Sistem Cerdas

Edisi pertama – versi online

MM Irfan Subakti, SKom, MScEng, MPhil, Аспирант
ММ Ирфан Субакти 司馬伊凡 老師

subakti.com & rusindo.com

@rusindocom

Surabaya, Indonesia

2018

Aku persembahkan buku ini buat:

Елена Калашникова (Elena Kalashnikova), Ярик (Yarik) dan keluargaku tercinta

KATA PENGANTAR

Salam sejahtera ^__^

Dalam buku ini kita belajar untuk membuat suatu mesin yang cerdas, terinspirasi dari cara kerja manusia dalam memperoleh kecerdasannya. Juga dipelajari bagaimana mesin dapat berevolusi sehingga seiring waktu berjalan, mesin yang kita buat juga akan semakin cerdas dan bisa mengikuti perkembangan jaman.

Adanya mesin yang cerdas tersebut tak hanya dalam tataran konseptual atau filosofi saja, tetapi juga dapat diimplementasikan secara nyata dalam kehidupan sehari-hari, sehingga bisa memberikan manfaat yang jelas di tengah kehidupan masyarakat. Materi dalam buku bisa diimplementasikan dengan kemampuran pemrograman yang baik.

Semoga apa yang ada di buku ini bermanfaat bagi pembaca semua. Tentu saja, kritik dan saran dipersilakan. Pembaca dapat menghubungi penulis di: yifana@gmail.com

Surabaya, September 2018

Penulis

Daftar Isi

Halaman

DAFTAR TABEL

Halaman

Daftar Gambar

Halaman

Bab 1 Pendahuluan

1.1 Gambaran Umum

Mesin yang cerdas, terinspirasi dari cara kerja manusia dalam memperoleh kecerdasannya. Mesin dapat berevolusi sehingga seiring waktu berjalan, mesin yang kita buat juga akan semakin cerdas dan bisa mengikuti perkembangan jaman.

Adanya mesin yang cerdas tersebut tak hanya dalam tataran teori, namun juga dapat diimplementasikan secara nyata dalam kehidupan kita. Materi dalam buku bisa diimplementasikan dengan kemampuran pemrograman yang baik, untuk memberikan manfaat yang sebenarnya di tengah kehidupan masyarakat.

1.2 Tujuan

Tujuan yang ingin dicapai setelah membaca dan mempraktekkan buku ini adalah:

- Mampu memahami dan mengimplementasikan Sistem Berbasis Pengetahuan (Knowledge-based Systems) khususnya Sistem Berbasis Aturan (Rule-Based Systems), dan juga Ripple Down Rules (RDR).
- Mampu memahami dan mengimplementasikan Simulated Annealing, Algoritma Genetika (Genetic Algorithms) dan Genetic Simulated Annealing.
- Mampu memahami dan mengimplementasikan Algoritma Genetika, Simulated Annealing dan Genetic Simulated Annealing dalam Sistem Sistem Berbasis Aturan.
- Mampu memahami dan mengimplementasikan Sistem Fuzzy dalam Sistem Sistem Berbasis Aturan.
- Mampu memahami dan mengimplementasikan Variable-Centered Intelligent Rule System (VCIRS).

1.3 Topik yang Dibahas

Topik-topik yang dibahas dalam buku ini adalah sebagai berikut:

- Sistem Pakar, Sistem Berbasis Pengetahuan dan Sistem Berbasis Aturan
- Ripple Down Rules
- Algoritma Genetika
- Simulated Annealing
- Genetic Simulated Annealing
- Sistem Fuzzy
- Variable-Centered Intelligent Rule System

1.4 Pustaka Acuan

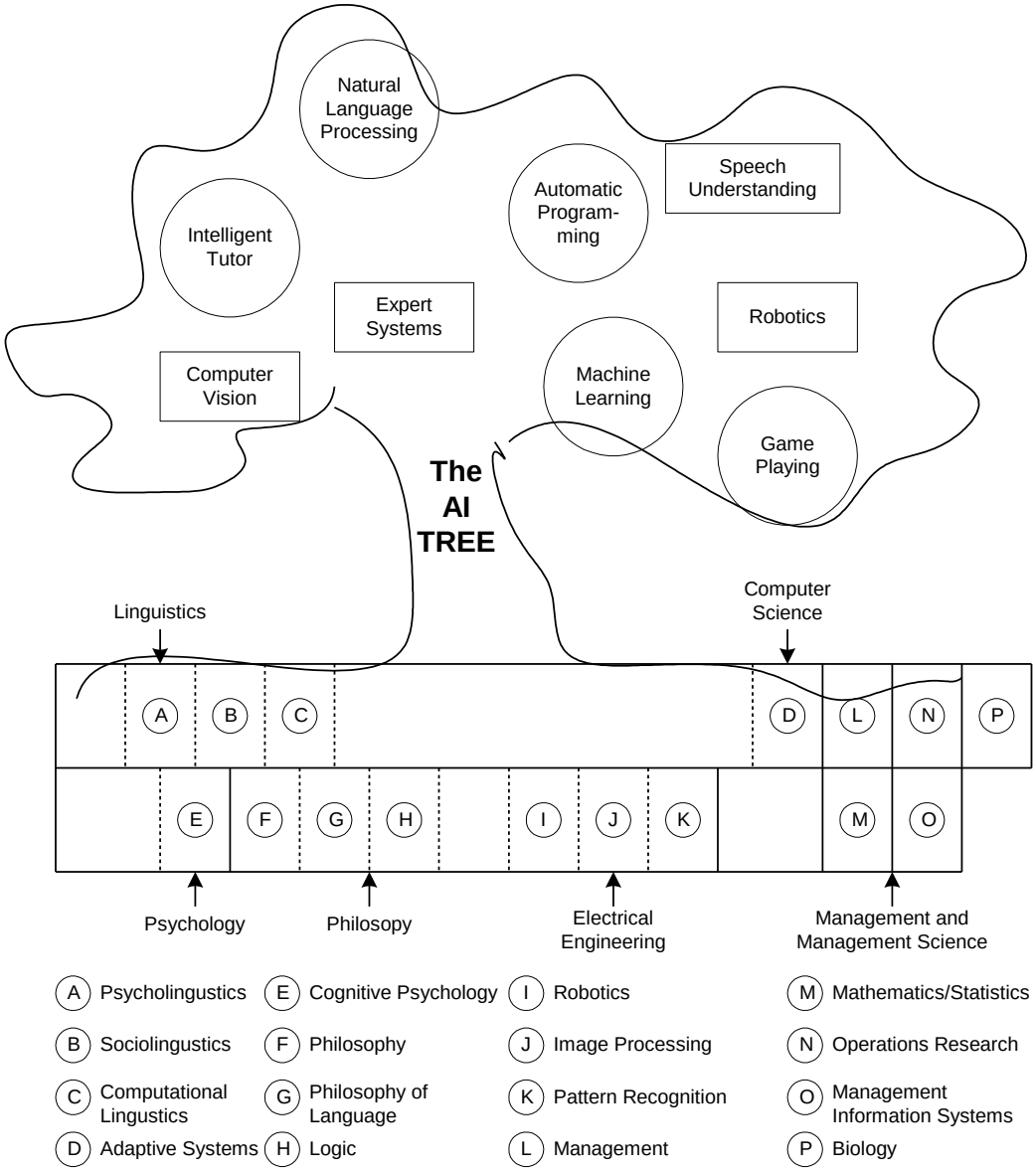
Pustaka yang dipakai sebagai acuan dalam buku ini dapat dilihat pada Daftar Pustaka di bagian terakhir.

Bab 2 Sistem Pakar dalam Kecerdasan Buatan

Dalam bab ini kita akan mempelajari sistem pakar dan hubungannya dengan bidang kecerdasan buatan. Pustaka yang dipakai adalah dari E. Turban [Tur95], A.J. Gonzalez dan D.D. Dankel [Gon93] dan J.P. Ignizio [Ign91].

2.1 Bidang Kecerdasan Buatan

Kecerdasan Buatan adalah bidang yang sangat luas cakupannya, seperti digambarkan pada gambar 2.1 di bawah ini.



Gambar 2.1 Bidang Kecerdasan Buatan

Seperti terlihat di gambar 2.1 di atas, Expert Systems (Sistem Pakar) adalah salah satu bagian dari bidang Kecerdasan Buatan. Salah satu dari Sistem Pakar yang ada adalah Sistem Berbasis Pengetahuan (SBP).

Sedangkan salah satu sistem dari SBP adalah Sistem Berbasis Rule (Rule-Based Systems) dimana basis pengetahuannya (Knowledge-Based) berupa aturan-aturan (rules).

2.2 Ide Dibuatnya Sistem Pakar

Semakin tak terstruktur suatu situasi, maka solusinya akan lebih spesifik. Sistem Pakar dibuat untuk menirukan seorang pakar/ahli.

Sistem Pakar adalah paket hardware dan software yang digunakan sebagai pengambil keputusan dan/atau penyelesaian masalah; yang dapat mencapai level yang setara atau kadang malah melebihi seorang pakar/ahli, pada satu area masalah yang spesifik dan biasanya lebih sempit.

Merupakan cabang dari aplikasi Kecerdasan Buatan.

Ide dasarnya sederhana. Kepakaran ditransfer dari seorang pakar ke komputer. Pengetahuan ini lalu disimpan disitu dan user dapat meminta saran spesifik yang dibutuhkannya. Komputer dapat mencari, mengolah dan menampilkan kesimpulan yang spesifik. Dan seperti seorang pakar, saran tersebut bisa dimanfaatkan oleh orang yang bukan pakar berikut penjelasannya yang berisi logika penalaran di balik saran itu.

2.3 Definisi Sistem Pakar

Adalah program pemberi advis/nasehat yang terkomputerisasi yang ditujukan untuk meniru proses reasoning (pertimbangan) dan pengetahuan dari pakar dalam menyelesaikan permasalahan masalah yang spesifik. Bidang ini digunakan lebih banyak daripada penggunaan bidang-bidang Kecerdasan Buatan lainnya. Sistem Pakar menarik minat yang besar dalam suatu organisasi disebabkan kemampuannya dalam meningkatkan produktifitas dan dalam meningkatkan gugus kerja di pelbagai bidang tertentu dimana pakar manusia akan mengalami kesulitan dalam mendapatkan dan mempertahankan kemampuan itu.

Pakar manusia cenderung untuk menjadi spesialis dalam bidang keahlian tertentu yang relatif sempit. Umumnya pakar memiliki karakteristik ini: mereka menyelesaikan masalah dengan cepat dan cukup akurat, menjelaskan what/apa (dan terkadang how/bagaimana) yang mereka kerjakan, mempertimbangkan reliabelitas konklusinya, mengetahui kapan

jalan buntu menghadang, dan mereka berkomunikasi dengan para pakar lainnya. Mereka juga belajar dari pengalaman, mengubah cara pandangnya untuk menyesuaikan dengan masalah, juga mentransfer pengetahuan dari satu domain ke domain yang lain. Akhirnya, mereka menggunakan pelbagai tool, seperti aturan jempol, model matematis, dan simulasi detil untuk mendukung keputusan yang diambil.

Pengetahuan (knowledge) adalah sumber utama, dan ini seringkali cuma dimiliki oleh sebagian kecil pakar. Tentu saja diperlukan untuk menyimpan pengetahuan ini sehingga orang lain dapat menggunakannya. Sang pakar bisa saja menderita sakit atau meninggal dunia dan pengetahuan yang biasanya ada menjadi tiada lagi. Buku dan manual bisa saja menyimpan pelbagai pengetahuan, tetapi ini juga memberikan persoalan lain dalam aplikasi menampilkan kembali pengetahuan itu kepada orang yang membutuhkannya. Sistem Pakar menyediakan pengertian langsung dari aplikasi kepakaran.

Tujuan dari Sistem Pakar bukanlah menggantikan para pakar, tetapi hanya untuk membuat pengetahuan dan pengalaman para pakar itu tersimpan dan tersedia lebih luas dan leluasa. Umumnya, memang lebih banyak masalah yang ada yang mendesak untuk diselesaikan daripada keberadaan para pakar untuk menangani pelbagai persoalan. Sistem Pakar mengijinkan orang lain untuk meningkatkan produktifitas, memperbaiki kualitas keputusannya, dan menyelesaikan masalah di saat seorang pakar tidak ada.

2.4 Pengolahan Bahasa Alami

Terkait dengan Sistem Pakar, maka muncullah bidang baru yang disebut dengan Natural Language Processing (Pengolahan Bahasa Alami) yang merupakan bagian penting dari suatu Sistem Pakar dalam hal menjembatani antara bahasa yang dikenal manusia dan bahasa komputer/mesin yang dikenal oleh komputer/mesin yang digunakan oleh Sistem Pakar.

Teknologi bahasa alami memberikan komputer kemampuan untuk berkomunikasi dengan komputer lain dengan bahasa aslinya. Teknologi ini mengijinkan suatu jenis percakapan antarmuka, yang berbeda bila dibandingkan dengan istilah, sintaks, perintah bahasa pemrograman.

Bidang Pengolahan Bahasa Alami ini dibagi menjadi 2 sub bidang:

1. Pemahaman bahasa alami (**Natural language understanding**), mempelajari metode yang menjadikan komputer memahami perintah-perintah yang diberikan dalam bahasa Inggris, sehingga komputer dapat lebih mudah memahami manusia.
2. Pembuatan bahasa alami (**Natural language generation**), dibuat agar komputer bisa berbahasa Inggris umum, sehingga manusia lebih mudah memahami komputer.

2.5 Sistem Pakar Dibandingkan dengan Sistem Lain

Dibandingkan dengan sistem yang lain, terdapat pelbagai atribut yang dapat membedakan Sistem Pakar dengan yang lain, seperti ditampilkan pada tabel 2.1 di bawah ini.

Tabel 2.1 Sistem Pakar dan Sistem yang Lain

Dimensi	**Transactions Processing Systems (TPS)**	**Management Information Systems (MIS)**	**Decision Support Systems (DSS)**	**Expert System (ES)**	**Executive Information Systems (EIS)**
Aplikasi	Payroll, inventory, record keeping, informasi produksi dan penjualan	Kontrol produksi, peramalan penjualan, monitoring	Perencanaan strategis jangka panjang, area permasalahan terintegrasi secara kompleks	Perencanaan strategis diagnosis, perencanaan kontrol internal, strategi-strategi	Dukungan pada pengambilan keputusan di level puncak, pemindaian lingkungan
Fokus	Transaksi data	Informasi	Keputusan-keputusan, fleksibelitas, kemudahan pengguna	Inferensia, transfer kepakaran	Penelusuran, kontrol, "Drill down"
Database	Unik untuk tiap aplikasi, update secara batch	Akses interaktif oleh programer	Sistem manajemen database, akses interaktif, pengetahuan faktual	Pengetahuan prosedural dan faktual; knowledge base (fakta-fakta, rules)	Eksternal (online) dan berskala perusahan (corporate), akses lebar level perusahaan (enterprise) (untuk semua basis data)
Kemampuan pengambilan keputusan	Tak ada keputusan yang dapat diambil	Permasalahan-permasalahan pengarahan terstruktur menggunakan tool-tool manajemen sains konvensional	Permasalahan-permasalahan semi terstruktur, model manajemen sains terintegrasi, paduan dari pendapat dan pemodelan	Sistem membuat keputusan yang kompleks, tak terstruktur; menggunakan rules (heuristics)	Hanya jika dikombinasikan dengan SPK
Manipulasi	Numerik	Numerik	Numerik	Simbolik	Numerik

Dimensi	Transactions Processing Systems (TPS)	Management Information Systems (MIS)	Decision Support Systems (DSS)	Expert System (ES)	Executive Information Systems (EIS)
					(utamanya); kadang simbolik
Jenis informasi	Laporan ringkasan, operasional	Laporan penjadwalan dan permintaan, alur terstruktur, pelaporan pengecualian	Informasi untuk mendukung keputusan spesifik	Advis dan penjelasan	Akses status, pelaporan pengecualian, indikator kunci
Tingkat organisasi tertinggi yang dilayani	Submanajerial, manajemen rendah	Manajemen menengah	Analis dan manajer	Manajer dan spesialis	(Hanya) eksekutif senior
Daya pendorong	Kegunaan	keefisienan	Keefektifan	Keefektifan dan kegunaan	Ketepatan waktu

2.6 Hubungan Sistem Pakar dan Sistem Pendukung Keputusan

Sistem Pakar dan Sistem Pendukung Keputusan (SPK) berbeda dan tak berhubungan dengan sistem yang terkomputerisasi.

Disiplin antara Sistem Pakar dan SPK berkembang paralel, tapi saling tak tergantung dan berjalan sendiri-sendiri. Cuma sekarang kita bisa mencoba menggabungkan potensi dari keduanya.

Menurut kenyataannya, disebabkan karena perbedaan kapabilitas diantara kedua tool, mereka dapat mengkomplemen (mengisi) satu sama lain, membuatnya menjadi powerful (berdaya guna), terintegrasi, sistem yang berbasis komputer, yang jelas dapat meningkatkan pengambilan keputusan manajerial.

2.7 Dukungan dari Pengambilan Keputusan

Tentu saja, bila dibandingkan dengan SPK, Sistem Pakar memiliki perbedaan bila dikaitkan dengan pelbagai hal yang berhubungan dengan pengambilan keputusan.

Perbedaan tersebut dapat dilihat pada tabel 2.2 di bawah ini.

Tabel 2.2 Sistem Pakar dan dukungan pengambilan keputusan

	SPK	Sistem Pakar
Tujuan	Membantu orang yang mengambil keputusan	Menirukan (menyerupakan) penasehat (orang) dan menggantikan mereka
Siapa yang membuat rekomendasi (keputusan)?	Orang dan/atau sistem	Sistem
Orientasi Utama	Pengambilan keputusan	Transfer kepakaran (orang-mesin-orang) dan sumbang saran
Arah query utama	Manusia menanyai (queries) mesin	Mesin menanyai (queries) manusia
Dukungan alamiah	Personal, kelompok dan institusional	Personal (utamanya) dan kelompok
Metode manipulasi	Numerik	Simbolik
Karakteristik area permasalahan	Kompleks, luas terintegrasi	Domain sempit
Jenis permasalahan	Ad hoc, unik	Repetisi
Konten database	Pengetahuan faktual	Prosedural dan pengetahuan faktual
Kemampuan reasoning	Tidak	Ya, terbatas
Kemampuan explanation	Terbatas	Ya

2.8 Proses Pengambilan Keputusan

Langkah-langkah yang harus dilakukan dalam proses pengambilan keputusan pada suatu Sistem Pakar dapat dijabarkan seperti di bawah ini.

1. **Step A**. Mengerti masalah (atau kesempatan yang ada). Sistem Pakar dapat membantu dalam mendesain alur informasi pada eksekutif (misalnya, bagaimana untuk memonitor, kapan) dan dalam penginterpretasian informasi. Disebabkan beberapa informasi bersifat fuzzy, maka kombinasi antara Sistem Pakar dan Jaringan Saraf Tiruan tentu akan membantu. Seluruh area dari proses scanning, monitoring, forecasting (misalnya, tren) dan penginterpretasian sangat dibantu oleh adanya komputerisasi. Demikian juga Natural Language Processors (NLP) akan berguna dalam menyimpulkan informasi.

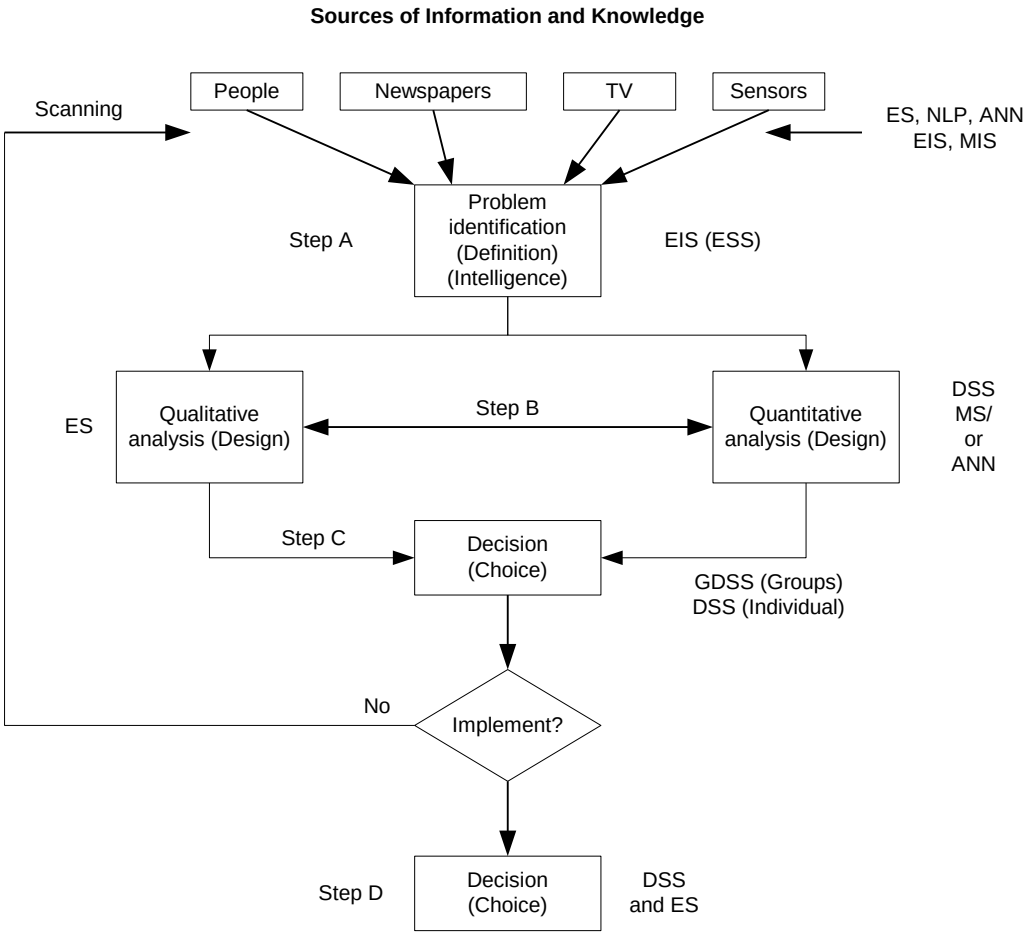
2. **Step B**. Analisis. Sekali suatu masalah (kesempatan) teridentifikasi, pertanyaan selanjutnya adalah apa yang harus dikerjakan dengan hal ini? Di sinilah langkah analisis berperanan. Analisis bisa bersifat kualitatif atau pun kuantitatif (atau kombinasinya). Analisis kuantitatif didukung oleh SPK dan oleh tool-tool analisis kuantitatif. Analisis kualitatif didukung oleh Sistem Pakar.

3. **Step C**. Memilih. Pada langkah ini, keputusan dibuat dengan memperhatikan masalahnya (atau kesempatan) berdasarkan hasil dari analisis. Langkah ini didukung

oleh SPK (jika pengambil keputusan adalah seseorang) atau oleh SPK Kelompok – Group Decision Support Systems (jika keputusan dibuat oleh sekelompok orang).

4. **Step D**. Implementasi. Pada tahap ini, keputusan untuk mengimplementasikan solusi tertentu dilakukan, dan SPK dan/atau Sistem Pakar bisa mendukung tahap ini.

Pada gambar 2.2 di bawah ini terlihat dukungan terkomputerisasi untuk proses pengambilan keputusan:



Gambar 2.2 Dukungan terkomputerisasi untuk proses pengambilan keputusan

Bab 3 Sistem Pakar

Dalam bab ini kita akan mempelajari Sistem Pakar secara lebih dalam sebagai kelanjutan dari bab sebelumnya. Sistem Pakar diturunkan dari istilah Knowledge-Based Expert System (KBES). Sistem Pakar adalah sistem yang mendapatkan dan menyimpan pengetahuan manusia ke dalam komputer untuk menyelesaikan permasalahan yang biasanya membutuhkan kepakaran seorang ahli. Pustaka yang dipakai di bab ini adalah dari E. Turban [Tur95], A.J. Gonzalez dan D.D. Dankel [Gon93] dan J.P. Ignizio [Ign91].

Area/bidang kepakaran ini disebut dengan domain.

Pengembangan Sistem Pakar terindikasi pada hal-hal di bawah ini:

- Ketersediaan pelbagai tool yang didesain untuk memudahkan pembangunan Sistem Pakar dan mengurangi biayanya.
- Penyebarluasan Sistem Pakar pada ribuan organisasi, beberapa diantaranya menggunakan ratusan atau malah ribuan sistem yang spesifik.
- Integrasi Sistem Pakar dengan pelbagai Sistem Informasi Berbasis Komputer (Computer-Based Information System) yang lain berkembang makin pesat, khususnya integrasi dengan database dan SPK.
- Penggunaan Sistem Pakar semakin meningkat pada pelbagai hal, mulai dari sistem bantuan (help) sampai ke aplikasi bidang militer dan ruang angkasa.
- Penggunaan teknologi Sistem Pakar sebagai metodologi yang mempermudah pembangunan sistem informasi reguler.
- Peningkatan penggunaan Object-Oriented Programming (OOP) dalam representasi pengetahuan.
- Pengembangan sistem utuh memiliki pelbagai sumber pengetahuan, reasoning (pertimbangan), dan informasi fuzzy.
- Penggunaan multiple Knowledge Base (KB).

3.1 Konsep Dasar Sistem Pakar

Expertise/Kepakaran

Kepakaran adalah pengetahuan yang ekstensif, spesifik yang didapatkan dari training,

membaca, dan pengalaman.

Pelbagai jenis pengetahuan di bawah ini adalah contoh dari kepakaran:

- Fakta mengenai area/daerah masalah.
- Teori mengenai area masalah.
- Aturan dan prosedur berkaitan dengan area masalah secara umum.
- Rules (heuristic) dari apa yang harus dikerjakan pada situasi masalah tertentu (contoh aturan yang berkaitan dengan penyelesaian masalah).
- Strategi global untuk menyelesaikan masalah tertentu.
- Meta-knowledge (pengetahuan mengenai pengetahuan itu sendiri).

Expert/Pakar

Aktualnya adalah derajat atau level dari kepakaran.

Umumnya, kepakaran seorang manusia terdiri dari aktivitas berikut ini:

- Mengenali dan merumuskan masalah.
- Menyelesaikan masalah secara cepat dan layak.
- Menjelaskan solusinya.
- Belajar dari pengalaman.
- Me-restrukturisasi pengetahuan.
- Breaking rules.
- Menentukan relevansi.
- Menguraikan dengan bijak (sesuai dengan keterbatasannya).

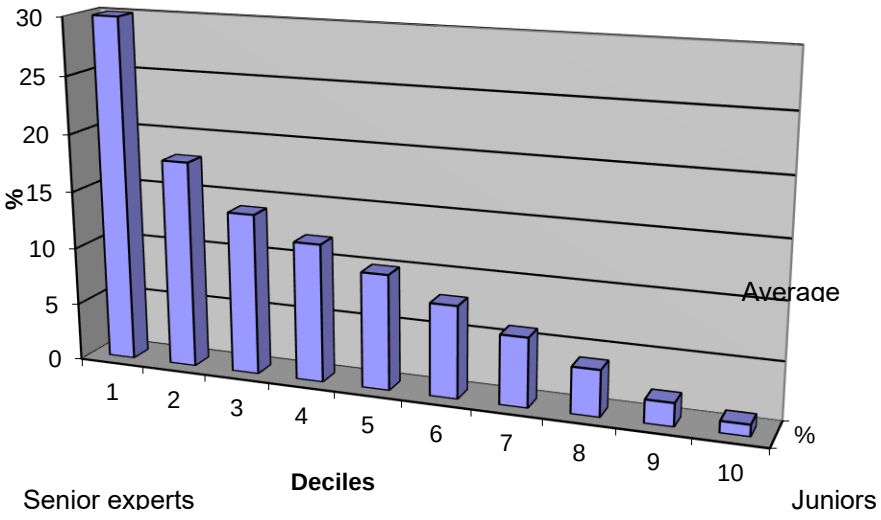
Fakta Mengenai Kepakaran

Terdapat pelbagai fakta mengenai kepakaran, diantaranya adalah sebagai berikut:

- Kepakaran biasanya diasosiasikan dengan kecerdasan tingkat tinggi tetapi ini tidak mesti berhubungan dengan orang tercerdas.
- Kepakaran biasanya diasosiasikan dengan kuantitas pengetahuan.

- Pakar belajar dari keberhasilan dan kegagalan masa lalu.
- Pengetahuan dari seorang pakar disimpan dengan baik, diorganisasi, dan dapat dicari lagi dengan cepat.
- Pakar dapat mengenali pola yang lebih tinggi dari pengalamannya (excellent recall).

Pada gambar 3.1 di bawah ini adalah diagram distribusi kepakaran:



Gambar 3.1 Diagram distribusi kepakaran

2 tipe pengetahuan yang dapat dibedakan adalah: fakta dan prosedur (biasanya rule) yang berkaitan dengan domain permasalahan.

Inferensia

Fitur khas dari Sistem Pakar adalah kemampuan untuk reasoning (mempertimbangkan). Kenyataan bahwa kepakaran disimpan dalam suatu knowledge base dan bahwa program memiliki akses ke database, maka komputer diprogram sehingga dapat berinferensia. Inferensia ini dilakukan oleh komponen yang disebut inference engine (mesin inferensia), yang di dalamnya terdapat prosedur-prosedur yang berkaitan dengan penyelesaian masalah.

Pada tabel 3.1 di bawah ini disajikan perbedaan antara Sistem Konvensional (sistem yang biasanya ada di masyarakat saat ini) dan Sistem Pakar.

Tabel 3.1 Perbedaan antara Sistem Konvensional dan Sistem Pakar

Sistem Konvensional	Sistem Pakar
Informasi dan pemrosesan biasanya dikombinasikan dalam satu program yang berurutan	Knowledge base benar-benar dipisahkan dari mekanisme pemrosesan (inferensia). Contoh: knowledge rules dipisahkan dari kontrol.
Program tidak melakukan kesalahan (tapi pemrogramannya iya)	Program mungkin melakukan kesalahan
(Biasanya) tidak menjelaskan mengapa suatu data input diperlukan atau bagaimana suatu konklusi didapatkan	Penjelasan adalah bagian dari kebanyakan Sistem Pakar
Perubahan dalam program sulit dilakukan	Perubahan dalam rule sangat mudah dilakukan
Sistem akan beroperasi hanya jika proses sudah komplit	Sistem dapat beroperasi dengan hanya sedikit rule (sebagai prototipe awal)
Eksekusi dilakukan pada basis (algoritmis) langkah-demi-langkah	Eksekusi dilakukan dengan menggunakan heuristics dan lojik
Manipulasi efektif untuk basis data yang besar	Manipulasi efektif untuk knowledge base yang besar
Representasi dan penggunaan data	Representasi dan penggunaan pengetahuan
Efisiensi adalah tujuan utama	Efektifitas adalah tujuan utama
Mudah menangani data kuantitatif	Mudah menangani data kualitatif
Menangkap, meningkatkan dan mendistribusikan akses ke data numerik atau ke informasi	Menangkap, meningkatkan dan mendistribusikan akses ke pertimbangan dan pengetahuan

Sedangkan contoh dari tindakan-tindakan heuristic itu sendiri dapat digambarkan dalam tabel 3.2 di bawah ini:

Tabel 3.2 Contoh tindakan-tindakan heuristic

Aksi	Penjelasan
Urutan job yang melalui suatu mesin	Kerjakan job yang membutuhkan waktu tersingkat lebih dulu dibandingkan job yang lain
Pembelian saham	Jangan memberli saham yang memiliki rasio price-to-earnings (harga dibanding pendapatan) yang lebih besar daripada 10
Travel (perjalanan)	Jangan pergi melewati jalan tol pada waktu antara pukul 8 dan 9 pagi
Investasi modal dalam projek teknologi tinggi	Pertimbangkan hanya projek yang memiliki perkiraan periode untungnya kurang dari dua tahun
Pembelian rumah	Beli hanya yang memiliki lingkungan tetangga yang baik, tetapi pembelian ini hanya dalam jangkauan harga yang murah

Rules

Kebanyakan Sistem Pakar komersial menggunakan sistem yang berbasis rule (Rule-Based Systems); yaitu pengetahuan disimpan dalam bentuk rule-rule, yang merupakan prosedur untuk menyelesaikan masalah.

Kemampuan Menjelaskan

Fitur unik lain dari Sistem Pakar adalah kemampuan untuk menjelaskan nasehat atau rekomendasi yang diberikan. Penjelasan dan justifikasi ini dilakukan oleh subsistem yang disebut dengan justifier atau explanation subsystem. Ini menjadikan sistem dapat memeriksa pertimbangannya dan menjelaskan operasi-operasi yang dilakukan.

3.2 Struktur Sistem Pakar

Sistem Pakar dibagi menjadi 2 bagian utama: lingkungan pengembangan (development environment) dan lingkungan konsultasi (consultation (runtime) environment). Lingkungan pengembangan digunakan oleh Pembangun Sistem Pakar (ES builder) untuk membangun komponen dan untuk membawa pengetahuan ke dalam knowledge base. Lingkungan konsultasi digunakan oleh orang yang bukan ahli untuk mendapatkan pengetahuan dan saran setara pakar.

Komponen-komponen yang ada di dalam Sistem Pakar:

- **Knowledge acquisition subsystem**. Pengetahuan dapat diperoleh dari seorang pakar, buku teks (textbooks) atau laporan penelitian, dengan dukungan dari seorang knowledge engineer (seorang pakar yang memiliki spesialisasi dalam akuisisi pengetahuan).
- **Knowledge base**. 2 jenis knowledge base adalah fakta (yaitu situasi dan teori) dan heuristics atau rule-rule.
- **Inference engine**. Ia adalah otak dari suatu Sistem Pakar, bisa juga disebut dengan struktur kontrol (control structure) atau penerjemah rule (rule interpreter dalam Rule-Based Systems). Ia adalah program komputer yang memiliki metodologi untuk melakukan reasoning (pertimbangan) mengenai informasi yang tersimpan dalam knowledge base dan dalam "blackboard (workplace)", dan ia digunakanuntuk memformulasikan konklusi. Ia memiliki 3 elemen utama: interpreter, scheduler, consistency enforcer.
- **Blackboard** (workplace). Ia adalah tempat menyimpan sementara untuk memproses

rencana (plan), agenda, solusi, dan deskripsi masalah yang didapat dari knowledge base selama sesi konsultasi.

- **User**. Umumnya user yang dimaksud ini adalah: (1) Klien (yaitu bukan pakar) yang menginginkan advis/nasehat. Disini, Sistem Pakar bertindak seperti seorang konsultan atau penasehat. (2) Learner (pelajar) untuk mempelajari bagaimana Sistem Pakar menyelesaikan permasalahan. Disini, Sistem Pakar bertindak sebagai seorang instruktur. (3) Expert system builder (pembangun sistem pakar) yang ingin meningkatkan knowledge base-nya. Disini, Sistem Pakar bertindak sebagai seorang rekan. (4) Pakar. Disini, Sistem Pakar bertindak sebagai seorang kolega atau asisten.
- **User interface**. Sistem Pakar haruslah user friendly dan berorientasi pada masalah dalam hal antarmukanya.
- **Explanation subsystem**. Ini adalah kemampuan penelusuran kebenaran dari konklusi yang didapat dari sumber-sumbernya. Hal ini krusial untuk transformasi kepakaran dan penyelesain masalah. Komponen ini mampu menelusuri kebenaran dan untuk menerangkan perilaku Sistem Pakar, secara interaktif, menjawab pertanyaan seperti: Mengapa pertanyaan tertentu ditanyakan oleh Sistem Pakar? Bagaimana konklusi tertentu dicapai? Mengapa alternatif tertentu ditolak? Rencana apakah yang ada untuk mencapai solusi? Dan apa-apa saja selanjutnya yang harus dilakukan sebelum diagnosis final dapat ditentukan?
- **Knowledge refining system**. Dengan komponen ini, pakar mampu untuk menganalisis kinerja dari Sistem Pakar, belajar daripadanya, dan meningkatkannya pada konsultasi selanjutnya.

Inference Engine

Otak dari Sistem Pakar adalah Inference Engine (mesin inferensia), disebut juga control structure atau the rule interpreter (pada Sistem Pakar berbasis rule/aturan). Komponen ini sebenarnya adalah program komputer yang menyediakan metodologi untuk reasoning (pertimbangan) mengenai informasi dalam knowledge base dan dalam "blackboard", dan digunakan untuk merumuskan kesimpulan.

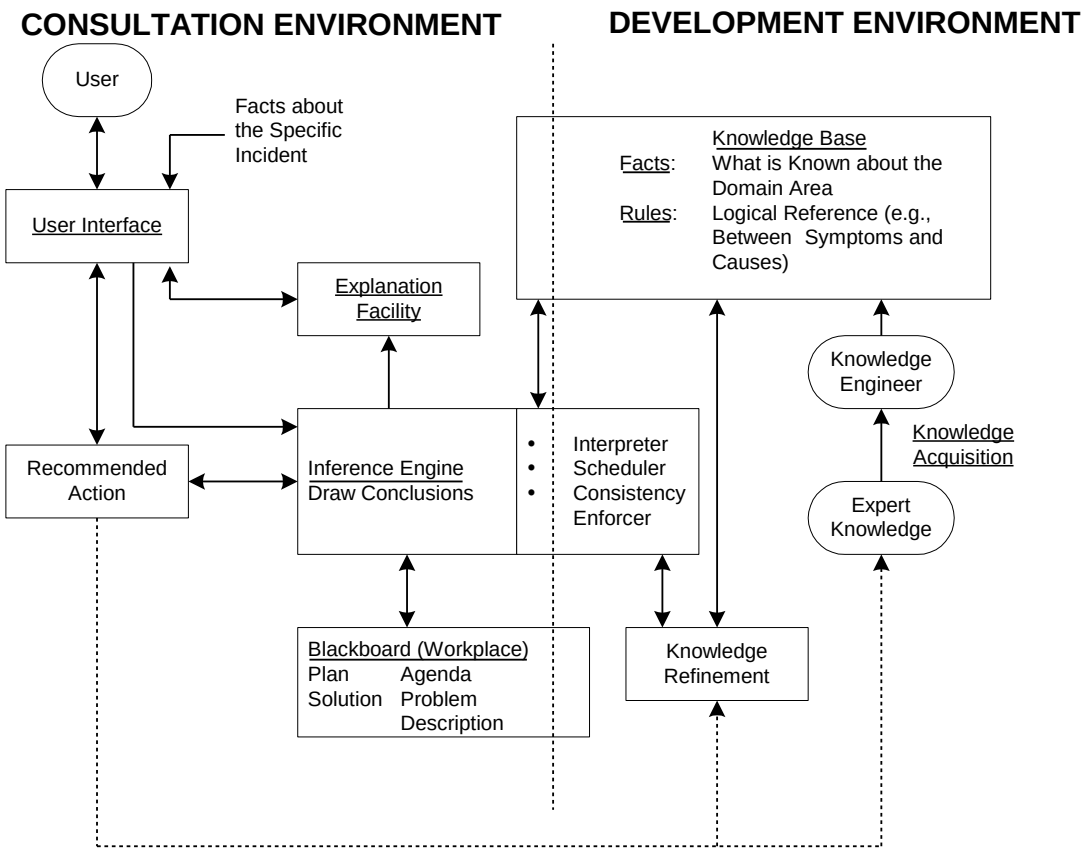
3 elemen utamanya adalah:

- **Interpreter** (rule interpreter dalam kebanyakan sistem), yang mengeksekusi item agenda yang dipilih dengan mengaplikasikannya pada knowledge base rule yang berhubungan.
- **Scheduler**, yang menjaga kontrol di sepanjang agenda. Memperkirakan akibat dari

pengaplikasian rule inferensia yang menampakkan prioritas item atau kriteria lain pada agenda.

- **Consistency enforcer**, yang mencoba menjaga konsistensi representasi solusi yang muncul.

Pada gambar 3.2 di bawah ini adalah diagram struktur dari Sistem Pakar:



Gambar 3.2 Diagram struktur Sistem Pakar

Explanation Subsystem (Justifier)

Kemampuan untuk melacak kebenaran dari kesimpulan yang didapat dari sumber-sumbernya merupakan hal yang krusial baik dalam tranformasi kepakaran maupun dalam penyelesaian masalah. Bagian ini dapat melacak kebenaran dan menjelaskan perilaku Sistem Pakar dengan secara interaktif menjawab pertanyaan seperti ini:

- Why was a certain question asked by the expert system?
- How was a certain conclution reached?
- Why was a certain alternative rejected?

- What is the plan to reach the solution? For example, what remains to be established before a final diagnosis can be determined?

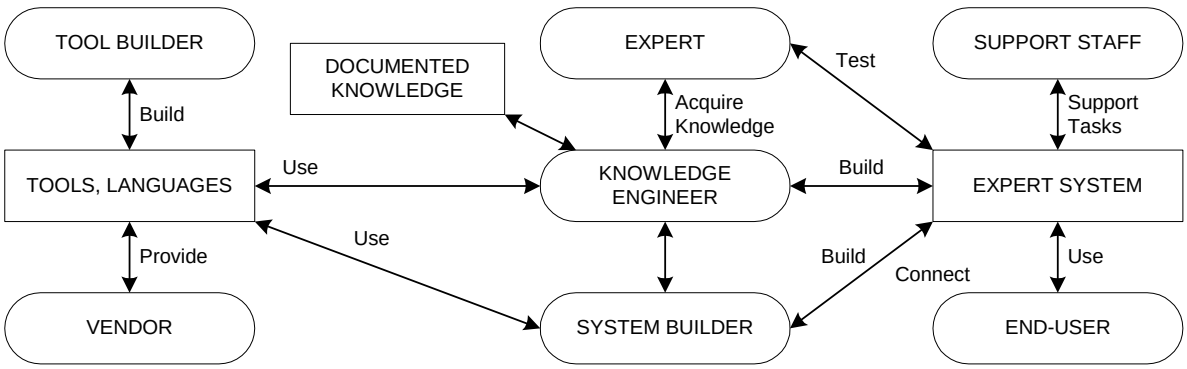
Knowledge Refining System

Dengan adanya komponen ini pakar dapat menganalisis kinerja mereka, belajar daripadanya, dan meningkatkan kemampuannya pada konsultasi berikutnya.

3.3 Elemen Manusia dalam Sistem Pakar

Orang-orang yang terlibat dalam pengembangan dan penggunaan Sistem Pakar:

1. **Pakar**.
2. **Knowledge Engineer**.
3. **User**. Yang terdiri dari:
   - Klien yang bukan pakar yang menginginkan nasehat langsung. Disini Sistem Pakar bertindak sebagai konsultan atau advisor/penasehat.
   - Pelajar yang ingin belajar. Sistem Pakar disini bertindak sebagai instruktur.
   - Pembangun Sistem Pakar yang ingin meningkatkan knowledge base-nya. Disini Sistem Pakar bertindak sebagai partner.
   - Pakar. Sistem Pakar disini bertindak sebagai kolega atau sebagai asisten.
4. **Pihak lain**. Misalnya: system builder, tool builder, vendor, staf pendukung.
   Lengkapnya dapat dilihat pada gambar 3.3 di bawah ini:



Gambar 3.3 Diagram peranan manusia dalam Sistem Pakar

3.4 Keuntungan Sistem Pakar

Pelbagai keuntungan potensial yang bisa diperoleh dari Sistem Pakar adalah:

- Meningkatkan output dan produktivitas.
- Meningkatkan kualitas.
- Mengurangi waktu kerusakan (downtime).
- Mengatasi kelangkaan kepakaran.
- Fleksibelitas.
- Pengoperasian peralatan lebih mudah.
- Menghilangkan kebutuhan akan peralatan yang mahal.
- Operasi pada lingkungan yang membahayakan.
- Akses ke pengetahuan (knowledge) dan help desk (sistem bantuan).
- Kehandalan.
- Meningkatkan kemampuan pelbagai sistem terkomputerisasi lainnya.
- Integrasi dari pelbagai opini para pakar.
- Kemampuan bekerja dengan informasi yang tidak komplit dan tak pasti.
- Penyediaan pelatihan (training).
- Peningkatan dalam hal penyelesaian masalah.
- Kemampuan menyelesaikan masalah yang kompleks.
- Transfer pengetahuan ke lokasi yang berbeda.
- Peningkatan kepada Sistem Informasi Berbasis Komputer lainnya.

3.5 Permasalahan dan Keterbatasan Sistem Pakar

Di bawah ini adalah pelbagai hal yang menghambat Sistem Pakar:

- Pengetahuan tak selalu tersedia.
- Kepakaran sulit diekstraksi dari manusia.
- Pendekatan untuk setiap pakar pada situasi tertentu selalu berbeda, dan tak mesti benar.

- Walaupun pakar tersebut memiliki ketrampilan yang tinggi, sukar untuk mengabstraksikan kepakarannya pada situasi tertentu, apalagi pakar tersebut bekerja di bawah tekanan.
    - User dari Sistem Pakar memiliki batasan kognitif alamiah.
    - Sistem Pakar bekerja baik hanya pada domain yang terbatas/sempit.
    - Kebanyakan pakar tak memiliki rasa pengertian pengecekan yang independen walaupun konklusi mereka masuk akal.
    - Kosa kata, atau jargon, yang digunakan pakar untuk mengekspresikan fakta dan relasinya biasanya jarang digunakan dan dimengerti oleh orang lain.
    - Help seringkali dibutuhkan oleh knowledge engineer yang biasanya jarang tersedia dan mahal biayanya – sebuah fakta yang dapat membuat pembangunan Sistem Pakar lebih banyak memakan biaya.
    - Kendala kepercayaan pada end-user bisa menghalangi penggunaan Sistem Pakar.
    - Transfer pengetahuan bergantung pada persepsi dan bisa bias dalam prasangka.

3.6 Jenis-jenis Sistem Pakar

Sistem Pakar muncul dalam pelbagai variasi, seperti tersebut di bawah ini:

- **Sistem Pakar vs. Knowledge-based Systems**. Sistem Pakar mendapatkan pengetahuannya dari para pakar, sedang KBS dari sumber-sumber terdokumentasi. KBS lebih murah dan lebih cepat dibangun dibandingkan Sistem Pakar.
- **Rule-Based Expert Systems**. Pengetahuan direpresentasikan sebagai serangkaian rule-rule (production rules).
- **Frame-Based Systems**. Pengetahuan direpresentasikan sebagai frame, yaitu representasi dari pendekatan Pemrograman Berbasis Objek (OOP).
- **Hybrid Systems**. Melibatkan pelbagai pendekatan representasi pengetahuan, paling tidak frame dan rule, tapi biasanya lebih dari itu.
- **Model-Based Systems**. Tersusun di sekitar model yang mensimulasikan struktur dan fungsi dari sistem yang dipelajari. Model digunakan untuk menghitung nilai-nilai, yang dibandingkan dengan sedang diamati. Pembandingan tersebut memicu aksi (jika diperlukan) atau diagnosis lebih lanjut.
- **Sistem yang diklasifikasikan oleh sifat alamiahnya**. Ada 3 jenis. (1)

berhubungan dengan evidence gathering (pengumpulan bukti-bukti), (2) stepwise refinement system. Sistem ini berhubungan dengan sejumlah besar keluaran dari level-level detil sesudahnya. (3) stepwise assembly, dimana domain subjek dapat mempunyai jumlah yang luar biasa besar keluaran yang mungkin. Jenis khusus dari ini disebut dengan catalog selection. Sistem ini berhubungan dengan masalah seperti pemilihan bahan kimia, baja yang benar.

- Sistem siap pakai (**Off-the-Shelf Systems**). Sebagai hasil dari produksi masal membuatnya lebih murah dibandingkan dengan sistem yang memenuhi keinginan user (customized system). Sayangnya sistem ini bersifat terlalu umum, dan nasehat/advis yang dihasilkan mungkin tak bernilai pada user yang dihadapkan pada situasi yang kompleks.
- **Real-Time Expert Systems**. Sistem ini berkenaan dengan waktu, jadi harus cukup cepat mengontrol proses terkomputerisasi. Sistem selalu menghasilkan respon sesuai waktu yang diperlukan.

Bab 4 Membangun Sistem Pakar

Bab ini menjelaskan mengenai cara pembangunan dari sistem pakar, dimana pustakanya diambilkan dari E. Turban [Tur95], A.J. Gonzalez dan D.D. Dankel [Gon93], J.P. Ignizio [Ign91] dan G.R. Baur and D.V.Pigford [Baur90].

3 aktivitas yang harus dilakukan dalam rangka membangun suatu Sistem Pakar dapat dituliskan sebagai berikut:

- Pengembangan (development)
- Konsultasi (consultation)
- Peningkatan (improvement)

4.1 Bidang Permasalahan yang Ditangani Sistem Pakar

Sebelum kita membangun suatu Sistem Pakar, tentu ada baiknya kita mengetahui lebih dulu bidang-bidang apa saja yang dapat ditangani oleh sebuah Sistem Pakar.

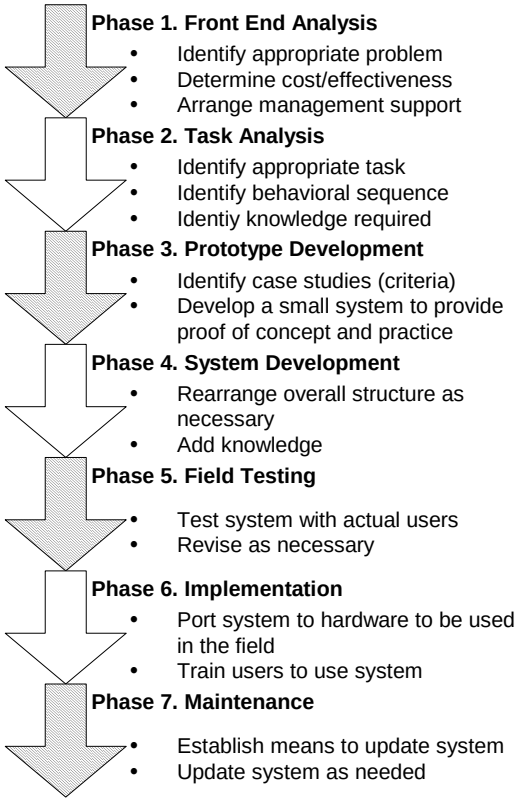
Bidang-bidang ini dapat dilihat pada tabel 4.1 di bawah ini.

Tabel 4.1 Bidang permasalahan yang ditangani Sistem Pakar

Kategori	**Bidang permasalahan yang ditangani**
Interpretasi	Mengambil kesimpulan deskripsi situasi dari observasi
Prediksi	Mengambil kesimpulan seperti halnya konsekuensi dari situasi tertentu
Diagnosis	Mengambil kesimpulan kegagalan sistem dari observasi
Desain	Mengatur objek-objek dalam batasan-batasan tertentu
Perencanaan	Mengembangkan rencana-rencana untuk mencapai tujuan (tujuan)
Monitoring	Membandingkan observasi ke rencana, menandai perkecualian-perkecualian
Debugging	Meresepkan obat (penyelesaian) untuk kasus kegagalan
Perbaikan	Mengeksekusi suatu rencana untuk menatausaha obat (penyelesaian) yang diresepkan
Instruksi	Pendiagnosisan, debugging dan pengkoreksian kinerja murid-murid
Kontrol	Interpretasi, prediksi, perbaikan dan monitoring perilaku sistem

4.2 Tahapan Membangun Sistem Pakar

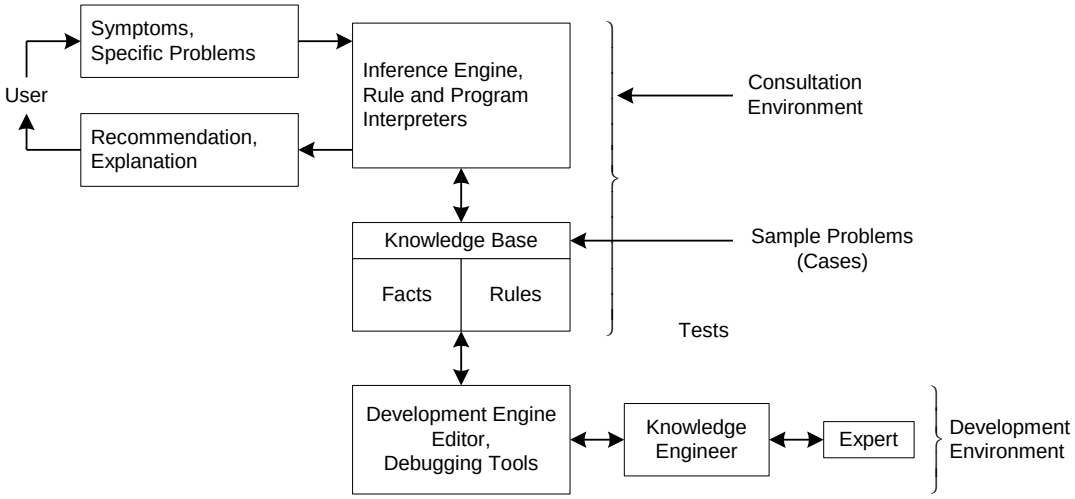
7 langkah yang diperlukan dalam pengembangan suatu Sistem Pakar dapat dilihat pada diagram pada gambar 4.1 di bawah ini:



Gambar 4.1 7 tahap pembangunan Sistem Pakar

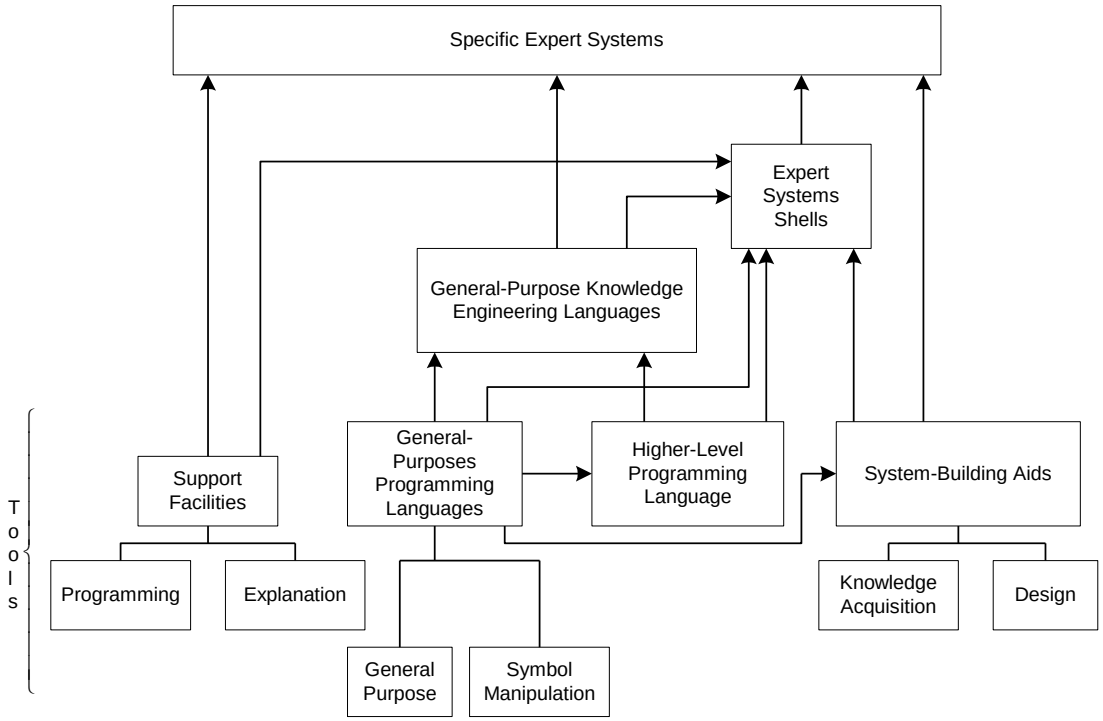
4.3 Membangun Shell Sistem Pakar yang Spesifik

Sedangkan untuk membangun Sistem Pakar yang spesifik dengan suatu shell dapat dilihat pada gambar 4.2 di bawah ini:



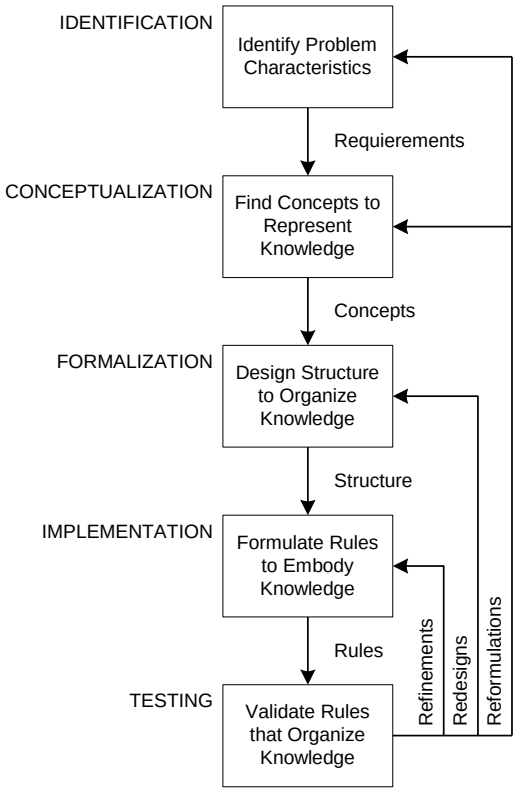
Gambar 4.2 Membangun shell Sistem Pakar yang spesifik

Perangkat lunak untuk membangun Sistem Pakar yang spesifik dapat digambarkan pada gambar 4.3 berikut ini:



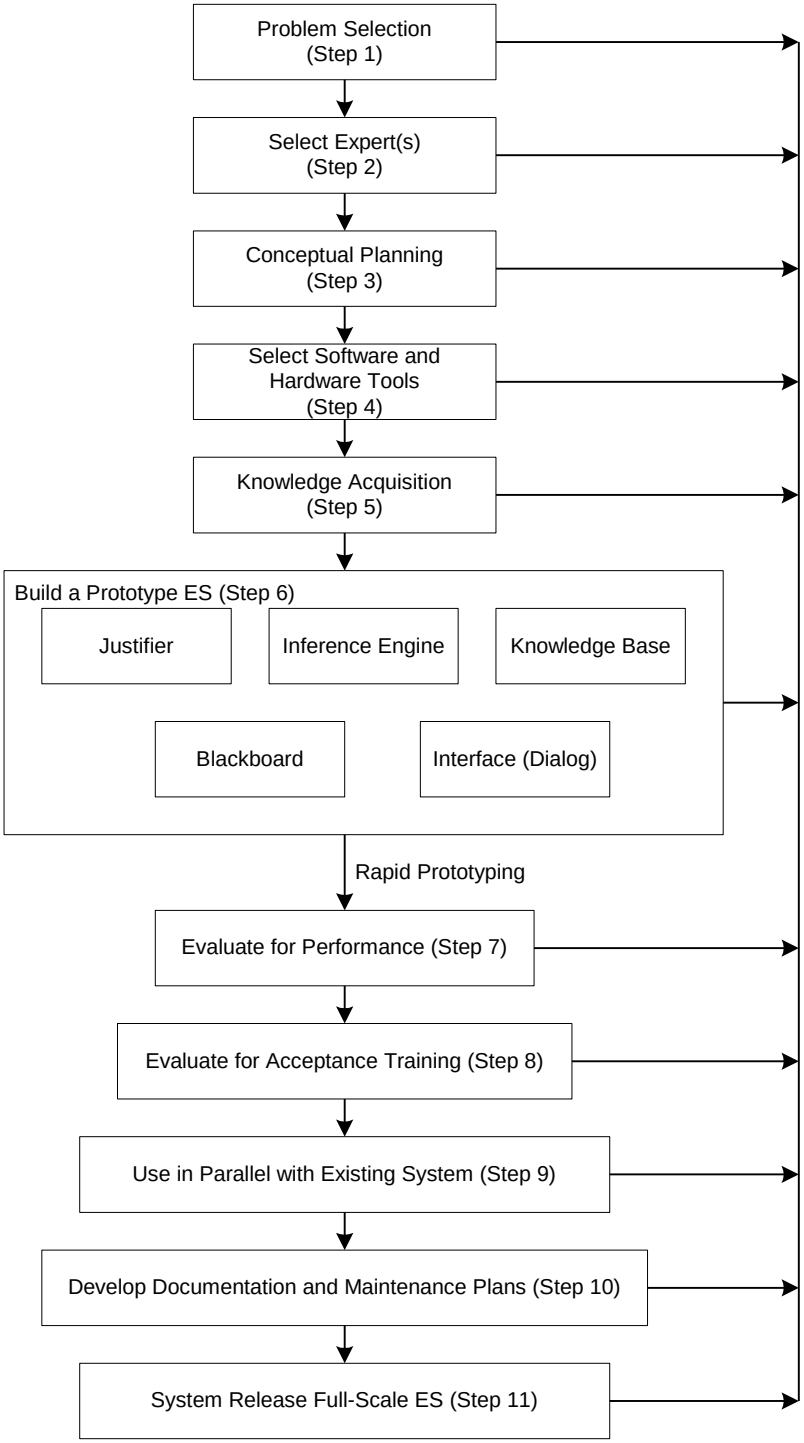
Gambar 4.3 Perangkat lunak pembangun Sistem Pakar yang spesifik

Tahapan-tahapan dalam mengakuisisi pengetahuan digambarkan pada gambar 4.4 seperti terlihat di bawah ini:



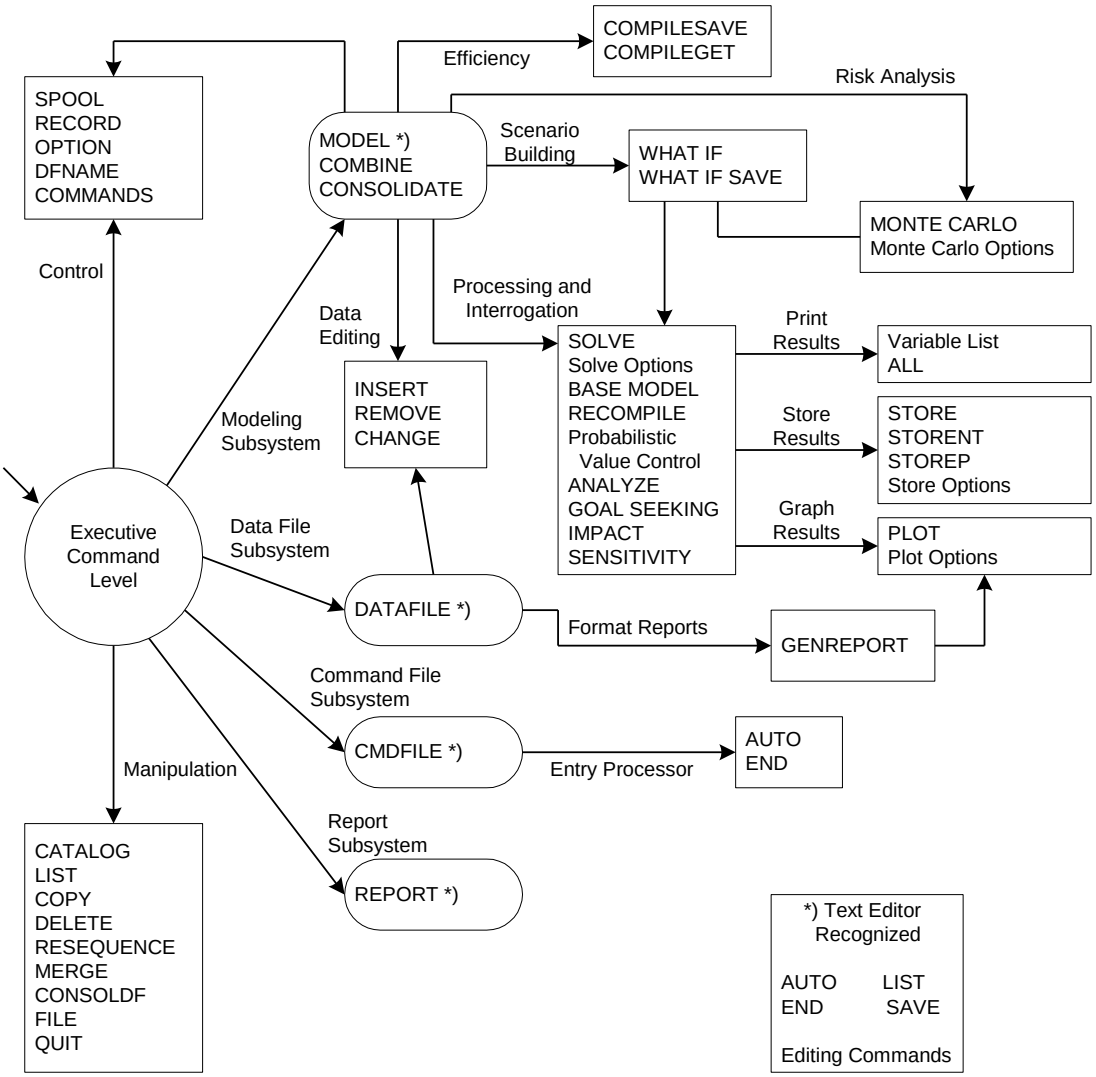
Gambar 4.4 Tahapan-tahapan akuisisi pengetahuan

Sedangkan proses pengembangan Sistem Pakar itu sendiri digambarkan pada gambar 4.5 sebagai berikut:



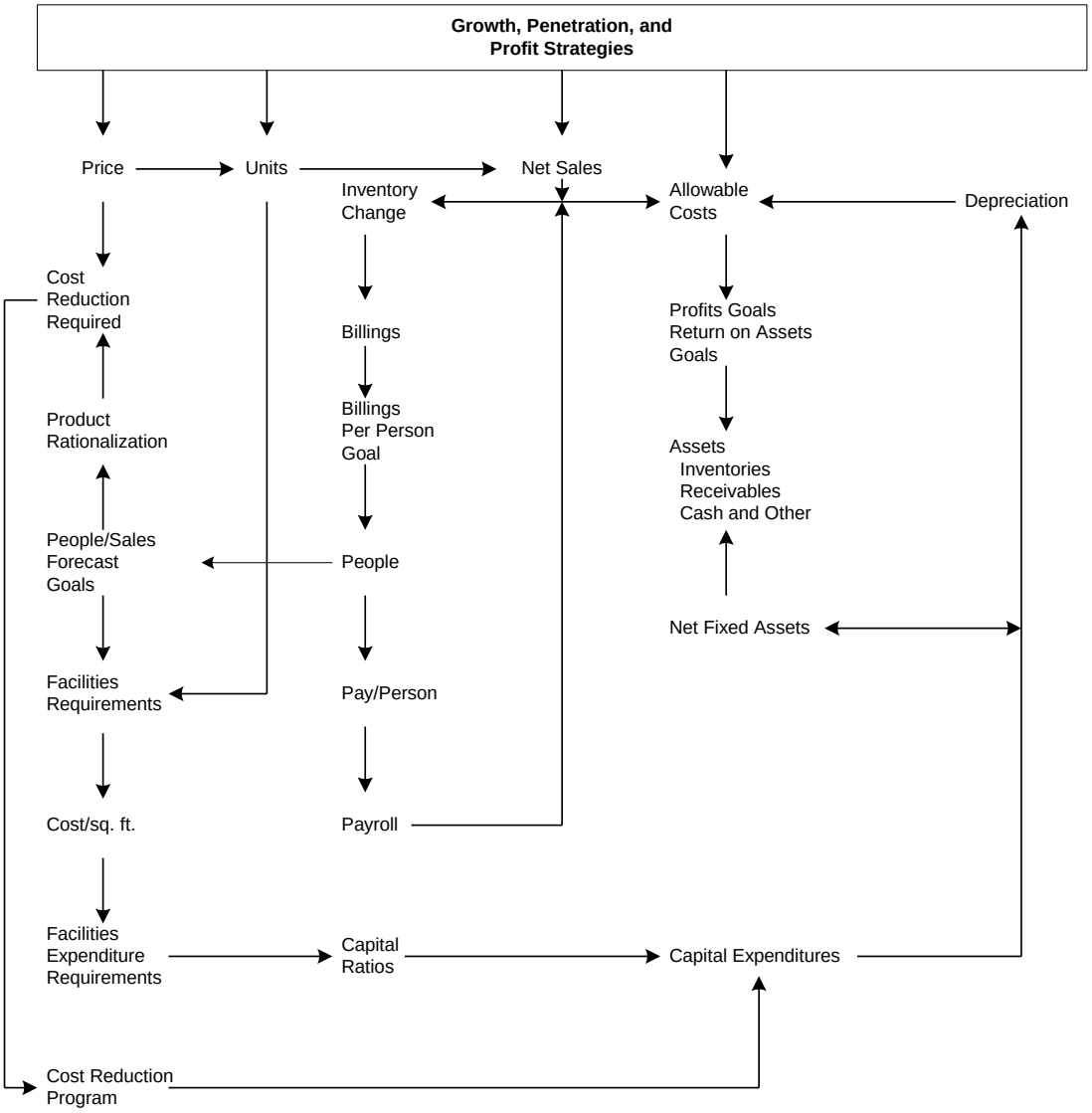
Gambar 4.5 Proses pengembangan Sistem Pakar

Salah satu contoh dari Sistem Pakar, yaitu IFPS – Interactive Financial Planning System, dapat digambarkan dalam diagram pada gambar 4.6 di bawah ini:



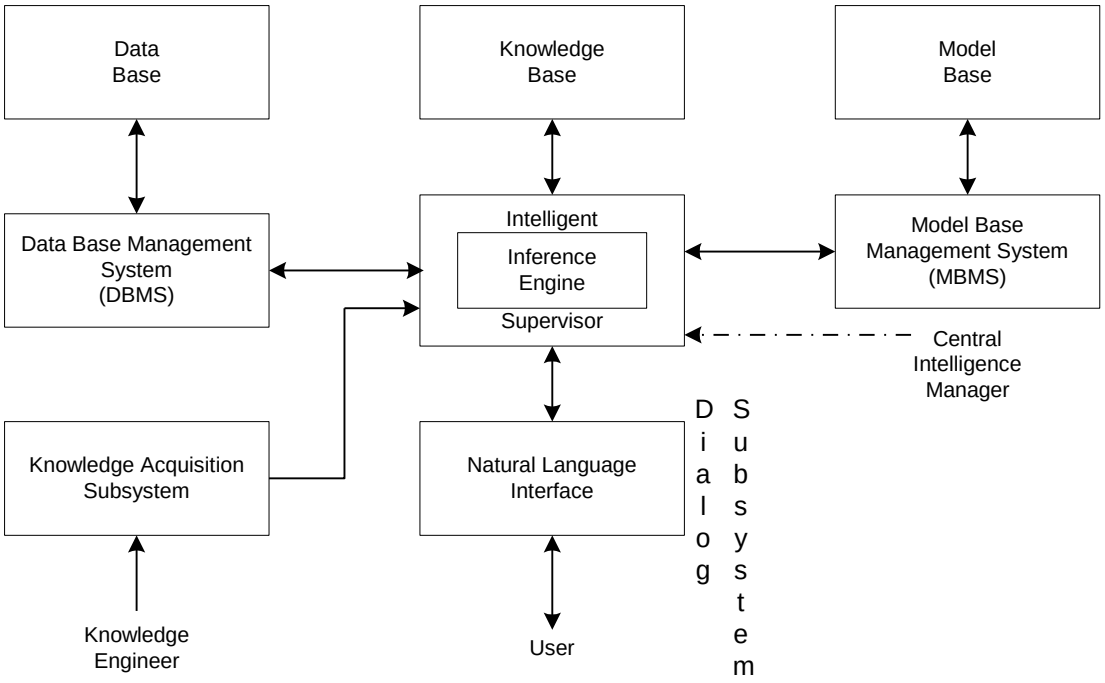
Gambar 4.6 Diagram dari Interactive Financial Planning System

Sedangkan pada gambar 4.7 di bawah ini disajikan diagram Financial Decision Support Model.



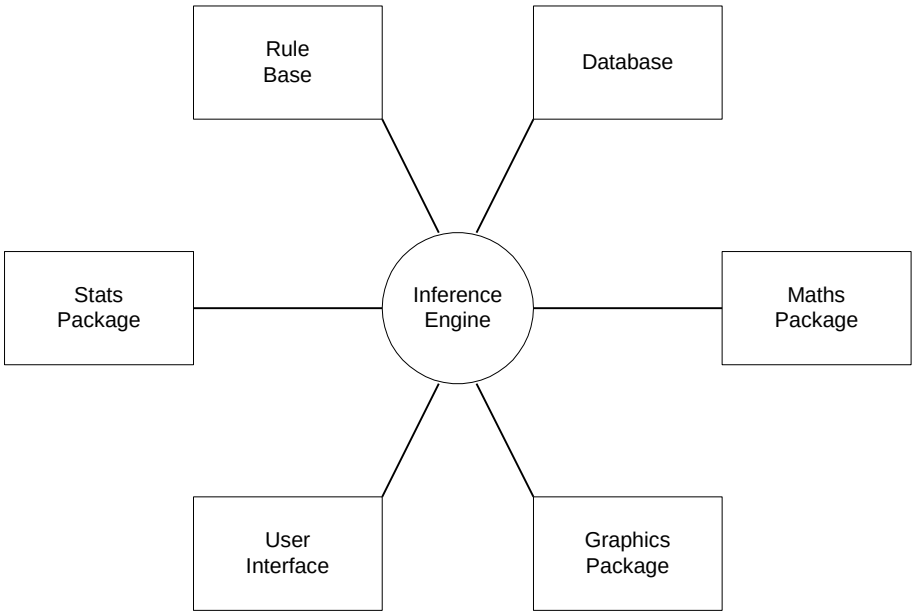
Gambar 4.7 Diagram dari Financial Decision Support Model

Pada gambar 4.8 di bawah ini disajikan diagram arsitektur terpadu untuk Intelligent Decision Support System:



Gambar 4.8 Diagram arsitektur terpadu untuk Intelligent Decision Support System

Integrasi antara Sistem Pakar dengan perangkat lunak lain dapat digambarkan dalam gambar 4.9 seperti di bawah ini:



Gambar 4.9 Integrasi Sistem Pakar dengan perangkat lunak lain

4.4 Ringkasan Pembangunan Sistem Pakar

- Sistem Pakar menirukan proses reasoning (pertimbangan) dari pakar untuk menyelesaikan masalah yang sulit
- Pendahulu Sistem Pakar adalah GPS (General-purpose Problem Solver). GPS dan yang serupa dengannya gagal disebabkan mereka mencoba menangani terlalu banyak dan mengabaikan pentingnya pengetahuan khusus yang dibutuhkan.
- Kekuatan Sistem Pakar diturunkan dari pengetahuan khusus yang dimiliki, dan bukan dari representasi pengetahuan tertentu dan skema inferensia yang dijalankan.
- Kepakaran adalah pengetahuan task khusus yang didapatkan dari pelatihan/training, membaca, dan pengalaman.
- Pakar dapat membuat keputusan yang cepat dan baik berkenaan dengan situasi yang kompleks
- Kebanyakan pengetahuan dalam organisasi dimiliki oleh segelintir pakar.
- Teknologi Sistem Pakar mencoba untuk mentransfer pengetahuan dari pakar dan sumber-sumber terdokumentasi ke komputer dan bisa digunakan oleh yang bukan pakar
- Kemampuan reasoning (pertimbangan) dalam Sistem Pakar disediakan oleh mesin inferensia (inference engine)
- Pengetahuan dalam Sistem Pakar dipisahkan dari inferensia (pemrosesannya)
- Sistem Pakar menyediakan kemampuan menjelaskan (explanation) yang terbatas
- Terdapat perbedaan diantara lingkungan pengembangan (membangun Sistem Pakar) dan lingkungan konsultasi (menggunakan Sistem Pakar)
- Komponen utama dari Sistem Pakar adalah subsistem pengakuisisian pengetahuan, knowledge base, inference engine, blackboard, user interface, dan explanation subsystem.
- Knowledge engineer menangkap pengetahuan dari pakar dan memprogramnya ke dalam komputer.
- Walaupun user utama dari Sistem Pakar adalah yang bukan pakar, user yang lain (seperti pelajar, pembuat Sistem Pakar, dan mungkin para pakar juga) juga menggunakan Sistem Pakar.
- Pengetahuan dapat berupa deklarasi (fakta) atau prosedur

- Sistem Pakar dapat ditingkatkan dalam langkah-langkah iterasi menggunakan proses yang disebut dengan rapid prototyping (prototipe cepat)
- 10 kategori umum Sistem Pakar adalah: interpretasi, prediksi, diagnosis, desain, perencanaan, pemantauan, debugging, perbaikan, instruksi, dan kontrol.
- Sistem Pakar dapat memberikan banyak keuntungan. Yang terpenting adalah peningkatan dalam produktivitas dan/atau kualitas, penanganan kepakaran yang jarang didapatkan, peningkatan sistem yang lain, penanganan informasi yang tak lengkap, dan penyediaan training/pelatihan.
- Walaupun ada pelbagai keterbatasan penggunaan Sistem Pakar, dengan adanya perkembangan teknologi semakin lama keterbatasan tersebut akan makin hilang.
- Sistem Pakar, seperti halnya pakar, dapat membuat kesalahan.
- Terdapat pelbagai pembedaan diantara Sistem-sistem Pakar, dimana kebanyakan pengetahuan datang dari para pakar; serta knowledge systems, dimana mayoritas pengetahuan datang dari sumber-sumber terdokumentasi.
- Pelbagai Sistem Pakar tersedia sebagai sistem yang siap pakai; ia mengolah dan memberikan advis/nasehat umum untuk situasi standar.
- Sistem Pakar dapat juga bekerja dalam mode real-time (waktu nyata).

Bab 5 Sistem Pakar Berbasis Pengetahuan

Dalam bab ini akan dibahas sistem pakar yang mencoba membantu manusia dalam pengambilan keputusan dimana dalam bekerjanya sistem ini mendasarkan dirinya pada pengetahuan (knowledge) yang diakuisisi dari pakar. Pustaka yang digunakan adalah J.P. Ignizio [Ign91], G.R. Baur and D.V. Pigford [Baur90], P.H. Winston [Wins92], serta hasil riset penulis sendiri dan mahasiswa bimbingan penulis [Sub03].

5.1 Pengertian

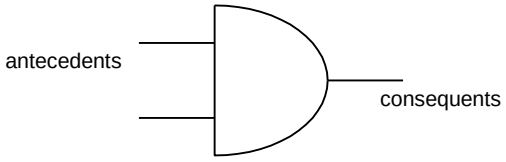
Sistem Pakar Berbasis Pengetahuan (Knowledge-Based Expert Systems) pengertiannya serupa dengan Sistem Pakar, yaitu program pemberi advis/nasehat yang terkomputerisasi yang ditujukan untuk menirukan atau menggantikan proses reasoning (pertimbangan) dan pengetahuan (knowledge) dari para pakar dalam menyelesaikan permasalahan masalah yang spesifik.

Karakteristiknya:

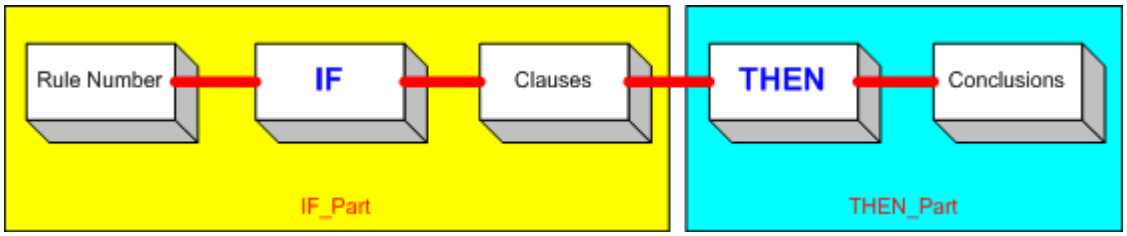
- Dapat belajar dari pengalaman
- Mentransfer pengetahuan dari satu domain ke domain yang lain
- Dapat memberikan proses reasoning (pertimbangan) dalam pelbagai level
- Menggunakan tool-tool: heuristics (rule of thumb), mathematical models, simulations.

5.2 Rule dan Rangkaian Rule

Pada gambar 5.1 di bawah ini terlihat gambaran satu rule yang terdiri dari 2 klausa (clauses = antecedents = IF_part), yang dihubungkan dengan operator AND, dan 1 konklusi (conclusions = consequents = THEN_part).



IF (antecedents) THEN consequents



Gambar 5.1 Diagram dari satu buah rule

Contoh Kasus:

Identifikasi Binatang

Sebuah robot dapat mempersepsikan fitur-fitur dasar: color (warna), size (ukuran), has hair (berambut) dan gives milk (menyusui).

Tetapi, kemampuannya mengidentifikasi objek berdasarkan fitur-fitur tersebut diatas adalah terbatas. Dia dapat membedakan binatang-binatang dari objek yang lain, tetapi ia tak dapat menggunakan fakta bahwa sebagian binatang yang memiliki leher panjang disebut dengan jerapah, sedang yang lain disebut dengan unta atau pun burung unta.

Rules (aturan-aturannya) diambilkan dari Zookeeper Rule Base oleh Winston [Wins92] yang direpresentasikan kembali seperti gambar 5.2 berikut ini.

Z1 IF ?x has hair

THEN ?x is a mammal

Z2 IF ?x gives milk

THEN ?x is a mammal

Z3 IF ?x has feathers

THEN ?x is a bird

Z4 IF ?x flies

?x lays eggs

THEN ?x is a bird

Z5 IF ?x is a mammal

?x eats meat

THEN ?x is a carnivore

Z6 IF ?x is a mammal

?x has pointed teeth

?x has claws

?x has forward-pointing eyes

THEN ?x is a carnivore

Z7 IF ?x is a mammal

?x has hoofs

THEN ?x is an ungulate

Z8 IF ?x is a mammal

?x chews end

THEN ?x is an ungulate

Z9 IF ?x is a carnivore

?x has tawny color

?x has dark spots

THEN ?x is a cheetah

Z10 IF ?x is a carnivore

?x has tawny color

?x has black strips

THEN ?x is a tiger

Z11 IF ?x is a ungulate

?x has long neck

?x has long legs

?x has tawny color

?x has dark spots

THEN ?x is a giraffe

Z12 IF ?x is a ungulate

?x has white color

?x has black strips

THEN ?x is a zebra

Z13 IF ?x is a bird

?x does not fly

?x has long legs

?x has long neck

?x has black and white

THEN ?x is a ostrich

Z14 IF ?x is a bird

?x does not fly

?x swims

?x has black and white

THEN ?x is a penguin

Z15 IF ?x is a bird

?x is a good flyer

THEN ?x is an albatross

Gambar 5.2 Zookeeper Rule Base

Dalam inferencing (inferensia - penarikan kesimpulan dan penjelasannya), sistem deduksi (pencarian hal-hal yang khusus dari hal-hal yang umum) terdiri atas 2 sistem, yaitu:

- Forward chaining
- Backward chaining

Di bawah ini akan dijelaskan masing-masing dari sistem penarikan kesimpulan tersebut beserta dengan penjelasannya.

5.3 Pemilihan Forward dan Backward Chaining

Dalam sistem inferensia yang akan dibuat/dikembangkan tentu ada pertanyaan, manakah sistem yang harus dipilih, forward ataukah backward chaining?

Di bawah ini terdapat panduan untuk memilih sistem yang mana yang lebih cocok diantara keduanya untuk sistem yang akan kita kembangkan:

- Bagaimanakah hubungan antara rule dengan fakta-faktanya, sehingga didapatkan konklusinya.
- Jika masalah yang dihadapi lebih dekat ke fan out (sekumpulan fakta yang bisa menuju ke banyak konklusi), maka pilihlah backward chaining.
- Jika masalah yang dihadapi lebih dekat ke fan in (sekumpulan hipotesis yang bisa menuju ke banyak pertanyaan), maka pilihlah forward chaining.
- Banyak cara untuk mendapatkan sedikit konklusi → forward chaining
- Sedikit cara untuk mendapatkan banyak konklusi → backward chaining
- Jika kita belum mendapatkan pelbagai fakta, dan kita tertarik hanya pada satu konklusi yang mungkin, maka digunakanlah backward chaining.
- Jika kita benar-benar sudah mendapatkan pelbagai fakta, dan kita ingin untuk mendapatkan konklusi dari fakta-fakta itu, maka digunakanlah forward chaining.

Tipe sistem yang dapat dicari dengan forward chaining:

1. Sistem yang dipresentasikan dengan satu atau beberapa kondisi
2. Untuk setiap kondisi, sistem mencari rule-rule dalam knowledge base untuk rule-rule yang berkorespondensi dengan kondisi dalam bagian IF.

3. Setiap rule dapat menghasilkan kondisi baru dari konklusi yang diminta pada bagian THEN. Kondisi baru ini ditambahkan ke kondisi lain yang sudah ada.

4. Setiap kondisi yang ditambahkan ke sistem akan diproses. Jika ditemui suatu kondisi, sistem akan kembali ke langkah 2 dan mencari rule-rule dalam knowledge base kembali. Jika tidak ada konklusi baru, sesi ini berakhir.

Tipe sistem yang dapat dicari dengan backward chaining:

1. Sistem yang dipresentasikan dengan satu atau beberapa kondisi

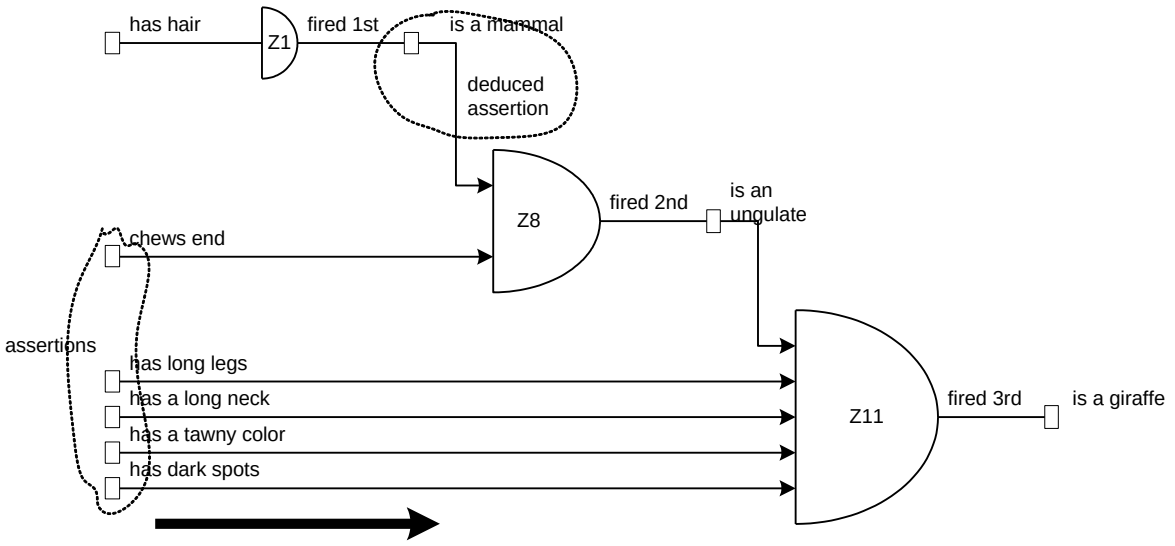
2. Untuk setiap konklusi, sistem mencari rule-rule dalam knowledge base untuk rule-rule yang berkorespondensi dengan konklusi pada bagian THEN.

3. Setiap konklusi dihasilkan dari kondisi-kondisi yang terdapat pada bagian IF. Selanjutnya kondisi-kondisi tersebut menjadi konklusi baru yang dimasukkan ke stack di atas konklusi yang sudah ada.

4. Setiap konklusi yang ditambahkan ke sistem akan diproses. Jika ditemui suatu konklusi, sistem akan kembali ke langkah 2 dan mencari rule-rule dalam knowledge base kembali. Jika tidak ada konklusi baru, sesi ini berhenti.

5.4 Forward Chaining

Pada gambar 5.3 di bawah ini terdapat contoh proses inferensia dari Zookeeper rule base dengan menggunakan metode forward chaining, dimana konklusi yang didapat adalah mesin inferensia kita mengenali objek yang diberikan sebagai seekor jerapah (giraffe).



Gambar 5.3 Diagram contoh proses inferensia menggunakan forward chaining

Sedangkan algoritma dari forward chaining itu sendiri dapat dilihat pada gambar 5.4 di bawah ini:

→ Identifikasi seekor binatang

→ Sampai tidak ada rule yang menghasilkan pernyataan baru atau seekor binatang telah teridentifikasi

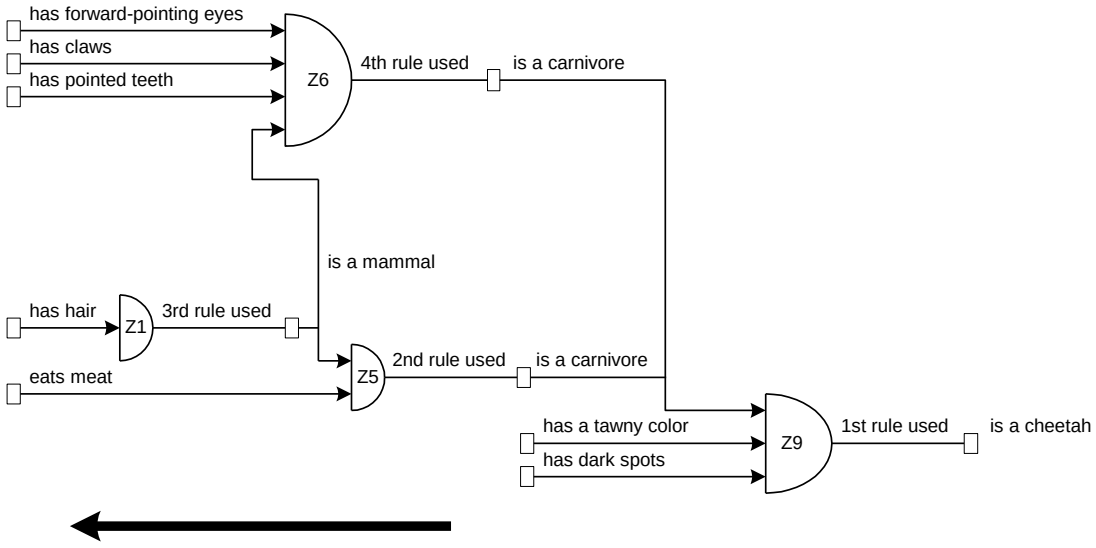
→ Untuk setiap rule:

- Cobalah untuk mendukung setiap antecedent rule dengan mencocokkannya dengan fakta-fakta yang diketahui
- Jika semua antecedent rule tersebut memang didukung, nyatakan pada consequent rule bahwa memang terdapat suatu pernyataan yang identik.
- Laporkan semua alternatif kecocokan dan instantiation (pemberian nilai pada antecedent)

Gambar 5.4 Algoritma forward chaining

5.5 Backward Chaining

Pada gambar 5.5 di bawah ini terdapat contoh proses inferensia dengan menggunakan metode backward chaining dari Zookeeper rule base, dimana konklusi yang ingin diujicoba mesin inferensia kita adalah seekor cheetah.



Gambar 5.5 Diagram contoh proses inferensia menggunakan backward chaining

Algoritma dari backward chaining itu sendiri dapat dilihat pada gambar 5.6 seperti tersaji di bawah ini:

→ Ulang terus sampai semua hipotesis telah dicoba dan tak satu pun didukung atau sampai seekor binatang telah teridentifikasi

→ Untuk setiap hipotesis,

→ Untuk setiap rule yang mempunyai consequent yang cocok dengan hipotesis yang ada saat itu,

- Cobalah untuk mendukung setiap antecedent rule dengan mencocokkannya dengan pengisian fakta di antecedent (assertion)

- Jika semua antecedent rule telah didukung, umumkan bahwa sukses yang didapat dan simpulkan bahwa hipotesis tadi adalah benar

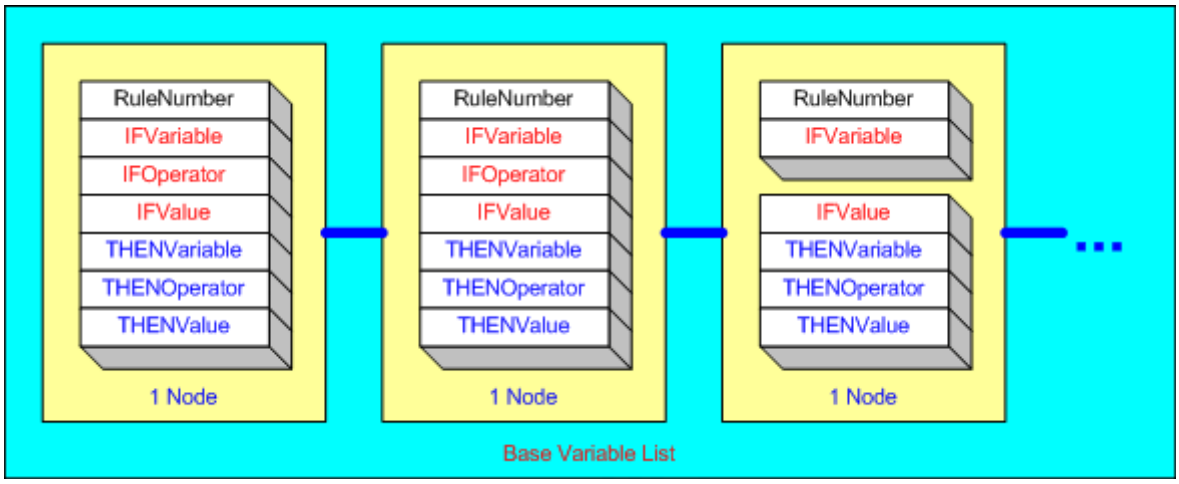
Catatan: penelusuran rangkaian (chaining) tidak sukses jika pengisian fakta yang diperlukan oleh antecedent ternyata tak mendukung suatu antecedent.

Gambar 5.6 Algoritma backward chaining

5.6 Desain Implementasi Stuktur

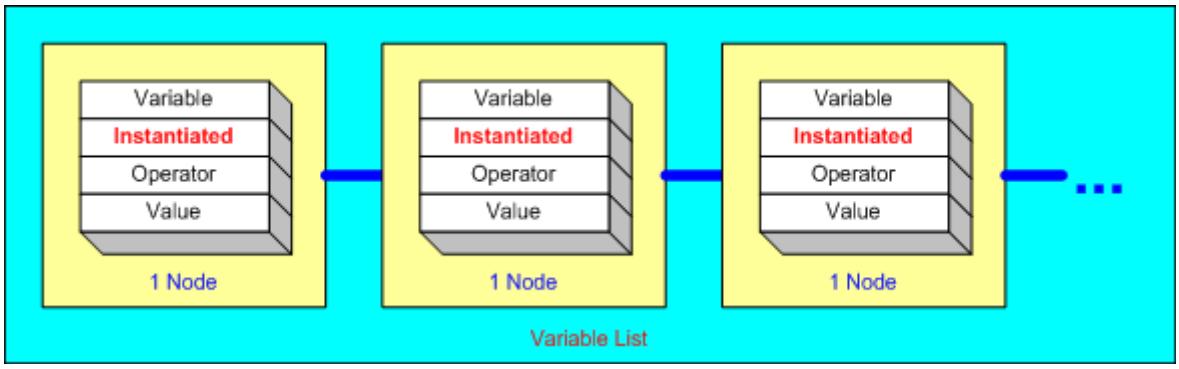
Struktur yang digunakan dalam mesin inferensia dapat dilihat seperti gambar 5.7-5.11 di bawah ini.

Base Variable List. Struktur ini mencatat semua variabel, operator dan value yang ada dalam rule base.



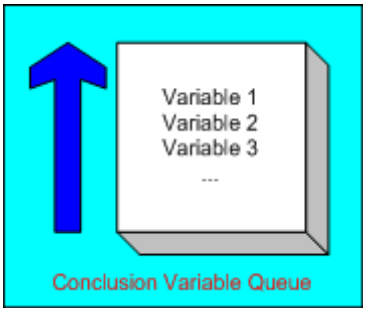
Gambar 5.7 Base Variable List

Variable List. Struktur ini hanya mencatat variabel, operator dan value yang ada dalam bagian IF (IF_PART) rule-rule yang ada dalam rule base.



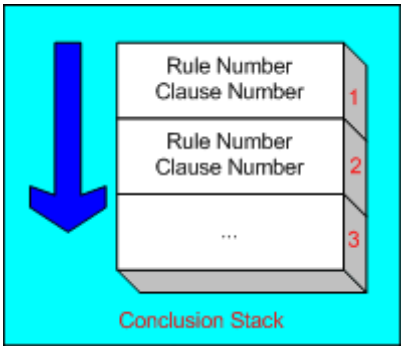
Gambar 5.8 Variable List

Conclusion Variable Queue. Struktur ini mencatat variabel yang sedang diproses saat itu dalam proses forward chaining.



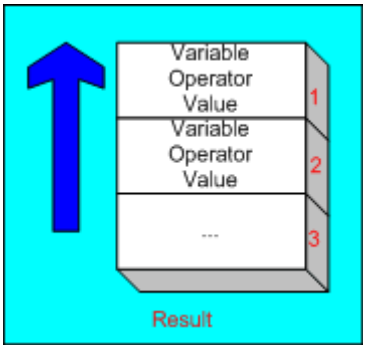
Gambar 5.9 Conclusion Variable Queue

Conclusion Stack. Struktur ini mencatat nomor rule dan nomor clause yang sedang diproses saat itu dalam proses backward chaining.



Gambar 5.10 Conclusion Stack

Result. Struktur ini mencatat semua konklusi (yang terdiri dari variable-operator-value) dari rule-rule yang berhasil dieksekusi (fired) baik dalam proses forward maupun backward chaining.



Gambar 5.11 Result

5.7 Desain Implementasi Forward Chaining

Algoritma pengimplementasian sistem inferensia yang menggunakan forward chaining dapat dilihat pada gambar 5.12 di bawah ini.

1. Identifikasi kondisi
2. Variabel kondisi ditempatkan pada Conclusion Var. Queue dan nilainya dicatat pada Variable List.
3. Diadakan pencarian pada Clause Var. List untuk variabel yang namanya sama dengan nama pada awal queue. Jika ketemu, nomor rule dan 1 diisikan pada Clause Var. Pointer. Jika tak ketemu, ke langkah 6.
4. Setiap variabel dalam IF clause dari rule yang belum diisi, selanjutnya diisi. Variabel-variabel ditempatkan dalam Clause Var. List. Jika semua clause benar kondisinya, bagian THEN dijalankan.
5. Pengisian bagian THEN pada variabelnya ditempatkan pada bagian belakang di Conclusion Var. Queue.
6. Jika tak ada lagi statemen IF yang mengandung variabel yang berada di awal Conclusion Var. Queue, maka variabel tersebut dihapus.
7. Jika tak ada lagi variabel pada Conclusion Var. Queue, pencarian berakhir; jika masih ada variabel yang lain, kembali ke langkah 3.

Gambar 5.12 Algoritma implementasi forward chaining

5.8 Desain Implementasi Backward Chaining

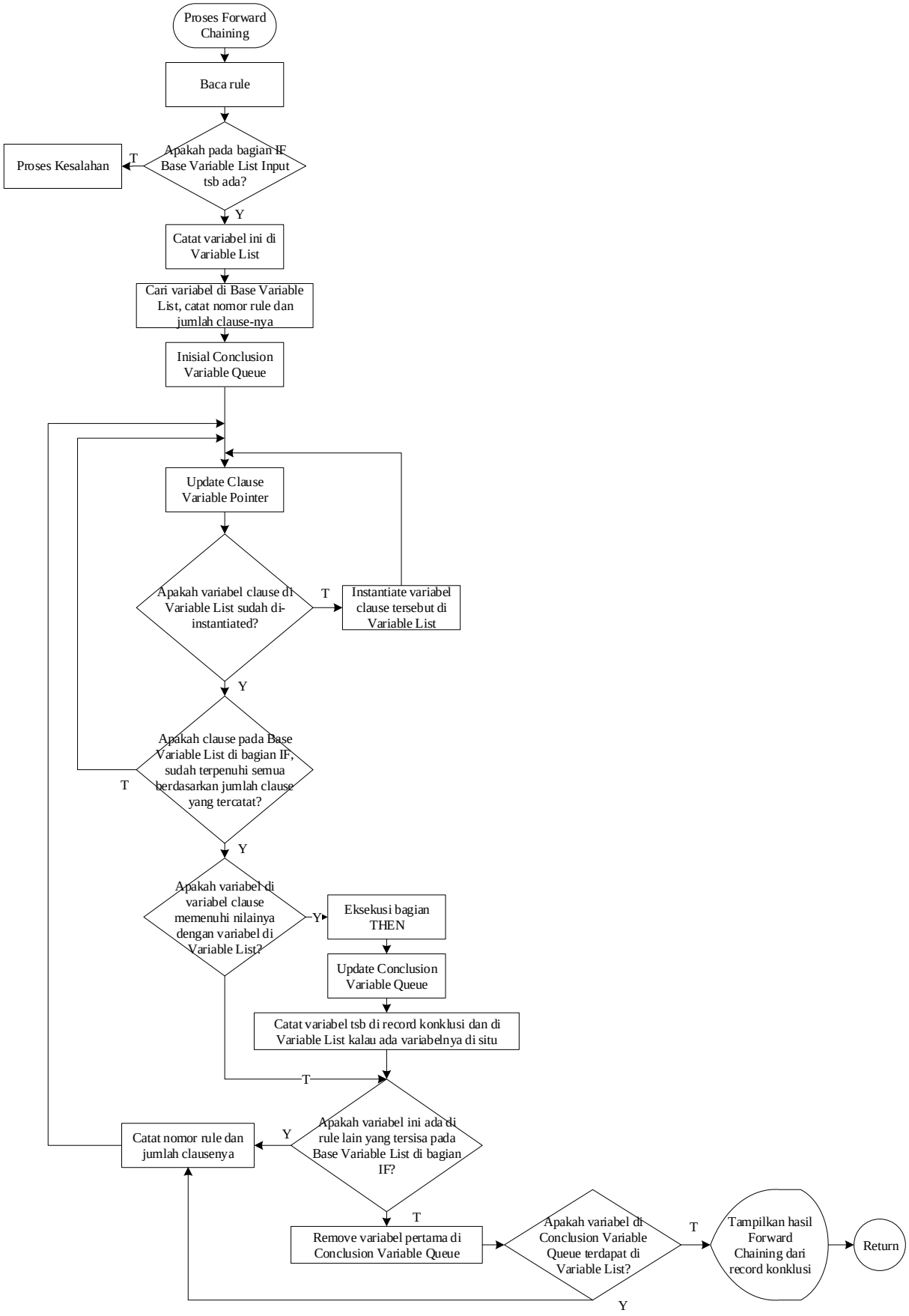
Algoritma pengimplementasian sistem inferensia yang menggunakan backward chaining dapat dilihat pada gambar 5.13 di bawah ini.

1. Identifikasi konklusi
2. Cari pada Conclusion List untuk pengisian pertama kali dari nama konklusi. Jika ketemu, tempatkan rule pada Conclusion Stack berdasarkan nomor rule dan 1 yang merepresentasikan nomor clause. Jika tak ketemu, beritahu user bahwa jawaban tersebut tak ada.
3. Isi IF clause (yaitu, setiap variabel kondisi) dari statement.
4. Jika satu dari variabel pada IF clause belum diisi, yang diindikasikan oleh Var. List, dan bukan merupakan variabel konklusi, yaitu tak ada dalam Conclusion List, tanyakan user untuk memasukkan suatu nilai.
5. Jika satu dari clause adalah variabel konklusi, tempatkan nomor rule dari variabel konklusi di top of stack dan kembali ke langkah 3.
6. Jika statement pada top of stack tak dapat di-instantiated menggunakan statement IF-THEN yang ada, hapus dari top of stack dan cari pada Conclusion List untuk pengisian lain dari nama variabel konklusi.
7. Jika suatu statement ditemukan, kembali ke langkah 3.
8. Jika tak ada konklusi yang tersisa pada Conclusion Stack, rule untuk konklusi sebelumnya adalah salah. Jika tak ada konklusi sebelumnya, maka beritahu user jawaban tak ditemukan. Jika ada konklusi sebelumnya, kembali ke langkah 6.
9. Jika rule pada top of stack dapat di-instantiated, hapus rule tersebut dari stack. Jika ada variabel konklusi lain di bawahnya (pada stack), increment nomor clause, dan untuk clause yang tersisa kembali ke langkah 3. Jika tak ada variabel konklusi lain di bawahnya, didapatkan jawabannya.

Gambar 5.13 Algoritma implementasi backward chaining

5.9 Diagram Alur untuk Forward Chaining

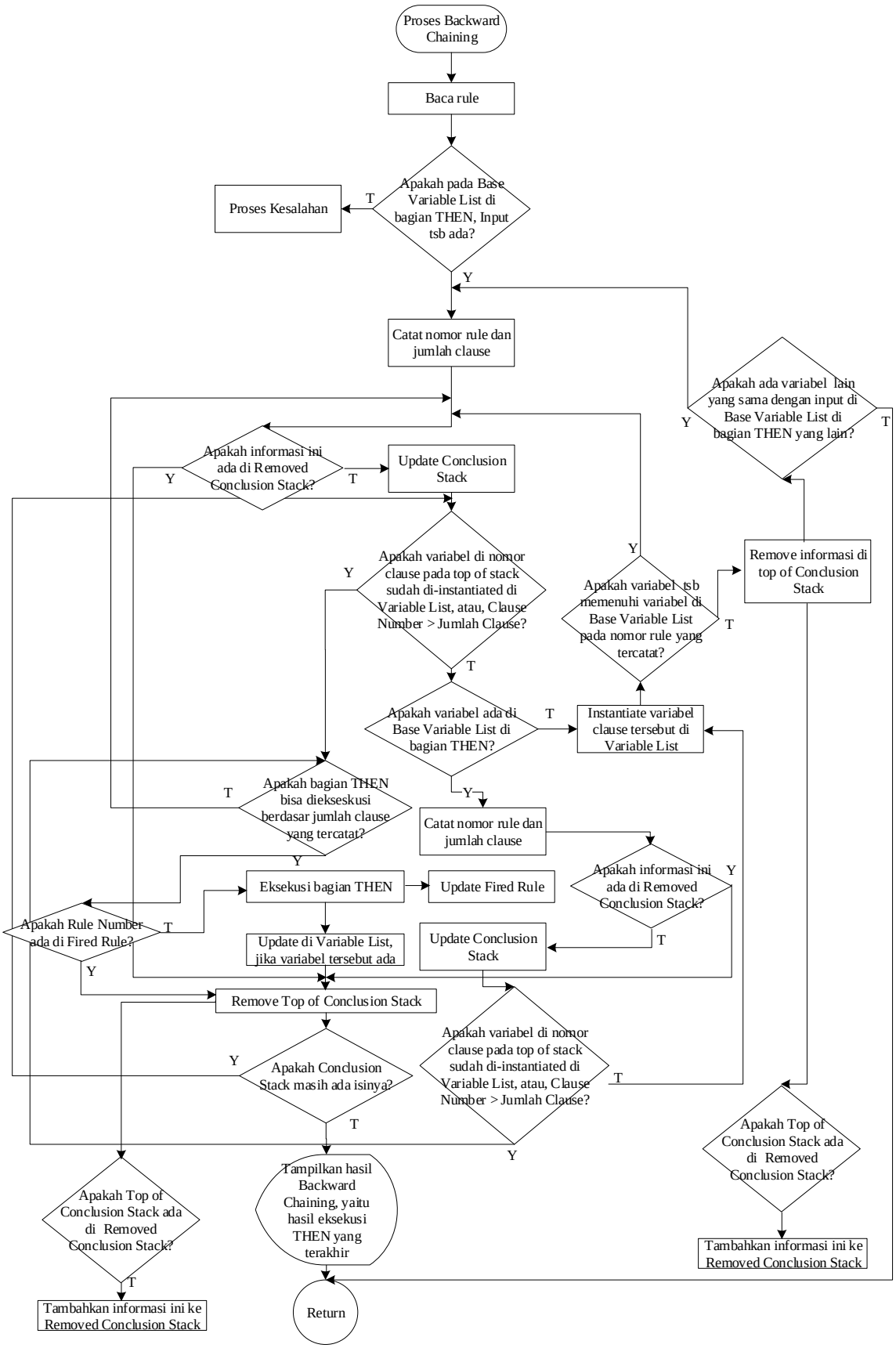
Diagram alur (flowchart) dari program forward chaining dari algoritma yang disajikan pada gambar 5.12 sebelumnya dapat dilihat pada gambar 5.14 di bawah ini:



Gambar 5.14 Diagram alur untuk forward chaining

5.10 Diagram Alur untuk Backward Chaining

Diagram alur (flowchart) dari program backward chaining dari algoritma yang disajikan pada gambar 5.13 sebelumnya dapat dilihat pada gambar 5.15 di bawah ini:



Gambar 5.15 Diagram alur untuk backward chaining

5.11 Contoh Kasus untuk Forward Chaining

Sebagai contoh kasus untuk proses forward chaining maka digunakan rule base yang memiliki rule-rule sebagai berikut ini:

```
rule 10
if interest = fall
then stock = rise
rule 20
if interest = rise
then stock = fall
rule 30
if dollar = fall
then interest = rise
rule 40
if dollar = rise
then interest = fall
rule 50
if fedint = fall
and fedmon = add
then interest = fall
```

Inisialisasi

Sebagai langkah awal program adalah membaca rule-rule tadi, yang selanjutnya membuat Base Variable List dan meng-inisialisasi Variable List.

Dalam Base Variable List, rule-rule tadi disusun sebagai berikut:

10	20	30	40	50	50
interest	interest	dollar	dollar	fedint	fedmon
=	=	=	=	=	=
fall	rise	fall	rise	fall	add
stock	stock	interest	interest	interest	interest
=	=	=	=	=	=
rise	fall	rise	fall	fall	fall
1	1	1	1	1	2

Inisialiasi Variable List, disini yang dicatat adalah hal-hal di bagian IF (kondisi)

Variable	Sign	Operator	Value
interest	NI		
dollar	NI		
fedint	NI		
fedmon	NI		

Sign NI = Not Instantiated, I = Instantiated, Operator dan Value masih bernilai kosong

1. Kemudian program akan menanyakan kepada user, hal-hal yang diketahuinya sebagai langkah awal Forward Chaining, dengan sintaks: <variable> <operator> <value>. Misal user memasukkan:

   ```
   fedmon = add
   ```

2. Apakah pada bagian IF Base Variable List, input tersebut ada?

   ```
   Ada (di rule nomor 50)
   ```

3. Catat variabel ini di Variable List

   ```
   Variable     Sign Operator  Value
   interest     NI
   dollar       NI
   fedint       NI
   fedmon       I    =         add
   ```

4. Cari variabel di Base Variable List, catat nomor rule dan jumlah clause-nya

   ```
   RuleNumber     = 50
   Jumlah Clause = 2
   ```

5. Inisial Conclusion Variable Queue

   ```
   fedmon
   ```

6. Update Clause Variable Pointer

   ```
   RuleNumber   = 50
   ClauseNumber = 1
   ```

   Berarti yang dicek sekarang adalah fedint

7. Apakah variable clause di Variable List sudah di-instantiate?

   ```
   fedint belum di-instantiated (Variable : fedint, Sign : NI)
   ```

8. Instantiate variable tadi di Variable List, disini ditanyakan kepada user. Misal user menginputkan:

   ```
   fedint = fall
   ```

   Maka di Variable List dilakukan updating, sebagai berikut:

   ```
   Variable     Sign Operator  Value
   interest     NI
   dollar       NI
   fedint       I    =         fall
   fedmon       I    =         add
   ```

9. Update Clause Variable Pointer

   ```
   RuleNumber   = 50
   ClauseNumber = 2
   ```

   Berarti yang dicek sekarang adalah fedmon

10. Apakah variable clause di Variable List sudah di-instantiate?

    fedmon sudah di-instantiate (Variable : fedmon, Sign : I)

11. Apakah clause pada Base Variable List di bagian IF, sudah terpenuhi semua berdasarkan jumlah clause yang tercatat?

    Ya, (Jumlah Clause = 2, ClauseNumber = 2)

12. Apakah variabel di variable clause memenuhi nilainya dengan variabel di Variable List?

    Ya, (fedint = fall dan fedmon = add sebagai variable clause sama nilainya dengan di Variable List : fedint = fall dan fedmon = add)

13. Eksekusi bagian THEN

    interest = fall

14. Update Conclusion Variable Queue

    fedmon
    interest

15. Catat variabel tersebut di record konklusi dan di Variable List kalau ada variabelnya di situ

    di Result dicatat:

    interest = fall

    di Variable List karena variabel interest ada, maka dicatat:

Variable	Sign	Operator	Value
interest	I	=	fall
dollar	NI		
fedint	I	=	fall
fedmon	I	=	add

16. Apakah variabel ini ada di rule lain yang tersisa pada Base Variable List di bagian IF?

    Tidak, karena RuleNumber = 50 adalah rule terakhir yang ada.

17. Remove variabel pertama di Conclusion Variable Queue

    ~~fedmon~~ (di-remove)
    interest

    Sehingga Conclusion Variable Queue menjadi:

    interest

18. Apakah variabel di Conclusion Variable Queue terdapat di Variabel List?

    Ya, interest di Conclusion Variable Queue terdapat di Variabel List

19. Catat nomor rule dan jumlah clause-nya

    RuleNumber = 10
    Jumlah Clause = 1

20. Update Clause Variable Pointer

```
RuleNumber   = 10
ClauseNumber = 1
```

21. Apakah variable clause di Variable List sudah di-instantiate?

```
interest sudah di-instantiate (Variable : interest, Sign : I)
```

22. Apakah clause pada Base Variable List di bagian IF, sudah terpenuhi semua berdasarkan jumlah clause yang tercatat?

```
Ya, (Jumlah Clause = 1, ClauseNumber = 1)
```

23. Apakah variabel di variable clause memenuhi nilainya dengan variabel di Variable List?

```
Ya, (interest = fall sebagai variable clause sama nilainya dengan di Variable List :
interest = fall)
```

24. Eksekusi bagian THEN

```
stock = rise
```

25. Update Conclusion Variable Queue

```
interest
stock
```

26. Catat variabel tersebut di record konklusi dan di Variable List kalau ada variabelnya di situ

    di Result dicatat:

```
stock = rise
```

    di Variable List karena variabel stock tak ada, maka tidak dicatat

27. Apakah variabel ini ada di rule lain yang tersisa pada Base Variable List di bagian IF?

```
Ya, ada di rule 20 (yaitu : interest = rise)
```

28. Catat nomor rule dan jumlah clause-nya

```
RuleNumber    = 20
Jumlah Number = 1
```

29. Update Clause Variable Pointer

```
RuleNumber   = 20
ClauseNumber = 1
```

30. Apakah variable clause di Variable List sudah di-instantiate?

```
interest sudah di-instantiate (Variable : interest, Sign : I)
```

31. Apakah clause pada Base Variable List di bagian IF, sudah terpenuhi semua berdasarkan jumlah clause yang tercatat?

Ya, (Jumlah Clause = 1, ClauseNumber = 1)

32. Apakah variabel di variable clause memenuhi nilainya dengan variabel di Variable List?

    Tidak, (interest = rise sebagai variable clause tidak sama nilainya dengan di Variable List : interest = fall)

33. Apakah variabel ini ada di rule lain yang tersisa pada Base Variable List di bagian IF?

    Tidak

34. Remove variabel pertama di Conclusion Variable Queue

    ~~interest~~ (di-remove)
    stock

    Sehingga Conclusion Variable Queue menjadi:

    stock

35. Apakah variabel di Conclusion Variable Queue terdapat di Variabel List?

    stock di Conclusion Variable Queue tidak ada di Variabel List

36. Tampilkan hasil Forward Chaining dari record konklusi dari Result:

    interest = fall
    stock = rise

37. Return

5.12 Contoh Kasus untuk Backward Chaining

Contoh kasus yang digunakan untuk proses backward chaining adalah rule base yang memiliki rule-rule sebagai berikut ini:

```
rule 10
if degree = no
then position = no
rule 20
if degree = yes
then qualify = yes
rule 30
if degree = yes
and discovery = yes
then position = research
rule 40
if qualify = yes
and grade < 3.5
and experience >= 2
then position = service engineering
rule 50
if qualify = yes
and grade < 3.5
and experience < 2 then position = no
rule 60
if qualify = yes
and grade >= 3.5
then position = product engineer
```

Inisialisasi

Sebagai langkah awal program adalah membaca rule-rule tadi, yang selanjutnya membuat Base Variable List dan meng-inisialisasi Variable List.

Dalam Base Variable List, rule-rule tadi disusun sebagai berikut:

```
10          20          30          30          40          40
degree      degree      degree      discovery   qualify     grade
=           =           =           =           =           <
no          yes         yes         yes         yes         3.5
position    qualify     position    position    position    position
=           =           =           =           =           =
no          yes         research    research    serv eng    serv eng
1           1           1           2           1           2


40          50          50          50          60          60
experience  qualify     grade       experience  qualify     grade
>=          =           <           <           =           >=
2           yes         3.5         2           yes         3.5
position    position    position    position    position    position
=           =           =           =           =           =
serv eng    no          no          no          prod eng    prod eng
3           1           2           3           1           2
```

Inisialiasi Variable List, disini yang dicatat adalah hal-hal di bagian IF (kondisi)

```
Variable   Sign     Operator  Value
degree     NI
discovery  NI
qualify    NI
grade      NI
experience NI
```

Sign NI = Not Instantiated, I = Instantiated, Operator dan Value masih bernilai kosong

1. Kemudian program akan menanyakan kepada user, hal-hal yang diketahuinya sebagai langkah awal Backward Chaining, dengan sintaks: <variable>. Misal user menginputkan:

   ```
   position
   ```

2. Apakah pada Base Variable List di bagian THEN, input tersebut ada?

   ```
   ada (di rule nomor 10)
   ```

3. Catat nomor rule dan jumlah clause

   ```
   RuleNumber     = 10
   Jumlah Clause = 1
   ```

4. Update Conclusion Stack. Namun sebelum updating dilakukan, kita buat struktur data bantu, Removed Conclusion Stack, untuk mencatat bagian antecedent (kondisi di bagian IF) yang telah diremove (dihapus) dari Conclusion Stack. Kondisi awal Removed Conclusion Stack adalah kosong, seperti di bawah ini.

   Karena informasi tersebut tidak ada di Removed Conclusion Stack, maka informasi ini bisa dimasukkan ke Conclusion Stack.

   ```
   RuleNumber   = 10
   ClauseNumber = 1
   ```
   (yaitu degree = no)

5. Apakah variabel di nomor clause pada top of stack sudah di-instantiate di Variable List, atau, ClauseNumber > Jumlah Clause

   ```
   degree pada Variable List belum di-instantiate
   ClauseNumber = 1 sama dengan JumlahClause
   ```

6. Apakah variabel ada di Base Variable List di bagian THEN?

   ```
   Tidak (degree tidak ada di bagian THEN)
   ```

7. Instantiate variable tadi di Variable List, disini ditanyakan kepada user. Misal user menginputkan:

   ```
   degree = yes
   ```

Maka di Variable List dilakukan updating, sebagai berikut:

Variable	Sign	Operator	Value
degree	I	=	yes
discovery	NI		
qualify	NI		
grade	NI		
experience	NI		

8. Apakah variabel tersebut memenuhi variabel di Base Variable List pada nomor rule yang tercatat?

   Tidak, (degree = yes tidak sama dengan di Base Variable List : degree = no)

9. Remove variabel di top of Conclusion Stack

   ~~RuleNumber = 10~~
   ~~ClauseNumber = 1~~

   Sehingga Conclusion Stack tidak ada isinya sekarang.

   Masukkan informasi ini ke Removed Conclusion Stack bila memang belum pernah ada di situ. Karena belum pernah ada, maka informasi tersebut bisa dimasukkan.

   RuleNumber = 10
   ClauseNumber = 1

10. Apakah ada variabel lain yang sama dengan input di Base Variable List di bagian THEN yang lain?

    Ada (yaitu di rule nomor 30, variabel position ada di bagian THEN)

11. Catat nomor rule dan jumlah clause

    RuleNumber = 30
    Jumlah Clause = 2

12. Update Conclusion Stack, dengan terlebih dahulu mengecek pada Removed Conclusion Stack ada tidaknya informasi tersebut. Karena memang belum ada di Removed Conclusion Stack, maka informasi ini bisa dimasukkan ke Conclusion Stack.

    RuleNumber = 30
    ClauseNumber = 1 (yaitu degree = yes)

13. Apakah variabel di nomor clause pada top of stack sudah di-instantiate di Variable List, atau, ClauseNumber > Jumlah Clause

    Ya, degree sudah di-instantiate (Variable : degree, Sign : I)

14. Apakah bagian THEN bisa dieksekusi berdasar jumlah clause yang tercatat?

    Tidak, ClauseNumber yang dicek baru nomor 1 sedangkan Jumlah Clause = 2

15. Update Conclusion Stack, dengan terlebih dahulu mengecek pada Removed Conclusion Stack ada tidaknya informasi tersebut. Karena memang belum ada di Removed Conclusion Stack, maka informasi ini bisa dimasukkan ke Conclusion Stack.

RuleNumber = 30 ClauseNumber = 2
RuleNumber = 30 ClauseNumber = 1

16. Apakah variabel di nomor clause pada top of stack sudah di-instantiate di Variable List, atau, ClauseNumber > Jumlah Clause

    Tidak, discovery belum di-instantiate (Variable : discovery, Sign : NI)

17. Apakah variabel ada di Base Variable List di bagian THEN?

    Tidak (discovery tidak ada di bagian THEN)

18. Instantiate variable tadi di Variable List, disini ditanyakan kepada user. Misal user menginputkan:

    discovery = no

    Maka di Variable List dilakukan updating, sebagai berikut:

Variable	Sign	Operator	Value
degree	I	=	yes
discovery	I	=	no
qualify	NI		
grade	NI		
experience	NI		

19. Apakah variabel tersebut memenuhi variabel di Base Variable List pada nomor rule yang tercatat?

    Tidak, (discovery = no tidak sama dengan di Base Variable List : discovery = yes)

20. Remove variabel di top of Conclusion Stack

~~RuleNumber = 30~~ ~~ClauseNumber = 2~~
RuleNumber = 30 ClauseNumber = 1

    Masukkan informasi ini ke Removed Conclusion Stack bila memang belum pernah ada di situ.

RuleNumber = 10 ClauseNumber = 1
RuleNumber = 30 ClauseNumber = 2

Conclusion Stack sekarang menjadi:

```
RuleNumber   = 30
ClauseNumber = 1
```
(yaitu degree = yes)

Dari sini, langkah seperti no. 13, no. 14 dan no. 15 kembali dilakukan.

Informasi di bawah ini sebenarnya akan di-update ke Conclusion Stack.

```
RuleNumber   = 30
ClauseNumber = 2     (yaitu discovery = yes)
```

Namun, seperti langkah sebelumnya, kita cek dulu pada Removed Conclusion Stack, apabila informasi tersebut sudah pernah ada di situ. Karena sudah pernah ada, updating informasi ini tidak jadi dilakukan pada Conclusion Stack, kemudian program meneruskan langkahnya dengan me-remove top of Conclusion Stack seperti di bawah ini.

~~RuleNumber = 30~~
~~ClauseNumber = 1~~

Masukkan informasi ini ke Removed Conclusion Stack bila memang belum pernah ada di situ. Informasi tersebut belum pernah ada, maka bisa dimasukkan ke Removed Conclusion Stack.

```
RuleNumber   = 10
ClauseNumber = 1
```
```
RuleNumber   = 30
ClauseNumber = 2
```
```
RuleNumber   = 30
ClauseNumber = 1
```

21. Apakah ada variabel lain yang sama dengan input di Base Variable List di bagian THEN yang lain?

    ```
    Ada (yaitu di rule nomor 40, variabel position ada di bagian THEN)
    ```

22. Catat nomor rule dan jumlah clause

    ```
    RuleNumber     = 40
    Jumlah Clause = 3
    ```

23. Update Conclusion Stack, dengan terlebih dahulu mengecek pada Removed Conclusion Stack ada tidaknya informasi tersebut. Karena memang belum ada di Removed Conclusion Stack, maka informasi ini bisa dimasukkan ke Conclusion Stack.

    ```
    RuleNumber   = 40
    ClauseNumber = 1
    ```
    (yaitu qualify = yes)

24. Apakah variabel di nomor clause pada top of stack sudah di-instantiate di Variable List, atau, ClauseNumber > Jumlah Clause

    Tidak, qualify belum di-instantiate (Variable : qualify, Sign : NI)

25. Apakah variabel tersebut (qualify) ada di Base Variable List di bagian THEN?

    Ya (qualify ada di bagian THEN)

26. Catat nomor rule dan jumlah clause

    RuleNumber = 20
    Jumlah Clause = 1

27. Update Conclusion Stack, dengan terlebih dahulu mengecek pada Removed Conclusion Stack ada tidaknya informasi tersebut. Karena memang belum ada di Removed Conclusion Stack, maka informasi ini bisa dimasukkan ke Conclusion Stack.

RuleNumber = 20 ClauseNumber = 1
RuleNumber = 40 ClauseNumber = 1

28. Apakah variabel di nomor clause pada top of stack sudah di-instantiate di Variable List, atau, ClauseNumber > Jumlah Clause

    Ya, degree sudah di-instantiate (Variable : degree, Sign : I)

29. Apakah bagian THEN bisa dieksekusi berdasar jumlah clause yang tercatat?

    Ya, ClauseNumber yang dicek nomor 1 sedangkan Jumlah Clause = 1

30. Sebelum kita bisa mengeksekusi bagian THEN, kita buat struktur data bantu, Fired Rule, untuk menyimpan informasi Rule Number mana yang telah berhasil dieksekusi (fired). Kondisi awal Fired Rule adalah kosong, seperti di bawah ini.

    Cek dulu pada bagian Fired Rule. Karena memang belum ada, maka Rule Number = 20 bisa dieksekusi seperti di bawah ini.

31. Eksekusi bagian THEN dan catat di bagian Result.

    qualify = yes

32. Update Rule Number ini dalam Fired Rule, seperti di bawah ini.

Rule Number: 20

33. Update di Variable List jika variabel tersebut ada

    qualify ada di Variable List, maka Variable List di-update

Variable	Sign	Operator	Value
degree	I	=	yes
discovery	I	=	no
qualify	I	=	yes
grade	NI		
experience	NI		

34. Remove Top of Conclusion Stack

~~RuleNumber = 20~~
~~ClauseNumber = 1~~

RuleNumber = 40
ClauseNumber = 1

Sehingga Conclusion Stack sekarang menjadi:

RuleNumber = 40
ClauseNumber = 1

Masukkan informasi ini ke Removed Conclusion Stack bila memang belum pernah ada di situ. Karena belum pernah ada, maka informasi tersebut bisa dimasukkan.

RuleNumber = 10
ClauseNumber = 1

RuleNumber = 30
ClauseNumber = 2

RuleNumber = 30
ClauseNumber = 1

RuleNumber = 20
ClauseNumber = 1

35. Apakah Conclusion Stack masih ada isinya?

    Ya, yaitu

RuleNumber = 40
ClauseNumber = 1

36. Apakah variabel di nomor clause pada top of stack sudah di-instantiate di Variable List, atau, ClauseNumber > Jumlah Clause

    Ya, qualify sudah di-instantiate (Variable : qualify, Sign : I)

37. Apakah bagian THEN bisa dieksekusi berdasar jumlah clause yang tercatat?

    Tidak, ClauseNumber yang dicek baru nomor 1 sedangkan Jumlah Clause = 3

38. Update Conclusion Stack, dengan terlebih dahulu mengecek pada Removed Conclusion Stack ada tidaknya informasi tersebut. Karena memang belum ada di Removed Conclusion Stack, maka informasi ini bisa dimasukkan ke Conclusion Stack.

RuleNumber = 40 ClauseNumber = 2
RuleNumber = 40 ClauseNumber = 1

39. Apakah variabel di nomor clause pada top of stack sudah di-instantiate di Variable List, atau, ClauseNumber > Jumlah Clause

    Tidak, grade belum di-instantiate (Variable : grade, Sign : NI)

40. Apakah variabel ada di Base Variable List di bagian THEN?

    Tidak, grade tidak ada di bagian THEN

41. Instantiate variable tadi di Variable List, disini ditanyakan kepada user. Misal user menginputkan:

    grade = 3.0

    Maka di Variable List dilakukan updating, sebagai berikut:

Variable	Sign	Operator	Value
degree	I	=	yes
discovery	I	=	no
qualify	I	=	yes
grade	I	=	3.0
experience	NI		

42. Apakah variabel tersebut memenuhi variabel di Base Variable List pada nomor rule yang tercatat?

    Ya, (grade = 3.0 memenuhi di Base Variable List : grade < 3.5)

43. Update Conclusion Stack

RuleNumber = 40 ClauseNumber = 3
RuleNumber = 40 ClauseNumber = 2
RuleNumber = 40 ClauseNumber = 1

44. Apakah variabel di nomor clause pada top of stack sudah di-instantiate di Variable List, atau, ClauseNumber > Jumlah Clause

    Tidak, experience belum di-instantiate (Variable : experience, Sign : NI)

45. Apakah variabel ada di Base Variable List di bagian THEN?

    Tidak, experience tidak ada di bagian THEN

46. Instantiate variable tadi di Variable List, disini ditanyakan kepada user. Misal user menginputkan:

    experience = 4.5

    Maka di Variable List dilakukan updating, sebagai berikut:

Variable	Sign	Operator	Value
degree	I	=	yes
discovery	I	=	no
qualify	I	=	yes
grade	I	=	3.0
experience	I	=	4.5

47. Apakah variabel tersebut memenuhi variabel di Base Variable List pada nomor rule yang tercatat?

    Ya, (experience = 4.5 memenuhi di Base Variable List : experience >= 2)

48. Update Conclusion Stack

RuleNumber = 40 ClauseNumber = 4
RuleNumber = 40 ClauseNumber = 3
RuleNumber = 40 ClauseNumber = 2
RuleNumber = 40 ClauseNumber = 1

49. Apakah variabel di nomor clause pada top of stack sudah di-instantiate di Variable List, atau, ClauseNumber > Jumlah Clause

    ClauseNumber 4 pada RuleNumber 40 sebenarnya tidak ada, jadi tidak mungkin ada variabelnya yang sudah di-instantiate di Variable List. Namun, karena ClauseNumber (4) sudah melebihi Jumlah Clause (3), maka hasil dari kondisi ini adalah true (benar).

50. Apakah bagian THEN bisa dieksekusi berdasar jumlah clause yang tercatat?

    Ya, ClauseNumber sudah dicek semuanya, sejumlah Jumlah Clause

51. Cek dulu pada bagian Fired Rule. Rule Number = 20 belum ada. Hanya Rule Number = 20 yang pernah dieksekusi, maka Rule Number = 40 bisa dieksekusi seperti di bawah ini.

Rule Number: 20

52. Eksekusi bagian THEN dan catat di bagian Result.

    position = service engineering

53. Update Rule Number ini dalam Fired Rule, seperti di bawah ini.

Rule Number: 20
Rule Number: 40

54. Update di Variable List jika variabel tersebut ada

Karena position tidak ada di Variable List maka tidak terjadi updating

55. Remove Top of Conclusion Stack

~~RuleNumber = 40~~ ~~ClauseNumber = 4~~
RuleNumber = 40 ClauseNumber = 3
RuleNumber = 40 ClauseNumber = 2
RuleNumber = 40 ClauseNumber = 1

Masukkan informasi ini ke Removed Conclusion Stack bila memang belum pernah ada di situ.

RuleNumber = 10 ClauseNumber = 1
RuleNumber = 30 ClauseNumber = 2
RuleNumber = 30 ClauseNumber = 1
RuleNumber = 20 ClauseNumber = 1
RuleNumber = 40 ClauseNumber = 4

56. Apakah Conclusion Stack masih ada isinya?

Ya, yaitu

RuleNumber = 40 ClauseNumber = 3
RuleNumber = 40 ClauseNumber = 2
RuleNumber = 40 ClauseNumber = 1

57. Apakah variabel di nomor clause pada top of stack sudah di-instantiate di Variable List, atau, ClauseNumber > Jumlah Clause

Ya, experience sudah di-instantiate (Variable : experience, Sign : I)

58. Apakah bagian THEN bisa dieksekusi berdasar jumlah clause yang tercatat?

Ya, ClauseNumber sudah dicek semuanya, sejumlah Jumlah Clause

59. Cek dulu pada bagian Fired Rule. Kalau memang belum ada, maka rule dengan Rule Number tersebut, bisa dieksekusi. Isi Fired Rule adalah sebagai berikut:

Rule Number: 20
Rule Number: 40

Sehingga rule dengan Rule Number = 40 di Conclusion Stacks tidak bisa di-eksekusi.

60. Remove Top of Conclusion Stack

~~RuleNumber = 40~~ ~~ClauseNumber = 3~~
RuleNumber = 40 ClauseNumber = 2
RuleNumber = 40 ClauseNumber = 1

Masukkan informasi ini ke Removed Conclusion Stack bila memang belum pernah ada di situ.

RuleNumber = 10 ClauseNumber = 1
RuleNumber = 30 ClauseNumber = 2
RuleNumber = 30 ClauseNumber = 1
RuleNumber = 20 ClauseNumber = 1
RuleNumber = 40 ClauseNumber = 4
RuleNumber = 40 ClauseNumber = 3

61. Apakah Conclusion Stack masih ada isinya?

Ya, yaitu

RuleNumber = 40 ClauseNumber = 2
RuleNumber = 40 ClauseNumber = 1

62. Apakah variabel di nomor clause pada top of stack sudah di-instantiate di Variable List, atau, ClauseNumber > Jumlah Clause

Ya, grade sudah di-instantiate (Variable : experience, Sign : I)

63. Apakah bagian THEN bisa dieksekusi berdasar jumlah clause yang tercatat?

Ya, ClauseNumber sudah dicek semuanya, sejumlah Jumlah Clause

64. Cek dulu pada bagian Fired Rule. Kalau memang belum ada, maka rule dengan Rule Number tersebut, bisa dieksekusi. Isi Fired Rule adalah sebagai berikut:

Rule Number: 20
Rule Number: 40

Sehingga rule dengan Rule Number = 40 di Conclusion Stacks tidak bisa di-eksekusi.

65. Remove Top of Conclusion Stack

~~RuleNumber = 40~~ ~~ClauseNumber = 2~~
RuleNumber = 40 ClauseNumber = 1

Masukkan informasi ini ke Removed Conclusion Stack bila memang belum pernah ada di situ.

RuleNumber = 10 ClauseNumber = 1
RuleNumber = 30 ClauseNumber = 2
RuleNumber = 30 ClauseNumber = 1
RuleNumber = 20 ClauseNumber = 1
RuleNumber = 40 ClauseNumber = 4
RuleNumber = 40 ClauseNumber = 3
RuleNumber = 40 ClauseNumber = 2

66. Apakah Conclusion Stack masih ada isinya?

Ya, yaitu

RuleNumber = 40 ClauseNumber = 1

67. Apakah variabel di nomor clause pada top of stack sudah di-instantiate di Variable List, atau, ClauseNumber > Jumlah Clause

Ya, qualify sudah di-instantiate (Variable : experience, Sign : I)

68. Apakah bagian THEN bisa dieksekusi berdasar jumlah clause yang tercatat?

Ya, ClauseNumber sudah dicek semuanya, sejumlah Jumlah Clause

69. Cek dulu pada bagian Fired Rule. Kalau memang belum ada, maka rule dengan Rule Number tersebut, bisa dieksekusi. Isi Fired Rule adalah sebagai berikut:

Rule Number: 20
Rule Number: 40

Sehingga rule dengan Rule Number = 40 di Conclusion Stacks tidak bisa di-eksekusi.

70. Remove Top of Conclusion Stack

~~RuleNumber = 40~~ ~~ClauseNumber = 1~~

Masukkan informasi ini ke Removed Conclusion Stack bila memang belum pernah ada di situ.

RuleNumber = 10 ClauseNumber = 1
RuleNumber = 30 ClauseNumber = 2
RuleNumber = 30 ClauseNumber = 1
RuleNumber = 20 ClauseNumber = 1
RuleNumber = 40 ClauseNumber = 4
RuleNumber = 40 ClauseNumber = 3
RuleNumber = 40 ClauseNumber = 2
RuleNumber = 40 ClauseNumber = 1

71. Apakah Conclusion Stack masih ada isinya?

Tidak, Conclusion Stack sudah kosong, seperti ini.

72. Tampilkan hasil Backward Chaining yang diambil dari bagian Result

```
qualify = yes
position = service engineering
```

73. Return

Bab 6 Confidence Factor pada SBP

Bab ini membahas mengenai penjeladan dan penggunaan confidence factor pada sistem berbasis pengetahuan. Pustaka yang digunakan adalah berasal dari J.P. Ignizio [Ign91], G.R. Baur and D.V. Pigford [Baur90], serta MM Irfan Subakti dan Alexander L. Romy [Sub03].

6.1 Confidence Factor

Sistem Berbasis Pengetahuan (SBP) yang telah kita bahas dalam bab 5 sebelumnya masih banyak memiliki keterbatasan. Salah satunya adalah belum disediakan suatu mekanisme untuk menilai tingkat kepercayaan kita terhadap rule (clause, konklusi, variabel dan nilainya). Kita anggap bahwa semua rule memiliki tingkat kepercayaan yang sama, yaitu 100%. Padahal dalam dunia nyata, jarang sekali kita dapati tingkat kepercayaan seperti ini.

Confidence Factor (CF, tingkat kepercayaan, boleh juga diartikan sebagai Certainty Factor) datang untuk mengatasi permasalahan di atas. Dengannya SBP akan dapat mengakomodasikan persepsi yang digunakan dalam menentukan keputusan. Dan memang dengan persepsi inilah, bukan fakta sebenarnya, hampir semua keputusan manusia diambil.

CF adalah ukuran/tingkat kepercayaan seseorang terhadap rule yang ada. Meliputi tingkat kepercayaan terhadap suatu variabel dalam suatu rule, maupun tingkat kepercayaan terhadap rule itu sendiri. Makin tinggi CF-nya jelas makin bisa dipercaya suatu rule, sehingga diasumsikan rule tersebut berarti makin baik.

Dengan adanya CF ini, urutan rule pada suatu knowledge base tidak berpengaruh pada prioritas eksekusi program. Sehingga prioritas/keutamaan suatu rule ditentukan oleh besarnya CF yang ada padanya.

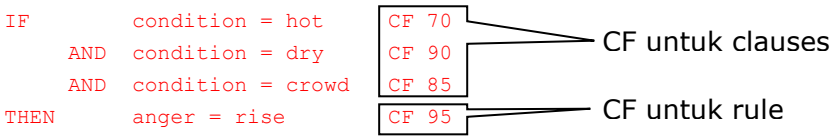
Kita bisa menentukan jangkauan dari CF, misal diantara 0 s/d 100, boleh juga 0 s/d 1.

Sebagai catatan, perlu dikemukakan disini bahwa seorang pakar berhak untuk menentukan nilai CF untuk variabel-variabel dalam clauses suatu rule dan juga CF untuk rule yang bersangkutan. Namun perlu diingat, adalah hak pengguna/user untuk menentukan nilai CF yang dipercayainya sewaktu proses inferensia. Juga dalam proses inferensia, kita dapat mengasumsikan untuk pengguna yang tidak tahu/tidak mau menentukan nilai CF suatu variabel/rule, maka bisa ditawarkan pada pengguna untuk

menggunakan nilai-nilai CF yang telah diisikan oleh pakar dalam rule base-nya. Bisa juga dengan suatu perjanjian dimuka dimana pengguna harus mengisikan nilai CF untuk nilai CF default, untuk nilai yang tak diketahui/tak dimaui pengguna untuk CF dalam masukkan untuk proses inferensianya.

6.2 CF Aktual Rule untuk Rule dengan Operator AND

Sebagai contoh, dimisalkan kita mempunyai rule-rule dengan nilai-nilai variabel, operator dan nilai seperti berikut ini.



CF untuk kondisi di atas adalah:

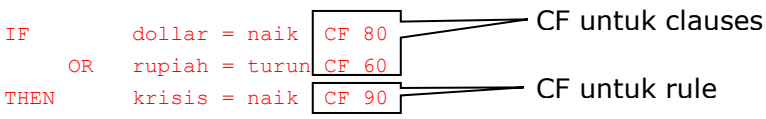
CF = minimum (CF1, CF2, CF3) = minimum (70, 90, 85) = 70

Sehingga CF aktual untuk rule tersebut adalah:

CF(rule) = 70 * 95 = 6650/100 = 67

6.3 CF Aktual Rule untuk Rule dengan Operator OR

Untuk rule-rule dengan operator OR kita ambil sebagai contoh, misalkan kita mempunyai rule-rule dengan nilai-nilai variabel, operator dan nilai seperti berikut ini.



Maka CF = 80 + 60 – (80 * 60)/100

= 140 – 48 = 92

CF aktual rule didapat = 92 * 90

= 8280, lalu bagi dengan 100 = 8280/100 = 82,8

Setelah di-ROUND, maka didapatkan CF aktual rule, yaitu 83

Untuk contoh lain, dimisalkan kita memiliki rule dengan kondisi variabel, operator dan nilai seperti ini.

```
IF        activity = letters  CF 80
     OR   activity = resumes  CF 70
     OR   activity = notices  CF 85
THEN      advice = need       CF 90
```

CF = 80 + 70 – (80 * 70)

= 150 – 56; 56 didapat dari (80 * 70)/100

= 94;

Dari sini didapatkan CF untuk (activity = letters CF 80 OR activity = resumes CF 70)

Lalu dilanjutkan dengan (OR activity = notices CF 85)

CF = 94 + 85 – (94 * 85) = 99

Maka CF aktual rule = 99 * 90 = 89; dimana 89 didapat dari (99 * 90)/100 setelah di-ROUND.

6.4 CF Aktual Rule untuk Rule dengan Operator AND/OR

Untuk contoh kasusnya, dimisalkan kita mempunyai rule-rule dengan kondisi variabel, operator dan nilai sebagai berikut ini.

```
RULE A    IF   activity = letters OR
               activity = resumes AND
               cost = low
          THEN word_processor = D CF 90
```

CF untuk rule

Dari sini terlihat adanya operato OR dan AND sekaligus dalam RULE A, maka untuk mencari CF-nya, rule seperti ini dapat dipecah menjadi:

```
RULE A1:  IF   activity = letter AND
               cost = low
          THEN word_processor = D CF 90
OR
RULE A2:  IF   activity = resume AND
               cost = low
          THEN word_processor = D CF 90
```

Agar lebih jelas akan diberikan contoh lengkap untuk rule-nya. Dimisalkan bahwa CF

telah dimasukkan oleh pengguna sehingga kondisi rule-nya sampai saat ini adalah:

```
RULE A1:  IF   activity = letter CF 90 AND
               cost = low CF 80
          THEN word_processor = D CF 90
OR
RULE A2:  IF   activity = resume CF 70 AND
               cost = low CF 80
          THEN word_processor = D CF 90
```

CF kondisi pada RULE A1 = minimum (90, 80) = 80

CF aktual untuk RULE A1 = 80 * 90 = 7200/100 = 72

CF kondisi pada RULE A2 = minimum (70, 80) = 70

CF aktual untuk RULE A2 = 70 * 90 = 6300/100 = 63

CF untuk kedua RULE didapatkan dengan rumus OR sehingga didapat:

72 + 63 – (72 * 63) = 135 – 4536/100

= 135 – 45

= 90

Jadi, CF aktual dari rule aslinya pada contoh di atas yaitu 90

6.5 Pengembangan Program Forward dan Backward Chaining

Disamping pernyataan IF .. THEN .. pada RULE yang sudah dibahas, sebenarnya kita dapat mengembangkan program yang telah dibahas pada bab selanjutnya, yaitu dengan format yang lebih lengkap lagi yaitu IF .. THEN .. ELSE .. BECAUSE .., seperti tertulis di bawah ini:

```
IF   condition
THEN conclusion1
ELSE conclusion2
BECAUSE "..."
```

Ada yang menarik disini mengenai kata cadangan ELSE. Dengan adanya ELSE secara cerdas sistem dapat menyimpan 2 kondisi cukup dalam 1 rule saja. Artinya pada kondisi A, maka sistem akan mengambil nilai dari clauses dan bila clauses-nya memenuhi syarat maka conclusion1 akan dieksekusi.

Sedangkan secara implisit sistem juga akan tahu kalau pengguna mengisikan lawan nilai

dari kondisi A (sebut saja kondisi B, yang didapat dari "NOT kondisi A") maka yang akan dieksekusi adalah conclusion2 (yang merupakan lawan dari conclusion1).

Tentu saja diperlukan mekanisme lebih lanjut agar sistem kita mengetahui mana-mana kondisi yang menjadi lawan dari suatu kondisi lain, sehingga sistem secara cerdas akan dapat memberikan hasil yang diharapkan.

Sebagai contoh:

```
RULE 7
IF   functionallity = complex
THEN word_processor = product_B
ELSE word_processor = product_A
BECAUSE "Product B can process more complicated documents that Product A"
```

Program juga dapat mengenali dan menangani persamaan, pertidaksamaan, dan juga jangkauan nilai tertentu untuk nilai-nilai yang dibandingkan dengan variabel-variabelnya.

Contoh:

```
IF warna = hitam, IF warna <> hitam, IF warna < 15, IF warna IN [0..7], IF kondisi = TRUE,
dsb.
```

Pembagian Rule Base

Rule base dibagi dalam 3 bagian: ACTION, RULE, dan STATEMENT BLOCK, seperti contoh di bawah ini:

```
//The file finding_word_processor.kbs uses a PLURAL variable,
//Activity, in a compound OR IF-THEN-ELSE-BECAUSE

//ACTION BLOCK
ACTIONS
     DISPLAY   "WPHELP will advise you on whether you need a word " +
               "processor or not. Press OK to start the consultation."
     FIND advice    //variabel yang akan dicari, langsung saja ditandai
                    //dengan kata FIND
     DISPLAY   "You {advice} a word processor at this time. " +
               "Press OK to exit WPHELP";
//tanda {} pada {advice} berarti bahwa yang diapit oleh tanda itu
//ialah variabel yang ditampilkan
//setiap selesai 1 statemen maka diakhiri dengan tanda ";" titik koma


//RULES BLOCK
RULE 1
IF   activity = letters
     OR activity = resumes
     OR activity = notices
     OR activity = papers
```

```
THEN advice = need
ELSE advice = do_not_need
BECAUSE   "If you generate frequent letters, resumes, notices, " +
          "or papers, then you need a word processor.";

//STATEMENTS BLOCK
ASK activity:  "Which of the following documents do you need to write " +
               "for your business? Select you answers by select " +
               "your desired answer. After you have made all " +
               "your selections, click or press OK button. " +
               "If you don't need to write any of these documents, " +
               "select none_of_these";
//Pada program begitu program membutuhkan masukan dari user,
//maka ditanyakan dulu ke user
//kemudian setelah pertanyaan tadi, akan langsung disajikan dari
//nilai-nilai yang mungkin
//ada pada variabel, setelah selesai user dapat
//meng-klik/meng-enter tombol OK
//Kemungkinan nilai pada variabel tadi di-declare dengan kata CHOICES
//seperti di bawah ini

CHOICES activity: letters, resumes, notices, papers, none_of_these;
PLURAL: activity;
//Reserved word PLURAL menyatakan bahwa variabel setelahnya adalah
//variabel yang nilainya
//majemuk=PLURAL, jadi tidak hanya 1 saja, bisa 2 atau lebih
```

Contoh rule base yang lain:

```
ACTIONS
     FIND word_processor;
     DISPLAY "The value of word_processor is {word_processor}"

RULE 0
IF   functionality = simple AND cost = low
THEN word_processor = Product_A;
RULE 1
IF   functionality = simple AND cost = high
THEN word_processor = Product_A;
RULE 2
IF   functionality = complex AND cost = low
THEN word_processor = none;
RULE 3
IF   functionality = complex AND cost = high
THEN word_processor = Product_B;

ASK functionality: "What is the value of Functionality?";
CHOICES functionality: simple,complex;
ASK cost: "What is the value of Cost?";
CHOICES cost: low,high;
```

Implementasi Pengembangan Program

Bisa juga dikembangkan sintaks rule menurut yang diinginkan. Semakin lengkap jelas akan semakin baik. Sebagai panduan, bisa dilihat hal-hal di bawah ini:

- Ada fasilitas *trace*/penelusuran program.
- User interface sebaik dan semudah mungkin digunakan.
- Memberikan data yang lengkap kepada user.
- Bisa menampilkan jendela WHAT, HOW, WHY dengan isi yang lengkap. Isi dari jendela WHAT adalah hasil dari proses forward maupun backward chaining yang paling sederhana dan yang pertama kali didapatkan dari program, bahkan sebelum tahap pengembangan ini. Ia menjawab pertanyaan mengenai apa yang dihasilkan program. Sedangkan isi dari jendela jendela HOW adalah log/catatan bagaimana suatu hasil (WHAT) didapatkan, rule-rule mana saja yang telah diproses dan ditelusuri. Sedangkan isi dari jendela WHY, lebih banyak mengacu pada alasan hasil (WHAT) tadi didapat. Tentu saja komponen CF yang banyak mengambil peranan dalam penjelasan WHY ini disamping isi pernyataan-pernyataan yang disimpan di rule base yang ditandai dengan kata cadangan BECAUSE. Singkatnya adalah WHAT adalah hasilnya, HOW adalah bagaimana menghasilkan WHAT tadi, sedangkan alasan suatu hasil bisa didapat dijelaskan oleh WHY.
- Bisa mengeksekusi semua rule, dengan catatan mengikuti sintaks program yang ada.
- Bisa menganalisis rule yang ada.
- Pencarian berjalan dengan cepat, apalagi untuk rule-rule yang kompleks dan besar.
- Untuk pendekatan Forward Chaining, ada tambahan reserved word: WHENEVER.

```
WHENEVER cost_too_high
IF cost >= 500
THEN DISPLAY "Consider volume discounts.";
```

Begitu kata WHENEVER ditemukan di bagian awal rule, maka rule akan dites setiap saat nilai dari variabel yang ada di kondisi (bagian IF) berubah nilainya.

Jika rule yang mengandung WHENEVER ditemukan sebagai konklusi yang benar, maka bagian THEN dieksekusi. Dengan kata lain, bila ada perubahan variabel di bagian IF, rule secara otomatis dicek untuk dipilih, tanpa memperhatikan strategi pencariannya apakah Forward atau Backward Chaining.

Jadi WHENEVER berlaku seperti "daemon" sebab dia akan mengawasi variabel yang diacu oleh kondisi IF, dan mengeksekusinya setiap waktu kondisinya true/benar.

- Adanya reserved word FIND untuk mengidentifikasikan variabel tujuan.

- Reserved word FIND dalam rule hanya boleh ada dalam bagian THEN saja, tidak boleh di bagian IF.

Contoh:

```
RULE Networking
IF   Environment = Networked
THEN Network = Yes
     FIND Networked_word_processor
```

- FIND tidak bisa digabungkan dengan WHENEVER, sebab WHENEVER mengindikasikan rule, sedang FIND mengidentifikasikan variabel tujuan.

Contoh yang salah:

```
WHENEVER  Networking
IF        Environment = Networked
THEN      Network = Yes
          FIND Networked_word_processor
```

- Untuk nilai dari variabel yang belum diketahui, ada satu nilai khusus yaitu UNKNOWN, yang berarti program memang tidak tahu harus mengisikan apa pada variabel yang ditanyakan. Jadi cukup diisi saja dengan UNKNOWN bila kita memang tidak tahu jawaban apa yang harus diberikan.

```
RULE 7
IF   Document = UNKNOWN
THEN Word_processor = wait_for_now
     DISPLAY "Examine your busines functions for word processing before buying";
RULE 8
IF   Cost = UNKNOWN
THEN Word_processor = none_for_now
     DISPLAY "Consult your manager or boss to determine your budget.";
```

Bab 7 Algoritma Genetika

Yang dibahas dalam bab ini adalah algoritma genetika dan contoh praktisnya dalam kehidupan sehari-hari yang dibahas secara lengkap untuk kepentingan pemrograman. Pustka yang digunakan adalah dari Mitsuo Gen and Runwei Chen [Gen97].

7.1 Pendahuluan

Pelbagai masalah optimasi baik di dunia rekayasa industri, utamanya pada sistem manufakturing, begitu kompleksnya untuk ditangani dengan teknik optimasi konvensional.

Mulai tahun 1960-an mulai banyak studi yang berusaha untuk menirukan perilaku kehidupan dalam rangka menyelesaikan masalah optimasi yang pelik seperti contoh di atas. Simulasi proses evolusioner alamiah manusia menghasilkan teknik optimasi stokastik yang disebut *algoritma evolusioner*, yang seringkali dapat menampilkan kinerja yang lebih baik daripada metode konvensional ketika diaplikasikan pada permasalahan dunia nyata yang sulit.

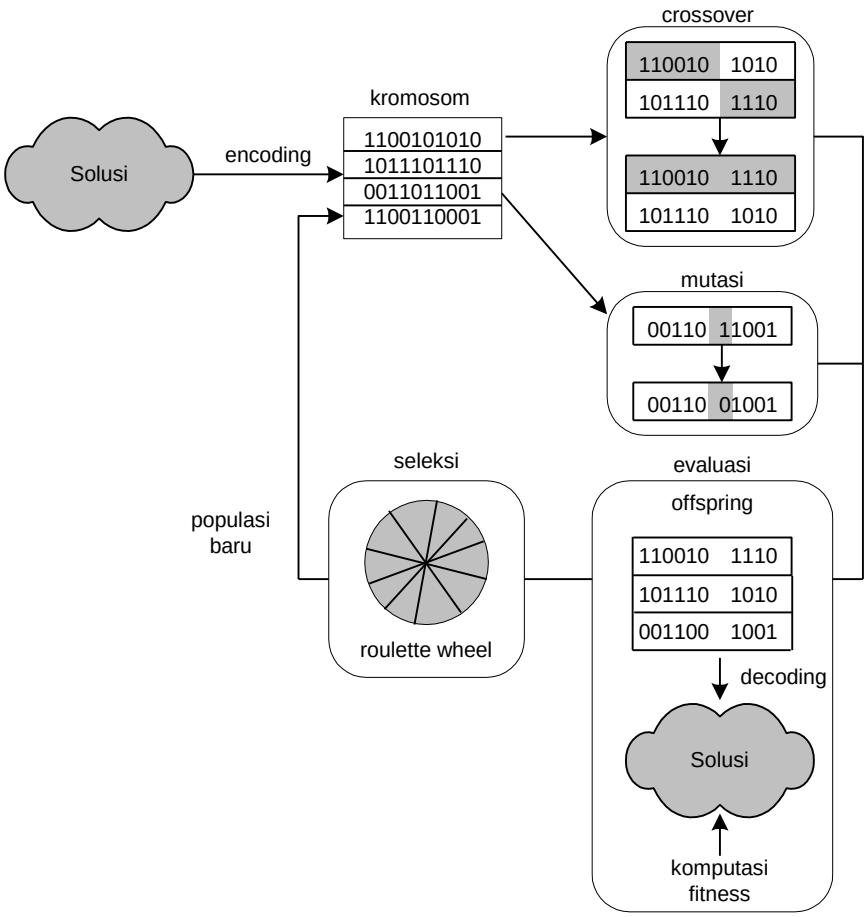
Ada tiga jalan utama dalam riset tersebut: Genetic Algorithms (GA, algoritma genetika), Evolutionary Programming (EP, pemrograman evolusioner), dan Evolution Strategies (ESs, Strategi-strategi Evolusi). Diantara ketiganya, algoritma genetika dikenal sebagai algoritma yang paling banyak dipakai pada saat ini.

Penerapan algoritma genetika yang sudah diterapkan di bidang rekayasa industri adalah dalam hal: penjadwalan dan prioritas urutan, desain kehandalan, rute dan penjadwalan kendaraan, group technology, layout dan lokasi fasilitas, transportasi, dan lain-lain.

Dalam hubungannya dengan Sistem Cerdas, GA ini diharapkan dapat mengoptimasi pencarian baik dalam Forward maupun Backward chaining, dan harapan selanjutnya adalah bersama-sama dengan logika fuzzy dapat lebih mengoptimalkan sistem dalam hal kecepatan pembelajaran, kecepatan berfikir, dan kecepatan memberikan solusi untuk menjawab semua pertanyaan yang berkaitan dengan WHAT, HOW, dan WHY.

7.2 Struktur Umum Algoritma Genetika

Untuk lebih memudahkan kita memahami algoritma genetika, terlebih dulu akan digambarkan diagram struktur umum dari algoritma genetika seperti gambar 7.1.



Gambar 7.1 Diagram struktur umum algoritma genetika

Struktur umum dari algoritma genetika di atas dapat diimplementasikan dalam program dengan algoritma yang dapat dituliskan dalam pseudocode seperti di bawah ini.

```
Prosedur: Algoritma Genetika
begin
     t ← 0;
     initialize P(t);
     evaluate P(t);
     while (not termination condition) do
          recombine P(t) to yield C(t);
          evaluate C(t);
          select P(t+1) from P(t) and C(t);
          t ← t + 1;
     end
end
```

Sebenarnya, cuma ada 2 jenis operasi dalam algoritma genetika:

1. Operasi genetika: crossover (pindah silang) dan mutasi
2. Operasi evolusi: seleksi

Algoritma genetika berbeda dengan optimasi konvensional dan prosedur pencarian dalam pelbagai hal, seperti hal-hal di bawah ini:

1. GA bekerja dengan coding (transformasi bentuk) dari kumpulan solusi, bukan solusi itu sendiri.
2. GA mencari dari populasi solusi, bukan solusi tunggal.
3. GA menggunakan informasi hasil (fungsi fitness), bukan penurunan atau pengetahuan bantuan lainnya.
4. GA menggunakan rule transisi probabilistik, bukan rule deterministik.

7.3 Eksploitasi dan Eksplorasi

2 isu penting dalam strategi pencarian: eksploitasi solusi terbaik dan eksplorasi ruang pencarian.

GA merupakan metode pencarian umum yang mengkombinasikan elemen-elemen pencarian terarah dan stokastik yang dapat membuat keseimbangan diantara eksplorasi dan eksploitasi dari ruang pencarian.

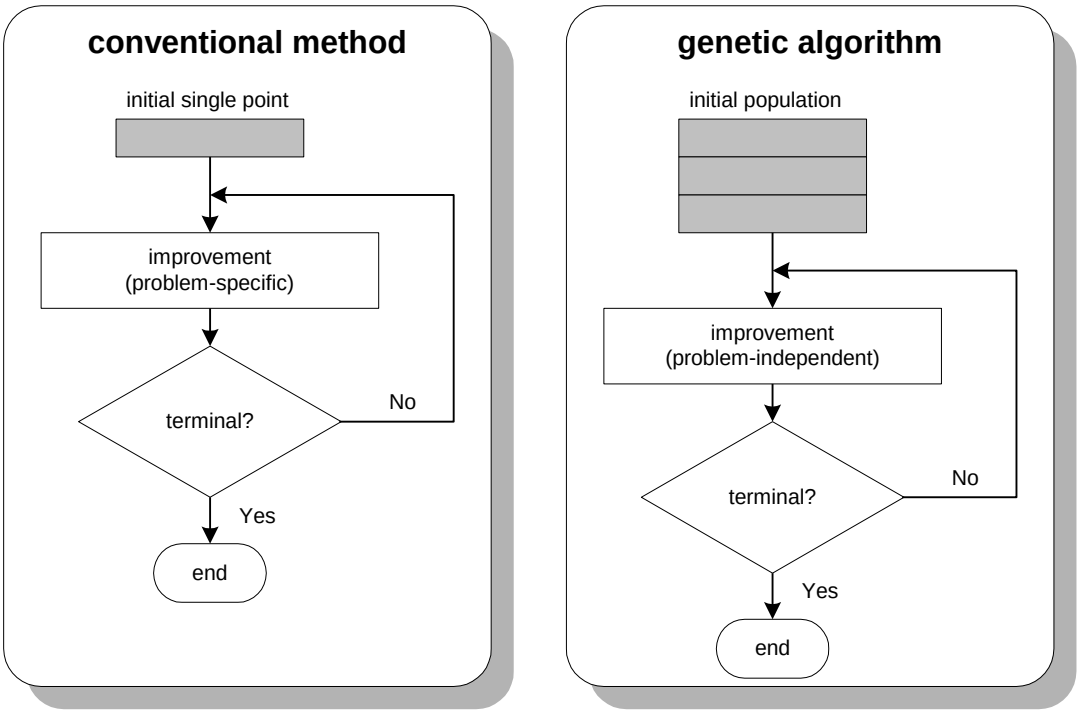
Pada awal pencarian dalam pencarian genetika, terdapat populasi yang berbeda dan sangat acak, lalu operator crossover cenderung untuk menampilkan pencarian melebar untuk mengeksplorasi semua ruang solusi.

Dari hasil pengembangan solusi yang memiliki fitness tinggi, operator crossover menampilkan eksplorasi pada tetangga dari setiap solusi tersebut. Dengan kata lain, jenis apapun pencarian (eksploitasi atau eksplorasi) kinerja crossover ditentukan oleh lingkungan sistem genetika (keanekaragaman populasi), tetapi bukan oleh operator itu sendiri.

Sebagai tambahan, operator-operator genetika sederhana didesain sebagai metode pencarian umum (metode pencarian yang tak tergantung domain); mereka menampilkan secara esensial pencarian "buta" dan tak dapat menjamin untuk menghasilkan offspring yang makin baik.

7.4 Pencarian Berdasarkan Populasi

Perbandingan antara pendekatan konvensional dan dengan menggunakan algoritma genetika dapat dilihat pada gambar 7.2 di bawah ini.



Gambar 7.2 Perbandingan metode konvensional dan algoritma genetika

Meta-heuristic

GA sederhana sulit diaplikasikan secara langsung dan berhasil dalam banyak masalah optimasi yang "sulit".

Pelbagai implementasi yang tak standar dibuat untuk bermacam-macam masalah tertentu dimana GA digunakan sebagai meta-heuristic.

GA + Stuktur Data = Program Evolusi

7.5 Keuntungan Utama

Ada 3 keuntungan utama dalam mengaplikasikan GA pada masalah-masalah optimasi:

1. GA tak banyak memerlukan kebutuhan matematis mengenai masalah optimasi
2. Kemudahan dan kenyamanan dan pada operator-operator evolusi membuat GA sangat efektif dalam melakukan pencarian global (dalam probabilitas)
3. GA menyediakan banyak fleksibelitas untuk digabungkan dengan metode heuristic yang tergantung domain, untuk membuat implementasi yang efisien pada masalah-masalah khusus.

Kamus GA

Terdapat beberapa istilah dalam GA yang untuk lebih baiknya akan dijelaskan dalam tabel 7.1 di bawah ini.

Tabel 7.1 Istilah-istilah dalam GA dan penjelasannya

Genetic Algorithms	Explanation
Chromosome (string, individual)	Solution (coding)
Genes (bits)	Part of solution
Locus	Position of gene
Alleles	Values of gene
Phenotype	Decoded solution
Genotype	Encoded solution

7.6 Program Algoritma Genetika Sederhana

Masalah yang akan dipecahkan dalam program yang menggunakan metode algoritma genetika sederhana (simple genetic algorithm) ini adalah masalah optimasi (yaitu nilai maksimal) nilai dari suatu variabel yang terlibat pada suatu fungsi.

Diberikan fungsi di bawah ini:

Maksimalkan

$f(x_1, x_2) = 21.5 + x_1 \sin(4\pi x_1) + x_2 \sin(20\pi x_2)$

$-3.0 <= x_1 <= 12.1$

$4.1 <= x_2 <= 5.8$

Domain dari variabel x_j adalah $[a_j, b_j]$ dan presisinya adalah 4 tempat di belakang koma, maka range dari setiap variabel dapat dibagi dalam $(b_j - a_j) \times 10^4$ unit.

Bit yang dibutuhkan (dilambangkan dengan m_j) untuk variabel tadi dapat dihitung dari:

$2^{m_j - 1} < (b_j - a_j) \times 10^4 <= 2^{m_j} - 1$

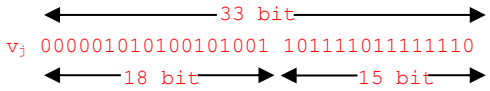
Variabel keputusan (decision variable) $x_j = a_j + \text{decimal}(\text{substring}_j) \times \frac{b_j - a_j}{2^{m_j} - 1}$

Dimana decimal (substring$_j$) merepresentasikan nilai desimal dari substring$_j$ untuk x_j.

Dimisalkan presisinya adalah 4 tempat di belakang koma. Bit yang dibutuhkan untuk variabel x_1 dan x_2 adalah:

$(12.1 - (-3.0)) \times 10{,}000 = 151{,}000$
$2^{17} < 151{,}000 <= 2^{18}$, $m_1 = 18$
$(5.8 - 4.1) \times 10{,}000 = 17{,}000$
$2^{14} < 17{,}000 <= 2^{15}$, $m_2 = 15$

$m = m_1 + m_2 = 18 + 15 = 33$ → Panjang total kromosom adalah 33 bit yang direpresentasikan sbb:



Nilai yang bisa didapatkan untuk variabel x_1 dan x_2 adalah sbb:

	Biner	Desimal
x_1	000001010100101001	5417
x_2	101111011111110	24318

$$x_1 = -3.0 + 5417 \times \frac{12.1-(-3.0)}{2^{18}-1} = -2.687969$$

$$x_2 = 4.1 + 24318 \times \frac{5.8-4.1}{2^{15}-1} = 5.361653$$

Prosedur-prosedur yang ada dalam program

Terdapat beberapa prosedur dalam GA seperti di bawah ini:

Initial Population. Generate secara random (*pop_size* misalnya 10 berarti generate 10 populasi awal), misal sbb:

v_1=[00000101010010100110111011111110] → nilai desimal v_1=[x_1,x_2]= [-2.687969,5.361653]
v_2=[00111010111001100000001010101001000] → nilai desimal v_2=[x_1,x_2]= [0.474101,4.170144]
v_3=[11100011100000100001010101001000110] → nilai desimal v_3=[x_1,x_2]= [10.419457,4.661461]
v_4=[10011011010010110100000000101110001] → nilai desimal v_4=[x_1,x_2]= [6.159951,4.109598]

v_5=[00001011110110001000111000111010000]→ nilai desimal v_5=[x_1,x_2]= [-2.301286,4.477282]
v_6=[11111010101101100000000101100111001]→ nilai desimal v_6=[x_1,x_2]= [11.788084,4.174346]
v_7=[11010000100111110001001100111011101]→ nilai desimal v_7=[x_1,x_2]= [9.342067,5.121702]
v_8=[00101101010000110001011001100111100]→ nilai desimal v_8=[x_1,x_2]= [-0.330256,4.694977]
v_9=[11111000101110110001110100011110 1]→ nilai desimal v_9=[x_1,x_2]= [11.671267,4.873501]
v_{10}=[11110100111010101000001010110101 0]→ nilai desimal v_{10}=[x_1,x_2]=[11.446273,4.171908]

Evaluation. Dalam proses ini evaluasi nilai *fitness* dari kromosom yang terdiri dari 3 langkah:

Konversikan genetipe kromosom ke fenotipe-nya, artinya mengkonversikan string biner ke nilai real relatif dari $x^k = (x_1^k, x_2^k)$, dengan k = 1,2, ..., *pop_size*.

Evaluasi fungsi obyektifnya: $f(x^k)$.

Konversikan nilai dari fungsi obyektifnya ke *fitness*. Untuk kasus memaksimalisasikan, *fitness*-nya adalah sama dengan nilai fungsi obyektifnya: $eval(v_k) = f(x^k)$, k = 1,2, ..., *pop_size*.

eval(v_1) = f(-2.687969,5.361653) = 19.805119

eval(v_2) = f(0.474101,4.170144) = 17.370896

eval(v_3) = f(10.419457,4.661461) = 9.590546

eval(v_4) = f(6.159951,4.109598) = 29.406122

eval(v_5) = f(-2.301286,4.477282) = 15.686091

eval(v_6) = f(11.788084,4.174346) = 11.900541

eval(v_7) = f(9.342067,5.121702) = 17.958717

eval(v_8) = f(-0.330256,4.694977) = 19.763190

eval(v_9) = f(11.671267,4.873501) = 26.401669

eval(v_{10}) = f(11.446273,4.171908) = 10.252480

Terlihat disitu bahwa kromosom v_4 adalah kromosom terkuat dan kromosom v_3 adalah yang terlemah.

Selection. Dalam kebanyakan kasus, pendekatan *roulette wheel* (roda rolet) sering digunakan untuk prosedur seleksi; ini dimiliki oleh seleksi *fitness-proportional* dan dapat memilih populasi baru yang berhubungan dengan distribusi probabilitas pada nilai fitness-nya. Roda rolet ini dapat dijelaskan sbb:

Hitung nilai fitness eval(v_k) untuk setiap kromosom v_k:

eval(v_k) = f(x), k = 1,2, ..., *pop_size*.

Hitung total fitness untuk populasi:

$$F = \sum_{k=1}^{pop\_size} eval(v_k)$$

Hitung probabilitas seleksi p_k untuk setiap kromosom v_k:

$$p_k = \frac{eval(v_k)}{F}, \quad k = 1,2, ..., pop\_size.$$

Hitung probabilitas kumulatif q_k untuk setiap kromosom v_k:

$$q_k = \sum_{j=1}^{k} p_j \ , \quad k = 1,2, ..., pop\_size$$

Prosedur *selection* dimulai dengan memutar roda rolet sebanyak *pop_size* kali; setiap waktu satu kromosom dipilih untuk populasi baru selanjutnya, dengan langkah sbb:

Generate bilangan random r dengan range [0,1].

Jika r <= q_1, maka pilih kromosom pertama: v_1; jika tidak, pilih kromosom ke-k: v_k(2 <= k <= *pop_size*) sehingga q_{k-1} < r <= q_k.

Total *fitness* F dari populasi adalah: $F = \sum_{k=1}^{10} eval(v_k) = 178.135372$

Probabilitas seleksi p_k untuk setiap kromosom v_k(k = 1, ..., 10) adalah sbb:

```
p1 = 0.111180   p2 = 0.097515   p3 = 0.053839   p4 = 0.165077   p5 = 0.088057
p6 = 0.066806   p7 = 0.100815   p8 = 0.110945   p9 = 0.148211   p10 = 0.057554
```

Probablitas kumulatif q_k untuk setiap kromosom v_k(k = 1, ..., 10) adalah sbb:

```
q1 = 0.111180   q2 = 0.208695   q3 = 0.262534   q4 = 0.427611   q5 = 0.515668
q6 = 0.582475   q7 = 0.683290   q8 = 0.794234   q9 = 0.942446   q10 = 1.000000
```

Sekarang marilah kita coba memutar roda rolet sebanyak 10 kali, dan setiap waktu kita memilih 1 kromosom untuk populasi baru selanjutnya. Diasumsikan misalnya 10 bilangan random dengan range [0,1] adalah sbb:

```
0.301431   0.322062   0.766503   0.881893   0.350871
0.583392   0.177618   0.343242   0.032685   0.197577
```

Bilangan pertama r_1: 0.301431 lebih besar daripada q_3 dan lebih kecil daripada q_4, yang

berarti bahwa kromosom q_4 dipilih untuk populasi baru berikutnya; bilangan kedua r_2: 0.322062 lebih besar daripada q_3 dan lebih kecil daripada q_4, yang berarti bahwa kromosom q_4 lagi-lagi dipilih untuk populasi baru berikutnya; dan demikian seterusnya. Akhirnya populasi baru yang terbentuk adalah sebagai berikut:

v'_1 = [10011011010010110100000010111001] (v_4)

v'_2 = [10011011010010110100000010111001] (v_4)

v'_3 = [00101101010000110001011001100] (v_8)

v'_4 = [11111000101110110001110100011101] (v_9)

v'_5 = [10011011010010110100000010111001] (v_4)

v'_6 = [11010001001111100010011001101101] (v_7)

v'_7 = [00111010111001100000010101001000] (v_2)

v'_8 = [10011011010010110100000010111001] (v_4)

v'_9 = [00000101010010100110111011111110] (v_1)

v'_{10}= [00111010111001100000010101001000] (v_2)

Crossover. Digunakan sebagai metode pemotongan kromosom, yang secara random memilih titik potong pada kromosom dan menggantinya dengan bagian kanan dari 2 kromosom induk (*parent*) untuk menghasilkan kromosom anak (*offspring*). Pada contoh di bawah ini terdapat 2 kromosom induk, dan titik potongnya secara random dipilih pada posisi *gen* ke-17:

v_1 = [1001101101001011**0** 1000000010111001]

v_2 = [0010110101000011**0** 0010110011001100]

Didapatkan hasil *Offspring* dengan menggantikan bagian kanan dari kromosom induk sebagai berikut:

v'_1 = [1001101101001011**0** 0010110011001100]

v'_2 = [0010110101000011**0** 1000000010111001]

Probabilitas crossover di set sebagai $\mathbf{p_r}$ = 0.25, maka kita dapat berharap bahwa sekitar

25% dari kromosom yang ada akan mengalami crossover. Crossover dapat dilakukan dengan langkah sbb:

```
procedure: Crossover
begin
     k ← 0;
     while (k <= 10) do
          rk ← random number from [0,1];
          if (rk < 0.25) then
               select vk as one parent of crossover;
          end
          k ← k + 1;
     end
end
```

Diasumsikan bahwa urutan bilangan random-nya adalah sbb:

```
0.625721  0.266823  0.288644  0.295114  0.163274
0.567461  0.085940  0.392865  0.770714  0.548656
```

Dari situ bisa kita lihat bahwa kromosom v'_5 dan v'_7 akan dipilih untuk dilakukan crossover padanya. Kita generate bilangan integer *pos* secara random dengan range [1,32] (sebab 33 adalah panjang dari kromosom) sebagai titik potong atau dengan kata lain adalah posisi titik crossover. Diasumsikan bahwa bilangan *pos* yang dihasilkan sama dengan 1, dua kromosom dipotong setelah bit ke-1, dan offspring dihasilkan dengan menggantikan bagian kanan dari kromosom yang bersangkutan sbb:

v'_5 = [10011011010010110100000010111001]

v'_7 = [00111010111001100000010101001000]

menjadi:

v'_5 = [10111010111001100000010101001000]

v'_7 = [00011011010010110100000010111001]

Mutation. Mutasi mengubah satu atau lebih *gen* dengan probabilitas sama dengan angka mutasi. Diasumsikan bahwa gen ke-18 dari kromosom v_1 dipilih untuk bermutasi. Karena gen di posisi itu (posisi ke-18) bernilai 1, maka setelah bermutasi nilainya menjadi 0. Maka setelah bermutasi, kromosom menjadi:

v_1 = [10011011010010110**1**000000010111001]

v_1 = [10011011010010110**0**000000010111001]

Probabilitas mutasi diset sebagai $\mathbf{p_m}$ = 0.01, sehingga diharapkan bahwa sekitar 1% total bit dari populasi akan bermutasi. Ada m x *pop_size* = 33 x 10 = 330 bit dalam keseluruhan populasi; diharapkan 3.3 mutasi terjadi setiap generasi. Setiap bit memiliki kesempatan yang sama untuk bermutasi. Maka kita perlu meng-generate serangkaian bilangan random r_k (k= 1, ..., 330) dengan range [0,1]. Dimisalkan bahwa gen-gen berikut ini akan bermutasi seperti digambarkan dalam tabel 7.2 di bawah ini.

Tabel 7.2 Gen yang akan bermutasi

bit_pos	chrom_num	bit_no	random_num
105	4	6	0.009857
164	5	32	0.003113
199	7	1	0.000946
329	10	32	0.001282

Setelah mutasi, didapatkan populasi baru sebagai generasi berikutnya, sebagai berikut:

v'1=[10011011010010110100000010111001]→ nilai desimal f(6.159951,4.109598) = 29.406122
v'2=[10011011010010110100000010111001]→ nilai desimal f(6.159951,4.109598) = 29.406122
v'3=[00101101010000110001011001100100]→ nilai desimal f(-0.330256,4.694977) = 19.763190
v'4=[11111100101110110001111010001111101]→ nilai desimal f(11.907206,4.873501) = 5.702781
v'5=[10111010111001100000001010100101010]→ nilai desimal f(8.024130,4.170248) = 19.91025
v'6=[11010001001111100010011001110110101]→ nilai desimal f(9.342067,5.121702) = 17.958717
v'7=[10011011010010110100000010111001]→ nilai desimal f(6.159951,4.109598) = 29.406122
v'8=[10011011010010110100000010111001]→ nilai desimal f(6.159951,4.109598) = 29.406122
v'9=[00000101010010100110111011111110]→ nilai desimal f(-2.687969,5.361653) = 19.805119
v'10=[00111010111001100000001010100101010]→ nilai desimal f(0.474101,4.170248) = 17.370896

Baru saja tadi kita sudah menyelesaikan satu iterasi dari algoritma genetika (GA). Program nantinya akan dihentikan setelah 1000 iterasi (1000 generasi). Didapatkan kromosom terbaik pada generasi ke-419 sbb:

$v^* = (11111000000011100011101001010110)$

$eval(v^*) = f(11.631407,5.724824) = 38.818208$

$x^*_1 = 11.631407$

$x^*_2 = 5.724824$

$f(x^*_1,x^*_2) = 38.818208$

Dari sini dapat dibuat program yang mengimplementasikan hal-hal diatas, dan sebagai masukannya, minimal adalah jumlah populasi awal **pop_size**, probabilitas crossover $\mathbf{p_r}$, probabilitas mutation $\mathbf{p_m}$, banyak iterasi, kromosom terbaik dan nilai-nilai yang ada padanya.

Bab 8 Ripple Down Rules

Hal yang dibahas dalam bab ini adalah mengenai Ripple Down Rules (RDR), khususnya RDR untuk pengklasifikasian tunggal. Pustaka yang digunakan adalah dari Compton, P. J. dan Jansen, R. [Comp89]; P. Preston, G. Edwards dan P. Compton [Pres93a]; P. Preston, G. Edwards dan P. Compton [Pres93b]; P. Compton dan D. Richards [Comp00]; dan P. Compton, G. Edwards, B. Kang, L. Lazarus, R. Malor, T. Menzies, P. Preston, A. Srinivasan dan S. Sammut [Comp91].

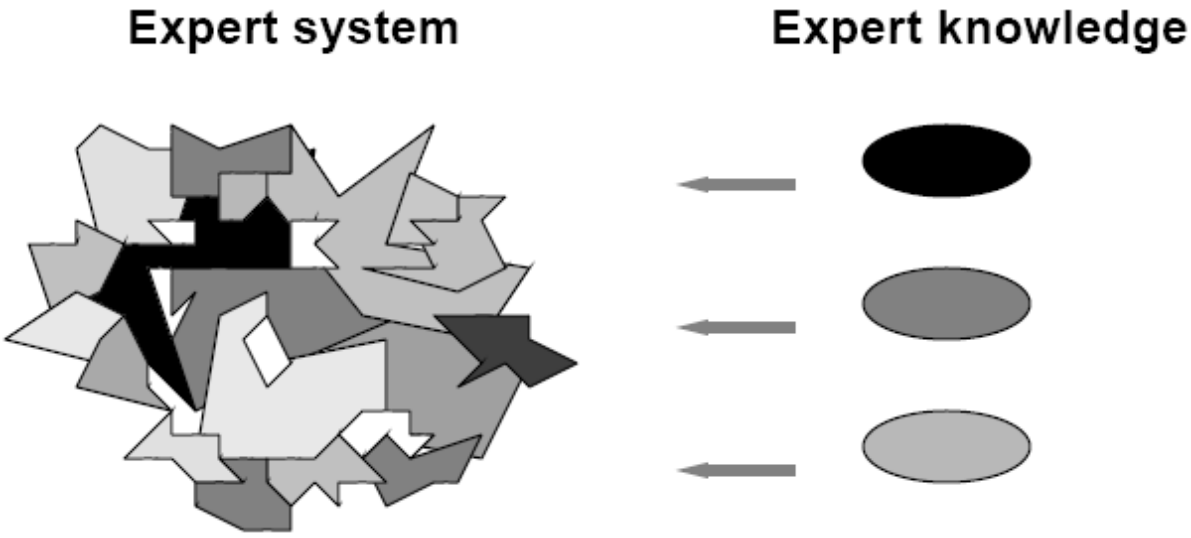
Ripple Down Rules (RDR) bertujuan untuk mendapatkan pengembangan yang sederhana dan bertahap dari suatu Sistem Berbasis Pengetahuan (SBP) pada saat SBP tadi sedang digunakan, sehingga seiring dengan berjalannya waktu seorang pakar dapat melakukan evolusi pada SBP yang canggih ini sebagai tugas kecil tambahan dari tugas-tugas normal mereka.

RDR dipandang sukses dalam pengembangan klasifikasi SBP. Lebih lanjut, RDR juga dikembangkan untuk pencarian konfigurasi dan heuristic dan tugas-tugas lainnya.

8.1 Representasi Pengetahuan dan RDR

RDR adalah struktur sederhana yang ditujukan untuk menangkap, paling tidak sebagian, konteks dimana suatu pengetahuan didapatkan dari seorang pakar.

Tentu saja terdapat perbedaan yang amat besar pada representasi pengetahuan oleh seorang pakar dan oleh sistem pakar (contohnya RDR), seperti terlihat pada gambar 8.1.

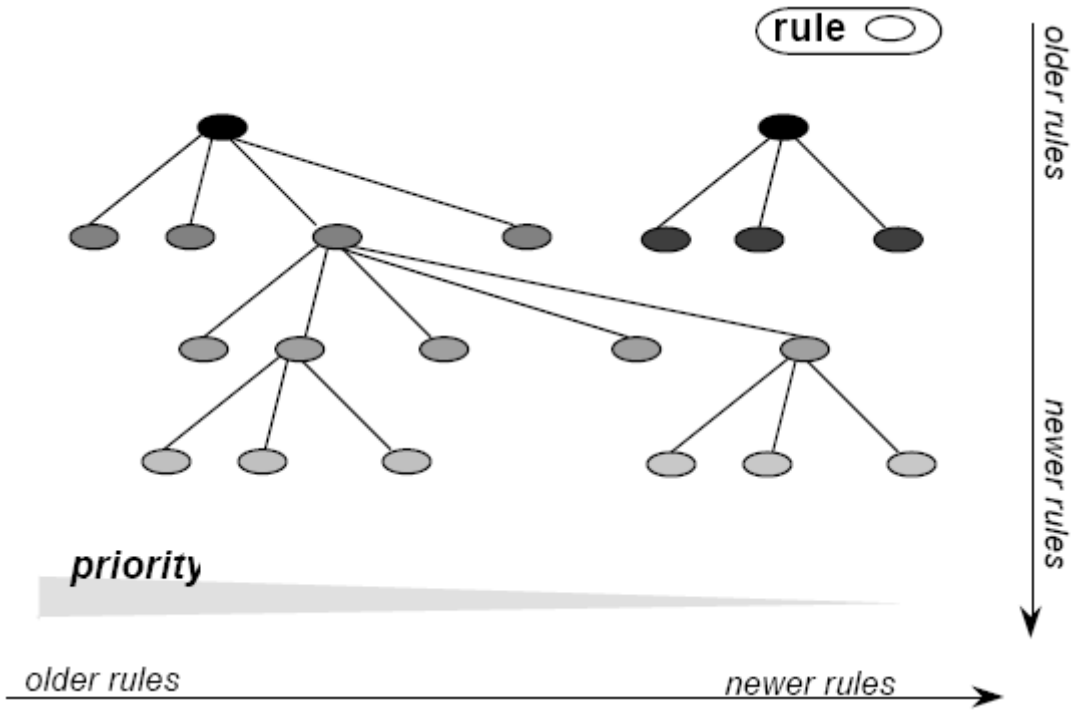


Gambar 8.1 Representasi pengetahuan pada pakar dan dalam sistem pakar

Diagram pada gambar 8.1 di atas menjelaskan bagaimana rule yang diekspresikan oleh seorang pakar harus dimanipulasi dan diubah untuk memastikan bahwa mereka tak saling bertumpang tindih antara satu rule dengan rule-rule yang lainnya.

Tentu saja implementasi dari pengetahuan dalam setiap sistem pakar mungkin tidak sama antara satu sistem pakar dengan yang lainnya. Hanya saja semua perbedaan yang mungkin ada tak akan meninggalkan tujuan dari tujuan representasi awalnya yaitu agar tidak terjadi tumpang tindih antar pengetahuan yang menyebabkan kerancuan dan kesulitan dalam pemanfaatan pengetahuan lebih lanjut.

Dalam gambar 8.2 berikut ini disajikan penyederhanaan representasi tree (pohon) pada Ripple Down Rules.



Gambar 8.2 Penyederhanaan representasi tree pada RDR

Pada gambar 8.2 di atas, terlihat 2 elips hitam pada puncak tree yang disebut dengan dua rule LAST_FIRED(0), sehingga prekondisi dari satu ataupun dua rule tadi untuk dieksekusi adalah tak ada satu pun rule lain yang dieksekusi.

Rule di bagian kiri, yang lebih dulu dianggap sebagai yang pertama. Jika rule ini tak bisa dieksekusi, rule LAST_FIRED(0) selanjutnya yang paling tua yang akan dipertimbangkan lebih lanjut.

Jika suatu rule LAST_FIRED(0) dieksekusi, maka hanya rule-rule yang terkoneksi dengan

rule tersebut pada level berikutnya yang menjadi kandidat untuk dieksekusi.

Sekali lagi, yang menjadi pertimbangan suatu rule dieksekusi adalah: satu demi satu, dari kiri ke kanan, dari yang paling tua ke yang paling muda. Setiap dari hal-hal tadi ditambahkan ke knowledge base kita, disebabkan parent-nya salah menginterpretasikan suatu case, tetapi tak ada satu pun rule lebih lanjut yang bisa dieksekusi. Setiap dari hal-hal ini melibatkan kondisi LAST_FIRED(parent). Sekali saja satu dari rule anak-anaknya ditambahkan maka hanya rule-rule yang terkoneksi padanya yang menjadi kandidat untuk dieksekusi.

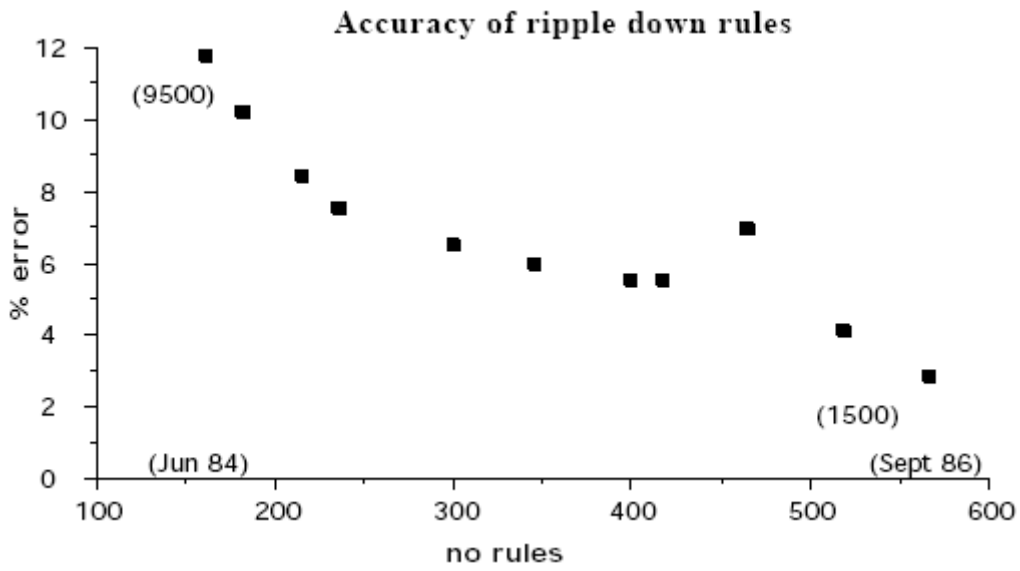
Fitur utama dari RDR adalah ia dapat menambahkan pengetahuan ke knowledge base jauh lebih cepat dibandingkan SBP konvensional sebab rule-rule ditambahkan seperti apa adanya tanpa modifikasi. Kedua, karena RDR digunakan hanya dalam konteksnya saja maka RDR amat jauh terkena dampak dari korupsi yang mungkin terjadi dalam knowledge base.

Dalam kenyataan sehari-hari, RDR telah digunakan untuk pelbagai bidang, misalnya untuk menggantikan GARVAN-ES1, sebuah sistem pakar medis yang telah ada dengan konstruksi RDR.

Contoh lain aplikasi RDR adalah PEIRS (Pathology Expert Interpretive Reporting System). PEIRS adalah sistem pakar medis yang besar untuk menginterpretasikan laporan-laporan penyakit-penyakit secara kimiawi (chemical pathology) pada Rumah Sakit St. Vincent di Sydney, Australia.

Solusi RDR untuk SISYPHUS-I adalah satu lagi contoh dari aplikasi RDR yang telah berhasil diterapkan. SISYPHUS-I adalah program untuk mengalokasikan ruang dalam komunitas graf konseptual.

Pada interpretasi ulang RDR atas GARVAN-ES1, ternyata hasilnya memuaskan, mendekati aslinya dengan gambaran kesalahan seperti pada gambar 8.3 di bawah ini.



Gambar 8.3 Persentase kesalahan interpretasi RDR atas GARVAN-ES1

Gambar 8.3 di atas mengilustrasikan persentase kesalahan dalam interpretasi yang dihasilkan oleh RDR saat dibandingkan dengan interpretasi aslinya yang dihasilkan oleh GARVAN-ES1, sebagai fungsi pertumbuhan dalam hal ukuran rule base. Jumlah test case berkurang selama pertumbuhan knowledge base dari 9500 ke 1500, dimana interpretasi yang tidak benar/salah dari test case digunakan dasar dari penambahan pengetahuan lebih lanjut. Tak peduli apakah case diinterpretasikan secara benar atau tidak, sekali saja suatu case menjadi kandidat untuk perubahan rule, ia tidak boleh digunakan kembali untuk memastikan bahwa test tersebut valid. Tanggal di atas mengindikasikan pada periode mana arsip case diambil.

Dalam implementasinya pada data base relasional, RDR ternyata mudah dilakukan. Ini digambarkan dalam gambar 8.4 di bawah ini.

PRODUCTION RULE

RULE(42)
IF FTI is high
and T3 is high
and TSH is undetectable
and not on_t4

THEN **thyrotoxic**

ELEMENT RELATIONSHIP TABLE

owner	relationship	member
-------	-------	-------
-------	-------	-------
RULE_42	presence	FTI_high
RULE_42	presence	T3_high
RULE_42	presence	TSH_undetect
RULE_42	absence	on_t4
RULE_42	outcome	thyrotoxic
-------	-------	-------
-------	-------	-------

Gambar 8.4 Representasi RDR dalam data base relasional

Production rule (di bagian atas) dapat direpresentasikan sebagai kumpulan tuple (record) dalam tabel relasional, dimana rule sebagai sebuah objek memiliki hubungan presence (ada) dan absence (tak ada) dengan pelbagai variasi fakta dan hubungan outcome (hasil) dengan suatu interpretasi. Ini adalah representasi penyederhanaan karena tipe objek yang lain tak terindikasikan/ada disini.

PEIRS (Pathology Expert Interpretive Reporting System)

Demonstrasi utama dari RDR adalah PEIRS dimana ia adalah sistem pakar medis yang besar untuk menginterpretasikan laporan-laporan penyakit-penyakit secara kimiawi pada Rumah Sakit St. Vincent di Sydney, Australia.

Sistem ini sekarang memiliki lebih dari 2300 rules dan mencakup kebanyakan dari penyakit secara kimiawi (chemical pathology), membuatnya menjadi sistem pakar medis

yang sangat besar.

Namun demikian penambahan pengetahuan seluruhnya dilakukan oleh para pakar tanpa bantuan rekayasa pengetahuan (knowledge engineering), juga tanpa kemampuan rekayasa pengetahuan atau pun kemampuan pemrograman.

Sistem ini diletakkan dalam suatu rutin yang menggunakan 200 rule, dengan semua rule tambahan berikutnya ke sistem dalam penggunaan rutin secara aktual. Sehingga 200 rule awal ini ditambahkan ke sistem secara off-line sebagai awalan dari sistem.

Pada gambar 8.5 di bawah ini disajikan contoh case yang ditangani oleh PEIRS. Interpretasi dari data secara langsung mengikuti data. Abnormalitas yang lain dalam data ini tak tercakup dalam knowledge base.

09-Jun-92 ST. VINCENT'S HOSPITAL 12:58 PM
DEPARTMENT OF CHEMICAL PATHOLOGY

Patient ..... : ISPEAD,U.P., Male, 20 years

M.R.N. ..... : 29-03-44 Ward : CW13

Test	Range	Units	BLOOD 847509 13:56 03 Jun 1992	BLOOD 847674 09:14 04 Jun 1992	BLOOD 847693 07:43 05 Jun 1992	BLOOD 847945 10:12 07 Jun 1992	BLOOD 848203 11:25 09 Jun 1992
Sodium	137-146	mmol/L	138	137	....	137	....
Potassium	3.5-5.0	mmol/L	....	4.4	3.9	....	3.6
Chloride	95-105	mmol/L	....	105	103	....	103
Bicarb.	24-31	mmol/L	....	17	17	....	16
Urea	3.0-8.5	mmol/L	....	11.9	11.4	....	6.8
Creatinine	0.04-0.10	mmol/L	....	0.10	0.12	....	0.13
Inorg. Phos.	0.70-1.40	mmol/L	....	2.37	....	2.02	....
Magnesium	0.85-1.05	mmol/L	....	0.96	....	....	....
Calcium	2.10-2.60	mmol/L	....	2.39	....	2.52	....
pH	7.35-7.45		7.49	....	....	....	....
PCO2	32-45	mmHg	25	....	....	....	....
PO2	75-105	mmHg	107	....	....	....	....
BIC	24-31	mmol/L	19	....	....	....	....
BXS	-3-+3	mmol/L	-1	....	....	....	....
PTI	65-155	mmol/L	....	105	....	....	....
TT4	60-150	mmol/L	....	83	....	....	....
TU	0.60-1.25	mmol/L	....	0.79	....	....	....
TSH	0.2-6.0	mmol/L	....	13.9	....	....	....
T3	1.2-2.8	mmol/L	....	1.3	....	....	....

Results consistent with:
Compensated hypothyroidism.
Suggest:
Thyroid antibody studies.

Gambar 8.5 Case yang ditangani oleh sistem

Sedangkan pada gambar 8.6 di bawah ini disajikan daftar (list) perbedaan-perbedaan yang disajikan kepada seorang pakar sehingga ia dapat memilih kondisi rule. Hal-hal ini perbedaan untuk analisis tunggal.

```
find differences for:

tests:
1: BLOOD_OST     2: BLOOD_HLH     3: BLOOD_HFSH    4: BLOOD_T3      5: BLOOD_TSH
6: BLOOD_TU      7: BLOOD_TT4     8: BLOOD_FTI     9: BLOOD_GGT     10: BLOOD_ALT
11: BLOOD_AP     12: BLOOD_BILI   13: BLOOD_TP     14: BLOOD_ALB    15: BLOOD_CA
16: BLOOD_CREA   17: BLOOD_UREA   18: BLOOD_CO2    19: BLOOD_CL     20: BLOOD_K
21: BLOOD_NA     22: TIME         23: HOSPITAL     24: WARD         25: MRN
26: AGE 27: SEX
select an option (ctrl-z to quit): 7


test: BLOOD_TT4

conditions possible:
1:         AVG(BLOOD_TT4) is NORMAL
2:         NOT AVG(BLOOD_TT4) is INCREASING
3:         NOT AVG(BLOOD_TT4) is STEADY
4:         NOT AVG(BLOOD_TT4) is DECREASING
5:         AVG(BLOOD_TT4) > -100000.00
6:         AVG(BLOOD_TT4) < 100000.00
7:         AVG(BLOOD_TT4) = 147.67
8:         NETCH(BLOOD_TT4) is NEGATIVE
9:         NOT NETCH(BLOOD_TT4) is INCREASING
10:        NOT NETCH(BLOOD_TT4) is STEADY
11:        NOT NETCH(BLOOD_TT4) is DECREASING
12:        NETCH(BLOOD_TT4) > -100000.00
13:        NETCH(BLOOD_TT4) < 100000.00
14:        NETCH(BLOOD_TT4) = -199.00
15:        CURR(BLOOD_TT4) is NORMAL
16:        NOT CURR(BLOOD_TT4) is INCREASING
17:        NOT CURR(BLOOD_TT4) is STEADY
18:        NOT CURR(BLOOD_TT4) is DECREASING
19:        CURR(BLOOD_TT4) > -100000.00
20:        CURR(BLOOD_TT4) < 100000.00
21:        CURR(BLOOD_TT4) = 68.00
22:        MAX(BLOOD_TT4) is HIGH
23:        NOT MAX(BLOOD_TT4) is INCREASING
24:        NOT MAX(BLOOD_TT4) is STEADY
25:        NOT MAX(BLOOD_TT4) is DECREASING
26:        MAX(BLOOD_TT4) > -100000.00
27:        MAX(BLOOD_TT4) < 100000.00
28:        MAX(BLOOD_TT4) = 267.00
29:        MIN(BLOOD_TT4) is NORMAL
30:        NOT MIN(BLOOD_TT4) is INCREASING
31:        NOT MIN(BLOOD_TT4) is STEADY
32:        NOT MIN(BLOOD_TT4) is DECREASING
33:        MIN(BLOOD_TT4) > -100000.00
34:        MIN(BLOOD_TT4) < 100000.00
35:        MIN(BLOOD_TT4) = 68.00

select conditions (ctrl-z) to quit:
>
```

Gambar 8.6 Daftar perbedaan-perbedaan untuk pemilihan kondisi

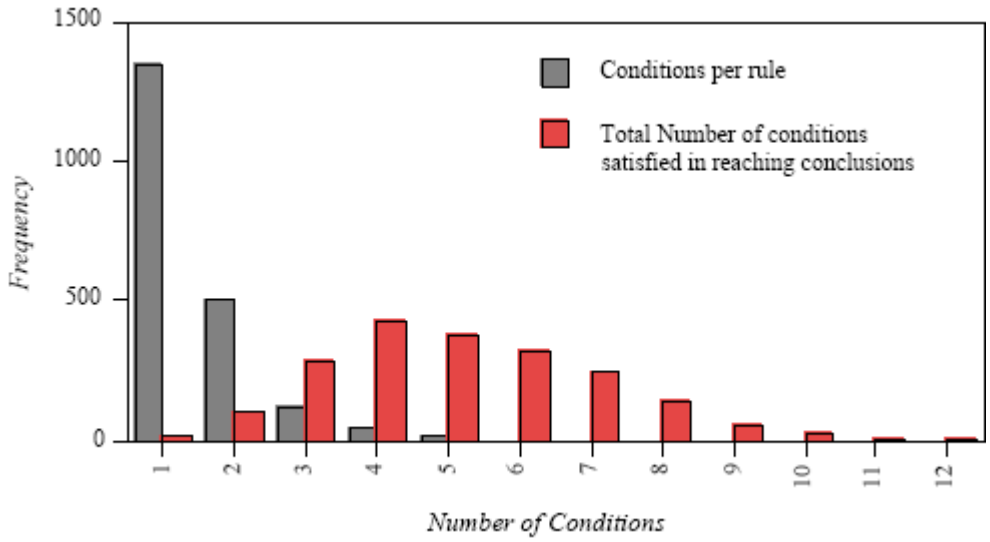
Pada gambar 8.7 di bawah ini disajikan contoh rule yang menunjukkan fungsi bawaan yang digunakan. Perlu dicatat dalam RULE 369 suatu ekspresi matematika yang dimasukkan oleh seorang pakar. Fungsi VAL mengembalikan nilai untuk parameter pertama pada waktu yang ditentukan oleh parameter kedua; dalam rule 452 ia

mengembalikan nilai dari BLOOD_RLU pada waktu minimum BLOOD_ST. TDIFF mengembalikan perbedaan waktu antara 2 titik waktu yang ditentukan. Perhatikan dalam RULE 369 bahwa urutan relasi ada tak hanya diantara nilai numerik tetapi diantara klasifikasi primer dari data.

```
RULE   369 IF   ((CURR(AGE)*(-0.27))   +   94)   -   CURR(BLOOD_PO2)   >   0.00   &
           MIN(BLOOD_BIC) < NORMAL THEN

RULE   379 IF    CURR(BLOOD_PH)  is  NORMAL  &  CURR(BLOOD_PCO2)  is  LOW  &
           CURR(BLOOD_PO2) is HIGH & CURR(BLOOD_BIC) is LOW  THEN

RULE 410   IF TDIFF(CURR(BLOOD_TSH),MAX(BLOOD_TSH) < 30.00 THEN

RULE 424   IF AVG(BLOOD_PO2) < 60.0 THEN

RULE   452 IF  VAL(BLOOD_RGLU,  MIN(BLOOD_ST))  <  5.80  &  CURR(BLOOD_DYNT)  is
           "GT100" & MAX(BLOOD_RLU) < 10.50  THEN
```

Gambar 8.7 Contoh rule yang menampilkan fungsi bawaan yang digunakan

Kompleksitas dari rule dalam hubungannya dengan kondisi-kondisi dalam rule-rule individual dan jumlah total dari kondisi-kondisi dalam rule yang mencapai konklusi diilustrasikan dalam gambar 8.8 di berikut ini.



Gambar 8.8 Ilustrasi kompleksitas rule

8.2 Pengetahuan dalam Evolusi di RDR

RDR dikembangkan untuk menangani permasalahan-permasalahan dimana para pakar tak pernah memerikan penjelasan secara luas untuk penyelesaian masalah yang mereka ambil. Para pakar lebih suka untuk membenarkan saja bahwa kesimpulan mereka adalah benar dan pembenaran tersebut dilakukan untuk dan dibentuk oleh konteks yang ada

saat kesimpulan tadi diambil.

Fitur-fitur kritis dari RDR sepanjang evolusinya dapat dituliskan seperti di bawah ini:

- Pengetahuan ditambahkan ke sistem untuk menangani case (kasus) yang spesifik. Ini adalah case dimana sistem akan melakukan kesalahan.
- Sistem secara bertahap berevolusi seiring dengan waktu, pada saat ia digunakan.
- Sistemlah, bukannya knowledge engineer yang mengorganisasi bagaimana pengetahuan ditambahkan.
- Ia memvalidasi sembarang akuisisi pengetahuan yang dilakukan sehingga ia menyediakan penambahan bertahap ke knowledge system (sistem pengetahuan) dan tak menurunkan kualitas pengetahuan sebelumnya.
- Untuk menambahkan pengetahuan baru, para hanya perlu mengidentifikasi fitur dalam suatu case yang dapat membedakannya dengan case yang lain yang didapat oleh sistem.

8.3 Fitur Kunci Suksesnya RDR

Fitur-fitur kunci dari RDR yang menyebabkannya sukses dalam aplikasi komersial adalah seperti di bawah ini:

- Suatu task (tugas) diidentifikasikan memiliki karakteristik:
  - Adanya sistem informasi, yang secara berulang menghasilkan data untuk case yang sedang diproses. (case data yang asli tidaklah penting)
  - Adanya kebutuhan interpretasi seorang pakar dari case data
  - Adalah normal dan nyaman untuk para pakar dalam memonitor output case data. (Sebagai contoh dalam patologi – ilmu penyakit, adalah biasa bagai seorang patolog atau ilmuwan lab senior untuk memonitor report/laporan. Penambahan rule (untuk mengkoreksi kesalahan) dalam rule tadi, membutuhkan waktu 15 menit perhari, dan tidak mengganngu tugas normal pakar tadi)
- Pemodelan data dan rekayasa perangkat lunak yang cukup harus dilakukan untuk menjadikan case data dapat dilewatkan/diberikan dari sistem informasi ke SBP dan untuk direpresentasikan ke pakar dengan cara yang layak buat mereka dalam rangka mengidentifikasi fitur dan menambah rule.
- Sistem RDR selanjutnya diletakkan dalam rutin yang digunakan.

- Jika dalam monitoring, suatu case diidentifikasi dimana output SBP adalah tidak benar dengan suatu cara, sang pakar memasukkan konklusi yang benar untuk bagian dari output tadi dan case tadi ditandai untuk updating rule.
- SBP selanjutnya telah diupdate. Namun demikian hanya satu komponen tunggal dari output, dan hanya konklusi rule tunggal yang telah dikoreksi pada satu waktu.
- Jika rule telah diupdate, sang pakar akan diperlihatkan tampilan dari suatu case, mungkin juga case yang telah disimpan sebelumnya yang mana juga memperlihatkan perbedaan penting yang mungkin diantara case-case yang ada. Harus dicatat disini bahwa bantuan ini tidaklah sekritis pada saat pakar telah mengidentifikasi fitur dalam case dalam penentuan mana yang memerlukan koreksi.
- Seorang pakar memilih fitur-fitur yang diperlukan untuk membuat sebuah rule untuk menghilangkan case yang telah disimpan sebelumnya. (Sang pakar bisa juga memberikan konklusi pada case yang telah disimpan jika diperlukan. Ini juga dapat terjadi jika suatu sistem dikembangkan sedikit demi sedikit dan sistem tersebut menjadi layak untuk membuat tanggapan yang lebih lengkap untuk suatu case)
- Sistem menambahkan rule ke knowledge base, dengan cara dimana case yang sama akan selalu diproses dengan urutan rule yang sama.
- Input case ditambahkan ke data base pada case-case yang ada dan dihubungkan ke rule baru. Case-case tersimpan ini, yang menyebabkan terjadinya penambahan rule disebut dengan “cornerstone cases”.
- Suatu case mungkin saja dijalankan kembali dan lebih banyak komponen yang berubah atau bertambah sesuai dengan kebutuhan

Terdapat banyak sekali struktur RDR yang memenuhi persyaratan di atas. Ini termasuk binary tree (pohon biner), n-ary trees, multiple binary trees dan n-ary trees dimana terdapat inferensia yang berulang dan beberapa bentuk dari penyelesaian konflik. Namun demikian, fitur utama dari struktur-struktur tersebut adalah:

- Jika suatu koreksi dilakukan yang membuat suatu rule ditambahkan; rule-rule yang ada tidak berubah.
- Rule ini (yang baru saja ditambahkan) dihubungkan dengan rule yang memberikan konklusi yang salah

- Hanya satu konklusi yang diberikan oleh sembarang rule path (urutan dari rule-rule pengkoreksian yang terhubung); yaitu konklusi dari rule terakhir yang memenuhi dalam path tersebut.
- Sebagaimana dicatat di atas, jika suatu rule koreksi ditambahkan, ini harus mengeluarkan/tidak mengikutkan case-case lain yang layak memenuhi oleh rule di atasnya (parent rule) dan rule-rule lain dalam sistem.
- Inferensia diorganisasikan sedemikian sehingga cornerstone case yang sama akan selalu diproses dengan urutan yang sama meskipun ada penambahan lebih lanjut ke knowledge base

8.4 Inferensia dan Akuisisi Pengetahuan

RDR memiliki struktur inferensia seperti akan dijelaskan berikut ini.

Jika suatu case diberikan ke SBP:

1. Inferensia dimulai dengan rule pertama ditambahkan ke knowledge base. "Working tentative conclusion" (konklusi kerja sementara) mengacu pada di bawah ini dan diatur di awal/diinisialisasi sebagai null. Rule sekarang yang sedang diproses disebut dengan rule X.
2. Proseslah kumpulan rule-rule yang bersaudara diurutkan berdasarkan umur, dimulai dari yang paling tua. Rule sekarang yang sedang diproses disebut dengan rule X.

```
if      rule X dieksekusi
then    konklusi rule X menggantikan konklusi sementara
        if   rule X memiliki anak
        then ini akan dievaluasi berdasarkan umurnya (secara rekursif mengulang step 2)
if      tidak ada satu pun dari anak-anak rule X dieksekusi
then    konklusi rule X diaktifkan, membuatnya perubahan tetap pada case. Konklusi
        sementara direset ke konklusi null. Kontrol lalu diberikan pada saudara rule X
        berikutnya yang tertua, atau jika tidak ada saudara yang lebih muda, kembali ke
        parent rule-nya.
```

3. Inferensia lalu diulang di sepanjang knowledge base sampai tak ada perubahan lebih lanjut dapat dibuat pada case itu. Terdapat banyak variasi bagaimana inferensia berulang ini dilakukan.

Walaupun struktur inferensia ini kelihatannya mirip seperti dalam inferensia konvensional, ia memiliki perbedaan yang signifikan. Tak ada strategi penyelesaian konflik yang dibutuhkan: rule-rule diproses dalam urutan yang tegas dan sekali konklusi final pada sembarang path yang sedang diperbaiki tercapai, maka secepatnya ia ditambahkan ke case dan tak dapat dibatalkan pada tahap inferensia berikutnya. Satu-satunya cara penambahan pada case tadi dapat dihapuskan atau diubah adalah sang pakar menambahkan rule perbaikan pada rule yang memberikan konklusi tadi.

Konsekuensi dari kebijakan ini adalah rule dengan konklusi (atau alternatif konklusi) yang sama pada rule sebelumnya yang sedang aktif tak akan dipertimbangkan untuk dieksekusi dan juga tak ada satupun dari anak-anaknya yang akan dievaluasi. Sebab jika hal itu dilakukan, maka hanya dibutuhkan satu rule anak untuk membatalkan suatu konklusi dan juga penjagaan pada urutan inferensia yang sama akan berantakan.

Akuisisi pengetahuan mempunyai langkah-langkah berikut ini:

1. Output dari sistem untuk case tertentu diidentifikasi sebagai tidak benar/salah dan case dijalankan/diproses kembali dalam mode akuisisi pengetahuan
2. Dalam akuisisi pengetahuan, inferensia dihentikan setelah setiap konklusi ditambahkan ke case.
3. Jika seorang pakar tak setuju dengan konklusi tertentu, rule perbaikan ditambahkan untuk mengkoreksi (menggantikan) konklusi tersebut.
4. Seperti biasanya dalam RDR seorang pakar harus memilih kondisi yang cukup untuk suatu rule untuk mengeluarkan sembarang case yang telah tersimpan dari eksekusi rule. Case yang menyebabkan penambahan rule ini sekarang juga disimpan sebagai satu cornerstone case. Ia akan menjadi cornerstone case untuk semua rule yang tereksekusi pada case ini dalam pembangunan output final.
5. Inferensia berlanjut langkah demi langkah dan koreksi lebih lanjut terus dibuat sampai solusi yang benar ditetapkan.

Sebagaimana perubahan prioritas dari konklusi rule dengan proses perbaikan, proses di atas akan mengkoreksi urutan dimana rule-rule dievaluasi dan diaktifkan untuk merefleksikan urutan "ideal" sang pakar yang secara implisit ajukan untuk jenis-jenis case yang berbeda.

Kita lihat satu dari pelbagai macam daya tarik kunci dari ide ini adalah seorang pakar tak akan ditanyai "bagaimana" cara dia menyelesaikan masalah. Mereka menyediakan suatu contoh dari penyelesaian masalah yang mempunyai urutan yang spesifik, tetapi setiap keputusan dalam pengambilan keputusan ini hanyalah berdasarkan ada atau tidaknya fitur-fitur dalam case, fitur yang telah ada dalam data aslinya atau yang telah ditambahkan dalam langkah sebelumnya dalam proses.

Bab 9 Multiple Classification Ripple Down Rules

Sebagai kelanjutan dari RDR untuk pengklasifikasian tunggal, maka dalam bab ini akan dibahas mengenai RDR untuk pengklasifikasian majemuk (multiple classification). Pustaka yang digunakan adalah dari B. Kang [Kang96], B. Kang, P. Compton dan P. Preston [Kang95], serta V. Ho, W. Wobcke dan P. Compton [Ho03].

9.1 Pendahuluan

Tujuan dari Multiple Classification Ripple Down Rules (MCRDR) adalah untuk menjaga keuntungan dan strategi esensial dari RDR dalam menangani klasifikasi multiple (majemuk) yang mandiri.

MCRDR seperti halnya RDR berdasarkan pada asumsi bahwa pengetahuan yang pakar sediakan adalah justifikasi yang esensial untuk suatu konklusi dalam konteks tertentu. Komponen utama dari konteks adalah case yang memberikan klasifikasi yang salah dan bagaimana hal ini berbeda dari case-case yang lain untuk mana suatu klasifikasi adalah benar adanya. Seperti yang akan kita lihat, konteks dalam MCRDR dipertahankan secara berbeda dan hanya mengikutkan rule yang memenuhi data dan validasinya diperluas untuk membedakan case baru dari sejumlah case yang berbeda.

9.2 Inferensia

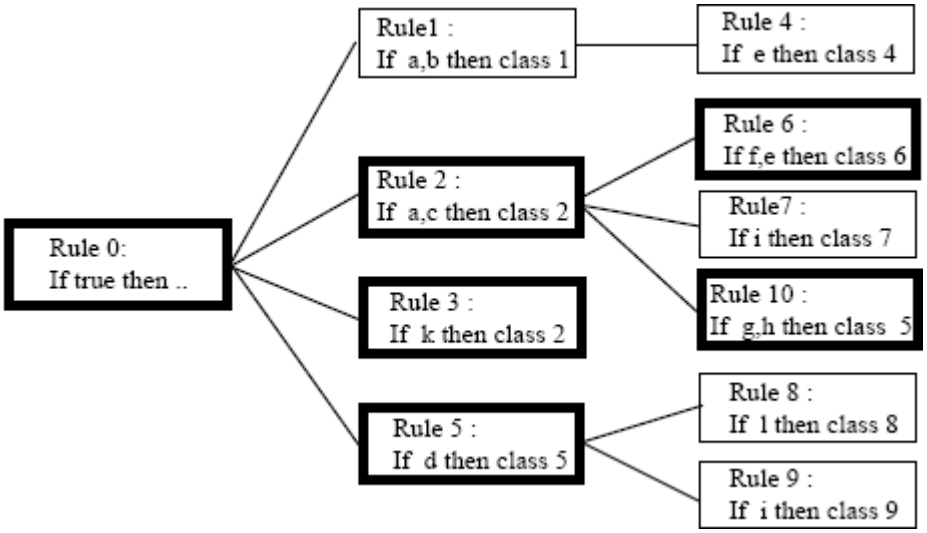
Operasi inferensia pada RDR didasarkan pada pencarian knowledge base (KB) yang direpresentasikan sebagai daftar keputusan (decision list) yang setiap keputusan mungkin diperbaiki lagi oleh daftar keputusan yang lain.

Sekali suatu rule telah memenuhi syarat maka rule-rule dibawahnya tak akan dievaluasi.

MCRDR secara kontras mengevaluasi semua rule dalam level pertama KB. Selanjutnya ia akan mengevaluasi rule-rule pada level perbaikan berikutnya untuk setiap rule yang telah memenuhi syarat pada level puncak dan begitu seterusnya. Proses berhenti bila tak ada lagi rule-rule anak yang harus dievaluasi atau tak ada satu pun dari rule-rule ini tak memenuhi syarat dengan case yang ada. Selanjutnya ia akan dikerjakan pada alur majemuk (multiple paths), dengan setiap path merepresentasikan urutan perbaikan tertentu dan tentunya banyak konklusi (multiple conclusions).

Struktur dari KB dari MCRDR dapat digambarkan sebagai tree n-ary dengan setiap node

merepresentasikan sebuah rule. Gambar 9.1 menunjukkan suatu struktur dan juga menunjukkan inferensia untuk case tertentu. Kotak yang dicetak tebal merepresentasikan rule-rule yang memenuhi case {a, c, d, e, f, h, k}



Gambar 9.1 MCRDR knowledge base system

Proses inferensia dapat dipahami dalam istilah menangkap "path" seperti yang ditunjukkan pada gambar 9.2 di bawah ini. Ketika path diproduksi terdapat sejumlah pertanyaan mengenai apakah path memproduksi suatu klasifikasi, apakah suatu klasifikasi saling bertumpuk (redundant) sebab ia juga diproduksi di tempat lain, dan lain-lain. Pada gambar 9.2 ini rule-rule yang memproduksi konklusi diberi garis bawah. Info n […] mengindikasikan nomor rule lain dengan klasifikasi yang sama.

Path 1 [(Rule 0, ...), (Rule 2, Class 2), (Rule 6, Class 6)]	Info 1
Path 2 [(Rule 0, ...), (Rule 2, Class 2), (Rule 10, Class 5)]	Info 2 [4]
Path 3 [(Rule 0, ...), (Rule 3, Class 2)]	Info 3
Path 4 [(Rule 0, ...), (Rule 5, Class 5)]	Info 4 [2]

Gambar 9.2 Jalur path melalui KB berdasarkan gambar 9.1

9.3 Akuisisi Pengetahuan

Jika suatu case diklasifikasikan tidak benar/salah atau klasifikasinya tak ada, maka akuisisi pengetahuan dibutuhkan disini dan ia dapat dibagi menjadi 3 bagian.

Yang pertama, sistem membutuhkan klasifikasi yang benar dari seorang pakar.

Kedua, sistem memutuskan sebuah lokasi dari rule baru.

Ketiga, sistem membutuhkan rule baru dari seorang pakar dan menambahkannya untuk mengkoreksi KB.

Kelihatannya para pakar menemukan sistem lebih kelihatan alami jika urutan dari langkah kedua dan ketiga dibalik, dengan cara demikian adalah lebih baik menyembunyikan rekayasa pengetahuan yang implisit yang sedang diproses. Namun demikian, pengurutan bukanlah hal yang krusial dalam hal algorithma.

9.4 Akuisisi Klasifikasi Baru

Akuisisi klasifikasi baru adalah hal yang mudah, pakar cuma perlu untuk menyatakannya. Sebagai contoh, jika sistem menghasilkan klasifikasi klas 2, klas 5, klas 6 untuk permasalahan yang diberikan, sang pakar memutuskan bahwa klas 6 tidak perlu diubah tetapi klas 2 dan klas 5 harus dihapus dan klas 7 dan klas 9 ditambahkan.

9.5 Melokalisasi Rule

Sistem harus menemukan lokasi untuk rule baru yang akan menyediakan klasifikasi-klasifikasi baur. Tak dapat diasumsikan bahwa lokasi yang benar untuk rule baru adalah seperti perbaikan untuk satu rule yang memberikan satu klasifikas tidak benar. Ini mungkin adalah klasifikasi mandiri dan klasifikasi yang tidak benar adalah jelas salah. Kemungkinannya ditunjukkan dalam tabel 9.1.

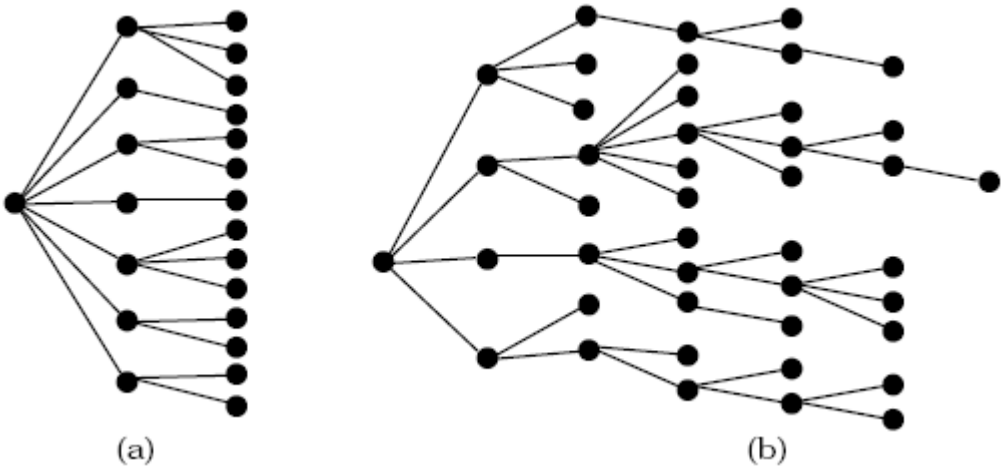
Tabel 9.1 Tiga cara dimana rule baru mengkoreksi KB

Klasifikasi Salah	**Untuk mengkoreksi KB**
Klasifikasi salah yang akan dihentikan	Tambahkan sebuah rule (rule pemberhentian) pada akhir path untuk mencegah klasifikasi
Klasifikasi salah digantikan dengan klasifikasi baru	Tambahkan sebuah rule pada akhir path untuk menghasilkan klasifikasi baru
Klasifikasi mandiri baru	Tambahkan sebuah rule pada level yang lebih tinggi untuk menghasilkan klasifikasi baru

Perlu diingat bahwa ide dari rule pemberhentian, rule yang tak menghasilkan konklusi apa-apa, atau klasifikasi null. Rule pemberhentian memainkan aturan utama dalam MCRDR dalam mencegah klasifikasi salah yang diberikan oleh suatu case.

Sebagaimana halnya mencoba memutuskan apakah suatu klasifikasi lebih baik dipandang sebagai suatu perbaikan atau klasifikasi mandiri, kita catat bahwa dalam suatu cara adalah tidak menjadi soal – keduanya merupakan solusi yang dapat diandalkan untuk sembarang klasifikasi.

Efek kunci dari lokasi rule adalah untuk mempengaruhi evolusi dan perawatan KB. Jika ada strategi yang cenderung untuk menambahkan rule pada level puncak, KB akan mencakup domain lebih cepat tetapi dengan kecenderungan salah yang lebih besar. Jika strategi cenderung untuk menambahkan rule baru pada akhir path, cakupan domain akan lebih sedikit tapi dengan kesalahan yang lebih kecil dari rule baru, hanya disebabkan mereka melihat case yang lebih sedikit. Hal ini digambarkan dalam gambar 9.3 di bawah ini.



Gambar 9.3 Struktur dari MCRDR tree jika rule ditambahkan utamanya pada puncak (a) atau sebagai perbaikan (b)

Sebuah rule dapat ditempatkan pada level menengah yang layak. Kita dapat juga mengubah strategi ini disaat sistem berkembang. Keputusan ini adalah jenis pertimbangan rekayasa pengetahuan yang baru – masalahnya adalah jenis pengembangan apa yang layak untuk domain tertentu, daripada hanya sekedar struktur dari pengetahuan.

Keputusan mengenai evolusi dari KB tidak harus menjadi keputusan rekayasa pengetahuan yang menumpuki segala pilihan pakar. Malahan pakar bebas saja menentukan jenis keputusan apa saja, tetapi antarmuka dapat didesain sehingga menjadikan para pakar lebih mudah menambahi rule melalui sisi atas atau pun bawah.

9.6 Mendapatkan Kondisi Rule – Validasi Rule

Verifikasi dan validasi kita perhatikan dengan pemastian bahwa sebuah sistem SBP berjalan seperti yang diinginkan.

Riset verifikasi umumnya berkaitan dengan pemastian konsistensi internal dari suatu KB. Pendekatan normal dalam verifikasi mencoba untuk mengurangi KB ke jalur-jalur path dari data ke konklusi dan lalu melihat hubungan diantara jalur-jalur path, data yang digunakan, konklusi level menengah yang ada, dan lain-lain.

Validasi dalam istilah perawatan dan akuisisi bertingkat, berkaitan dengan pengetesan apakah case-case yang lain yang telah dikoreksi dengan benar sebelumya akan salah diklasifikasi dengan rule yang baru, seperti halnya pemastian rule baru melingkupi case baru.

Disini kita akan lebih membahas mengenai validasi, khususnya dalam memvalidasi SBP dengan mengetesnya pada case-case yang ada.

Teknik standar yang digunakan adalah sebuah database dari case-case standar. Dalam situasi ini kita tergantung pada case-case yang direpresentasikan oleh case yang sistem maksudkan untuk dicakup. Dengan RDR, suatu case diasosiasikan dengan sebuah rule sebab rule ditambahkan untuk berhubungan dengan case tertentu itu. Sebuah rule baru harus dibedakan diantara case yang menyebabkan dibuatnya rule baru tersebut dan case yang diasosiasikan dengan rule yang memberikan klasifikasi salah sebelumnya.

Dengan MCRDR, sejumlah case (cornerstone cases) dapat mencapai sebuah rule baru dan yang lebih tinggi dalam tree maka lebih banyak case dapat mencapai rule. Rule baru seharusnya dapat dibedakan antara case baru dan semua cornerstone cases. Maka dari itu, MCRDR mempunyai banyak cornerstone cases (multiple cornerstone cases) untuk sebuah rule, dibandingkan dengan RDR (Single Multiplication RDR) yang hanya mempunyai satu cornerstone case untuk setiap rule.

Sebuah rule pada suatu level dapat diproses dengan semua case yang diasosiasikan dengan saudara-saudaranya pada level yang sama dan level anak-anaknya yang lebih rendah dalam sistem. Sehingga, rule harus dibuat layak secara spesifik sehingga tak satu pun dari case-case yang lain memenuhi rule. Namun demikian, tak jadi masalah jika case-case yang lain yang memiliki klasifikasi yang sama mencapai rule tertentu ini. Jika sebuah rule ditambahkan pada level di bawah level puncak, hanya case-case yang memenuhi syarat rule orang tua di atasnya yang perlu untuk dipertimbangkan sebagai cornerstone case. Sebagai catatan disaat sistem dikembangkan, case-case yang lain mungkin muncul yang memenuhi rule dengan benar, tetapi mungkin ditambahkan ke sistem disebabkan sebuah rule diperlukan di tempat lain untuk menambahkan klasifikasi lebih lanjut. Case seperti ini akan menjadi sebuah cornerstone case untuk rule baru di

bawah rule yang memenuhi dan untuk mana klasifikasi adalah benar. Saat tree dikembangkan, rule-rule di bagian bawah akan secara alami memiliki lebih sedikit cornerstone case yang diasosiasikan dengan mereka.

Tujuan berikutnya adalah untuk membuat rule baru layak dan tepat sehingga ia memenuhi hanya case yang akan ditambahkan dan tak ada case lain yang sudah disimpan, kecuali bahwa adalah tidak penting jika ini terjadi untuk memenuhi case-case yang termasuk pada klasisfikasi yang sama. Algoritma untuk pemilihan kondisi untuk membuat rule layak dan tepat adalah sangat sederhana.

Dimisalkan case baru A dan dua cornerstone case B dan C. Dalam pembuatan rule baru, kita dapat membayangkan bahwa pakar harus memilih paling tidak satu diantara kondisi-kondisi dari

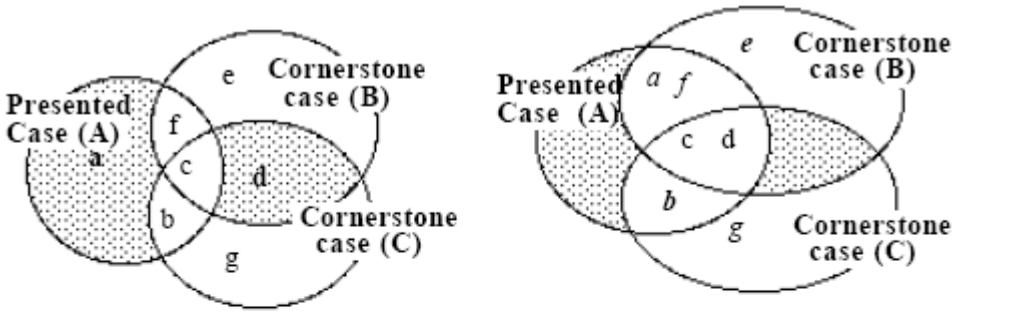
$$(\text{Case A} - (\text{Case B} \cup \text{Case C}))$$

atau

negasikan kondisi-kondisi dari $((\text{Case B} \cap \text{Case C}) - \text{Case A})$

Namun demikian, seperti terlihat pada gambar 9.4, ini bisa saja kosong – mengarah pada situasi dimana tak ada satu pun kondisi rule dapat ditemukan. Alternatifnya daftar perbedaan mengandung hanya kondisi yang sepele yang tak relevan. Dengan kata lain tak ada kondisi yang sama yang membedakan case baru yang disediakan dari semua cornerstone case, tetapi sejumlah kondisi yang berbeda membedakan case-case yang berbeda dan kondisi-kondisi ini harus dilibatkan dalam rule-rule baru.

Dalam gambar 9.4 terlihat dua case yang mirip dimana daftar perbedaan akan ditemukan untuk membedakan case yang disediakan (case A) dari dua cornerstone case B dan C. Perlu dicatat dalam bagian (a) perbedaan diantara case yang disediakan dan irisan dari case-case lain memperlihatkan kondisi yang pantas. Untuk bagian (b) rule baru harus melibatkan kondisi dari kedua daftar perbedaan diantara A dan B dan dari daftar perbedaan diantara B dan C.



(a)

Difference list between A and (B and C)	*a* *NOT d*

(b)

Difference list between A and (B ∩ C)	empty
Difference list between A and B	*b* *NOT e*
Difference list between A and C	*a, f* *NOT g*

Gambar 9.4 Diagram case A dan cornerstone case B dan C

Algoritma yang digunakan adalah sebagai berikut ini. Pertama, sistem dapat membentuk daftar cornerstone case yang dapat mencapai rule baru dan harus dibedakan dari case yang disediakan. Pakar ditanyai untuk memilih dari daftar perbedaan diantara case yang tersedia dan satu dari cornerstone cases dalam daftar cornerstone cases yang telah ada.

Sistem lalu mengetes semua cornerstone case dalam daftar dibandingkan dengan kondisi-kondisi yang terpilih dan menghapus cornerstone case dari daftar yang tak memenuhi kondisi yang terpilih.

Selanjutnya pakar ditanyai untuk memilih kondisi-kondisi dari daftar perbedaan diantara case yang ada saat ini dan satu dari cornerstone case yang tersisa dalam daftar. Kondisi-kondisi yang terpilih ditambahkan sebagai penghubung ke rule. Sistem mengulangi proses ini sampai tak ada lagi cornerstone case dalam daftar yang memenuhi rule. Pertanyaan krusial untuk evaluasi berikut ini adalah apakah proses ini membutuhkan terlalu banyak langkah dari penambahan kondisi-kondisi ke rule-rule.

Setelah sistem menambahkan sebuah rule dengan kondisi-kondisi terpilih, ia mengetes cornerstone case yang tersisa yang diasosiasikan dengan parent rule dan sembarang case yang dapat memenuhi rule baru lalu disimpan sebagai cornerstone case dari rule baru.

Perlu dicatat lagi bahwa dibolehkan untuk case-case yang melibatkan klasifikasi yang diberikan oleh rule untuk memenuhi rule, dan cornerstone case yang telah disimpan

melibatkan case-case ini.

Akhirnya case baru ditambahkan ke data base cornerstone case. Daftar dari cornerstone case untuk rule-rule yang lain yang secara benar dipenuhi oleh case (yaitu yang memberikan klasifikasi yang benar untuk case ini) juga diupdate untuk melibatkan case baru ini. Sistem sekarang siap untuk menjalankan case yang lain dan, jika klasifikasi-klasifikasi yang ada adalah tidak benar/salah, untuk akuisisi pengetahuan lebih banyak lagi.

Sistem MCRDR dikembangkan untuk seluruh domain atau secara bertahap, sub-domain pada waktu tertentu untuk mencegah tuntutan-tuntutan pada sang pakar. Sehingga case yang memiliki lebih dari satu klasifikasi mungkin hanya memberikan satu klasifikasi awal. Dalam hal penanganan dengan sub-domain secara bertahap, algoritmanya adalah sama kecuali untuk kebutuhan konsultasi ekstra sang pakar yaitu apakah klasifikasi baru diaplikasikan untuk setiap cornerstone case yang dapat memenuhi rule. Namun demikian, fitur yang tertinggal adalah rule menjadi lebih tepat untuk mengeluarkan case yang spesifik, case-case lain yang juga dikeluarkan tak akan lagi dipertimbangkan lebih lanjut.

Bab 10 Pemrograman RDR

Pemrograman Ripple Down Rules (RDR) yang telah dibahas pada bab 8 dan 9 akan dikupas habis disini, sehingga diharapkan dapat dipahami bagaimana RDR itu sebenarnya diimplementasikan dalam pemrograman. Pustaka yang digunakan dalam bab ini adalah dari B. Kang [Kang96], B. Kang, P. Compton dan P. Preston [Kang95], serta V. Ho, W. Wobcke dan P. Compton [Ho03] dan juga asumsi serta pengembangan yang dilakukan oleh penulis sendiri berdasarkan metode yang telah diajukan dan dikembangkan penulis yaitu Variable-Centered Intelligent Rule System (VCIRS) yang nantinya akan dibahas pada bab 18 di bagian terakhir modul ini.

10.1 Ringkasan RDR

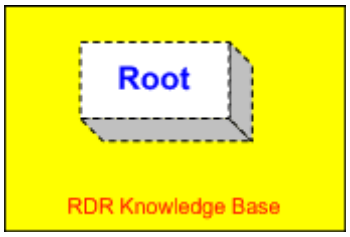
Sekarang tiba saatnya kita untuk mengimplemetasikan RDR secara nyata dalam pemrograman sehingga kita bisa merasakan manfaatnya secara langsung dalam kehidupan kita sehari-hari.

Dari bab 9 yang telah kita pelajari, bisa kita ringkaskan hal-hal mengenai RDR, yaitu:

- RDR adalah sistem yang knowledge buildingnya bersamaan waktunya dengan knowledge inferencing. Jadi penekanan adalah pada knowledge buildingnya, knowledge inferencing sudah tercakup dalam proses knowledge buildingnya.
- RDR bekerja berdasarkan Cornerstone Cases (CC). CC adalah rule-rule dalam Knowledge Base (KB) yang membuat sistem RDR melakukan kesalahan klasifikasi (yang akhirnya nanti membuat rule baru dalam KB).
- Single Classification RDR (SCRDR) bekerja dengan hanya mencocokkan 1 CC untuk setiap case baru yang akan diolah ke KB. SCRDR memiliki tree berbentuk biner.
- Multi Classification RDR (MCRDR) bekerja dengan mencocokkan semua CC yang ada untuk setiap case baru yang akan diolah ke KB. MCRDR tree-nya bisa berupa n–ary tree.
- Dalam knowledge building (dan inferensia tentunya) sistem akan mencari rule-rule yang bersesuaian, bila sesuai maka baru anak-anaknya yang akan terus dicek. Bila tak sesuai maka saudara dengan urutan umur mulai yang tertua yang akan dicek selanjutnya.

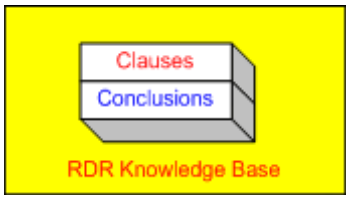
10.2 RDR Tree

Bila dalam RDR kita masih belum memiliki apa-apa dalam KB-nya, maka tree yang ada dapat digambarkan seperti gambar 10.1 seperti di bawah ini. Terlihat adanya 1 buah root imajiner dalam KB.



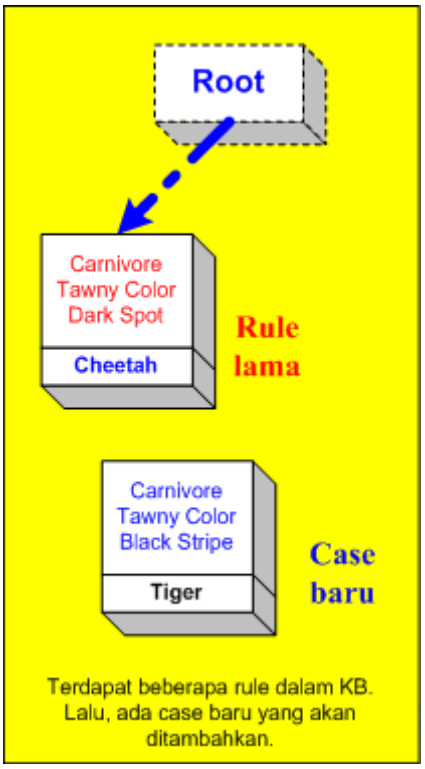
Gambar 10.1 Knowledge Base dalam RDR saat masih kosong

Setiap 1 node dalam KB adalah merupakan representasi dari 1 rule dalam RBS, seperti disajikan dalam gambar 10.2 di bawah ini.

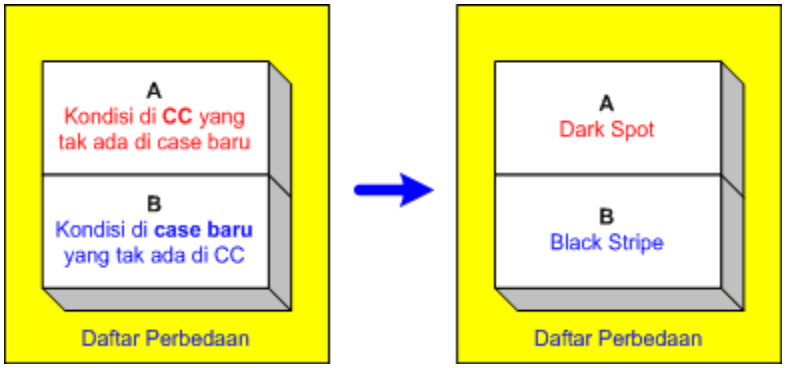


Gambar 10.2 Representasi 1 node dalam KB pada RDR

Setiap case baru yang diolah dalam RDR (seperti digambarkan pada gambar 10.3 di bawah), dibandingkan dengan rule-rule yang telah ada dalam KB (tentu juga termasuk CC di dalamnya) lalu dibuatlah satu Daftar Perbedaan yang memuat 2 bagian, yaitu bagian untuk CC (A) dan bagian untuk rule baru (B), seperti digambarkan pada gambar 10.4 di bawah ini.



Gambar 10.3 Gambaran case baru yang akan ditambahkan dalam KB



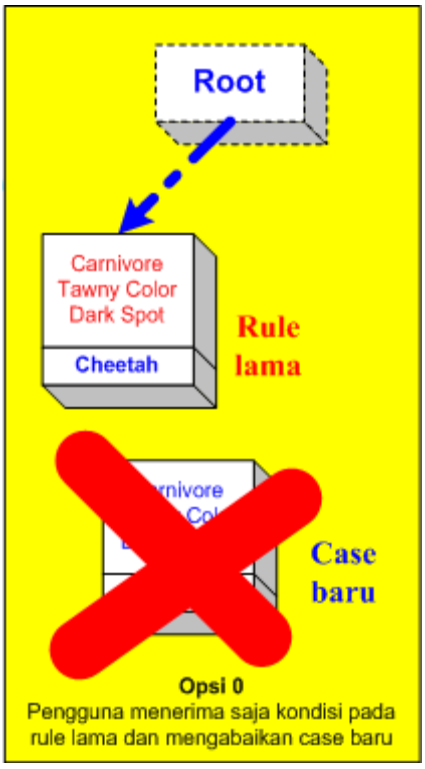
Gambar 10.4 Daftar Perbedaan

Selanjutnya sistem akan memberikan 6 opsi kepada pengguna, berdasarkan Daftar Perbedaan yang telah dibuat tadi, yaitu:

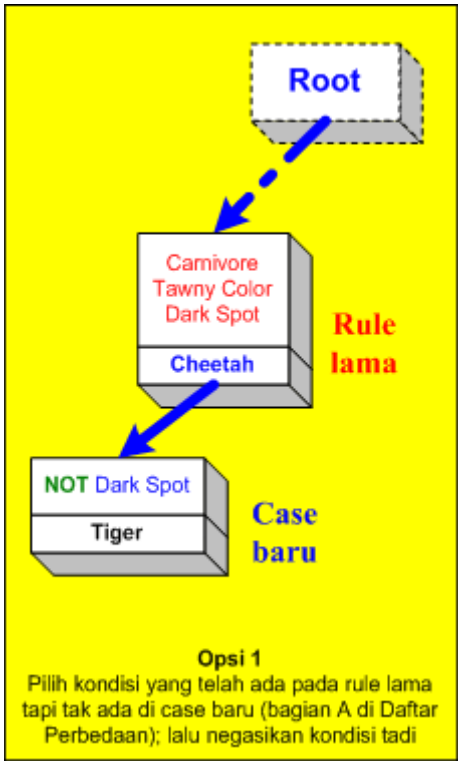
1. Pengguna menerima rule yang telah ada, tak memasalahkan adanya case baru yang tersedia → inferensia saja
2. Rule baru yang akan dihasilkan adalah negasi dari kondisi pada bagian A di Daftar Perbedaan
3. Rule baru yang akan dihasilkan adalah kondisi pada bagian B di Daftar Perbedaan
4. Rule baru yang akan dihasilkan adalah gabungan opsi 1 dan opsi 2

5. Rule baru yang akan dihasilkan semuanya berasal dari case baru yang tersedia **tanpa** melihat Daftar Perbedaan, namun letaknya adalah pada level yang sama dari CC yang sedang dicek tadi. Artinya rule baru ini menjadi saudara termuda dari CC yang sedang dicek atau dengan kata lain rule baru ini akan memiliki orang tua yang sama dengan CC, hanya saja karena urutan dia adalah yang terakhir dibentuk maka rule baru tersebut menjadi anak yang paling muda dari orang tuanya CC yang sedang dicek tadi.
6. Rule baru yang akan dihasilkan semuanya berasal dari case baru yang tersedia **tanpa** melihat Daftar Perbedaan, dan letaknya langsung di bawah root imajiner. Dengan kata lain ini adalah rule yang terletak pada level puncak KB.

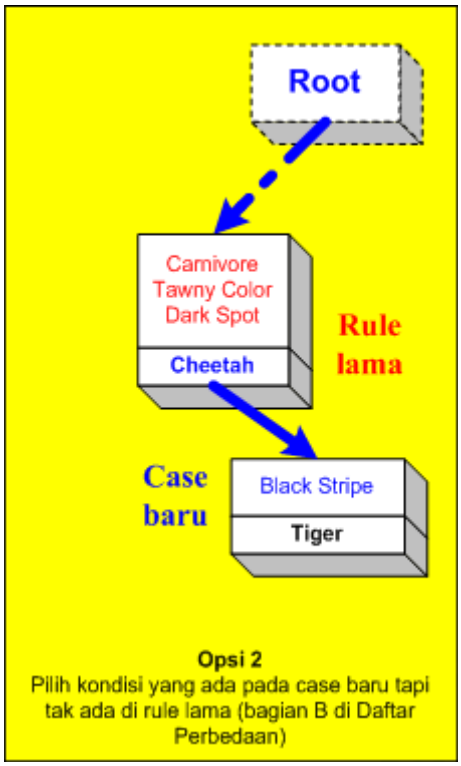
Penjelasan dari opsi-opsi di atas disajikan dalam gambar 10.5-10.10 berikut ini.



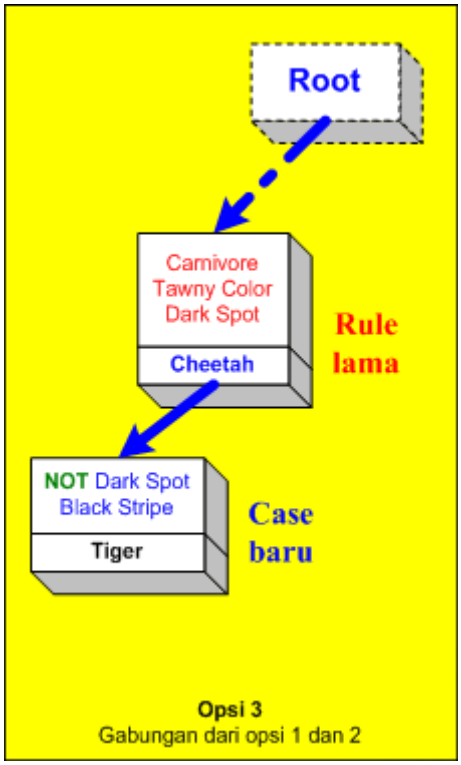
Gambar 10.5 Representasi opsi 0



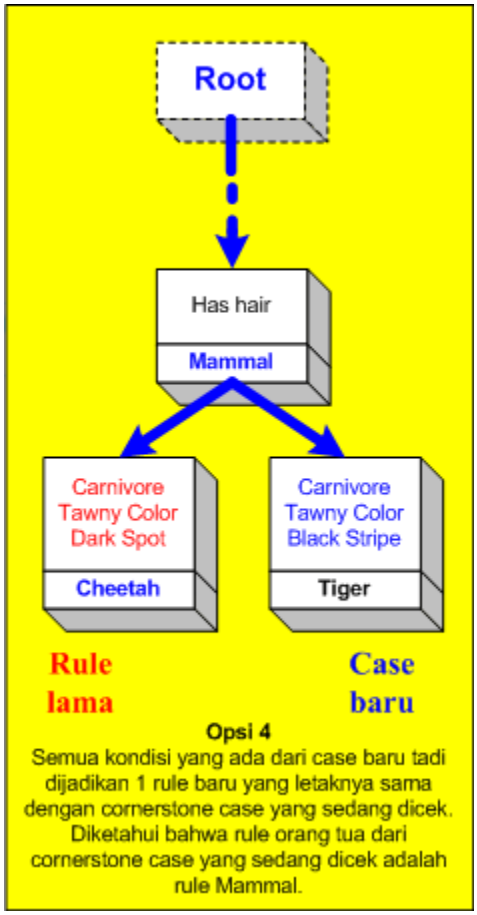
Gambar 10.6 Representasi opsi 1



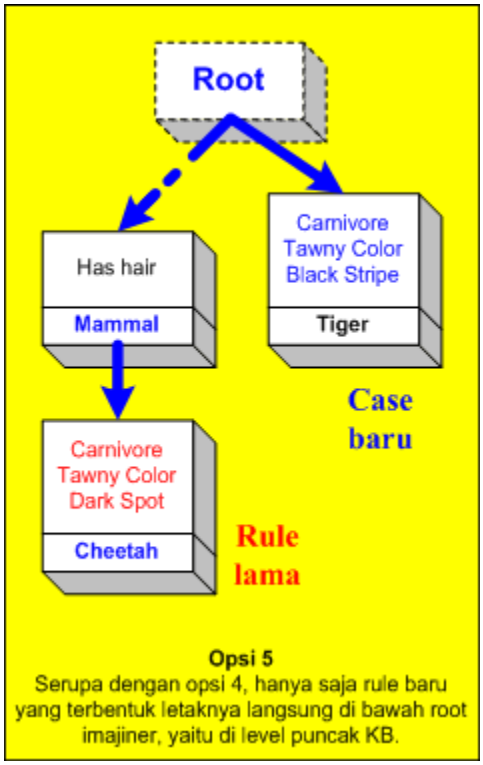
Gambar 10.7 Representasi opsi 2



Gambar 10.8 Representasi opsi 3

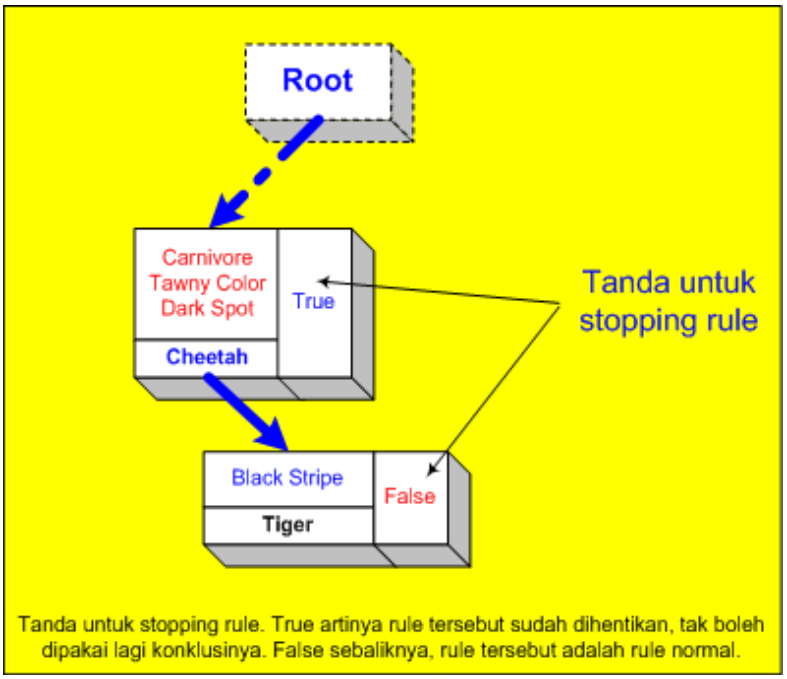


Gambar 10.9 Representasi opsi 4



Gambar 10.10 Representasi opsi 5

Kalau diiinginkan suatu rule tak dipakai lagi maka cukup ditambahkan satu tanda padanya yang menandakan bahwa rule ini tak boleh dipakai lagi (artinya konklusinya tidak boleh digunakan, namun clausesnya tetap masih boleh dipakai oleh rule-rule di bawahnya). Hal tersebut (disebut dengan stopping rule) disajikan dalam gambar 10.11 di bawah ini.

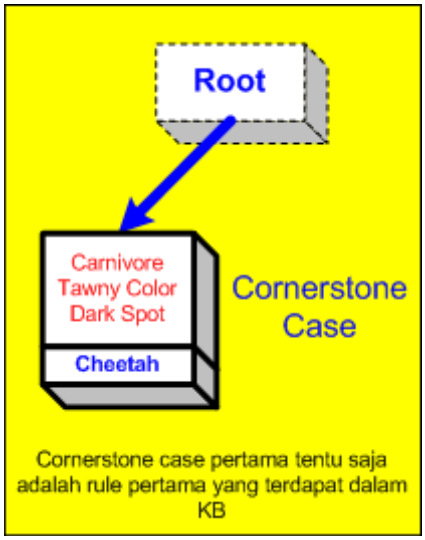


Gambar 10.11 Representasi stopping rule

10.3 Cornerstone Cases

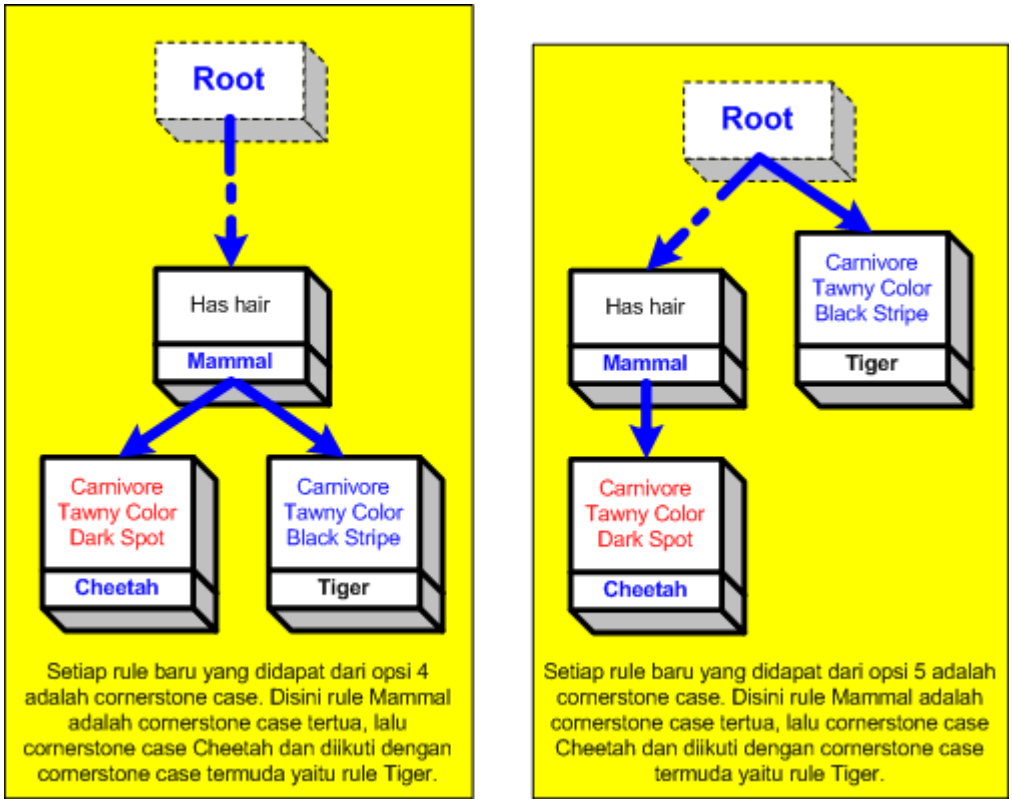
RDR bekerja berdasarkan Cornerstone Cases (CC). CC adalah rule juga, namun ia adalah rule-rule dalam KB yang membuat sistem RDR melakukan kesalahan klasifikasi. Namun dengan adanya kesalahan klasifikasi ini, justru dari sinilah malahan RDR bisa membuat rule baru dalam KB.

CC pertama tentu saja adalah rule pertama yang menempati KB, yaitu rule pertama yang langsung di bawah root imajiner. Ini ditunjukkan oleh gambar 10.12 di bawah ini.



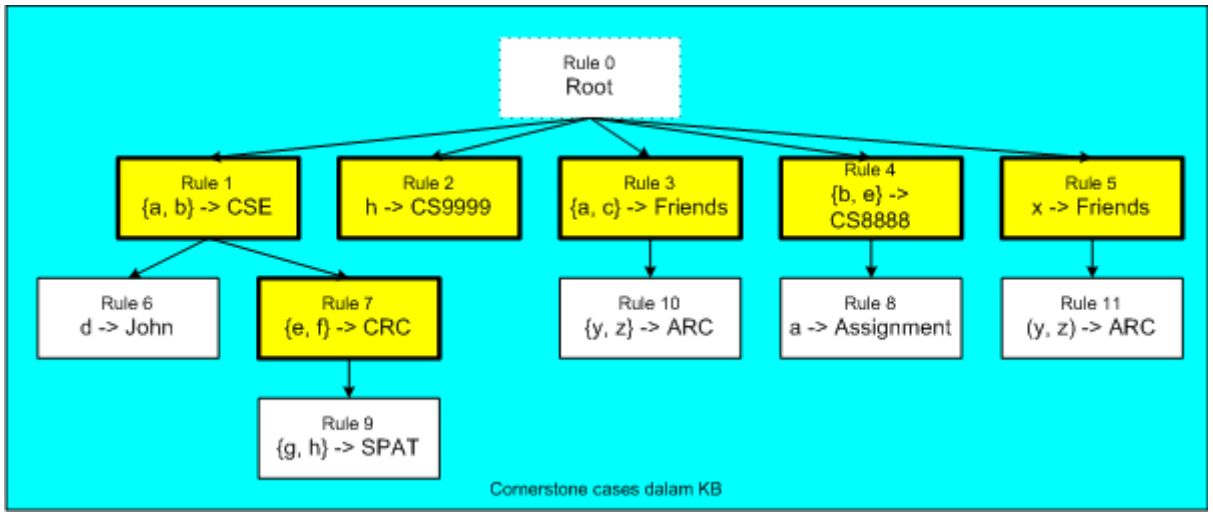
Gambar 10.12 CC pertama

Selanjutnya, setiap rule baru yang ditambahkan dalam KB berdasarkan opsi 4 dan opsi 5 juga akan menjadi CC. Ini digambarkan pada gambar 10.13 di bawah ini.



Gambar 10.13 Rule dari opsi 4 dan opsi 5 menjadi CC

Sebagai contoh, dari gambar 10.14 di bawah ini, maka CC-nya adalah yang garisnya dicetak tebal dan berwarna kuning. Selanjutnya untuk menandai bahwa suatu rule adalah CC dapat digunakan satu tanda seperti disajikan pada gambar 10.15.

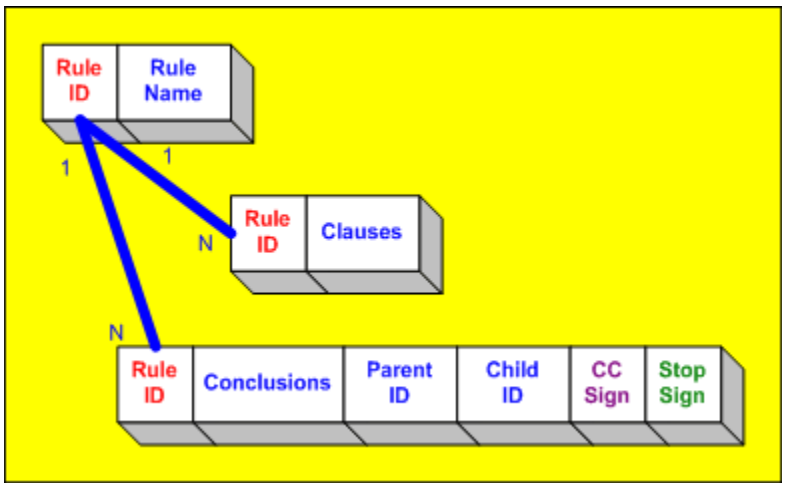


Gambar 10.14 Contoh CC dalam suatu KB

10.4 Struktur Data

Walaupun KB dalam bentuk tree, namun untuk lebih mudahnya secara praktis akan kita terapkan pada database.

Gambaran dari database untuk RDR ini dapat dilihat pada gambar 10.15 di bawah ini.



Gambar 10.15 Struktur data KB untuk RDR

Bab 11 Sistem Fuzzy

Bab ini membahas mengenai sistem fuzzy. Pustaka yang digunakan adalah dari situs http://www.comp.nus.edu.sg/~pris/FuzzyLogic/ yaitu pustaka [Nus].

Sistem fuzzy diawali oleh Lotfi A. Zadeh pada tahun 1960, pada dasarnya berhubungan dengan bagaimana manusia menangani ketidaktepatan (imprecise) dan informasi yang tidak pasti (uncertain). Ia menirukan bagaimana manusia menggunakan perkiraan pertimbangan (approximate reasoning) dalam hal berhubungan dengan ketidaktepatan (impresion), ketidakpastian (uncertainty), ketidakakurasian (inaccuracy), ketidakpersisan (inexactness), kerancuan (ambiguity), ketidakjelasan (vagueness), kekualitatifan (qualitativeness), subjektifitas (subjectivity) dan persepsi (perception) yang dialami setiap hari dalam pengambilan keputusan.

Sistem fuzzy mampu untuk menangani istilah-istilah samar-samar seperti quite (sungguh), very (amat) dan extremely (ekstrim) yang normal dipakai dalam bahasa sehari-hari. Ini secara langsung ada dalam konsep "komputasi dengan kata – computing with words (CW)" yang secara kontras beda dengan manipulasi bilangan dan simbol. Konsep komputasi dengan kata berhubungan dengan bahasa dan persepsi sedangkan manipulasi bilangan dan simbol berhubungan dengan pengukuran crisp (krispi, kaku, kering, garing) dan tertentu.

Menurut Zadeh manusia sukses dalam pengembangan mulai dari persepsi menuju pengukuran (sehingga sukses berhubungan dengan pengukuran yang tepat seperti halnya operasi-operasi di bidang penerbangan) tetapi masih harus mengembangkan pengukuran menuju ke persepsi (sehingga sukses dalam menangani fungsionalitas fleksibel seperti halnya pergerakan robot yang seperti manusia).

"Komputasi lunak – soft computing" diciptakan oleh Zadeh. Area ini, seperti yang dia katakan adalah perpaduan dari teknik dan metodologi diantara fuzzy logic (logika fuzzy), neuro-computing (komputasi syaraf) dan probabilistic reasoning (pertimbangan probabilistik) yang selanjutnya diikuti oleh algoritma genetika, chaotic systems (sistem chaos), belief networks (jaringan kepercayaan) dan bagian-bagian learning theory (teori pembelajaran).

11.1 Logika Fuzzy

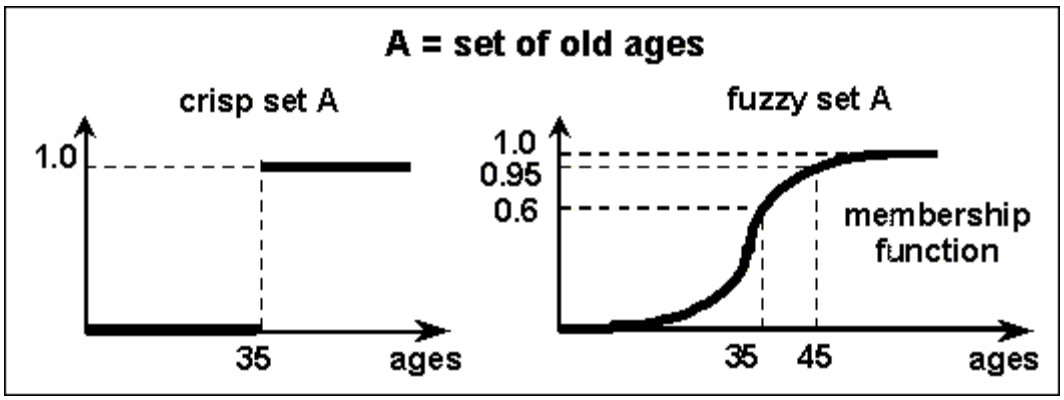
Logika fuzzy adalah superset (bagian yang melingkupi) logika boolean konvensional yang dikembangkan untuk menangani konsep kebenaran sebagian – nilai kebenaran diantara "kebenaran lengkap" dan "kesalahan lengkap". Transisi dari nilai kebenaran dari "kebenaran lengkap" ke "kesalahan lengkap" ditampilkan dalam fuzzy sets dan tidak dalam crisp sets.

Dalam crisp sets, seperti halnya set A dari umur tua

{A | A >= 35}

ada perbedaan yang tajam atau batasan diantara anggota set/himpunan, dan mereka yang bukan anggota dari himpunan. Dalam contoh, hanya usia yang lebih besar atau sama dengan 35 dipertimbangkan sebagai anggota dari himpunan A, dan usia yang lain bukanlah anggota. Tetapi bagaimana dengan 34.8? Akankah ia dianggap sebagai tua? Apakah usia diantara 35 dan 85 juga membawa konsep yang sama dengan "tua"? Dengan kata lain, bagaimana tua itu "tua"? Masalah yang sama timbul dalam mengklasifikasikan pelbagai pengukuran yang lain seperti tingkat panas-dingin dari suatu suhu, tinggi badan dan kemahalan dalam harga. Ketidaktepatan dan ketidakpastian dalam istilah seperti halnya fleksibelitas dalam nilai kebenaran dapat ditangani dalam fuzzy sets tetapi hal ini tidak bisa dalam crisp sets.

Dalam fuzzy sets, transisi dari fungsi anggota ke fungsi kebukananggotaan dalam himpunan adalah gradual (berjenjang) dan tidak berubah secara mendadak. Gambar 11.1 di bawah ini menjelaskan perbedaan diantara crisp set dan fuzzy set dari usia tua.



Gambar 11.1 Perbedaan crisp dan fuzzy set untuk usia tua

Dalam ilustrasi fuzzy set A, usia 35 atau di bawahnya masih mejadi anggota fuzzy set A usia tua tetapi derajat keanggotaannya lebih rendah. Derajat keanggotaan dalam fuzzy set adalah dalam jangkauan 0.0 (bukan anggota) sampai 1.0 (anggota penuh) yang

kontras dengan hanya 0.0 (member) atau 1.0 (bukan anggota) dalam crisp set. Umumnya, derajat keanggotaan dalam fuzzy set usia muda berkurang pada saat usia juga berkurang. Jika kita memiliki fuzzy set B usia muda, kita harapkan derajat keanggotaannya meningkat bersama dengan meningkatnya usia.

Dari gambar 11.1 di atas, usia 45 dapat dipersepsikan dalam logika fuzzy sebagai "tua" pada angka sekitar 0.95 dan sebagai "muda" pada angka sekitar 0.5. Hingga kita berpindah dari pengukuran eksak, crisp ke non eksak, persepsi fuzzy yang secara normal diasosiasikan ke kata atau istilah linguistik (berhubungan dengan bahasa).

Logika fuzzy sudah diaplikasikan dalam sistem pakar untuk menangani ketidakpastian bahasa yang digunakan oleh pakar ketika mereka mengungkapkan dengan kata-kata pengetahuan mengenai domain tertentu. Derajat ketidakpastian digunakan tidak hanya dalam merepresentasikan pengetahuan pakar, tetapi juga dalam pemrosesan tugas-tugas pakar.

Pengetahuan direpresentasikan dalam sistem pakar fuzzy menggunakan variabel linguistik, nilai linguistik, istilah linguistik, fungsi keangggotaan dan rule IF-THEN fuzzy.

11.2 Ketidaktepatan dan Ketidakpastian

Ke-fuzzy-an seharusnya tidak dibingungkan dengan format-format lain dari ketidaktepatan dan ketidakpastian. Terdapat beberapa jenis ketidaktepatan dan ketidakpastian dan ke-fuzzy-an hanyalah salah satu aspek dari hal-hal tersebut. Ketidaktepatan dan ketidakpastian merupakan aspek-aspek dari pengukuran, probabilitas dan deskripsi.

Ketidaktepatan dalam pengukuran diasosiasikan dengan kekurangan pengetahuan yang presisi. Kadang-kadang kita memiliki pengukuran yang tidak akurat, tak eksak, atau memiliki kepercayaan yang rendah.

Ketidaktepatan sebagai bentuk probabilitas diasosiasikan dengan ketidakpastian mengenai kejadian atau fenomena di masa datang. Ia memperhatikan kemungkinan dari kejadian-kejadian yang tidak pasti (ketidakpastian stokastik). Sebagai contoh adalah pernyataan "besok mungkin akan hujan" yang menampilkan derajat keacakan.

Ketidaktepatan dalam deskripsi adalah jenis ketidaktepatan yang berkenaan dengan logika fuzzy. Ia adalah kerancuan, ketidakjelasan, kekualitatifan atau subjektifitas dalam bahasa sehari-hari kita (ketidakpastian bahasa, leksikal atau semantik). Kerancuan ditemukan dalam definisi konsep atau arti dari istilah seperti halnya "bangunan tinggi" atau "nilai rendah". Terdapat juga kerancuan dalam pemikiran manusia, yaitu persepsi dan interpretasi. Contoh pernyataan yang bersifat fuzzy dalam kehidupan kita sehari-hari

adalah "jumlah hemoglobinnya sangat rendah." dan "Teddy lebih berat dibandingkan dengan Ike."

Selanjutnya sifat alamiah ke-fuzzy-an dan keacakan sangat berbeda. Mereka berbeda dalam aspek ketidakpresisian dan ketidakpastian. Yang pertama membawa pemikiran, perasaan atau bahasa manusia yang subjektif, dan yang kedua mengindikasikan statistik objektif dalam ilmu pengetahuan alam.

Dalam cara pandang pemodelan, model fuzzy dan model statistika juga memiliki jenis informasi yang secara filosofi berbeda: keanggotaan fuzzy merepresentasikan kemiripan objek untuk secara tak presisi mendefinisikan properti, dimana probabilitas membawakan informasi mengenai frekuensi relatif. Jadi, ke-fuzzy-an berhubungan dengan hal-hal pasti yang masuk akal dan probabilitas yang tidak pasti.

11.3 Variabel Linguistik, Nilai Linguistik dan Istilah Linguistik

Seperti variabel numerik mengambil nilai numerik, dalam logika fuzzy, variabel linguistik mengambil nilai linguistik yang terdiri dari kata-kata (istilah linguistik) yang diasosiasikan dengan derajat keanggotaan dalam himpunan. Jadi, variabel "tinggi" bukan diasumsikan sebagai nilai numerik 1.75 meter, namun ia diperlakukan sebagai variabel linguistik yang diasumsikan; sebagai contoh, nilai linguistik "tinggi" dengan derajat keanggotaan 0.92. Sedangkan "sangat pendek" dengan derajat 0.06 atau "sangat tinggi" dengan derajat 0.7. Konsep ini diperkenalkan oleh Zadeh untuk menyediakan pengertian pendekatan karakterisasi fenomena yang terlalu kompleks atau terlalu jelek didefinisikan agar dapat diterima secara deskripsi dalam istilah kuantitatif konvensional.

Variabel linguistik mengambil nilai-nilai yang didefinisikan dalam himpunan istilah – yaitu istilah linguistik. Istilah linguistik adalah kategori subjektif untuk variabel linguistik. Sebagai contoh, untuk variabel linguistik "usia", istilah set T (usia) dapat didefinisikan sebagai berikut:

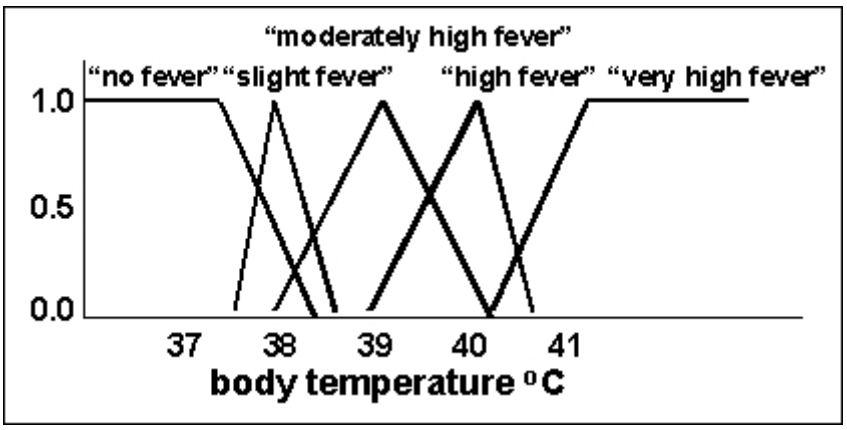
T (usia) = {"muda", "tidak muda", "tidak terlalu muda", "sangat muda", …, "usia pertengahan", "bukan usia pertengahan", …, "tua", "tidak tua", "sangat tua", "kurang lebih tua", "amat tua", …, "tidak sangat muda dan tidak sangat tua", …}

11.4 Fuzzy Set dan Fungsi Anggota

Setiap istilah linguistik diasosiasikan dengan fuzzy set, yang masing-masing memiliki fungsi anggota (*membership function* - MF) yang telah didefinisikan. Secara formal, sebuah fuzzy set A dalam U diekspresikan sebagai set dari pasangan berurutan

$A = \{ (x, m_A(x)) \mid x \text{ in } U \}$

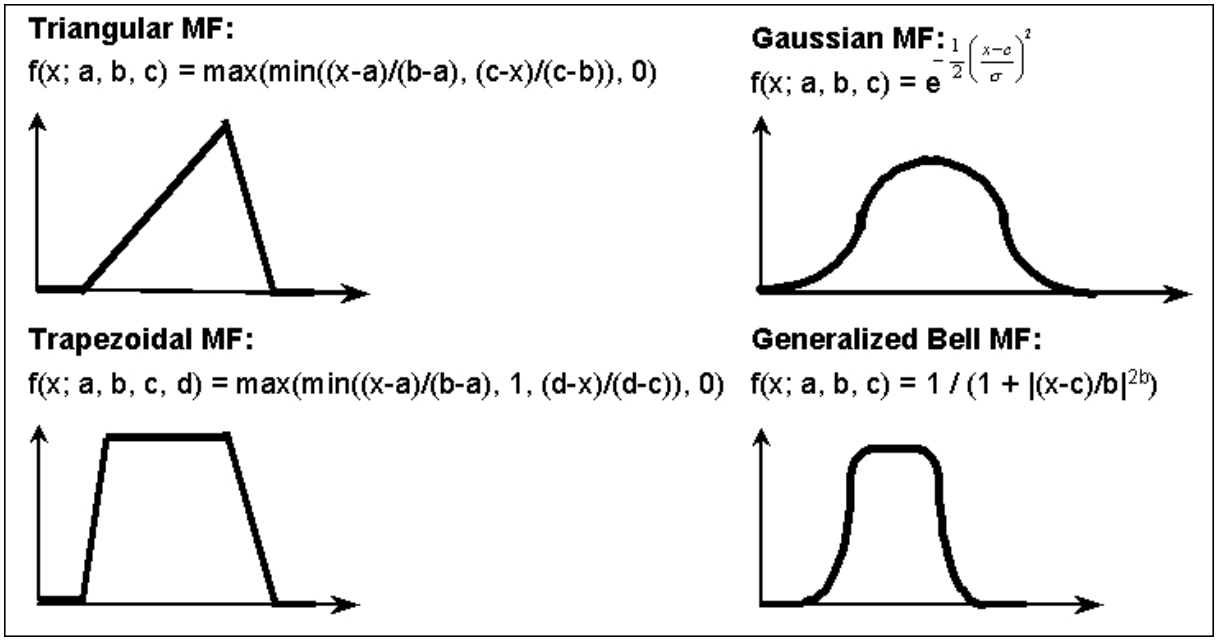
Dimana $m_A(x)$ adalah fungsi anggota yang menentukan derajat keanggotaan x. Ini mengindikasikan derajat yang mana x memilikinya dalam set A. Gambar 11.2 di bawah ini mengilustrasikan variabel linguistik "suhu tubuh" dengan lima istilah linguistik terasosiasikan yaitu "tidak demam – no fever", "sedikit demam – slight fever", "demam cukup tinggi – moderately high fever", "demam tinggi – high fever" dan "demam sangat tinggi – very high fever". Setiap istilah linguistik ini diasosiasikan dengan fuzzy set yang didefinisikan oleh fungsi anggota yang sesuai.



Gambar 11.2 Contoh variabel dan istilah linguistik, fuzzy set dan fungsi anggota

Haruslah diulangi bahwa fungsi anggota adalah pengukuran subjektif untuk istilah linguistik dan bukanlah fungsi probabilitas.

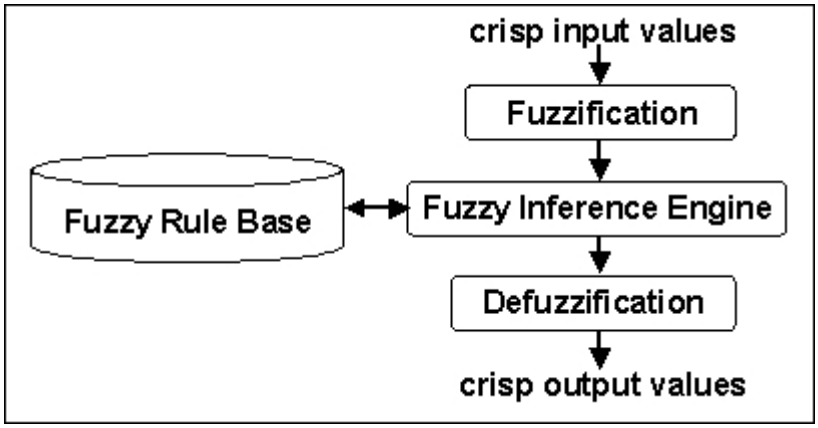
Terdapat banyak jenis fungsi anggota. Salah satu yang paling umum adalah *triangular membership functions* (seperti fungsi pada gambar 11.2 di atas), *trapezoidal MF*, *gaussian MF* dan *generalized bell MF*. Gambar 11.3 di bawah ini menggambarkan definisi dan graf dari fungsi-fungsi anggota ini.



Gambar 11.3 Pelbagai jenis fungsi anggota

11.5 Model Sistem Pakar Fuzzy

Sebuah sistem pakar fuzzy terdiri dari 4 buah komponen yang diberi nama: *fuzzifier* (pemfuzzifikasi), *inference engine* (mesin inferensia), *defuzzier* (defuzzifikasi) dan *fuzzy rule base*. Untuk lebih mudahnya, hal tersebut dapat disajikan pada gambar 11.4 berikut ini.



Gambar 11.4 Model sistem pakar fuzzy

Dalam *fuzzifier*, *crisp input* difuzzifikasikan kedalam nilai-nilai linguistik yang diasosiasikan ke input variabel linguistik. Setelah *fuzzification* (pemfuzzifikasian), mesin inferensia mengacu pada *fuzzy rule base* yang mengandung rule-rule IF-THEN untuk menurunkan nilai-nilai linguistik untuk variabel linguistik menengah dan output. Sekali

nilai linguistik output tersedia, *defuzzifier* menghasilkan nilai akhir crisp dari nilai-nilai linguistik output.

Fuzzification

Melalui penggunaan fungsi anggota yang mendefinisikan setiap fuzzy set untuk setiap variabel linguistik, derajat keanggotaan dari nilai crisp dalam setiap fuzzy set dapat ditentukan. Seperti contoh pada gambar 11.5 di bawah ini variabel numerik "usia" yang memberikan nilai 25.0 di fuzzifikasikan menggunakan *triangular MF* mendefinisikan setiap fuzzy set untuk variabel linguistik "usia". Sebagai hasil dari fuzzification, variabel linguistik usia memiliki nilai linguistik "muda" dengan derajat keanggotaan 0.666, "sangat tua" dengan nilai 0.333 dan untuk nilai linguistik yang lain dengan nilai 0.0.

Gambar 11.5 Contoh fuzzification dari crisp input

Dalam aplikasi sistem pakar fuzzy, setiap nilai crisp variabel input pertama kali difuzzifikasikan ke dalam nilai linguistik sebelum mesin inferensia mengolahnya dalam pemrosesan dengan rule base.

Fuzzy Rule Base dan Fuzzy IF-THEN Rules

Sistem pakar fuzzy menggunakan fuzzy IF-THEN rules. Sebuah fuzzy IF-THEN rule memiliki bentuk:

IF $X_1 = A_1$ and $X_2 = A_2$ ... and $X_n = A_n$ THEN $Y = B$

Dimana X_i dan Y adalah variabel linguistik, sedangkan dan A_i dan B adalah istilah linguistik. Bagian IF adalah *antecedent* atau *premise* (dasar pikiran/alasan), sedangkan bagian THEN adalah *consequence* atau *conclusion* (konklusi). Sebagai contoh dari fuzzy IF-THEN rule adalah:

IF *pressure* = "low" THEN *volume* = "big"

Contoh lain adalah seperti di bawah ini untuk masalah profile investasi.

IF (*age* = "young") and (*fund* = "small") THEN (*tolerance* = "risk neutral")
IF (*age* = "quite old") and (*fund* = "large") THEN (*tolerance* = "venturesome")
IF (*fund* = "very large") THEN (*tolerance* = "venturesome")

Dalam sistem pakar fuzzy, koleksi dari fuzzy IF-THEN rules disimpan dalam fuzzy rule base yang tereferensi oleh mesin inferensia saat mengolah input.

Mesin Inferensia

Sekali semua nilai *crisp input* telah difuzzifikasikan dalam nilai linguistik yang bersesuaian, mesin inferensia akan mengakses fuzzy rule base dari sistem pakar fuzzy untuk menurunkan nilai-nilai linguistik untuk hasil variabel linguistik antara (*intermediate*) sebagaimana variabel linguistik output.

Dua langkah utama dalam proses inferensia adalah *aggregation* (agregasi - pengumpulan) dan *composition* (komposisi - penyusunan). Agregasi adalah proses penghitungan untuk nilai-nilai dari bagian IF (*antecedent*) dari rule dimana komposisi adalah proses penghitungan untuk nilai-nilai di bagian THEN (konklusi) dari suatu rule.

Selama agregasi, setiap kondisi dalam bagian IF suatu rule diberi nilai derajat kebenaran berdasarkan pada derajat keanggotaan istilah linguistik yang sesuai. Dari sini, baik *minimum* (MIN) atau *product* (PROD) dari derajat kebenaran dari kondisi-kondisi biasanya dihitung untuk memotong derajat kebenaran dari bagian IF. Ini diberi nilai sebagaimana pada derajat kebenaran pada bagian THEN.

Sebagai contoh, diberikan rule-rule seperti di bawah ini.

Rule 1: min{0.6, 0.2} = 0.2
Rule 2: min{0.3, 0.35} = 0.3
Rule 3: min{0.45} = 0.45

Sehingga, bagian THEN dari rule 1, 2 dan 3 adalah 0.2, 0.3 dan 0.45 berturut-turut.

Langkah terakhir dalam proses inferensia adalah menentukan derajat kebenaran setiap istilah linguistik dari variabel linguistik output. Biasanya, baik *maximum* (MAX) maupun *sum* (SUM) dari derajat kebenaran dari rule dengan istilah linguistik yang sama dalam bagian THEN lalu dihitung untuk menentukan derajat kebenaran untuk setiap istilah linguistik dari variabel linguistik output.

Dari contoh sebelumnya, hasil perhitungan derajat kebenaran untuk istilah linguistik untuk toleransi variabel output menggunakan MAX adalah:

"risk neutral": max{0.2} = 0.2
"venturesome": max{0.3, 0.45} = 0.45

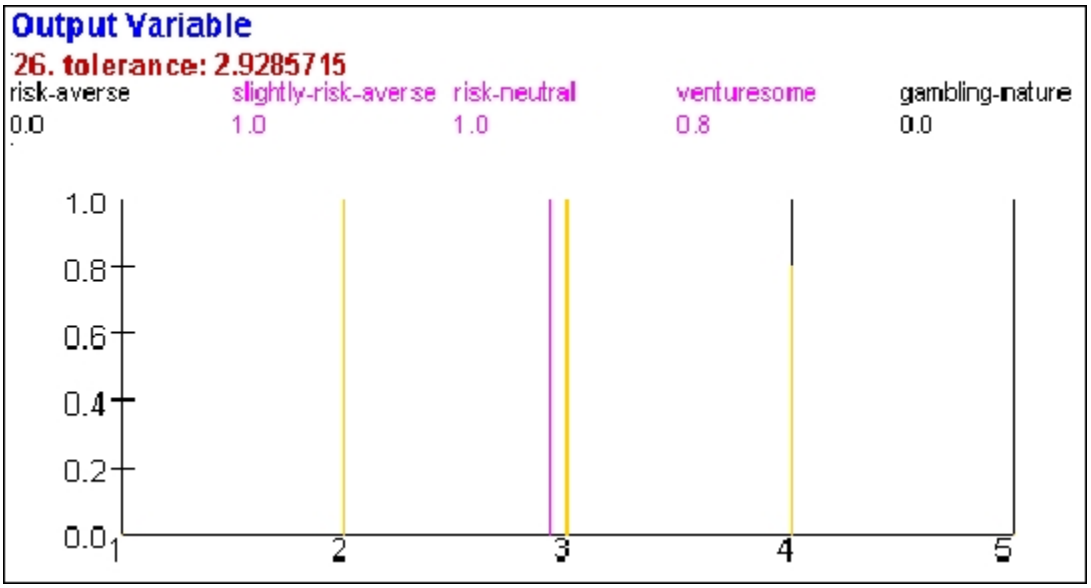
Ada banyak variasi metode agregasi dan komposisi yang tersedia dalam pelbagai literatur yang berbeda.

Defuzzification

Fase terakhir dari sistem pakar fuzzy adalah defuzzifikasi nilai-nilai linguistik dari variabel linguistik output kedalam nilai-nilai crisp. Teknik yang paling umum untuk defuzzifikasi adalah *Center-of-Maximum* (CoM) dan *Center-of-Area* (CoA).

CoM pertama kali menentukan nilai paling khas untuk setiap istilah linguistik untuk variabel linguistik output, dan lalu menghitung nilai crisp sebagai kompromi terbaik untuk nilai-nilai yang khas dan derajat keanggotaannya masing-masing.

Nilai paling khas dari setiap istilah linguistik adalah maksimum dari fungsi keanggotaan berturut-turut. Jika fungsi anggota memiliki interval maksimal, median dari interval yang diambil. Untuk toleransi variabel output dalam ilustrasi pada gambar 11.6 berikut ini, nilai paling khas untuk istilah linguistik "risk averse – penghindar resiko", "slightly risk averse – sedikit penghindar resiko", "risk neutral – resiko netral", "venturesome – suka berpetualang" dan "gambling nature – penjudi sejati" adalah 1, 2, 3, 4 dan 5 berturut-turut.



Gambar 11.6 Contoh defuzzification dari variabel linguistik output

Setelah mengidentifikasi nilai-nilai khasnya, nilai crisp dihitung sebagai kompromi terbaik untuk nilai-nilai khas yang ada dan derajat keanggotaannya masing-masing menggunakan *weighted mean* (rata-rata terbobot). Derajat masing-masing digunakan sebagai bobotnya. Dari contoh di atas, nilai crisp untuk toleransi variabel output dihitung sebagai (0.0*1 + 1.0*2 + 1.0*3 + 0.8*4 + 0.0*5)/(0.0 + 1.0 + 1.0 + 0.8 + 0.0) atau singkatnya didapat 2.9285715.

Metode umum lainnya, CoA, atau terkadang disebut dengan *Center-of-Gravity* (CoG), pertama-tama memotong fungsi anggota dari setiap istilah linguistik pada nilai yang berhubungan dengan nilai linguistik. *Superimposed area* (daerah yang melapiskan ke atas) di bawah setiap potongan fungsi anggota diseimbangkan untuk memberikan nilai kompromi. Kerugian dari teknik ini adalah adanya komputasi yang tinggi untuk daerah di bawah fungsi anggota.

Ada pelbagai variasi lain dalam penghitungan nilai crisp dari nilai-nilai linguistik. Mereka adalah *Mean-of-Maximum* (MoM), *Left-of-Maximum* (LoM) atau *Smallest-of-Maximum* (SoM), *Right-of-Maximum* (RoM) atau *Largest-of-Maximum* (LoM), dan *Bisector-of-Area* (BoA).

Bab 12 Aplikasi GA dan Fuzzy Set pada RDB (1)

Dalam bab ini akan dibahas aplikasi GA dan fuzzy set yang diterapkan pada database relasional, sehingga diharapkan dapat dipahami implementasi nyata GA dan fuzzy set dalam kehidupan kita sehari-hari. Pustaka yang digunakan adalah dari dari C.M. Huang [Huang02]; S.M. Chen dan M.S. Yeh [Chen97]; S.M. Chen dan H.H. Chen [Chen00]; K.S. Leung dan W. Lam [Leung98]; serta S.M. Chen dan C.M. Huang [Chen03].

Nama Aplikasi: **Pembangkitan Fuzzy Rule Terboboti dari Database Relasional untuk Estimasi Nilai Null**

Pada bab ini akan dibahas aplikasi GA (genetic algorithm – algoritma genetika) dan fuzzy set pada database yang memiliki hubungan relasional (RDB - Relational Database).

Aplikasi ini akan memperkirakan/menghasilkan nilai dari nilai null dari database relasional yang memiliki rule-rule fuzzy terboboti. Nilai null disini bisa jadi timbul dari kerusakan pada suatu record database akibat suatu kesalahan/bencana atau memang nilai tersebut belum terdefinisikan sebelumnya. Manfaat dari aplikasi ini adalah dapat mengembalikan kembali/memperkirakan nilai null tadi sehingga suatu database yang tadinya timpang karena ada nilai null-nya dapat didayagunakan lebih baik lagi seperti halnya database yang normal/tidak memiliki nilai null.

12.1 Konsep Dasar Fuzzy Set

Sebelumnya, kita ingat lagi konsep dasar dari fuzzy set yang telah kita pelajari pada bab 11 sebelumnya.

Fuzzy subset A dari semesta pembicaraan U dapat direpresentasikan dengan notasi berikut ini:

$$A = \mu_A(u_1) / u_1 + \mu_A(u_2) / u_2 + \ldots + \mu_A(u_n) / u_n \qquad (12.1)$$

Dimana μ_A adalah fungsi anggota dari fuzzy subset A, $\mu_A: U \rightarrow [0,1]$, and $\mu_A(u_i)$ mengindikasikan derajat keanggotaan dari u_i dalam fuzzy set A. Jika U adalah set yang kontinyu, maka fuzzy subset A dapat direpresentasikan kembali sebagai berikut:

$$A = \int_U \mu_A(u)/u, \quad u \in U \qquad (12.2)$$

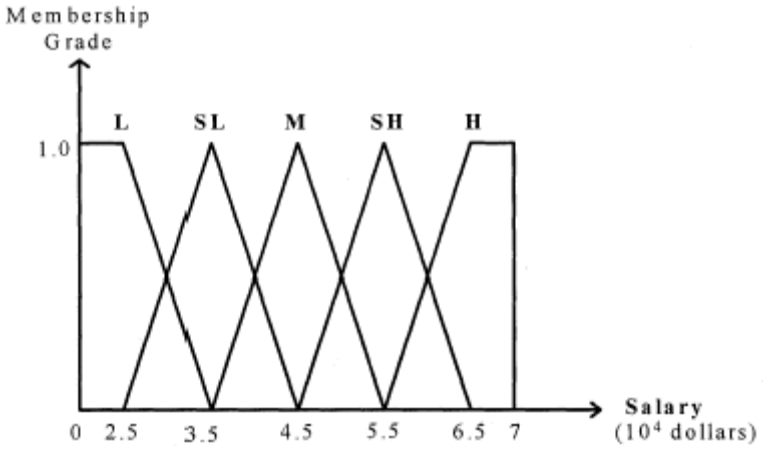
12.2 Fuzzy Set pada Database Relasional

Suatu istilah linguistik dapat direpresentasikan oleh fuzzy set dengan fungsi anggota. Dalam bab ini, fungsi anggota dari istilah linguistik "L", "SL", "M", "SH" dan "H" dari atribut "Salary" dan "Experience" dalam sistem database relasional adalah seperti yang ditulis dalam tabel 12.1 di bawah ini.

Tabel 12.1 Istilah linguistik yang dipakai

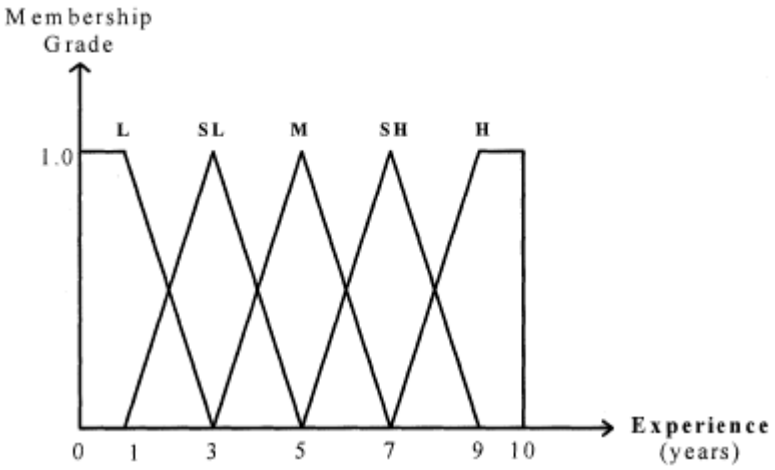
Notasi	Arti
L	Low
SL	Somewhat Low
M	Medium
SH	Somewhat High
H	High

Pada gambar 12.1 di bawah ini, terlihat derajat keanggotaan dari gaji seseorang, misal 48000 dolar, yang dimiliki oleh istilah linguistik "Medium" (M) dan "Somewhat High" (SH) adalah berturut-turut 0.7 dan 0.3.



Gambar 12.1 Fungsi anggota dari istilah linguistik dari atribut "Salary"

Sedangkan gambar 12.2 di bawah ini, terlihat derajat keanggotaan dari pengalaman seseorang, misal 4 tahun, yang dimiliki oleh istilah linguistik "Somewhat Low" (SL) dan "Medium" (M) adalah berturut-turut 0.5 dan 0.5.



Gambar 12.2 Fungsi anggota dari istilah linguistik dari atribut "Experience"

Kita dapat menggunakan matriks similaritas fuzzy untuk merepresentasikan relasi fuzzy. Diasumsikan bahwa variabel linguistik V memiliki istilah linguistik v_1, v_2, …, dan v_n, dan diasumsikan bahwa matriks similaritas (kemiripan) fuzzy digambarkan pada gambar 12.3 di bawah ini, dimana u_{ij} menyatakan derajat kemiripan diantara v_i dan v_j. Derajat kedekatan (closeness degree) diantara v_i dan v_j variabel linguistik V dinyatakan sebagai $CD_v(v_i,v_j)$, dimana $CD_v(v_i,v_j) = u_{ij}$, $1 \le i \le n$, dan $1 \le j \le n$. Jelas bahwa $CD_v(v_i,v_j) = 1$, dimana $1 \le i \le n$. Yang patut dicatat disini adalah, setiap elemen dalam matriks relasi fuzzy ditentukan oleh seorang pakar.

	v_1	v_2	⋯	v_n
v_1	1	u_{12}	⋯	u_{1n}
v_2	u_{21}	1	⋯	u_{2n}
⋮	⋮	⋮	⋱	⋮
v_n	u_{n1}	u_{n2}	⋯	1

Gambar 12.3 Matriks similaritas fuzzy

Selanjutnya, seorang pakar juga menentukan fungsi rangking untuk istilah linguistik dalam rangka merangking istilah linguistik. Diasumsikan bahwa variabel lingustik V mempunyai istilah linguistik v_i dan v_j, dan diasumsikan pula bahwa rangking dari istilah linguistik v_i lebih utama daripada rangking dari istilah linguistik v_j, maka urutan rangkind diantara v_i dan v_j dapat didefinisikan sebagai:

$$Rank(v_i) > Rank(v_j)$$

dimana $1 \le i \le n$, dan $1 \le j \le n$, dan $i \ne j$.

Rule Base

Rule base digunakan untuk menyatakan relasi dimana suatu atribut menentukan atribut yang lain.

Sebagai contoh, gambar 12.4 di bawah ini menunjukkan sekumpulan rule, termasuk juga di dalamnya bobot (weight) dari atribut, dimana semua rule dalam rule base ditentukan oleh seorang pakar, w_{ij} menyatakan bobot dari atribut A_j dari rule ke-i dalam rule base, $w_{ij} \in [0, 1]$, $1 \le i \le n$, dan $1 \le j \le n$.

Rule 1: IF $A_1 = a_{11}$ $(W = w_{11})$ AND $A_2 = a_{12}$ $(W = w_{12})$ AND ... AND $A_n = a_{1n}$ $(W = w_{1n})$ THEN $B_1 = t_1$
Rule 2: IF $A_1 = a_{21}$ $(W = w_{21})$ AND $A_2 = a_{22}$ $(W = w_{22})$ AND ... AND $A_n = a_{2n}$ $(W = w_{2n})$ THEN $B_2 = t_2$
⋮ ⋮
Rule m: IF $A_1 = a_{m1}$ $(W = w_{m1})$ AND $A_2 = a_{m2}$ $(W = w_{m2})$ AND ... AND $A_n = a_{mn}$ $(W = w_{mn})$ THEN $B_m = t_m$

Gambar 12.4 Rule base yang mengandung rule fuzzy terboboti

Sedangkan relasi yang ada pada database relasional kita, dituliskan dalam tabel 12.2 di bawah ini.

Tabel 12.2 Relasi pada database relasional

EMP-ID	Degree	Experience	Salary
S1	Ph.D.	7.2	63,000
S2	Master	2.0	37,000
S3	Bachelor	7.0	40,000
S4	Ph.D.	1.2	47,000
S5	Master	7.5	53,000
S6	Bachelor	1.5	26,000
S7	Bachelor	2.3	29,000
S8	Ph.D.	2.0	50,000
S9	Ph.D.	3.8	54,000
S10	Bachelor	3.5	35,000
S11	Master	3.5	40,000
S12	Master	3.6	41,000
S13	Master	10.0	68,000
S14	Ph.D.	5.0	57,000
S15	Bachelor	5.0	36,000
S16	Master	6.2	50,000
S17	Bachelor	0.5	23,000
S18	Master	7.2	55,000
S19	Master	6.5	51,000
S20	Ph.D.	7.8	65,000
S21	Master	8.1	64,000
S22	Ph.D.	8.5	70,000

Berdasarkan gambar 12.1 dan gambar 12.1 sebelumnya, nilai dari atribut "Degree" dan "Experience" pada tabel 12.2 di atas dapat difuzzifikasi sehingga didapat hasil seperti ditunjukkan pada tabel 12.3 di bawah ini.

Tabel 12.3 Hasil fuzzifikasi pada atribut "Degree" dan "Experience" pada database relasional

EMP-ID	Degree	Experience	Salary
S1	Ph.D./1.0	SH/0.9	63,000
S2	Master/1.0	L/0.5	37,000
S3	Bachelor/1.0	SH/1.0	40,000
S4	Ph.D./1.0	L/0.9	47,000
S5	Master/1.0	SH/0.75	53,000
S6	Bachelor/1.0	L/0.75	26,000
S7	Bachelor/1.0	SL/0.65	29,000
S8	Ph.D./1.0	L/0.5	50,000
S9	Ph.D./1.0	SL/0.6	54,000
S10	Bachelor/1.0	SL/0.75	35,000
S11	Master/1.0	SL/0.75	40,000
S12	Master/1.0	SL/0.7	41,000
S13	Master/1.0	H/1.0	68,000
S14	Ph.D./1.0	M/1.0	57,000
S15	Bachelor/1.0	M/1.0	36,000
S16	Master/1.0	SH/0.6	50,000
S17	Bachelor/1.0	L/1.0	23,000
S18	Master/1.0	SH/0.9	55,000
S19	Master/1.0	SH/0.75	51,000
S20	Ph.D./1.0	SH/0.6	65,000
S21	Master/1.0	H/0.55	64,000
S22	Ph.D./1.0	H/0.75	70,000

12.3 Derajat Kemiripan

Terdapat dua derajat kemiripan (degree of similarity) yang digunakan dalam aplikasi kita kali ini, yaitu:

- Derajat kemiripan diantara nilai-nilai dari atribute "Degree". Ini dituliskan dalam tabel 12.4 di bawah.
- Derajat kemiripan diantara nilai-nilai nonnumerik yang dijelaskan dalam gambar 12.5 berikut ini.

Tabel 12.4 Derajat kemiripan diantara nilai-nilai dari atribute "Degree"

	Bachelor	Master	Ph.D.
Bachelor	1	0.6	0.4
Master	0.6	1	0.6
Ph.D.	0.4	0.6	1

Rank(Bachelor) = 1

Rank(Master) = 2

Rank(Ph.D.) = 3

Gambar 12.5 Derajat kemiripan diantara nilai-nilai nonnumerik

12.4 Rumus-rumus yang Digunakan

Diasumsikan X adalah atribut nonnumerik.

Berdasarkan nilai $T_i.X$ dari atribute X pada tuple T_i dan nilai $T_j.X$ dari atribut X pada tuple T_j, dimana $i \neq j$, degree of closeness (derajat kedekatan) Closeness(T_i, T_j) diantara tuples T_i dan T_j dapat dihitung dengan persamaan (12.3) atau (12.4), dimana Weight(T_j.Degree) dan Weight(T_j.Experience) diartikan berturut-turut sebagai bobot dari atribut "Degree" dan "Experience", didapatkan dari nilai-nilai terfuzzifikasi dari atribut "Degree" dan "Experience" dari tuple T_j, yang semuanya diturunkan dari sebuah kromosom.

If Rank($T_i.X$) $\geq$ Rank($T_j.X$) then

$$\text{Closeness}(T_i, T_j) = \text{Similarity}(T_i.X, T_j.X) \times \text{Weight}(T_j.\text{Degree}) + \frac{T_i.Experience}{T_j.Experience} \times \text{Weight}(T_j.\text{Experience}) \quad (12.3)$$

If Rank($T_i.X$) < Rank($T_j.X$) then

$$\text{Closeness}(T_i, T_j) = 1/\text{Similarity}(T_i.X, T_j.X) \times \text{Weight}(T_j.\text{Degree}) + \frac{T_i.Experience}{T_j.Experience} \times \text{Weight}(T_j.\text{Experience}) \quad (12.4)$$

Dimana Similarity($T_i.X$, $T_j.X$) adalah derajat kemiripan diantara $T_i.X$ dan $T_j.X$, dan nilai-nilainya didapatkan dari matriks kemiripan fuzzy dari istilah-istilah linguistik dari atribut X yang didefinisikan oleh seorang pakar.

Kromosom-kromosom yang digunakan dapat digambarkan pada gambar 12.6 di bawah ini. Jumlah gen adalah 15, yang didapat dari kombinasi 5 buah istilah linguistik yang dipakai ("L", "SL", "M", "SH" dan "H") dan 3 buah gelar/degree yang ada (Bachelor, Master dan Ph.D.).

Nilai pecahan (real) yang nampai pada setiap gen diartikan sebagai bobot dari gelar/degree. Karena bobot total dari suatu atribut (misal B-H, yang berarti Bachelor-High) adalah 1, maka bobot dari pengalaman/experience dihitung dengan 1 dikurangi bobot degree (bobot experience = 1 – bobot degree).

Gene Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	Real value	Real value	Real value	Real value	Real value	Real value	Real value	Real value	Real value	Real value	Real value	Real value	Real value	Real value	Real value
	B-H	**M-H**	**P-H**	**B-SH**	**M-SH**	**P-SH**	**B-M**	**M-M**	**P-M**	**B-SL**	**M-SL**	**P-SL**	**B-L**	**M-L**	**P-L**

Gambar 12.6 Format kromosom

Nilai perkiraan (estimated value) "ET_i.Salary" dari atribut "Salary" dari tuple T_i dirumuskan sebagai berikut ini:

$$ET_i.Salary = T_i.Salary \times Closeness\ (T_i, T_j) \qquad (12.5)$$

Perkiraan kesalahan (estimated error) dari setiap tuple dirumuskan pada persamaan (12.6), dimana $Error_i$ adalah perkiraan kesalahan diantara nilai perkiraan ET_i.Salary dari atribut "Salary" dari tuple T_i dan nilai aktual T_i.Salary dari atribut "Salary" dari tuple T_i adalah:

$$Error_i = \frac{ET_i.Salary - T_i.Salary}{T_i.Salary} \qquad (12.6)$$

Diasumsikan Avg_Error adalah kesalahan perkiraan rata-rata (average estimated error) dari tuple-tuple berdasarkan kombinasi bobot (weight) dari atribut yang diturunkan dari kromosom, dirumuskan sebagai:

$$Avg\_Error = \frac{\sum_{i=1}^{n} Error_i}{n} \qquad (12.7)$$

Selanjutnya, kita bisa mendapatkan derajat kecocokan (fitness degree) dari kromosom ini seperti rumus berikut:

$$Fitness\ Degree = 1 - Avg\_Error \qquad (12.8)$$

Pada tabel 12.5 berikut ini disajikan contoh dari nilai null yang terjadi pada suatu database relasional.

Tabel 12.5 Contoh nilai null pada suatu database relasional

EMP-ID	Degree	Experience	Salary
S1	Ph.D.	7.2	63,000
S2	Master	2.0	37,000
S3	Bachelor	7.0	40,000
S4	Ph.D.	1.2	47,000
S5	Master	7.5	53,000
S6	Bachelor	1.5	26,000
S7	Bachelor	2.3	29,000
S8	Ph.D.	2.0	50,000
S9	Ph.D.	3.8	54,000
S10	Bachelor	3.5	35,000
S11	Master	3.5	40,000
S12	Master	3.6	41,000
S13	Master	10.0	68,000
S14	Ph.D.	5.0	57,000
S15	Bachelor	5.0	36,000
S16	Master	6.2	50,000
S17	Bachelor	0.5	23,000
S18	Master	7.2	55,000
S19	Master	6.5	51,000
S20	Ph.D.	7.8	65,000
S21	Master	8.1	64,000
S22	Ph.D.	8.5	Null

12.5 Contoh Kasus

Kasus dari suatu database yang memiliki nilai null disajikan pada tabel 12.5 di atas. Kemudian kita menyusun kromosom telah terBerdasar

Kromosom

Gambaran dari suatu kromosom yang akan kita proses adalah seperti gambar 12.7 di bawah ini.

Gene Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0.01	0.071	0.343	0.465	0.505	0.303	0.495	0.081	0.778	0.717	0.303	0.869	0.869	0.828	0.434
	B-H	M-H	P-H	B-SH	M-SH	P-SH	B-M	M-M	P-M	B-SL	M-SL	P-SL	B-L	M-L	P-L

Gambar 12.7 Contoh suatu kromosom

Gambar 12.7 tersebut dapat kita terjemahkan ke dalam 15 rule sebagai berikut ini:

Rule 1: **IF** Degree = Bachelor **AND** Experience = High, **THEN** the Weight of Degree = 0.010 **AND** the Weight of Experience = 0.99

Rule 2: **IF** Degree = Master **AND** Experience = High, **THEN** the Weight of Degree = 0.071 **AND** the Weight of Experience = 0.929

Rule 3: **IF** Degree = Ph.D. **AND** Experience = High, **THEN** the Weight of Degree = 0.343 **AND** the Weight of Experience = 0.657

Rule 4: **IF** Degree = Bachelor **AND** Experience = Somewhat High, **THEN** the Weight of Degree = 0.465 **AND** the Weight of Experience = 0.535

Rule 5: **IF** Degree = Master **AND** Experience = Somewhat High, **THEN** the Weight of Degree = 0.505 **AND** the Weight of Experience = 0.495

Rule 6: **IF** Degree = Ph.D. **AND** Experience = Somewhat High, **THEN** the Weight of Degree = 0.303 **AND** the Weight of Experience = 0.697

Rule 7: **IF** Degree = Bachelor **AND** Experience = Medium, **THEN** the Weight of Degree = 0.495 **AND** the Weight of Experience = 0.505

Rule 8: **IF** Degree = Master **AND** Experience = Medium, **THEN** the Weight of Degree = 0.081 **AND** the Weight of Experience = 0.919

Rule 9: **IF** Degree = Ph.D. **AND** Experience = Medium, **THEN** the Weight of Degree = 0.778 **AND** the Weight of Experience = 0.222

Rule 10: **IF** Degree = Bachelor **AND** Experience = Somewhat Low, **THEN** the Weight of Degree = 0.717 **AND** the Weight of Experience = 0.283

Rule 11: **IF** Degree = Master **AND** Experience = Somewhat Low, **THEN** the Weight of Degree = 0.303 **AND** the Weight of Experience = 0.697

Rule 12: **IF** Degree = Ph.D. **AND** Experience = Somewhat Low, **THEN** the Weight of Degree = 0.869 **AND** the Weight of Experience = 0.131

Rule 13: **IF** Degree = Bachelor **AND** Experience = Low, **THEN** the Weight of Degree = 0.869 **AND** the Weight of Experience = 0.131

Rule 14: **IF** Degree = Master **AND** Experience = Low, **THEN** the Weight of Degree = 0.828 **AND** the Weight of Experience = 0.172

Rule 15: **IF** Degree = Ph.D. **AND** Experience = Low, **THEN** the Weight of Degree = 0.434 **AND** the Weight of Experience = 0.566

Operasi Seleksi (Selection)

Setelah populasi baru dibangkitkan, kita dapat menghitung nilai rata-rata kecocokan (fitness value) dari populasi tersebut.

Diasumsikan bahwa kromosom 1, kromosom 2 dan kromosom 3 adalah tiga buah kromosom dari suatu populasi serta diasumsikan bahwa nilai fitness mereka berturut-turut adalah 0.975, 0.744 dan 0.480.

Selanjutnya, diasumsikan bahwa nilai rata-rata fitness dari populasi ini adalah 0.627, maka:

- Untuk kromosom 1: 0.975/0.627 = 1.55 ≅ 2
- Untuk kromosom 2: 0.744/0.627 = 1.18 ≅ 1
- Untuk kromosom 3: 0.480/0.627 = 0.77 ≅ 1

Maka kromosom 1 akan mendapat 2 posisi, kromosom 2 dan kromosom 3 akan mendapat 1 posisi, untuk berlanjut pada operasi pindah silang (crossover). Setelah kita cek semua kromosom dalam populasi, jika jumlah dari kromosom yang terseleksi pada saat operasi seleksi kurang dari jumlah kromosom dalam populasi, maka kita akan secara acak mengambil sembarang kromosom untuk mengisi posisi tersebut.

Operasi Pindah Silang (Crossover)

Dalam bab ini, angka probabilitas α terjadinya pindah silang kita set 1.0. Sehingga, setelah operasi seleksi, maka sejumlah kromosom dalam populasi akan melakukan operasi pindah silang.

Ini dilakukan sistem dengan cara mengambil secara acak 2 kromosom sebagai orang tua dan secara acak pula memilih titik pindah silang. Maka sistem akan melakukan operasi pindah silang pada 2 kromosom pada titik pindah silang tersebut untuk membangkitkan 2 anak. Ini digambarkan pada gambar 12.8 berikut ini.

Sebagai contoh, setelah operasi pindah silang, kromosom yang dihasilkan akan seperti gambar 12.9.

The First Chromosome — Crossover Point

Gene Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0.222	0.798	0.848	0.030	0.141	0.596	0.828	0.636	0.939	0.192	0.495	0.152	0.899	0.000	0.727
	B-H	**M-H**	**P-H**	**B-SH**	**M-SH**	**P-SH**	**B-M**	**M-M**	**P-M**	**B-SL**	**M-SL**	**P-SL**	**B-L**	**M-L**	**P-L**

The Second Chromosome — Crossover Point

Gene Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0.323	0.465	0.424	0.000	0.354	0.677	0.707	0.061	0.475	0.889	0.535	0.010	0.758	0.030	0.030
	B-H	**M-H**	**P-H**	**B-SH**	**M-SH**	**P-SH**	**B-M**	**M-M**	**P-M**	**B-SL**	**M-SL**	**P-SL**	**B-L**	**M-L**	**P-L**

Gambar 12.8 Kromosom sebelum operasi pindah silang

The First Chromosome

Gene Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0.222	0.798	0.848	0.030	0.141	0.596	0.828	0.636	0.939	0.192	0.495	0.152	0.899	0.030	0.030
	B-H	**M-H**	**P-H**	**B-SH**	**M-SH**	**P-SH**	**B-M**	**M-M**	**P-M**	**B-SL**	**M-SL**	**P-SL**	**B-L**	**M-L**	**P-L**

The Second Chromosome

Gene Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0.323	0.465	0.424	0.000	0.354	0.677	0.707	0.061	0.475	0.889	0.535	0.010	0.758	0.000	0.727
	B-H	**M-H**	**P-H**	**B-SH**	**M-SH**	**P-SH**	**B-M**	**M-M**	**P-M**	**B-SL**	**M-SL**	**P-SL**	**B-L**	**M-L**	**P-L**

Gambar 12.9 Kromosom setelah operasi pindah silang

Operasi Mutasi (Mutation)

Setelah operasi pindah silang, kromosom dalam populasi akan berlanjut ke operasi mutasi.

Ini dilakukan dengan cara sistem secara acak mengambil kromosom yang akan melakukan operasi mutasi, dan juga secara acak menentukan gen mana yang akan dikenai mutasi.

Harus diingat bahwa tidak setiap generasi akan melakukan operasi mutasi. Kita dapat

memilih angka probabilitas mutasi β, dimana β ∈[0, 1], untuk melakukan operasi mutasi.

Saat operasi pindah silang selesai, sistem membangkitkan bilang acak dalam jangkauan [0, 1]. Jika bilangan acak tadi kurang dari β, maka sistem akan melakukan operasi mutasi pada kromosom tertentu dalam populasi.

Sebagai contoh, diasumsikan bahwa terdapat suatu kromosom seperti ditunjukkan pada gambar 12.10. Lalu diasumsikan bahwa sistem secara acak memilih gen ketujuh dari kromosom untuk melakukan operasi mutasi, maka sistem secara acak memberi gen terpilih tadi nilai pecahan baru (contoh, 0.624) sehingga kromosom akan menjadi seperti gambar 12.11.

Gene Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0.01	0.071	0.343	0.465	0.505	0.303	0.495	0.081	0.778	0.717	0.303	0.869	0.869	0.828	0.434
	B-H	M-H	P-H	B-SH	M-SH	P-SH	B-M	M-M	P-M	B-SL	M-SL	P-SL	B-L	M-L	P-L

Gambar 12.10 Kromosom sebelum operasi mutasi

Gene Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0.01	0.071	0.343	0.465	0.505	0.303	0.624	0.081	0.778	0.717	0.303	0.869	0.869	0.828	0.434
	B-H	M-H	P-H	B-SH	M-SH	P-SH	B-M	M-M	P-M	B-SL	M-SL	P-SL	B-L	M-L	P-L

Gambar 12.11 Kromosom setelah operasi mutasi

Untuk setiap generasi, algoritma genetika akan terus melakukan operasi-operasi seleksi, pindah silang dan mutasi. Setelah algoritma genetikanya konvergen, maka kita akan mendapatkan kombinasi terbaik dari bobot-bobot atributnya. Jika ada tuple dalam database yang memiliki nilai null, maka kita dapat mengestimasi nilai null dengan menggunakan bobot-bobot atribut yang dibangkitkan oleh algoritma genetika.

Untuk lebih jelas lagi kita aplikasikan algoritma genetika dengan parameter: ukuran populasi = 60, jumlah generasi = 300, angka pindah silang = 1.0 dan angka mutasi = 0.2; dan didapat kromosom terbaik yang mengandung kombinasi bobot dari atribut "Degree" dan "Experience" seperti ditunjukkan pada gambar 12.12 di bawah ini.

Gene Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0.010	0.071	0.343	0.465	0.505	0.303	0.495	0.081	0.778	0.717	0.303	0.869	0.869	0.828	0.434
	B-H	M-H	P-H	B-SH	M-SH	P-SH	B-M	M-M	P-M	B-SL	M-SL	P-SL	B-L	M-L	P-L

Gambar 12.12 Kromosom terbaik yang didapat

Dalam rangka mengestimasi nilai null dari atribut "Salary" dari tuple T_{22} (lihat tabel 12.5) dengan EMP-ID adalah S22, kita harus menemukan tuple yang paling mendekati tuple T_{22}.

Proses komputasi derajat kedekatan (degree of closeness) diantara 2 tuple dapat diilustrasikan sebagai berikut. Kita lihat lagi tuple T_1 yang ditunjukkan oleh tabel 12.5 dengan EMP-ID adalah S1 sebagai contoh.

Kita dapat melihat bahwa nilai dari atribut "Degree" dan "Experience" dari tuple T_1 berturut-turut adalah "Ph.D." (P) dan "7.2".

Kita juga dapat melihat bahwa derajat similaritas (degree of similarity) diantara nilai dari atribut "Degree" dari tuple T_1 dan tuple T_{22} adalah 1 (yaitu, Similarity(T_{22}.Degree, T_1.Degree) = 1). Maka, berdasarkan nilai dari atribut "Experience" dari tuple T_1 dan tuple T_{22}, kita dapatkan:

$$\frac{T_{22}.Experience}{T_1.Experience} = \frac{8.5}{7.2} = 1.180$$

Dari tabel 12.3, kita dapat melihat bahwa nilai terfuzzifikasi dari atribut "Experience" dari tuple T_1 dengan EMP-ID = S1 adalah "SH" (Somewhat High). Kemudian, kita pilih nilai dari gen keenam berlabel "P-SH" dari gambar 12.12 (yaitu: 0.303). Ini artinya bahwa bobot dari atribut "Degree" dan "Experience" adalah 0.303 dan 0.697 (didapat dari 1 – 0.303). Berdasarkan persamaan (12.4), derajat kedekatan diantara tuple T_{22} dan T_1 dapat dihitung sebagai berikut:

$$\begin{aligned} \text{Closeness}(T_{22}, T_1) &= 1 \times 0.303 + 1.180 \times (1 - 0.303) \\ &= 0.303 + 0.802 \\ &= 1.125 \end{aligned}$$

Setelah derajat kedekatan dari semua tuple dibandingkan dengan tuple T_{22} telah dihitung, kita dapat melihat bahwa tuple T_{20} adalah yang terdekat dengan tuple T_{22} (yaitu derajat kedekatan Closeness(T_{22}, T_{20}) diantara tuple T_{22} dan T_{20} terdekat ke nilai 1.0), dimana derajat kedekatan Closeness(T_{22}, T_{20}) dihitung sebagai berikut:

$$\text{Closeness}(T_{22}, T_{20}) = 1 \times 0.303 + \frac{8.5}{7.8} \times (1 - 0.303)$$

$$= 0.303 + 1.0897 \times 0.697$$

$$\cong 1.0625$$

Berdasarkan persamaan (12.5), dengan mengalikan nilai dari atribut "Salary" dari tuple T_{20} dan derajat kedekatan Closeness(T_{22}, T_{20}) diantara tuple T_{20} dan T_{22}, nilai estimasi "ET_{22}.Salary" dari atribut "Salary" dari tuple T_{22} dapat dihitung sebagai:

$$ET_{22}.Salary = 65000 \times 1.0625$$

$$= 69065.83$$

Dari tabel 12.2, kita dapat melihat bahwa nilai aktual untuk atribut "Salary" dari tuple T_{22} adalah 70000. Berdasarkan persamaan (12.6), kita menghitung perkiraan kesalahan (estimated error) dari nilai estimasi dari atribut "Salary" dari tuple T_{22} sebagai berikut:

$$Error_{22} = \frac{69065.83 - 70000}{70000}$$

$$= - \frac{934.17}{70000}$$

$$\cong - 0.01$$

Dengan mengulangi proses yang sama, perkiraan gaji dan perkiraan kesalahan dari setiap tuple dalam database pada tabel 12.2 dapat diperoleh dan ditunjukkan dalam tabel 12.6 di bawah ini.

Tabel 12.6 Perkiraan gaji dan perkiraan kesalahan untuk setiap tuple

EMP-ID	Degree	Experience	Salary	Salary (Estimated)	Estimated Error
S1	Ph.D.	7.2	63,000	61,515.00	-0.024
S2	Master	2.0	37,000	36,967.44	-0.001
S3	Bachelor	7.0	40,000	40,634.14	0.016
S4	Ph.D.	1.2	47,000	46,873.66	-0.003
S5	Master	7.5	53,000	56,134.37	0.059
S6	Bachelor	1.5	26,000	26,146.40	0.006
S7	Bachelor	2.3	29,000	27,822.08	-0.041
S8	Ph.D.	2.0	50,000	50,067.20	0.001
S9	Ph.D.	3.8	54,000	53,958.94	-0.001
S10	Bachelor	3.5	35,000	35,152.00	0.004
S11	Master	3.5	40,000	40,206.19	0.005
S12	Master	3.6	41,000	40,796.57	-0.005
S13	Master	10.0	68,000	68,495.74	0.007
S14	Ph.D.	5.0	57,000	56,240.72	-0.013
S15	Bachelor	5.0	36,000	34,277.54	-0.048
S16	Master	6.2	50,000	49,834.85	-0.003
S17	Bachelor	0.5	23,000	23,722.40	0.031
S18	Master	7.2	55,000	51,950.6	-0.055
S19	Master	6.5	51,000	51,197.58	0.004
S20	Ph.D.	7.8	65,000	64,813.75	-0.003
S21	Master	8.1	64,000	60,853.28	-0.049
S22	Ph.D.	8.5	70,000	69,065.83	-0.013
Average Estimated Error					0.018

Dari tabel 12.6 juga dapat kita lihat bahwa rata-rata perkiraan kesalahan adalah 0.018.

Jelas bahwa terdapa parameter dalam algoritma genetika, seperti ukuran populasi, angka pindah silang, angka mutasi, jumlah generasi dan lain-lain. Dalam bab ini, kita estimasi nilai dari atribut “Salary” dari setiap tuple seperti yang telah ditunjukkan pada tabel 12.2 dengan mempertimbangkan 4 situasi dengan 4 parameter (yaitu ukuran populasi, jumlah generasi, angka mutasi dan angka pindah silang), dimana rata-rata perkiraan kesalahan untuk situasi yang berbeda ditunjukkan pada tabel 12.7 di bawah ini.

Tabel 12.7 Rata-rata perkiraan kesalahan pada parameter berbeda untuk algoritma genetika

Size of Population	Number of Generations	Crossover Rate	Mutation Rate	Average Estimated Error
30	100	1.0	0.1	0.036
40	150	1.0	0.1	0.032
50	200	1.0	0.2	0.027
60	300	1.0	0.2	0.018

Dari tabel 12.7 di atas, kita dapat melihat bahwa kromosom terbaik dengan nilai kecocokan (fitness) terbesar terjadi pada situasi keempat (dengan parameter: ukuran populasi = 60, jumlah generasi = 300, angka pindah silang = 1.0 dan angka mutasi = 0.2), dimana kromosom terbaik mengindikasikan kombinasi terbaik dari bobot atribut "Degree" dan "Experience" seperti telah ditunjukkan pada gambar 12.12.

Hasil percobaan diatas berdasarkan metode yang diajukan oleh S.M. Chen dan C.M. Huang [Chen03] ini, ternyata dapat memberikan akurasi estimasi rata-rata yang lebih tinggi dari metode yang diajukan oleh S.M. Chen dan M.S. Yeh [Chen97] serta S.M. Chen dan H.H. Chen [Chen00].

Bab 13 Aplikasi GA dan Fuzzy Set pada RDB (2)

Dalam bab ini akan dibahas aplikasi GA dan fuzzy set yang diterapkan pada database relasional seperti halnya pada bab 12, namun disini relasi ketergantungan diantara atribut-atributnya adalah negatif bukan positif seperti di bab 12. Diharapkan dapat dipahami implementasi nyata GA dan fuzzy set dalam kehidupan kita sehari-hari untuk relasi ketergantungan negatif diantara atribut-atributnya yang terlibat dalam sistem. Pustaka yang digunakan adalah dari C.M. Huang [Huang02]; S.M. Chen dan M.S. Yeh [Chen97]; S.M. Chen dan H.H. Chen [Chen00]; K.S. Leung dan W. Lam [Leung98]; serta S.M. Chen dan C.M. Huang [Chen03].

Nama Aplikasi: **Pembangkitan Fuzzy Rule Terboboti dari Database Relasional untuk Estimasi Nilai Null dengan Relasi Ketergantungan Negatif diantara Atribut-atributnya**

Pada bab ini akan dibahas aplikasi GA (genetic algorithm – algoritma genetika) dan fuzzy set pada database yang memiliki hubungan relasional (RDB - Relational Database) seperti halnya bab 12, namun dengan perbedaan bahwa diantara atribut-atributnya mereka memiliki relasi ketergantungan negatif.

13.1 Database Relasional yang Dipakai

Sebagai contoh kasus disini adalah database dari sebuah persewaan mobil dengan nama Benz Secondhand Cars.

Relasi ketergantungan negatif diantara atributnya adalah antara tahun pembuatan mobil sewa (year) dan harga sewa (price).

Semakin lama tahun pembuatan mobil (artinya semakin lama mobil itu) yang disewakan maka akan semakin murah harga sewanya, demikian pula sebaliknya semakin muda tahun pembuatan (artinya semakin baru mobil itu) maka akan semakin mahal harga sewanya.

Tabel 13.1 menjelaskan relasi negatif diantara atribut tahun pembuatan mobil dan harga sewanya disamping atribut-atribut yang lain.

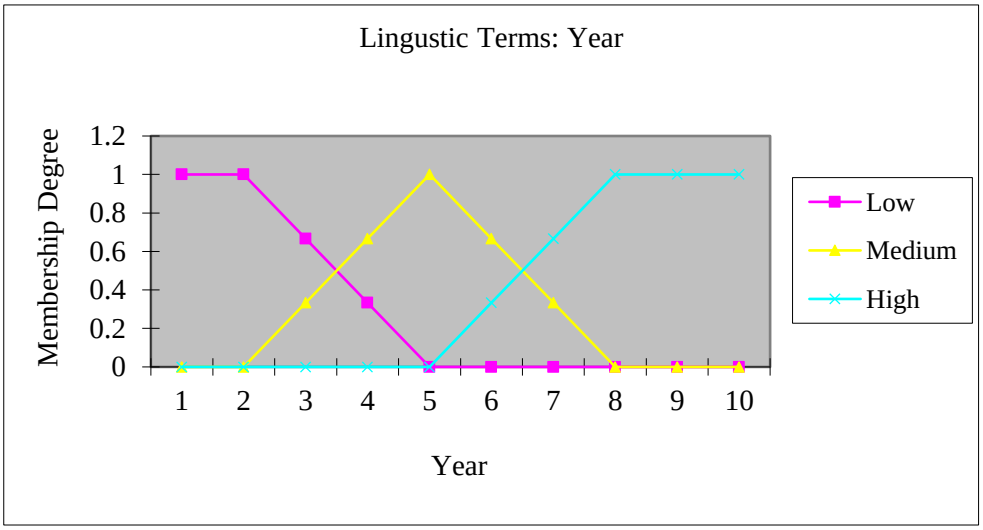
Tabel 13.1 Database relasional Benz Secondhand Cars

Car-ID	Style	Year	C.C.	Price*
1	E	10	2300	550000
2	S	3	3200	2520000
3	C	7	2800	790000
4	S	10	5000	1180000
5	S	10	3200	880000
6	E	2	2400	1650000
7	S	5	2800	1890000
8	S	6	3200	1490000
9	E	3	2400	1550000
10	C	2	2400	1630000
11	E	3	2400	1530000
12	E	1	2400	1890000
13	S	5	3200	1680000
14	E	5	2300	1230000
15	E	9	2200	760000
16	E	9	2000	530000
17	C	8	1800	580000
18	E	6	3200	1350000
19	S	7	6000	2150000
20	S	2	3200	3200000

Simbol "*" di atas artinya bahwa atribut ini memiliki relasi ketergantungan negatif ke atribut yang akan kita estimasi nilai null-nya (yaitu atribut Year).

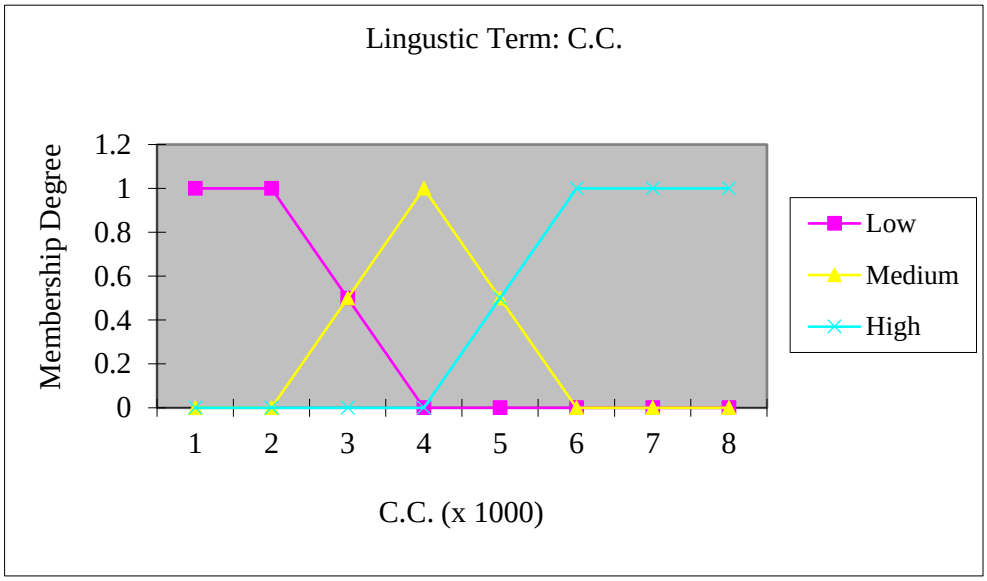
13.2 Fungsi Anggota dari Istilah Linguistik yang Dipakai

Fungsi anggota dari istilah linguistik pada atribut "Year" dapat digambarkan pada gambar 13.1 di bawah ini.



Gambar 13.1 Fungsi anggota dari istilah linguistik pada atribut "Year"

Sedangkan fungsi anggota dari istilah linguistik pada atribut "C.C." dapat digambarkan pada gambar 13.2 di bawah ini.



Gambar 13.2 Fungsi anggota dari istilah linguistik pada atribut "C.C."

13.3 Relasi Terfuzzifikasi

Relasi pada database relasional yang telah terfuzzifikasi ditampilkan pada tabel 13.2 di bawah ini.

Tabel 13.2 Relasi terfuzzifikasi pada database relasional Benz Secondhand Cars

Car-ID	Style	Year	C.C.	Price*	Year (Fuzzified)	C.C. (Fuzzied)
1	E	10	2300	550000	HIGH/1.000	MEDIUM/0.650
2	S	3	3200	2520000	LOW/0.667	MEDIUM/0.900
3	C	7	2800	790000	HIGH/0.667	MEDIUM/0.900
4	S	10	5000	1180000	HIGH/1.000	HIGH/1.000
5	S	10	3200	880000	HIGH/1.000	MEDIUM/0.900
6	E	2	2400	1650000	LOW/1.000	MEDIUM/0.700
7	S	5	2800	1890000	MEDIUM/1.000	MEDIUM/0.900
8	S	6	3200	1490000	MEDIUM/0.667	MEDIUM/0.900
9	E	3	2400	1550000	LOW/0.667	MEDIUM/0.700
10	C	2	2400	1630000	LOW/1.000	MEDIUM/0.700
11	E	3	2400	1530000	LOW/0.667	MEDIUM/0.700
12	E	1	2400	1890000	LOW/1.000	MEDIUM/0.700
13	S	5	3200	1680000	MEDIUM/1.000	MEDIUM/0.900
14	E	5	2300	1230000	MEDIUM/1.000	MEDIUM/0.650
15	E	9	2200	760000	HIGH/1.000	MEDIUM/0.600
16	E	9	2000	530000	HIGH/1.000	LOW/0.500
17	C	8	1800	580000	HIGH/1.000	LOW/0.600
18	E	6	3200	1350000	MEDIUM/0.667	MEDIUM/0.900
19	S	7	6000	2150000	HIGH/0.667	HIGH/1.000
20	S	2	3200	3200000	LOW/0.667	MEDIUM/0.900

13.4 Format Gen pada Kromosom

Format gen pada kromosom yang digunakan dalam aplikasi ini ditampilkan pada tabel 13.3 di bawah ini.

Tabel 13.3 Format gen pada Kromosom yang digunakan

Style	Year	CC
Bobot dari atribut Style	Bobot dari atribut Year	Bobot dari atribut CC

Sedangkan setiap kromosom merepresentasikan kombinasi dari bobot-bobot atributnya, dan ia merupakan sebuah string dari bobot-bobot atributnya tersebut. Kromosom ini yang nantinya akan digunakan untuk mengestimasi nilai-nilai null. Gambarannya ditampilkan pada gambar 13.3 di bawah ini.

Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC
0.455	0.495*	0.050	0.616	0.141*	0.243	0.101	0.586*	0.313	0.384	0.051	0.565*	0.566	0.253*	0.182	0.808	0.071	0.121*	0.960	0.040	0.000	0.101	0.505	0.394	0.111	0.242	0.647*
C-Low-Low			C-Low-Medium			C-Low-High			C-Medium-Low			C-Medium-Medium			C-Medium-High			C-High-Low			C-High-Medium			C-High-High		

Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC
0.505	0.131*	0.364	0.636	0.051*	0.313	0.717	0.030*	0.253*	0.717	0.212*	0.071	0.616	0.283*	0.101*	0.364	0.071*	0.566*	0.899	0.030*	0.071	0.394	0.596*	0.010*	0.687	0.192	0.121*
E-Low-Low			E-Low-Medium			E-Low-High			E-Medium-Low			E-Medium-Medium			E-Medium-High			E-High-Low			E-High-Medium			E-High-High		

Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC	Style	Year	CC
0.374	0.131	0.495	0.111	0.616*	0.273*	0.000	0.707*	0.293*	0.697	0.121	0.182*	0.242	0.515	0.243*	0.505	0.051	0.243*	1.000	0.000	0.000	0.273	0.657	0.071	0.515	0.010*	0.475
S-Low-Low			S-Low-Medium			S-Low-High			S-Medium-Low			S-Medium-Medium			S-Medium-High			S-High-Low			S-High-Medium			S-High-High		

Gambar 13.3 Kromosom dan gen-gen yang membentuknya

13.5 Menghitung Nilai Fitness Kromosom

Derajat kemiripan (degree of similarity) diantara nilai-nilai atribut Style dapat dilihat pada tabel 13.4 di bawah ini.

Tabel 13.4 Derajat kemiripan pada nilai-nilai dari atribut Style

	C	E	S
C	1	0.6	0.4
E	0.6	1	0.6
S	0.4	0.6	1

Sedangkan derajat kemiripan (degree of similarity) $Sim(T_{i.x}, T_{j.x})$ diantara nilai dari atribut X pada tuple T_i dan nilai dari atribut X pada tuple T_j ditunjukkan sebagai berikut:

$$Sim(T_{i.x}, T_{j.x}) = \begin{cases} Similarity(T_i.X, T_j.X), if\ \ Rank(T_i.X) < Rank(T_j.X) \\ \dfrac{1}{Similarity(T_i.X, T_j.X)}, otherwise \end{cases} \quad (13.1)$$

Ranking dari 3 nilai non-numerik "C", "E", "S" adalah:

Rank(C) = 1

Rank(E) = 2

Rank(S.) = 3

Ada 4 case (kasus) yang harus dipertimbangkan untuk mengambil bobot (weight) dari atribut untuk menghitung derajat kedekatan (degree of closeness) $Closeness(T_i, T_j)$ diantara tuples T_i dan T_j yaitu:

Case 1: Jika tak ada tanda "*" yang muncul dalam bobot dari atribut "Year" and "C.C." yang ada dalam suatu gen pada kromosom, maka:

$$Closeness(T_i, T_j) = Sim(T_i.Style, T_j.Style) \times Weight(T_i.Style) + \frac{T_i.Year}{T_j.Year}$$

$$\times Weight(T_i.Year) + \frac{T_i.CC}{T_j.CC} \times Weight(T_i.CC) \quad (13.2)$$

Case 2: Jika ada tanda "*" muncul dalam bobot dari atribut "Year" dan tak ada tanda "*" yang muncul dalam bobot dari atribut "C.C." yang ada dalam suatu gen pada kromosom, maka:

$$Closeness(T_i, T_j) = Sim(T_i.Style, T_j.Style) \times Weight(T_i.Style) + \frac{T_j.Year}{T_i.Year}$$

$$\times Weight(T_i.Year) + \frac{T_i.CC}{T_j.CC} \times Weight(T_i.CC) \quad (13.3)$$

Case 3: Jika tak ada tanda "*" yang muncul dalam bobot dari atribut "Year" dan ada tanda "*" muncul dalam bobot dari atribut "C.C." yang ada dalam suatu gen pada kromosom, maka:

$$\text{Closeness}(T_i, T_j) = \text{Sim}(T_i.\text{Style}, T_j.\text{Style}) \times \text{Weight}(T_i.\text{Style}) + \frac{T_i.Year}{T_j.Year}$$

$$\times \text{Weight}(T_i.\text{Year}) + \frac{T_j.CC}{T_i.CC} \times \text{Weight}(T_i.\text{CC}) \qquad (13.4)$$

Case 4: Jika ada tanda "*" muncul baik pada bobot dari atribut "Year" dan dalam bobot dari atribut "C.C." yang ada dalam suatu gen pada kromosom, maka:

$$\text{Closeness}(T_i, T_j) = \text{Sim}(T_i.\text{Style}, T_j.\text{Style}) \times \text{Weight}(T_i.\text{Style}) + \frac{T_j.Year}{T_i.Year}$$

$$\times \text{Weight}(T_i.\text{Year}) + \frac{T_j.CC}{T_i.CC} \times \text{Weight}(T_i.\text{CC}) \qquad (13.5)$$

Nilai estimasi "ET_i.Price" dari atribut "Price" pada tuple T_i:

$$ET_i.\text{Price} = T_j.\text{Price} \times \text{Closeness}(T_i, T_j) \qquad (13.6)$$

Estimasi kesalahan diantara nilai estimasi "ET_i.Price" dan nilai aktual "T_i.Price" dari atribut "Price" pada tuple T_i

$$\text{Error}_i = \frac{ET_i.\text{Price} - T_i.\text{Price}}{T_i.\text{Price}} \qquad (13.7)$$

Diasumsikan Avg_Error menyatakan rata-rata estimasi kesalahan dari tuple-tuple yang dihitung dengan bobot-bobot yang muncul dalam gen pada suatu kromosom, dimana

$$\text{Avg\_Error} = \frac{\sum_{i=1}^{n} Error_i}{n} \qquad (13.8)$$

Maka, kita dapat memperoleh nilai kecocokan (fitness) dari suatu kromosom yang dirumuskan sebagai:

$$\text{Fitness Degree} = 1 - \text{Avg\_Error} \quad (13.9)$$

Operasi Seleksi

Jika populasi baru dibangkitkan, pertama kali hitung nilai fitness dari setiap kromosom.

Saat melakukan operasi seleksi, sistem akan secara acak mengambil 2 kromosom dari populasi baru.

Kromosom dengan nilai fitness yang lebih besar dapat terus hidup dan menyiapkan diri untuk melakukan operasi pindah silang.

Ulangi proses ini sampai sejumlah kromosom dalam populasi baru tersebut terpenuhi jumlah yang diinginkan (sesuai dengan populasinya).

Operasi Pindah Silang

Angka pindah silang = 1.

Kromosom diperlakukan sebagai sebuah kubus.

Secara acak memilih 2 kromosom sebagai orang tua untuk menghasilkan 2 buah anak (offspring).

Titik pindah silang terdiri dari 3 titik (Style, Year, CC).

Operasi Mutasi

Sistem secara acak membangkitkan 2 bilangan yang merepresentasikan berapa anyak kromosom dan berapa banyak gen dari kromosom-kromosom tersebut yang akan bermutasi.

Kemudian, secara acak sistem mengambil kromosom-kromosom untuk melakukan operasi mutasi, dan juga secara acak menentukan gen mana yang dipilih dari kromosom terpilih untuk melakukan operasi mutasi.

Angka mutasi dapat diatur oleh pengguna.

Kromosom diperlakukan sebagai sebuah kubus.

Sistem secara acak memberikan gen terpilih nilai pecahan baru, dan gen baru akan menggantikan gen lama.

Kapan akan Berhenti?

Sistem menyimpan rata-rata nilai fitness dari semua kromosom dalam populasi setiap evolusi yang terjadi.

Disaat perbedaan diantara nilai fitness dari suatu generasi dan generasi terakhir sebelumnya lebih kecil dari suatu nilai batasan (threshold value), sistem menghentikan proses evolusi. Contoh nilai batasan, misalkan 0.001.

Menterjemahkan Kembali Kromosom Terbaik

Sistem mengambil kromosom dengan nilai fitness terbesar dari populasi pada generasi terakhir disaat sistem telah konvergen.

Lalu sistem akan menyimpan bobot-bobot dari atribut yang muncul pada kromosom terbaik itu, kedalam rule base untuk mengestimasi nilai null.

13.6 Contoh Kasus

Dimisalkan database relasional yang memiliki nilai null seperti tabel 13.5 di bawah ini akan diestimasi nilai null-nya.

Tabel 13.5 Contoh nilai null pada database Benz Secondhand Cars

Car-ID	Style	Year	C.C.	Price*
1	E	10	2300	Null
2	S	3	3200	2520000
3	C	7	2800	790000
4	S	10	5000	1180000
5	S	10	3200	880000
6	E	2	2400	1650000
7	S	5	2800	1890000
8	S	6	3200	1490000
9	E	3	2400	1550000
10	C	2	2400	1630000
11	E	3	2400	1530000

12	E	1	2400	1890000
13	S	5	3200	1680000
14	E	5	2300	1230000
15	E	9	2200	760000
16	E	9	2000	530000
17	C	8	1800	580000
18	E	6	3200	1350000
19	S	7	6000	2150000
20	S	2	3200	3200000

Rata-rata estimasi kesalahan dimana parameter-parameternya berbeda (ukuran populasi = 30, angka pindah silang = 1, angka mutasi = 0.05, jumlah generasi = 30, dan setelah dijalankan 10 kali)

Number of training instances and testing instances / Average estimated Error / Methods	50 kali latihan dan 150 kali testing	100 kali latihan dan 100 kali testing	150 kali latihan dan 50 kali testing
Metode Huang-and-Chen [Chen00]	0.0993	0.1033	0.1003
Metode ini [Huang02]	0.0969	0.0938	0.0876

Hasil percobaan diatas berdasarkan metode yang diajukan oleh C.M. Huang [Huang02] ini, ternyata dapat memberikan akurasi estimasi rata-rata yang lebih tinggi dari metode yang diajukan oleh S.M. Chen dan H.H. Chen [Chen00].

Bab 14 Aplikasi Fuzzy Set pada Intelligent Car Agent

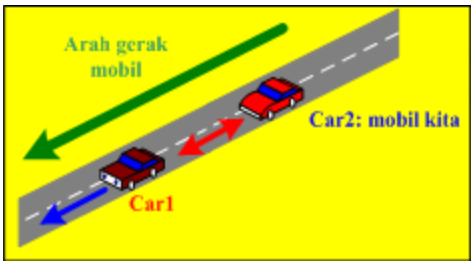
Dalam bab ini akan dibahas mengenai penerapan fuzzy set pada prototipe aplikasi nyata yaitu agent yang ditempatkan di sebuah mobil untuk menjaga jarak dan kecepatan mobil ini dengan mobil di depannya. Pustaka yang digunakan adalah Stuart J. Russell dan Peter Norvig [Rus03] serta hasil riset penulis sendiri.

14.1 Tujuan

Dalam bab ini akan diimplementasikan suatu agen (yaitu agen mobil) yang cerdas, dimana kecerdasannya adalah dapat mengatur jarak antara dirinya sendiri dengan mobil di depannya.

Kecerdasan yang didapat adalah berkat penerapan fuzzy set pada agen mobil cerdas ini (intelligent car agent).

Gambaran dari agen yang ingin kita terapkan dapat dilihat pada gambar 14.1 di bawah ini, dimana kita ada di mobil Car2 dan berusaha menjaga jarak dengan mobil Car1 (agen lain).



Gambar 14.1 Gambaran intelligent car agent

14.2 Implementasi Agen

Agen: entitas komputasi yang melaksanakan delegasi tugas dari pengguna (user) secara otonomi.

Karakteristik suatu agen adalah:

1. Delegasi – dapat mewakili pengguna dan menjalankan tugas yang diperintahkan kepadanya
2. Kemampuan berkomunikasi – dapat berkomunikasi dengan agen yang lain

3. Otonomi – berdiri sendiri sebagai konsekuensi dari nomor 1 dan 2 di atas, jadi ia memiliki perilaku yang berdiri sendiri, lepas dari campur tangan langsung pengguna (otonomi).
4. Monitoring – mampu untuk mempersepsikan (mengindra) lingkungannya
5. Aktuasi (actuation) – mampu untuk beraksi dalam lingkungannya
6. Cerdas (intelligence) – reaktif (sederhana) atau kognitif (lebih kompleks) dan dapat berevolusi dalam lingkungannya

Kita akan implementasikan agen dalam rangka mendukung pengguna dalam hal mengatur/mengelola kecepatan (speed) dan jarak (distance) diantara kita dan mobil di depan kita.

Agen mobil kita didukung oleh teori fuzzy set dan bekerja secara otonomi, berpikir dengan sendirinya namun tetap dapat dimonitor dengan baik.

Sebagian besar aktifitas pengguna dapat didelegasikan pada agen mobil ini, dan komunikasi untuk mewujudkan hal ini dilakukan dengan pelayanan GPS (Global Position System).

Pengguna dapat dengan mudah mengaktuasi (menggerakkan/menjalankan) agen ini, semudah ia mematikannya.

14.3 Pengetahuan dari Intelligent Car agent

Dengan mengingat teori fuzzy set yang telah diberikan sebelumnya, maka dalam istilah teknologi agen, adalah menarik untuk mengimplementasikan beberapa teori tersebut ke dalam dunia nyata (misalnya: bahasa pemrograman).

Knowledge (pengetahuan) dalam agen mobil kita, didefinisikan seperti gambar 14.2 di bawah ini.

1. IF Distance is Narrow AND Speed is Slow THEN Command=KeepSpeed
2. IF Distance is Narrow AND Speed is Fast THEN Command=SlowDown
3. IF Distance is Wide AND Speed is Slow THEN Command=SpeedUp
4. IF Distance is Wide AND Speed is Fast THEN Command=KeepSpeed

Gambar 14.2 Pengetahuan untuk intelligent car agent

Penjelasan:

- 'Narrow' adalah diantara 0 ~ 30 m
- 'Wide' adalah diantara 10 ~ dan lebih dari 30 m
- 'Slow' adalah diantara 0 ~ 70 km/jam
- 'Fast' adalah diantara 30 ~ dan lebih dari 70 km/jam
- Semua indeks maksimum diset ke nilai MaxIdx (misal, 100) untuk membuatnya lebih mudah.
- Perlu diingat bahwa indeks adalah diantara 0 ~ MaxIdx
- Namun jumlah semua indeks adalah MaxIdx + 1
- 'KeepSpeed' adalah diantara -10 ~ 10 km/jam2
- 'SlowDown' adalah diantara -10 ~ 0 km/jam2
- 'SpeedUp' adalah diantara 0 ~ dan lebih dari 10 m/jam^2

14.4 Algoritma

Disini mobil pertama yang diberi nama Car1 letaknya di depan mobil kedua yang diberi nama Car2.

Kita berada di Car2.

Apa yang harus kita lakukan adalah mengelola mobil kita (Car 2) dengan memperhatikan speed (kecepatan) dari mobil kita (Car2) dan distance (jarak) diantara mobil kita dan Car1.

Kita dapat memerintahkan mobil kita untuk mengerjakan sesuatu (artinya diberi command/perintah) apakah harus mempertahankan kecepatan kita sekarang, memelankan mobil kita dengan mengurangi kecepatan atau mempercepatnya dengan cara menekan pedal gas.

Kita definisikan fuzzy set dari:

- Distance → Narrow dan Wide
- Speed → Slow dan Fast
- Command → KeepSpeed, SlowDown dan SpeedUp

Akselerasi dari Car1 kita acak nilai awalnya, lalu set kecepatan (speed) Car1 dan update posisi dari Car1. Lakukan hal yang sama untuk Car2.

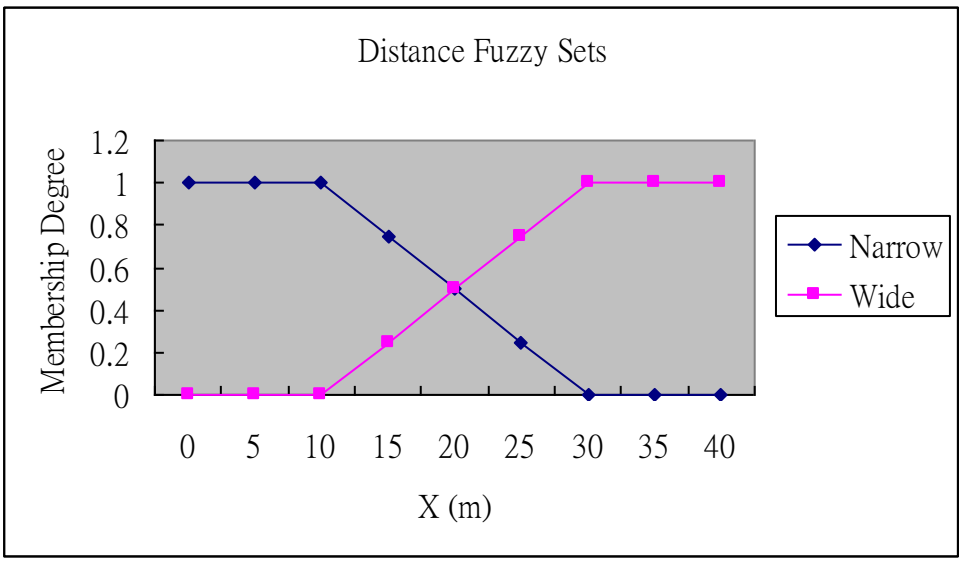
Temukan distance (jarak) antara Car1 dan Car2.

Dari speed (kecepatan) Car2 dan distance (jarak) antara 2 mobil tersebut, applikasikan 4 rule pada gambar 14.2 sebelumnya, untuk mendapatkan command (perintah) yang harus dilakukan oleh Car2 (apakah harus KeepSpeed (mempertahankan kecepatan sekarang), SlowDown (memperlambat laju mobil) atau SpeedUp (mempercepat kecepatan mobil)).

Kemudian untuk visualisasinya update grafis dan nilai-nilai yang diperoleh selama proses, demikian juga dengan posisi dari Car2 (dan tentunya Car1) pada suatu saat.

14.5 Graf

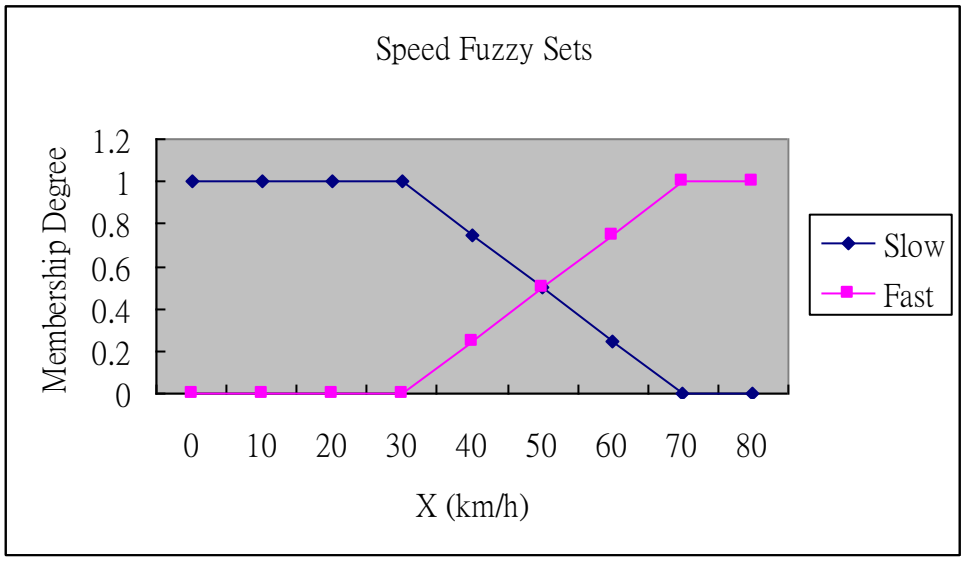
Gambar 14.3-14.5 menggambarkan graf yang merupakan implementasi dari fuzzy set untuk distance (jarak), speed (kecepatan) dan command (perintah).



Gambar 14.3 Graf untuk fuzzy set: distance (jarak)

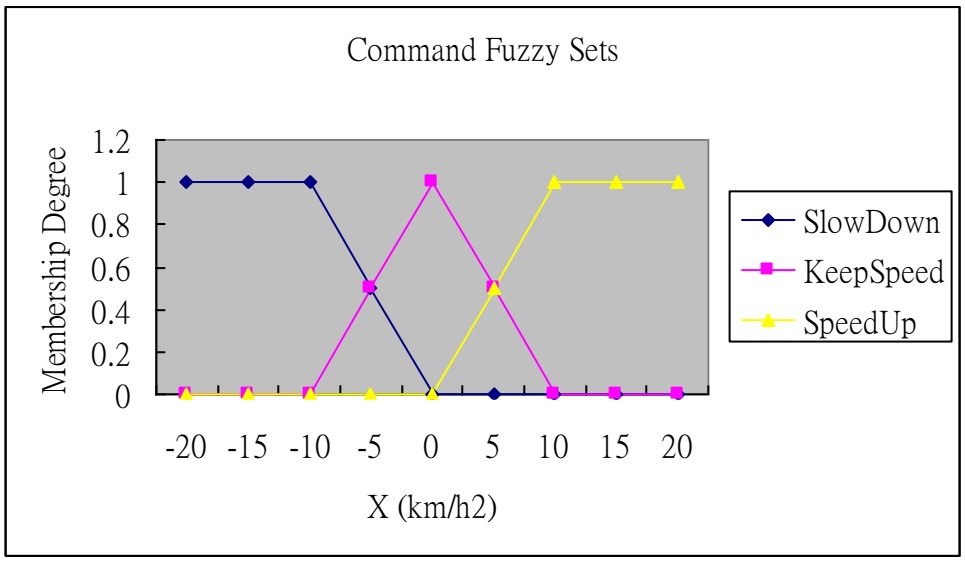
Untuk lebih mudah memahami, kita jelaskan sebagai berikut. Dimulai dengan dimisalkan bahwa jarak mobil kita (Car2) dengan Car1 adalah 15 meter. Maka kita lihat pada gambar 14.3 untuk mengetahui status jarak kita. Didapati bahwa jarak 15 meter itu termasuk narrow (dekat) dengan derajat keanggotaan (membership degree) 0.75 dan

juga wide (jauh) dengan derajat keanggotaan 0.25. Karena setelah dibandingkan 0.75 lebih besar daripada 0.25, maka status jarak yang lebih tepat dari mobil kita adalah narrow (dekat).



Gambar 14.4 Graf untuk fuzzy set: speed (kecepatan)

Dimisalkan juga kecepatan mobil kita (Car2) saat ini adalah 60 km/jam. Kita lihat pada gambar 14.4 untuk mengetahui status kecepatan Car2. Ternyata didapat bahwa kecepatan 60 km/jam itu termasuk fast (cepat) dengan derajat keanggotaan 0.75 dan juga slow (lambat) dengan derajat keanggotaan 0.25. Karena setelah dibandingkan 0.75 lebih besar daripada 0.25, maka status kecepatan yang lebih tepat dari Car2 adalah fast (cepat).



Gambar 14.5 Graf untuk fuzzy set: command (perintah)

Lalu kita aplikasikan pengetahuan pada gambar 14.2. Dari status jarak kita yang narrow (dekat) dan status kecepatan (speed) yang fast (cepat), maka command (perintah) yang paling sesuai dengan kondisi kita saat ini untuk diberikan pada agen kita adalah SlowDown, seperti telah tertulis pada pengetahuan no. 2 di bawah ini.

IF Distance is Narrow AND Speed is Fast THEN Command=SlowDown

Implementasinya sesungguhnya adalah dengan melakukan operasi OR dan AND pada nilai-nilai fuzzy set distance dan speed. Berdasarkan penjelasan bab 11 sebelumnya, operasi OR dilakukan dengan mengambil nilai maksimum dari nilai-nilai yang dioperasikan, sedangkan operasi AND dilakukan dengan mengambil nilai minimum dari nilai-nilai yang dioperasikan. Sehingga di hal di atas dapat dituliskan lagi sebagai berikut:

IF [Distance = Narrow (MD = membership degree = 0.75) OR Distance = Wide (MD = 0.25)] AND

[Speed = Fast (MD = 0.75) OR Speed = Slow (MD = 0.25)]

THEN

Command = ?

IF [Max(Distance=Narrow(MD=0.75), Distance=Wide(MD = 0.25))]

AND

[Max(Speed=Fast(MD=0.75), Speed=Slow(MD=0.25))]

THEN

Command = ?

IF Distance=Narrow(MD=0.75) AND Speed=Fast(MD=0.75)

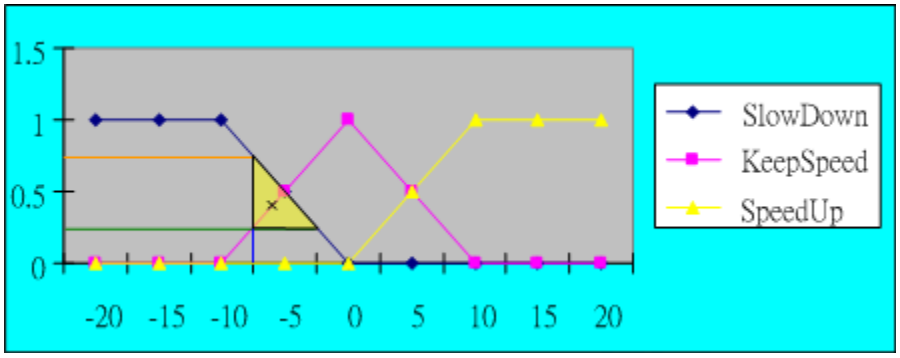
THEN Command = **SlowDown** → pengetahuan no. 2 pada gambar 14.2

Untuk mencari nilai percepatan pada daerah di SlowDown, kita cari dengan mencari nilai tengah dari area yang terbentuk (Center-of-Area, CoA atau Center-of-Gravity, CoG) yang didapat dari:

Min(Distance=Narrow(MD=0.75), Speed=Fast(MD=0.75)) = (MD=0.75)

Min(Distance=Wide(MD=0.25, Speed=Slow(MD=0.25)) = (MD=0.25)

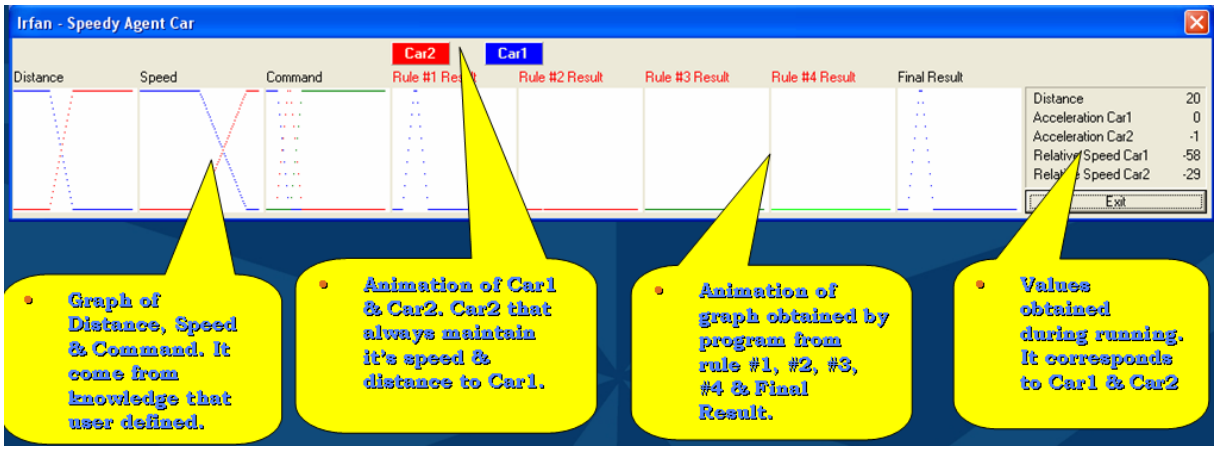
Lalu dicari daerah batasannya seperti diperlihatkan pada gambar 14.6 di bawah ini. Disini terlihat bahwa daerah batasan untuk SlowDown yang terbentuk adalah MD=0.75 dan MD=0.25 dimana daerah yang terbentuk adalah segitiga. CoA di dapat dengan mencari titik tengah dari suatu bentukan (tanda × pada gambar 14.6). Misalnya didapat CoA adalah -6 km/jam^2, maka mobil kita (Car2) harus dikurangai kecepatannya dengan percepatan 6 km/jam^2. Demikian seterusnya sistem akan terus bekerja berdasarkan jarak dan kecepatan mobil kita saat ini untuk menentukan aksi (command) selanjutnya.



Gambar 14.6 CoA (tanda ×) dari hasil yang didapat

14.6 Jalannya Program

Contoh tampilan sewaktu program dijalankan dapat dilihat pada gambar 14.7 di bawah ini.



Gambar 14.7 Tampilan sewaktu program dijalankan

14.7 Hasil dan Jalannya Program

Dari prototipe ini kita telah berhasil mengimplementasikan teori fuzzy set kedalam suatu agen, agen yang mengelola speed (kecepatan) dan distance (jarak) dari suatu mobil, dibandingkan dengan mobil yang lain.

Ini hanyalah prototipe, sehingga kita asumsikan bahwa posisi mobil lain didapatkan dari pelayanan GPS.

Sejujurnya ini hanyalah simulasi, sehingga untuk implementasi sesungguhnya, kita harus mempertimbangkan hal-hal berikut ini:

- GPS
- Implementasi perangkat keras
- Koordinasi canggih diantara bagian-bagian yang terlibat
- Regulasi pemerintah
- Dukungan masyarakat/komunitas
- Studi kelayakan dari banyak pihak

14.8 Kemungkinan Pengembangan

Disini kita hanya memperhatikan posisi dari mobil/kendaraan di depan atau di belakang mobil lain, kita dapat mengembangkannya untuk semua kemungkinan posisi di jalan.

Hal-hal lain yang perlu dipertimbangkan adalah:

- Objek-objek yang bergerak di jalan, tak hanya mobil/kendaraan
- Halangan/rintangan di jalan
- Pengaturan lampu lalu-lintas (traffic lights)
- Permasalahan kemacetan di jalanan
- Distribusi arus multi jalur di jalanan
- Jalur perjalanan efektif
- Untuk transportasi umum: di darat, laut atau udara
- Sistem militer

BAB 15 GENETIC SIMULATED ANNEALING

Bab ini membahas mengenai gabungan metode Genetic Algorithms (GA) dan Simulated Annealing (SA) yang disebut dengan Genetic Simulated Annealing (GSA). GSA ternyata menampilkan kinerja yang lebih baik daripada GA saja atau SA saja, karena ia mengambil kebaikan dari masing-masing metode aslinya (yaitu GA saja atau SA saja). Pustaka yang digunakan adalah dari Sirag dan Weisser [Sig87]; Adler [Adl93]; Brown et al. [Bro89]; Lin et al. [Lin93]; Koakutsu et al. [Koa90, Koa92]; Koakutsu et al. [Koa96].

15.1 SIMULATED ANNEALING

Simulated Annealing (SA) adalah metode perbaikan iteratif untuk menyelesaikan permasalahan optimasi kombinatorial.

SA membangkitkan satu urutan solusi dan mencarinya untuk satu solusi optimum pada jalur pencarian ini.

SA memulai dengan solusi awal x_0.

Pada setiap langkah, SA membangkitkan kandidat solusi x' dengan mengubahnya ke pecahan kecil dari solusi x saat itu.

SA menerima kandidat solusi sebagai solusi baru dengan probabilitas min $(1, e^{-\Delta f/T})$, dimana $\Delta f = f(x') - f(x)$ adalah pengurangan cost (biaya) dari solusi saat ini x ke kandidat solusi x', dan T adalah parameter kontrol yang disebut dengan temperatur.

Titik kunci dari SA adalah bahwa SA menerima pergerakan up-hill (ke atas bukit) dengan probabilitas $e^{-\Delta f/T}$.

Hal ini menjadikan SA dapat menghindar dari local minima.

Namun SA tak dapat mencakup daerah yang luas dari daerah solusi dalam waktu komputasi yang terbatas disebabkan SA mendasarkan dirinya pada pergerakan kecil.

Pseudocode dari SA, dikemukakan oleh Koakutsu et al. [Koa96] dapat dilihat pada gambar 15.1 di bawah.

```
SA_algorithm(Na, T0, α)
{
    x ← x0; /* initial solution */
    T ← T0; /* initial temperature */
    while (system is not frozen) {
        for (loop = 1; loop ≤ Na; loop++) {
            x' ← Mutate(x);
            Δf ← f(x') - f(x);
            r ← random number between 0 and 1
            if (Δf < 0 or r < exp(-Δf/T))
                x ← x';
        }
        T ← T * α /*  lower temperature */
    }
    return x
}
```

Gambar 15.1 Pseudocode SA

15.2 Genetic Simulated Annealing

Untuk Genetic Algorithms (GA) sudah dibahas pada bab 7, sehingga tidak perlu dibahas lagi. Kita hanya tinggal menggunakan konsep GA tersebut dikombinasikan dengan SA, sehingga diperoleh suatu metode yang disebut dengan Genetic Simulated Annealing (GSA).

Dalam rangka meningkatkan kinerja dari GA dan SA, beberapa algoritma persilangan telah diusulkan.

SA-Based

Sirag dan Weisser [Sig87] mengajukan operator genetika thermodinamik, yang menggabungkan jadwal annealing untuk mengontrol probabilitas pengaplikasian mutasi. Sedangkan Adler [Adl93] menggunakan fungsi penerimaan SA-based untuk mengontrol probabilitas penerimaan solusi baru yang dihasilkan oleh mutasi.

GA-Based

Penilitian yang lebih terkini pada persilangan berbasis GA adalah metode Simulated Annealing Genetic Algorithm (SAGA) yang diajukan oleh Brown et al. [Bro89] dan metode Annealing Genetic (AG) yang diusulkan oleh Lin et al. [Lin93].

Baik metode SA-Based maupun GA-Based masing-masingnya membagi "generasi" menjadi 2 fase: fase GA dan fase SA.

GA membangkitkan sekumpulan solusi baru mengggunakan operator crossover dan

kemudian SA memperbaiki lebih lanjut setiap solusi dalam populasi.

Jika SAGA menggunakan jadwal annealing yang sama untuk setiap fase SA, sedangkan AG mencoba untuk mengoptimasi jadwal yang berbeda untuk fase SA yang berbeda.

Metode persilangan berbasis GA diatas mencoba untuk memasukkan fitur-fitur hill climbing stokastik lokal kedalam GA.

Karena mereka memasukkan SA lengkap kedalam setiap generasi dan jumlah generasi biasanya sangat besar, metode persilangan berbasis GA sangat memakan waktu dalam komputasinya.

Pada sisi lain, pendekatan persilangan berbasis SA mencoba untuk mengadopsi operasi crossover global dari GA kedalam SA.

Parallel Genetic Simulated Annealing (PGSA) yang diajukan oleh Koakutsu et al. [Koa90, Koa92], adalah versi paralel dari SA yang memadukan fitur-fitur GA.

Selama pencarian paralel berbasis SA, crossover digunakan untuk membangkitkan solusi baru dalam rangka memperbesar daerah pencarian dari SA.

15.3 Cara Kerja GSA

GSA diajukan oleh Koakutsu et al. [Koa96].

Jika PGSA membangkitkan bibit-bibit pencarian loka SA secara paralel, yaitu urutan aplikasi setiap pencarian lokal SA adalah independen, GSA membangkitkan bibit-bibit SA secara sekuensial/berurutan, yaitu bibit-bibit pencarian lokal SA tergantung pada solusi terbaik saat itu (best-so-far) dari semua pencarian lokal SA sebelumnya.

Pendekatan secara sekuensial ini kelihatannya dapat membangkitakan solusi anak yang lebih baik.

Sebagai tambahan, bila dibandingkan dengan PGSA, GSA menggunakan lebih sedikit operasi crossover karena ia hanya menggunakan operasi crossover jika pencarian lokal SA mencapai permukaan yang mendatar dan inilah waktunya untuk melompat, dalam ruang solusi.

GSA memulai prosesnya dengan sebuah populasi $X = \{x_1, \ldots, X_{Np}\}$ dan secara berulang mengaplikasikan 3 operasi: pencarian lokal berbasis SA, operasi crossover berbasis GA, dan mengupdate populasi.

Pencarian lokal berbasis SA menghasilkan kandidat solusi x’ dengan mengubah pecahan kecil dari state x.

Kandidat solusi diterima sebagai solusi baru dengan probabilitas min $\{1, e^{-\Delta f/T}\}$.

GSA menjaga nilai dari solusi lokal best-so-far (terbaik sampai saat ini) x^*_L selama pencarian lokal berbasis SA.

Jika pencarian mencapai permukaan mendatar/rata atau sistem didapati membeku (tidak bergerak ke titik baru), GSA menghasilkan lompatan besar dalam ruang solusi dengan menggunakan crossover berbasis GA.

GSA mengambil pasangan parent solutions (solusi induk) x_j dan x_k secara acak dari populasi X sehingga $f(x_j) \neq f(x_k)$, menjalankan operator crossover, dan lalu menggantikan solusi terjelek x_i dengan solusi baru yang dihasilkan oleh operator crossover.

Pada setiap akhir dari setiap pencarian lokal berbasis SA, GSA mengupdate populasi dengan menggantikan solusi saat itu x_i dengan solusi lokal best-so-far x^*_L.

GSA berhenti jika waktu CPU mencapai batas yang ditentukan, dan melaporkan solusi global best-so-far x^*_G.

Pseudocode GSA yang dikemukakan oleh Koakutsu et al. [Koa96] pada gambar 15.2 berikut ini.

```
GSA_algorithm(Np, Na, T0, α)
{
    X ← {x1, ..., XNp}; /* initialize population */
    x*L ← the best solution among X; /* initialize local best-so-far */
    x*G ← x*L /* initialize global best-so-far */
    while (not reach CPU time limit) {
      T ← T0; /* initialize temperature */
      /* jump */
      select the worst solution xi from X;
      select two solutions xj, xk from X such that f(xj) ≠ f(xk);
      xi ← Crossover(xj, xk);
      /* SA-based local search */
      while (not frozen or not meet stopping criterion) {
         for (loop = 1; loop ≤ Na; loop++) {
            x' ← Mutate(xi);
       Δf ← f(x') - f(xi);
       r ← random number between 0 and 1
       if (Δf < 0 or r < exp(-Δf/T))
          xi ← x';
       if (f(xi) < f(x*L))
          x*L ← xi; /* update local best-so-far */
     }
     T ← T * α /* lower temperature */
      }
      if (f(x*L) < f(x*G))
         x*G ← x*L; /* update global best-so-far */
      /* update population */
      xi ← x*L;
      f(x*L) ← +∞; /* reset current local best-so-far */
   }
   return x*G;
}
```

Gambar 15.2 Pseudocode GSA

15.4 KESIMPULAN

- SA membangkitkan satu urutan tungal solusi dan mencari suatu solusi optimum dalam jalur pencarian ini
- Pada setiap langkah, SA membangkitkan kandidat solusi x' dengan mengubah satu pecahan kecil dari solusi x saat itu.
- Titik kunci dari SA adalah bahwa SA menerima pergerakan ke atas bukit (up-hill moves) dengan probabilitas $e^{-\Delta f/T}$
- Hal ini membolehkan SA untuk menghindar dari local minima
- Tetapi SA tak dapat mencakup sebuah daerah yang lebar dari ruang solusi dalam waktu komputasi yang terbatas disebabkan SA adalah berdasarkan pada pergerakan-pergerakan kecil

- GSA membangkitkan bibit-bibit SA secara berurutan/sekuensial, yaitu bibit-bibit dari pencarian lokal SA tergantung pada solusi terbaik saat ini (the best-so-far solutions) dari semua pencarian lokal SA.
- Pendekatan sekuensial ini nampaknya dapat menghasilkan solusi-solusi anak yang lebih baik
- GSA menggunakan operasi crossover yang lebih sedikit, disebabkan ia hanya menggunakan operasi crossover disaat pencarian lokal SA mencapai permukaan yang rata/mendatar (flat surface) dan itulah waktunya untuk melompat dalam ruang solusi.
- GSA diajukan untuk mengatasi: ketidakmampuan dalam mencakup daerah yang lebar dari ruang solusi SA dan ketidakmampuan pencarian daerah lokal dari ruang solusi GA.

Bab 16 Aplikasi GSA pada RDB (1)

GSA yang telah dibahas pada bab 15 akan diimplementasikan pada bab ini, yaitu pada kasus estimasi nilai null dalam pembangkitan fuzzy rule terboboti yang diterapkan pada database relasional. Pustaka yang digunakan adalah dari Sirag dan Weisser [Sig87]; Adler [Adl93]; Brown et al. [Bro89]; Lin et al. [Lin93]; Koakutsu et al. [Koa90, Koa92]; Koakutsu et al. [Koa96]; S.M. Chen and C.M. Huang [Chen03]; dan yang terpenting adalah dari hasil riset penulis sendiri [Sub05a].

Nama Aplikasi: **GSA untuk Mengestimasi Nilai Null dalam Pembangkitan Fuzzy Rule Terboboti dari Database Relasional**

Pada bab ini akan dibahas aplikasi GSA (Genetic Simulated Annealing) pada database yang memiliki hubungan relasional (RDB - Relational Database).

Aplikasi ini akan memperkirakan/menghasilkan nilai dari nilai null dari database relasional yang memiliki rule-rule fuzzy terboboti, seperti halnya bab 12 sebelumnya. Kita ingat kembali, bahwa nilai null disini bisa jadi timbul dari kerusakan pada suatu record database akibat suatu kesalahan/bencana atau memang nilai tersebut belum terdefinisikan sebelumnya. Manfaat yang bisa didapat dari aplikasi ini, seperti halnya bab 12 sebelumnya adalah dapat mengembalikan kembali/memperkirakan nilai null tadi sehingga suatu database yang tadinya timpang karena ada nilai null-nya dapat didayagunakan lebih baik lagi seperti halnya database yang normal/tidak memiliki nilai null.

16.1 Konsep Dasar Fuzzy Set

Telah jelas diterangkan pada sub bab 12.1.

16.2 Fuzzy Set pada Database Relasional

Telah jelas diterangkan pada sub bab 12.2.

16.3 Derajat Kemiripan

Telah jelas diterangkan pada sub bab 12.3.

16.4 Rumus-rumus yang Digunakan

Telah jelas diterangkan pada sub bab 12.4.

16.4 Estimasi Nilai Null dalam RDB dengan GSA

Pseudocode dari prosedur *Evaluasi dan Seleksi Terbaik* dari program GSA kita dapat dilihat pada gambar 16.1 di bawah ini.

```
Procedure EvaluationAndBestSelection
{find the best solution among population. Also it initializes
LocalBestChromosomeSoFar and GlobalBestChromosomeSoFar:
X ← {x1, ..., XNp}; {initialize population}
x*L ← the best solution among X; {initialize local best-so-far}
x*G ← x*L {initialize global best-so-far}
FitnessDegreeEval ← FitnessDegree from global best-so-far
}
for i:= 1 to number-of-generations do begin
    T ← T0;
    EvaluationAndWorstSelection; {select the worst solution xi from X}
    CrossOver; {select two solutions xj, xk from X such that f(xj) ≠
f(xk):
      xi ← Crossover(xj, xk);
    }
    Mutation; {update local best-so-far if value is better
      repeat
         for i:= 0 to number-of-mutation do begin
           f(xi) ← Get Fitness Degree from chromosome before mutation
           x' ← Mutate(xi)
           f(x') ← Get Fitness Degree from chromosome after mutation
           Δf ← f(xi) - f(x')
           r ← random number between 0 and 1
           ft ← f(x')
           if (Δf >= 0) or (r >= exp(-Δf/T)) then begin
              xi ← x';
              ft ← f(xi);
           end;
           if (ft >= FitnessDegreeEval) then begin
              x*L ← xi; {update local best-so-far}
              FitnessDegreeEval ← ft
              FDLocalBestSoFar ← ft {Get local best Fitness Degree}
           end
         end
         T ← T * α; {lower temperature}
      until T <= FrozenValue;
    }
    CountCloseness(x*L); {get FD from LocalBestChromosomeSoFar}
    AvgError:= AvgError / NumData;
    FDLocalBestSoFar:= 1 - AvgError;
    CountCloseness(x*G); {get FD from GlobalBestChromosomeSoFar}
    AvgError:= AvgError / NumData;
    FDGlobalBestSoFar:= 1 - AvgError;
```

```
        if FDLocalBestSoFar >= FDGlobalBestSoFar then begin
          x*G ← x*L; {update global best-so-far}
          FitnessDegreeEval:= FDGlobalBestSoFar;
        end;
        xi ← x*L; {update population}
    end;
```

Gambar 16.1 Pseudocode untuk prosedur EvaluationAndBestSelection

Sedangkan pseudocode untuk prosedur CountCloseness dapat dilihat pada gambar 16.2.

```
Procedure CountCloseness
AvgError:= 0.0;
for i:= 0 to NumData - 1 do begin  {base on all data available}
  BestClosenessEval:= MaxInt;
  IdxClosestCloseness:= i;
  for j:= 0 to NumData - 1 do
    if i <> j then begin
      if Rank(Ti.X) ≥ Rank(Tj.X) then begin
        ClosenessE(Ti,Tj)= Similarity(Ti.X,Tj.X) × Weight(Tj.Degree) +
            × Weight(Tj.Experience);
      end
      else begin {If Rank(Ti.X) < Rank(Tj.X)}
        ClosenessE:= 1/Similarity(Ti.X,Tj.X) × Weight(Tj.Degree) +
            × Weight(Tj.Experience);
      end;
      {find a tuples which is closest to 1.0 as a}
      {closest tuple to tuple Ti}
      ClosestCloseness:= Abs(1 - ClosenessE);
      if ClosestCloseness <= BestClosenessEval then begin
        BestClosenessEval:= ClosestCloseness;
        IdxClosestCloseness:= j;
      end;
    end;
    {Then we find Estimated Salary and Error for every record}
    {if this record was null value, so we must find}
    {another record that closest to 1}
  if IsNullValue(i) and IsNullValue(IdxClosestCloseness) then begin
    PreferIdx:= GetPreferIdx;
    ETi.Salary:= Ti. Salary × GetClosenessValue(PreferIdx);
    if Tprefer-index.Salary <> 0 then
      Errori:=
  end
  else begin
    ETi.Salary:= Ti. Salary × GetClosenessValue(IdxClosestCloseness);
    if Ti.Salary <> 0 then
      Errori:=
  end;
  AvgError:= AvgError + Abs(Errori);
end;
```

$$\frac{T_i.Experience}{T_j.Experience}$$

$$\frac{T_i.Experience}{T_j.Experience}$$

$$\frac{ET_i.Salary - T_{prefer-index}.Salary}{T_{prefer-index}.Salary}$$

$$\frac{ET_i.Salary - T_i.Salary}{T_i.Salary}$$

Gambar 16.2 Pseudocode untuk prosedur CountCloseness

Pseudocode untuk function GetClosenessValue dapat dilihat pada gambar 16.3.

```
Function GetClosenessValue(Idx)
  Result ← find value in ClosenessE which have the same index with Idx
```

Gambar 16.3 Pseudocode untuk function GetClosenessValue(Idx)

Pseudocode untuk function GetPreferIdx dapat dilihat pada gambar 16.4 di bawah ini.

```
Function GetPreferIdx
  Result ← find value in ClosenessE that closest to 1, and it's not null
value
```

Gambar 16.4 Pseudocode untuk function GetPreferIdx

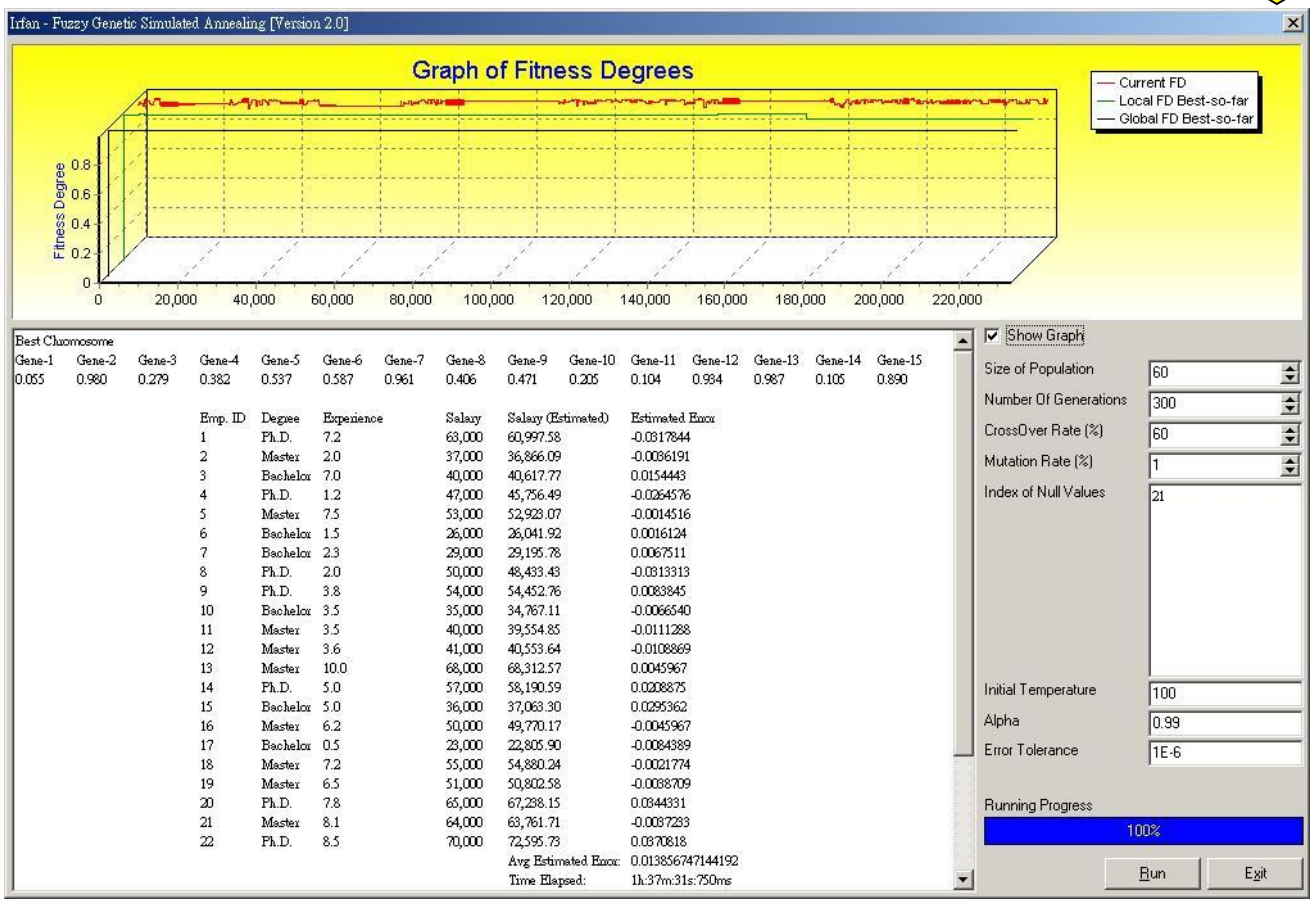
16.5 Percobaan

Kita jalankan program ini dengan pelbagai parameter yang berbeda, yang masing-masingnya dijalankan 10 kali setiap saat. Didapatkan hasil-hasilnya sebagai berikut:

Percobaan Tipe 1

- Mutation Rate = 0.01 = 1%
- Initial Temperature = 100
- Alpha = 0.7
- Frozen Value = 0.00001
- Index of Null Values = 21 (ini artinya baris/tuple ke-22 dalam database relasional)

Sedangkan tampilan dari program dan hasilnya dapat dilihat pada gambar 16.5 berikut ini.



Gambar 16.5 Tampilan program dan hasil dari percobaan tipe 1

Hasil lebih lanjut dapat dilihat pada tabel 16.1-16.4 di bawah ini.

Tabel 16.1 Hasil 1 dari percobaan tipe 1

Size Of Population 30
Number of Generations 100

Running #	Avg. Estimated Error	Hour	Minute	Second	Mili-second	Total Time
1	0.011162	0	2	16	781	0h:2m:16s:781ms
2	0.009827	0	2	10	844	0h:2m:10s:844ms
3	0.008853	0	2	36	797	0h:2m:36s:797ms
4	0.008853	0	3	12	250	0h:3m:12s:250ms
5	0.008853	0	3	5	406	0h:3m:5s:406ms
6	0.008853	0	2	45	766	0h:2m:45s:766ms
7	0.007029	0	2	34	469	0h:2m:34s:469ms
8	0.006138	0	2	44	46	0h:2m:44s:46ms
9	0.006138	0	2	41	235	0h:2m:41s:235ms
10	0.006138	0	2	32	515	0h:2m:32s:515ms
Min	**0.006138**	**0**	**2**	**5**	**46**	
Average	**0.008184**	**0**	**2.2**	**27.5**	**510.9**	
Max	**0.011162**	**0**	**3**	**45**	**844**	

Tabel 16.2 Hasil 2 dari percobaan tipe 1

Size Of Population 40
Number of Generations 150

Running #	Avg. Estimated Error	Hour	Minute	Second	Mili-second	Total Time
1	0.018120004	0	6	9	578	0h:6m:9s:578ms
2	0.014019037	0	8	23	937	0h:8m:23s:937ms
3	0.008319306	0	6	56	344	0h:6m:56s:344ms
4	0.004907964	0	7	13	969	0h:7m:13s:969ms
5	0.004907964	0	9	16	703	0h:9m:16s:703ms
6	0.003288305	0	7	40	547	0h:7m:40s:547ms
7	0.003172975	0	6	53	187	0h:6m:53s:187ms
8	0.003172975	0	7	9	438	0h:7m:9s:438ms
9	0.003153504	0	7	34	594	0h:7m:34s:594ms
10	0.003153504	0	7	39	406	0h:7m:39s:406ms
Min	**0.003153504**	**0**	**6**	**9**	**187**	
Average	**0.006621554**	**0**	**7**	**29.2**	**570.3**	
Max	**0.018120004**	**0**	**9**	**56**	**969**	

Tabel 16.3 Hasil 3 dari percobaan tipe 1

Size Of Population 50
Number of Generations 200

Running #	Avg. Estimated Error	Hour	Minute	Second	Mili-second	Total Time
1	0.009636727	0	10	36	531	0h:10m:36s:531ms
2	0.007631349	0	10	31	250	0h:10m:31s:250ms
3	0.007631349	0	9	54	484	0h:9m:54s:484ms
4	0.007631349	0	9	47	719	0h:9m:47s:719ms
5	0.007631349	0	9	47	735	0h:9m:47s:735ms
6	0.007631349	0	9	54	62	0h:9m:54s:62ms
7	0.007631349	0	9	50	469	0h:9m:50s:469ms
8	0.007631349	0	9	48	531	0h:9m:48s:531ms
9	0.006410408	0	9	53	828	0h:9m:53s:828ms
10	0.006334942	0	9	45	610	0h:9m:45s:610ms
Min	**0.006334942**	**0**	**9**	**31**	**62**	
Average	**0.007580152**	**0**	**9.2**	**46.5**	**521.9**	
Max	**0.009636727**	**0**	**10**	**54**	**828**	

Tabel 16.4 Hasil 4 dari percobaan tipe 1

Size Of Population 60
Number of Generations 300

Running #	Avg. Estimated Error	Hour	Minute	Second	Mili-second	Total Time
1	0.01365808	0	20	28	453	0h:20m:28s:453ms
2	0.01365808	0	20	32	328	0h:20m:32s:328ms
3	0.004348226	0	20	29	500	0h:20m:29s:500ms
4	0.004348226	0	20	30	125	0h:20m:30s:125ms
5	0.004348226	0	20	27	78	0h:20m:27s:78ms
6	0.004348226	0	20	23	282	0h:20m:23s:282ms
7	0.004348226	0	20	27	984	0h:20m:27s:984ms
8	0.004348226	0	20	30	328	0h:20m:30s:328ms
9	0.004348226	0	20	31	578	0h:20m:31s:578ms
10	0.004348226	0	20	30	16	0h:20m:30s:16ms
Min	**0.004348226**	**0**	**20**	**23**	**16**	
Average	**0.006210197**	**0**	**20**	**28.7**	**367.2**	
Max	**0.01365808**	**0**	**20**	**32**	**984**	

Percobaan Tipe 2

- Mutation Rate = 0.1 = 10%
- Initial Temperature = 100
- Alpha = 0.7
- Frozen Value = 0.00001
- Index of Null Values = 21 (ini artinya baris/tuple ke-22 dalam database relasional)

Sedangkan tampilan dari program dan hasilnya dapat dilihat pada gambar 16.5-16.8 berikut ini.

Tabel 16.5 Hasil 1 dari percobaan tipe 2

Size Of Population 30
Number of Generations 100

Running #	Avg. Estimated Error	Hour	Minute	Second	Mili-second	Total Time
1	0.00905076	0	21	56	125	0h:21m:56s:125ms
2	0.00905076	0	21	35	563	0h:21m:35s:563ms
3	0.007650809	0	20	59	672	0h:20m:59s:672ms
4	0.007650809	0	20	53	484	0h:20m:53s:484ms
5	0.004252343	0	20	58	453	0h:20m:58s:453ms
6	0.003712235	0	20	57	891	0h:20m:57s:891ms
7	0.003712235	0	21	1	203	0h:21m:1s:203ms
8	0.003194167	0	20	57	734	0h:20m:57s:734ms
9	0.003194167	0	21	2	641	0h:21m:2s:641ms
10	0.003194167	0	21	10	687	0h:21m:10s:687ms
Min	**0.003194167**	**0**	**20**	**1**	**125**	
Average	**0.005466245**	**0**	**20.5**	**38.8**	**545.3**	
Max	**0.00905076**	**0**	**21**	**59**	**891**	

Tabel 16.6 Hasil 2 dari percobaan tipe 2

Size Of Population 40
Number of Generations 100

Running #	Avg. Estimated Error	Hour	Minute	Second	Mili-second	Total Time
1	0.007768472	0	46	50	906	0h:46m:50s:906ms
2	0.007720671	0	47	0	282	0h:47m:0s:282ms
3	0.007720671	0	46	55	15	0h:46m:55s:15ms
4	0.005481832	0	47	2	891	0h:47m:2s:891ms
5	0.005177396	0	47	8	125	0h:47m:8s:125ms
6	0.005177396	0	47	1	890	0h:47m:1s:890ms
7	0.004702165	0	47	57	907	0h:47m:57s:907ms
8	0.003876522	0	50	1	31	0h:50m:1s:31ms
9	0.003410507	0	50	9	594	0h:50m:9s:594ms
10	0.003410507	0	50	36	234	0h:50m:36s:234ms
Min	**0.003410507**	**0**	**46**	**0**	**15**	
Average	**0.005444614**	**0**	**47.7**	**21.9**	**487.5**	
Max	**0.007768472**	**0**	**50**	**57**	**907**	

Tabel 16.7 Hasil 3 dari percobaan tipe 2

Size Of Population 50
Number of Generations 100

Running #	Avg. Estimated Error	Hour	Minute	Second	Mili-second	Total Time
1	0.009740355	1	23	28	125	1h:23m:28s:125ms
2	0.009450571	1	23	14	938	1h:23m:14s:938ms
3	0.009450571	1	23	25	219	1h:23m:25s:219ms
4	0.008133827	1	23	18	734	1h:23m:18s:734ms
5	0.008133827	1	23	21	969	1h:23m:21s:969ms
6	0.005986674	1	23	24	234	1h:23m:24s:234ms
7	0.003821698	1	23	24	188	1h:23m:24s:188ms
8	0.003821698	1	23	30	375	1h:23m:30s:375ms
9	0.003821698	1	23	27	437	1h:23m:27s:437ms
10	0.003821698	1	23	30	938	1h:23m:30s:938ms
Min	**0.003821698**	**1**	**23**	**14**	**125**	
Average	**0.006618262**	**1**	**23**	**24.1**	**515.7**	
Max	**0.009740355**	**1**	**23**	**30**	**969**	

Tabel 16.8 Hasil 4 dari percobaan tipe 2

Size Of Population 60
Number of Generations 100

Running #	Avg. Estimated Error	Hour	Minute	Second	Mili-second	Total Time
1	0.003090779	3	17	21	94	3h:17m:21s:94ms
2	0.00300359	3	14	42	203	3h:14m:42s:203ms
3	0.00300359	3	14	42	250	3h:14m:42s:250ms
4	0.00300359	3	16	31	188	3h:16m:31s:188ms
5	0.00300359	3	18	55	156	3h:18m:55s:156ms
6	0.00300359	3	10	8	625	3h:10m:8s:625ms
7	0.002890637	3	8	29	797	3h:8m:29s:797ms
8	0.002890637	3	21	7	125	3h:21m:7s:125ms
9	0.002890637	3	18	13	125	3h:18m:13s:125ms
10	0.002890637	3	15	50	968	3h:15m:50s:968ms
Min	**0.002890637**	**3**	**8**	**7**	**94**	
Average	**0.002967128**	**3**	**15.1**	**29.8**	**353.1**	
Max	**0.003090779**	**3**	**21**	**55**	**968**	

Kesimpulan dari hasil-hasil percobaan yang telah dilakukan dapat disarikan pada tabel 16.9 berikut ini.

Tabel 16.9 Kesimpulan dari hasil-hasil percobaan

Avg. Estimation Error	Size Of Population	Number of Generations	Mutation Rate (%)	Initial Temperature	Alpha	Frozen Value
0.002967	60	300	10	100	0.7	0.00001
0.005445	40	150	10	100	0.7	0.00001
0.005466	30	100	10	100	0.7	0.00001
0.00621	60	300	1	100	0.7	0.00001
0.006618	50	200	10	100	0.7	0.00001
0.006622	40	150	1	100	0.7	0.00001
0.00758	50	200	1	100	0.7	0.00001
0.008184	30	100	1	100	0.7	0.00001

Min. Estimation Error	Size Of Population	Number of Generations	Mutation Rate (%)	Initial Temperature	Alpha	Frozen Value
0.002967128	60	300	10	100	0.7	0.00001
0.005444614	40	150	10	100	0.7	0.00001
0.005466245	30	100	10	100	0.7	0.00001
0.006210197	60	300	1	100	0.7	0.00001
0.006618262	50	200	10	100	0.7	0.00001
0.006621554	40	150	1	100	0.7	0.00001
0.007580152	50	200	1	100	0.7	0.00001
0.008184308	30	100	1	100	0.7	0.00001

Max. Estimation Error	Size Of Population	Number of Generations	Mutation Rate (%)	Initial Temperature	Alpha	Frozen Value
0.003090779	60	300	10	100	0.7	0.00001
0.007768472	40	150	10	100	0.7	0.00001
0.00905076	30	100	10	100	0.7	0.00001
0.009636727	50	200	1	100	0.7	0.00001
0.009740355	50	200	10	100	0.7	0.00001
0.011162399	30	100	1	100	0.7	0.00001
0.01365808	60	300	1	100	0.7	0.00001
0.018120004	40	150	1	100	0.7	0.00001

Sebagai perbandingan kita sajikan lagi hasil-hasil dari penelitian Chen et al. [Chen03] pada gambar 16.6 berikut ini.

Gene Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0.010	0.071	0.343	0.465	0.505	0.303	0.495	0.081	0.778	0.717	0.303	0.869	0.869	0.828	0.434
	B-H	M-H	P-H	B-SH	M-SH	P-SH	B-M	M-M	P-M	B-SL	M-SL	P-SL	B-L	M-L	P-L

Gambar 16.6 Kromosom terbaik yang didapat dari [Chen03]

Tabel 16.10 di bawah ini adalah hasil dari ukuran populasi: 60; jumlah generasi: 300; crossover rate: 1.0; dan mutation rate: 0.2.

Tabel 16.10 Perkiraan gaji dan perkiraan kesalahan untuk setiap tuple

EMP-ID	Degree	Experience	Salary	Salary (Estimated)	Estimated Error
S1	Ph.D.	7.2	63,000	61,515.00	-0.024
S2	Master	2.0	37,000	36,967.44	-0.001
S3	Bachelor	7.0	40,000	40,634.14	0.016
S4	Ph.D.	1.2	47,000	46,873.66	-0.003
S5	Master	7.5	53,000	56,134.37	0.059
S6	Bachelor	1.5	26,000	26,146.40	0.006
S7	Bachelor	2.3	29,000	27,822.08	-0.041
S8	Ph.D.	2.0	50,000	50,067.20	0.001
S9	Ph.D.	3.8	54,000	53,958.94	-0.001
S10	Bachelor	3.5	35,000	35,152.00	0.004
S11	Master	3.5	40,000	40,206.19	0.005
S12	Master	3.6	41,000	40,796.57	-0.005
S13	Master	10.0	68,000	68,495.74	0.007
S14	Ph.D.	5.0	57,000	56,240.72	-0.013
S15	Bachelor	5.0	36,000	34,277.54	-0.048
S16	Master	6.2	50,000	49,834.85	-0.003
S17	Bachelor	0.5	23,000	23,722.40	0.031
S18	Master	7.2	55,000	51,950.6	-0.055
S19	Master	6.5	51,000	51,197.58	0.004
S20	Ph.D.	7.8	65,000	64,813.75	-0.003
S21	Master	8.1	64,000	60,853.28	-0.049
S22	Ph.D.	8.5	70,000	69,065.83	-0.013
Average Estimated Error					0.018

Untuk jalannya program yang lain, kita dapatkan kesalahan perkiraan rata-rata untuk parameter GA yang berbeda [Chen03] seperti terlihat pada tabel 16.11.

Tabel 16.11 Rata-rata perkiraan kesalahan pada parameter berbeda untuk algoritma genetika

Size of Population	Number of Generations	Crossover Rate	Mutation Rate	Average Estimated Error
30	100	1.0	0.1	0.036
40	150	1.0	0.1	0.032
50	200	1.0	0.2	0.027
60	300	1.0	0.2	0.018

Contoh dari kromosom terbaik yang didapat dari penggunaan GSA adalah seperti diperlihatkan pada gambar 16.7 berikut ini.

```
Size of Population:      60
Number of Generations:   300
Mutation Rate (%):       10
Initial Temperature:     100
Alpha:              0.7
Frozen Value:       1E-5
Index of Null Values:    21
Best Chromosome
Gene-1    Gene-2    Gene-3    Gene-4    Gene-5    Gene-6    Gene-7    Gene-8
0.719     0.995     0.989     0.485     0.095     0.896     0.277     0.416
Gene-9    Gene-10   Gene-11   Gene-12   Gene-13   Gene-14   Gene-15
0.085     0.997     0.183     0.583     0.350     0.652     0.241
```

Gambar 16.7 Contoh kromosom terbaik yang didapat dengan GSA

Sedangkan hasil penggunaan GSA untuk memperkirakan gaji dan perkiraan kesalahan untuk setiap tuple diperlihatkan pada tabel 16.12 berikut ini.

Tabel 16.12 Perkiraan gaji dan perkiraan kesalahan untuk setiap tuple menggunakan GSA

Emp. ID	Degree	Experience	Salary	Salary (Estimated)	Estimated Error
1	Ph.D.	7.2	63,000	62,889.86	-0.0017482
2	Master	2.0	37,000	36,847.97	-0.0041090
3	Bachelor	7.0	40,000	40,128.33	0.0032082
4	Ph.D.	1.2	47,000	46,538.60	-0.0098170
5	Master	7.5	53,000	52,978.58	-0.0004042
6	Bachelor	1.5	26,000	25,970.00	-0.0011540
7	Bachelor	2.3	29,000	28,967.01	-0.0011375
8	Ph.D.	2.0	50,000	50,341.15	0.0068230
9	Ph.D.	3.8	54,000	53,836.28	-0.0030319
10	Bachelor	3.5	35,000	35,060.59	0.0017310
11	Master	3.5	40,000	39,876.06	-0.0030986
12	Master	3.6	41,000	40,875.72	-0.0030312
13	Master	10.0	68,000	68,087.03	0.0012798
14	Ph.D.	5.0	57,000	56,731.71	-0.0047068
15	Bachelor	5.0	36,000	36,051.19	0.0014219
16	Master	6.2	50,000	49,936.01	-0.0012798
17	Bachelor	0.5	23,000	22,940.28	-0.0025966
18	Master	7.2	55,000	54,966.66	-0.0006062
19	Master	6.5	51,000	50,945.03	-0.0010778
20	Ph.D.	7.8	65,000	64,938.81	-0.0009414
21	Master	8.1	64,000	63,933.65	-0.0010367
22	Ph.D.	8.5	70,000	70,654.72	0.0093531

```
Avg Estimated Error: 0.002890636946022
Time Elapsed:   3h:15m:50s:968ms
```

Dari sini kita dapat membuktikan bahwa GSA dapat memperbaiki hasil yang diperoleh dari Chen et al. [Chen03] yang ditandai dengan semakin kecilnya kesalahan yang terjadi, namun tentu hal ini harus dibayar dengan waktu pemrosesan yang lebih lama.

Bab 17 Aplikasi GSA pada RDB (2)

GSA yang telah dibahas pada bab 15 akan diimplementasikan pada bab ini, yaitu pada kasus estimasi nilai null majemuk (multiple null values) dalam pembangkitan fuzzy rule terboboti yang diterapkan pada database relasional. Pustaka yang digunakan adalah dari Sirag dan Weisser [Sig87]; Adler [Adl93]; Brown et al. [Bro89]; Lin et al. [Lin93]; Koakutsu et al. [Koa90, Koa92]; Koakutsu et al. [Koa96]; S.M. Chen and C.M. Huang [Chen03]; dan yang terpenting adalah dari hasil riset penulis sendiri [Sub05b].

Nama Aplikasi: **GSA untuk Mengestimasi Nilai Null Majemuk dalam Pembangkitan Fuzzy Rule Terboboti dari Database Relasional**

Pada bab ini akan dibahas aplikasi GSA (Genetic Simulated Annealing) pada database yang memiliki hubungan relasional (RDB - Relational Database) seperti halnya bab 16 sebelumnya, yaitu untuk memperkirakan/menghasilkan nilai dari nilai null majemuk (multiple null values) dari database relasional yang memiliki rule-rule fuzzy terboboti.

17.1 Permasalahan Estimasi Nilai Null Majemuk

Pada bab 16 telah diterangkan bagaimana penerapan GSA untuk estimasi nilai null dalam pembangkitan fuzzy rule terboboti.

Pada bab ini kita mencoba mengestimasi banyak nilai null, tidak hanya satu saja seperti sebelumnya.

Kita ingat kembali prosedur CountCloseness pada gambar 16.2 sebelumnya. Lalu kita amati bagian di bawah ini, seperti terlihat pada gambar 17.1.

```
...
...
    {Then we find Estimated Salary and Error for every record}
    {if this record was null value, so we must find}
    {another record that closest to 1}
  if IsNullValue(i) and IsNullValue(IdxClosestCloseness) then begin
    PreferIdx:= GetPreferIdx;
    ETi.Salary:= Ti.Salary × GetClosenessValue(PreferIdx);
    if Tprefer-index.Salary <> 0 then
      Errori:= (ETi.Salary − Tprefer−index.Salary) / Tprefer−index.Salary
  end
  else begin
    ETi.Salary:= Ti.Salary × GetClosenessValue(IdxClosestCloseness);
    if Ti.Salary <> 0 then
      Errori:= (ETi.Salary − Ti.Salary) / Ti.Salary
  end;
```

Gambar 17.1 Bagian prosedur CountCloseness yang diamati

Selanjutnya pada tabel 17.1 diperlihatkan tabel dimana terdapat banyak nilai null, dan dengan GSA kita coba untuk mendapatkan kembali nilai null ini dari nilai-nilai yang masih ada.

Tabel 17.1 Contoh pelbagai nilai null pada suatu database relasional

EMP-ID	Degree	Experience	Salary
S1	Ph.D.	7.2	63,000
S2	Master	2.0	Null
S3	Bachelor	7.0	40,000
S4	Ph.D.	1.2	47,000
S5	Master	7.5	Null
S6	Bachelor	1.5	26,000
S7	Bachelor	2.3	29,000
S8	Ph.D.	2.0	50,000
S9	Ph.D.	3.8	54,000
S10	Bachelor	3.5	35,000
S11	Master	3.5	Null
S12	Master	3.6	41,000
S13	Master	10.0	Null
S14	Ph.D.	5.0	57,000
S15	Bachelor	5.0	36,000
S16	Master	6.2	50,000
S17	Bachelor	0.5	23,000
S18	Master	7.2	55,000
S19	Master	6.5	51,000
S20	Ph.D.	7.8	65,000
S21	Master	8.1	64,000
S22	Ph.D.	8.5	Null

Disebabkan terdapat proses pengecekan yang berhubungan dengan nilai null, maka kita dapat mengatur satu atau banyak nilai null yang diinginkan untuk diestimasi. Proses ini dilakukan dalam function GetPreferIdx seperti telah diperlihatkan pada gambar 16.4 sebelumnya.

Tentu saja sebagai batasan/kuota, paling tidak harus ada satu nilai dalam kolom/field **Salary** agar kita dapat mengestimasi nilai-nilai yang lain (jika ini adalah nilai null).

17.2 Percobaan

Percobaan

Kita jalankan program dengan pelbagai paramater, yang masing-masing dijalankan sebanyak 10 kali.

Kita dapatkan hasil seperti di bawah ini:

Percobaan Tipe 1

- Size Of Population = 60
- Number of Generations = 300
- Mutation Rate = 0.01 = 1%
- Initial Temperature = 100
- Alpha = 0.7
- Frozen Value = 0.00001
- Index of Null Values = 0 (ini artinya baris/tuple pertama dalam database relasional)

Untuk index of Null Values = 0 (ini artinya baris/ tuple pertama dalam database relasional) dapat dilihat pada tabel 17.2 di bawah ini.

Tabel 17.2 Hasil dari percobaan tipe 1 untuk baris pertama

Running #	Avg. Estimated Error	Total Time
1	0.009122	0h:22m:56s:46ms
2	0.009122	0h:22m:47s:188ms
3	0.00595	0h:22m:43s:484ms
4	0.00595	0h:22m:45s:500ms
5	0.005608	0h:22m:40s:922ms
6	0.005297	0h:22m:39s:219ms
7	0.005297	0h:22m:42s:437ms
8	0.003524	0h:23m:13s:922ms
9	0.003524	0h:22m:47s:47ms
10	0.003524	0h:22m:46s:281ms
Min	**0.003524**	**0h:22m:39s:219ms**
Average	**0.005692**	**0h:22m:48s:205ms**
Max	**0.009122**	**0h:23m:13s:922ms**

Untuk index of Null Values = 0 dan 1 (ini artinya baris/ tuple pertama dan kedua dalam database relasional) dapat dilihat pada tabel 17.3 berikut.

Tabel 17.3 Hasil dari percobaan tipe 1 untuk baris pertama dan kedua

Running #	Avg. Estimated Error	Total Time
1	0.005932516	0h:23m:0s:906ms
2	0.004748638	0h:23m:1s:281ms
3	0.004748638	0h:22m:59s:954ms
4	0.004376414	0h:22m:56s:937ms
5	0.004307416	0h:22m:53s:797ms
6	0.004307416	0h:22m:46s:844ms
7	0.002597917	0h:22m:52s:484ms
8	0.002543098	0h:22m:53s:688ms
9	0.002543098	0h:22m:53s:406ms
10	0.002036744	0h:22m:56s:109ms
Min	**0.002036744**	**0h:22m:46s:844ms**
Average	**0.003814189**	**0h:22m:55s:541ms**
Max	**0.005932516**	**0h:23m:1s:281ms**

Untuk index of Null Values = 0, 1 dan 2 dapat dilihat pada tabel 17.4 berikut.

Tabel 17.4 Hasil dari percobaan tipe 1 untuk baris 1, 2 dan 3

Running #	Avg. Estimated Error	Total Time
1	0.009309616	0h:23m:17s:62ms
2	0.009309616	0h:23m:10s:281ms
3	0.003133243	0h:23m:4s:672ms
4	0.003133243	0h:23m:9s:141ms
5	0.003133243	0h:23m:5s:422ms
6	0.003053508	0h:23m:2s:937ms
7	0.003053508	0h:23m:7s:172ms
8	0.002865291	0h:23m:11s:47ms
9	0.002865291	0h:23m:11s:422ms
10	0.002645873	0h:23m:12s:468ms
Min	**0.002645873**	**0h:23m:2s:937ms**
Average	**0.004250243**	**0h:23m:9s:162ms**
Max	**0.009309616**	**0h:23m:17s:62ms**

Untuk index of Null Values = 0, 1, 2 dan 3 dapat dilihat pada tabel 17.5 berikut.

Tabel 17.5 Hasil dari percobaan tipe 1 untuk baris 1, 2, 3 dan 4

Running #	Avg. Estimated Error	Total Time
1	0.002757452	0h:23m:27s:406ms
2	0.002034423	0h:23m:22s:313ms
3	0.001884244	0h:23m:20s:453ms
4	0.001884244	0h:23m:20s:469ms
5	0.001884244	0h:23m:19s:687ms
6	0.001884244	0h:23m:18s:360ms
7	0.001884244	0h:23m:20s:62ms
8	0.001884244	0h:23m:22s:875ms
9	0.001884244	0h:23m:22s:563ms
10	0.001884244	0h:23m:24s:218ms
Min	**0.001884244**	**0h:23m:18s:360ms**
Average	**0.001986583**	**0h:23m:21s:841ms**
Max	**0.002757452**	**0h:23m:27s:406ms**

Untuk index of Null Values = 0, 1, 2, 3 dan 4 dapat dilihat pada tabel 17.6 berikut.

Tabel 17.6 Hasil dari percobaan tipe 1 untuk baris 1, 2, 3, 4 dan 5

Running #	Avg. Estimated Error	Total Time
1	0.004815809	0h:23m:45s:500ms
2	0.003993718	0h:23m:46s:922ms
3	0.002474414	0h:23m:36s:156ms
4	0.002474414	0h:23m:39s:219ms
5	0.002474414	0h:23m:35s:843ms
6	0.002474414	0h:23m:33s:891ms
7	0.002474414	0h:23m:40s:875ms
8	0.002474414	0h:23m:40s:656ms
9	0.002436838	0h:24m:9s:657ms
10	0.002436838	0h:24m:42s:156ms
Min	**0.002436838**	**0h:23m:33s:891ms**
Average	**0.002852969**	**0h:23m:49s:88ms**
Max	**0.004815809**	**0h:24m:42s:156ms**

Untuk index of Null Values = 0, 1, 2, 3, 4 dan 5 dapat dilihat pada tabel 17.7 berikut.

Tabel 17.7 Hasil dari percobaan tipe 1 untuk baris 1, 2, 3, 4, 5 dan 6

Running #	Avg. Estimated Error	Total Time
1	0.00963128	0h:25m:0s:578ms
2	0.008187739	0h:24m:55s:828ms
3	0.008187739	0h:24m:52s:125ms
4	0.008187739	0h:24m:46s:94ms
5	0.008187739	0h:24m:48s:766ms
6	0.007679531	0h:24m:48s:109ms
7	0.007307143	0h:24m:47s:531ms
8	0.007294954	0h:24m:52s:610ms
9	0.005584063	0h:24m:57s:78ms
10	0.005584063	0h:24m:52s:703ms
Min	**0.005584063**	**0h:24m:46s:94ms**
Average	**0.007583199**	**0h:24m:52s:142ms**
Max	**0.00963128**	**0h:25m:0s:578ms**

Untuk index of Null Values = 0, 1, 2, 3, 4, 5 dan 6 dapat dilihat pada tabel 17.8 berikut.

Tabel 17.8 Hasil dari percobaan tipe 1 untuk baris 1, 2, 3, 4, 5, 6 dan 7

Running #	Avg. Estimated Error	Total Time
1	0.012435219	0h:26m:0s:546ms
2	0.012435219	0h:25m:45s:125ms
3	0.012435219	0h:25m:49s:547ms
4	0.012435219	0h:25m:54s:32ms
5	0.007584039	0h:25m:53s:890ms
6	0.007584039	0h:25m:54s:391ms
7	0.007584039	0h:25m:50s:312ms
8	0.007584039	0h:25m:46s:860ms
9	0.006789061	0h:25m:50s:218ms
10	0.006789061	0h:25m:56s:891ms
Min	**0.006789061**	**0h:25m:45s:125ms**
Average	**0.009365515**	**0h:25m:52s:181ms**
Max	**0.012435219**	**0h:26m:0s:546ms**

Untuk index of Null Values = 0, 1, 2, 3, 4, 5, 6 dan 7 dapat dilihat pada tabel 17.9 berikut.

Tabel 17.9 Hasil dari percobaan tipe 1 untuk baris 1, 2, 3, 4, 5, 6, 7 dan 8

Running #	Avg. Estimated Error	Total Time
1	0.010822578	0h:27m:12s:218ms
2	0.009297596	0h:27m:18s:485ms
3	0.009297596	0h:27m:1s:312ms
4	0.009297596	0h:27m:2s:63ms
5	0.009297596	0h:27m:44s:31ms
6	0.009297596	0h:28m:26s:109ms
7	0.009297596	0h:29m:48s:407ms
8	0.009297596	0h:28m:58s:812ms
9	0.008205089	0h:27m:18s:797ms
10	0.008205089	0h:38m:45s:266ms
Min	**0.008205089**	**0h:27m:1s:312ms**
Average	**0.009231593**	**0h:28m:57s:550ms**
Max	**0.010822578**	**0h:38m:45s:266ms**

Untuk index of Null Values = 0, 1, 2, 3, 4, 5, 6, 7 dan 8 dapat dilihat pada tabel 17.10 berikut.

Tabel 17.10 Hasil dari percobaan tipe 1 untuk baris 1, 2, 3, 4, 5, 6, 7, 8 dan 9

Running #	Avg. Estimated Error	Total Time
1	0.00732093	0h:49m:53s:547ms
2	0.00732093	0h:37m:49s:813ms
3	0.006890407	0h:30m:46s:46ms
4	0.0061114	0h:30m:18s:938ms
5	0.0061114	0h:30m:1s:0ms
6	0.0061114	0h:27m:1s:203ms
7	0.005756958	0h:27m:3s:625ms
8	0.005756958	0h:27m:6s:109ms
9	0.005267458	0h:27m:0s:422ms
10	0.005267458	0h:27m:8s:94ms
Min	**0.005267458**	**0h:27m:0s:422ms**
Average	**0.00619153**	**0h:31m:24s:880ms**
Max	**0.00732093**	**0h:49m:53s:547ms**

Untuk index of Null Values = 0, 1, 2, 3, 4, 5, 6, 7, 8 dan 9 dapat dilihat pada tabel 17.11 berikut.

Tabel 17.11 Hasil dari percobaan tipe 1 untuk baris 1, 2, 3, 4, 5, 6, 7, 8, 9 dan 10

Running #	Avg. Estimated Error	Total Time
1	0.019225473	0h:28m:36s:500ms
2	0.012163566	0h:28m:31s:531ms
3	0.012163566	0h:28m:38s:235ms
4	0.011188971	0h:28m:29s:390ms
5	0.010730344	0h:28m:23s:313ms
6	0.010730344	0h:28m:41s:312ms
7	0.010730344	0h:28m:34s:188ms
8	0.010730344	0h:28m:39s:15ms
9	0.010730344	0h:28m:26s:672ms
10	0.00988304	0h:28m:34s:860ms
Min	**0.00988304**	**0h:28m:23s:313ms**
Average	**0.011827634**	**0h:28m:33s:502ms**
Max	**0.019225473**	**0h:28m:41s:312ms**

Untuk index of Null Values = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 dan 10 dapat dilihat pada tabel 17.12 berikut.

Tabel 17.12 Hasil dari percobaan tipe 1 untuk baris 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 dan 11

Running #	Avg. Estimated Error	Total Time
1	0.014521021	0h:29m:28s:156ms
2	0.014521021	0h:29m:21s:922ms
3	0.014521021	0h:32m:17s:750ms
4	0.014521021	0h:33m:6s:860ms
5	0.006890925	0h:31m:44s:203ms
6	0.006890925	0h:32m:17s:797ms
7	0.006793239	0h:32m:21s:360ms
8	0.006793239	0h:30m:7s:343ms
9	0.00676117	0h:30m:17s:438ms
10	0.006020978	0h:30m:0s:844ms
Min	**0.006020978**	**0h:29m:21s:922ms**
Average	**0.009823456**	**0h:31m:6s:367ms**
Max	**0.014521021**	**0h:33m:6s:860ms**

Untuk index of Null Values = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 dan 11 dapat dilihat pada tabel 17.13 berikut.

Tabel 17.13 Hasil dari percobaan tipe 1 untuk baris 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 dan 12

Running #	Avg. Estimated Error	Total Time
1	0.021770419	0h:34m:5s:765ms
2	0.019711743	0h:33m:33s:797ms
3	0.015642323	0h:33m:43s:875ms
4	0.015642323	0h:33m:41s:16ms
5	0.015249024	0h:33m:27s:250ms
6	0.014037324	0h:33m:30s:531ms
7	0.012811843	0h:33m:47s:187ms
8	0.012811843	0h:33m:31s:641ms
9	0.012811843	0h:33m:35s:844ms
10	0.012811843	0h:33m:34s:437ms
Min	**0.012811843**	**0h:33m:27s:250ms**
Average	**0.015330053**	**0h:33m:39s:134ms**
Max	**0.021770419**	**0h:34m:5s:765ms**

Untuk index of Null Values = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 dan 12 dapat dilihat pada tabel 17.14 berikut.

Tabel 17.14 Hasil dari percobaan tipe 1 untuk baris 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 dan 13

Running #	Avg. Estimated Error	Total Time
1	0.015772303	0h:36m:9s:94ms
2	0.015772303	0h:35m:53s:781ms
3	0.011702639	0h:36m:5s:407ms
4	0.011702639	0h:38m:38s:171ms
5	0.011702639	0h:41m:51s:375ms
6	0.011702639	0h:40m:25s:672ms
7	0.005224971	0h:36m:7s:860ms
8	0.005224971	0h:41m:35s:828ms
9	0.004756509	0h:42m:24s:359ms
10	0.004345932	0h:43m:48s:906ms
Min	**0.004345932**	**0h:35m:53s:781ms**
Average	**0.009790754**	**0h:39m:18s:45ms**
Max	**0.015772303**	**0h:43m:48s:906ms**

Untuk index of Null Values = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 dan 13 dapat dilihat pada tabel 17.15 berikut.

Tabel 17.15 Hasil dari percobaan tipe 1 untuk baris 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 dan 14

Running #	Avg. Estimated Error	Total Time
1	0.019457373	0h:42m:1s:281ms
2	0.019457373	0h:37m:38s:547ms
3	0.01705593	0h:37m:34s:860ms
4	0.012688683	0h:37m:40s:718ms
5	0.012458219	0h:37m:28s:47ms
6	0.012458219	0h:37m:14s:188ms
7	0.012458219	0h:37m:41s:156ms
8	0.010389006	0h:37m:32s:953ms
9	0.009943857	0h:37m:36s:641ms
10	0.009713394	0h:37m:25s:422ms
Min	**0.009713394**	**0h:37m:14s:188ms**
Average	**0.013608027**	**0h:37m:59s:381ms**
Max	**0.019457373**	**0h:42m:1s:281ms**

Untuk index of Null Values = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 dan 14 dapat dilihat pada tabel 17.16 berikut.

Tabel 17.16 Hasil dari percobaan tipe 1 untuk baris 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 dan 15

Running #	Avg. Estimated Error	Total Time
1	0.042230748	0h:40m:39s:125ms
2	0.026024609	0h:40m:49s:750ms
3	0.026024609	0h:40m:46s:688ms
4	0.026024609	0h:43m:11s:781ms
5	0.022683928	0h:42m:56s:94ms
6	0.022683928	0h:42m:14s:922ms
7	0.022683928	0h:40m:8s:671ms
8	0.022683928	0h:41m:11s:329ms
9	0.022683928	0h:48m:1s:203ms
10	0.022683928	0h:48m:29s:15ms
Min	**0.022683928**	**0h:40m:8s:671ms**
Average	**0.025640814**	**0h:42m:50s:858ms**
Max	**0.042230748**	**0h:48m:29s:15ms**

Untuk index of Null Values = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 dan 15 dapat dilihat pada tabel 17.17 berikut.

Tabel 17.17 Hasil dari percobaan tipe 1 untuk baris 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 dan 16

Running #	Avg. Estimated Error	Total Time
1	0.02546262	0h:55m:2s:62ms
2	0.010683394	0h:52m:13s:797ms
3	0.010683394	0h:42m:11s:531ms
4	0.009199557	0h:42m:28s:141ms
5	0.009199557	0h:42m:32s:344ms
6	0.009199557	0h:43m:50s:797ms
7	0.009199557	0h:45m:44s:468ms
8	0.009199557	0h:48m:57s:360ms
9	0.009199557	0h:54m:33s:250ms
10	0.009199557	0h:50m:24s:422ms
Min	**0.009199557**	**0h:42m:11s:531ms**
Average	**0.011122631**	**0h:47m:47s:817ms**
Max	**0.02546262**	**0h:55m:2s:62ms**

Kesimpulan dari percobaan-percobaan di atas dapat disarikan pada tabel 17.18 berikut.

Tabel 17.18 Kesimpulan dari hasil dari percobaan tipe 1

# Null Values	Avg. Estimated Error	Total Time
1	0.005691855	0h:22m:48s:205ms
2	0.003814189	0h:22m:55s:541ms
3	0.004250243	0h:23m:9s:162ms
4	0.001986583	0h:23m:21s:841ms
5	0.002852969	0h:23m:49s:88ms
6	0.007583199	0h:24m:52s:142ms
7	0.009365515	0h:25m:52s:181ms
8	0.009231593	0h:28m:57s:550ms
9	0.00619153	0h:31m:24s:880ms
10	0.011827634	0h:28m:33s:502ms
11	0.009823456	0h:31m:6s:367ms
12	0.015330053	0h:33m:39s:134ms
13	0.009790754	0h:39m:18s:45ms
14	0.013608027	0h:37m:59s:381ms
15	0.025640814	0h:42m:50s:858ms
16	0.011122631	0h:47m:47s:817ms

Terlihat bahwa terjadi hubungan linier yang signifikan atas bertambahnya nilai null yang harus diestimasikan terhadap waktu yang diperlukan untuk mengestimasi, dan hal ini wajar terjadi. Untuk hasil estimasi, berdasarkan tingkat kesalahan yang diperoleh terlihat juga mengikuti hubungan tersebut, yaitu makin banyak nilai null yang diestimasikan makin besar juga kesalahan yang terjadi.

Bab 18 Variable-Centered Intelligent Rule System

Dalam bab ini akan dibahas mengenai sistem berbasis rule yang cerdas yang menitikberatkan pada variabel, sehingga ia disebut dengan Variable-Centered Intelligent Rule System (VCIRS). VCIRS merupakan thesis master penulis [Sub05c] dan telah dipublikasikan pada Information and Communication Technology Seminar 2005 [Sub05d]. Sedikit bahasa mengenai algoritma Rete pustakanya diambil dari Forgy [For82].

18.1 Gambaran Umum

Sistem Berbasis Aturan (SBA – Rule Base Systems (RBS)) adalah sistem yang baik untuk mendapat jawaban dari pertanyaan mengenai What (apa), How (bagaimana) dan Why (mengapa) dari Rule Base (RB) selama proses inferensia. Jawaban dan penjelasannya dapat disediakan dengan baik. Masalah yang dengan SBP adalah ia tak dapat secara mudah menjalankan proses akuisisi knowledge (pengetahuan) dan ia tak dapat mengupdate rule (aturan) secara otomatis. Hanya pakar yang dapat mengupdate Knowledge Base (KB) secara manual dengan dukungan dari knowledge engineer (insinyur pengetahuan). Lebih jauh kebanyakan peneliti dalam SBA lebih memperhatikan masalah optimasi pada rule yang sudah ada daripada pembangkitan rule baru dari rule yang sudah ada. Namun demikian, optimasi rule tak dapat mengubah hasil dari inferensia secara signifikan, dalam hal cakupan pengetahuan. Penjelasan selengkapnya mengenai RBS/SBA dapat dilihat kembali pada bab 5 dan bab 6.

Ripple Down Rule (RDR) datang untuk mengatasi permasalahan utama dari sistem pakar: pakar tak perlu lagi selalu mengkomunikasikan pengetahuan dalam konteks yang spesifik. RDR membolehkan akuisisi yang cepat dan sederhana secara ekstrim tanpa bantuan dari knowledge engineer. Pengguna tak perlu menguji RB dalam rangka mendefinisikan rule baru: pengguna hanya perlu untuk mampu mendefinisikan rule baru yang secara benar mengklasifikasikan contoh yang diberikan, dan sistem dapat menentukan dimana suatu rule harus ditempatkan dalam hirarki rulenya. Keterbasan dari RDR adalah kekurangan dalam hal inferensia yang berdayaguna. Tak seperti SBA yang dilengkapi dengan inferensia melalui forward dan backward chaining, RDR kelihatannya menggunakan Depth First Search (DFS) yang memiliki kekurangan dalam hal fleksibelitas dalam hal penjawaban pertanyaan dan penjelasan yang tumbuh dari inferensia yang berdayaguna. Penjelasan selengkapnya mengenai RDR ini dapat dilihat kembali pada bab 8.

Variable-Centered Intelligent Rule System (VCIRS) merupakan perkawinan dari SBA dan RDR. Arsitektur sistem diadaptasi dari SBA dan ia mengambil keuntungan-keuntungan yand ada dari RDR. Sistem ini mengorganisasi RB dalam struktur spesial sehingga pembangunan pengetahuan, inferensia pengetahuan yang berdayaguna dan peningkatan evolusional dari kinerja sistem dapat didapatkan pada waktu yang sama. Istilah "Intelligent" dalam VCIRS menekankan pada keadaan sistem ini yang dapat "belajar" untuk meningkatkan kinerja sistem dari pengguna sistem selama pembangunan pengetahuan (melalui analisis nilai) dan penghalusan pengetahuan (dengan pembangkitan rule).

18.2 Motivasi

Motivasi dari dibuatnya VCIRS adalah:

- Rule Base System (RBS) atau Sistem Berbasis Aturan (SBA)
  - Dengan inferensianya yang berdayaguna (powerful)
- Ripple Down Rules (RDR)
  - Dengan akuisisi pengetahuannya yang berdayaguna

Rule Base Systems

Menjawab pertanyaan seperti:

- What (apa) hasil dari proses inferensia?
- How (bagaimana) hal itu dilakukan?
- Why (mengapa) itu bisa dilakukan?

Strategi:

- Forward chaining
  - Clause sebagai premise/dasar pemikirian dari sebuah rule dicocokkan dengan data fakta, suatu pencocokan yang menyebabkan sebuah proses untuk menyatakan konklusi
- Backward chaining
  - Tujuannya adalah mencocokkan dengan fakta suatu konklusi dari beberapa rule; suatu pencocokan yang menyebabkan sebuah proses untuk menentukan apakah

premise clause cocok dengan data fakta.

Certainty Factor (CF, sebagian orang mengartikan sebagai confidence factor)

- Meningkatkan kemanfaatan dari ketidakpastian baik pada pengetahuan dan data fakta

Permasalahan

- Pembangunan pengetahuan
  - Diupdate secara manual oleh pakar dengan dukungan knowledge engineer

Ripple Down Rules

Memiliki kemampuan akuisisi pengetahuan (Knowledge Acquisition - KA) dengan cepat dan sederhana secara ekstrim.

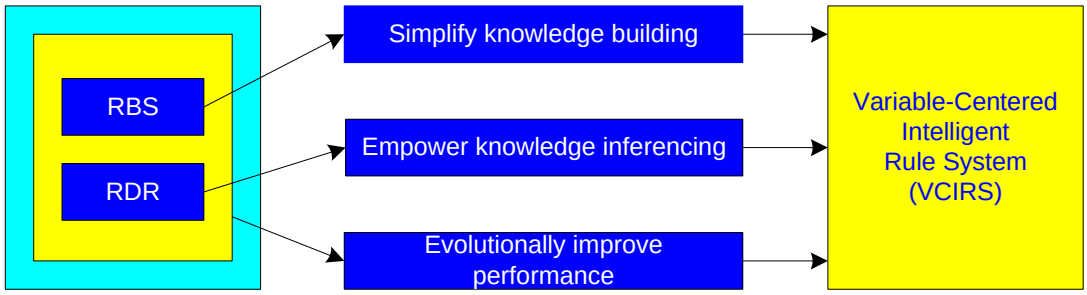
- Tanpa bantuan dari knowledge engineer
- Pengguna tidak perlu menguji RB untuk menentukan rule baru
  - Pengguna hanya perlu untuk mendefinisikan rule baru yang secara benar mengklasifikasikan contoh yang diberikan, dan sistem dapat menentukan dimana rule-rule itu seharusnya ditempatkan dalam hirarki rule.
  - Satu-satunya tugas akuisisi pengetahuan dalam RDR untuk pakar adalah memilih dari daftar kondisi (nantinya diimplementasikan dalam daftar perbedaan).
  - Sang pakar mempunyai tugas yang sangat terbatas dan tidak perlu terlibat dengan struktur KB.

Permasalahan

- Inferensia pengetahuan
  - Kelihatannya menggunakan Depth First Search (DFS) sehingga menimbulkan kekurangan dalam hal fleksibelitas penjawaban pertanyaan dan penjelasan yang tumbuh dari inferensia berdayaguna seperti yang disajikan dalam SBA.

18.3 METODE

Metode dari VCIRS dapat digambarkan pada gambar 18.1, yang merupakan teknik persilangan dari SBA dan RDR.



Gambar 18.1 Diagram metode VCIRS

VCIRS mempunyai struktur yang mengorganisasi RB sehingga pembangunan pengetahuan yang mudah, inferensia pengetahuan yang berdayaguna dan peningkatan evolusional sistem dapat didapatkan pada waktu yang sama.

Pertama, pembangunan pengetahuan disederhanakan dengan langkah-langkah sederhana dalam proses pembangunan pengetahuannya. Pengguna tidak perlu mempertimbangkan mengenai struktur KB dan dapat mengupdate KB secara langsung. VCIRS membolehkan pengguna untuk memperbaiki atau menambahkan node/rule kedalam KB yang telah ada. Seperti dalam RDR, perbaikan rule adalah pembuatan sebuah rule pengecualian untuk membuat benar pengklasifikasian yang salah, dimana penambahan mengacu pada penambahan rule baru pada level puncak tree KB. Sistem memandu pengguna selama proses pembangunan pengetahuan.

Inferensia pengetahuan dipertajam oleh pengetahuan (yaitu, hasil dari analisis variabel dan nilai) dari urutan derajat kepentingan (important degree) dan tingkat penggunaan (usage rate) dari case/kasus pada KB. Mekanisme inferensia SBA dibawa kembali dalam VCIRS, sehingga pengguna mendapatkan lebih banyak jawaban dan penjelasan dari inferensia.

Kinerja sistem ditingkatkan oleh struktur RB yang mendukung analisis variabel dan nilai untuk pembangkitan rule. Pembangkitan rule meningkatkan hasil dari inferensia dalam hal cakupan pengetahuan. Lebih jauh, analisis nilai juga memandu pengguna selama pembangunan dan inferensia pengetahuan. Bersamaan dengan pembangkitan rule, kemampuan ini dapat meningkatkan kinerja sistem dalam hal inferensia pengetahuan.

Kata "intelligent" digunakan dalam VCIRS adalah untuk menekankan bahwa sistem ini dapat "belajar" dari pengguna, selama pembangunan pengetahuan (yaitu analisis nilai)

dan perbaikan pengetahuan (yaitu pembangkitan rule). Lebih jauh, pembangkitan rule bersama dengan kemampuan sistem yang mampu untuk melakukan inferensia ala SBA, dapat secara evolusional meningkatkan kinerja sistem.

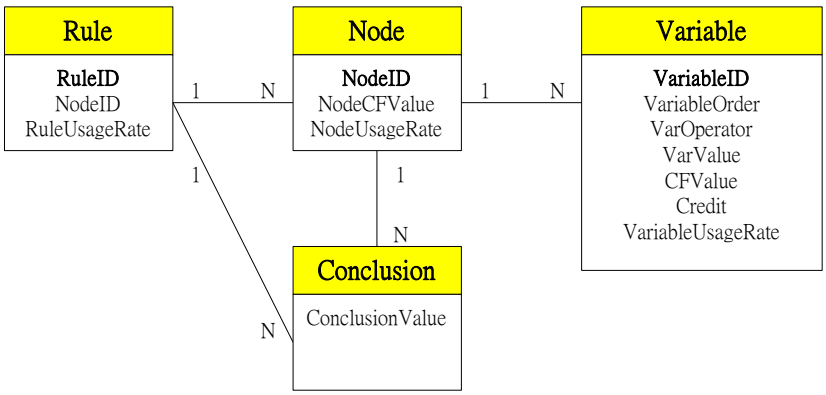
18.4 Modifikasi

VCIRS adalah sistem yang melakukan modifikasi terhadap sistem yang sudah ada (yakni SBA dan RDR) sebagai berikut:

- SBA
  - Pembangunan pengetahuan → mudah
- RDR
  - Inferensia → kemampuan inferensia ala SBA
- Kinerja sistem
  - Cakupan pengetahuan → ditingkatkan oleh pembangkitan rule

18.5 Definisi Istilah

Istilah-istilah berikut ini banyak digunakan dalam penjelasan VCIRS, sehingga dalam bagian ini diberikan definisi yang jelas untuk mencegah kesalahpahaman dan menghamparkan jalan yang nyaman untuk berdiskusi. Istilah-istilah dipresentasikan dalam BNF (Backus Naur Form). Gambar 18.2 membantu kita menangkap relasi konseptual diantara istilah-istilah yang dipakai.



Gambar 18.2 Relasi istilah-istilah

Sebuah rule adalah rangkaian dari node-node. Ia bisa memiliki satu atau lebih node. Sebuah node adalah rangkaian dari variabel-variabel. Ia bisa memiliki satu atau lebih variabel. Sebuah variabel memiliki pelbagai nilai (value). Sebuh node atau rule bisa memiliki satu atau lebih konklusi.

A. Rule Based Systems

/** **Rule Base**: sebuah rule base mengandung satu atau lebih rule. **/

```
<Rule Base> ::= <Rules>
<Rules> ::= <Rule> | <Rule> <Rules>
```

/** **Rule**: rule adalah elemen fundamental untuk membangun sebuah rule base. **/

```
<Rule> ::= <Rule ID> <Rule CF> IF <Clauses> THEN <Conclusions>
<Rule ID> ::= ALPHANUMERIC
<Rule CF> ::= [0..100]
<Clauses> ::= <Clause> | <Clause> <Clause Operator> <Clauses>
<Clause Operator> ::= AND
<Clause> ::= <Variable> <Operator> <Value> <Clause CF>
<Variable> ::= ALPHANUMERIC
<Operator> ::= "<" | "<=" | ">" | ">=" | "=" | "<>"
<Value> ::= ALPHANUMERIC | NUMERIC
<Clause CF> ::= [0..100]
<Conclusions> ::= <Conclusion> | <Conclusion> <Conclusion Operator>
              <Conclusions>
<Conclusion> ::= ALPHANUMERIC
<Conclusion Operator> ::= AND
```

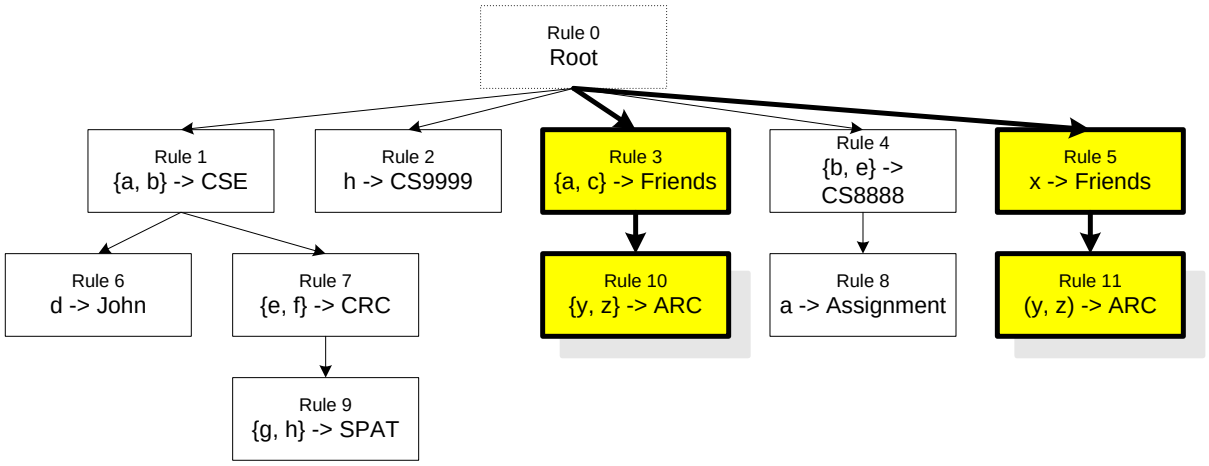
B. Variable-Centered Rule Structure

Catatan:

Rule adalah rangkaian dari satu atau lebih node.

Sebuah rule dalam VCIRS serupa dengan rangkaian rule dalam RBS/SBA dengan satu pengecualian bahwa konklusi dari rule adalah konklusi yang dimiliki oleh node terakhir. **Node** serupa dengan rule dalam SBA. Ia adalah rangkaian dari satu atau lebih variabel dan terdiri dari satu atau lebih konklusi.

Contoh pertama dari rule pada gambar 18.3 adalah Rule *John#1* yang memiliki 2 node: node *Rule 1* dan node *Rule 6*. Node *Rule 1* memiliki 2 variabel, yaitu *a* dan *b*. Node *Rule 6* memiliki satu variabel *d*. Rule *John#1* hanya memiliki satu konklusi, yaitu *John*.



Gambar 18.3 Gambaran knowledge base dalam bentuk tree/pohon di VCIRS

Contoh kedua adalah Rule *CSE#1* yang hanya memiliki 1 node, node *Rule 1*. Node *Rule 1* memiliki 2 variabel, yaitu *a* and *b*; dan Rule *CSE#1* memiliki satu konklusi, yaitu *CSE*.

/** **Rule Structure** **/

```
<Rule Structure> ::= <Rules>
<Rules> ::= <Rule> | <Rule> <Rules>
<Rule> ::= <Rule ID> <NodeIDOrders>
<NodeIDOrders> ::= <Node ID> <Node Order> | <Node ID> <Node Order>
             <NodeIDOrders>
<Rule ID> ::= ALPHANUMERIC
<Node ID> ::= ALPHANUMERIC
<Node Order> ::= NUMERIC
```

/** <Node ID> mengacu pada <Node ID> dalam Node Structure **/

/** **Node Structure** **/

```
<Nodes> ::= <Node> | <Node> <Nodes>
<Node> ::= <Parent Node ID> <Node ID> <Node CF Value> <Variables>
           <Conclusions> <Credit> <VUR> |
           <Parent Node ID <Node ID> <Node CF Value> <Variables>
           <Conclusions> <Credit> <VUR> <Node>
<Parent Node ID> ::= <Node ID>
<Node ID> ::= ALPHANUMERIC
```

<Node CF Value> ::= [0..100]

<Variables> ::= <Variable> | <Variable> <Variables>

<Variable> ::= <Variable ID> <Variable Order> <Value Part>

<Credit> ::= NUMERIC

/** Credit adalah jumlah kejadian/terjadinya suatu pengaksesan sebuah variabel dalam sebuah node **/

<VUR> ::= NUMERIC

/** VUR = Variable Usage Rate adalah nilai penggunaan suatu variabel dalam sebuah node **/

<Variable ID> ::= ALPHANUMERIC

<Variable Order> ::= NUMERIC

<Conclusions> ::= <Conclusion> | <Conclusion> <Conclusions>

<Conclusion> ::= ALPHANUMERIC

/** **Conclusion**: konklusi terdiri dari satu atau lebih nilai/value. **/

<Value Part> ::= <Operator> <Value> <CF Value>

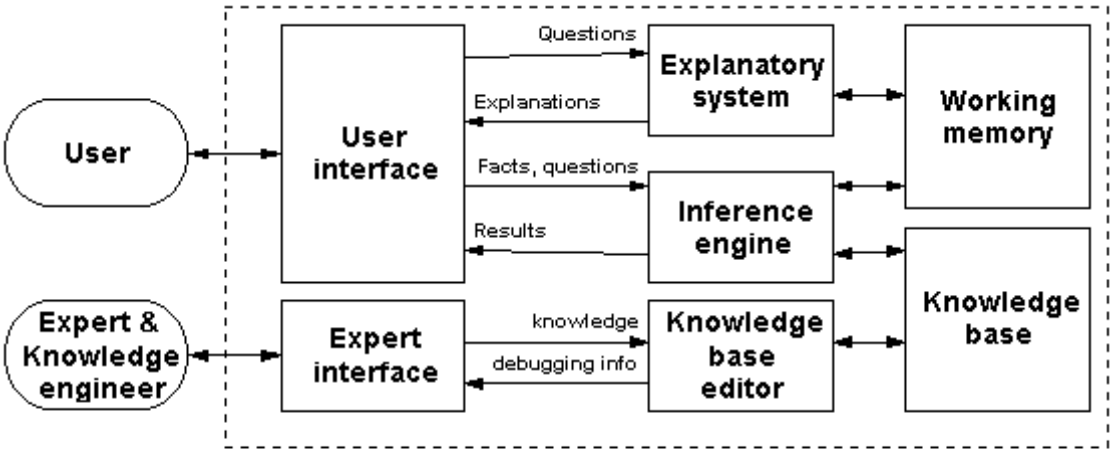
<Operator> ::= "<" | "<=" | ">" | ">=" | "=" | "<>"

<Value> ::= ALPHANUMERIC | NUMERIC

<CF Value> ::= [0..100]

18.6 Arsitektur Sistem

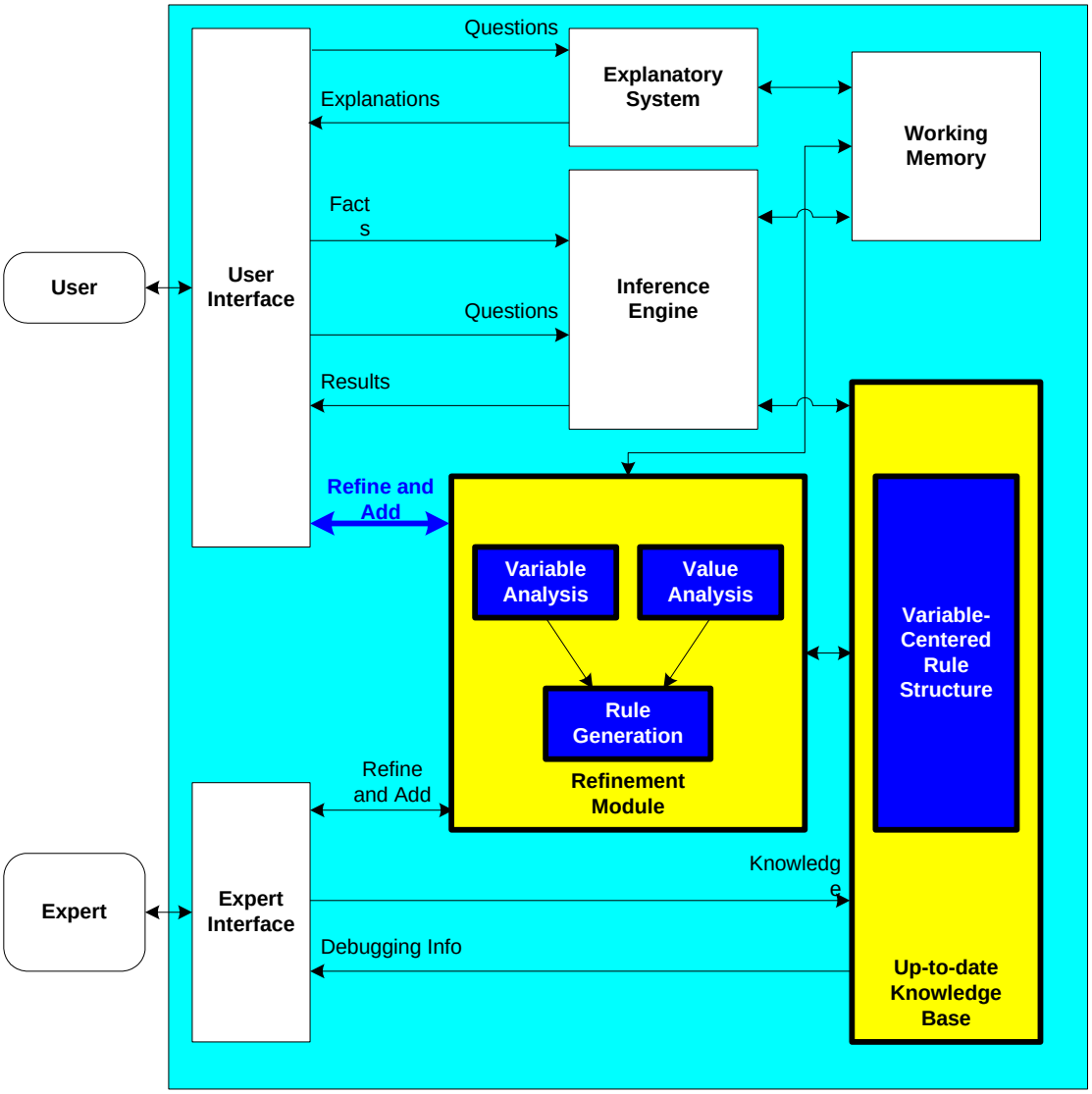
Gambar 18.5 memperlihatkan arsitektur dari VCIRS yang mengadaptasi arsitektur tradisional RBS (gambar 18.4). Sebuah modul baru yang disebut dengan Refinement Module (modul perbaikan) ditambakan untuk melaksanakan 3 tugas: analisis variabel, analisis nilai dan pembangkitan rule.



Gambar 18.4 Arsitektur SBA tradisional

Variable-Centered Rule Structure digunakan untuk merepresentasikan KB dan mendukung Refinement Module untuk mengelola KB yang up-to-date. Ia juga mencatat case-case/ kasus-kasus dan kejadiannya. Elemen fundamental dari Variable-Centered Rule Structure adalah variabel, yang ditempatkan/dipasang oleh pengguna. VCIRS mengelola secara cermat variabel ini mengenai nilainya, struktur dan kejadiannya. Rangkaian dari variabel membentuk node, sedangkan rangkaian dari node menyusun rule. Maka Variable-Centered Rule Structure mengandung struktur rule dan struktur node yang berpusat pada variabel-variabel.

Case yang dipresentasasikan oleh pengguna menuju ke memory kerja selama pembangunan pengetahuan, lalu disimpan secara permanen kedalam Variable-Centered Rule Structure disaat sistem menyimpan informasi rule dan menghitung kejadian dari setiap case. Lalu, informasi rule yang tersimpan tadi digunakan oleh Variabel Analysis (analisis variabel) untuk mendapatkan important degree (derajat kepentingan).



Gambar 18.5 Arsitektur VCIRS

Di sisi lain, kejadian dari setiap case digunakan oleh Value Analysis (analisis nilai) untuk mendapatkan usage degree (tingkat kegunaan). Usage degree akan membantu pengguna sebagai garis pedoman selama pembangunan dan inferensia pengetahuan untuk penentuan variabel mana yang diinginkan untuk dikunjungi pertama kalinya. Bersama dengan important degree, usage degree akan mendukung Rule Generation (pembangkitan rule) untuk memproduksi rule/node baru.

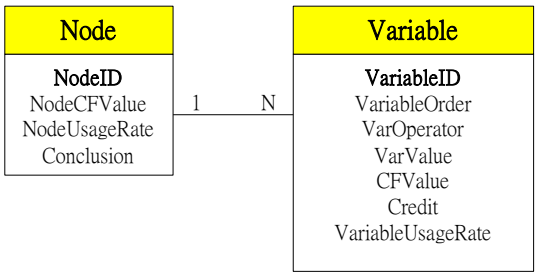
Di bagian selanjutnya akan dijelaskan VCIRS dalam level algoritmik.

18.7 Variable-Centered Rule Structure

Struktur ini mengandung 2 struktur, yang disebut dengan Node Structure (struktur node) dan Rule Structure (struktur rule). Struktur node menyimpan case yang dipresentasikan oleh pengguna dan menghitung kejadian dari setiap case. Ia serupa dengan sebuah rule

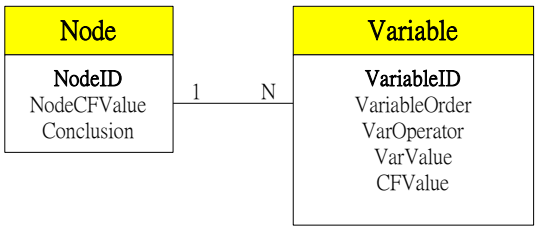
dalam RB dari SBA. Struktur rule menyimpan rangkaian dari node yang direpresentasikan oleh struktur node. Setiap node dalam KB memiliki rangkaian, yaitu paling tidak ada satu node untuk satu rule. Kedua struktur ini dijelaskan lebih jelas sebagai berikut ini.

18.7.1 Node Structure



Gambar 18.6 Node Structure

Gambar 18.6 menjelaskan graf konseptual dari struktur node. Diberikan case baru yang disediakan oleh pengguna, dari memory kerja VCIRS memasukkannya kedalam struktur node dan lalu menggunakan struktur rule sebagai bagian dari KB yang up-to-date. Struktur node juga menyimpan kejadian dari variabel-variabel dan node-node, untuk usage assignment (penugasan tingkat kegunaan). Setiap case terdiri dari kumpulan field data seperti digambarkan pada gambar 18.7.



Gambar 18.7 Case fields

VCIRS menggunakan <Variable ID> sebagai ID untuk sebuah variabel. ID ini unik, artinya sekali ID dari suatu variabel disimpan, sistem tak akan menyimpan variabel yang berbeda dengan ID yang sama.

ID untuk node diambilkan dari <Conclusion> pertama, yang disediakan oleh pengguna; <Node ID> akan dibuat dari <Conclusion> itu. Sebuah node adalah unik asalkan ia memiliki kesamaan variabel dan nilai variabel. Jika node berikutnya memiliki nama konklusi pertama yang sama dan memiliki kesamaan variabel, namun nilai variabelnya berbeda, maka <Node ID>nya akan sama dengan <Node ID> yang telah ada dengan penambahan nomor serial (#1, #2 ...) untuk membedakannya dengan yang telah dulu

ada.

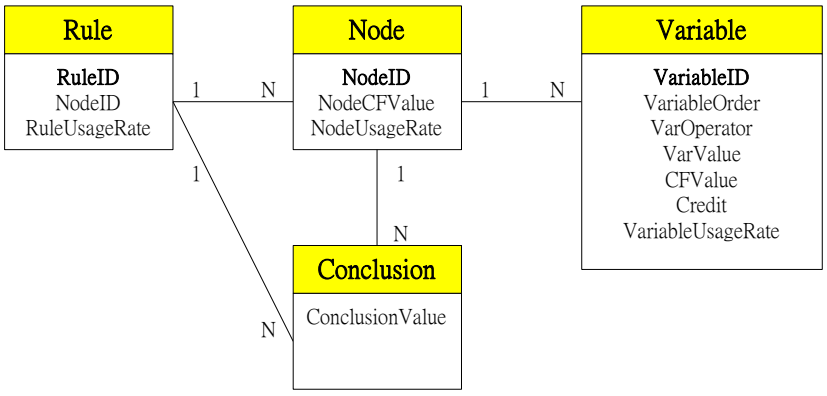
Setiap kali pengguna memasukkan case-nya, struktur node mengelola nilai dan posisinya. Informasi ini akan digunakan dalam usage assignment.

Dalam VCIRS pengguna dibolehkan untuk memperbaiki atau menambahkan node kedalam KB yang telah ada. Perbaikan rule adalah pembuatan dari rule pengecualian untuk membenarkan klasifikasi yang salah, sedangkan penambahan rule mengacu pada pengambahan rule baru pada level puncak dari tree di KB. Sistem memandu pengguna selama proses pembangunan pengetahuan, akan dijelaskan lebih detil dalam bagian selanjutnya.

Seperti halnya dalam RDR, rule (node) dalam VCIRS tak pernah dimodifikasi atau dihapus. Ini menjamin konsistensi data yang berkaitan dengan filosofi bahwa sistem ini adalah sistem rule "if-then" yang diorganisasikan dalam hirarki rule dan rule pengecualian (rule pengecualian ini mungkin juga memiliki rule pengecualian lagi, dan seterusnya) [Ho03]. Walaupun rule-rule tak pernah dihapus dari KB, sebuah rule dapat "dihentikan", untuk pengabaian suatu konklusi yang biasanya secara normal akan dihasilkan menggunakan rule ini. Namun demikian, hal ini tidaklah menyebabkan pengabaian konklusi yang didapat dari rule pengecualian (dan juga rule pengecualian dari rule pengecualian tadi, dan seterusnya).

Bersamaan dengan berjalannya inferensia pengetahuan, VCIRS menggunakan Node Structure (dan juga Rule Structure) untuk menstrukturkan source/sumber data. Rule Structure mengelola informasi rule, sedangkan Node Structure menjaga informasi mengenai kejadian (occurence) dari case-case yang disimpan. Pengguna dapat memasukkan fakta (case-case) dan mendapatkan hasil setelah inferensia.

18.7.2 Rule Structure



Gambar 18.8 Graf konseptual untuk Rule Structure

Gambar 18.8 menggambarkan graf konseptual dari Rule Structure. Seperti telah dinyatakan sebelumnya, case yang dimasukkan oleh penguna pertama kali disimpan dalam Node Structure; lalu ia akan digunakan dalam Rule Structure.

ID dari sebuah rule sama dengan <Node ID> terkecil dari setiap rule. Jika rule berikutnya memiliki candidate <Rule ID> yang sama maka sistem akan menamainya seperti penamaan pada node: nama dari rule yang telah ada diikuti dengan nomor serial (#1, #2 …).

18.8 Perbaikan Pengetahuan

Ada 3 tugas (task) dalam Refinement Module: analisis variabel, analisis nilai dan pembangkitan rule. Analisis variabel menentukan manakah variabel/node yang paling penting, sedangkan analisis nilai menentukan seberapa sering sebuah rule/node/variabel itu digunakan. Pembangkitan rule adalah hasil dari analisis variabel dan nilai. Perbaikan/penghalusan rule (node) dalam KB adalah pembuatan sebuah rule pengecualian (exception rule) untuk membenarkan klasifikasi yang salah. Sistem akan memperbaiki KB jika pengguna membuka KB, menurut jadwal atau atas permintaan pengguna. Desain ini membolehkan pakar untuk memperbaiki dan menambahkan KB secara langsung, sebagai tambahan permintaan yang bisa dilakukan ke Refinement Module seperti yang dapat dilakukan oleh pengguna biasa.

18.8.1 Variable Analysis

Di dalam Variable-Centered Rule Structure sistem mengetahui node mana yang di-share (sama-sama menggunakan) oleh pelbagai rule, dan variabel mana yang di-shared oleh node. Semakin banyak rule yang memakai suatu node; maka node tersebut akan semakin penting. Pertimbangan yang sama terjadi pada variabel yang di-share didalam node. Karena analisis ini sangat tergantung pada struktur implementasinya, kita akan menunda sejenak demonstrasi dari analisis ini sampai di bagian 18.8.2.

Fakta ini menunjukkan seberapa penting suatu node/variabel, sebagai titik awal untuk membangkitakan rule baru, bersama dengan analisis nilai.

18.8.2 Value Analysis

Analisis nilai yang ditawarkan oleh VCIRS didasarkan pada data yang dimasukkan oleh

pengguna. Tujuan dari analisis nilai ini adalah:

1. Memberikan garis pedoman kepada pengguna/sistem selama pembangunan pengetahuan, mengenai rule/node/variabel mana yang dikunjungi pertama kali pada penelusuran tree, contoh: rule/node/variabel yang paling banyak dipakai adalah pertama kali dikunjungi, diikuti oleh yang kedua paling banyak dipakai dan seterusnya. Ini akan membantu pengguna dalam menentukan case mana yang akan dimasuki.

2. Memberikan garis pedoman kepada pengguna/sistem selama inferensia pengetahuan, mengenai rule/node/variabel mana yang dikunjungi pertama kali dalam pencarian tree rule, sebagaimana dalam pembangunan pengetahuan. Ini berguna dalam membantu pengguna untuk fokus pada rule/node/variabel yang paling menarik dalam KB.

3. Memberikan garis pedoman kepada sistem, bersama dengan analisis variabel untuk proses pembangkitan rule. Dengan demikian sistem akan mendapat perspektif lain, sebagai tambahan pada kejadian sharing (sharing occurence) dari analisis variabel (yaitu node dan variabel) sebelum menentukan untuk membangkitkan rule. Dengan hasil dari analisis nilai sistem mengetahui derajat kegunaan (usage degree) dari rule/node/variabel, sedangkan dari analisis variabel sistem akan mengetahui derajat kepentingan (important degree) dari node/variabel dalam suatu rule/node.

Proses analisis nilai, yang disebut dengan usage assignment (pemberian nilai kegunaan), adalah untuk menentukan derajat kegunaan dari rule/node/variabel dalam KB. Usage assignment menggunakan informasi yang disimpan dalam Variable-Centered Rule Structure.

Kita memiliki tiga usage degree. Pertama, Variabel Usage Rate (VUR) digunakan untuk mengukur tingkat kegunaan dari suatu variabel di dalam node yang sedang dan telah digunakan. Kedua, kita menggunakan Node Usage Rate (NUR) untuk mengukur kegunaan suatu node pada pengeksekusian (firing). Yang terakhir adalah Rule Usage Rate (RUR) yang mengukur kegunaan suatu rule pada pengeksekusian (firing). Makin besar indikator kegunaan, maka makin bergunalah nilai tersebut dan begitu pula sebaliknya.

Persamaan (1) menghitung VUR untuk variabel ke-*i*, (2) menghasilkan NUR untuk node ke-*j*, sedangkan (3) mendefinisikan RUR untuk rule ke-*k*.

$$VUR_i = Credit_i \times Weight_i \quad (1)$$

$$NUR_j = \frac{\sum_{1}^{N} VUR_{ij}}{N}, \qquad VUR_{ij} \text{ untuk variabel ke-}i \text{ dalam node } j \qquad (2)$$

$$RUR_k = \frac{\sum_{1}^{N} NUR_{jk}}{N}, \qquad NUR_{jk} \text{ untuk node ke-}j \text{ dalam rule } k \qquad (3)$$

Dimana:

- $Credit_i$ = kejadian dari variable *i* dalam Node Structure (4)

Credit didapatkan dari Node Structure. Nilainya akan meningkat saat pengguna membuat node yang menyetujui nilai dari case lama.

- $Weight_i = NS_i \times CD_i$ (5)

Weight menghitung bobot (weight) dari variabel ke node yang memilikinya. Ada 2 faktor yang berkontribusi ke bobot dari sebuah variabel. Pertama adalah jumlah node yang berbagi (sharing) sebuah variabel, dan kedua adalah CD (Closeness Degree), yaitu derajat kedekatan sebuah variabel pada sebuah node.

NS_i = jumlah node yang berbagi (sharing) variabel *i* (6)

$$CD_i = \frac{VO_i}{TV} \qquad (7)$$

CD adalah singkatan dari Closeness Degree, yaitu derajat kedekatan sebuah variabel pada sebuah node. CD_i dalam node *j*, menghitung derajat kedekatan dari variable *i* dalam node *j*. Makin dekat sebuah variabel pada konklusi yang dipunyai suatu node, makin baiklah ia. Sehingga, CD mendasarkan dirinya pada urutan dari variabel dalam suatu node (catatan: node adalah rangkaian dari variabel-variabel). CD dihitung dengan urutan variabel VO, dibagi dengan total variabel TV, yang dimiliki oleh sebuah node.

VO_i = urutan dari variabel *i* dalam suatu node (8)

TV = total variabel yang dimiliki oleh suatu node (9)

18.8.3 Rule Generation

Pembangkitan rule bekerja berdasarkan hasil dari analisis variabel dan nilai. Perlu dicatat bahwa dari analisis variabel kita menghitung important degre dari suatu node/variabel, sedangkan dari analisis nilai kita dapatkan usage degree dari rule/node/variabel.

Informasi mengenai shared node/variable dari analisis variabel berguna untuk memilih

kandidat yang baik untuk membuat kombinasi. Shared node/variabel yang paling tinggi nilainya berarti bahwa ialah yang merupakan node/variabel terpenting dalam KB yang ada, sebab ia digunakan di banyak tempat pada struktur saat ini. Usage degree dengan nilai tertinggi yang diperoleh dari analisis nilai mengandung arti bahwa node/variabel tersebut memiliki kejadian (occurrence) tertinggi di dalam struktur.

Kombinasi variabel mengkombinasikan variabel untuk menghasilkan node baru, sedangkan kombinasi node mengkombinasikan node untuk menghasilkan rule baru. Kombinasi ini dapat dilakukan asalkan urutan dari variabel/node yang akan dihasilkan tidak menyalahi urutan variabel/node yang telah ada dalam KB. Disebabkan terdapat banyak kemungkinan kombinasi, kita akan memberikan pendekatan baru untuk melakukan kombinasi variabel dan node tersebut seperti di bawah ini.

Diberikan KB dengan *m* rule, dimana setiap rule (R) adalah rangkaian dari *n* node (N), dan setiap node adalah rangkaian dari *o* variabel (V) dan sebuah conclusion/konklusi (C). Nilai dari *n* untuk node yang dimiliki oleh sebuah rule bisa bervariasi sebagaimana halnya dengan nilai *o* untuk variabel yang dimiliki oleh sebuah node. Urutan dari setiap variabel dalam node memiliki arti seperti telah dijelaskan pada analisis nilai diatas. Menggunakan simbol ini, sebuah KB dapat direpresentasikan seperti gambar 18.9 berikut ini.

R_1: $N_{11}(V_{111}\rightarrow..\rightarrow V_{11o};C_{11})\rightarrow N_{12}(V_{121}\rightarrow..\rightarrow V_{12o};C_{12})\rightarrow..\rightarrow N_{1n}(V_{1n1}\rightarrow..\rightarrow V_{1no};C_{1n})$

R_2: $N_{21}(V_{211}\rightarrow..\rightarrow V_{21o};C_{21})\rightarrow N_{22}(V_{221}\rightarrow..\rightarrow V_{22o};C_{22})\rightarrow..\rightarrow N_{2n}(V_{2n1}\rightarrow..\rightarrow V_{2no};C_{2n})$

.

.

R_m: $N_{m1}(V_{m11}\rightarrow..\rightarrow V_{m1o};C_{m1})\rightarrow N_{m2}(V_{m21}\rightarrow..\rightarrow V_{m2o};C_{m2})\rightarrow..\rightarrow N_{mn}(V_{mn1}\rightarrow..\rightarrow V_{mno};C_{mn})$

Gambar 18.9 KB dipresentasikan oleh simbol-simbol

Kita bangkitkan rule dengan mengkombinasikan node-node berdasarkan urutan dari node-node yang telah ada dalam rule. Algoritma pembangkitan rule digambarkan pada gambar 18.10 di bawah ini.

1. Untuk setiap node terpenting (most important), yang dihasilkan oleh analisis variabel, pilihlah satu sebagai konklusi dari kandidat rule. Node ini menjadi node terakhir dari kandidat rule.
2. Susunlah node-node yang diproses tadi berdasarkan urutan relatif node yang dihitung oleh "Algoritma penghitungan urutan relatif node" (gambar 18.12). Tambahkan karakter "G" kedalam <Rule ID> untuk membedakannya dengan rule yang telah ada, "G" diartikan dengan system-generated rule (rule yang dihasilkan oleh sistem).
3. Tampilkan rule ke pengguna untuk konfirmasi lebih dahulu sebelum rule tersebut disimpan sebagai rule tambahan dalam KB.

Gambar 18.10 Algoritma pembangkitan rule

"Algoritma penghitungan urutan relatif node" ditunjukkan pada gambar 18.12. Dalam gambar 18.11, "CurrentRule" menyimpan the RuleID yang sedang diproses; dimulasi dari rule yang memiliki RUR terendah. "NodeOrderQueue" menyimpan urutan dari node-node dari rule yang sedang diproses dalam "CurrentRule", yang prosesnya dimulai dari urutan pertama. "RuleUsed" menyimpan setiap rule yang berbagi (sharing) suatu node dalam "NodeOrderQueue". "PreCandidateNode" menyimpan node-node yang dimiliki oleh rule-rule dalam "RuleUsed" yang sedang dibandingkan, sebelum disimpan ke "CandidateNode". Pemilihan node dalam "PreCandidateNode" adalah berdasarkan NUR dari node-node. Node yang memiliki NUR terendah akan diambil lebih dulu. "CandidateNode" menyimpan urutan relatif node. "RuleStack" mem-push (mendorong) rule ke stack (tumpukan), setelah suatu rule selesai diproses dalam "CurrentRule".

Step	Current Rule (RUR)	Node Order Queue (NUR)	Rule Used (RUR)	Pre Candidate Node (NUR)	Candidate Node	Rule Stack
1	...	...	...	...	...	...
2	...	...	...	...	...	...

Gambar 18.11 Struktur data penghitungan urutan relatif node

1. Dimulai dari rule dengan RUR terendah, ambil sebuah rule dan tempatkan dalam CurrentRule. Dapatkan semua node dan masukkan mereka kedalam NodeOrderQueue.
2. Dimulai dari node pertama pada NodeOrderQueue.

   Jika sebuah node di-shared oleh rule yang lain, dapatkan rule tersebut dan masukkan ke RuleUsed jika rule ini belum berada pada RuleUsed atau RuleStack. Jika ada lebih dari satu rule, pengambilan node akan dipilih didasarkan pada NUR terendah, yang terendah akan diambil lebih dulu. Dapatkan semua node pada urutan sebelum node yang sedang diproses saat ini dan juga node itu sendiri dari semua rule dalam PreCandidateNode. Hapus node tersebut dari NodeOrderQueue. Masukkan node dari PreCandidateNode ke CandidateNode jika node itu belum ada dan ini berdasarkan urutannya. Jika urutannya sama, node dengan NUR yang lebih rendah akan lebih dulu diambil.
3. Jika rule dalam RuleUsed tidak lagi punya node lebih lanjut, masukan ia kedalam RuleStack dan hapus ia dari RuleUsed.
4. Jika rule dalam CurrentRule yang memiliki node yang sedang diproses tidak lagi memiliki node lebih lanjut, masukkan ia kedalam RuleStack, dan hapus ia dari CurrentRule.
5. Dapatkan sebuah rule dari RuleUsed yang memiliki RUR terendah dan tempatkan ia dalam CurrentRule. Jika RuleUsed sudah kosong, tetapi masih ada rule dalam KB yang belum muncul dalam RuleStack, ambil rule dari rule/rule-rule tadi dimulai dari rule dengan RUR terendah. Kembali ke langkah 2 dan kerjakan semua langkah sampai semua rule dalam KB muncul dalam RuleStack.

Gambar 18.12 Algoritma penghitungan urutan relatif node

Proses penghitungan urutan relatif node memakan waktu lama karena ia harus mengecek setiap rule dalam *n* rule di KB. Juga jika pada node yang sedang diproses terdapat rule yang saling berbagi (sharing) dengan node ini, dilakukan proses pembandingan setiap node dalam setiap shared rules. Dengan demikian, kompleksitas waktu dari proses ini = O ($n \times$ jumlah rule yang berbagi (sharing) node $\times$ jumlah node dalam setiap shared rules).

Pada kenyataannya, dari gambar 18.12 juga bisa didapatkan urutan relatif rule dari RuleStack dengan mengeluarkan rule-rule dari stack.

Selama proses pembangkitan rule kita juga dapat melakukan pembangkitan node. Variabel terakhir dari kandidat node didapatkan dari variabel yang terpenting (most important variable). Kita bangkitkan node dengan mengkombinasikan variabel-variabel menurut urutan relatif variabel yang telah ada dalam node. Algoritma pembangkitan node dijelaskan dalam gambar 18.13 di bawah ini.

1. Untuk setiap variabel yang terpenting (most important variable), yang dihasilkan oleh analisis variabel pilihlah satu-satu sebagai variabel terakhir dari kandidat node.
2. Susunlah variabel-variabel yang sedang diproses tadi menurut urutan relatif yang dihitung dengan "Algoritma penghitungan urutan relatif variabel" (gambar 18.15). Tambahkan karakter "G" kedalam <Node ID> untuk membedakannya dari variabel-variabel yang telah ada, "G" menandakan variabel yang dihasilkan oleh sistem (system-generated variable).
3. Tampilkan rule kepada pengguna untuk konfirmasi sebelum ia disimpan sebagai node tambahan dalam rule terbangkitkan (generated rule) dalam KB.

Gambar 18.13 Algoritma pembangkitan node

"Algoritma penghitungan urutan relatif variabel" diperlihatkan pada gambar 18.15. Pada gambar 18.14, "CurrentNode" menyimpan NodeID yang sedang diproses; dimulai dari ode dengan NUR terendah. "VariableOrderQueue" menyimpan urutan dari variabel-variabel dari node yang sedang diproses dalam "CurrentNode", yang prosesnya dimulai dari urutan pertama. "NodeUsed" menyimpan setiap node yang berbagi (sharing) suatu variable dalam "VariableOrderQueue". "PreCandidateVariable" menyimpan variabel yang dimiliki oleh node dalam "NodeUsed" yang sedang dibandingkan, sebelum disimpan ke "CandidateVariable". Pemilihan variabel dalam "PreCandidateVariable" didasarkan pada VUR dari node. Variabel yang mempunya VUR terendah akan diambil lebih dulu. "CandidateVariable" menyimpan urutan relatif variabel. "NodeStack" memasukkan (push) node ke stack, setelah suatu node selesai diproses dalam "CurrentNode".

Step	Current Node (NUR)	Variable Order Queue (VUR)	Node Used (NUR)	Pre Candidate Variable (VUR)	Candidate Variable	Node Stack
1	...	...	...	...	...	...
2	...	...	...	...	...	...

Gambar 18.14 Struktur data penghitungan urutan relatif variabel

1. Dimulai dari node dengan NUR terendah, ambillah sebuah node dan tempatkan dalam CurrentNode. Dapatkan semua <Variable ID><VarOperator><VarValue> dan masukkan ke VariableOrderQueue.
2. Dimulai dari variabel pertama dari VariableOrderQueue.

   Jika suatu variabel sama-sama digunakan (shared) oleh beberapa node yang lain, temukan node-node itu dan masukkan ke NodeUsed jika node tersebut belum ada dalam NodeUsed atau NodeStack. Jika ada lebih dari satu node, pengambilan variabel akan didasarkan pada VUR, yaitu yang terendah akan diambil lebih dulu. Dapatkan semua variabel dalam urutan sebelumnya dari variabel yang sedang diproses dan juga variabel itu sendiri dari semua node dalam PreCandidateVariable. Hapus variabel saat ini dari VariableOrderQueue. Masukkan variabel dari PreCandidateVariable ke CandidateVariable jika variabel tersebut belum ada dan hal ini didasarkan pada urutannya. Jika urutannya sama, maka variabel dengan VUR yang lebih rendah akan diambil lebih dulu.
3. Jika node dalam NodeUsed tidak lagi mempunyai variabel lebih lanjut, masukkan ia kedalam NodeStack dan hapus node itu dari NodeUsed.
4. Jika node dalam CurrentNode yang memiliki variabel yang sedang diproses tidak mempunyai variabel lebih lanjut, masukkan ia kedalam NodeStack, dan hapus node itu dari CurrentNode.
5. Dapatkan suatu node dari NodeUsed yang memiliki NUR terendah dan tempatkan ia kedalam CurrentNode. Jika NodeUsed telah kosong, tetapi masih ada node dalam KB yang belum muncul dalam NodeStack, ambil node dari node-node yang memiliki NUR terendah. Kembali ke langkah 2 dan kerjakan semua langkah sampai semua node dalam KB muncul dalam NodeStack.

Gambar 18.15 Algoritma penghitungan urutan relatif variabel

Proses penghitungan urutan relatif variabel memakan waktu lama karena ia harus

mengecek setiap node dalam *n* node di KB. Juga jika pada variabel yang sedang diproses terdapat node yang saling berbagi (sharing) dengan variabel ini, dilakukan proses pembandingan setiap variabel dalam setiap shared nodes. Dengan demikian, kompleksitas waktu dari proses ini = *O* (*n* × jumlah node yang berbagi (sharing) variabel × jumlah variabel dalam setiap shared nodes).

Pada kenyataannya, dari gambar 18.15 juga bisa didapatkan urutan relatif node dari NodeStack dengan mengeluarkan node-node dari stack.

18.9 Pembangunan Pengetahuan

Arsitektur sistem dari VCIRS mendukung tiga operasi yang berhubungan dengan KB. Pertama, terdapat pembangunan pengetahuan yang membolehkan pengguna untuk membuat KB dari tidak ada sama sekali (scratch) atau untuk mengupdate KB yang telah ada. Kedua, kita memiliki perbaikan pengetahuan. Ini membolehkan pengguna untuk mendapatkan important degree (derajat kepentingan) dan usage degree (derajat kegunaan) dari suatu variabel/node/rule, atau untuk membangkitkan rule baru. Ketiga, kita memiliki inferensia pengetahuan untuk melakukan inferensia/reasoning dari KB. Ada dua pendekatan inferensia yang dipakai dalam operasi ini, yaitu pendekatan RBS dan RDR.

Serupa dengan RDR, VCIRS dapat membangun KB dari scratch; yaitu tidak tersedianya KB sama sekali, namun pengguna menginginkan untuk membangun KB baru. VCIRS membangun KB baru berdasarkan case-case yang disediakan oleh pengguna.

Berdasarkan isi dari suatu case, VCIRS akan mencari dalam KB untuk node yang layak. Node disini serupa dengan rule dalam RBS, yang mengandung satu atau lebih variabel bersamaan dengan nilainya (clause part), dan satu atau lebih konklusi (conclusion part). Gambar 18.16 menjelaskan algoritma dari pembangunan pengetahuan.

I. Jika node yang layak ditemukan, sistem akan memberikan pengguna satu dari beberapa pilihan berikut ini.

1. Jika ia tidak setuju dengan node saat itu, ia dapat membuat node baru pada level puncak KB. Secara otomatis, sistem akan membuatkan rule ba untuk node baru ini. Variabel-variabel yang tak tersetujui dari case yang dimasukkan pengguna akan disimpan dibawah posisi node baru.
2. Jika ia setuju dengan satu atau lebih variabel dalam node saat itu, tetapi tidak menyetujui dengan yang lainnya; ia dapat membuat node pengecualian yang mengandung variabel-variabel yang tidak disetujuinya tadi. Variabel yang tersetujui dari case yang dimasukkan pengguna akan disimpan dalam kolom Credit dari Node Structure dibawah posisi dari node lama (parent node), sedangkan variabel yan tak tersetujui akan disimpan dibawah posisi node baru.
3. Ia dapat memilih untuk melanjutkan penelusuran KB tanpa merubah apa pun. Disini Variable-Centered Rule Structure tak mendapat apa pun dari pengguna. Kerja sistem kalau seperti ini seperti sebuah proses inferensia dan membolehkan pengguna untuk menguji KB, yaitu mekanisme verifikasi sambil jalan (verification-on-the-fly mechanism).

II. Jika tak ada satu pun node yang ditemukan, sistem akan meminta pengguna untuk membuat node baru pada level puncak dari KB dan case yang dimasukkan pengguna akan disimpan di bawah posisi node baru tadi.

Gambar 18.16 Algoritma pembangunan pengetahuan

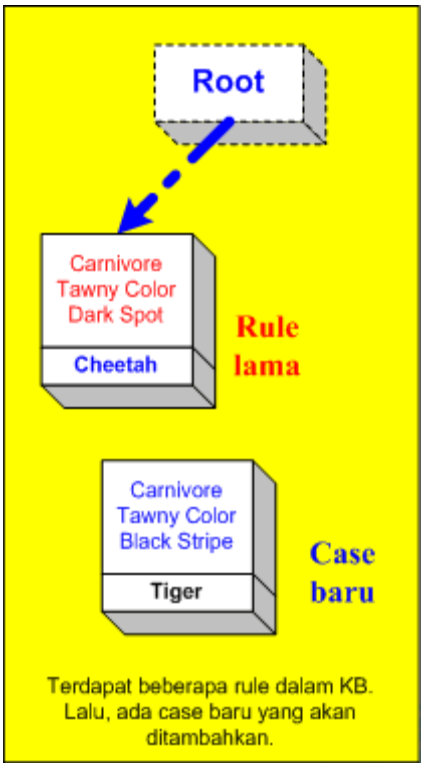
Pencarian node yang layak dalam KB dilakukan oleh algoritma seperti nampak pada gambar 18.17 berikut ini.

1. Cari semua node yang memiliki variable ID dan conclusion ID yang sama dengan case yang dimasukkan pengguna, dari Node Structure dan Conclusion tables. Hapus IDs yang sama/berlebihan (redundant). Jadikan ini sebagai node pre-candidate. Jika lebih dari satu node ditemukan dalam node pre-candidate, maka ikutilah urutan prioritas di bawah ini untuk menemukan kandidat node.
2. Prioritas pertama, temukan node-node sempurna (perfect nodes), yaitu node-node yang mengandung variable IDs dan nilai-nilai variabel (variable values) yang sama yang juga memiliki nilai-nilai konklusi (conclusion values) yang sama. Jika perfect nodes ditemukan, simpan ia kedalam kandidat node-node (candidate nodes).
3. Prioritas kedua; temukan node-node yang mengandung variable IDs yang sama (tanpa memiliki values/nilai-nilai yang sama) yang juga memiliki nilai-nilai konklusi (conclusion values) yang sama. Jika node seperti ini ditemukan, simpanlah kedalam kandidat node-node.
4. Prioritas ketiga; temukan node-node yang, paling tidak memiliki variable ID yang sama yang juga memiliki paling tidak satu conclusion value yang sama. Jika node seperti ini ditemukan, simpanlah kedalam kandidat node-node.
5. Prioritas keempat; temukan node-node yang mengandung paling tidak satu variable ID yang sama. Jika node seperti ini ditemukan, simpanlah kedalam kandidat node-node.
6. Prioritas kelima; temukan node yang memiliki paling tidak satu conclusion value yang sama. Jika node seperti ini ditemukan, simpanlah kedalam kandidat node-node.
7. Jika tak ada node yang ditemukan dalam pre-candidate nodes, sistem akan memberitahu pengguna bahwa case yang dia masukkan tak cocok dengan sembarang case dalam KB.
8. Proses di atas berhenti saat pengguna puas dengan sebuah node atau The pengguna memutuskan untuk berhenti/memulai lagi proses dari awal lagi.

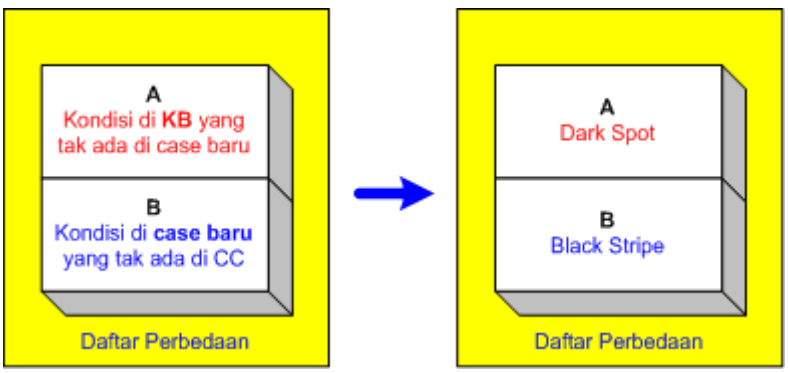
Gambar 18.17 Algoritma pencarian node yang layak

Jika sembarang kandidat node ditemukan, maka sistem akan menyajikan kepada pengguna beberapa opsi/pilihan yang dapat dikerjakan untuk proses selanjutnya. Pengguna selanjutnya memilih apakah salah satu atau keduanya dari case pengecualian yang terjadi sebagai case baru untuk node baru.

Untuk lebih memudahkan, maka untuk setiap case baru yang diolah dalam VCIRS (seperti digambarkan pada gambar 18.18 di bawah), dibandingkan dengan rule-rule yang telah ada dalam KB lalu dibuatlah satu Daftar Perbedaan yang memuat 2 bagian, yaitu bagian untuk rule lama di KB (A) dan bagian untuk rule baru (B), seperti digambarkan pada gambar 18.19 di bawah ini.



Gambar 18.18 Gambaran case baru yang akan ditambahkan dalam KB



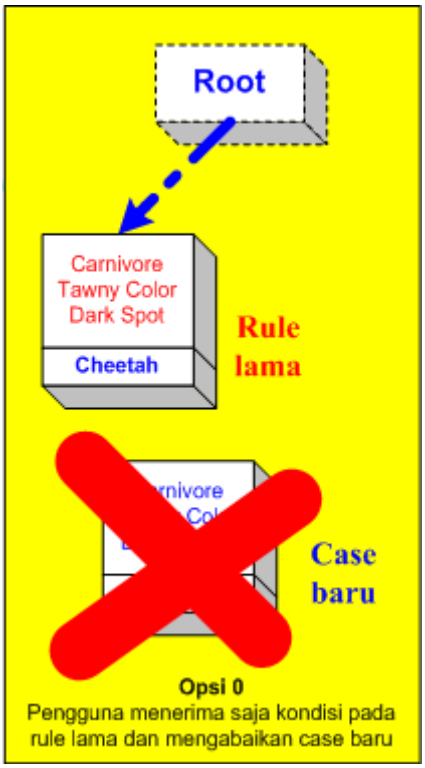
Gambar 18.19 Daftar Perbedaan

Selanjutnya VCIRS akan memberikan 6 opsi kepada pengguna, berdasarkan Daftar Perbedaan yang telah dibuat tadi, seperti dijabarkan dalam algoritma pembuatan node pada gambat 18.20 berikut ini.

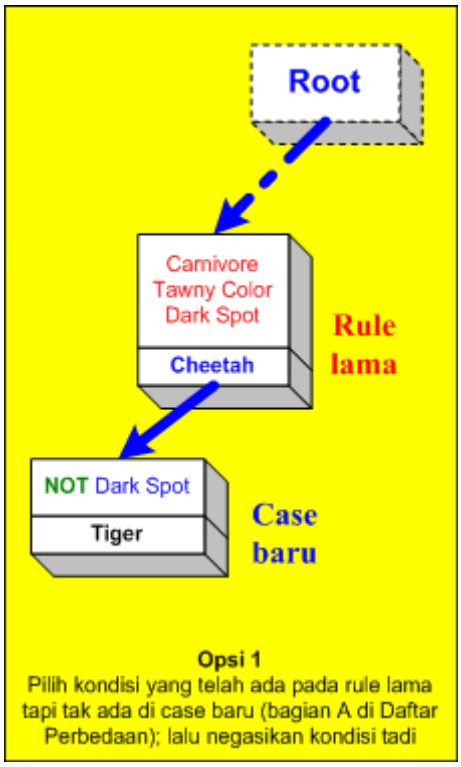
0. Pengguna menerima rule yang telah ada, tak memasalahkan adanya case baru yang tersedia → inferensia saja
1. Rule baru yang akan dihasilkan adalah negasi dari kondisi pada bagian A di Daftar Perbedaan
2. Rule baru yang akan dihasilkan adalah kondisi pada bagian B di Daftar Perbedaan
3. Rule baru yang akan dihasilkan adalah gabungan opsi 1 dan opsi 2
4. Rule baru yang akan dihasilkan semuanya berasal dari case baru yang tersedia tanpa melihat Daftar Perbedaan, namun letaknya adalah pada level yang sama dari rule lama yang sedang dicek tadi. Artinya rule baru ini menjadi saudara termuda dari rule lama yang sedang dicek atau dengan kata lain rule baru ini akan memiliki orang tua yang sama dengan rule lama, hanya saja karena urutan dia adalah yang terakhir dibentuk maka rule baru tersebut menjadi anak yang paling muda dari orang tuanya rule lama yang sedang dicek tadi.
5. Rule baru yang akan dihasilkan semuanya berasal dari case baru yang tersedia tanpa melihat Daftar Perbedaan, dan letaknya langsung di bawah root imajiner. Dengan kata lain ini adalah rule yang terletak pada level puncak KB.

Gambar 18.20 Algoritma pembuatan node

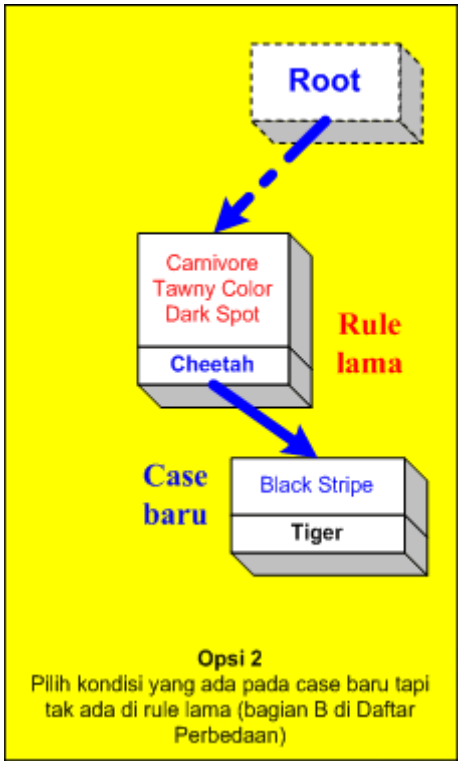
Penjelasan dari opsi-opsi di atas disajikan dalam gambar 18.21-18.26 berikut ini.



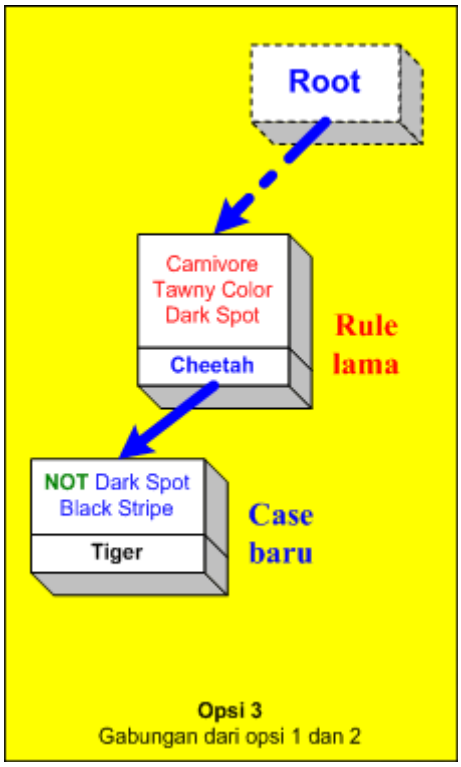
Gambar 18.21 Representasi opsi 0



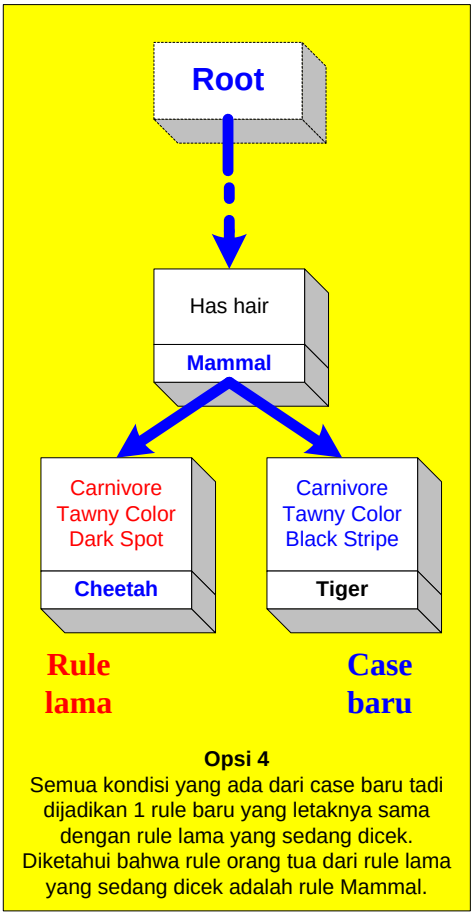
Gambar 18.22 Representasi opsi 1



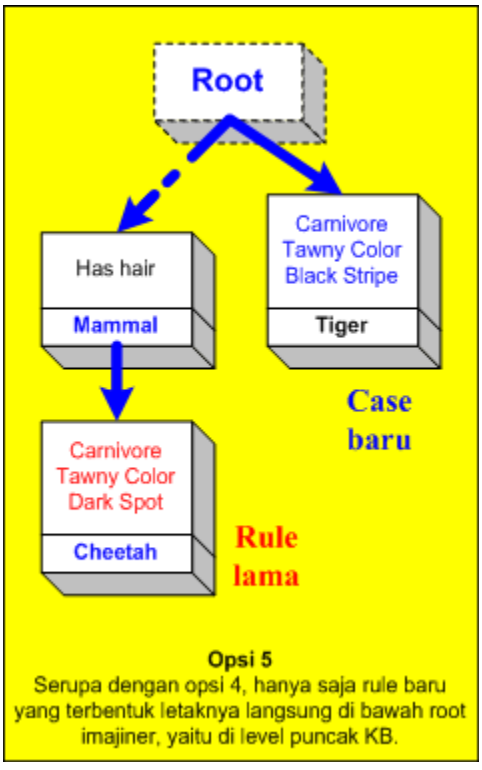
Gambar 18.23 Representasi opsi 2



Gambar 18.24 Representasi opsi 3



Gambar 18.25 Representasi opsi 4



Gambar 18.26 Representasi opsi 5

Setiap node baru dibuat akan menjadikan sistem mengupdate kejadian (oocurence) dari nilai-nilai (values) dalam Node Structure. Saat node baru sedang dibuat, nilai dari variabel yang ada baik pada rule lama maupun case baru akan disimpan dalam Node Structure di bawah posisi rule lama (yaitu disimpan pada kolom/field Credit).

Disaat pengguna membuat sebuah node, itu berarti ia juga membuat rule baru, karena node baru itu adalah node yang tidak memiliki anak. Nama dari rule baru ini didapatkan dari node terakhir, sehingga nama dari node baru tadi menjadi nama dari rule baru tersebut.

Proses pembangunan berhenti setelah pengguna puas dengan node yang menjadi konklusi finalnya, atau pengguna memutuskan untuk berhenti/memulai lagi proses dari awal lagi. Selama proses, pengguna dapat mengulang melakukan aksi-aksi di atas untuk membuat peningkatan KB yang ada atau menelusuri node-node untuk memverifikasi/menguji proses inferensia. Hal terakhir ini mengimplikasikan bahwa sistem kita juga melakukan verifikasi sambil jalan (verification-on-the-fly), seperti yang dilakukan oleh RDR, yang tidak dilakukan oleh RBS. Hal ini akan menjamin KB untuk menjadi KB seperti yang diinginkan pengguna.

Untuk mendayagunakan proses pembangunan pengetahuan, disaat pengguna sedang memasukkan case-nya, maka sistem akan memandu pengguna dengan derajat kegunaan (usage degree) dari rule/node/variabel dalam KB. Dengan garis pedoman ini, pengguna mengetahui dengan mudah status dari setiap variabel/konklusi dalam case yang sedang dimasukkannya: apakah nilai ini ada dalam KB atau status dari variabel yang paling banyak digunakan (most usage variable). Informasi ini akan membantu pengguna untuk memutuskan dari variabel/node/rule mana dia akan memulai membangun pengetahuan dan mengupdate KB.

18.10 Inferensia Pengetahuan

Inferensia pengetahuan secara mudah adalah proses pembangunan pengetahuan tanpa aksi yang dilakukan oleh pengguna. Pengguna memasukkan case dan sistem berjalan melalui penelusuran proses, saat VCIRS melakukan proses forward chaining sederhana.

Diilhami dari RDR, VCIRS cenderung menggunakan pendekatan cornerstone case untuk melaksanakan proses inferensia.

Dalam VCIRS proses penelusuran kelihatan seperti DFS, namun sesungguhnya jelas berbeda. Dengan Variable-Centered Rule Structure dimana setiap variabel (elemen fundamental dalam VCIRS) disimpan bersama dengan posisinya, proses inferensia dapat

berjalan sangat cepat. VCIRS menemukan sebuah variabel, node atau rule dengan mudah melalui posisi mereka.

Selama proses inferensia, VCIRS memperlakukan sebuah rule sebagai rangkaian dari node (rule dalam RBS). Ia mengabaikan isi konklusi dari setiap node, kecuali konklusi pada node terakhir sebagai konklusi dari rule. Dari titik pandang ini inferensia memperlakukan sebuah rule sebagai rule (besar) dimana clause part-nya mengandung semua clause part dalam setiap node dari rangkaian, dan conclusion part-nya adalah konklusi dari node terakhir. Sehingga dari sini, operator clause adalah operator AND (dari semua clauses), yang juga merupakan jenis operator konklusi jika ada lebih dari satu nilai konklusi dalam suatu node (dan lalu ia menjadi node terakhir dalam suatu rule).

VCIRS membutuhkan konsistensi pengguna dalam penggunaan nama variabel baik pada bagian clause maupun konklusi, dan mengeloloa konsistensi lojikal rule di sepanjang KB. Ini tidaklah seketat yang dibayangkan, sebab sistem menyajikan kepada pengguna daftar nilai yang dapat diambil dari nilai variabel dalam KB. Pengguna tentu saja bebas untuk menentukan nilai mana yang ia inginkan/tidak inginkan untuk digunakan. Proses inferensia baik dengan pendekatan RDR dan RBS menggunakan KB sebagaimana adanya.

Dua mekanisme inferensia, RBS dan RDR disajikan di bawah ini.

18.10.1 Mekanisme Inferensia RDR

Saat pengguna menyajikan case ke sistem untuk mendapatkan hasil, proses inferensia dimulai dengan pencarian node yang layak. Lalu, jika node yang layak ditemukan, sistem akan terus menelusuri tree/pohon rule dan meminta konfirmasi pengguna untuk setiap nilai seperti pendekatan forward chaining sampai suatu konklusi dieksekusi jikalau nilainya cocok atau tak ada hasil yang didapaatkan jikalau nilainya tidak cocok. Algoritma pencarian node yang layak ini sama dengan yang digambarkan pada gambar 18.17. Gambar 18.27 menjelaskan mekanisme inferensia RDR.

1. Proses case yang dimasukkan pengguna untuk menemukan node yang layak dengan algoritma pencarian node yang layak. Hasilnya disimpan pada candidate nodes. Jika hasilnya tak ada pada langkah ini, proses berhenti disini. Langkah 2 akan berlanjut jika terdapat paling tidak satu node dalam candidate nodes.
2. Dimulai dari node layak pertama dalam candidate nodes, dapatkan <Rule ID> dan <Node Order> yang memiliki node <Node ID> ini.
3. Jika node layak tadi adalah perfect node, eksekusi node layak ini, simpan konklusinya dalam ConclusionQueue dan lanjutkan ke langkah 4. Jika ada nilai yang tak terkonfirmasi, minta konfirmasi pada pengguna. Jika nilai yang dimasukkan pengguna tidak cocok, mulailah lagi dari langkah 2 dengan node layak berikutnya. Simpan setiap kejadian dalam EventLog.
4. Cek node berikutnya berdasarkan <Node Order> berikutnya dari <Rule ID> saat ini, dan minta pengguna untuk mengkonfirmasi nilai-nilai dalam node itu. Jika semua nilai-nilai telah terkonfirmasi, eksekusi node itu, dan simpan konklusi dari node itu dalam ConclusionQueue. Ulangi langkah ini, sampai tak ada node sama sekali pada <Rule ID>, selama nilai dari node cocok dengan nilai yang dimasukkan pengguna. Jika ketidakcocokan nilai terjadi, ulangi lagi dari langkah 2 dengan node layak berikutnya. Jika node terakhir dari <Rule ID> sukses untuk dieksekusi, Sajikan konklusi terakhir ke pengguna sebagai konklusi final bersama dengan semua konklusi dari ConclusionQueue sebagai konklusi-konklusi intermediate/perantara. Tanyakan pada pengguna jika ia ingin melanjutkan untuk menemukan konklusi lebih jauh, jika ada node-node lain dalam candidate nodes. Jika pengguna menyetujui, ulangi lagi dari langkah 2 dengan node layak berikutnya untuk mencari konklusi lebih jauh. Jika pengguna tidak ingin lagi, proses inferensia dapat berhenti disini. Catat setiap kejadian dalam EventLog.
5. Jika tak ada node yang dieksekusi, katakan pada pengguna bahwa tak ada satupun node yang cocok dengan case-nya.

Gambar 18.27 Inferensia RDR dalam VCIRS

Dari ConclusionQueue kita dapat menjawab: What (apa) hasil dari inferensia. Jawaban dari: How (bagaimana) dan Why (mengapa) dapat diperoleh dari EventLog.

18.10.2 Mekanisme Inferensia RBS

Proses inferensia ini didasarkan pada Rule Structure dan Node Structure. Seperti yang sudah ditulis sebelumnya, sebuah node dalam Node Structure adalah serupa dengan rule di RBS, yang mengandung satu clause dan conclution part. Sebuah rule dalam Rule Structure juga serupa dengan rule di RBS, yaitu rule besar RBS. Sebuah RBS rule dibuat dari node yang didapat dengan mengambil nama node, nama variabel dan nilai-nilainya dari Node Structure, dan konklusi dari Conclusion Node yang bersesuaian. Rule RBS juga dapat dibuat dari Rule Structure. Ia didapat dengan cara serupa dengan pembuatan rule dari Node Structure, kecuali nama dan konklusi dari setiap rule diperoleh dari node terakhir dari rule yang diberikan. Rule-rule RBS dibuat dari baik Node Structure dan Rule Structure yang dikombinasikan bersama untuk melakukan inferensia RBS. Proses pembuatan rule RBS ini disebut dengan Knowledge Base Transformation (transformasi basis pengetahuan) dan akan didiskusikan pada bagian 18.11 di bawah.

Karena mekanisme inferensia yang digunakan adalah mekanisme pada RBS, maka pendekatan yang digunakan adalah forward dan backward chaining bersama dengan Confidence/Certainty Factor (CF) seperti yang sudah dibahas pada bab 5 dan bab 6.

Baik dengan forward maupuan backward chaining, VCIRS dapat menyajikan kepada pengguna hasil dan penjelasan dari nilai yang dimasukkannya:

- Konklusi didapatkan dari jawaban dari pertanyaan: What (apakah) hasil dari proses inferensia menurut nilai yang sedang dimasukkan itu.
- Semua nilai-nilai yang tercatat selama proses inferensia (contoh: rangkaian rule yang sedang dieksekusi atau gagal untuk dieksekusi, nilai yang diproses dalam clause dan conclusion part) adalah penjelasan dari pertanyaan: How (bagaimana) suatu hasil itu didapatkan.
- Perhitungan CF untuk nilai yang sedang diproses dan konfirmasi dari setiap clause menentukan rule mana yang akan dieksekusi atau gagal untuk dieksekusi adalah jawaban dari pertanyaan: Why (mengapa) suatu hasil bisa diperoleh seperti itu.

18.11 Knowledge Base Transformation

Pertama, Node Structure dalam VCIRS memiliki kemiripan dengan struktur dari Rule Base (RB) dalam RBS. Sehingga kita dapat mentransformasi/mengubah KB dari VCIRS kedalam RB dari RBS. Struktur dalam Rule Structure menggunakan Node Structure. Sehingga, ia juga dapat ditransformasi/diubah kedalam RB dari RBS, kecuali bahwa nama dan konklusi dari setiap rule didapat dari node terakhir dari rule tersebut. Gambar 18.28 menjelaskan algoritma transformasi dari Node Structure ke rule RBS, sedangkan

gambar 18.29 untuk transformasi Rule Structure ke rule RBS.

Tujuannya adalah rule saat ini yang sedang dibuat, satu demi satu.

1. Dari Node Structure, dimulai dari <Node ID> pertma, dapatkan setiap <Node ID> dan transformasikan ke <Rule ID>. Lalu, dapatkan <Node CF Value> dari record saat ini dan transformasikan ke <Rule CF>.
2. Letakkan kata kunci/keyword 'IF' setelah <Rule CF>.
3. Dapatkan <Variables> dari record saat ini dan transformasikan mereka ke <Clauses> RBS (lihat bagian 18.5), sebuah clause untuk setiap <Variable ID> diikuti dengan <Variable Order>. Hilangkan <Variable Order> pada saat transformasi dan letakkan keyword 'AND' sebelum <Variable ID> berikutnya, jika <Variables> memiliki lebih dari satu <Variable ID>.
4. Letkkan keyword 'THEN' setelah <Clauses>.
5. Berdasarkan <Parent Node ID> saat ini dan <Node ID> yang sedan diproses, cari <Parent Node ID> dan <Node ID> dari Conclusion table. Dimulai dari conclusion value pertma, transformasikan <Conclusion> ke <Conclusions> RBS, dan letakkan keyword 'AND' sebelum <Conclusion> berikutnya, jika <Conclusion> memilki lebih dari satu <Conclusion>.
6. Proses akan diulang untuk setiap node dalam Node Structure sampai node terakhir.

Gambar 18.28 Transformasi Node Structure ke rule base

Nama dari rule didapatkan dari <Node ID> terakhir dalam rule, dan konklusi dari rule didapatkan dari <Conclusion> yang dimiliki oleh <Node ID> terakhir dalam rule.

1. Dari Rule Structure yang memiliki lebih dari node, dapatkan <Node ID> terakhir, yaitu <Node ID> yang memiliki <Node Order> tertinggi, dan jadikan ia sebagai <Rule ID> dari rule RBS jika <Rule ID> tersebut belum ada dalam rule base RBS. Jika ia sudah ada letakkan karakter tambahan "R" setelah <Rule ID> ini untuk membuatnya berbeda dari yang sudah ada. "R" menandakan bahwa rule yang dibangun ini dibentuk dari rangkaian node, bukan dari satu node. Lalu, informasi diperoleh dari Node Structure. Dimulasi dari <Node ID> pertama dan didasarkan pada <Node Order> untuk mendapatkan <Node ID> berikutnya untuk memperoleh <Node CF Value> dari record saat ini dan transformasikan ia ke <Rule CF>.
2. Letakkan kata kunci/keyword 'IF' setelah <Rule CF>.
3. Dapatkan <Variables> dari record saat ini dan transformasikan mereka ke <Clauses> RBS, sebuah clause untuk setiap <Variable ID> berdasarkan <Variable Order>. Hilangkan <Variable Order> saat transformasi dan letakkan keyword 'AND' sebelum <Variable ID> berikutnya, jika <Variables> memiliki lebih dari satu <Variable ID>. Cek <Variable ID> dalam <Rule ID> saat ini, jika ia sudah ada ganti dengan <Variable ID> yang lebih baru bersama dengan <Variables>-nya.
4. Cek <Node ID> berikutnya. Jika <Node ID> berikutnya adalah node terakhir dalam rule saat ini, letakkan keyword 'THEN', dan ke langkah 5. Jika <Node ID> berikutnya bukanlah node terakhir, mulailah dari langkah 1 lagi dengan <Node ID> berikutnya dalam rule.
5. Dimulai dari conclusion value pertama dalam <Node ID> terakhir dari <Rule ID> rule dari Conclusion table transformasikan <Conclusion> ke <Conclusions> RBS, dan letakkan keyword 'AND' sebelum <Conclusion> berikutnya, jika <Conclusion> memiliki lebih dari <Conclusion>.
6. Proses akan diulangi lagi untuk setiap rule dalam Rule Structure.

Gambar 18.29 Transformasi Rule Structure ke rule base

18.12 Evaluasi Sistem

Pertama, seperti halnya RDR, VCIRS hanya memberi perhatian pada variabel (clause part) daripada bagian konklusi. Konklusi tidak mempunyai peranan dalam sistem; ia hanyalah hasil yang begitu saja diperoleh. Ini berarti kita dapat mengabaikan konklusi yang terang-terang tanpa resiko apa pun.

Ketidakpentingan konklusi selama proses penelusuran tree rule memiliki konsekuensi: tak ada mekanisme untuk melakukan forward dan backward chaining seperti dalam RBS. RDR tidak menyebutkan mengenai inferensia, sebab RDR ditujukan utamanya untuk akuisisi pengetahuan yang cepat dan sederhana, bukan untuk inferensia. Inferensia dalam RDR dilakukan pada waktu yang sama dengan waktu pengguna melakukan pembangunan pengetahuan dengan penyediaan case dan mengikuti sistem bekerja. Dia dapat memilih untuk hanya menelusuri tree rule selama operasi tanpa membuat rule baru, yang artinya dia hanya ingin melakukan inferensia (yaitu forward chainig sederhana). Fakta bahwa dia menelusuri tree (inferensia) dan mengupdate KB pada waktu yang sama mengandung arti bahwa verifikasi dari KB dilakukan pada saat berjalannya sistem. VCIRS mewarisi kedua keuntungan ini, yaitu akuisisi pengetahuan yang mudah dan sederhana dan verfikasi sambil jalan (verification-on-the-fly). Sebagai tambahan untuk pendekatan penelusuran tree seperti RDR untuk membuat pembangunan (update) pengetahuan mudah, VCIRS juga menyediakan mekanisme untuk melakukan transformasi KB sehingga pengguna dapat melakukan inferensia RBS yang berdayaguna. Yang dibutuhkan oleh VCIRS adalah pengguna harus konsisten dalam menggunakan variable ID baik pada clause maupun conclusion part, dan ia mampu untuk mengelola konsistensi rule-rule lojik dalam keseluruhan KB. Hal ini bukanlah pembatasan yang serius, sebab sistem akan menyediakan untuk pengguna suatu daftar nilai-nilai yang diambilkan dari nilai-nilai variabel yang telah dimasukkan dalam KB. Pengguna, tentu saja bebas untuk memilih atau tidak memilih nilai-nilai tersebut.

Dibandingkan dengan RDR, VCIRS memilik dayaguna yang sama dalam pembangunan pengetahuan. Ia juga menyimpan ruang untuk struktur rule yang tidak menyimpan setiap rangkaian node-nodenya. Ia cukup mengingat posisi dari setiap node dan selanjutnya adalah mudah untuk menyusun ulang rule sebagai rangkaian dari node-node. VCIRS menyimpan setiap kejadian dari case dalam Node Structure untuk setiap node, membantu sistem menghasilkan rule baru dari perbaikan pengetahuan, dimana RDR tak dapat melakukan hal ini.

RDR mengambil keuntungan dari kecepatannya dalam pembangunan pengetahuan dengan penyediaan Cornerstone Case (CC), contoh yang digunakan disaat pengguna membuat rule. RDR tidak perlu mengecek semua case, hanya case yang berhubungan

dengan case baru yang akan dibetulkan/dibenarkan. CC disimpan secara terpisah dari rule-rule RDR. Lebih lanjut, hal ini akan memperburuk kebutuhan ruang dari RDR dan membuat perulangan/repetitsi/redundansi data lebih buruk lagi terjadi. Kontras dengan hal itu, VCIRS hanya menyimpan node dalam Node Structure. Rule dalam Rule Structure secara praktis hanya menyimpan urutan dan parent node dari setiap node, sehingga kebutuhan ruang dapat dikurangi.

Dibandingkan dengan RBS, VCIRS dapat melakukan verification-on-the-fly pengetahuan, dimana RBS tidak dapat melakukannya. Dalam RBS, verifikasi harus dilakukan oleh pakar secara manual yang amat memakan waktu dan mempunyai kecenderungan untuk terjadinya inkonsistensi yang lebih buruk. Dalam RBS, rule yang inkonsistensi cenderung akan berdampak pada rule-rule yang lain. Hal ini bukan merupakan hal yang serius dalam VCIRS, karena rule yang inkonsisten hanya berakibat pada sharing nodes (rules) yang dimiliki oleh beberapa rule saja, bukan keseluruhan rule. Walaupun beberapa nodes (rules) memiliki nilai-nilai yang sama dengan yang lain, kecuali node-node dalam rangkaian yang sama, maka tak ada apa pun yang terjadi.

VCIRS selalu melakukan kalkulasi ulang untuk VUR pada variabel yang terkait, jika suatu variabel dimasukkan kedalam Variable-Centered Rule Structure. Ia juga mengkalkulasi ulang NUR dari semua node yang menggunakan variabel tersebut, dan mengupdate RUR dari semua rule yang menggunakan node ini. Ini akan nampak sebagai tugas yang menjemukan, namun ia berguna untuk memandu pengguna dalam proses pembangunan dan inferensia pengetahuan. Ini juga berguna untuk mendukung pembangkitan rule.

Disaat melakukan pembangkita rule, VCIRS meminta pengguna untuk mengkonfirmasikan rule/node yang sedang dibangkitkan, sebab pembetulan/pembenaran lojik (semantik) dari rule/node terbangkitkan masih memerlukan pertimbangan pengguna. Sistem hanya menghasilkan rangkaian alternatif dari variabel-variabel yang penting untuk membentuk node baru dan rangkaian dari node untuk membentuk rule masing-masing menurut urutan relatif dari variabel dan node. Ini secara sintaktik dijamin benar, namun pembetulan secara semantik masih membutuhkan pengguna untuk menjustifikasi.

Walaupun VCIRS mengatasi masalah inferensia dari RDR, algorithma Rete yang dikembangkan oleh Forgy [For82] masih memiliki kinerja yang lebih baik dibandingkan dengan langkah-langkah (cycles) yang harus dilakukan sebelum mendapatkan hasil. Algoritma Rete digunakan dalam kebanyakan RBS kinerja tinggi, sebab ia adalah metode yang sangat efisien untuk permasalahan pengenalan pola. Kelemahannya adalah ia memiliki kompleksitas ruang tinggi (high space complexity). VCIRS yang memiliki Variable-Centered Rule Structure menyimpan posisi (hanya posisi!) dari setiap variabel dalam suatu node dan rule menyediakan kinerja yang baik dan masuk akal, tidak terlalu

sempurna seperti halnya algoritma Rete, namun masalah kompleksitas ruang cenderung dapat lebih rendah diwujudkan.

18.13 Kesimpulan

VCIRS didesain berpusat pada node yang mendukung variabel dan struktur rule untuk mengimplementasikan KB pada suatu sistem rule. Ia mengadopsi arsitektur sistem dan proses inferensia pengetahuan dari RBS; juga mengadopsi kekuatan dari proses akuisisi pengetahuan dari RDR. RBS menderita dari proses akuisisi pengetahuan yaitu memaksa pengguna untuk memahami struktur dari KB. RDR sukses mengatasi masalah ini, dengan harga yang harus dibayar yaitu berkurangnya kemampuan inferensia yang berdayaguna. VCIRS datang diantara keduanya. Dengan mengadopsi fitur-fitur dari RBS, VCIRS mendapatkan keuntungan dari format yang familiar mengenai KB (yaitu rule base) serta proses dan hasil inferensia yang berdayaguna, yang mampu menjawab pertanyaan: What, How dan Why dari pengguna. Dan dengan mengadopsi fitur-fitur dari RDR, VCIRS membolehkan akuisisi pengetahuan secara ekstrim cepat dan sederhana tanpa bantuan dari knowledge engineer. Seperti RDR, proses pembangunan pengetahuan sederhana ini pada saat yang sama melakukan juga verification-on-the-fly. Ia menjamin KB untuk menjadi seperti apa yang diinginkan oleh pengguna.

VCIRS mampu untuk mentransformasikan Kbnya ke rule base standar, yang sudah dikenal baik oleh pengguna umum. Dengan hal ini, VCIRS dapat mendukung metode forward dan backward chainig selama proses inferensia. Disaat mentransformasikan Kbnya kedalam rule base, VCIRS menambahi sesuatu kedalam rule standar untuk setiap rangkaian rule sebagai tambahan dalam mentransformasikan setiap node kedalam rule standar.

VCIRS mampu untuk melakukan inferensia baik untuk pendekatan RDR maupun RBS. Dengan pendekatan RBS ia menggunakan baik forward maupun backward chaining, yang lebih baik untuk ekstraksi informasi KB dan pencarian relasi diantara rule-rule. Pendekatan RDR adalah seperti laiknya proses forward chaining sederhana dengan penelusuran tree rule menggunakan DFS.

VCIRS memiliki modul baru yang disebut dengan Refinement Module yang memiliki 3 tugas: analisis variabel, analisis nilai dan pembangkitan rule. Analisis variabel menentukan variabel/node apa yang terpenting (yaitu important degree), sedangkan analisis nilai menentukan seberapa sering suatu rule/node/variable itu digunakan (yaitu usage degree). Usage degree akan membantu pengguna sebagai garis pedoman selama pembangunan dan inferensia pengetahuan untuk menentukan variabel mana yang pengguna inginkan untuk dikunjungi. Bersama dengan important degree, usage degree

akan mendukung pembangkitan rule untuk menghasilkan rule/node baru. Analisis variabel, analisis nilai dan pembangkitan rule; secara bersamaan perbaikan KB secara evolusional.

Kontribusi dari bab ini dapat disimpulkan sebagai berikut: telah diajukan dan diimplementasikan Variable-Centered Intelligent Rule System (VCIRS), yang memakai node berpusat pada variabel (variable-centered) dan rule structure untuk mengorganisasikan rule base sehingga pembangunan pengetahuan yang mudah, inferensia pengetahuan yang berdayaguna dan perbaikan evolusional dari kinerja sistem dapat didapatkan pada saat yang sama.

18.14 Riset di Masa Depan

Terdapat pelbagai kemungkinan untuk meningkatkan sistem yang sudah diajukan. Beberapa peningkatan tersebut adalah:

- Menghapuskan permasalahan perulangan akuisisi pengetahuan, yang masih muncul dalam VCIRS, seperti yang terjadi pada RDR. Algoritma Reter nampaknya dapat mengatasi permasalahan ini.
- Menyertakan operator lojikal lainnya untuk mendayagunakan kemampuan representasi VCIRS. VCIRS hanya menggunakan operator AND untuk penyederhanaan.

DAFTAR PUSTAKA

[Adl93] D. Adler, "Genetic algorithms and simulated annealing: a marriage proposal," *in Proc. Int. Conf. Neural Network*, pp.1104-1109, 1993.

[Baur90] G.R. Baur and D.V.Pigford, *Expert Systems for Business: Concepts and Applications*, Boyd & Fraser Publishing Company, Boston-USA, 1990.

[Bro89] D. Brown, C. Huntley, and A. Spillane, "A parallel genetic heuristic for the quadratic assignment problem," *in Proc. 3rd Int. Conf. Genetic Algorithms*, pp.406-415, 1989.

[Chen97] S.M. Chen and M.S. Yeh, "Generating fuzzy rules from relational database systems for estimating null values," Cybern. Syst., vol. 28, no. 8, 1997, pp. 695-723.

[Chen00] S.M. Chen and H.H. Chen, "Estimating null values in the distributed relational databases environment," *Cybern. Syst.*, Vol. 31, No. 8, pp. 851-871, 2000.

[Chen03] S.M. Chen and C.M. Huang, "Generating weighted fuzzy rules from relational database systems for estimating null values using genetic algorithms," *IEEE Transactions on Fuzzy Systems*, Vol. 11, No. 4, August 2003, pp. 495-506.

[Comp89] Compton, P. J. and Jansen, R., "A philosophical basis for knowledge acquisition," *Knowledge Acquisition 2*, 1990, pp. 241-257. (*Proceedings of the 3rd European Knowledge Acquisition for Knowledge-Based Systems Workshop*, Paris, 1989, pp. 75-89)

[Comp91] P. Compton, G. Edwards, B. Kang, L. Lazarus, R. Malor, T. Menzies, P. Preston, A. Srinivasan and S. Sammut, "Ripple down rules: possibilities and limitations," in Boose, J.H. and Gaines, B.R., (Eds.), *Proceedings of the Sixth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Calgary, Canada, University of Calgary, 1991, pp. 6-1-6-20.

[Comp00] P. Compton and D. Richards, "Generalising ripple-down rules," *Knowledge Engineering and Knowledge Management (12th International Conference EKAW)*, France, 2000, pp. 380-386.

[For82] C.L. Forgy, "Rete: a fast algorithm for the many pattern/many object pattern match problem", *Artificial Intelligence 19* (1982), 17-37.

[Gen97] Mitsuo Gen and Runwei Chen, *Genetic Algorithms and Engineering Design*, John Wiley & Sons, Inc., New York, USA, 1997.

[Gon93] A.J. Gonzalez and D.D. Dankel, *The Engineering of Knowledge-Based Systems: Theory and Practice/Book*, 1993.

[Ho03] V. Ho., W. Wobcke and P. Compton, "EMMA: an e-mail management assistant," in Liu, J., Faltings, B., Zhong, N., Lu, R., Nishida, T. (Eds.), *in Proc. of IEEE/WIC International Conference on Intelligent Agent Technology*, Los Alamitos, CA, 2003, pp. 67-74.

[Huang02] Chung-Ming, Huang, "New Methods to Estimate Null Values in Relational Database Systems Based on Genetic Algorithm," *Master Thesis*, Department of Computer Science and Information Engineering, College of Electrical and Computer Engineering, National Taiwan University of Science and Technology, 2002.

[Ign91] J.P. Ignizio, *Introduction to Expert Systems: The Development and Implementation of Rule-based Expert Systems*, McGraw-Hill International Editions, 1991.

[Kang95] B. Kang, P. Compton and P. Preston, "Multiple classification ripple down rules: evaluation and possibilities," in Gaines, B. and Musen, M., *Proceedings of the 9th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, University of Calgary, 1995, pp. 17.1-17.20.

[Kang96] B. Kang, "Multiple Classification Ripple Down Rules," *Ph.D. Thesis*, University of New South Wales, Australia, 1996.

[Koa90] S. Koakutsu, Y. Sugai, H. Hirata, "Block placement by improved simulated annealing based on genetic algorithm," *Trans. of the institute of Electronics, Information and Communication Engineers of Japan*, vol.J73-A, No.1, pp.87-94, 1990.

[Koa92] S. Koakutsu, Y. Sugai, H. Hirata, "Floorplanning by improved simulated annealing based on genetic algorithm," *Trans. of the institute of Electrical Engineers of Japan*, vol.112-C, No.7, pp.411-416, 1992.

[Koa96] S. Koakutsu, M. Kang, and W. W.-M. Dai, "Genetic simulated annealing and application to non-slicing floorplan design," *in Proc. 5th ACM/SIGDA Physical Design Workshop*, (Virginia, USA), pp. 134--141, April 1996.

[Leung98] K.S. Leung and W. Lam, "Fuzzy concepts in expert systems," *IEEE Computer*,

September 1998.

[Lin93] T.-T. Lin, C.-Y. Kao, and C.-C. Hsu, "Applying the genetic approach to simulated annealing in solving some NP-Hard problems," *IEEE Trans. System, Man, and Cybernetics*, vol.23, no.6, pp.1752-1767, 1993.

[Pres93a] P. Preston, G. Edwards and P. Compton, "A 1600 Rule Expert System Without Knowledge Engineers", In J. Leibowitz, editor, *Second World Congress on Expert Systems*, 1993.

[Pres93b] P. Preston, G. Edwards and P. Compton, "*A 2000 Rule Expert System Without Knowledge Engineers*", 1993.

[Rus03] Stuart J. Russell and Peter Norvig, *Artificial Intelligence – A Modern Approach*, Second Edition, Prentice Hall – Pearson Education, Inc., New Jersey, USA, 2003.

[Sir87] D. Sirag and P. Weisser, "Toward a unified thermodynamic genetic operator," *in Proc. 2nd Int. Conf. Genetic Algorithms*, pp.116-122, 1987.

[Sub03] Irfan Subakti and Alexander L. Romy, "Universal inference engine," *Proc. of the 4th National Seminar of Computer Science and Information Technology 2003 (SNIKTI2003)*, Vol. 4, No. 1, 1st August 2003, Institute Technology of Sepuluh Nopember (ITS), Surabaya, Indonesia, 2003, pp. 94-100.

[Sub05a] Irfan Subakti, "Genetic simulated annealing for null values estimating in generating weighted fuzzy rules from relational database systems," *Proc. of the 1st Annual International Conference: Information and Communication Technology Seminar 2005 (ICTS2005)*, Vol. 1, No. 1, August 2005, Institute Technology of Sepuluh Nopember (ITS), Surabaya, Indonesia, 2005, pp. 181-188.

[Sub05b] Irfan Subakti, "Multiple null values estimating in generating weighted fuzzy rules using genetic simulated annealing," *Proc. of the 1st Annual International Conference: Information and Communication Technology Seminar 2005 (ICTS2005)*, Vol. 1, No. 1, August 2005, Institute Technology of Sepuluh Nopember (ITS), Surabaya, Indonesia, 2005, pp. 175-180.

[Sub05c] Irfan Subakti, "A variable-centered intelligent rule system," *Master Thesis*, Department of Computer Science and Information Engineering, College of Electrical and Computer Engineering, National Taiwan University of Science and Technology, 2005.

[Sub05d] Irfan Subakti, "A variable-centered intelligent rule system," *Proc. of the 1st*

Annual International Conference: Information and Communication Technology Seminar 2005 (ICTS2005), Vol. 1, No. 1, August 2005, Institute Technology of Sepuluh Nopember (ITS), Surabaya, Indonesia, 2005, pp. 167-174.

[Tur95] E. Turban, *Decision Support and Expert Systems: Management Support Systems*, Fourth Edition, Prentice-Hall, Inc., United States of America, 1995.

[Nus] http://www.comp.nus.edu.sg/~pris/FuzzyLogic/

[Wins92] P.H. Winston, *Artificial Intelligence*, Addison-Wesley, Third Edition, 1992.