. Wijayanto, “Penerapan konsep fuzzy dalam Variable-Centered Intelligent Rule System (studi kasus: pemilihan jurusan di Chinese University of Hongkong).”