BAB I
PENDAHULUAN

1.1. Pengertian dan Sejarah Dari Artificial Intelligence(AI)

AI mempelajari bagaimana membuat komputer melakukan sesuatu pada suatu kejadian/peristiwa yang mana orang melakukannya dengan baik. [1).]

a. Pengertian AI

Definisi AI : *merupakan proses di mana peralatan mekanik dapat melaksanakan kejadian-kejadian dengan menggunakan pemikiran atau kecerdasan seperti manusia.*

Pengertian AI dapat ditinjau dari dua pendekatan : [2)]

1). Pendekatan Ilmiah(*A Scientific Approach*)

Pendekatan dasar ilmiah timbul sebelum invansi ke komputer, ini tidak sama dengan kasus mesin uap. Pendekatan ilmiah melihat batas sementara dari komputer, dan dapat diatasi dengan perkembangan teknologi lanjutan. Mereka tidak mengakibatkan tingkatan pada konsep.

2). Pendekatan Teknik(*An Engineering Approach*)

Usaha untuk menghindari definisi AI, tetapi ingin mengatasi atau memecahkan persoalan-persoalan dunia nyata(*real world problem*).

Dalam kuliah ini, kita menggunakan kedua pendekatan ini.

Mengapa kita mempelajari AI ? karena

- *AI merepresentasikan bagian tengah atau inti dari ilmu komputer(Computer Science).*
- *AI mewujudkan suatu bentuk ketidak tepatan dari komputasi (karakteristik dalam matematika).*
- *AI mempunyai suatu kekuatan alami antar cabang ilmu, AI adalah bagian ilmu teknik dari Cognitive Science, Cognitive Science adalah suatu perpaduan ilmu filsafat, ilmu liguistik dan ilmu fisikologi.*
- AI memperlakukan representasi pengetahuan dan manipulasinya.

---

1). Rich, Elaine, and Knight, Kevin, “Artificial Intelligence”, Second Edition, page 3, McGraw-Hill Inc., 1991
2). Charniack, Eugene and McDermott, Drew, “Introduction To Artificial Intelligence”, page 1, McGraw-Hill Inc., 1985.

- *Pengetahuan (knowledge) adalah pusat dari semua ilmu teknik dan AI adalah pusat dari semua ilmu teknik.*
- *Alasan penting lainnya adalah penelitian AI diharapkan me-nemukan atau membongkar bentuk krisis besar dalam waktunya. Krisis dibuat oleh interaksi dari teknologi, ilmiah(*science*) dan filsafat.*

Program *Intelligent:* program yang mampu menyimpan kenyataan (*facts*) dan proposisi dan hubungannya yang beralasan.

b. Sejarah dari AI

Awal pekerjaan dipusatkan pada seperti *game playing* (misalnya: audio dengan kecerdasan dan permainan catur(*chess player*), pembuktian teorema (theorem proving) pada Tugas-tugas formal (*Formal Tasks*).

Samual(1963) menulis sebuah program yang diberi nama check-er-playing program, yang tidak hanya untuk bermain *game*, tetapi digunakan juga pengalamannya pada permainan untuk mendukung kemampuan sebelumnya.

Catur juga diterima, karena banyak sekali perhatian terhadap permaianan catur yang merupakan permainan yang lengkap atau kompleks, program catur di sini situasinya harus jelas dan rule atau ketentuannya harus seperti dunia nyata. Kandidat AI harus mampu menangani masalah-masalah yang sulit.

Logic theorist diawal percobaan untuk membuktikan teorema matematika. Ia mampu membuktikan beberapa teorema dari bab 1 Prinsip Matematika Whiteheat dan Russell.

Theorema Gelernter (1963) membuktikan pencarian area yang lain dari matematika yaitu geometri.

Pada tahun 1963, pemecahan masalah umum menggunakan *object,* pembuktian dengan atraksi(eksternal).

Dari awal pekerjaan AI ini memindahkan lebih khusus tugas yang sering berguna antara lain: **3)**

a. Tugas biasa/keduniaan(*Mundane Tasks*)

- Persepsi : - *vision*
  - *speech*

3) opcit [1)], page 5.

- *Natural Language :* *- understanding*
  - generation
  - translation
- *Commonsence Reasoning* (pertimbangan berdasarkan pikiran sehat): *- robot control*

b. Tugas Formil(*Formal Tasks*)

- Games : - chess
  - checkers
- *Matematics:* *- geometri*
  - logic
  - proving properties of programs

c. Tugas Ahli(*Expert Tasks*)

- Teknik : *- Design*
  - Fault Diagnosis
  - Planning
- Scientific Analysis
- Medical : *Diagnosis & Theraphy.*

1.2. Aplikasi-aplikasi Dari AI

Kecerdasan tiruan (AI) telah dipelajari selama kira-kira 40 tahun. Hingga saat ini telah dihasilkan beberapa produk aplikasi AI secara komersial. Produk-produk tersebut umumnya dapat dijalankan pada perangkat keras komputer mulai dari komputer pribadi(PC) se-harga USA$5000 sampai dengan komputer besar(*mainframe*) seharga USA$50,000. Secara khas masukkan untuk produk-produk tersebut berbentuk data simbolis. Aplikasi-aplikasi AI antara lain:

- *Game Playing*
- *Sistem Bahasa Alami*
- *Sistem Perancangan dan Pembuatan CAD/CAM*
- *Sistem Pakar VLSI*
- *Sistem Pakar Reparasi Perangkat Keras*

- *Manajemen Data Cerdas*
- *Sistem Otomatisasi Kantor*
- *Analisa Kecerdasan Militer*
- *Kendali dan Pemanggilan informasi disk video*
- *Kendali Robot*
- *Analisis Program Komputer*
- *Diagnosis Penyakit*
- *Konfigurasi komputer*
- *Ramalan senyawa kimia*
- *Sintesis ucapan*
- *Sistem Pakar Operator Komputer*
- *Manajemen Kendali Senjata*

Apa AI hari ini adalah sesuatu yang baru sekarang. Bagian-bagian dari AI antara lain :

- *Parallel Distributed Processing (Neural Network)*
- *Machine Vision*
- *Automatic Programming Tools*
- *Memory Management*
- *Pattern Recognition*
- *Natural Language Processing*
- *Development Of Knowledge Base*

Kecerdasan tiruan(*Artificial Intelligence*) adalah sub bagian dari ilmu komputer yang merupakan suatu teknik perangkat lunak yang pemrogramannya dengan cara menyatakan data, pemrosesan data dan penyelesaian masalah secara simbolik, dari pada secara numerik.

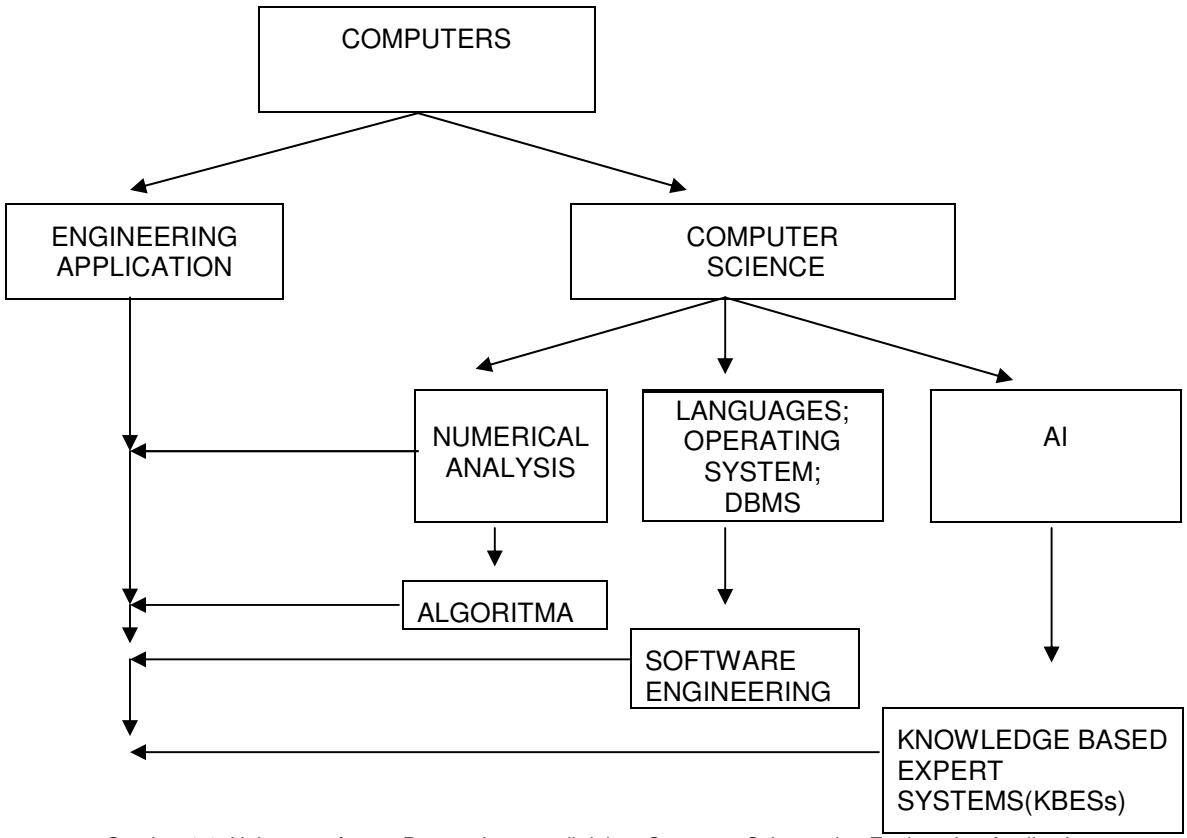
Masalah-masalah dalam bentuk simbolik ini adalah masalah-masalah yang sering kita jumpai dalam kehidupan sehari-hari. Masalah-masalah ini lebih berhubungan dengan simbol dan konsep simbol dari pada dengan angka-angka.

Di sini dengan kecerdasan tiruan diusahakan untuk membuat komputer seakan dapat berpikir secara cerdas.

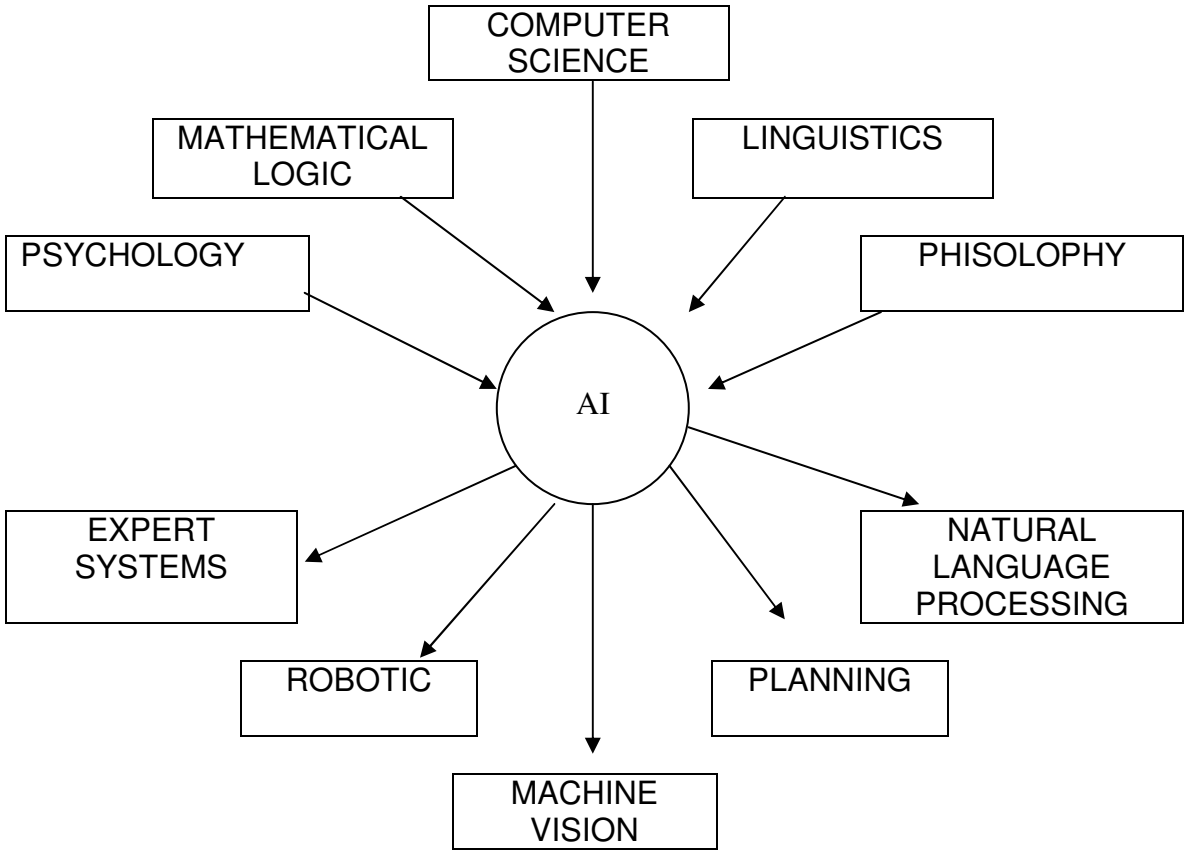
Untuk mudah dapat dimengerti dengan apa yang dimaksud dengan kecerdasan tiruan atau proses simbolik adalah dengan cara membandingkannya dengan program konvensional yang melakukan operasi numerik.

Program konvensional dapat menjawab "X + Y" jika harga X dan Y diberikan, tetapi program ini tidak dapat menjawab bahwa "X + X = 2X", atau tidak dapat menjawab mengapa mobil tidak dapat distart.

Program kecerdasan tiruan berbeda dari program konvensional tidak saja dalam hal mengerti akan simbol atau informasi, tetapi juga program simbolik, karakter digit, kata, data dan lainnya saling berhubungan. Informasi dan bentuk hubungannya yang bervariasi digunakan untuk merepresentasikan hubungan antar-informasi. Hubungan antara simbol dan informasi secara tidak langsung menyatakan apa yang diinterpretasikan oleh manusia sebagai arti atau pengetahuan.



Gambar 1.1. Hubungan Antara Pengembangan di dalam Computer Science dan Engineering Applications.
Sumber: Clive L. Dym & Raymond E. Levitt, "Knowledge Based Systems in Engineering", page 9, Mc Graw-Hill, 1991



Gambar 1.2. Suatu Input-Output Model Untuk Artificial Intelligence
. (Source : Kamran Parsaye & Mark Chignell, "Expert Systems For Experts", page 11, John Wiley & Sons Inc, 1988)

Apa saja yang dikerjakan dalam bagian-bagian AI berikut ini:

- *Machine Vision :* Bertujuan pada pengenalan pola dalam beberapa jalan yang sama sebagai kegiatan sistem visual/indera manusia.
- *Robotics :* Difokuskan pada produksi alat-alat mekanik yang dapat mengendalikan gerak. Sebagai contoh: sebuah robot sederhana mampu atau dapat bergerak/pindah ke depan, belakang, kanan atau kiri atau pindah tempat ke ruangan berbeda. Sebuah robot sebenarnya buta akan bentuk urutan dari aksi bila tanpa usaha untuk mengganti komponennya atau bisa mendeteksi dan memperbaiki kesalahan dalam rencananya akan menjadi sulit bila tanpa kecerdasan. Sering sebuah robot akan diformulasikan pada sebuah rencana dasar pada informasi yang tidak lengkap dan benar dalam menjalankan sebuah rencana

- *Speech Processing :* Bertujuan pada pengenalan dan sintesa pembicaraan manusia.
- *Theorem Proving :* Usaha untuk membuktikan secara otomatis masalah-masalah dalam matematika dan logika.
- *General Problem Solving :* Bertujuan pada pemecahan kelas-kelas dari masalah-masalah yang ditekankan dalam sebuah bahasa formal.
- *Pattern Recognation:* Difokuskan pada pengenalan dan klasifikasi dari pola-pola.
- *Game Playing:* Pembuatan program-program bermain permainan.
- *Machine Learning :* Bertujuan pada produksi mesin-mesin yang mengakumulasi pengetahuan dengan contoh-contoh observasi.
- *Learning* merupakan sebuah persoalan sulit untuk program AI, dalam mencapai kesuksesan diperlukan dalam pemecahan persoalan. Bagian kemampuan untuk mempelajari komponen terpenting dari tindak tanduk/jalan kecerdasan.Sebuah sistem pakar harus berkemampuan ekstensif dan dapat menghitung kerugian dalam memecahkan sebuah persoalan. Tidak seperti manusia, bilamana jika ia diberikan persoalan yang serupa pada waktu berikutnya, dia tidak akan ingat solusinya. Dia membentuk urutan yang sama untuk menghitung lagi. Learning merupakan sebuah area yang sulit diteliti, beberapa program te-lah ditulis dengan tujuan bahwa ini bukan merupakan hasil(goal) yang diinginkan.
- *Planning* adalah aspek terpenting pendukung untuk mendesain atau merancang robot-robot dengan kemampuan menyele-saikan tugas mereka dengan tingkat fleksibelitas dan tanggap terhadap dunia luar.Planning merupakan masalah sulit dari sejumlah alasan yang tidak lebih dari ukuran tempat kosong(space) yang mungkin diu-rutkan dan dipindahkan.
- *Neural Network atau Parallel Distributed* : teknik-teknik terbaik untuk merepresentasikan pengetahuan dan merancang algoritma pencarian yang hati-hati untuk implementasi kecerdasan.

1.3. Kerja Kecerdasan Manusia

Karena kecerdasan tiruan adalah ilmu yang berdasarkan proses manusia berpikir, maka penelitian bagaimana proses manusia berpikir adalah hal yang pokok.

Pada saat ini para peneliti hanya mulai mengerti sedikit dari proses berpikir tersebut, tetapi sudah cukup diketahui untuk membuat asumsi-asumsi yang pasti tentang bagaimana cara berpikir dan menggunakan asumsi-asumsi tersebut untuk mendesain suatu pro-gram komputer yang mempunyai kecerdasan secara tiruan.

Semua proses berpikir menolong manusia untuk menyelesaikan sesuatu masalah. Pada saat otak manusia mendapat informasi dari luar, maka suatu proses berpikir memberikan petunjuk tindakan atau respon apa yang dilakukan. Hal ini merupakan suatu reaksi otomatis dan respon yang spesifik dicari untuk menyelesaikan masalah tertentu. Hasil akbar dari semua proses berpikir tersebut disebut tujuan (goal).

Pada saat tujuan telah dicapai, pikiran akan segera berha-dapan dengan tujuan-tujuan lainnya yang akan dicapai. Di mana se-mua tujuan-tujuan ini bila terselesaikan akan mengantar ke suatu tujuan utama. Dalam proses ini tidak ada satupun cara berpikir yang mengarah ke tujuan akhir dilakukan secara acak dan sembarangan.

Kecerdasan manusia dapat dipecah-pecah menjadi kumpulan fakta-fakta (*facts*) dan fakta-fakta ini yang digunakan untuk mencapai tujuan. Hal ini dilakukan dengan memformulasikan sekelompok aturan-aturan(*rules*) yang berhubungan dengan fakta-fakta yang disimpan dalam otak.

Contoh jenis fakta dan aturan yang berhubungan, yang digunakan sehari-hari, adalah:

Fakta 1 : *Air sangat mendidih*

Aturan 1 : **IF** *saya menaruh tangan ke air panas* **THEN** *sakit*

Di sini aturan ditulis dalam bentuk IF-THEN yang berdasarkan fakta, dimana IF adalah kondisi tertentu yang ada, dan THEN adalah respon atau aksi yang akan dihasilkan.

Dalam proses berpikir, proses ini berhubungan dengan fakta-fakta yang sangat banyak sebelum memberikan suatu tindakan atau respon. Selama proses ada suatu sistem yang mengarahkan pemilihan respon yang tepat. Proses ini disebut dengan pemotongan(*prunning*). Proses ini mengeliminasi lintasan dari berpikir yang tidak relevan dalam usaha mencapai tujuan. Jadi proses ini akan memotong setiap fakta-fakta atau aturan-aturan yang tidak akan mengarah ke tujuan.

Teknik pemrograman dengan kecerdasan tiruan melakukan prosesnya sama dengan apa yang dilakukan oleh otak manusia. Kecerdasan tiruan juga meniru proses belajar manusia di mana informasi yang baru diserap dan dimungkinkan untuk digunakan sebagai referensi pada waktu yang akan datang. Di sini informasi yang baru dapat disimpan tanpa harus mengubah cara kerja pikiran atau mengganggu seluruh fakta-fakta yang sudah ada. Sehingga dengan kecerdasan tiruan dimungkinkan untuk membuat program di mana se-tiap bagian dari program benar-benar independen. Di sini setiap bagian dari program seperti potongan-potongan informasi dalam otak manusia.

Secara umum kecerdasan tiruan dibagi menjadi tiga kategori dasar, yaitu:

1. Sistem Berbasis Pengetahuan atau sistem pakar(*Expert System/Knowledge Based System*), yaitu program komputer yang berisi pengetahuan manusia yang digu-nakan untuk menyelesaikan masalah dalam domain tertentu.
2. Sistem bahasa alami(*Natural Language System)*, yaitu pemrograman yang mengerti bahasa manusia.
3. Sistem dengan kemampuan memahami(*Perception System*), yaitu sistem untuk penglihatan, pembicaraan atau sentuhan.

Dari ketiga jenis kecerdasan tiruan itu, sistem pakar adalah yang paling banyak aplikasinya dalam membantu menyelesaikan masalah-masalah dalam dunia nyata.

Contoh aplikasi dari program ini antara lain yaitu : [4)]

- Delta *dari General Electric untuk konsultasi kerusakan lokomotif.*
- Prospector :
    - *Merupakan sistem pakar klasik yang lainnya*

- *Dikembangkan oleh Stanford Research Institute*
- *Digunakan untuk penaksiran prospek mineral, sehingga bisa membedakan kemungkinan informasi lokasi dan tipe dari dasar lubang endapan geologi di suatu tempat.*

➢ Xycon

- *Dikembangkan oleh Digital Equipment Corp's .*
- *Digunakan untuk mengkonfigurasi bagian-bagian komputer VAX.*
- *Telah digunakan sejak 1981.*

➢ Dendral:

- *Dikembangkan di Stanford pada akhir tahun 1960.*
- *Dirancang untuk menduga informasi struktur dari formula-formula molukel-molekul organik dan banyak informasi spekto-grafik tentang kimia yang ditampilkan dalam molekul. Sebab molekul-molekul organik cendrung menjadi besar, jumlah ke-mungkinan struktur untuk molekul-molekul ini cendrung menjadi sangat besar.*

➢ Mycin

- *Dikembangkan di Stanford pada pertengahan 1970.*
- *Salah satu program pertama yang dialamatkan pada masalah pe-mikiran dengan ketidak pastian dan tidak lengkapnya informasi.*

➢ Internist

- *Program untuk mendiagnosa penyakit dalam.*

➢ Dipmeter Advisor

- *Digunakan untuk menafsirkan hasil pengeboran minyak.*
- *Dibuat oleh Smith & Backer, tahun 1983.*

---

4). F. Luger, George, and A. Stubblefield William, "Artificial Intelligence : Structure and Strategies For Complex Problem Solving", Second Edition, page 16, Benyamin/Cummings Publishing Company, Inc., 1993.

Program kecerdasan tiruan ini dapat dilakukan dengan menggunakan suatu program paket, yaitu alat pengembangan sistem aplikasi pengetahuan *(knowledge system application development tool)* seperti:

➢ VP-Expert

➢ PC PLUS

- GURU
- JESS(*Java Expert System Shell*) Version 5.0
  http://herzberg.ca.sandia.gov/jess/FAO.html
- EXSYS, *dan lain-lain.*

Atau dengan menggunakan bahasa untuk pemrograman kecerdasan tiruan seperti *:*

- PROLOG (*Programming Logic*)
- WIN-PROLOG 4.040 (*Windows-Programming Logic)*
  http://www.lpa.co.uk/web386/8f922bf1.zip
- LISP(*Lisp Processing*)
- CLIPS(*C Language Integrated Production System*)
  http://www.ghgcorp.com/clips/download/source/

Lapangan dari *Artificial Intelligence* adalah gabungan beberapa area study, yaitu:[5)]

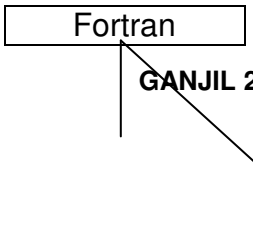
- *logic*
- *Searching*
- *Vision, Recognition dan Pattern Matching*
- *Natural Language Processing*
- *Expert System*
- *Robotik*
- *Learning*
- *Uncertainty dan Fuzzy Logic*

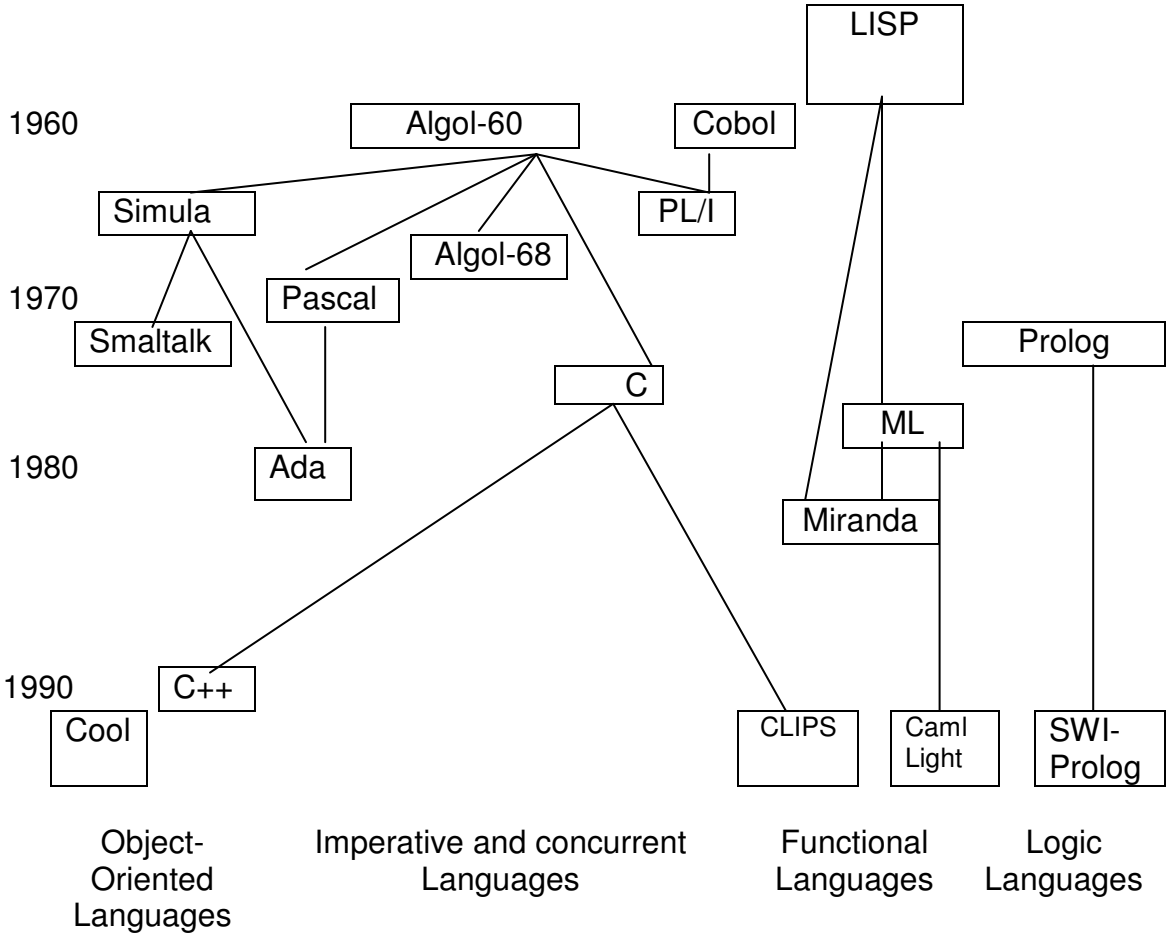
---

5) Schildt, Herbert,"Artificial Intelligence Using C", page 11, McGraw-Hill, 1987

Perangkat-perangkat lunak ini dapat dijalankan dengan komputer pribadi(PC), sehingga pengembangan untuk aplikasi kecerdasan tiruan dapat dilakukan dengan mudah dan dengan biaya yang murah.

1950





Gambar 1.3. Tahun dan Awal mulanya dari Bahasa Pemrogramman Utama
(Sumber : David A. Watt, "Programming Language Concepts & Paradigms", page 4, Prentice-Hall, 1990.)

Penjelasan mengenai Lapangan dari *Artificial Intelligence,* sebagai berikut*:*

- *LOGIC*

  Program dapat digunakan untuk mempelajari perbaikan logika dari sebuah argumen dengan menerapkan aturan logika standar.

- *SEARCHING*

  Diterapkan pada AI mengacu pada pencarian untuk penyelesaian sebuah masalah.

- *VISION, RECOGNITION DAN PATTERN MATCHING*

  Penting untuk beberapa aplikasi, termasuk robotik dan pengolah citra(*image processing*). Pada hal ini dibutuhkan untuk memperbolehkan komputer berhubungan secara langsung ke dunia dan manusia. Jika komputer

berhubungan secara menyeluruh dengan dunia manusia, maka dibutuhkan beberapa kemampuan bayangan (*vision*).

- *NATURAL LANGUAGE PROCESSING(NLP)*

  Bagian yang paling sulit dari sasaran AI untuk mendapatkannya karena NLP memperbolehkan komputer untuk mengerti bahasa manusia secara langsung.

  Permasalahan:

  1. Ukuran kekompleksan bahasa manusia.
  2. Mencoba untuk membuat komputer mengerti informasi secara konteks.

  Produk komersial pertama dari AI yang memiliki 2 buah atribute :

  a. Diperbolehkan memasukkan informasi tentang subyek ke dalam komputer (*knowledge Base*/dasar pengetahuan)
  b. Menyelidiki *knowledge base* dan berlaku sebagai *expert* atau pakar pada subyek

- *ROBOTIK*

  Digunakan untuk mempelajari mengontrol gerakan.

- *LEARNING*

  Bertransaksi dengan pembuatan program yang belajar dari kesalahan dari observasi atau permintaan komputer mempunyai kemampuan untuk mengambil keuntungan dari pengalaman.

- *UNCERTAINTY(Ketidak Pastian) DAN FUZZY* LOGIC

  Komputer dapat berpikir dengan menggunakan pengetahuan yang tidak lengkap dengan menerapkan penggunaan *Fuzzy Logic.*

1.4. Teknik-teknik AI

Suatu teknik kecerdasan tiruan(AI) adalah sebuah metode yang memanfaatkan pengetahuan yang akan direpresentasikan sedemikian rupa, sehingga: [6)]

- Penyamarataan penangkapan pengetahuan.

  Dengan kata lain, dia tidak diperlukan untuk menampilkan secara terpisah menurut situasi individu. Malahan sifat berbagi situasi yang terpenting adalah kerjasama dalam kelom-pok atau grup. Jika pengetahuan tidak dipunyai sifat ini, jumlah memory banyak sekali dan memperbaruinya akan diperlukan. Jadi kita biasanya memanggil sesuatu tanpa sifat dari data daripada pengetahuan.

- Dapat dimengerti oleh orang yang semestinya menyediakannya.

  Meskipun untuk beberapa program, bagian terbesar dari data dapat diperoleh secara otomatis, dalam beberapa daerah AI, pada akhirnya kebanyakan program AI semestinya disediakan oleh orang yang mengerti.

- Dapat dengan mudah memodifikasi yang salah menjadi benar dan mencerminkan perubahan di dalam dunia dan di luar permukaan dunia.
- Dapat digunakan dalam beberapa situasi kejadian yang besar, jika ini tidak sepenuhnya akurat dan lengkap.
- Dapat digunakan untuk membantu menanggulangi permasalahan yang sangat penting dengan sendirinya, dengan bantuan seksama dari kemungkinan jarak yang semestinya dipertimbangkan.

Meskipun teknik-teknik AI seharusnya dirancang dengan menjaga ketidak leluasaan ditentukan oleh masalah-masalah AI, ada beberapa tingkatan diantara masalah-masalah dan teknik pemecahan masalah/persoalan. Memungkinkan saja dalam memecahkan persoalan AI tanpa menggunakan teknik-teknik AI (meskipun dianjurkan di atas, solusinya tidak baik). Dan mungkin saja digunakan teknik-teknik AI untuk memecahkan bukan persoalan AI. Ini seperti sesuatu yang baik dilakukan untuk persoalan-persoalan yang mempunyai beberapa persamaan karakteristik sebagai persoalan-persoalan AI. Urutan

---

6) **opcit** [1)], **page 8.**

dalam mencoba mencirikan teknik-teknik AI dalam masalah yang berdiri sendiri sebagai kemungkinan jalan .

Tiga Teknik AI yang penting : [7)]

1). Pencarian/penelusuran(*Search*)

Menyediakan sebuah jalan untuk memecahkan persoalan yang lebih dari beberapa pendekatan tidak langsung yang tersedia, sebaik sebuah kerangka ke dalam beberapa teknik langsung yang dapat disimpan.

2). Penggunaan dari Pengetahuan(*Use of Knowledge*)
Memberikan sebuah jalan untuk memecahkan struktur-struktur dari objek yang dilibatkan.

3). Abstraksi(*Abstraction)*
Memberikan sebuah jalan yang mengutamakan pemisahan dan variasi dari beberapa yang tidak penting, kalau tidak meliputi beberapa proses.

Untuk solusi dari masalah yang sulit, program memanfaatkan teknik-teknik ini yang mempunyai beberapa keuntungan di atas atau bisa juga tidak. Mereka lebih sedikit ke pinggir, mereka akan mengeluarkan dengan lengkap oleh suatu gangguan kecil dalam masukan mereka. Orang dapat mudah mengerti, apakah pengetahuan program dan teknik ini dapat bekerja untuk masalah-masalah yang besar, di mana beberapa metoda perincian langsung.

Kita masih tidak memberikan suatu definisi khusus mengenai teknik AI, ini kemungkinan yang tidak mungkin dilaksanakan. Tetapi kita telah memberikan beberapa contoh apa yang bisa dan yang tidak bisa.

1.5. Tingkatan Dari Model

Sebelum mengatur untuk melakukan sesuatu, ini merupakan ide bagus untuk memutuskan dengan tepat apakah ini merupakan salah satu yang sedang diuji coba untuk melakukannya. Jadi kita semestinya bertanya kepada diri kita sendiri, apakah goal kita dalam uji coba untuk memproduksi program-

---

7) opcit [1), page 22.

program yang melakukan sesuatu kecerdasan seperti yang orang lakukan? Apakah kita mencoba untuk memproduksi program-program yang dapat melakukan tugas-tugas yang sama seperti manusia? atau, apakah kita mencoba untuk memproduksi program-program yang dapat melakukan secara sederhana

tugas-tugas apa saja dengan jalan memunculkannya dengan mudah? Ini telah memotivasi proyek-proyek AI oleh setiap goal ini.

Usaha untuk mengembangkan program-program yang melakukan tugas-tugas yang bisa dikerjakan manusia, dapat dibagi dalam 2 kelas:

1). Mencoba memecahkan persoalan yang tidak benar-benar patut/pantas sebagai definisi dari sebuah tugas AI.

Mereka merupakan masalah yang dapat dipecahkan dengan mudah oleh sebuah komputer, meskipun solusinya mudah, yang akan memanfaatkan mekanisme-mekanisme yang tidak dapat seperti manusia.

Sebuah contoh klasik untuk program kelas ini adalah:

EPAM (*Elementary Perceiver And Memorizer*), dengan memori yang dipetakan berpasangan dengan suku kata-suku kata omong kosong adalah mudah untuk sebuah komputer. Masukan EPAM sederhana, untuk menerima suatu relasi suku kata memberikan pendorong yang menghubungkan dengan yang satunya. Lalu pengamatan untuk mendorong salah satunya dan sebagai aki-batnya disimpan pada berikutnya, tetapi tugas ini sulit untuk orang. EPAM mensimulasikan sebauh jalan buat seseorang untuk bisa meningkatkan tugasnya.

Program ini dapat, bilamana alat-alat pembantu untuk ahli Psikologi yang ingin mengetes teori kemampuan manusia.

2). Mencoba kemampuan model manusia, merupakan sesuatu yang lebih jelas dilakukan tanpa definisi-definisi dari tugas AI, maka melakukan sesuatu yang tidak sepele\sederhana buat kom-puter.

Ada beberapa alasan untuk kemampuan model manusia yang menginginkan urutan-urutan tugas :

a). Ujicoba teori psikologi dari kemampuan manusia

Sebuah contoh menarik dari sebuah program yang telah ditulis untuk alasan ini adalah PARRY, yang merupakan sebuah model kelakuan manusia gila ketakutan. Model ini cukup baik, beberapa ahli psikologi diberikan kesem-patan untuk mengkonversikan

dengan program lewat sebuah terminal, mereka mendiagnosa gila ketakutan.

b). Memungkinkan komputer mengerti jalan pikiran manusia

Sebagai contoh, untuk sebuah komputer yang dapat memba-ca sebuah cerita surat kabar dan lalu menjawab sebuah pertanyaan, seperti: Apakah resiko mengadakan invasi? Program ini semestinya dapat mensimulasikan alasan proses dari orang.

c). Memungkinkan orang mengerti alasan komputer

Dalam beberapa lingkungan, seseorang akan menjadi engan untuk mengandalkan keluaran dari sebuah komputer. Tidak sedikit dari mereka akan mengerti bagaimana mesin tiba pada hasilnya. Jika proses jalan pikiran komputer sama dengan orang, lalu produksi dengan sebuah penjelasan yang dapat diterima, ini akan menjadi mudah.

d). Memanfaatkan apakah pengetahuan kita akan dapat dikumpulkan dari sedikit orang

Sejak orang merupakan pembentuk pengertian terbaik dari beberapa tugas dengan perputaran, membuat sejumlah rasa, untuk melihat mereka untuk menunjukkan bagaimana prosesnya.

Kita memerlukan metoda-metoda untuk membantu kita memecahkan secara serius dilema-dilema atau persoalan AI :

a) Sebuah sistem kecerdasan tiruan yang semestinya terdiri dari sejumlah pengetahuan, jika ia untuk menangani sesuatu tetapi masalah-masalah permainan sepele.

b) Tetapi sebagai perkembangan sejumlah pengetahuan, ia menjadi sulit untuk mengakses sesuatu yang tepat ketika diperlukan. Jadi beberapa pengetahuan semestinya ditambahkan untuk mem-bantu, tetapi sekarang ada beberapa pengetahuan untuk me-nanganinya, jadi beberapa semestinya ditambah dan sejauh mungkin.

Dalam goal AI kita, dikontruksikan/dibangun suatu program yang kerjanya memecahkan masalah yang kita anggap menarik.

Kecerdasan Tiruan(AI) masih merupakan disiplin ilmu yang muda. Kita telah mempelajari banyak hal, tetapi ini masih sulit untuk diketahui secara tepat gambaran dari yang semestinya digambarkan.

1.6. Representasi Persoalan/Masalah

Kita akan menerangkan 3(tiga) langkah utama yang termasuk untuk membangun sebuah sistem untuk memecahkan suatu masalah-masalah khusus:

1. Definisikan masalah/persoalan secara jelas Definisi ini termasuk kejelasan spesifikasi dari situasi harga awal(initial) atau start *state* akan lebih baik bahwa situasi akhir atau goal *state* berpedoman pada solusi yang pantas untuk persoalannya
2. Analisa Masalah
3. Sebagian kecil dari ciri-ciri penting dapat berpengaruh yang luas sekali pada kelayakan dari berjenis-jenis teknik yang mungkin untuk memecahkan masalah/persoalan.

Pilih teknik terbaik dan pakai teknik tersebut untuk masalah khusus.

Dasar untuk aplikasi-aplikasi AI pada umumnya adalah penyelesaian masalah (*Problem Solving*).

Dua tipe dari masalah:

a. tipe pertama dapat diselesaikan dengan menggunakan beberapa tipe prosedur deterministik, yaitu menjamin keberhasilan dan disebut: *Computation*

b. tipe kedua terdiri dari masalah-masalah yang diselesaikan oleh pencarian(*searching*) untuk penyelesaian.

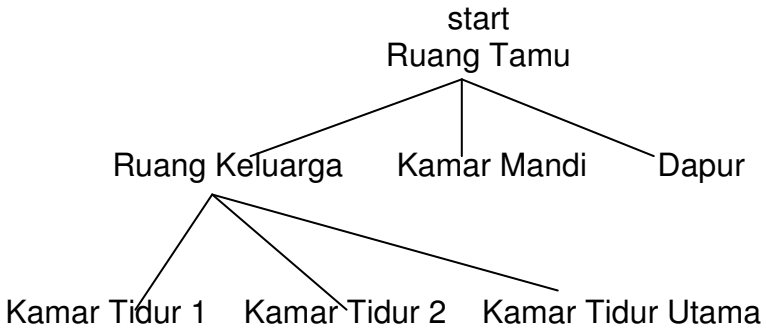
Contoh:

Bila terdapat denah rumah seperti gambar 1.4, Si Dul akan ujian Kalkulus di STMIK Budi Luhur, baru saja ia meninggalkan rumahnya dalam jarak 100 meter, ia teringat kalkulatornya tertinggal. Dan ia kembali ke rumahnya untuk mencari kalkulatornya.

Kamar Tidur 2	Kamar Tidur 1	Kamar Mandi	Dapur
Ruang Keluarga			
Kamar Tidur Utama		Ruang Tamu	

Untuk memudahkan si Dul mencari kalkulator di rumahnya yang dimulai dari ruang tamu hingga dapur, maka dibuatlah graph sebagai berikut:

GRAPH:



Gambar 1.4. Denah Rumah dan Graph kemungkinan pencarian

1.6.1. Representasi Tempat Kosong (*Representation in state space*)

Dalam *state* space representation dari sebuah masalah, node-node atau titik-titik dari sebuah graph korespondensi ke solusi-solusi tetap persoalan khusus dan tangan-tangan bersentuhan atau korespondensi ke langkah-langkah proses pemecahan sebuah persoalan.

Sebuah *state* awal(*start state*), korespondensi untuk memberikan informasi dalam cakupan sebuah masalah, bentuk akar(*root*) dari graph.

Graph juga mendefinisikan kondisi tujuan(*State Goal*), yang merupakan solusi dari cakupan sebuah persoalan.

State Space mencari karakteristik pemecahan persoalan, sebagai proses dari penemuan sebuah path/jalan kecil/garis edar solusi dari *state* awal sampai menghasilkan goal.

Sebuah goal boleh mendeskripsikan sebuah *state* atau ketetapan, seperti papan kemenangan Tic-Tac-Toe atau konfigurasi *Goal* dalam *8-Puzzle.*

Contoh : *8-Puzzle*[8)]

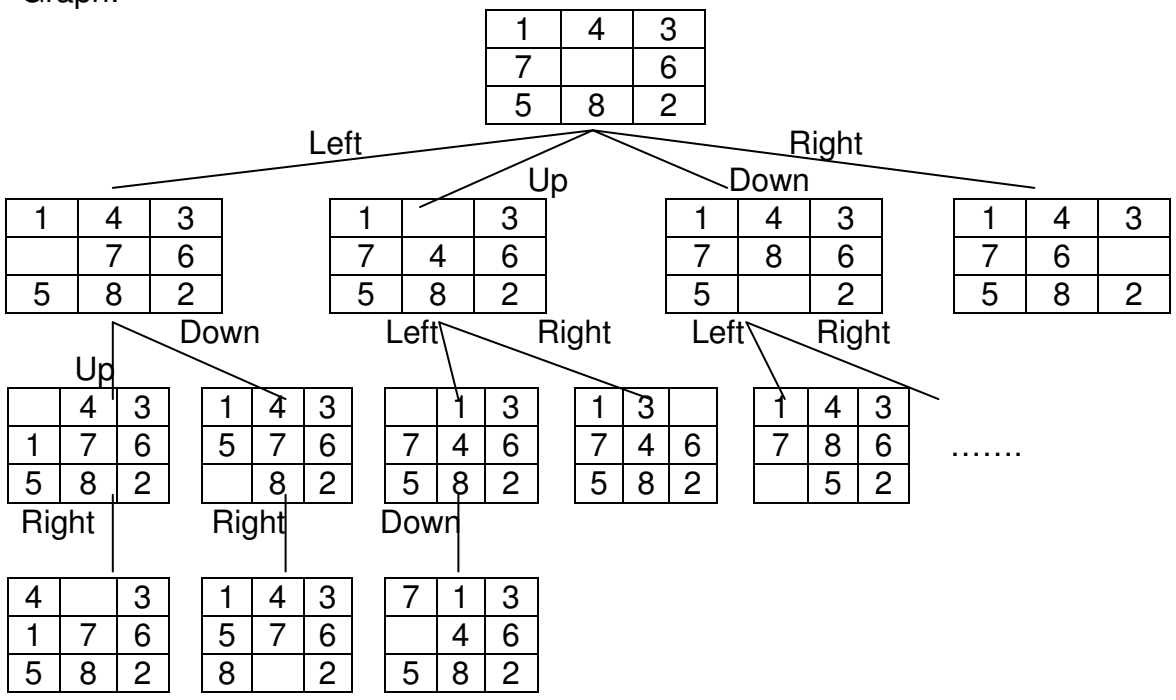
Start State :

1	4	3
7		6
5	8	2

Goal State :

1	2	3
8		4
7	6	5

Graph:



Gambar 1.5. Graph pencarian *8-Puzzle* dengan memindahkan tempat kosong

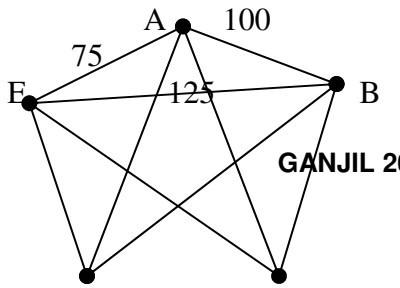
Dengan alternatif, sebuah goal dapat menggambarkan beberapa ketentuan dari solusi path sendiri.

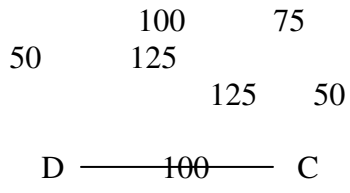
---

8) OPCIT [1)], page 47.

Dalam masalah perjalanan seorang pedagang(*Travelling Sales Person Problem*) dapat digambarkan seperti gambar 1.6.

Contoh : Search pada Travelling Salesperson Problem





Gambar 1.6. Jangkauan dari masalah perjalanan seorang pedagang

Pencarian berakhir ketika path terpendek ditemukan, yang mencakup semua titik(node) dari graph. Dalam masalah parsing, path analisis berhasil dari sebuah indikasi berakhir.

Hubungan dari kedudukan tempat kosong(*state Space*) korespondensi ke langkah-langkah proses solusi dan path melalui solusi representasi tempat kosong. Dalam berbagai tingkatan path lengkap dicari, mula-mula pada awal *state* dan dilanjutkan ke graph, sampai deskripsi goal didapatkan atau mereka batalkan.

Definisi Pencarian *state space*

Sebuah *state* space direpresentasikan dengan 4 komponen [N,A,S,GD] di mana:

N, adalah himpunan dari node atau *state* dari graph yang korespondensi ke *state-state* dalam sebuah proses pemecahan persoalan.

A, adalah himpunan dari hubungan/arc/links antara node-node yang korespondensi ke langkah-langkah dalam sebuah proses pemecahan persoalan.

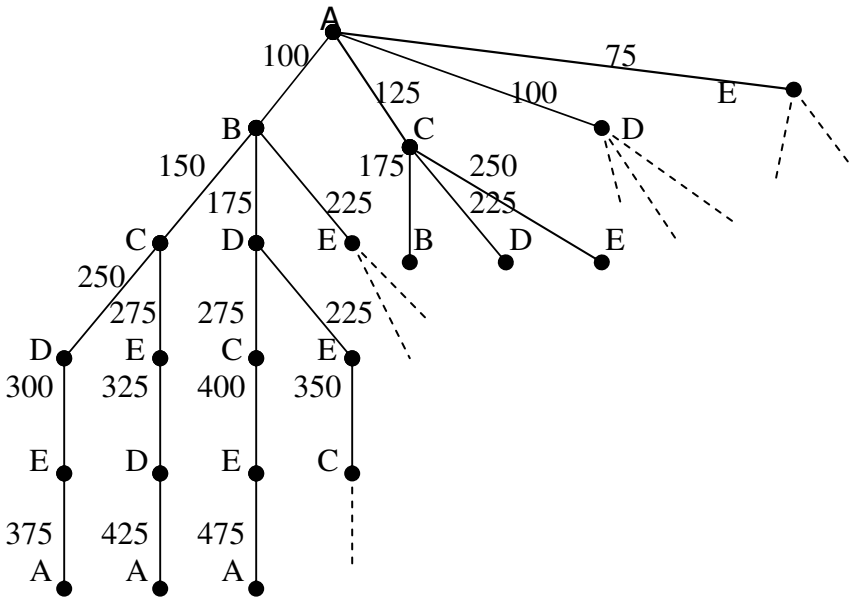
S, adalah himpunan bagian yang tidak kosong dari N, terdiri dari *state* awal(Start *State*) dari masalah.

GD, adalah himpunan bagian yang tidak kosong dari N, terdiri dari Goal *State* dari masalah. *State* dalam GD dideskripsikan menggunakan: ukuran ketentuan/rule dari *state-state* yang ditemukan dalam pencarian dan ketentuan dari pengembangan path.

Path Solusi adalah sebuah path yang melalui graph ini dari sebuah node dalam S ke sebuah Node dalam GD.

Gambar 1.7. menceritakan proses pencarian dari *Travelling Sales Person Problem*, setiap hubungan ditandai dengan total biaya dari semua path yang dilalui dari node awal (A) ke node akhir.

GRAPH:



Gambar 1.7. Pencarian dari *Travelling Sales Person Problem*

Kesimpulan :

Path:	Path:	Path:
ABCDEA	ABCEDA	ABDCEA
Biaya: 375	Biaya: 425	Biaya: 475

Kompleksitas dari pencarian yang mendalam *dalam travelling Sales person problem* adalah (n-1)!, di mana N adalah jumlah kota dalam graph. Untuk 9 kota, kita bisa mencoba secara mendalam semua path, tetapi untuk beberapa masalah menyangkut ukuran kepentingan, seperti 50 kota, pencarian tidak baik kemampuannya tanpa sebuah kriteria waktu/lamanya perjalan.

1.6.2. Pengetahuan(*Knowledge*) Untuk Pemecahan Persoalan

Contoh :

"The bank president ate a dish of pasta salad with the fork"

Dari kalimat di atas, tentu persepsi dari beberapa orang yang membacanya akan berbeda dari pengetahuan yang dimilikinya. Dan pengetahuan itu diperoleh dari pengalaman orang tersebut terhadap suatu objek. Sehingga ada anggapan sebagai berikut:

- Kata "*bank*" bisa dihubungkan pada suatu institusi keuangan atau pada suatu pingiran/tepi dari sebuah sungai, tetapi hanya satu yang mempunyai seorang presiden.

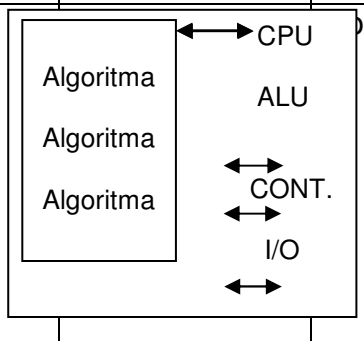
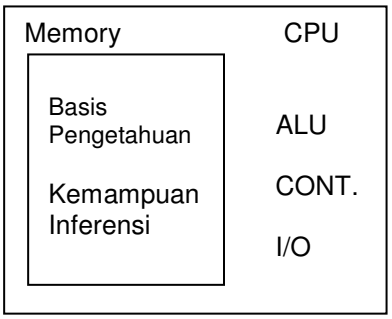
- Kata "*dish*" adalah objek dari kata kerja "*eat*". Ini mungkin saja bahwa sebuah piring(*a dish*) dimakan, tetapi lebih disukai yang dimakan adalah *pasta salad* di dalam piring.

- *Pasta salad* adalah suatu *salad* yang mengandung *pasta*, tetapi di sini ada pengertian lain yang dapat dibentuk dari padanan kata benda. Seperti : "*dog food*", secara normal tidak mengandung anjing/*dog*.

- Ungkapan "with the fork" dapat dimodifikasi menjadi beberapa bagian dari kalimat. Dalam kasus ini modifikasi kata kerja "*eat*". Tetapi, jika ungkapan telah diubah "*with vegetables*", maka perbedaannya terletak pada modifikasi struktur. Dan jika ungkapan "*with her friend*", struktur masih berbeda.

PERBEDAAN PEMROGRAMAN KONVENSIONAL DENGAN AI

		KONVENSIONAL *PROGRAMMING*	**AI *PROGRAMMING***
1	Yang Disimpan	Data	Pengetahuan
2	Yang Diolah	Angka, String --> Kuantitatif	Simbol-simbol Pengertian --> Kualitatif
3	Masalah	Menyelesaikan masalah	Menentukan cara solusi

4	Dasar Kerja	Dengan Algoritma	Manipulasi Simbol
5	Cara Kerja	Langkah demi Langkah --> hasil	Melacak Pengetahuan --> Jawaban
6	Tingkat Kerja	Perhitungan	Pengambilan Konklusi
7	Kemampuan Nalar	Ada Tidak punya kemampuan mengambil kesimpulan	Tidak Ada Punya kemampuan mengambil kesimpulan (Inferensi)
8	Kebutuhan Memori	Cukup	Sangat Besar
			Komputer 

Catatan :

- Dalam Implementasi AI Programming, algoritma masih diperlukan tapi bukan dasar kerja.
- Penyelesaian masalah dalam AI Programming --> melacak pengetahuan.

BAB II
LOGIC

Logic merupakan jantung dari program, para pemrogram mempunyai keyakinan jika sebuah komputer dapat dibuat mengerti logika, maka komputer dapat dibuat untuk berpikir, karena logika kelihatannya menjadi inti dari kecerdasan.

2.1. Sejarah Singkat Logika

Ahli logika pertama yang dikenal : Aristotle(384-322 BC), filosofi dan saintis alami Yunani. Aristotle telah mengembangkan banyak teori yang dikenal dengan *syllogistic* atau *classical logic.*

Syllogistic logic pada dasarnya bertransaksi dengan penurunan kebenaran (atau yang bersifat salah) dari argumen seorang filosofi. [9)]

Contoh: John is a man
All men used to the boys
Therefore, John used to be a boy

Logikanya adalah: John adalah seorang anak laki-laki sebelum dia menjadi orang dewasa.

Contoh diatas dikonversikan ke *Syllogistic logic*:

J ---> M
all M ---> B
hence: J ---> B

Symbolic logic dimulai dengan G.W. Leibniz(1646-1717), tetapi dilupakan setelah ia meninggal, kemudian seluruh lapangan tersebut dicakup kembali oleh: George Boole(1815-1864) dan logikanya dikenal dengan *Boolean Logic*. Symbolic logic berinteraksi dengan konsep abstraksi ke dalam simbol-simbol dan interkoneksi simbol-simbol oleh operator tertentu. [10)]

Contoh: if P is true
Q is false
Then P or Q is true
P and Q is false

[9)] Herbert Schildt, "Artificial Intelligence Using C", Mc.Graw-Hill, Singapura, 1987, halaman 268.
[10)] IBID [9)], halaman 269.

Dalam simbolic logic, terdapat dua perbedaan :

1. *Propotional Logic* : bertransaksi dengan kebenaran atau kesalahan dari sebuah proposition.

2. *Predicate Calculus*: memasukkan hubungan antara obyek-obyek dan kelas-kelas dari obyek.

Karena itu, sistem formal yang memanipulasi kalimat-kalimat standar menurut ketentuan(*rule*) yang dispesifikasikan dengan baik dan mengijinkan beberapa jenis dari *inference*(kesimpulan) yang dibuat. Sebuah sistem merupakan kombinasi dari *proposional logic* atau *proposional calculus* dan *first order predicate calculus*. Kesimpulan yang mendetail dari calculus ini perhatiannya pada mekanisme karakter pengikut. Ada juga suatu klasifikasi dari batas antara kalkulus dan tafsirannya(interpretasinya).

2.2. P*ropositional Logic*

Proposional logic berupa kalimat-kalimat lengkap dari fakta atau kenyataan (*facts/propositions*), seperti :[11)]

"john loves mary" atau "birds fly south in autumn"

Atau bisa dikatakan sebuah *propositional logic* bisa merupakan sebuah proposisi adalah kalimat yang terbentuk dengan sendirinya apakah bernilai *true*(benar) atau *false*(salah).

Propositional Logic menggunakan operator-operator untuk menghubungkan proposisi-proposisi(*propositions*) dalam bentuk ungkapan/ ekspresi / *expression* berupa kata penyambung *logika (Logical connectives*), yaitu berdasarkan tingkatan/*precedences*, sebagai berikut:

tingkatan	**Tanda/Sign**	**Arti** [12)]
1.	¬	NOT(*Negation*)
2.	∧	AND(*Conjuction*)
3.	∨	OR(*Disjunction*)
4.		IF ... THEN ...(Implikasi)
5.		IF and only IF...then... (iff)(*Double Implication*)
6.	≡	Assignment (*Equivalent*)

[11)] TR. Addis, "Designing Knowledge-Based Systems", KOGAN PAGE, 1986, halaman 182.
[12)] IBID [11)] , halaman 183

Contoh : 2 proposisi : P dan Q yang direpresentasikan sebagai ekspresi logika dengan menggunakan *logical connectives* dalam suatu tabel kebenaran, berikut ini:

P	Q	$P \wedge Q$	$\neg P$	$\neg P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$	$(\neg P \vee Q) \equiv (P \rightarrow Q)$
T	T	T	F	T	T	T	T
T	F	F	F	F	F	F	T
F	T	F	T	T	T	F	T
F	F	F	T	T	T	T	T

Ekpresi-ekspresi dibentuk menurut sebuah tata bahasa/grammar sederhana, dan ekspresi yang sesuai dengan tata bahasa ini disebut *well formed formulae(wffs).* Tanda kurung*(parantheses)* digunakan untuk membuat jelas urutan dari penempatan nilai kebenaran, jika tidak yang lain jelas. Suatu *Well Formed Formulae* merupakan salah satu suatu proposisi atau akan mempunyai salah satu bentuk seperti yang terlihat pada tabel berikut ini:

Jika P adalah sebuah Wff maka not P ($\neg P$) juga suatu Wff.

Jika P dan Q adalah dua Wffs, maka:

Wff
$(\neg P)$
$(P \wedge Q)$
$(P \vee Q)$
$(P \rightarrow Q)$
$(P \leftrightarrow Q)$

Sebuah Struktur (*atomic*) formula merupakan sebuah operasi *predicate letter* pada sebuah terminal(term) adalah sebuah Wff. Sebuah term merupakan sebuah variabel atau sebuah operasi fungsi pada sabuah variabel pada sebuah variabel.

contoh: f(x,y)

plus (S, 10)

<u>DOMAIN DARI INTERPRETASI (*INTERPRATATION*)</u>

Predicates……..Domain of Relations

Function ……Domain of Mapping

Variabel……..Domain of Constans, Numbers, dan sebagainya.

Kata penyambung cenderung mempunyai arti yang sama dengan bahasa alami mereka, jadi:

jika: [13)]

'*john love Mary's is True'* (P)

maka :

'*john does not love Mary's is False'* ($\neg$P)

dan jika :

'*Birds fly South in Autumn' is True*' (Q)

maka:

'*john loves Mary and Birds fly South in Autumn'* (P$\wedge$Q)

adalah sebuah proposisi gabungan yang bernilai *true.*

Bila mana, keadaan mesti diambil dalam menterjemahkan *Implication*, sejak ia boleh menggunakan suatu kondisi hubungan yang mempunyai sebuah sebab.

Jadi : '***if it is raining then the road are wet'***

Implikasi menghindari bahwa tafsiran memerlukan keduanya didefinisikan secara formal dan mengikutkan transaksi suatu perubahan lingkungan.

Disini bisa saja : John may not always love Mary dan

The road are not always wet.

Sebuah tafsiran (*Interpretation*) dari sebuah Wff merupakan suatu penempatan nilai kebenaran pada proposisi atom komponennya.

No. Tafsiran	It is raining (P)	The roads are wet (Q)	*Wff* P $\rightarrow$ Q
1	*T*	*T*	*T*
2	T	F	F
3	F	*T*	T
4	F	F	T

---

[13] IBID [11] , halaman 184 -185

2.3. *Predicate Calculus*

Kadang disebut *Predicate Logic* adalah penyederhanaan ekstensi *propositional logic*.

PREDICATE CALCULUS → keunggulan : Dalam pendefinisian Semantic(ARTI KATA).

→ Pembuktian kebenaran peraturan–peraturan kesimpulan dengan baik. (*INFERENCE RULE*).

PREDICATE CALCULUS merupakan salah satu dari skema-skema yang digunakan dalam Representasi Pengetahuan.

Anda dapat menganggap ini sebagai sebuah Bahasa Representasi untuk AI.

Setiap Bahasa mempunyai :

(a) SYNTAX

(b) SEMANTIC

Syntax dari *Predicate Calculus* terdiri dari :

Predicate Letters : P, Q, R, P_1, P_2 . . . . .

Function Letters : f, g, h, f_1, f_2 , ...

Variabel *(Variables)* : X, Y, Z, X_1, X_2, ...

Konstanta (*Contants*) : a,b,c,a_1,a_2, . . … .

Semantic adalah arti yang berhubungan dengannya. Setiap dari mereka boleh mempunyai sebuah daerah/domain. Definisi dari sebuah domain disebut sebuah INTERPRETASI dihimpun dari arti kata/*Semantic*.

Dasarnya: *PREDICATE*, dimana beberapa *function* yang mengembalikan sebuah nilai benar /salah tergantung pada argumennya.

Perbedaan dasar antara *Predicate Logic & Propositional Logic* adalah :

Pemisahan attribute dari obyek yang kemungkinan milik atribute, yaitu dalam *predicate calculus* dimungkinkan untuk membentuk sebuah fungsi yang menentukan kesulitan sebuah obyek yang diberikan. Dalam *Propositional Logic* , kita harus membentuk statement baru untuk setiap kasus. Jika yang ada hanya *Predicate Letters* (Dan tanpa variabel *Quantifier,* dan sebagainya) disebut *O_order Predicate calculus* atau *Prepositional calculus*, atau *Sentential calculus.*

Walaupun bentuk *Propositional Logic* dasar untuk kecerdasan dan bahasa komputer, tetapi kita tidak dapat menggunakan bentuk ini dengan sendirinya untuk menyatakan pengetahuan manusia di dunia, karena bentuk ini kurang mampu untuk menunjukkan hubungan antar obyek, bentuk ini terbatas hanya untuk penentuan kebenaran atau kesalahan dari sebuah contoh yang diberikan dan tidak dapat digunakan pada klasifikasi. Satu hal yang penting bahwa *Predicates* dapat memiliki beberapa argument.

Untuk *Propositional Expression* P,Q, & R : [14)]

P_1 . . . . . .~ (~P) ≡ P (*double negative/inverse*)

P_2 . . . . .(P ∨ Q) ≡ (~ P → Q) juga (P → Q) = (~ P∨ Q)

P_3 . . . .Hukum De Morgan 1 : ~ (P ∨ Q) ≡ (-P ∧ ~ Q)

P_4 . . . .Hukum De Morgan 2 : ~ (P ∧ Q) ≡ (-P ∨ ~ Q.)

P_5 . . . .Hukum Distributive 1 : P ∨ (Q ∧ R) ≡ (P ∨ Q) ∧ (P ∨ R)

P_6 . . . .Hukum Distributive 2 : P ∧ (Q ∨ R) ≡ (P ∧ Q) ∨ (P ∧ R)

P_7 . . . Hukum Komutatif 1 : (P ∧ Q) ≡ (Q ∧ P)

P_8 . . . Hukum Komutatif 2 : (P ∨ Q) ≡ (Q ∨ P)

P_9 . . . Hukum Assosiatif 1 : ((P ∧ Q) ∧ R) ≡ (P ∧ (Q ∧ R))

P_{10} . . . Hukum Assosiatif 2 : ((P ∨ Q) ∨ R) ≡ (P ∨ (Q ∨ R))

P_{11} . . . Hukum Kontrapositif : (P → Q) ≡ (~Q → ~ P)

Sebuah jalan yang tepat untuk menggambarkan persamaan logika dari jenis ini adalah sekumpulan *production rules* (yang mengkombinasikan dalam sebuah rule : *forward* dan *backward*) untuk memecahkan persoalan dalam *propositional calculus*. Untuk membuktikan sebuah exspresi Q (*goal*) dari pemberian sebuah Wff tunggal. (data base awal), sebuah proses pemilikan suatu urutan dari akhir persamaan dalam Q akan membuat Q dari Wff asli.

[14)] IBID [11)] , halaman 187

Persamaan adalah langkah-langkah dalam sebuah argumen gabungan dan setiap langkah adalah *valid* (benar) (selalu benar, tidak masalah apa tafsirannya).

Sebuah *Argument* adalah sekumpulan dari Wffs diikuti oleh sebuah Wff tunggal disebut kesimpulan (*Conclusion*).

Argument dapat dirangkai adalah sebuah *argument* tunggal atau *proof*. Sebuah argument lengkap dapat dibuat dalam sebuah kalimat:

‘ **if <** *premise**s*** **>** **then** <*conclusion*> ‘.

Dasar pikiran/alasan kesimpulan

Kesimpulan dikatakan mengikuti secara logika dari dasar pikiran dan tergantung dari proposisi (sebuah argument tetap valid tidak masalah apakah penambahan alasan di suplay) merupakan karakter *monotonic* dari Propositional dan *Predicate Calculus*.

Features monotonic tidak selalu di tampilkan dalam argumentasi alamiah sejak informasi baru (tafsiran) dapat mengubah sebuah kesimpulan. Mekanisme dari logika bukan *monotonic* tidak sepenuhnya dimengerti.

Untuk logika *monotonic* bentuk argumennya disebut

modus ponens: $(P, P \rightarrow Q) \therefore Q$

Untuk logika bukan *monotonic* bentuk argumennya disebut

Modus tollen : $(\sim Q, P \rightarrow Q) \therefore \sim P$

Dan kedua mekanisme tersebut yang Wff dapat di *generated*. Keduanya dapat diperiksa dengan tabel kebenaran untuk mengkonfirmasikan atau menyatakan kalimat-kalimat:

$(P \wedge (P \rightarrow Q) \rightarrow Q$ is valid

dan:

$(\sim Q \wedge (P \rightarrow Q)) \rightarrow \sim P$ is valid

Secara respektif untuk *modus ponens* :

P	Q	P → Q	P ∧ (P → Q)	(P∧(P→))→Q
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

Contoh yang lain : (Q, P → Q) ∴ P adalah *a fallacy (not valid)*

Contoh :

- Jika P merupakan *propositional* " hari ini hujan" dan Q merupakan *propositional* " saya ada kerjaan" maka himpunan dari *propositional* (P,Q) mempunyai 4 fungsional berbeda dalam nilai kebenaran (T,F).
- P merupakan kalimat " hujan pada hari selasa", kita dapat membuat predicat cuaca yang menjelaskan sebuah hubungan diantara hari dan cuaca, sebagai berikut:

  Cuaca (selasa , hujan)
- Mengarah pada ketentuan–ketentuan kesimpulan (*Inference Rules*), kita dapat merubah atau memanipulasi exspresi predikat kalkulus di atas, dengan mengakses komponennya sendiri-sendiri dan menduga kalimat baru.
- Sebagai contoh : kita dapat menetapkan semua nilai dari X, dimana X: satu hari dan semnggu. Kalimatnya menjadi sebagai berikut:

  Cuaca (X, hujan)

2.4. ***First Order Predicate Calculus***

Menggunakan *Quantifier :*

- *Universal Quantification* (∀x) ∀: *all, every*, untuk semua
- *Existential Quantification* (∃x) ∃ :*some, at least one*, terdapat, beberapa

Contoh:

Everybody loves Mary $\rightarrow (\forall x)$ [loves (x,Mary)]

Somebody loves Mary $\rightarrow (\exists x)$ [loves (x, Mary)]
All swans are white $\rightarrow$ $\boxed{(\forall x) \text{ [swan (x) } \rightarrow \text{colour (x, white)]}}$
There is ablack swan $\rightarrow (\exists x)$ [Swan (x) $\wedge$ colour (x, black)]

There is Somebody who loves every one :

$(\exists x)(\forall y)$ [loves (x,y)]

Everybody is loved by some one :

$(\forall x)(\exists y)$ [loves (x,y)]

Table persamaan logika tambahan untuk predicate calculus

Refereces	**Persamaan Logika**
P_{12}	$\sim(\exists_x)$ [P(x)] $\equiv$ $(\forall_x)$ [$\sim$ P(x)]
P_{13}	$\sim(\forall_x)$ [P(x)] $\equiv$ $(\exists_x)$ [$\sim$ P(x)]
P_{14}	$(\forall_x)$ [P(x) $\wedge$ Q(x)] $\equiv$ $(\forall_x)$ [P(x)]$\wedge$ $(\forall y)$ [Q(y)]
P_{15}	$(\exists_x)$ [P(x) $\vee$ Q(x)] $\equiv$ $(\exists_x)$ [P(x)]$\wedge$ $(\exists y)$ [Q(y)]
P_{16}	$(\forall_x)$ [P(x)] $\equiv$ $(\forall_x)$ [P(y)]
P_{17}	$(\exists_x)$ [P(x)] $\equiv$ $(\exists_x)$ [P(y)]

Clause formation

Merupakan kreasi clauses dari beberapa Wff (*Well formed formula*) *prosedure* mengikuti 9 langkah yang semestinya dilakukan berurutan, ketika *aplicable*

1. Eliminasi Simbol implikasi (Implication / $\rightarrow$) semua simbol implikasi di hilangkan dengan bersamaan logika $P_{2.}$ ($\sim$ P v Q $\equiv$ P $\rightarrow$ Q)
2. *Reduce Scope of the negation Symbol* : jadi tanda negasi dipindahkan kedalam tanda kurung dekat pada setiap atom formula yang di pakai. (gunakan P_1, P_3, P_4, P_{12} & P_{13})
3. Standarisasi variabel : Proses ini menamakan ulang setiap variabel juga setiap Quantifier mempunyai variabel sendiri. Ini menjamin variabel-variabel dihubungkan pada Quantifier mereka secara bebas dari bentuk wff. (gunakan P_{16} & P_{17})
4. Eliminasi *Existential Quantifier* ($\exists x$) : Ini menyediakan penggunaan fungsi (function)skolem.

   Sebuah fungsi skolem merupakan fungsi pemetaan tambahan yang

menggantikan sebuah variabel melewati *existential Quantifiers.*

Contoh: $(\forall x)\ (\exists y)$ [loves (x,y)]

Sekarang sebuah fungsi g dapat di definisikan bahwa y = g(x) dan setiap x akan secara otomatis dipetakan dalam beberapa y yang ada menurut formula (pilih y), maka:

$(\forall x)$ [loves (x, g(x))]

5. *Convert to Prenex Form:*

   Ini merupakan perpindahan secara sederhana dari semua universal Quantifier ke depan dari wff. Sejak setiap Quantifier mempunyai Variabelnya sendiri, *correct stops*, tidak hilang.

6. Letakkan badan utama dari wff kedalam bentuk *conjuctive* normal : jadi sekumpulan clauses di produksi. (Gunakan P_6)

7. Eliminasi *the universal Quantifier* $(\forall x)$

   Ini merupakan proses sederhana menghilangkan mereka dari depan wff. Asumsi ditetapkan bahwa semua unbound variabel adalah universally Quantifier, jadi tidak memerlukan secara exsplisit kedudukan ini.

8. Eliminasi semua simbol $\wedge$:

   Setiap *clause* dipisahkan sekarang. Sejak tafsiran selalu dikonsentrasikan dengan *conjuction* dari dasar pikiran , <u>*clauses*</u> boleh dipisahkan.Ini merupakan notasi.

9. Standarisasi sebagai variabel :

   Ini merupakan proses penamaan kembali setiap variabel, jadi tidak ada variabel lebih dari satu anak kalimat.

Contoh : *USE Algorithm Convert to clauses Form* :

$(\forall x)\ \{ P(x) \rightarrow \{ (\forall y)\ [\ P(y)\ \rightarrow P(f9x,y))] \wedge\ \sim (\forall y)\ [Q(x,y) \rightarrow P(y)]\}\}$

Langkah 1.

Menjadi:

$(\forall x)\ \{\sim P(X) \vee \{(\forall y)\ [\ \sim P(y)\ V\ P(f(x,y))] \wedge \sim (\forall y)\ [\ \sim Q(x,y) \vee P(y)]\}\}$

Langkah 2.

$(\forall x)\ \{\sim P(x) \vee \{(\forall y)\ [\ \sim P(y)\ V\ P(f(x,y))] \wedge (\exists y)\ [Q(x,y) \wedge \sim P(y)]\}\}$

Langkah 3.

$(\forall x)\ \{\sim P(x) \vee \{(\forall y)\ [\sim P(y)\ V\ P\ (f\ (x,y))] \wedge (\exists Z)\ [Q(x,Z) \wedge \sim P\ (Z)]\}\}$

Langkah 4.

$(\forall x)\ \{\sim P(x) \vee \{(\forall y)\ [\sim P(y)\ V\ P\ (f\ (x,y))] \wedge [Q(x,g(x)) \wedge \sim P\ (g(x))]\}\}$

Langkah 5.

$(\forall x)\ (\forall y)\ \{\sim P(x)\ V\ \{[\sim P\ (y) \vee P(f\ (x,y))] \wedge [Q(x,g(x\ )) \wedge \sim P\ (g(x))]\}\}$

Langkah 6.

$(\forall x)\ (\forall y)\ \{\ [\sim P(x) \vee \sim P(y) \vee P(f\ (x,y))] \wedge [\ \sim P(x) \vee Qn(x,g(x))]$

$\wedge\ [\sim P\ (x) \vee \sim P(g(x))]\}$

Langkah 7.

$\{\ [\sim P(x) \vee \sim P(y) \vee P\ (f(x,y))] \wedge [\ \sim P(x) \vee Q(x,g(x))] \wedge [\ \sim P(x) \vee \sim P\ (g(x))]\}$

Langkah 8.

$\sim P(x) \vee \sim P(y) \vee P\ (f(x,y))$.

$\sim P(x) \vee Q\ \ (x,g(x))$

$\sim P(x) \vee\ \ P(g(x))$.

Langkah 9.

$\sim P(x_1\ ) \vee \sim\ \ P\ (y)\ \ \vee P\ (\ f(x_1,y))$.

$\sim P(x_2) \vee Qn\ (x_2,g\ (x_2))$.

$\sim P(x_3\ ) \vee \sim\ \ P\ (g\ (x_3))$.

<u>Contoh yang lain:</u>

1). All Romans who knows Marcus either hate caesar or think that anyone who hates any one is crazy.

Kita dapat merepresentasikan kalimat diatas kedalam wff sebagai berikut:

$(\forall_x)$ [Roman (x) $\wedge$ know (x,Marcus)] $\rightarrow$ [hate (x ,Caesar) v $(\forall_y)$ $(\exists_Z\ )$ hate(y,z) $\rightarrow$ thinkcrazy (x,y))].

2). There is some one who's going to pay for all the service There for each of the service is going to be paid for by some one.

Bentuk predicate calculus kalimat diatas sebagai berikut :

P(x) = x is a person
S(y) = y is a service
G (x,y) = x is going to pay for y

Kita lihat bahwa jika dasar pikiran atau alasan :

$(\exists_x)$ [P(x) $\wedge$ $(\forall_y)$ [S (y) $\rightarrow$ G (x,y)]]

maka kesimpulan :

$(\exists_y)$ [S(y) $\rightarrow$ $(\exists_x)$ [P(x) $\wedge$ G (x,y)]]

maka wff:

$(\exists_x)$ [P(x) $\wedge$ $(\forall_y)$ [S (y) $\rightarrow$ G (x,y)]] $\rightarrow$ $(\exists_y)$ [S(y) $\rightarrow$ $(\exists_x)$ [P(x) $\wedge$ G (x,y)]]

Representasi kenyataan-kenyataan (*facts*) sederhana dalam *logic*

Penggunaan dari *propositional logic* sebagai langkah / cara merepresentasikan dari pengetahuan dunia yang diperlukan sebuah sistem AI. *Propositional logic* adalah menarik sebab dia sederhana untuk menghadapi prosedure keputusan untuk keberadaanya.

Kita dapat dengan mudah menampilkan *fact-fact* dunia nyata sebagai proposisi logika yang ditulis sebagai wffs. Beberapa *facts* sederhana dalam *propositional logic* :

It is raining
Raining

It is sunny
Sunny

It is windy
Windy

If is raining, then is not sunny
Raining $\rightarrow$ $\neg$ sunny

Contoh penggunaan *predicate logic* sebagai cara untuk merepresentasikan pengetahuan ;

1. Marcus was a man
   man (marcus)

2. Marcus was a pompeian
   pompeian (marcus)
3. All pompeian were Romans
   $\forall_X$: pompeian (x) $\rightarrow$ roman (x)

4. Caesar was a ruler
   ruler (caesar)

5. All romans were either loyal to caesar or hated him
   $\forall_{X:}$ roman (x) $\rightarrow$ loyal to (x, caesar) v hate (x ,caesar)

   Inclusive Interpretation
   $\forall_{X:}$ roman(x) $\rightarrow$ [(loyal to (x, caesar) v hate(x ,caesar) $\wedge$ $\neg$(loyalto(x, caesar) $\wedge$ hate(x,caesar))]

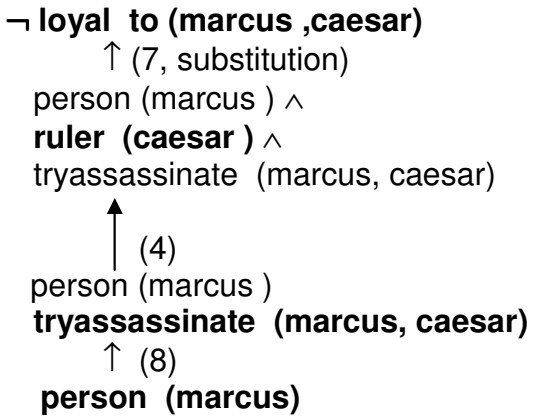
6. Every one is loyal to some one
   $\forall_{X:}$ ($\exists_y$) : loyal (x,y)

   some one to whom every one is loyal
   $\exists_x$:$\forall_y$: loyal to (x,y)

7. People only try to assassinate rulers they are not loyal to
   $\forall_{X:}$: :$\forall_y$: person(x) $\wedge$ rulers (y) $\wedge$ tryassassinate (x,y) $\rightarrow$ $\neg$ loyal to(x,y)

8. Marcus tried to assassinate Caesar
   tryassassinate (marcus, Caesar)

9. All men are people
   $\forall_X$: man (x) $\rightarrow$ person (x)



Gambar 2.1. memperlihatkan sebuah usaha membuktikan bahwa marcus tidak loyal (setia) kepada caesar:

¬ loyal to (marcus , caesar)

Bila pengetahuan di atas dan untuk membuktikan bahwa marcus tidak loyal

kepada caesar dituangkan dalam bahasa Turbo Prolog, sebagai berikut:

```
Domains
        Simbol = string

Predicates
        man(Simbol)
        ruler(Simbol)
        tryassassinate(Simbol, Simbol)
        person(Simbol)
        notloyalto(Simbol., Simbol )

Clauses
        man(marcus).
        ruler(caesar).
        tryassassinate(marcus, caesar).
        person(X)      :-      man(X).
        notloyalto(X, Y) :-    person(X),
                               ruler(Y),
                               tryassassinate(X, Y).
```

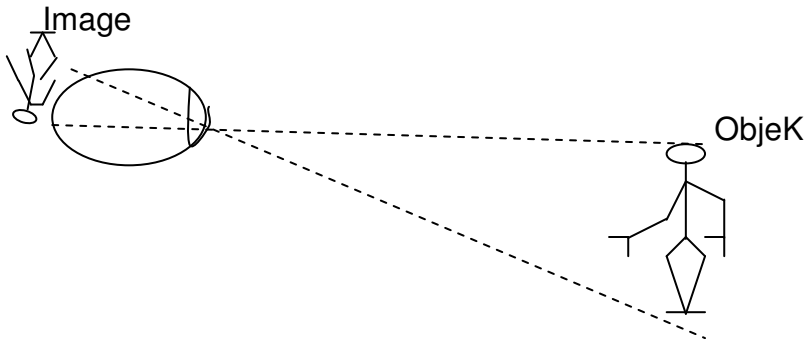
setelah itu tekan tombol ALT + R untuk RUN , sehingga muncul menu dialogue sebagai berikut:

```
Goal:  notloyalto(A,B).
       A=marcus;
       B=caesar;
       No
Goal:  notloyalto(P, caesar).
       P=marcus;
       No
```

BAB III
VISION

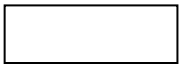
Penglihatan merupakan indera kita yang paling mengesankan. Ia memberiakan kita informasi yang detail / rinci tentang keadaan dunia di sekeliling kita.

Vision dimulai dengan mata, sebuah alat untuk menangkap dan menfokus penglihatan yang dikembangkan dari objek. Setiap titik pada sebuah objek (yang bukan sebuah cermin) mempunyai lensa lens / pinhole yang langsung mengarahkan penglihatan dari sebuah titik pada sebuah objek ke titik pada suatu sisi permukaan.[15]



Gambar 3.1. *Image* dari orang kurus / *stickman*

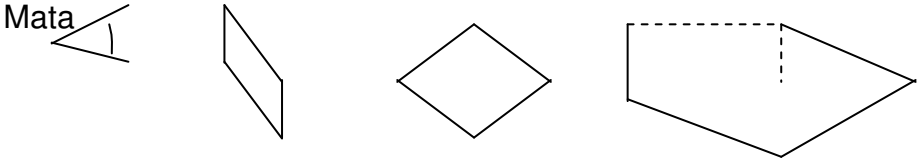
Sebuah *image* merupakan pola dari tingkatan arah penglihatan. *Image* merupakan sisi atas bawah dan pembiasan kiri ke kanan, tetapi ini tidak relevan, tidak satupun kelihatan, hanya bentuk komputasinya. Ini akan relevan bahwa image merupakan dua dimensi, ketika objek merupakan tiga dimensi.

Image: 

Gambar 3.2. Tiga Interpretasi dari sebuah *image* persegi panjang

[15] Charniak, Eugene & McDermott, Drew, "Introduction to Artificial Intelligence", page 87 – 160, Addision – Wesley, 1985.

Procedure kemungkinan bentuk kawat (wire frames):



3.1. Visi Komputer (*Computer Vision*)

Visi merupakan tugas pengolahan informasi dalam memahami suatu pemandangan (scene), dari citra-citra yang diproyeksikan.

Citra, atau *image* adalah fungsi dua dimensi f(x, y) yang didapatkan dari peralatan sensor yang mencatat harga ciri citra pada semua titik elemen (x, y).

Elemen citra disebut *pixel* atau *picture element.*

Harga meliputi:

- Tingkat keabuan (intensitas) ⟶ citra tonal
- vektor warna ⟶ citra berwarna

Secara matematis, citra merupakan kumpulan larik (matriks) {f(x, y)} atau sama dengan kumpulan harga-harga pengukuran pada setiap lokasi *pixel*.

Tugas sistem visi komputer, adalah:

Memahami scene yang dilukiskan oleh citra (kumpulan *pixel-pixel*)

Banyak bidang ilmu yang menyatakan tujuan serupa, yaitu:

- pengolahan citra
- pengenalan pola
- analisis scene
- dan lain-lain

Pada saat awal, *computer vision* memerlukan mata-mata dari *computer vision*:

- *vidicon tube*
- *ccd camera (Charge Coupled Device)*

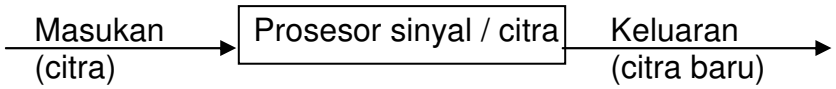
kemudian dilakukan proses pencuplikan (*acquisition*), yaitu mengubah informasi visual ke dalam suatu format yang selanjutnya dapat dimanipulasi.

Kamera akan mengambil gambar dengan melakukan scanning, selanjutnya membentuk sinyal analog dimana amplitudonya menyatakan kecerahan (*brightness*).

Digital komputer tidak dapat memproses sinyal analog, untuk bisa memproses gambar, maka perlu *interface*, yaitu ADC (*Analog – to Digital – Converter) card.*

Metode riset visi dibagi dalam 3 kategori:

1. Pengolahan Sinyal / Citra



Pengolahan citra membantu menyempurnakan / memperbaiki kualitas citra untuk dianalisa dan dipahami.

Citra digital: kumpulan dari bilangan-bilangan bulat atau digit objek pengamatan diukur tingkat intensitasnya, yaitu:

Gelap = Hitam = 0 disebut *Gray Level*

Atau skala keabu-abuan

Putih = Terang = 255

Pengolahan citra digital yang terbentuk dari kumpulan bilangan bulat yang kita kehendaki.

Citra: 1 dimensi → sinyal → pengolahan citra digital
2 dimensi → potret
3 dimensi → hologram

2. Klasifikasi citra

Klasifikasi citra adalah mengklasifikasikan citra ke dalam kelas-kelas yang telah ditentukan sebelumnya.

Contoh:

Pengenalan karakter untuk mendeteksi suatu tanda tangan palsu atau tidak

→ pengenalan pola(P*attern Recognation*).

Disini ada proses pembuatan keputusan dalam mencocokan sebuah kelas

dengan menggunakan metode keputusan berdasar statistik dalam ruang multidimensi.

3. Pemahaman citra

   Menjelaskan tidak hanya citra belaka, tetapi juga pemandangan (*scene*) yang dilukiskannya.

   → *Analisis scene*

   3 level pengolahan informasi dalam computer vision, yaitu:

   a. *low level* → pengolahan awal

   b. *intermediate level* → segmentasi

   c. *high level* → deskripsi scene

Ada perbedaan antara citra dengan gambar:

Gambar (picture):
- ♦ tidak bisa diproses
- ♦ untuk bisa diprose harus diubah ke bentuk citra
- ♦ representasi yang muncul dari lukisan gambar biasa.

Foto → citra

Citra (image):
- ✣ Gambar yang diubah ke bentuk matriks
- ✣ Representasi yang mengandung informasi deskriptif tentang objek.

Masalah-masalah yang ada dalam *computer vision*:

- Pengaruh terhadap kenampakan objek. (permukaan, tepi (*contour*) objek, dll.)
- Proyeksi 3D → 2D atau Scene → citra
- perlu informasi awal tentang objek
- keterbatasan manusia dalam mengintrospeksi yang dilihatnya → sulit membuat analisis protokol
- masalah teknis

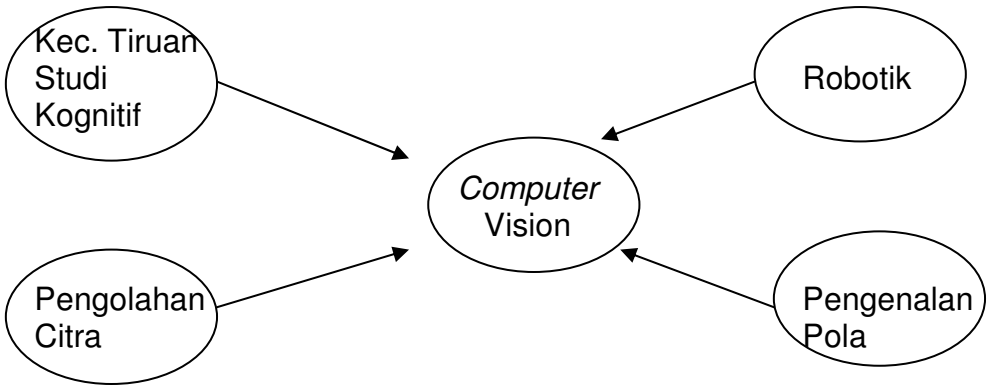
  jumlah informasi yang diolah besar sekali.

  1 bingkai bisa 3000 x 3000 → 9 M byte

KATEGORI DISIPLIN YANG SANGAT BERKAITAN:

INPUT	*OUTPUT*	
	CITRA	DESKRIPSI
CITRA	PENGOLAHAN CITRA	PENGENALAN CITRA VISI KOMPUTER
DESKRIPSI	KOMPUTER GRAFIK	LAINNYA

Visi komputer banyak digunakan dalam proses industri / otomatisasi jadi erat kaitannya dengan robotik

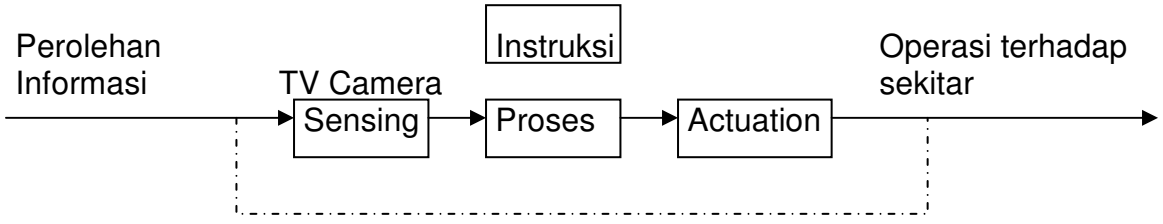


Pengenalan pola: identifikasi / interpretasi citra, tujuannya menyadap informasi mengenai citra yang ingin diperhatikan.

Robotik: Ilmu yang dapat mengendalikan gerakan terkoordinir dengan cerdas / ilmu yang merepresentasikan kecerdasan ke energi.

Kecerdasan Tiruan studi kognitif:

Memberikan panduan untuk melakukan pengenalan atau pengamatan objek.



Komponen sensor: sebagai mata untuk melihat sekitar, dapat berupa CCD atau TV Camera

Komponen Proses: mengontrol gerakan sistem, apa yang akan dijalankan.

Komponen Aktuator: tenaga mekanis (melaksanakan instruksi)

Jadi dalam kecerdasan tiruan:

Mata ⟶ *Computer Vision*
Syaraf ⟶ *Neural Network*
Otak ⟶ *Expert System*

Tiga elemen dasar sistem visi, yakni untuk mendapatkan:

- representasi digital
- memodifikasi data dan
- menyatakan keluaran citra

atau dengan kata lain: a. Akuisisi Citra: kamera, frame graber, dll.

b. Pengolahan: S / W dan H / W

c. Keluaran / penampilan: kontrol proses, pola, deskripsi.

PERBANDINGAN SISTEM VISI MANUSIA DAN MESIN

HAL	MANUSIA	MESIN
Fleksibilitas	Sangat fleksibel dan adatif terhadap tugas dan jenis masukan	Sangat kaku terhadap tugas, memerlukan data tercuplik
Abilitas	Dapat melakukan perkiraan yang relatif teliti, misal: deteksi objek berdasarkan warna dan bentuk	Dapat melakukan pengukuran dimensi berdasar data yang telah ditentukan sebelumnya
Warna	Subjektif terhadap warna	Warna ditentukan oleh parameter kromatisitas (RGB)
Sensitifitas	Adaptif terhadap kondisi cahaya, sifat fisis permukaan dan jarak terhadap objek	Sensitif terhadap tingkat dan frekuensi pencahayaan maupun keadaan fisik permukaan dan jarak objek
2D & 3D	Mudah menangani 3D dan warna	2D mudah, tapi 3D sukar, perlu algoritma, agak rumit
Data keluaran	Secara manual, laju kesalahan tinggi	Secara otomatis dapat menangani masukan diskrit

3.2. Pengenalan Bentuk Dan Bayangan

Jika komputer berhubungan secara menyeluruh dengan dunia manusia, maka dibutuhkan beberapa kemampuan bayangan(*vision*).

Pengolahan citra menjelaskan beberapa lapangan, yaitu:

1. Bayangan(*vision*)
2. Pengenalan bentuk (*Pattern Recognition*)
3. Peningkatan citra (*Image Enchancement*)

Lapangan-lapangan tersebut begitu besar sehingga dibagi ke dalam dua sub divisi:

1. Pengolahan dua dimensi
2. Pengolahan tiga dimensi (Pengolahan Dunia Nyata)

3.3. *Filtering, Contrast*, dan *Shading*

Vision system dapat diterapkan dalam metoda:

1. *Goal* : mengurangi bayangan garis yang membentuk *outline* setiap obyek.

   Digunakan oleh berbagai *Filter* untuk menghapus informasi dari bayangan dan pengembangan *contrast* untuk membuat semua bagian dari bayangan tersebut apakah hitam atau putih. Hal ini disebut dengan *Binary Image* karena tidak ada area warna lain kecuali hitam atau putih.

   Filtering dapat dikerjakan secara digital, tetapi secara umum terdapat dalam sistem sederhana dengan menggunakan rangkaian analog untuk menghasilkan *contrast* yang baik.

   Keuntungan *Binary Image*: Menentukan secara jelas batasan di mana komputer dapat dengan mudah mengenal alam menggunakan algoritma sederhana.

2. Mencoba memberikan komputer yang menyerupai pandangan manusia tentang *image*(bayangan).

   Metoda ini memberikan informasi tentang kejelasan bagian bayangan ke komputer.

Semua *vision system* menggunakan bayangan putih dan hitam untuk menggantikan warna, karena dua alasan:

a. Warna umumnya tidak dibutuhkan
b. Tambahan informasi warna menempatkan pada kebutuhan yang lebih besar baik pada komputer dan *software* yang mengolah bayangan.

3.4. Sistem Dua Dimensi

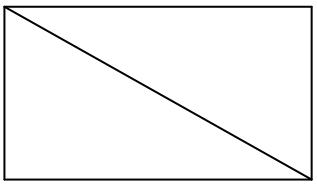
Sistem dua dimensi membutuhkan susunan lingkungan yang terbatas dan terkontrol dengan jelas, karena proses semua image adalah tapak/telapak, sering disebut: Pengolahan Citra Telapak (*Flat-Image Processing*).

Hal-hal yang harus diperhatikan:

1. Obyek sesungguhnya tidak membutuhkan tapak, tetapi biasanya tiga dimensi. Kebutuhan sistem dua dimensi adalah bayangan tiga dimensi dapat dikurangi ke bayangan dua dimensi, tanpa menghilangkan identitasnya.
2. Pada dasarnya kesuksesan *processor* bayangan dua dimensi yang sederhana adalah obyek dipandang dengan jelas untuk kontrol dan berbagai variasi tidak nampak.

   Masalah dalam sistem dua dimensi:

   Sistem mungkin sulit mengenal obyek bila secara parsial diblok atau ditutupi oleh obyek lain.

contoh:  Komputer tidak dapat membedakan antara dua segitiga dari bentuk segiempat sehingga permasalahan mengarah pada bayangan tiga dimensi.

3.5. Sistem Tiga Dimensi

Tujuannya:

- untuk menangani secara benar semua pandangan masalah yang dibentuk oleh obyek.

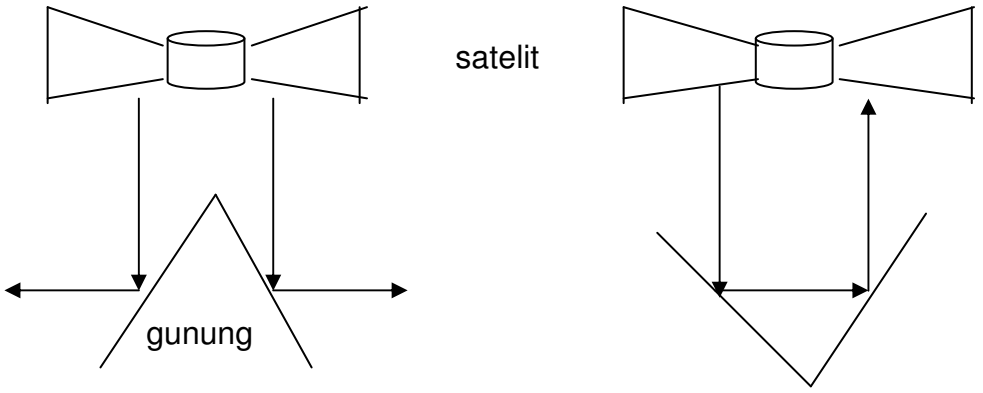
  Contoh: benda yang berada di atas atau di depan dari sebuah obyek.
- Dapat digunakan untuk menduplikasi informasi *topographical* dari sebuah bayangan.

Contoh: Program komputer dapat membentuk peta secara geografis dari suatu negara.

Sistem tiga dimensi harus dapat menangani berbagai masalah yang tidak ada dalam batasan pendekatan dua dimensi.

a. Pendeteksian arah permukaan

Bila sebuah komputer digunakan untuk menganalisa sketsa dari sebuah foto (contoh: sebuah gunung difoto oleh satelit), bagaimana komputer mengetahui bahwa yang dianalisa tersebut adalah gunung dan bukan lembah. contoh:



Lembah

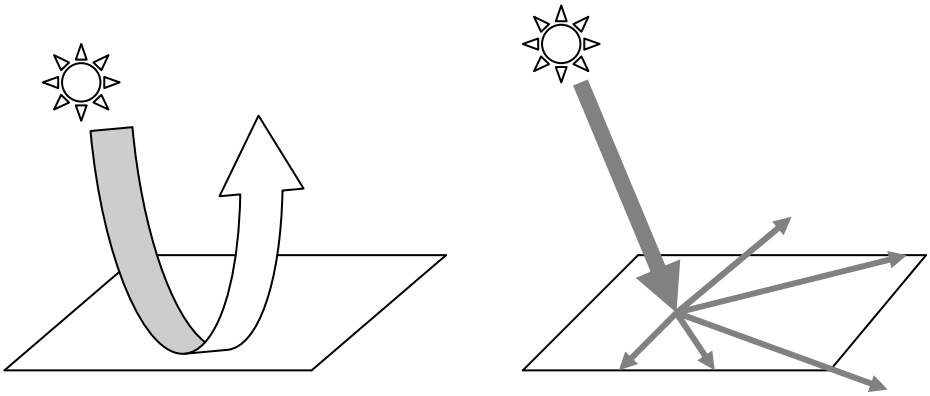
Analisa Program:

Menggunakan relatif kejelasan(*brightness*) dari permukaan untuk menentukan apakah komputer sedang memandang sebuah gunung atau sebuah lembah. Dan hal ini menjadi masalah yang kompleks walaupun dengan sebuah bayangan sederhana.

b. Penentuan Susunan Permukaan

Kita dapat menentukan susunan dari suatu obyek dengan mempelajari pemunculannya.

Contoh: refleksi sinar pada permukaan yang halus dan kasar



Permukaan halus ***Permukaan Kasar***

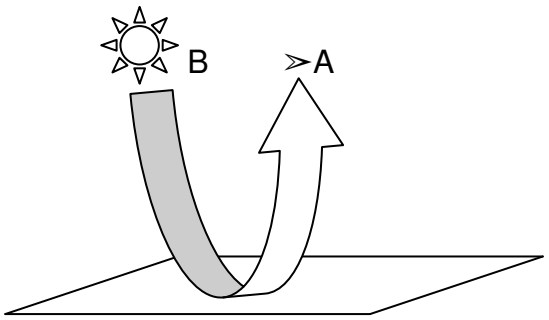
Kunci membedakan obyek yang halus dan kasar adalah dengan menginterpretasikan cara sinar berrefleksi.

Contoh: Gambar di atas bahwa karakteristik dari permukaan yang halus adalah obyek yang keras sedangkan karakteristik dari permukaan yang kasar adalah obyek yang lembut.

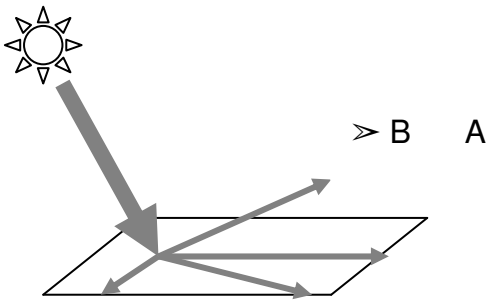
Dalam pengendalian, komputer dapat menggunakan relatif kejelasan dari setiap obyek untuk menentukan apakah permukaannya halus atau kasar.

Relatif kejelasan dari suatu obyek disebabkan oleh warnanya dan kualitas refleksi dari material yang digunakan untuk membuat obyek.

Oleh karena itu, penentuan susunan membutuhkan dua atau lebih bayangan obyek dari perbedaan titik pandang dalam relasi ke sumber sinar.



Susunan obyek kasar, bila dipandang dari titik A muncul sangat terang, karena permukaan tersebut merefleksikan hampir semua sinar yang menyentuhnya pada pemandang.



Bila dipandang dari titik B, jumlah sinar yang dipantulkan dari susunan obyek kasar sangat kecil, sementara itu jumlah sinar yang dipantulkan dari obyek

susunan halus hampir sama dengan sinar datang.

Oleh karena itu komputer dapat mengetahui perbedaan antara susunan oleh perbandingan perubahan kejelasan.

c. Pengenalan masalah-masalah umum

Kita dapat menyelesaikan maslah-masalah dengan tepat yang interpretasi bayangan oleh penggunaan sistem dua dimensi atau tiga dimensi, tetapi dalam hal ini kita masih memiliki masalah pengidentifikasian obyek atau gambar dengan tepat di mana dilakukan dengan memoles bayangan. Proses ini lebih sulit diselesaikan.

d. Pengenalan Obyek-obyek dengan klasifikasi

Masalah sulit lain adalah pemrograman komputer untuk mengenal kelas-kelas dari obyek yaitu sebuah pohon adalah sebuah pohon atau sebuah rumah adalah sebuah rumah. Jauh lebih mudah untuk membuat komputer mengenal obyek tertentu daripada membuat komputer mengenal obyek-obyek dalam klasifikasi tertentu.

Alasannya: kita dapat memberikan obyek-obyek tertentu dari sebuah himpunan yang jelas dengan batasan-batasannya di mana dapat dikenal, tetapi kita harus menyimpan definisi kelas yang sangat umum dan kehilangan untuk mencakup segala variasi-variasinya.

3.6. Overlapping Objects

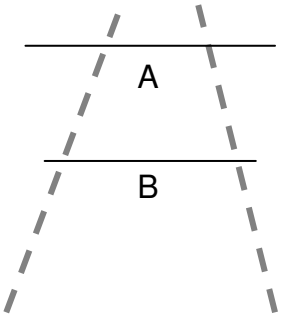
Salah satu masalah yang paling sulit adalah bila mencoba membentuk sebuah sistem bayangan dengan pengenalan *overlapping object.*

Masalahnya bukan karena komputer tidak dapat memberitahukan bahwa sebuah object di muka obyek lain. Kesulitan sesungguhnya adalah pemrograman komputer untuk mengenal secara parsial dari obyek.

Salah satu pendekatan yang dilakukan adalah mengikuti bekerjanya mata manusia yang dikenal: *Controlled Hallucination.*

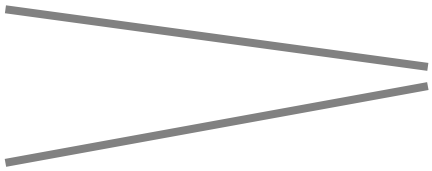
Dengan metoda ini komputer dibimbing oleh informasi awal, kemudian melakukan perhitungan.

3.7. *Optical Illusions*



Kelihatannya A lebih panjang tetapi komputer tidak membuat kesalahan yang sama.

Jalan yang semakin jauh kelihatan menyempit dan hilang pada titik temu.



Kekurang sempurnaan dari *vision-vision system*, komputer berfikir bahwa obyek-obyek disederhanakan menjadi kecil. Hal yang harus diwaspadai adalah sebuah himpunan yang berbeda dari *optical illusions* akan hadir bagi komputer dari pada untuk manusia.

BAB IV
SEARCH

Teknik-Teknik Search

Sebelum membahas tentang teknik-teknik pencarian, maka perlu dibicarakan tentang hal-hal penting yang muncul:

- Arah search
- Topologi proses search tersebut
- Memilih aturan-aturan yang dapat diterapkan
- Penggunaan fungsi-fungsi heuristik untuk untuk memandu proses search tersebut.

Arah search

Dapat dilakukan → Maju,bermula dari keadaan awal (*start state*)

Dapat dilakukan → Mundur ,diawali dari keadaan tujuan (*goal state*)

TOPOLOGI PROSES SEARCH:

Ada dua macam penggambaran problem, yaitu dalam bentuk:

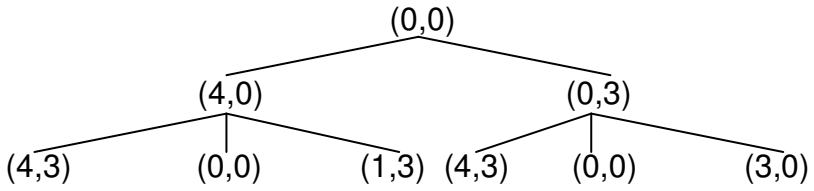
Pohon (tree)

Graf (graph)

POHON (TREE)

Merupakan graf dimana dua simbol memiliki paling banyak satu lintasan yang menghubungkannya. Tidak dimungkinkan adanya loop pada pohon.

Contoh: problem ember air (*water jug problem*)



GRAF (GRAPH)

Graf dibedakan antara:

- Graf berarah dan
- Graf tidak berarah

Graf disebut berarah bila lintasannya mempunyai arah, umumnya digambarkan dengan anak panah. Untuk garf berakar mempunyai simpul unik yang disebut akar sedemikian rupa sehingga terdapat lintasan dari akar tersebut ke semua simpul pada graf.

Ada beberapa cara untuk mencari kemungkinan penyelesaian, yaitu:

1. Depth-First Search
2. Breadth-First Search
3. Hill-Climbing Search
4. Least-Cost Search
5. Best-First Search

Evaluasi sebuah pencarian(*search*)

Evaluasi penampilan sebuah teknik pencarian (*search*) akan sangat kompleks.

Dasar pengukuran dari evaluasi:

a. seberapa cepat search menemukan penyelesaian

b. seberapa cepat search menemukan penyelesaian yang baik

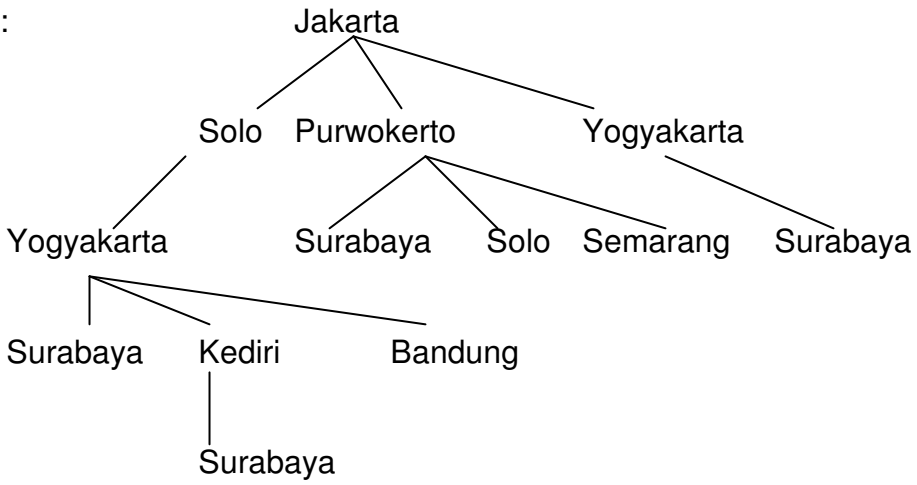
Kecepatan search ditentukan:

- panjang lintasan
- jumlah sesungguhnya penelusuran node

Kasus:

Jakarta	-	Solo	: 1000 KM
Solo	-	Yogyakarta	: 1000 KM
Jakarta	-	Purwokerto	: 800 KM
Jakarta	-	Yogyakarta	: 1900 KM
Purwokerto	-	Semarang	: 1500 KM
Purwokerto	-	Surabaya	: 1800 KM
Purwokerto	-	Solo	: 500 KM
Yogyakarta	-	Bandung	: 1000 KM
Yogyakarta	-	Kediri	: 1500 KM
Kediri	-	Surabaya	: 1500 KM
Yogyakarta	-	Surabaya	: 1000 KM

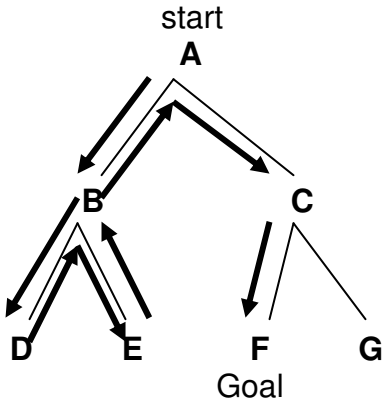
Graph:



4.1. Teknik Pencarian *Depth-First*

Penelusuran(pencarian) *Depth First* berarti setiap kemungkinan path ke goal digali(eksplorasi) ke kesimpulannya sebelum path lainnya dicoba.

Contoh: di mana F adalah goalnya.



Pencarian Depth-First akan menelusuri Graph di atas dengan susunan: ABDBEBACF

Pencarian Depth-First adalah kemungkinan metoda terbaik yang dapat diikuti di mana Heuristic tidak digunakan. Turbo Prolog menggunakan pencarian Depth-First.

Pada Graph Kasus: misalnya kita ingin pergi dari Jakarta ke Surabaya:

Percobaan 1: Jarak 3000
Jakarta - Solo – Yogyakarta - Surabaya

Percobaan 2: Jarak 5000
Jakarta - Solo - Yogyakarta – Kediri - Surabaya

Percobaan 3: Jarak 2600
Jakarta – Purwokerto - Surabaya

Evaluasi pencarian Depth First:

Percobaan 1: Penyelesaian yang baik karena tidak ada backtracking

Percobaan 2: Penyelesaian yang jelek

Percobaan 3: Penyelesaian Optimal

Contoh 8 puzzle:

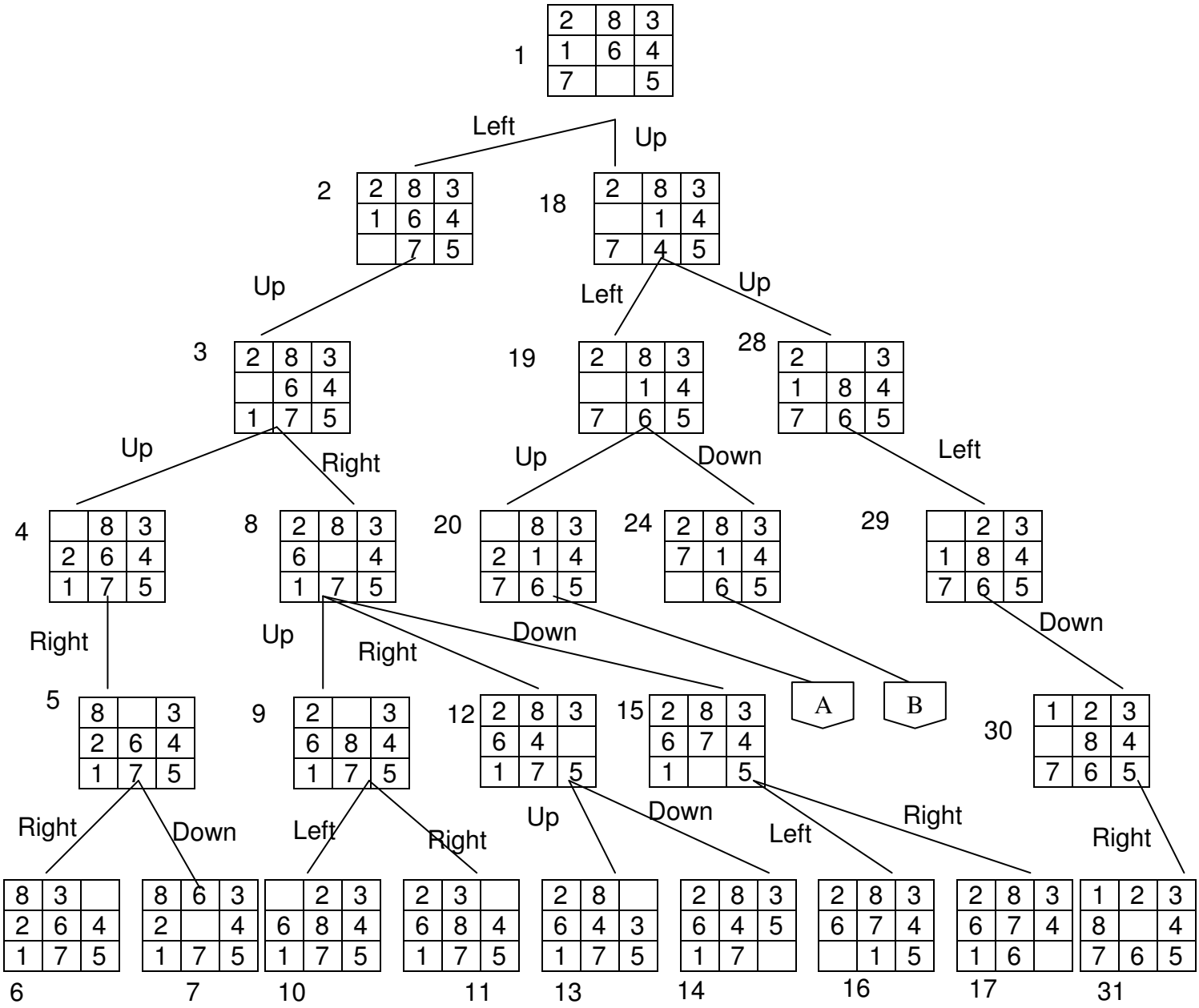
Start state:

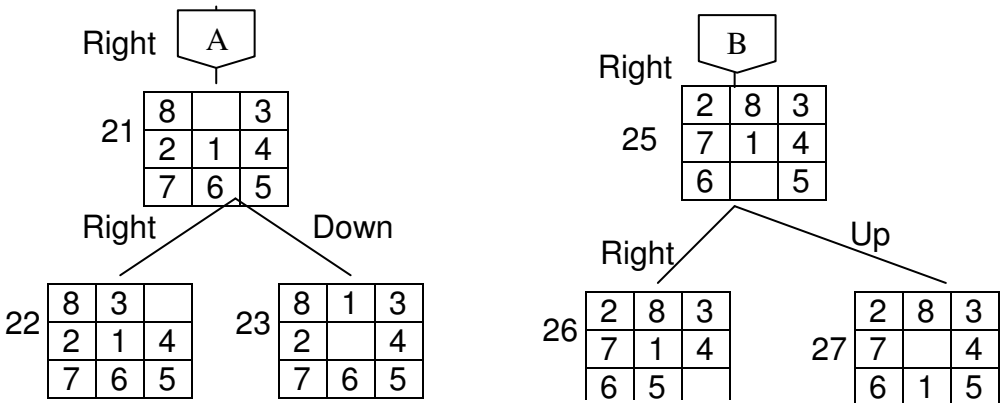
2	8	3
1	6	4
7		5

Goal State:

1	2	3
8		4
7	6	5

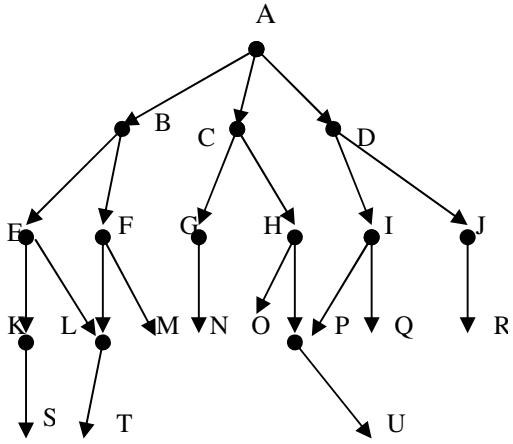
MENGGUNAKAN TEKNIK PENCARIAN DEFTH-FIRST:





PROSEDURE DEPTH-FIRST SEARCH [16)]

```
Begin
        Open :=[Start];                 {initialisasi}
        Closes:=[ ];
        While  open # [  ] do
                Begin
                        Remove  leftmost  state from, call  it  X;
                        If  X  is  a goal  then  return (succes)
                        Else  begin
                                        Generate children  of X;
                                        Put  X  in closed;
                                        Eliminate children of  X  on  open  or  closed;
                                        Put remaining  children  on  left  and  of  open
                                End
                End;
                Return (failure)
End.
```



Trace of Depth-First Search,
Asumsi goal statenya adalah U:

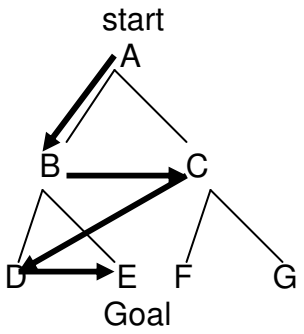
1. OPEN = [A]; CLOSED = [ ]
2. OPEN = [B,C,D]; CLOSED = [A]
3. OPEN = [E,F,C,D]; CLOSED = [B,A]
4. OPEN = [K,L,F,C,D]; CLOSED = [E,B,A]
5. OPEN = [S,L,F,C,D]; CLOSED = [K,E,B,A]
6. OPEN = [L,F,C,D]; CLOSED = [S,K,E,B,A]
7. OPEN = [T,F,C,D]; CLOSED = [L,S,K,E,B,A]
8. OPEN = [F,C,D]; CLOSED = [T,L,S,K,E,B,A]
9. OPEN = [M,C,D]; CLOSED = [F,T,L,S,K,E,B,A]
10. OPEN = [C,D]; CLOSED = [M,F,T,L,S,K,E,B,A]
11. OPEN = [G,H,D]; CLOSED = [C,M,F,T,L,S,K,E,B,A]
12. OPEN = [N,H,D]; CLOSED = [G,C,M,F,T,L,S,K,E,B,A]
13. OPEN = [H,D]; CLOSED = [N,G,C,M,F,T,L,S,K,E,B,A]
14. OPEN = [O,P,D,]; CLOSED = [H,N,G,C,M,F,T,L,S,K,E,B,A]
15. OPEN = [P,O]; CLOSED = [O,H,N,G,C,M,F,T,L,S,K,E,B,A]
16. OPEN = [U, D];CLOSED=[P,O,H,N,G,C,M,F,T,L,S,K,E,B,A]
    Success

[16)] Luger,George F. & Stubblefield,William A., "Artificial Intelligence:Structured & Strategies for Complex Problem Solving", 2nd Edition, page 96-98, Benjamin/Cumming Publishing Company, California, 1993

4.2. Teknik Pencarian Breadth-First

Merupakan kebalikan dari teknik pencarian Depth-First. Pada metoda ini diperiksa setiap node pada level yang sama sebelum mengolah ke level berikut yang lebih dalam.

Contoh: E adalah : Goalnya



node yang dilewati adalah: ABCDE

Pada Graph Kasus:

Percobaan 1:Jarak 2600

Jakarta – Purwokerto - Surabaya

Percobaan 2:Jarak 2900

Jakarta – Yogyakarta - Surabaya

Percobaan 3:arak 3000

Jakarta - Solo – Yogyakarta – Surabaya

Penambahan Heuristic

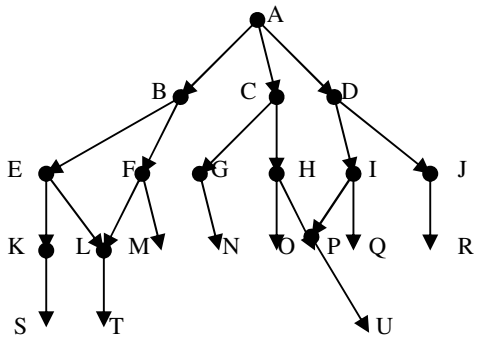
Pada metoda depth-First dan Breadth-First: Pencarian terletak pada pemindahan dari satu goal ke goal yang lain tanpa menggunakan tebakan yang terarah. Kedua metoda tersebut baik untuk situasi-situasi yang terkendali. Bila tidak maka dibutuhkan penambahan heuristic.

Dasar dari metoda pencarian heuristic:

Maximizing atau minimizing beberapa aspek dari problem(masalah)

```
Procedure breadth-first-search;[17)]
Begin
    Open: =[Start];
    Closed: =[ ];
    While  Open = [ ] do
        Begin
            Remove leftmost state from open, call it X;
            If X is a goal then return (success)
                Else Begin
                        Generate children of X;
                        Put X on closed;
                        Eliminate children of X on open or closed;
                        Put remaining children on right end of open
                     End
        End;
      Return (failure)
End.
```

Contoh:



Trace menggunakan procedure
Breadth-first-search, misalkan goal yang dicari : U

1. OPEN = [A]; CLOSED = [ ]
2. OPEN = [B,C,D]; CLOSED = [A]
3. OPEN = [C,D,E,F]; CLOSED = [B,A]
4. OPEN = [D,E,F,G,H]; CLOSED = [C,B,A]
5. OPEN = [E,F,G,H,I,J]; CLOSED = [D,C,B,A]
6. OPEN = [F,G,H,I,J,K,L]; CLOSED = [E,D,C,B,A]
7. OPEN = [G,H,I,J,K,LM]; CLOSED = [F,E,D,C,B,A]
8. OPEN = [H,I,J,K,L,M,N]; CLOSED = [G,F,E,D,C,B,A]
9. OPEN = [I,J,K,L,M,N,O,P]; CLOSED = [H,G,F,E,D,C.B,A]
10. OPEN = [J,K,L,M,N,O,P,Q]; CLOSED = [I,H,G,F,E,D,C,B,A]
11. OPEN = [K,L,M,O,P,Q,R]; CLOSED = [J,I,H,G,F,E,D,C,B,A]
12. OPEN = [L,M,N,O,P,Q,R,S]; CLOSED = [K,J,I,H,G,F,E,D,C,B,A]
13. OPEN = [M,N,O,P,Q,R,S,T]; CLOSED = [L,K,J,I,H,G,F,E,D,C,B,A]
14. OPEN = [N,O,P,Q,R,S,T]; CLOSED = [M,L,K,J,I,H,G,F,E,D,C,B,A]
15. OPEN = [O,P,Q,R,S,T]; CLOSED = [N,M,L,K,J,I,H,G,F,E,D,C,B,A]
16. OPEN = [P,Q,R,S,T]; CLOSED = [N,M,L,K,J,I,H,G,F,E,D,C,B,A]
17. OPEN = [Q,R,S,T,U]; CLOSED = [P,N,M,L,K,J,I,H,G,F,E,D,C,B,A]
18. OPEN = [R,S,T,U]; CLOSED = [Q,P,N,M,L,K,J,I,H,G,F,E,D,C,B,A]
19. OPEN = [S,T,U]; CLOSED = [R,Q,P,N,M,L,K,J,I,H,G,F,E,D,C,B,A]
20. OPEN = [T,U]; CLOSED = [S,R,Q,P,N,M,L,K,J,I,H,G,F,E,D,C,B,A]
21. OPEN = [U]; CLOSED = [T,S,R,Q,P,N,M,L,K,J,I,H,G,F,E,D,C,B,A]

Success

[17)] IBID [16)] , page 92-95.

HEURISTIK

- Dari kata Yunani : *heuriskein* artinya "***to discover***" = menemukan
- adalah : sebuah teknik yang memperbaiki hasil efisiensi dari sebuah proses penelusuran / pencarian, kemungkinan dengan klaim-klaim korban dari kesempurnaan.

Keuntungan dari Depth first search :

- *depth-first search* membutuhkan sedikit memory, karena hanya node-node pada path aktif (*current*) yang disimpan.Ini sangat berbeda dengan *breadth-first search*, dimana semua part dari tree sepanjang telah dihasilkan harus disimpan.
- Secara kebetulan (atau jika penanganan diambil dalam urutan state pengganti alternatif). *Depth first search* dapat menemukan sebuah solusi tanpa uji coba beberapa penelusuran tempat kosong pada keseluruhannya.Ini berbeda dengan *breadth-first search*, semua bagian dari tree harus di ujicoba pada level n sebelum beberapa node pada level n+1 dapat di ujicoba.

  Ini secara khusus pasti jika beberapa solusi dapat diterima. *Depth-first-search* dapat berhenti bila salah satu dari mereka ditentukan.

Fungsi/***Function*** **Heuristik** adalah : Fungsi yang melakukan pemetaan (*mapping*) dari diskripsi keadaan masalah (*problema*) ke pengukur kebutuhan, umumnya direpresentasikan berupa angka.

Sangat penting dalam *Expert system* sebagai komponen esensial dalam memecahkan persoalan.

George Polya mendefinisikan heuristik sebagai studi metode & aturan penemuan.

Dalam proses search ruang keadaan (*state space*), heuristik dinyatakan sebagai aturan untuk melakukan pemilihan cabang dalam ruang keadaan yang paling dapat diharapkan mencapai pemecahan masalah yang dapat diterima.

AI menggunakan heuristik dalam dua situasi dasar :

a. Persoalan/problema yang mungkin memiliki solusi eksak, namun biaya perhitungan untuk menemukan solusi tersebut sangat tinggi dalam kebanyakan persoalan (seperti catur), ruang keadaan bertambah secara luar biasa seiring dengan jumlah.
b. Persoalan yang mungkin tidak memiliki solusi eksak karena ambiquitas (keraguan atau ketidakpastian) mendasar dalam penyataan persoalan atau data yang tersedia, Diapnosa medis merupakan salah satu contohnya.

Heuristik menangani kerumitan masalah dengan cara memadu proses search pada sepanjang lintas yang paling dapat diharapkan. Namun heuristik juga bisa salah. Oleh karena itu heuristik hanyalah sebuah cara menerka langkah berikutnya yang harus diambil dalam memecahkan suatu persoalan berdasarkan informasi yang ada/tersedia.

4.3. Teknik Pencarian *Hill Climbing*

Pada tabel jarak dari Jakarta ke Surabaya terdapat 2 batasan kemungkinan di mana seseorang ingin meminimkan hal:

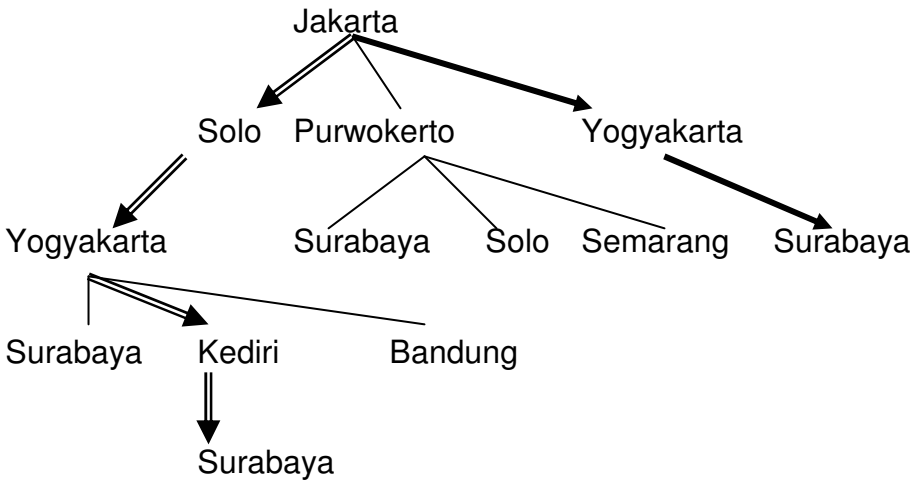
1. Jumlah hubungan(*connection*) yang harus dibuat
2. Panjang *Route*

(*Route* terpendek tidak berarti hubungan yang paling sedikit)

Algoritma pencarian mencoba untuk menemukan solusi pertama yang meminimkan jumlah hubungan dengan menggunakan informasi heuristic. Dalam AI metoda pencarian yang menggunakan informasi heuristic disebut: *Hill Climbing.*

.

Pada umumnya algoritma *Hill Climbing* memilih langkah berikut node yang berada ditempat yang terdekat dengan goal.



Percobaan pertama : 2900 Jakarta – Yogyakarta - Surabaya

Percobaan kedua : 4900 Jakarta - Yogyakarta- Kediri - Surabaya

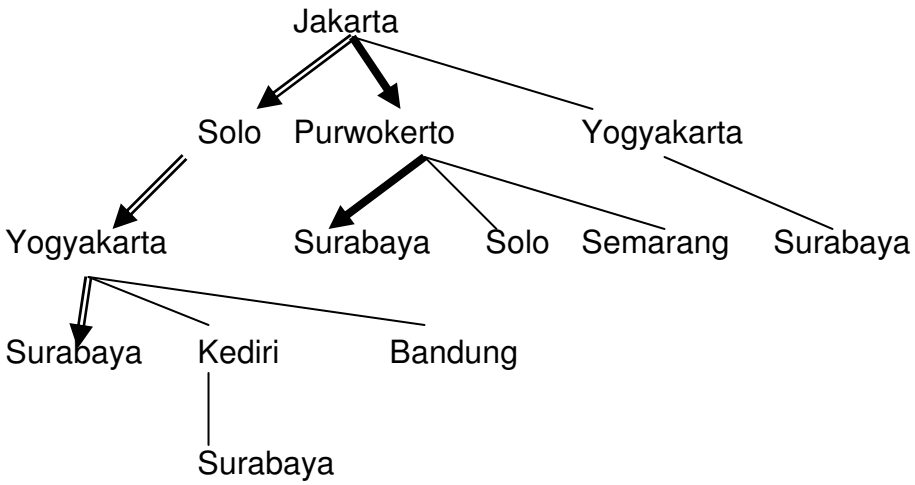
4.4. Teknik Pencarian *Least Cost*

Kebalikan dari pencarian *Hill Climbing* adalah pencarian *Least Cost.*

Pada metoda ini : hubungan terpendek akan diambil sehingga route yang ditemukan mempunyai kemungkinan yang baik untuk mendapat jarak terpendek.

Jadi *Hill Climbing* : meminimkan jumlah hubungan

Least Cost : meminimkan jumlah jarak yang ditelusuri



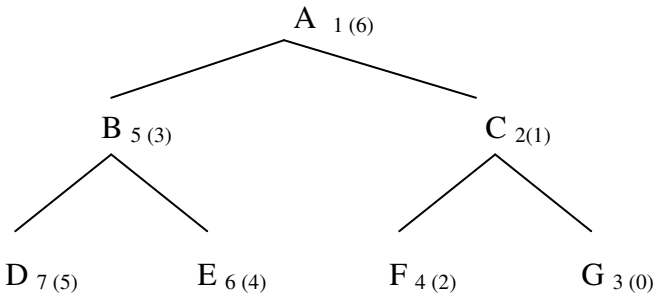
Percobaan pertama : 2600 Jakarta – Purwokerto - Surabaya

Percobaan kedua : 4900 Jakarta - Solo – Yogyakarta - Surabaya

4.5. *Best First Search*

Algoritma best first search menurut Thomas Dean, et all, AI teori & practice, page 145, sebagai berikut:

1. Himpun N menjadi sebuah list berurutan dari node awal(*initial nodes*)
2. Jika N adalah kosong, maka keluar dan berikan pesan gagal/*failure*
3. Himpun n menjadi node pertama dalam N, dan hapus n dari N
4. Jika n adalah node tujuan/*goal* maka keluar dan berikan pesan sukses
5. Selain itu, tambahkan anak dari n ke N, urutkan node-node dalam N menurut jarak estimasi dari sebuah goal, dan kembali ke langkah 2.



Implementasi best-first search dalam fungsi LISP (*LISP FUNCTION*):

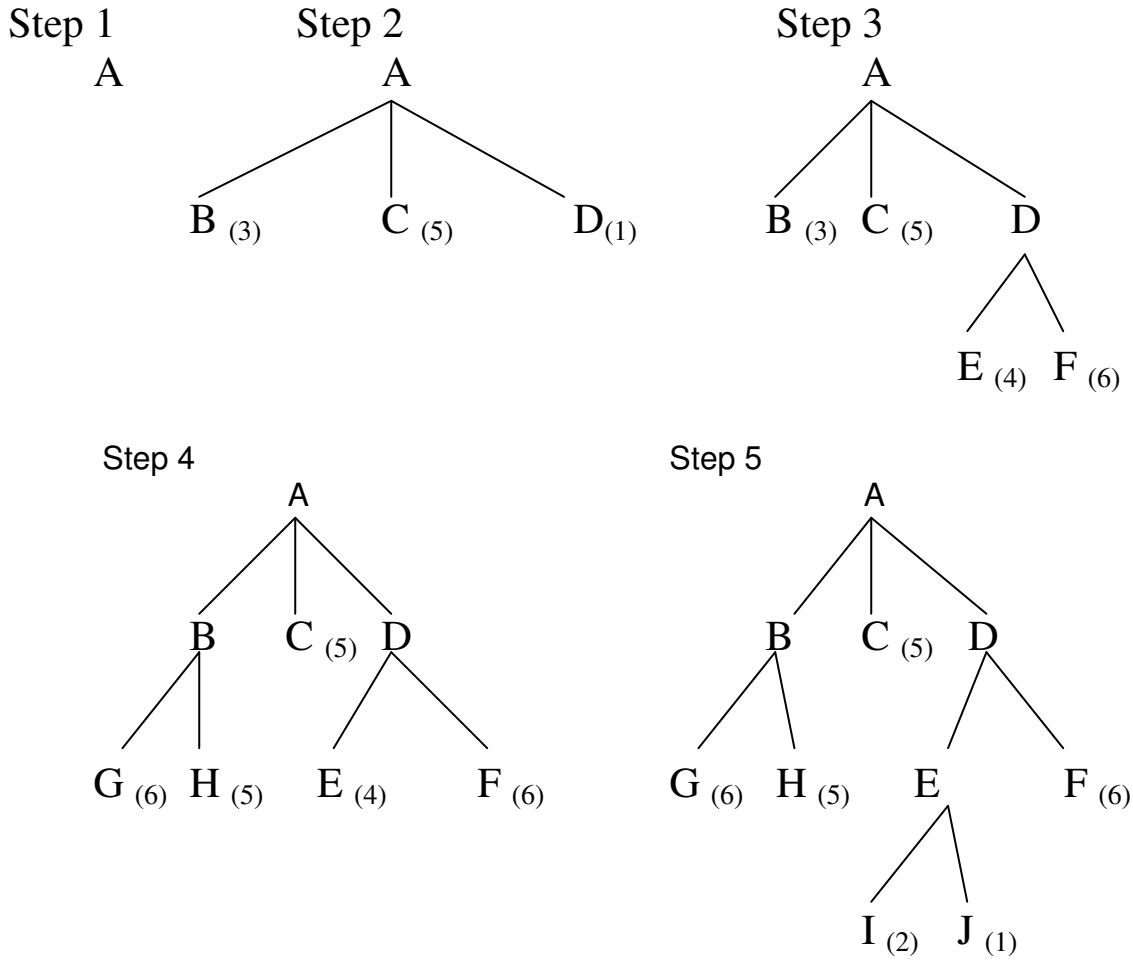
```
(defun best (nodes goalp next comparep)
        (cond ((null nodes) nil)
                ;; return the first node if it is a goal node
                ((funcall goalp (first nodes)) (first nodes))
                ;; append the children to the set of old nodes
                ;; and then sort them all according to value.
                (t ( best (sort (append (funcall next (first nodes))
                                        (rest nodes))
                            comparep)
                        goalp
                        next
                        comparep))))

(defun TREE-comparator (tree1 tree2)
        (< (TREE-value tree1) (TREE-value tree2)))
```

setelah didemontrasikan dengan goals sebagai berikut:

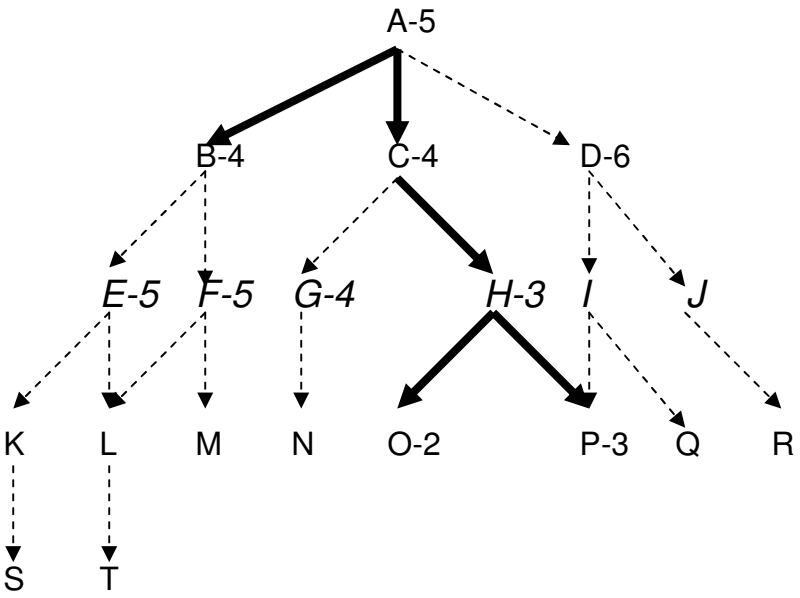
```
> (best (list tree)
        #'(lamda (x) (TREE-print x) nil)
        #'TREE-children
        #'TREE-comparator)
ACGFBED
NIL
```

Algoritma *Best First Search* menurut Elaine Rich & Kevin Knight, " Artificial Intelligence", page 73, sebagai berikut:

1. Start with OPEN containing just the initial state
2. Until a good is found or there are no nodes left on OPEN do:
   (a) Pick the best node on OPEN
   (b) Generate its successors
   (c) For each successor do :
      i. if it has not been generated before, evaluate it, add it to OPEN, and record its parent.
      ii. If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.



Algoritma *Best First Search* menurut George F. Luger, & William A.Stubblefield, "*Artificial Intelligence:Structured & Strategies for Complex Problem Solving*", 2nd Edition, page 120-123, sebagai berikut:



1. OPEN = [A-5]; CLOSED = [ ]
2. Evaluate A-5; OPEN = [B-4, C-4, D-6];
   CLOSED = [A-5]
3. Evaluate B-4; OPEN = [C-4, E-5, F-5, D-6];
   CLOSED = [B-4, A-5]
4. Evaluate C-4; OPEN = [H-3, G-4, E-5, F-5, D-6];
   CLOSED = [C-4, B-4, A-5]
5. Evaluate H-3; OPEN = [O-2, P-3, G-4, E-5, F-5, D-6];
   CLOSED =[H-3,C-4,B-4,A-5]
6. Evaluate O-2; OPEN = [P-3, G-4, E-5, F-5, D-6];
   CLOSED = [O-2,H-3,C-4,B-4,A-5]

Evaluate P-3; The Solution is found!

Kesimpulan : *best first search* merupakan suatu teknik pencarian yang mengkombinasikan yang terbaik diperoleh dari teknik *depth-first search* dan teknik *breadth-first search* ke dalam sebuah metode tunggal.

4.6. Teknik Pencarian Minimax

Prosedur Pencarian Minimax adalah langkah depth-first, idenya adalah memulai pada saat sebagian posisi dan menggunakan *plausible-move generator*

untuk menggerakkan himpunan pada posisi *possible successor*. Sekarang kita dapat mengaplikasikan fungsi *evaluasi statis* ke posisinya dan memilih salah satunya yang terbaik.

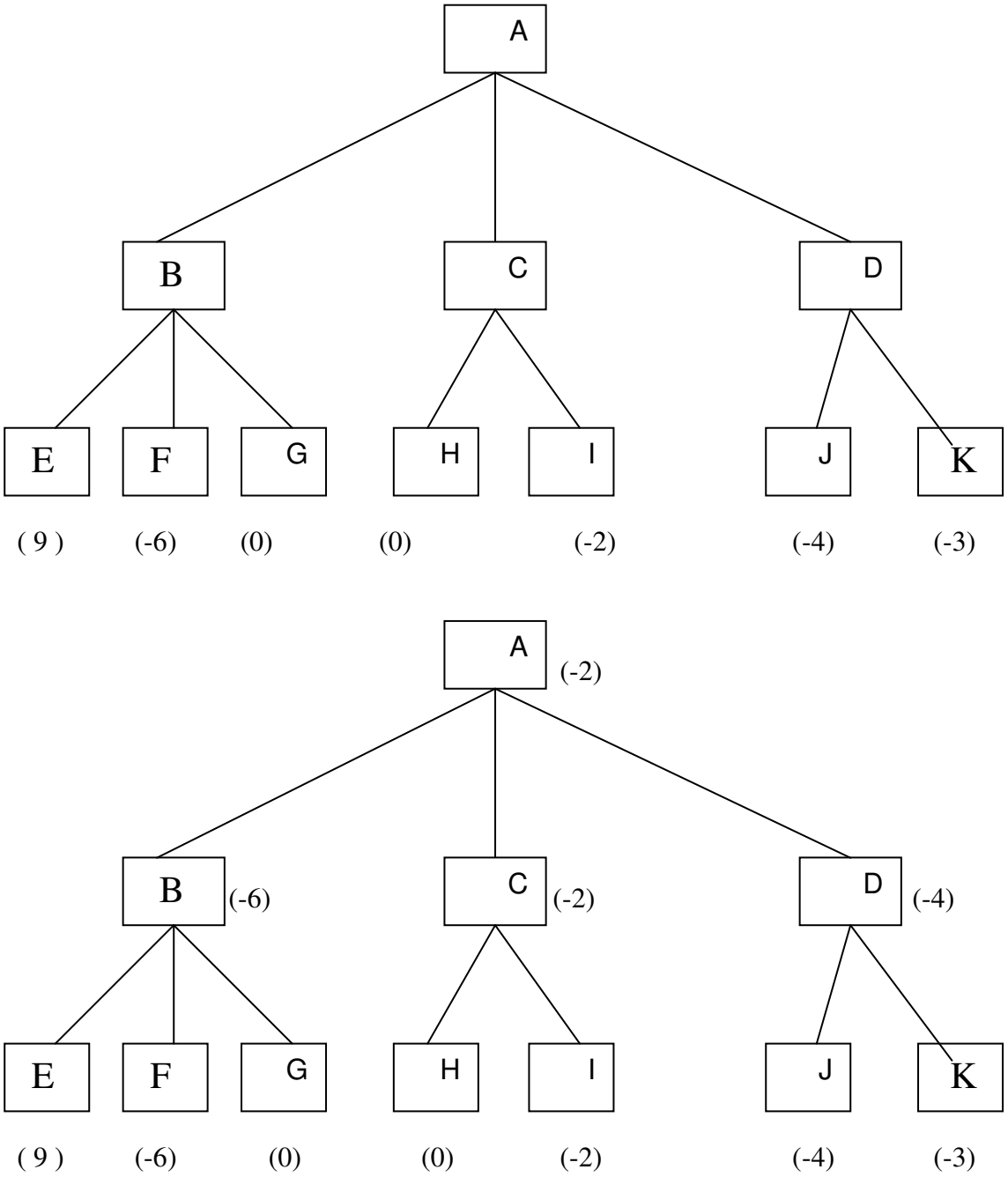
Setelah melakukannya, kita dapat mengembalikan nilai pada permulaan posisi untuk memperoleh hasil evaluasi. Permulaian posisi sangat tepat sekali untuk dilakukan oleh kita sebagai posisi yang digerakkan oleh langkah kita yang terbaik untuk selanjutnya. Disini kita mengamsumsikan bahwa fungsi evaluasi statis akan kembali ke nilai yang lebih besar yang mengindentifikasikan kita untuk situasi yang tepat, lalu tujuan akhir(goal) adalah memaksimalkan nilai pada fungsi evaluasi statis pada posisi berikutnya.

Contohnya dapat kita lihat pada gambar 4.1. pada gambar tersebut diamsumsikan bahwa fungsi evaluasi statis akan kembali pada nilai yang ditunjukkan pada –10 to 10, dengan 10 mengindentifikasikan kemenangan untuk kita, -10 menang untuk opponent, dan 0 untuk pertandingan berikutnya. Sejak tujuan akhir kita (goal) adalah untuk memaksimalkan nilai pada fungsi heuristic, kita memilih untuk mengerakkannya ke B, simpan nilai B ke A, kita dapat menyatakan bahwa nilai A adalah 8, sejak kita mengetahuinya kita dapat mengerakkannya ke posisi yang bernilai 8.

Tetapi sejak kita mengetahui bahwa fungsi evaluasi statis tidaklah lengkap, kita memutuskan untuk membawanya ke tempat yang lebih jauh di banding satu permainan. Ini penting misalnya pada permainan catur dimana kita berada ditengah diantara benteng.

Sesudah kita bergerak, situasinya haruslah baik, tetapi jika kita melihat satu pergerakkan raja, kita dapat melihat bahwa salah satu benteng kita akan di makan dan situasi ini tidaklah baik untuk dilihat, lalu kita dapat melihat raja lihatlah apa yang terjadi pada langkah berikutnya yang dapt dilihat oleh opponent-nya.

Paling sedikit pada pengaplikasian fungsi evaluasi statis untuk beberapa posisi yang kita kembangkan, kita dapat mengaplikasikan *plausible-move generator*, pergerakkan posisi successor untuk beberapa posisi.



Gambar 4.1.Memback Up Nilai Pada Two-Ply Search

Tetapi kita harus dapat membawa ke dalam perhitungan berarti *opponent* dapat kita pilih berdasarkan *successor* yang bergerak untuk membuat dan demikian yang mana nilai terminal seharusnya di *backup* untuk level berikutnya.

Berdasarkan yang kita buat pada langkah B. lalu *opponen*t harus memilih diantara langkah E,F dan G . pada *goal opponent* adalah meminimalkan nilai pada fungsi evaluasi, lalu dia dapat memilih untuk meggerakkan F. ini berarti jika kita

menggerakkan B, posisi aktualnya memungkinkan suatu konfigurasi yang dipersembahkan oleh E, yang sangat baik untuk kita .

Tetapi sejak kita berada pada level tersebut kita tidak dapat melakukan satu langkah, kita tidak akan melakukan pilihan tersebut. Pada gambar 4.1. kita dapat melihat hasil propaganda pada nilai yang baru pada sebuah tree. Pada level ini mempersembahkan pilihan opponent, nilai minimum dapat dipilih dan di back up. Pada level ini kita mempersembahkan pilihan kita, nilai maksimum yang kita pilih.

Pada nilai yang pertama dari permainan kedua di backup, ini akan menjadi langkah yang tepat untuk kita yang kita buat pada level pertama, memberikan informasi yang memungkinkan adalah C. Sejak tidak terdapat opponent maka tidak dapat dilakukan dari sini untuk memproduksi nilai salah lebih dari –2. Proses ini dapat diulang untuk beberapa waktu, dan lebih akurat evaluasinya, sehingga hasilnya dapat digunakan untuk langkah berikutnya pada level teratas.

Alternatif pada maximizing dan minimizing pada alternatif permaianan, ketika evaluasi ditekan oleh korespodennya untuk strategi mengopponent pada dua pemain dan memberikan pengertian nama minimax.

Memiliki informasi yang menjelaskan pengoperasian prosedure minimax, kita sekarang menggambarkan secara lengkap. Ini merupakan langkah tepat pada prosedur rekursif bahwa ada dua alasan yang menjelaskan secara spesifik mengenai permainan yang dimainkan :

1. MOVEGEN (position, Player)

   Plausible-move generator, yang mengembalikan node yang mempersembahkan langkah yang dapat dibuat oaleh pemain dalam suatu posisi.

   Kita menyebutnya sebagai dua pemain yaitu pemain 1 dan pemain 2; pada program catur, kita seharusnya menggunakan nama HITAM dan PUTIH.

2. STATIC (position, player)

   Fungsi evaluasi statis, yang mengembalikan angka yang mempersembahkan kebaikan pada suatu posisi dari suatu penilaian oleh pemain.

Dengan beberapa program rekursif, isu kritikal dalam suatu design prosedur minimax dapat diberhentikan pada pengrekursiannya dan dapat dipanggil dengan mudah oleh fungsi evaluasi statis. Dapat dinilai factor varietasnya yang menyebabkan suatu keputusan. Hal tersebut ialah :

- Apakah satu sisi menang ?
- Berapa banyak ply yang dimiliki kita siap dieksplorasi ?
- Apakah path menjanjikan ?
- Berapa waktu yang diperlukan ?
- Apakah dapat menjamin konfigurasinya ?

Untuk pusat dari prosedur minimax dibahas disini , kita dapat menggunakan DEEP-ENOUGH, yang mengasumsikan untuk megevaluasi semua faktor dan memberikan TRUE jika pencarian dapt dihentikan pada sebagian level dan sebaliknya FALSE.

Kita menggunakan sangat sederhana pada DEEP-ENOUGH yang dapat mengambil dua parameters, posisi dan Depth. Ini dapat menyebabkan ketidakperdulian pada posisi parameters dalam pemberian nilai TRUE, jika pada DEPTH parameter memotong nilai yang konstan.

Suatu permasalahan yang menjelaskan minimax sebai prosedur rekursif yang dibutuhkan untuk tidak mengembalikan salah satunya tetapi terdapat dua hasil :

- Pemback up-an nilai pada path yang dipilih.
- Path tersebut. Kita mengembalikannya kedalam bagian path yang memungkinkan hanya sebagian elemen, yang mempersembahkan langka yan g baik dari sebagian posisi, yang sangat dibutuhkan.

Kita mengasumsikannya bahwa pengembalian struktur minimax menghubungkan hasil keduanya dan terdapat dua fungsi yaitu VALUE dan PATH, yang mengekstrak sebagian komponen.

Sejak kita menemukan struktur prosedur minimax sebagai fungsi rekursif, kita harus dapat mejelaskan secara spesifik bagaimana ia dapat di panggil inisialnya. Ini membutuhkan tree parameter, pada board posisi, pada sebagian

DEPTH pada suatu pencarian dan pemain untuk menjalankannya. Maka penginisialannya dapat dipanggil untuk menghitung langkah yang terbaik pada posisi CURRENT seharusnya.

MINIMAX(CURRENT, 0, PLAYER-ONE)
IF PLAYER – ONE IS TO MOVE, OR
MINIMAX(CURRENT, 0, PLAYER-TWO)
IF PLAYER – TWO IS TO MOVE, OR

ALGORITMA : MINIMAX (POSITION, DEPTH, PLAYER)

1. IF DEEP-ENOUGH(POSITION, DEPTH) THEN RETURN THE STRUCTURE
   VALUE = STATIC (POSITION, PALYER)
   PATH = NIL
   Ini mengindentifikasikan bahwa tidak terdapat path dari node ini dan bahwa nilai dideterminasikan oleh fungsi evaluasi statis.
2. Sebaliknya, satu perkembangan ply lagi pada suatu tree oleh pemanggilan fungsi MOVEGEN(Posistion, Player) dan pengesetan SUCCESSORS pada suatu list.
3. jika SUCCESSORS kosong, maka tidak pergerakan yang dibuat, lalu kembali ke struktur yang sama bahwa akan terjadi perubahan yang dikembalikan jika DEEP-ENOUGH telah dinyatakan benar.
4. Jika SUCCESSOR tidak kosong, maka pelaksanaan beberapa elemen yang turun dan menjaga agar track tersebut adalah yang terbaik. Sesuai dengan yang mengikutinya.

Penginisialisaian skor terbaik untuk nilai minimum bahwa STATIC dapat dikembalikan. Ini dapat menjadi reflek dalam mengupdate nilai terbaik sehingga dapat mencapai suatu elemen pada SUCCESSORS.

Untuk beberapa elemen SUCC pada SUCCESSOR, dapat mengikuti :

- SET RESULT-SUCC TO
  MINIMAX(SUCC,DEPTH+1,OPPOSITE(PLAYER))[3]
  Pada rekursif memanggil minimax dapt secara aktual membawa pengeksplorasian pada SUCC.
- SET NEW-VALUE TO-VALUE(RESULT-SUCC).
  Ini dapt menimbulkan reflek dari kebaikan pada suatu posisi dari tempat yang prepekstif pada level terbawah berikutnya.
- IF NEW-VALUE > BEST-SCORE, maka dapat kita temukan pada successor bahwa lebih baik daripada beberapa hal yang kita pelajari selama ini. Pada Record ini dapat dilakukan dengan mengikuti :
- SET BEST-SCORE TO NEW-VALUE
  Yang terbaik yang perlu kita ketahui dari path ini adalah dari CURRENT ke SUCC dan maka untuk path ke bawah dengan tepat dari SUCC sebagai determinasi dari pemanggilan rekursif untuk minimax. Lalu set BEST-PATH untuk hasil pada lampiran SUCC ke depan pada PATH(RESULT-SUCC).

5. Sekarang semua Successor telah dipelajari, kita tahu nilai pada posisi baik seperti yang dapat diambil pada path tersebut. Lalu kembalikan ke struktur :

   VALUE = BEST-SCORE

   PATH = BEST-PATH

Ketika penginialisaian pemanggilan pada minimax , langkah yang terbaik dari CURRENT adalah elemen pertama pada suatu path. Untuk melihatnya dapat di lihat pada saat prosedur ini bekerja , anda seharusnya dapat mentracenya pada suatu eksekusu pada permainan game tree yang diperlihatkan pada gambar 4.1

Prosedur minimax dapat dijabarkan kedalam bentuk yang sederhana. Tetapi performancenya dapat di buktikan secara significant dengan beberapa perbaikan. Beberapa diantaranya menjelaskan langkah beberapa seksi.

BAB V
PEMROSESAN BAHASA ALAMI
(NATURAL LANGUAGE PROCESSING/NLP)

5.1. Pengenalan NLP

NLP adalah mencoba untuk membuat komputer dapat mengerti perintah-perintah yang ditulis dalam standar bahasa manusia.

NLP tidak memperdulikan bagaimana sebuah kalimat dimasukkan ke komputer tetapi mencopy informasi dari kalimat tersebut.

a. Pendekatan-pendekatan pada NLP: Inti dari NLP adalah *PARSER*

Dimana PARSER tersebut membaca setiap kalimat, kata demi kata, untuk menentukan apa yang dimaksud.

PARSER terdiri dari 3 jenis:

1. *PARSER STATE-MACHINE*
2. *PARSER CONTEXT-FREE RECURSIVE-DESCENT*
3. *PARSER NOISE-DISPOSAL*

Hal hal yang bertentangan dengan pendekatan NLP -:

1. Bahwa sesungguhnya NLP menggunakan semua informasi dalam sebuah kalimat, hanya manusia yang dapat melakukan hal tersebut.
2. Mencoba memperbolehkan komputer menerima perintah bahasa alami, tetapi hanya mengcopy inti informasi pada perintah (*command*).

b. Batasan Bahasa

Aspek yang paling sulit dalam pembentukan sistem pengendali NLP adalah: Pengakomodasian kekompleksan dan kefleksibelan bahasa manusia dalam sistem.

Contoh Bentuk standard : **Subyek-Verb-Obyek**

Diasumsikan:

- *Adjective* mengawali *Noun*
- *Adverb* mengawali *Verb*
- Semua kalimat diakhiri dengan titik

Contoh : *The child runs to the house*

The large child runs quickly to the window

PARSER akan menentukan:

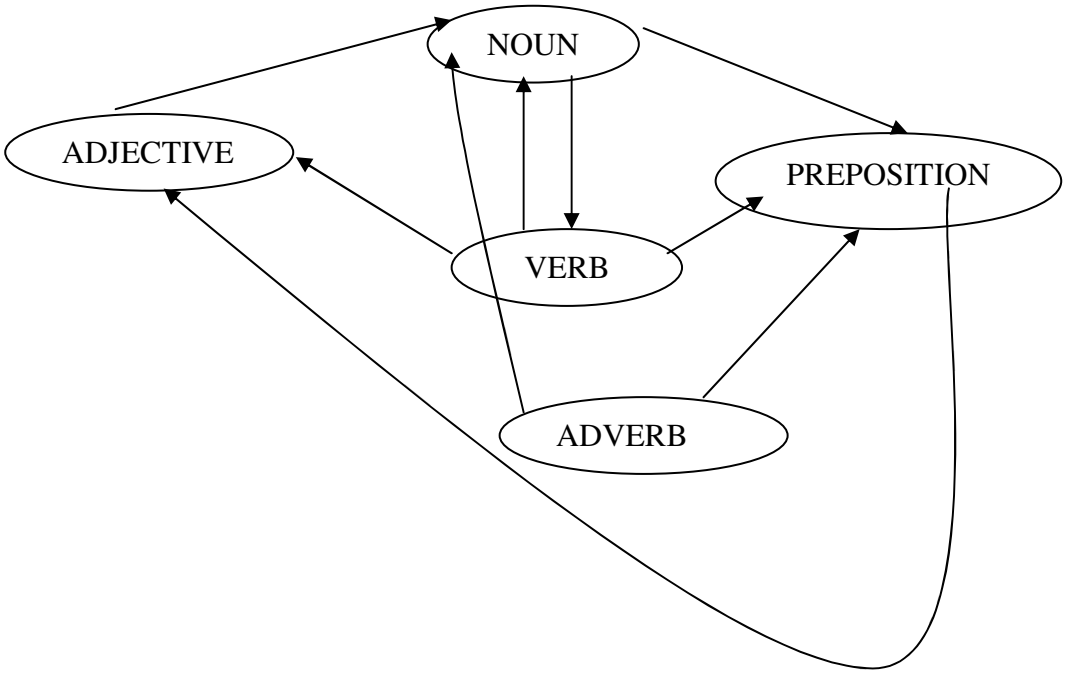
The child quickly runs to the house

5.2. Parser state-machine NLP

Parser state-machine menggunakan keadaan yang sesungguhnya dari kalimat untuk memprediksi tipe apa dari kata yang berlaku.

State-machine : *directed graph* yang menunjukkan transisi yang valid dari satu state ke yang lain.

Contoh: Grammar G1



Gambar 5.1. *State Machine Grammar* G1

Kegunaan *state-machine* :

a) dapat memotong atau memilah kalimat ke dalam komponen-komponennya.
b) menentukan apakah sebuah kalimat dibentuk dengan benar dalam batasan dari *grammar* G1

Database yang harus dibentuk sebelum implementasi:

i) Membentuk kosa kata(*vocabulary*) dari kata-kata yang dikenal ke sistem dengan mengikuti tipe yang ada.
ii) Menyimpan keadaan sesungguhnya dari kalimat.

Masalah yang paling buruk dengan *state-machine parser* adalah:

1. Kekompleksannya
   Contoh : Dalam grammar G1 dibutuhkan 14 clause yang terpisah untuk menunjukkan transisi keadaan.
2. Parser tidak mengetahui bagaimana mencapai suatu keadaan
   Contoh: Parser tidak dapat menghubungkan sebuah modifikasi phrase ke noun tertentu. Hal ini berarti tidak dapat memanggil parser state-machine untuk mendukung suatu informasi lain dari keadaan yang sesungguhnya.

Keuntungan *parser state-machine* : Ideal untuk aplikasi tertentu seperti: beberapa aplikasi database

5.3. Parser *Context-Free Recursive-Descent*

Contoh: sebuah kalimat adalah gabungan dari berbagai item dan item ini adalah gabungan dari item lain dan seterusnya sampai dipotong(dipilah) ke elemen-elemennya seperti Noun, Adjective, dan sebagainya.

Aturan-aturan yang ada pada setiap bagian yang telah dibentuk disebut: *Production rule* dari *grammar*.

Parser context-free menggunakan *production rule* untuk menganalisa sebuah kalimat.

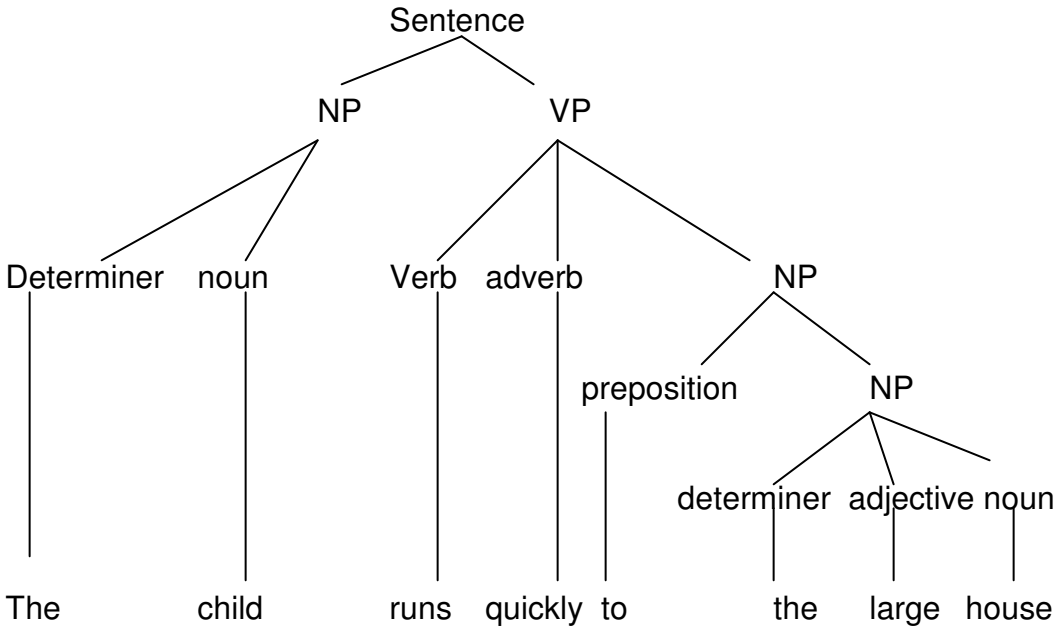
Production Rule untuk grammar G1:

Sentence	--> NP + VP
NP	--> determiner + noun
NP	--> determiner + adjective + noun
NP	--> preposition + NP
VP	--> Verb + NP
VP	--> Verb + adverb + NP
VP	--> Verb + adverb
VP	--> adverb

NP : Noun Phrase
VP : Verb Phrase

Noun Phrase: Definisi rekursif untuk proposisi clause

Verb Phrase: Rekursif tidak langsung karena *Verb Phrase* melibatkan *Noun Phrase* sebagai bagian dari definisinya.



Parser membentuk tipe *parse tree* yang disebut: '***context free***', sebab Tree bukan dasar dari konteks setiap elemen. Hal ini berarti bahwa aturan atau

rule akan bekerja untuk suatu *statement* yang menyerupai *grammar* G1 tanpa mengharapkan pada konteks setiap phrase.

Kegunaan *context free* selain untuk program-program AI NLP, juga untuk bahasa-bahasa komputer lainnya seperti:

- *PASCAL*
- *BASIC*
- *C*
- *MODULA-2*

Parser recursive-descent menggunakan:kumpulan rutin rekursif di mana menurunkannya melalui *production rule* sampai kalimat selesai ditelusuri seluruhnya.

Untuk membentuk *parser context-free recursive-descent* dibutuhkan beberapa *vocalbulary database* dan dukungan *predicate* untuk menyalin kata-kata dari sebuah kalimat sebagai *parser state-machine* yang digunakan.

Keuntungan *Parser Context-free recursive-descent*:

(1) Mudah diimplementasikan dalam turbo prolog.
(2) Dapat berkomunikasi dengan kalimat baik tingkatan kata dan phrase.
(3) Mengetahui di mana parser dalam kalimat pada setiap saat.

Kerugiannya: Tidak dapat menangani cara valid dalam jumlah besar di mana kalimat dalam suatu bahasa dibentuk.

5.4. Parser Noise-Disposal

Tipe parser ini sesungguhnya sangat umum dalam aplikasi tipe database, seperti : *Command processor*.

Contoh: sebuah database terdiri dari nama-nama perusahaan dan harga-harga stock.

Asumsi database menerima query seperti:

Lihatkan saya semua perusahaan dengan persediaan > 100

Lihatkan saya semua

Lihatkan saya xyz

Lihatkan saya satu dengan persediaan < 100

Tipe query: **Perintah<modifikasi><nama><operator><nilai>**

Perintah harus selalu ada, tetapi 4 elemen lainnya adalah optional. Namun demikian bila operator digunakan maka nilai harus digunakan.

Kerugian: tidak berguna untuk situasi tidak terbatas, karena didasari pada dua asumsi:

i. Kalimat mengikuti bentuk yang tegas
ii. Hanya beberapa *keyword* yang penting.

Keuntungan: sederhana untuk diimplementasikan dan mendapatkan informasi yang cepat.

5.5. *Natural Language Understanding*

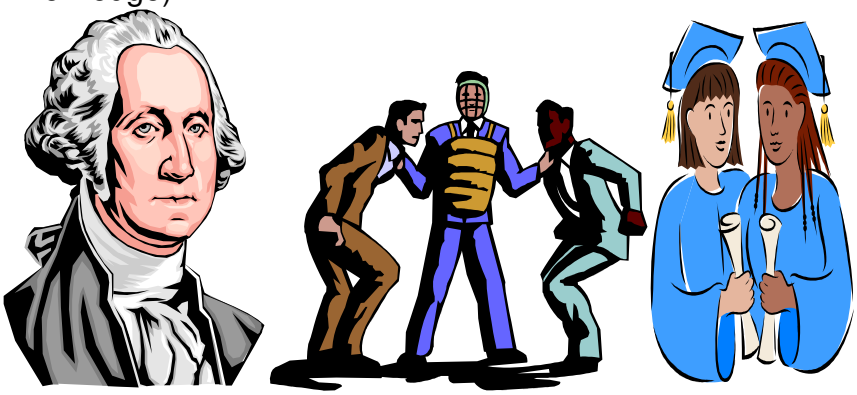
- Chomsky (*Psychologis*t) --> *Formal Language*
  S --> aSa

- *Machine Translation*
  English → Russian ≠ Russian → English

- Bagaimana struktur *grammar* daripada *sentence*?
  Bagaimana bahasa di organisis untuk mendapatkan artinya?

Conceptual Structure (Pengkodean Knowledge)

Conceptual Structure



string

1. object
2. relations
3. events

- *Object* : nama atau deskripsi atau *argument*

  :John :Black :Chair

- *Relation*: penghubung argument dan diatur

  (# loves :John :Mary)

  *argumen*t(object atau deskripsi)

  relation

contoh lain: 'I want you to pick up the black'

relation

(# want :I :Rel1)
(# pick up :you :Rel1)

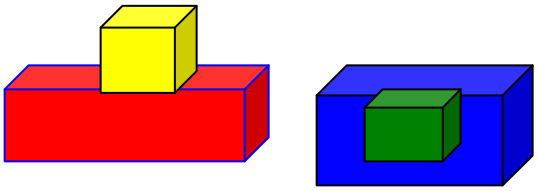
nama relation

- *Events* : *Relation* berdasarkan waktu

  contoh: 'Kemarin John pergi ke Toko'

  (# pergi :John :Toko :Kemarin)

Contoh natural language understanding: “SHRDLU” ---> MIT

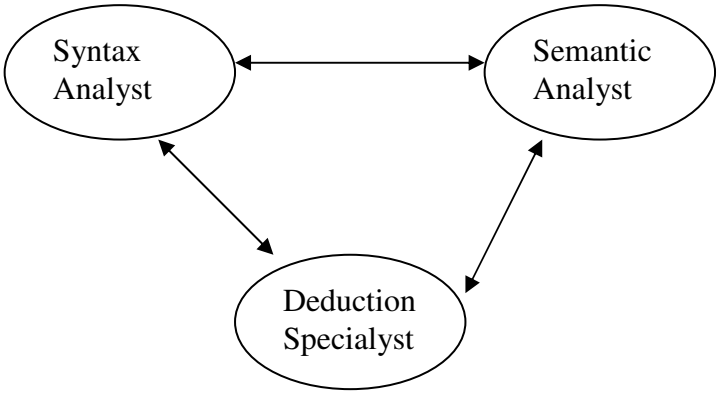


Word model :
1. (# support :B1 :B2)
2. (# yellow :B2)
3. (# Red :B1)
4. (# Contains :B3 :B4)
5. (# Blue :B3)
6. (# Green :B4)
7. (# Cube :B1)
8. (# Cube :B2)
9 (# Cube :B3)
10.(# Cube :B4)

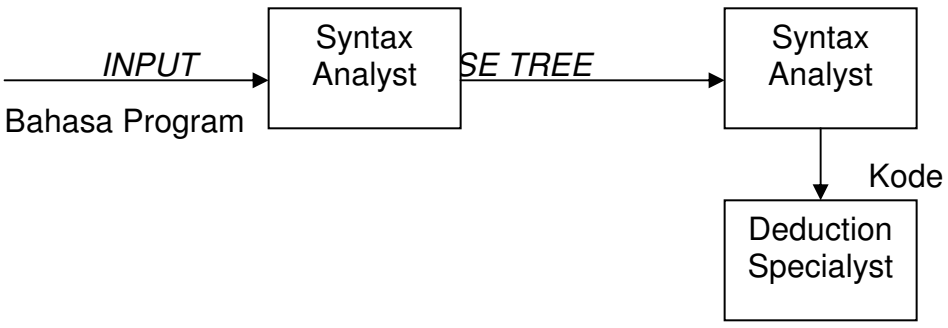
‘pick up the yellow block’

(# Grasp :B2) ;; hapus dan create model baru

Tiga aspek dasar dalam sistem *understanding*



Di *compiler* --> urutannya linear



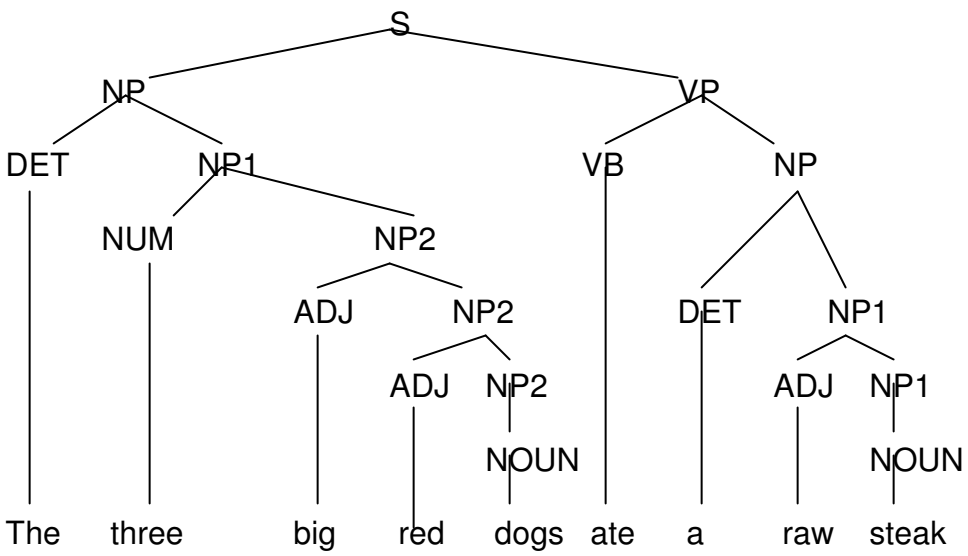
Bahasa program di input, lalu dicek atau *analyst word*nya ada/tidak di *dictionary*, sebelum dicek syntaknya, kemudian dicek *syntax*nya dengan melihat

*grammam*ya, hasilnya berupa *parse tree* atau *structure grammar*, setelah itu dianalysis arti atau makna katanya, kemudian direpresentasikan dalam kode yang sudah tahu maunya

Contoh:

Aturan(*Grammar)*		*Phrase(Group)*	
S	→ NG VERB NG	S	→ NP VP
NG	→ DET NOUN	NP	→ DET NP1
NG	→ DET ADJS NOUN	VP	→ VB NP
ADJS	→ADJ	NP1	→ NUM NP2
NG	→ PREP NG	NP2	→ ADJ NP2
NG	→ Q NOUN	NP2	→ NOUN
Q	→ NG	NP1	→ PREP NP2
NG	→ NOUN	NP	→ NP1
		NP1	→ ADJ NP1
		NP1	→ NOUN

Contoh: "*The three big red dogs ate a raw steak"*
"The graffis ate the apples and drank the vodka"



5.6. Proses Penterjemahan (*Transalation Process*)

Proses Penterjemahan dalam mengerti bahasa alami(*Natrural Language Understanding*), ada 4 tahap:

1. ***Lexical Analysis***: cek masing-masing word & *dictionary lock up*
2. ***Syntax Analysis***(Parsing): sesuai *grammar*
3. ***Semantic Analysis***: mengecek masing-masing arti kata
4. ***Discourse*** :
   - Melihat semua kalimat yang lain
   - Sifatnya kompleks
   - Mengecek arti kata secara keseluruhan
   - Hukum/sesuai daerahnya

Contoh: "**These Students had a good knowledge of LISP features**"

Lexical Analysis

Menghubungkan setiap kata dalam kalimat, dengan informasi tentang

- Kategori gramatik: Noun(NOUN), VERB(VB), Determiner(DET), Adjective(ADJ)
- Root(asal Kata) : have untuk "had", this untuk "these", student untuk "students"
- Tense atau form(bentuk) : present, past, ….
  Singular(SING), Plural (PLUR)

These (NOUN, this, Plur)	Students (NOUN, student, PLUR)	Had (VB, have, PAST)	A (DET, a, SING)	Good (ADJ, good)
Knowledge (NOUN, knowledge, sing)	Of (Prep)	LiSP (NOUN, LISP, Sing)	Features (Noun, feature, Plur)	

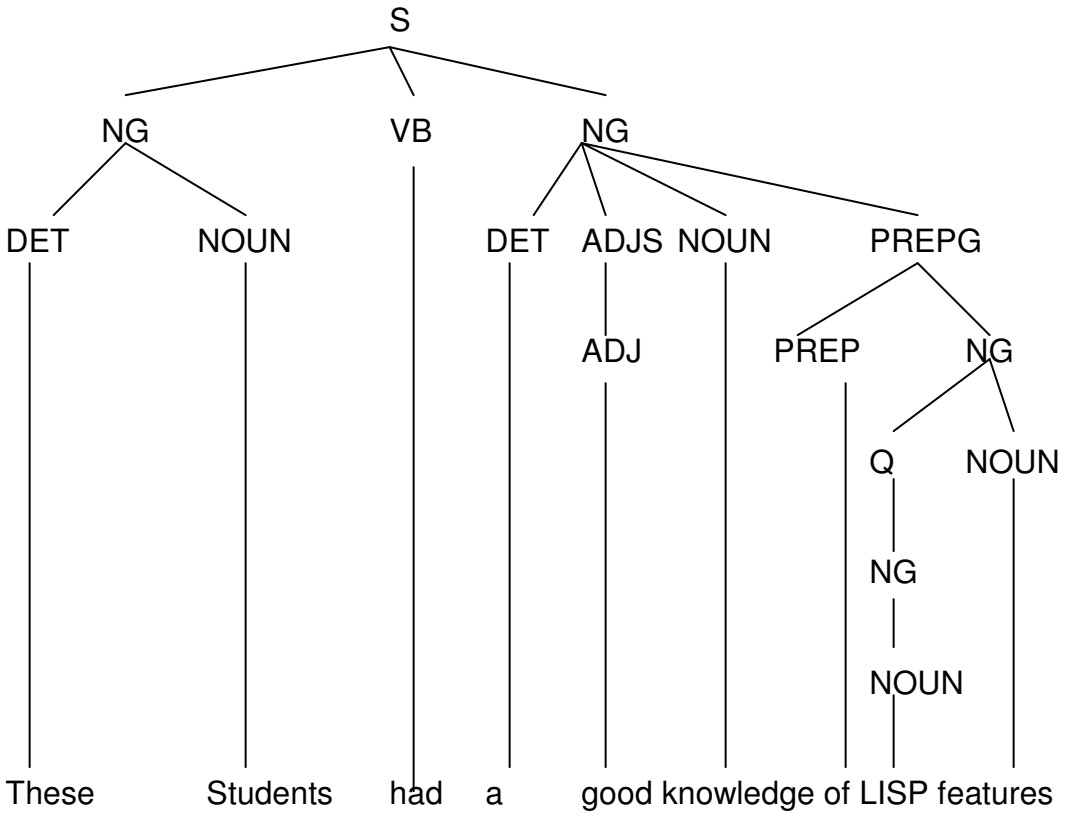
Syntax Analysis

Syntax Analysis

Menyusun *grammar* dari kalimat yang dapat direpresentasikan dengan "*Parse Tree*"

Setiap Node diberi Label:

S: Sentences	Adjs: Adjective Sequences	PREPG:Preposition Group
NG: Noun Group	Adj : Adjective	DET: Determiner
Q: Qualifier	VB: Verb	PREP: Preposition



Setelah parse tree di atas terbentuk, maka tersusunlah grammar sebagai berikut:

Grammar :

S	→ NG + VB + NG
NG	→ DET + NOUN
NG	→ DET + ADJS + NOUN + NOUN + PREPG
ADJS	→ ADJ
PREPG	→ PREP + NG
NG	→ Q + NOUN
Q	→ NG
NG	→ NOUN

↓

Semantic Analysis

Semantic Analysis

Menentukan kegiatan utama dalam kalimat walaupun verbnya "have", kegiatan utama : "***knowledge***" menentukan pelakunya, obyek dari kegiatan & karakteristik lainnya (waktu,lokasi)

Kegiatan : know

Qualifier : well

Pelaku : these students

Obyeks : lips

Waktu : past (lampau)

Lokasi :

DISCOURSE

Informasi yang dikumpulkan dari kalimat sebelumnya dari risalah dipergunakan untuk menyelesaikan hal-hal yang belum jelas :

-Two students = Peter, Jane

These

- Menghasilkan hubungan yang lebih spesifik.
- Menambah *knowledge*

Kegiatan : know

Qualifier : well

Pelaku : Peter & Jane

Obyek : lisp

Waktu : musim gugur 1988

Lokasi : San Fransisco

BAB VI

SISTEM PAKAR

(*EXPERT SYSTEM*)

Expert System yaitu program-program yang bertingkah laku seperti manusia pakar/ahli(*human expert*).

Sistem pakar atau sistem berbasis pengetahuan adalah yang paling banyak aplikasinya dalam membantu menyelesaikan masalah-masalah dalam dunia nyata.

Contoh aplikasi dari program ini antara lain yaitu :

- Delta dari *General Electric* untuk konsultasi kerusakan lokomotif.
- Prospector dari *Stanford Research Institute* untuk penaksiran prospek mineral.
- Xycon dari *Digital Equipment Corp's* untuk mengkonfigurasi bagian-bagian komputer.
- Mycin dikembangkan pada Universitas Stanford (1970), untuk menolong para ahli dalam mendiagnosa bakteri penyakit tertentu.

Program kecerdasan tiruan ini dapat dilakukan dengan menggunakan suatu program paket, yaitu alat pengembangan sistem aplikasi pengetahuan (*knowledge system application development tool*) seperti:

- VP-Expert
- PC PLUS
- GURU
- JESS(*Java Expert System Shell*) Version 5.0 http://herzberg.ca.sandia.gov/jess/FAO.html
- EXSYS, *dan lain-lain.*

Atau dengan menggunakan bahasa untuk pemrograman kecerdasan tiruan seperti *:*

- PROLOG (*Programming Logic*)
- WIN-PROLOG 4.040 (*Windows-Programming Logic)* http://www.lpa.co.uk/web386/8f922bf1.zip

- LISP(*Lisp Processing*)
- CLIPS(*C Language Integrated Production System*)
  http://www.ghgcorp.com/clips/download/source/

Perangkat-perangkat lunak ini dapat dijalankan dengan komputer pribadi (PC), sehingga pengembangan untuk aplikasi kecerdasan tiruan dapat dilakukan dengan mudah dan dengan biaya yang murah.

Sistem pakar adalah program komputer yang :

- Menangani masalah dunia nyata, masalah yang komplek yang membutuhkan interpretasi pakar.
- Menyelesaikan masalah dengan menggunakan komputer dengan model penalaran manusia dan mencapai kesimpulan yang sama dengan yang dicapai oleh seorang pakar jika berhadapan dengan masalah.

Komputer berbasis sistem pakar adalah program komputer yang mempunyai pengetahuan yang berasal dari manusia yang berpengetahuan luas(pakar) dalam domain tertentu, di mana pengetahuan di sini adalah pengetahuan manusia yang sangat minim penyebarannya, mahal serta susah didapat. Di sini keahlian dari manusia dimasukkan ke pengetahuan tersebut untuk menyelesaikan masalah, seperti yang dilakukan manusia.

Walaupun sistem pakar dapat menyelesaikan masalah dalam domain yang terbatas berdasarkan pengetahuan yang dimasukkan ke dalamnya, tetapi sistem pakar tidak dapat menyelesaikan masalah yang tidak dapat diselesaikan manusia. Oleh sebab itu keandalan dari sistem pakar terletak pada pengetahuan yang dimasukkan ke dalamnya.

Kondisi-kondisi dimana sistem pakar dapat membantu manusia dalam menyelesaikan masalahnya, antara lain:

- Kebutuhan akan tenaga ahli(pakar) yang banyak, tetapi pakar yang tersedia jumlahnya sangat terbatas.
- Pemakaian pakar yang berlebihan dalam membuat keputusan, walaupun dalam suatu tugas yang rutin.

- Pertimbangan kritis harus dilakukan dalam waktu yang singkat untuk menghindari hal-hal yang tidak diinginkan.
- Hasil yang optimal, seperti dalam perencanaan atau konfigurasi.
- Sejumlah besar data yang harus diteliti oleh pakar secara kontinu.

Dengan menggunakan sistem pakar dalam membantu memecahkan masalah, didapat beberapa keuntungan:

- Sifatnya yang permanen.
- Mudah untuk ditransfer atau direproduksi.
- Mudah didokumentasikan.
- Menghasilkan keluaran yang konsisten.
- Biaya yang murah.
- Dapat digunakan untuk 24 jam sehari
- Dapat dibentuk semenjak ada keterbatasan dari manusia pakar
- Sulit mendapatkan seorang yang *expert*/pakar sehingga sistem pakar dapat menggantikan tugas tersebut.
- Pengetahuan pada sistem pakar mudah disimpan dan di*copy*
- Pengetahuan yang ada tidak mudah hilang
- Selalu membentuk opini terbaik dalam batas pengetahuannya.

Kerugian *Expert System*:

- Kurang personalitinya
- Tidak dapat menyelesaikan masalah yang membutuhkan intuisi

Expert System terdiri dari 2 bagian yang harus dimilikinya:

1. *Knowledge Base*
2. *Inference Engine*

Knowledge Base

Sebuah database yang menyimpan informasi pengetahuan tertentu dan aturan-aturan tentang subyek tertentu.

Knowledge base terdiri 2 bentuk:

1. Obyek : kesimpulan yang didefinisikan oleh kelompok aturan.
2. Atribut : kualitas tertentu di mana bersama-sama dengan aturan membantu mendefinisikan obyek.

Jadi *knowledge base* dapat diartikan : daftar dari obyek-obyek dengan kelompok-kelompok aturan dan atribut.

Inference Engine

Inference Engine adalah bagian dari sistem pakar yang mencoba menggunakan informasi yang diberikan untuk menemukan obyek yang sesuai.

Kategori *inference engine*:

1. *deterministic*
2. *probabilistik*

Dasar untuk membentuk *inference engine*:

1. *Forward Channing*
2. *Backward Channing*
3. *Rule Value*

Metoda *Forward-Channing*

Kadang disebut: *data-driven* karena *inference engine* menggunakan informasi yang ditentukan oleh user untuk memindahkan keseluruh jaringan dari logika 'AND' dan 'OR' sampai sebuah terminal ditentukan sebagai obyek.

Bila *inference engine* tidak dapat menentukan obyek maka akan meminta informasi lain. Aturan(*Rule*) di mana menentukan obyek, membentuk Path(lintasan) yang mengarah ke Obyek. Oleh karena itu, hanya satu cara untuk mencapai obyek adalah memenuhi semua aturan.

Metoda *Backward Chainning*

Merupakan kebalikan dari *forward chainning* di mana mulai dengan sebuah hipotesa(sebuah obyek) dan meminta informasi untuk menyakinkan atau mengabaikan *backward chainning inference engine* sering disebut: *'Object – Driven/Goal-Driven'*

Karena sistem mulai dengan obyek dan mencoba untuk memverifikasi obyek.

Metoda *Rule-Value*

Merupakan pendahulu dari *forward* atau *backward channing inference engine system,* karena metoda ini membutuhkan informasi yang mempunyai kepentingan terbesar. Metoda ini umumnya membuktikan mekanisme *backward channing*.

Kesulitan dari sistem *'Rule Value'*:

1. Dalam situasi yang nyata, *knowledge base* sering menjadi besar jumlah kemungkinan kombinasi, sehingga tidak dapat menentukan informasi yang sesuai dan menghapus ketidakpastian pada keadaan yang ada.

2. Sistem ini membutuhkan *knowledge base* yang berisi tidak hanya standard informasi atribut obyek melainkan juga nilai penentu yang membuat bentuk *knowledge base* lebih sulit.

Pembentukan *Expert System(ES)* Tujuan Umum

Metoda yang dipakai: *Backward channing* karena sistem tersebut memperbolehkan mekanisme inferensi menggunakan berbagai *knowledge base*.

Mekanisme *inference*: campuran antara obyek dan atribut.

Spesifikasi yang harus ada dalam mekanisme inferensi:

1. Sistem pakar tidak boleh membutuhkan atribut yang sama lebih dari satu kali.
2. Sistem pakar segera menolak dan memindahkan untuk melewati suatu obyek yang tidak membutuhkan atribut yang perlu diketahui.
3. Sistem pakar harus dapat melaporkan mengapa perintah mengikuti alur alasan.

Sistem pakar adalah progam AI dengan basis pengetahuan (*knowledge base*) yang diperoleh dari pengalaman/pengetahuan pakar/ahli dalam memecahkan persoalan pada bidang tertentu dan didukung mesin inferensi/*inference engine* yang melakukan penalaran/pelacakan terhadap sesuatu/fakta-fakta yang diberikan oleh *user*/pemakai, dicocokkan/*matching* dengan fakta-fakta dan aturan/kaidah yang ada di basis pengetahuan setelah dilakukan pencarian, sehingga dicapai kesimpulan.

Sistem pakar memecahkan persoalan yang secara normal dipecahkan dengan keahlian/kepakaran manusia

Sistem pakar adalah perangkat lunak komputer yang memiliki basis pengetahuan (*knowledge base*) untuk domain tertentu & juga memerlukan untuk mengeksplotasikan satu atau lebih mekanisme penalaran atau pemikiran / pertimbangan (*inference*) yang menyerupai seorang pakar dalam memecahkan masalah.

Kemampuan ES: memecahkan masalah-masalah praktis pada saat sang pakar berhalangan.

Basis pengetahuan (*knowledge base*) merupakan bukan pengetahuan formal (**bukan *text book***), tetapi harus berupa pengalaman bekerja seorang pakar pada sebuah bidang tertentu disiplin ilmu tertentu.

ES : ***Learning by doing for knowledge*** (pengetahuan yang berasal dari belajar dari pengalaman)

6.1. Ciri-ciri Sistem Pakar

Ciri-ciri sistem pakar :

1. Terbatas pada domain keahlian tertentu
2. Dapat memberikan penalaran untuk data yang tidak pasti.
3. Dapat mengemukakan rangkaian alasan-alasan yang diberikannya dengan cara yang dapat dipahami.
4. Berdasarkan pada kaidah/ ketentuan*/ rule* tertentu.
5. Dirancang untuk dapat dikembangkan secara bertahap.

6. Pengetahuan & mekanisme penalaran (*inference*) jelas terpisah.
7. Keluarannya bersifat anjuran.
8. Sistem dapat mengaktifkan kaidah secara searah yang sesuai dituntun oleh dialog dengan *use*r.

Keuntungan sistem pakar :

1. Membuat seorang yang awam bekerja secara seperti layaknya seorang pakar.
2. Meningkatkan produktifitas akibat meningkatnya kualitas hasil pekerjaan, disebabkan meningkatnya efesiensi kerja.
3. Menghemat waktu kerja.
4. Menyederhanakan pekerjaan.
5. Merupakan arsip yang terpercaya dari sebuah keahlian.
6. Memperluas jangkauan, dari keahlian secara pakar.

   Dimana ES yang telah disahkan akan sama saja artinya :
   - seorang pakar yang tersedia dalam jumlah besar.
   - dapat diperoleh & dipakai dimana saja

6.2. Tipe-tipe sistem pakar

Tipe-tipe sistem pakar berdasarkan struktur program, ada 3(tiga) tipe :

A. Program Mandiri

   Sistem pakar yang murni dan berdiri sendiri, artinya program utamanya tanpa mengandung *subroutine* yang memakai algoritma konvensional.

B. Program Terkait

   Sistem pakar yang dikelilingi program lainnya, artinya sebuah *subroutine* yang akan dipanggil oleh program utama.

   Misalnya memiliki *subroutine* untuk:

   - perhitungan matematik
   - pembuatan grafik
   - keperluan lainnya

   > algoritma konvensional

C. Program Terhubungkan

Sistem pakar merupakan program yang dapat berhubungan dengan paket program lainnya, misalkan:

- spreadsheet (lotus, excel, quatro pro, dan sebagainya)
- *Database Management System* (dBase III+, foxbase, dan sebagainya)
- atau pembuat Grafik.

6.3. Komponen Sistem Pakar

Sebuah program sistem pakar terdiri atas komponen-komponen sebagai berikut :

a. Basis Pengetahuan (*knowledge base*)

- inti program sistem pakar
- merupakan representasi pengetahuan (*knowledge representation*) dari seorang pakar.
- Tersusun atas fakta yang berupa objek dan kaidah/ ketentuan (*rule*) yang merupakan informasi tentang cara bagaimana membangkitkan fakta baru dari fakta yang sudah diketahui.

  Facts list (daftar fakta-fakta) berisikan hasil observasi dan sesuatu kenyataan yang dibutuhkan selama pengolahan

  Bagian yang mengandung semua fakta-fakta, baik fakta awal pada saat sistem mulai beroperasi maupun fakta-fakta yang didapatkan pada saat pengambilan kesimpulan.

b. Mesin Inferensi (*Inference Engine*)

- Bagian yang mengandung mekanisme fungsi berpikir dan pola-pola penalaran sistem yang digunakan seorang pakar.
- Mekanisme ini akan menganalisa sesuatu masalah tertentu dan selanjutnya mencari jawaban / kesimpulan yang terbaik.
- Memilih pengetahuan yang relevan dalam rangka mencapai kesimpulan.

- Memulai pelacakannya dengan mencocokkan kaidah (*rule*) dalam basis pengetahuan dengan fakta-fakta yang ada dalam *facts list* disimpan dalam Basis Pengetahuan di harddisk.

Ada dua teknik penalaran *(inference*):

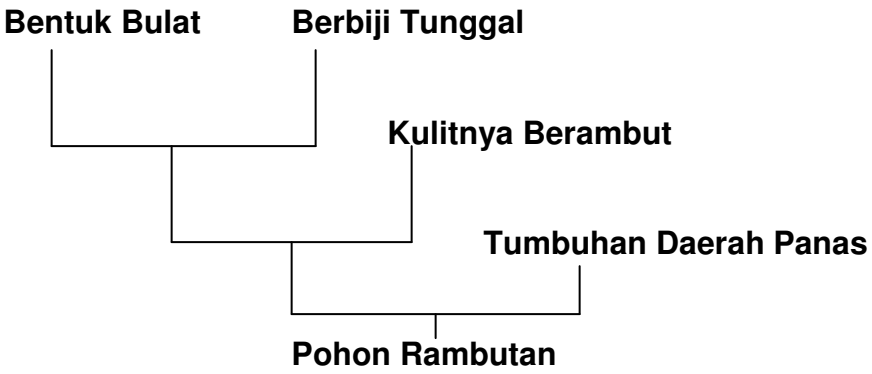
- Pelacakan ke belakang (*backward channing*) yang memulai penalarannya dari kesimpulan(*goal*), dengan mencari sekumpulan hipotesa-hipotesa yang mendukung menuju fakta-fakta yang mendukung sekumpulan hipotesa-hipotesa tersebut.
- Pelacakan ke depan (*forward channing*) memulai dari sekumpulan fakta-fakta (data) dengan mencari kaidah yang cocok dengan dugaan/hipotesa yang ada menuju kesimpulan.

Kedua teknik penalaran dipengaruhi oleh tiga macam teknik penelusuran (*searching*):

- *Depth-First Search*
- *Breadth-First Search*
- *Best-First Search*

1) *Forward Channing*

Contoh 1:



Contoh 2: F.C. menggunakan implikasi

Implikasi 1: p1: a college professor teaches in the summer

q1 : the professor can't do anything but teach

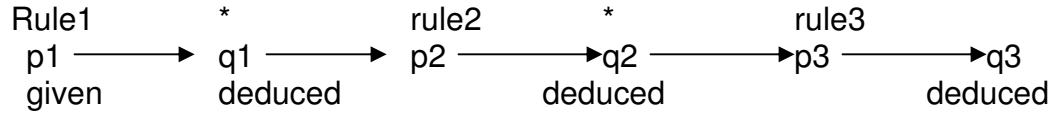
Implikasi 2: p2: the professor can't do anything but teach (= q1)

q2 : the professor does not have time to do research

Implikasi 3: p3: the professor does not have time to do research (= q2)

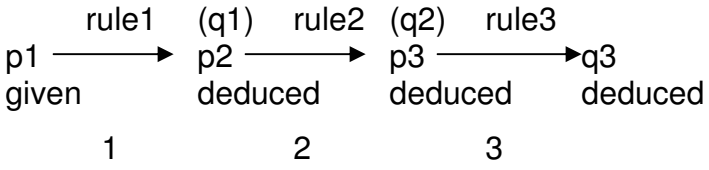
q3: the professor is unhappy

Basis pengetahuan (*knowledge base*) dapat dituliskan dengan menggunakan variabel substitusi :



Hasil : p1 ⟶ q3
given deduced

Tanda * adalah *symbolic matching approach*. Atau dapat digambarkan sebagai berikut:



Gambar di atas memperlihatkan bentuk dari *forward channing* yang akan menghasilkan : p1 ⟶ q3
given deduced

Basis aturannya(*rule base*):

Step1: ((p1 ⟶ p2) p1 = *True*) ⟶ (p2 = *True*)

Step1: ((p2 ⟶ p3) p2 = *True*) ⟶ (p3 = *True*)

Step1: ((p3 ⟶ q3) p3 = *True*) ⟶ (q3 = *True*)

2) *Backward Channing*

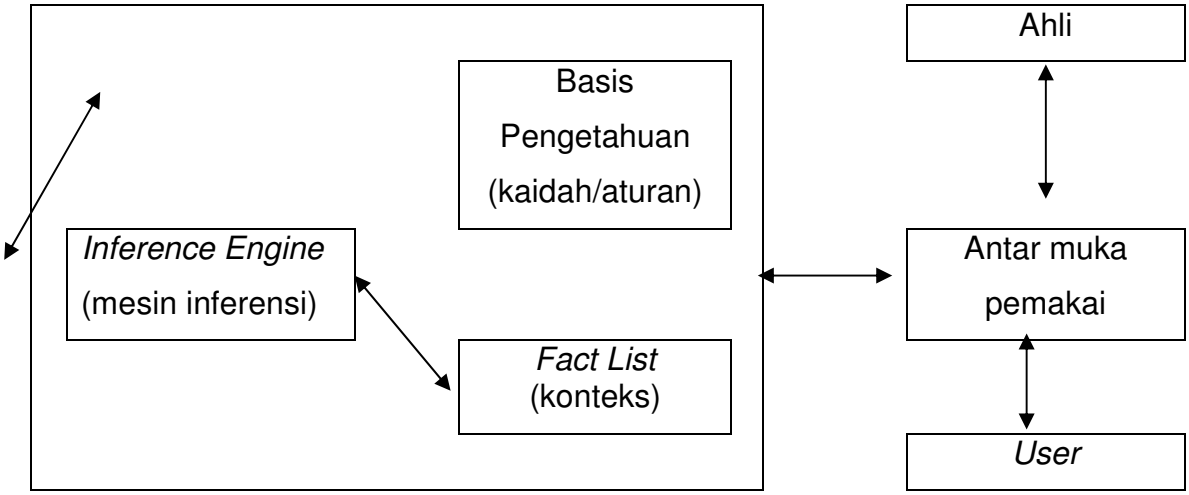
c. Antar Muka Pemakai(*User Interface*)

- Bagian penghubung antara sistem pakar dengan pemakai
- Akan terjadi dialog antara program dan pemakai
- Program akan mengajukan pertanyaan-pertanyaan dan jawaban berbentuk "ya"/"tidak", berbentuk panduan menu(*menu driven*), pernyataan-pernyataan bahasa alami(*natural language*), dan *graphics Interface style*. Program sistem pakar akan mengambil kesimpulan berdasarkan jawaban-jawaban dari pemakai tadi.

d. Development Engine

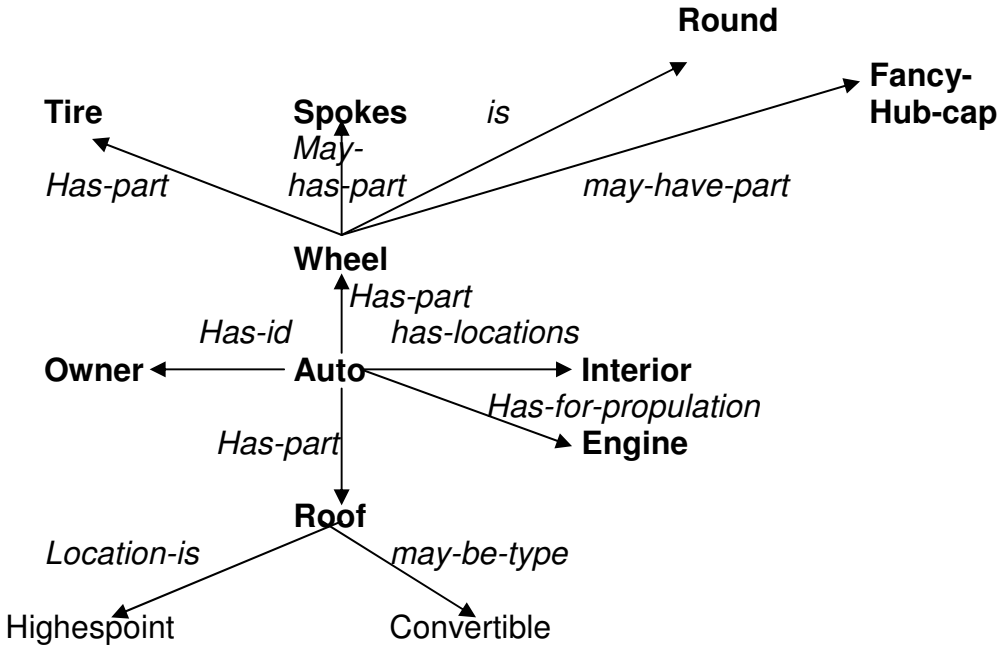
Bagian dari sistem pakar sebagi fasilitas untuk mengembangkan mesin inferensi dan penambahan basis pengetahuan yang akan dilakukan oleh *knowledge engineer*(harus punya keahlian dalam mengerti bagaimana pakar menerapkan pengetahuan mereka dalam memecahkan masalah, mampu mengekstrasi penjelasan(*knowledge acquisition*) mengenai pengetahuan dari pakar), bila si pakar menemukan pengetahuan dan aturan yang baru dari pengalaman ia bekerja.

Struktur sistem pakar dapat dirinci sebagai berikut:



Contoh Representasi pengetahuan dari *Automobile*:

Gambar Jaringan semantik dari auto:



Representasi *fact list* dari auto:

Auto	has-part	wheels
	has-id	owner
	has-locations	interior
	has-part	root
	has-for-propulation	engine

wheel	has-part	tire
	may-has-part	spokes
	may-have-part	fancy-hub-cap
	is	round
roof	locations-is	highes-point
	may-be-type	convertible

Dari segi bahasa yang digunakan :

- LISP , maka harus ditentukan terlebih dulu dengan tegas apakah ES yang akan dibangun akan menggunakan teknik pelacakan ke belakang / teknik pelacakan ke depan.
- Prolog , seperti turbo prolog, maka leibh baik menggunakan pelacakan ke belakang.

6.4. Kemampuan Tambahan Diperlukan ES

Untuk lebih meningkatkan kemampuan ES, diperlukan komponen-komponen tambahan:

a. fasilitas penjelasan
   - untuk menjelaskan bagaimana prosesnya sampai kesimpulan-kesimpulan tersebut diperoleh.
   - dengan cara memperlihatkan *rule – rule* yang digunakan

b. kemudahan memodifikasi (K Bs)
   - dikarenakan ilmu pengetahuan berkembang
   - kemampuan seorang pakar bertambah terus

c. kompatibilitas
   - dapat dijalankan pada berbagai jenis komputer.

d. kemampuan belajar
   - kemampuan ES untuk menambah sendiri pengolahannya, selama interaksi dengan pemakainya.

6.5. Klasifikasi sistem pakar

Klasifikasi sistem pakar berdasarkan kegunaannya:

a. Diagnosis :
   - ❖ Digunakan untuk merekomentasikan:
     - obat untuk orang sakit ,kerusakan mesin, kerusakan rangkaian elektronik.
   - ❖ Menemukan apa masalah/kerusakan yang terjadi.
   - ❖ Menggunakan pohon keputusan (*decision tree*) sebagai representasi pengetahuannya.

b. Pengajaran :
   - ❖ Digunakan untuk pengajaran, mulai dari SD s/d PT.
   - ❖ Membat diagnosa apa penyebab kekuranagn dari siswa, kemudian memberikan cara untuk memperbaikinya.

c. Interpretasi
   - ❖ Untuk menganalisa data yang tidak lengkap, tidak teratur, dan data yang kontradiktif.

   Misal: untuk interpretasi citra

d. Prediksi :
   - ❖ Contoh : - Bagaimana seorang pakar meteorologi memprediksi cuaca besok berdasarkan data-data sebelumnya.
   - ❖ Untuk peramalan cuaca.
   - ❖ Penentuan masa tanam.

e. Perencanaan :
   - ❖ mulai dari perencanaan mesin-mesin s/d manajemen bisnis.
   - ❖ Untuk menghemat biaya, waktu & material, sebab pembuatan model
   - ❖ sudah tidak diperlukan.
   - ❖ contoh : - sistem kontigurasi komputer.

f. Kontrol
   - ❖ Digunakan untuk mengontrol kegiatan yang membutuhkan presisi waktu tinggi.
   - ❖ Misal : pengontrolan pada industri-industri berteknologi tinggi.

Kategori dari aplikasi sistem berbasis pengetahuan

Category	Problem Addressed	Types of Systems
Diagnosis	Infers system malfunctions from observations	Medical, electronic, financial analysis, auditing, machine repair
Monitoring	Compares observations in order to identify variations	Management control, nuclear power, plant regulation
Debugging	Precribes remedies for malfunctions	Computer Software
Repair	Executes a plan to administer a prescribed remedy	Autompbile, computer, telephone
Instruction	Diagnoses, debugs, and corrects student behaviors	Tutorial, remedial
Control	Interprets, predicts, repairs, and monitors system behavior	Air traffic control, battle management, manufacturing process control
Prediction	Infers likely consequences of given situations	Weather forecasting, crop estimation, financial forecasting
Interpretation	Infers situations descriptions from sensor data	Speech understanding, image analysis, surveillance, mapping
Design	Configures objects within situation constraints	Circuit layout, budgeting, automatic program generation
Planning	Develops guidelines for action	Strategic palnning, proses scheduling, military planning
Classification	Prescribescategories for given sets of criteria	Planning, scheduling, layout, remedial,audit

Beberapa sistem pakar yang terkenal :

- Mycin , dirancang oleh Edward Short life of Standford University dalam tahun 70-an.
- Dendral , merupakan produk peneliti di Universitas Standford.
  - dengan menggunakan pengetahuannya struktur molekular dan kimia.
  - Berusaha mengidentifikasi struktur molekul campuran yang tak dikenal.
- XCON & XSEL

  XCON dikenal sebagai RI dalam tingkat prototype dini.
  - Membantu konfigurasi sistem komputer besar.
  - Dikembangkan bersama dengan digital Equipment Corporation (DEC) & para peneliti, Carnegia Mellon University (CNU).
  - Membantu melayani order langganan sistem computer DEC VAX II/780 le dalam sistem spesikasi final yang lengkap.

  XSEL : ES DEC-CMU lainnya.
  - Dirancang untuk membantu karyawan bagian penjualan dalam memilih komponen sistem VAX (dengan pengetahuannya sistem komputer VAX II/780).
  - Pengetahuan yang ada dalam XSEL membantu untuk memilih konfigurasi yang dikehendaki, kemudian XSEL memilih CPU, memori, peripheral, dalam menyarankan paket *Software* tertentu yang paling tepat dengan konfigurasinya.
  - Bisa merancang *Layout* dasar suatu sistem kompeter.
- Prospector
  - Membantu ahli geologi dalam mencari & menemukan hasil tambang di bumi.
  - Berisi taxonomi berbagai macam mineral & batu-batuan.
  - Berisi bentuk-bentuk kandungan mineral/ batu-batuan yang diperoleh dari para ahli geologi.
  - Mengevaluasi areal dalam bentuk pertanyaan dan menentukan apakah jenis batu-batuan tertentu potensial itu terdapat di tempat tertentu.

- Contoh aplikasi sistem berbasis pengetahuan komersial (*Commercial Knowledge Base System Applications*) yang lainnya :
  - ACE (AT & T)
    Digunakan untuk memberikan laporan trouble-shooting dan analisa untuk perawatan kabel telepon.
  - AS/ASQ (Arthur Young)
    Digunakan untuk membantu dalam prosedur auiditing
  - AUDITOR(University of Illinois)
    Memilihkan prosedur audit untuk memverifikasi rekening pendapatan sebuah perusahaan.
  - AUTHORIZER'S ASSISTANT(American Express)
    Membantu dalam meninjau penipuan kartu kredit.
  - BUSINESS PLAN(Sterling Wentworth Corp.)
    Membantu pegawai professional dan pemilik bisnis tentang semua aspek yang menyangkut perencanaan keuangan.
  - CASH VALUE(Heuros Ltd.)
    Mendukung perencanaan proyek modal.
  - COMPASS(GTE Corp.)
    Troubleshoots tidak berfungsinya sirkuit telepon
  - CONCEPT(Tyashare)
    Memproduksi model-model dari pasar yang disenangi konsumen
  - DELTA(GE)
    Membantu mendiagnosa dan memperbaiki kereta api listrik diesel.
  - EL
    - Digunakan untuk menganalisa sirkuit elektronik yang terbuat dari transistor, dioda & resistor.
    - Bekerja melalui diagram skematik dari sirkuit yang telah dimasukkan dalam komputer & EL menentukan karakteristik sirkuit, nilai voltage dan strum yang ada pada semua titik sirkuit.
    - Sangat baik untuk rekayasa rancangan & bantuan belajar operasi sirkuit elektronik & rancangan.

- EXPERT TAX(Coopers & Lybrand)
  Memberikan bimbingan menghitung pajak.
- FIN PLAN(Wright Patterson Air Force Base)
  Mendukung perencanaan keuangan pribadi.
- FINANCIAL ADVISOR(Palladian)
  Memberikan bimbingan keuangan pada proyek, produk dan penggabungan serta akuisisi.
- FOLIO(University of Stanford),
  Membantu manajer investasi portfolio memutuskan sasaran investasi kliennya dan memilihkan portfolio yang terbaik yang dibutuhkan .
- SOPHIE
  Untuk membantu mahasiswa belajar memecahkan masalah atau kesulitan sirkuit elektronik.(Dengan cara mensimulasikan sirkuit & masalahnya).
- GENESIS(Intelli Corp.)
  Membimbing insiyur genetic dalam menganalisa molekul DNA.
- INTELLIGENT SECRETARY(Nippon T & T)
  Menangani jadwal dari personel dalam sebuah perusahaan.
- TRADER'S ASSISTANT(A.D. Little)
  Membimbing pedagang sekuritas dalam mengakses pasar modal.

BAB VII
ROBOTIK

Robotik adalah ilmu yang mematerikan kecerdasan/*intelegencia* terhadap energi artinya pengendalian secara cerdas terhadap gerakan yang terkoordinasi secara nyata.

Robot berasal dari bahasa chekoslovakia 'robota' yang artinya tenaga kerja (*Create the Artificial life Robot → Cheko is "worker").* Robot diharapkan dapat melihar, mendengar,menganalisa lingkungannya dan dapat melakukan tindakan-tindakan yang terprogram. Sekarang ini robot digunakan untuk maksud-maksud tertentu dan yang paling banyak adalah untuk keperluan industri seperti robot *cybotech P15* yang banyak digunakan untuk pekerjaan mengecat. Robot juga banyak diperlukan untuk mengerjakan pekerjaan-pekerjaan yang berbahaya, kotor dan sulit(tiga dimensi/3D).

Robot modern pertama kali dikembangkan oleh Joseph Engelberger dan George Devoe yang kemudian mendirikan perusahaan Unimation Company. Pengaruh robot mulai terasa dalam industri ketika negara Jepang mulai menggunakan secara intensif.

Asimov's 3 laws of robotics :

1. Tidak boleh melukai manusia
2. Harus patuh terhadap perintah manusia, kecuali yang bertentangan dengan no. 1.
3. Harus mempertahankan diri, kecuali bertentangan dengan no. 1. dan 2.

Lovable Friends and loyal companions to humans.

Robot : motor primitif kapasitas *intelligence di computer science* dan *control theory (sensory perception, decision making*, kemampuan *intelligent*).

Aplikasi robotik terdapat pada :

- Bidang Industri : Keandalan Keefektivitas Biaya
- Bidang *Scientific* :
  - *Sensory Perception*
  - *Motor Control*
  - *Intelligent Behavior*

Sistem Pusat Urat Syaraf :

Sistem pusat urat syaraf, struktur paling kompleks, berisi trilliun urat syaraf, yangdihubungkan sedemikian rupa sehingga dapat menghasilkan sifat atau tindak tanduk dan imaginasi.

Pembagian secara hierarkhi menjadi 3 tingkat:

- Tingkat paling bawah : *Spinal Cord*
- Tingkat paling tengah : *Brain Stem*
- Tingkat paling atas : *Fore Brain*

Ketiga tingkatan ini membuat model otak.

Otak merupakan organ paling sulit, banyak komputasi yang berbeda, terlaksana pada saat yang bersamaan, dilaksanakan di berbagai tempat.

Komputer VS Otak

Otak masih lambat dibandingkan dengan *Digital Komputer* sekarang, tetapi operasi parallel yang banyak komputasi melebihi kecepatan atau kapasitas komputer paling cepat yang pernah ada.

Jadi robot merupakan *control system*.

APA :

- Ingatan
- Hubungan ingatan dan otak
- Pikiran
- Bagaimana mekanisme yang membangkitkan imaginasi
- Persepsi dan bagaimana hubungan dengan objek yang didapat
- Emosi dan mengapa kita mempunyainya

- Keinginan dan bagaimana kita memilih apa yang kita inginkan
- Bagaimana kita merubah keinginan menjadi kegiatan
- Bagaimana kita merencanakan dan bagaimana kita tahu apa yang diharapkan pada masa yang akan dating

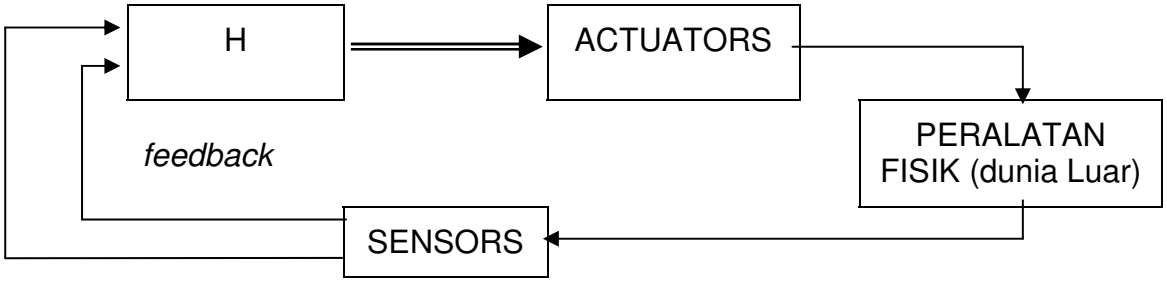
Elemen-elemen Dasar dari Otak :

Bagaimana struktur dasar dan fungsi urat syaraf.

Struktur otak setiap individu berlainan, begitu juga urat syaraf yang satu dengan yang lainnya.

Sensory Input

Suara, penglihatan, bau, rasa, raba + kekuatan, gerakan-gerakan otot adalah merupakan umpan balik ke pengatur mekanisme urat syaraf.



Gambar 7.1. *GOAL SEEKING CONTROL SYSTEM PADA SERUDMOTOR*

Penjelasan Gambar 7.1.:

S = SOURCE → C : Command, F : Feedback → Fungsi H menghasilkan vector P untuk membangkitkan *ACTUATORS*(Pengerak).

PERSEPSI

Menganalisis dan menentukan representasi object atau data Penglihatan, pendengaran, raba, bau, rasa. Tidak ada orang yang sama persis suaranya. Penglihatan merupakan masalah/*problem*? Yaitu jauh, dekatnya.

Pemecahan persoalan persepsi:

- *Distiration*
- *Shoothing*
- *Segmentation*
- *Modeling*
- Analisis dan perlu *contraints*

Penggunaan *contraints* dalam pemecahan persoalan:

Analisis domain persoalan untuk menentukan apa saja *contraints*nya.

Pecahkan persoalan dengan mengaplikasikan algoritma *contraints* yang mempergunakan *contraints* mulai tahap 1 untuk mengendalikan *search*.

Bila seseorang mendengar kata: **'ROBOT'**, orang akan menanggap bahwa robot adalah sebuah gambaran dari *hardware*(perangkat keras), yaitu peralatan mekanikal dan elektronikal yang membentuk fisik dari robot. Yang dapat mengerjakan tugas-tugas seperti halnya manusia.

Namun demikian sesungguhnya robot adalah: hubungan perangkat lunak(*software*) dengan perangkat keras(*hardware*), di mana perangkat lunak tersebut merupakan kecerdasan yang berada dibalik mesin itu yang mengendalikan seluruh gerak-gerik mesin tersebut dan dari kecerdasan inilah yang membedakan sebuah robot dari bentuk-bentuk otomatis lainnya.

7.1. Tipe Robot

Tipe robot ada 2(dua), yaitu:

1. Tipe pertama bentuknya sudah fix dan membutuhkan tempat yang tetap, misalnya robot-robot perangkat industri seperti yang digunakan untuk merakit mobil. Jenis ini dioperasikan hanya dalam lingkungan terkendali tinggi yang telah dirancang untuk hal tersebut.
2. Tipe kedua terdiri dari berbagai jenis robot otomatis/robot otonomous. Robot-robot ini dirancang untuk dioperasikan dalam dunia nyata. Robot jenis ini dibuat untuk bekerja seperti manusia. Robot jenis ini yang menarik bagi pemrogram AI.

Sebuah robot manipulator berfungsi ganda yang dapat diprogramkan kembali serta dirancang untuk menggerakkan bahan, onderdil, atau

peralatan khusus melalui gerakan-gerakan terprogram untuk melaksanakan berbagai tugas.

Motionmate adalah sebuah robot industri yang paling sederhana untuk melakukan proses mengambil dan meletakan komponen-komponen di dalam proses produksi. Robot ini dapat mengangkat komponen seberat 5 pound.

7.2. Gerakan-Gerakan Robot, Industri Robot

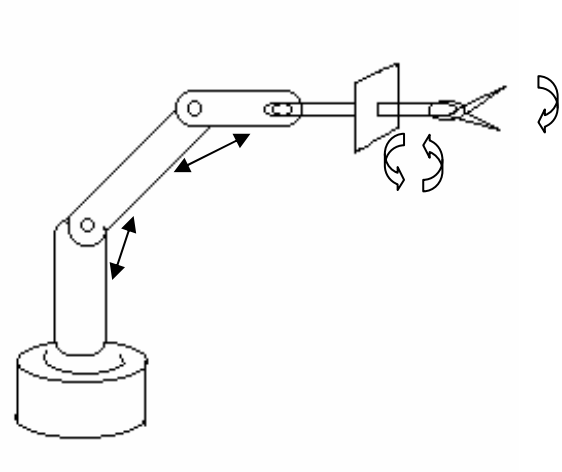
a. Tangan Robot:

Pada dasarnya menampilkan manipulasi peralatan dari sebuah robot baik unntuk tipe industri maupun otomatis.

Contoh: Mengambil sebuah gelas berisi air. Kelihatan gerakan tersebut tidak membutuhkan dukungan dan tanpa pemikiran, kenyataannya hal tersebut merupakan sebuah proses yang komplikasi di mana membutuhkan koordinasi dari beberapa otot.

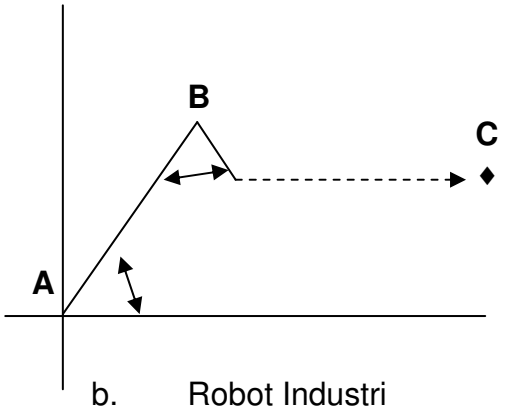
(Ingat! Bayi membutuhkan beberapa bulan untuk mempelajari hal tersebut).

Tangan robot umumnya dibentuk pada tangan manusia, kebanyakan tangan robot adalah *six-axis arm*, karena diperbolehkan untuk mendapatkan gerakan bebas.



Gambar 7.2. Tangan Robot

Kesulitan umum dari pengendalian tangan jenis robot ini bukan pada ketepatan pergerakan, tetapi masalah koordinasi dari 6 titik.



Untuk mencapai titik C, A dan B harus digerakkan

b. Robot Industri

Lapangan robot-robot jenis ini pada umumnya diterapkan untuk membentuk dan mempekerjakan robot-robot perakit industri, karena robot-robot ini digunakan dalam lingkungan terkendali, di mana robot-robot ini dipertimbangkan kurang pandai dibandingkan robot-robot otomatis.

Robot-robot industri hanya dapat menujukkan proses-proses di mana secara eksplisit telah diprogramkan untuk melakukan hal-hal tersebut.

Terdapat dua cara dalam mengajarkan robot, di mana robot-robot berpikir:

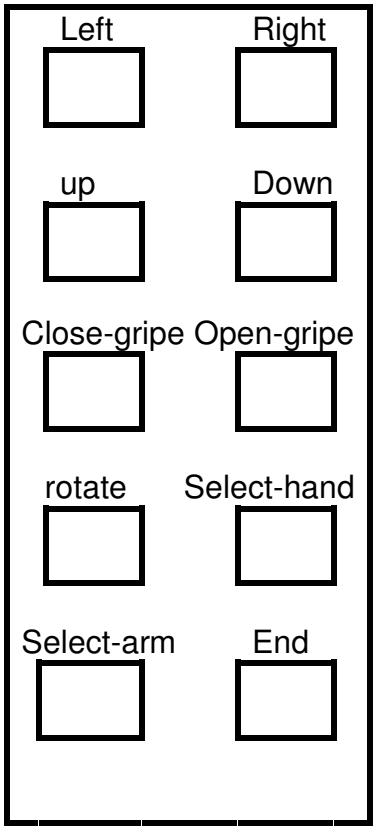
1. Dengan menggunakan *Teach Pendant.*
2. Diprogramkan dengan menggunakan bahasa kendali robot *robotic control language*.

1) *Teach Pendant*

Teach pendant merupakan metode yang paling banyak digunakan untuk mengajarkan robot hal-hal baru. Bentuk dan fungsi *teach pendant* mirip dengan remote control mobil-mobilan. *Teach pendant* digunakan untuk mengontrol setiap titik dari tangan robot. Misalkan kita ingin agar robot menutup gripnya, maka yang perlu kita lakukan adalah menekan tombol yang telah difungsikan untuk

menutup grip tangan robot tersebut. Walaupun setiap pembuat robot tidak akan sama persis bentuk *teach pendant*nya, namun prinsip kerja semua *teach pendant* adalah sama. Metoda umum dari pemrograman sebuah robot untuk menunjukkan proses baru melalui kegunaan *Teach Pendant*.

Teach pendant adalah tangan diletakkan pada kotak kendali yang memperbolehkan seorang operator untuk memudahkan berbagai hubungan dengan robot.



Teach pendant tidak terhubung secaralangsung ke robot, tetapi terhubung secara keseluruhan ke komputer kendali robot utama.

Jika diinginkan mengajari robot sebuah proses dapat digunakan *Teach Pendant* untuk membimbing robot ke rangkaian pergerakan yang membentuk proses tersebut.

Setiap pergerakkan, komputer merekam setiap posisi. Setelah pengajarannya selesai, robot dapat menunjukkan pekerjaannya sendiri tanpa membutuhkan pembantu.

Misalnya robot diajarkan mengambil benda di sebelah kiri robot maka robot digerakkan ke kiri dengan menempel panel kiri kemudian menutup gripnya. Setelah rangkaian pekerjaan tadi selesai, komputer merekam semua posisi yang dilalui robot agar robot dapat melakukan gerakan yang sama pada kesempatan berikutnya tanpa dipandu lagi

Gambar 7.3. *Teach pendant*

2) *Robotic Control Language*

Teach Pendant adalah sebuah metoda yang sangat baik untuk mengajari robot pekerjaan yang sederhana seperti mengelas dan mengecat. Namun ketika pekerjaan yang harus dilakukansemakin hari semakin kompleks dan *event* sinkronisasi eksternal semakin hari semakin penting atau ketika robot perlu untuk mengenali dan

menanggapi(merespon) berbagai kemungkinan situasi yang berbeda, maka sistem *teach pendant* lama kelamaan menjadi terbebani dan tidak sanggup lagi untuk menangani pekerjaan tersebut, sehingga *teach pendant* tidak berguna.

Hal ini menjadi alasan di mana bahasa kendali robotik (*Robotic Control Language)* dikembangkan. Bahasa kendali robotik adalah bahasa komputer yang secara khusus dirancang untuk mengendalikan sebuah robot..

Dalam *Robotic Control Language* tambahan terdapat perintah-perintah baru, seperti kendali LOOP(*Loop Control*) dan statement kondisional(conditional statement). Bahasa kendali robotik juga memasukkan perintah-perintah yang mengendalikan pergerakan robot. Bahasa ini mengendalikan pergerakan di mana menentukan bahasa kendali robotik yang merupakan bagian dari yang lain, seperti bahasa pemrograman untuk tujuan umum. Sebuah robotik dikendalikan oleh bahasa yang berisi sebuah basis data yang telah tersedia(*built in database*) di mana digunakan untuk menyimpan informasi tentang setiap gerakan yang akan dilakukan oleh robot. Bahasa kendali robotik bukan diciptakan untuk menggantikan *teach pendant* melainkan menyempurnakan *teach pendant.* Satu hal yang perlu dipahami bahwa bahasa kendali robotik dirancang untuk meletakkan kembali *teach pendant*, tetapi untuk mendukung bahasa tersebut. Oleh karena itu bahasa kendali robotik harus menyediakan *interface* untuk *teach pendant*.

Dalam metoda ini mengajarkan robot untuk menggunakan informasi dengan menggunakan arahan dan kemudian menggunakan bahasa kendali robotik untuk menjelaskan bagaimana robot seharusnya menggunakan informasi tersebut. Umumnya setiap lokasi tertentu diberikan sebuah nama simbolik di mana program dapat mengacu ke nama tersebut. Bahasa kendali robotik memiliki syntax yang serupa dengan bahasa BASIC dan paling banyak

digunakan adalah VAL dikembangkan oleh UNIMATION Corporation. Untuk lebih mengerti tentang program VAL, pelajarilah program berikut yang digunakan untuk mesin *conveyor belt* agar robot dapat memindahkan kotak pada roda ban berjalan.

contoh :

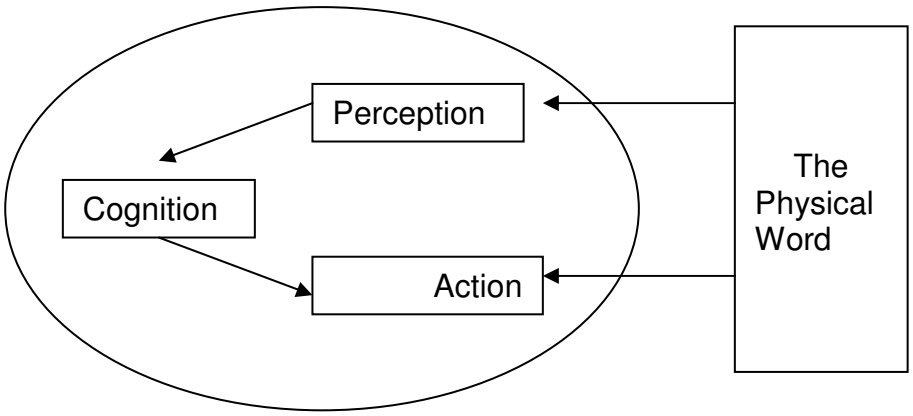
```
REMARK WAIT FOR OBJECT
10                    WAIT 2
REMARK OBJECT PRESENT, REMOVE IT
MOVE POS1
MOVE POS2
CLOSE1 REMARK CLOSE THE GRIP
MOVE BOX1
OPEN1 REMARK DROP THE OBJECT
GOTO 10
```

Dengan menggunakan program ini, robot akan menunggu sampai masukan sinyal 2 aktif: artinya:robot tidak akan melakukan apa-apa sampai sinyal nomor 2 meninggi(aktif). Sinyal aktif ini menandakan atau memberitahukan robot bahwa ada objek(kotak) pada mesin ban berjalan. Kemudian robot akan mendekat ke mesin ban berjalan(POS1), setelah itu robot akan berada di atas objek(POS2), lalu menutup grip untuk memegang objek dan memindahkannya dari ban berjalan. Akhirnya program akan loop dan menunggu objek berikutnya. VAL hanya mendukung huruf kapital. Ketiga posisi: POS1, POS2, dan BOX1 adalah nama simbolik dari lokasi dimana robot telah diajarkan oleh *teach pendant* (merekam lokasi tersebut).

7.3. Autonomous Robot

Autonomous Robot jauh lebih kompleks dari industrial robot, karena robot ini harus jauh lebih pandai. Jika *Autonomous Robot* berhasil beroperasi dalam lingkungan tak terkendali dari dunia nyata(kehidupan sebenarnya), maka robot ini akan membutuhkan berbagai keahlian di mana industrial robot tidak membutuhkan.

Misalkan saja diperlukan sensor agar robot dapat melihat dan mendengar berarti robot tersebut harus mengerti *natural language* dan arti dari bahasa tersebut. Memberikan robot dua kemampuan itu bukanlah pekerjaan yang mudah. Selain itu robot juga harus mampu menyelesaikan masalah yang merupakan pekerjaan programming yang paling sulit diantara pekerjaan lainnya. Hal ini penting agar robot dapat beradaptasi dengan banyak situasi sebab jelas kita tidak dapat memprogram robot untuk menyelesaikan semua masalah secara terperinci, yang dapat terjadi melainkan hanya agar robot dapat beradaptasi untuk kemudian memilih alternatif.



Gambar 7.4. Sebuah Rancangan untuk sebuah Autonomous Robot

Kita mendukung sebuah difinisi masalah dasar AI secara dauni yang ditangani, umumnya agar komputer mempunyai kemampuan seperti manusia.

Perception meliputi interpretasi penglihatan (*sights*), suara (*sounds*), *smell* (rasa) dan menyuruh / maraba (*tauch*).

Action *meliputi kemampuan mengendalikan / navigasi alam atau dunia dan memanipulasi objek.*

Salah satu pemecakannya kita membuat robot, yang …. Ia harus mengerti proses ini

7.4. Pembuatan Simulator Robot(*ROBOT SIMULATOR*)

Robot Simulator terdiri dari sebuah lingkungan yang diisi dengan 4(empat) obyek: segiempat, dua segitiga, dan robot dan sisa dari screen adalah *blank.*

Terdapat dua cara untuk memprogram robot:

1. Menggunakan *robotic-control language*, terdiri dari *command-command* :
   a. moveto
   b. move
   c. findt
   d. finds
   e. label
   f. goto
   g. ifsense/then

2. Mengajarkan robot

moveto

Menyuruh robot pergi ke lokasi baris, kolom tertentu pada screen. Sudut kiri atas screen adalah lokasi 0,0. Hal ini berarti baris-baris dinomori dari 0 sampai dengan 24 dan kolom dinomori dari 0 sampai dengan 79

contoh: moveto 12 60

move

Menyebabkab robot memindahkan satu posisi dalam arah yang ditentukan, di mana harus antara lain: kiri, kanan, atas atau bawah.

contoh : move up
move left

findt dan finds

Memerintahkan robot untuk menemukan sebuah segitiga atau sebuah segiempat. robot hanya dapat menemukan sebuah obyek jika obyek tersebut dilokasikan di atas obyek. Setelah robot menemukan obyek, robot akan ditempatkan satu kolom ke kiri dari sudut kanan paling atas obyek.

label

Sebuah label dapat berupa sebuah rangkaian karakter.

contoh: label one
label box38

goto

Statement ini menyebabkan eksekusi program dipindahkan ke label tertentu.

contoh : goto five

ifsense

ifsense menentukan apakah terdapat sebuah obyek baik di sisi kanan, kiri, atas atau bawah robot Jika terdapat obyek, maka *statement* yang mengikuti 'then' dieksekusi, jika tidak terdapat obyek, eksekusi melanjutkan ke baris berikut dari program arah 'sense' berupa up, down, left, atau right.

contoh :

```
label one
    move down
    ifsense right then goto one
```

a. Mengajarkan Robot

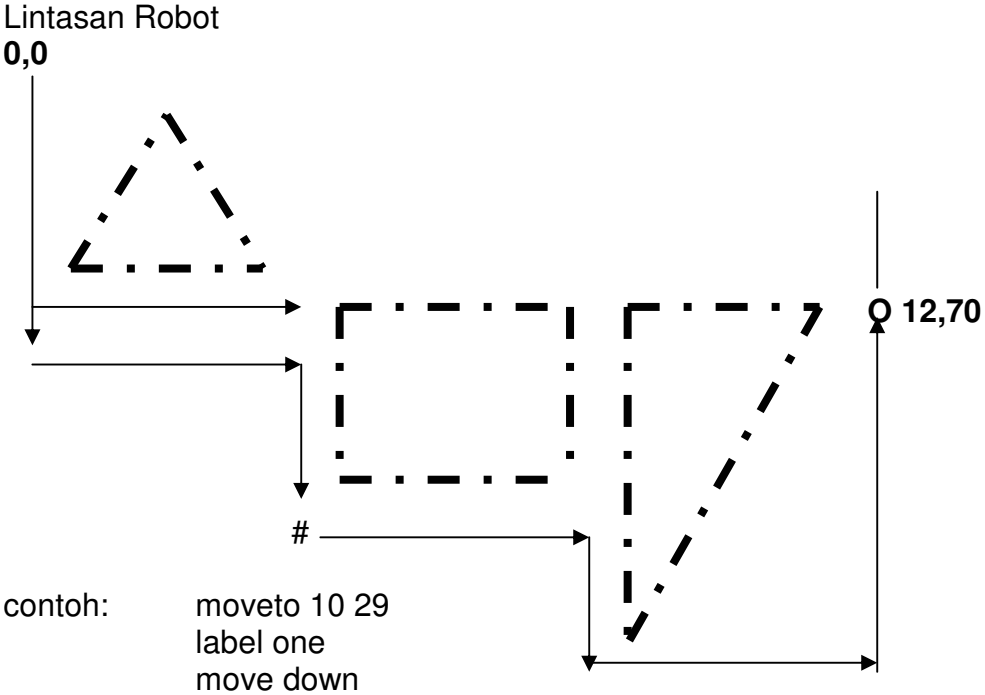
Perintah(*command*) '*teach*' dapat digunakan setiap saat dalam program. Perintah ini menyebabkan layar editor akan diletakkan kembali di layar. Robot ditempatkan pada posisi asalnya 0,0. Pada saat ini kita dapat menggunakan *keypad* angka untuk memindahkan robot dengan menggunakan tombol-tombol panah. Untuk mengembalikan posisi ke 0,0 dapat menggunakan tombol: ***Home***

Setiap gerakan direkam dan sesungguhnya menjadi bagian program. Untuk menghentikan mengajarkan robot maka menekan tombol: **End.**

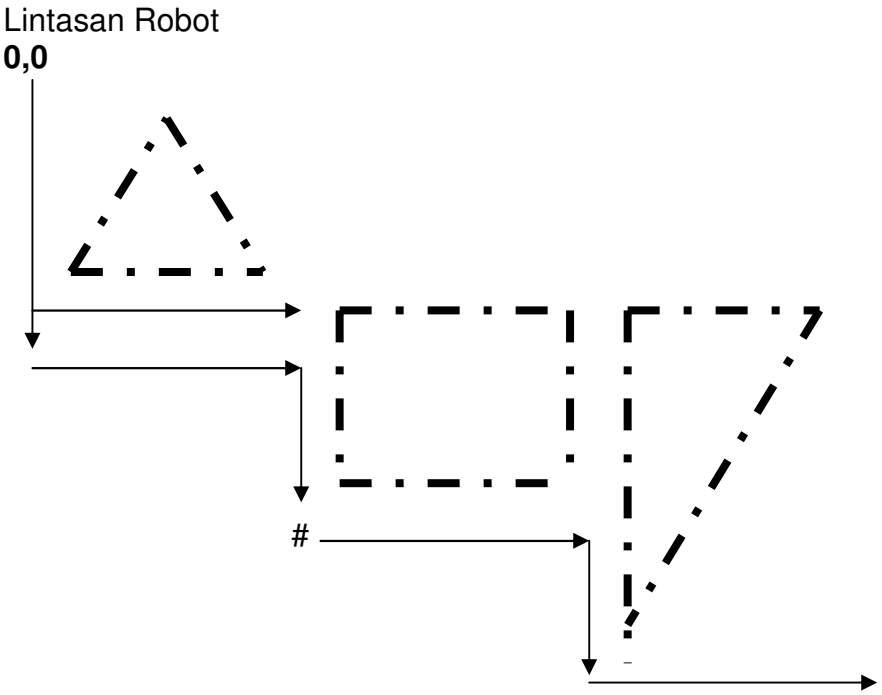
b. Penggunaan Simulator

Bila mengeksekusi program simulator, pertama kali diinstruksikan untuk memasukkan sebuah program. Hal ini dapat dilakukan dengan memasukkan sebuah garis pada setiap saat.

contoh : moveto 12 70
moveto 0 0
finds



contoh: moveto 10 29
label one
move down
ifsense right then goto one



BAB VIII
MACHINE LEARNING

Machine learning sangat erat kaitannya ke kecerdasan (*intelligence*) adalah belajar(*learning*). Pada kenyataannya, kecerdasan tidak ada tanpa kemampuan untuk belajar karena belajar ini berati kebutuhan akan pengetahuan baru. *Learning* memperbolehkan untuk beradaptasi ke dan menggunakan keuntungan berbagai situasi dan kejadian sehingga kemampuan *learning* menjadi suatu perangkat atau alat yang sangat berguna.

8.1. Jenis *Learning*

Ada sebuah pendapat yang saling kontradiksi di mana penerapan komputer untuk dapat belajar adalah: sangat mudah dan sangat sulit.

Pendapat tersebut berdasarkan adanya dua perbedaan yang mencolok dari *learning:*

1. *rote learning*
2. *cognitive learning*

Dari dua perbedaan di atas, kemungkinan besar timbul beberapa tipe atau jenis lain dari *learning* seperti:

1. learning by analogy
2. learning by example
3. learning by observation

Namun demikian, seluruh tipe *learning* dapat disimpulkan bahwa pengetahuan-pengetahuan yang ada dapat dibutuhkan atau tidak ke mekanisme sesungguhnya di mana memperbolehkan terjadinya *learning*.

1. *Learning by Rote*

   Banyak yang telah kita pelajari membentuk kenyataan di mana kita harus mengulang atau mengingatnya. Proses ini disebut: *Learning by Rote*.

   Contoh:
   - Ibukota R.I. : Jakarta
   - 1 + 1 = 2
   - Jarak dari Jakarta ke Bogor : 60 Km.

Rote learning tidak hanya terbatas pada kenyataan (*fact*), tetapi dapat diterapkan ke rangkaian kegiatan-kegiatan.

Contoh:

Seorang pekerja pabrik dapat belajar memindahkan kotak merah dan kuning dari lintasan rakitan dan meletakkannya ke tempat tertentu.

Rangkaian yang diingat adalah:

1. mengambil kotak-kotak merah dan kuning
2. meletakkan ke suatu tempat.

Inti dari *rote learning* adalah: spesialisasi di mana segala sesuatu yang dihafal adalah spesific dan rangkaian langkah-langkah yang membentuk proses tidak dapat digeneralisasi. Rangkaian penghafalan ini disebut: prosedur. Sebuah komputer dapat dengan mudah mempelajari sesuatu yang kita hafalkan, karena komputer diprogramkan.

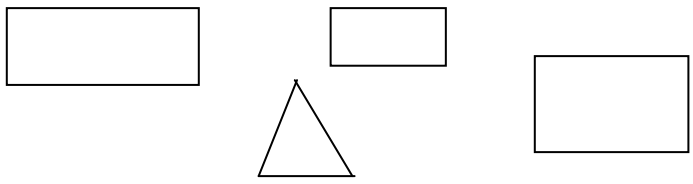
Rote Learning melibatkan : pengingatan baik kenyataan atau prosedur-prosedur dan tidak membutuhkan sesuatu generalisasi yang dirinci atau pemikiran tingkat tinggi.

2. *Cognitive Learning*

Suatu cara yang paling penting dalam belajar adalah: *cognitive learning* di mana cara ini paling sulit diterapkan. Dalam bentuk *learning* di sini, kita harus menggunakan alasan untuk: menganalisa, mengorganisasi, dan menghubungkan bagian-bagian tertentu pengetahuan.

Cognitive learning sangat menakjubkan di mana kita dapat belajar sebuah generalisasi.

Contoh:



Kita dapat membentuk deskripsi kelas untuk prosedur-prosedur, kemudian kita dapat beradaptasi terhadap prosedur-prosedur yang

tergeneralisasi ke berbagai situasi sejenis. Proses ini adalah kemampuan untuk menggeneralisasi prosedur yang membedakan manusia dari robot. Sebuah robot hanya dapat mengenal proses tertentu , tetapi tidak dapat menggeneralisasi. Kemampuan kita untuk belajar sebuah klasifikasi tidak terbatas pada obyek atau prosedur, tetapi dapat diterapkan pada ide dan konsep.

Kemampuan untuk mempelajari deskripsi kelas adalah dasar pembentukan sebuah komputer di mana berfikir seperti cara manusia lakukan.

8.2. Bagaimana deskripsi kelas dipelajari?

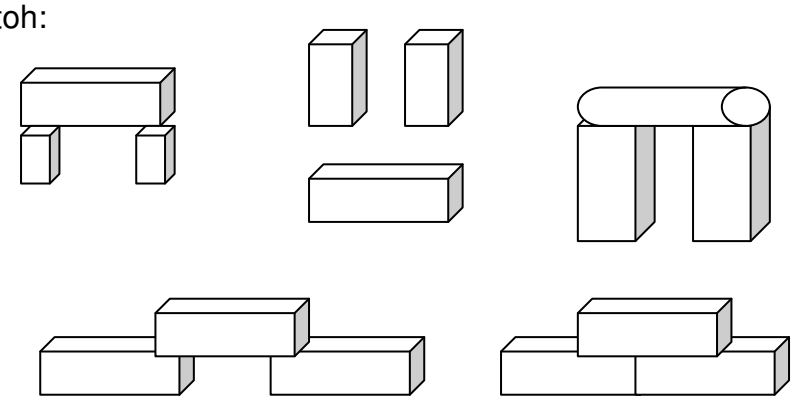
Kesulitan terbesar untuk menangani bila mencoba untuk membentuk sebuah intelligent computer, di mana kenyataannya bahwa kita sendiri memiliki sedikit pengertian tentang proses pemikiran manusia.

Pada umumnya, kita tidak memiliki ide bagaimana kita dapat membentuk deskripsi kelas tergeneralisasi dari obyek tertentu. Namun demikian tentunya percobaan untuk menemukan sebuah cara untuk menggantikan tipe *learning* ini dalam sebuah komputer, kadang melihat ke dalam pemrosesan pemikiran manusia.

Prof. Patrick Henry Winston, direktur laboratorium AI di MIT membuat solusi umum yang mengarah ke pengertian *cognitive learning*. Metoda *learning*nya tentang deskripsi kelas disebut:

"*hit-and-near-miss*"

Contoh:



Deskripsi kelas gapura:

1. harus memiliki sebuah blok atau cilinder pada atapnya dari dua blok lain yang tidak bersentuhan.
2. blok pendukung kemungkinan berdiri tegak atau tergeletak.

Procedure '***hit-and-near-miss***' memiliki 2 asumsi:

1. Terdapat petunjuk yang menampilkan salah satu contoh bagian dari kelas atau 'near-misses', dan tidak pernah salah tentang contoh-contoh tersebut.
2. Contoh peralatan harus contoh valid karena contoh tersebut satu dari bentuk-bentuk model awal.

Contoh:

Dari asumsi-asumsi di atas, maka algoritma *hit-and-near-miss*

```
observe the sample and form the initial model
repeat
        observe sample
        if hit the generalize
        else restrict
until done
```

Pada algoritma di atas, terdapat hal-hal penting yaitu:

1. Bagaimana model digeneralisasi
2. Bagaimana model dibatasi

Procedure Restrict :

```
determine the difference between the near-miss and the evolving model
if the model has an attribute not found in the near-miss,
        then require this attribute
if near-miss has an attribute not found in the model, then
        forbid this attribute
```

Procedure Generalize :

determine the difference between example and the evolving model

Reconcile the difference by enlarging the model

Dua prinsip penting yang harus dilibatkan dalam procedure *'hit-and-near-miss'* untuk menuingkatkan efisensi dan keandalannya, yaitu:

1. Disebut *'no-guessing'* dilibatkan, ketika komputer tidak dapat melihat perbedaan antara model yang benar dengan model yang tidak benar, yaitu bila terjadi suatu pengakuan terhadap obyek yang tidak, maka sesungguhnya komputer tidak mempelajari sesuatu.
2. Disebut : *'no-altering'*, jika petunjuk mendukung obyek yang benar, tetapi gagal untuk memadankan ke definisi yang ada, maka komputer membentuk secara terpisah klasifikasi kasus khusus dari pada mencoba untuk memperbesar klasifikasi yang ada untuk mencakup obyek tersebut.

8.3. *Knowledge Representation*

Masalah sekunder yang juga menjadi bahan pertimbangan adalah: bagaimana caranya untuk menyimpan pengetahuan yang dibutuhkan ke dalam komputer baik pengetahuan alami dan pemilihan pemrograman menentukan cara di mana pengetahuan dapat ditunjukkan.

Teknik-teknik untuk mengembangkan lapangan *knowledge representation*:

1. Trees
2. Lists
3. Networks

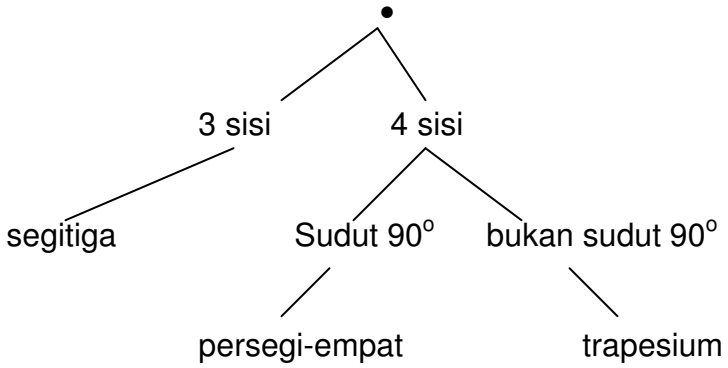
1. *Trees*

Bila kita berpikir kembali pada metoda *backward chainning* dalam *expert system*, dapat dilihat dengan jelas bahwa cara paling efisien untuk menunjukkan pengetahuan untuk tipe *expert system* adalah *tree*.

Kemajuan *expert system* melalui *tree*, akan menyingkat bagian besar dan menemukan *goal* secara cepat.

Expert system yang menggunakan *tree* pengetahuan akan selalu menanyakan user-nya untuk menjawab pertanyaan yang relevan. Secara alami, *tree* adalah hirarki, sehingga kemungkinan hanya digunakan untuk menyimpan pengetahuan hirarki, sehingga *tree* bukan merupakan batasan yang baik, karena banyak pengetahuan gagal ke dalam kategori yang berada dalam *tree*.

Gambar berikut ini, *tree* digunakan untuk menyimpan informasi tentang persegi-empat, segitiga, dan trapesium.



Kerugian terbesar dari penggunaan *tree*:

Kesulitan pembentukan dan pemeliharaan tree sehingga meninggalkan efisiensi.

2. *LIST*

List penting untuk keberhasilan pemrograman Turbo Prolog, dimana list selalu menggunakan metoda tradisional AI untuk *knowledge representation*.

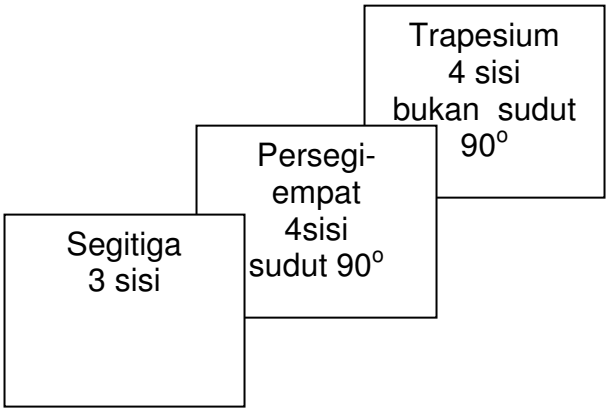
Contoh: Bahasa AI pertama LISP(*LISt Processing*) dirancang untuk menangani *list* secara efisien.

Fungsi dari *list,* sehingga sangat menarik untuk *knowledge representation* adalah: mudah dilakukan (dikerjakan) dalam prolog. Untuk

mengerti bagaimana *knowledge* ditunjukkan dalam *list* dapat diberikan contoh :

Sebuah kartu katalog dalam perpustakaan. Jika kita ingin mencari sebuah buku tentang topic tertentu maka kita harus mencari setiap kartu, tolak kartu yang tidak sesuai dan mencari kartu-kartu lain, kemudian berhenti bila menjumpai buku yang diinginkan.

Dengan kata lain bahwa *knowledge* disimpan sebagai sebuah *list* yang membutuhkan pencarian sekuensial untuk menemukannya.



Walaupun *list* hanya dapat diproses secara *sequential*, tetapi *list* sangat penting karena kita dapat menggunakannya untuk menunjukkan sesuatu dan seluruh tipe k*nowledge*.

List sangat fleksibel di mana membuat *list* sangat penting untuk AI sebagai alternatif utama. Dengan menggunakan berbagai skema index, kita dapat membuat *list* hampir seefesien *tree*.

3. *Network*

Representasi *knowledge* sebagai sebuah *network* adalah mengagumkan dan sangat berguna, juga bentuk *network* sangat kompleks.

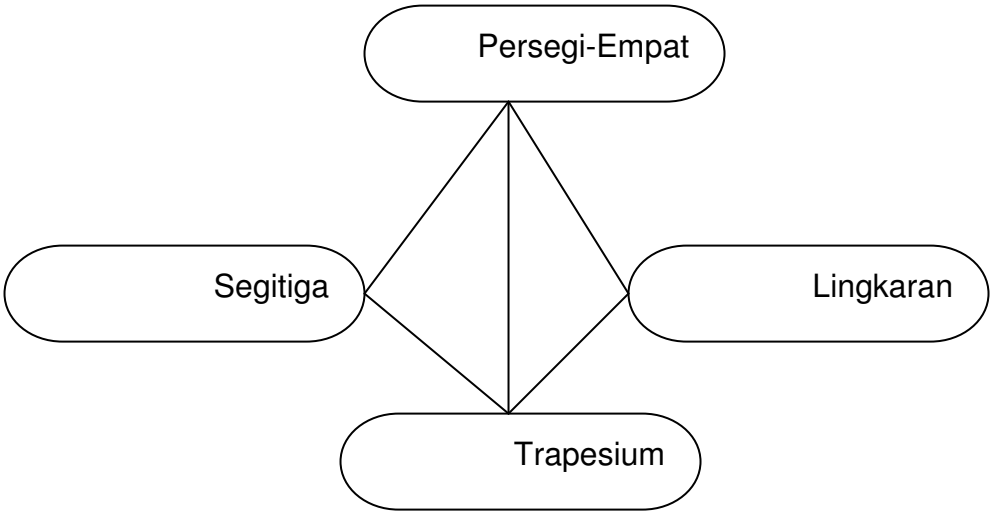
Representasi *knowledge* dimungkinkan di mana representasi *network* dari *knowledge* akan distandarisasi.

Network Knowledge didasari pada 2 kondisi:

1. *Knowledge* dalam *network* ditunjukkan oleh node-node dalam sebuah *graph non* hirarki, tidak seperti *tree*, semua node dalam

network memiliki kepentingan yang sama dan salah satu dari node-node tersebut dapat digunakan sebagai posisi awal.

2. Node-node diatur sehingga tipe *knowledge* sejenis dikelompokkan dekat satu sama lainnya, yaitu *adjacent no*de memiliki hubungan *'near-miss'*



Kita mengakses model *network* dengan memasukkan *network* pada sebuah posisi yang tepat dan kemudian memproses sepanjang sampai node sesungguhnya dicapai.

Secara teori, metoda ini seharusnya sangat efisien, karena setiap perpindahan ke node baru dibuat karena transisi(perpindahan) berada pada arah pengembangan yang sejenis, di mana merupakan sebuah bentuk *hill climbing*. Walaupun prosedur ini akan bekerja tanpa masalah di mana kita masuk ke dalam *network*, tetapi ada baiknya untuk masuk pada sebuah node yang agak mendekati *goal*. Kebanyakan model-model *network* juga berisi *list index* yang membantu memilih node masukkan untuk setiap situasi.

Kejadian terburuk :

Jika tiba-tiba kita memilih node yang jauh dari sejenis, maka *network* mengacak ke dalam *lis*t. Keuntungan metoda *network* dari *knowledge representation* adalah metoda ini dapat dengan mudah dan efisien menangani baik *knowledge* hirarki dan nonhirarki. *Knowledge*

nonhirarki akan mencoba dari sebelah luar batasan *network*, dengan *knowledge* hirarki yang dilokasikan lebih dekat dengan pusat.

Representasi *Knowledge*(Pengetahuan)

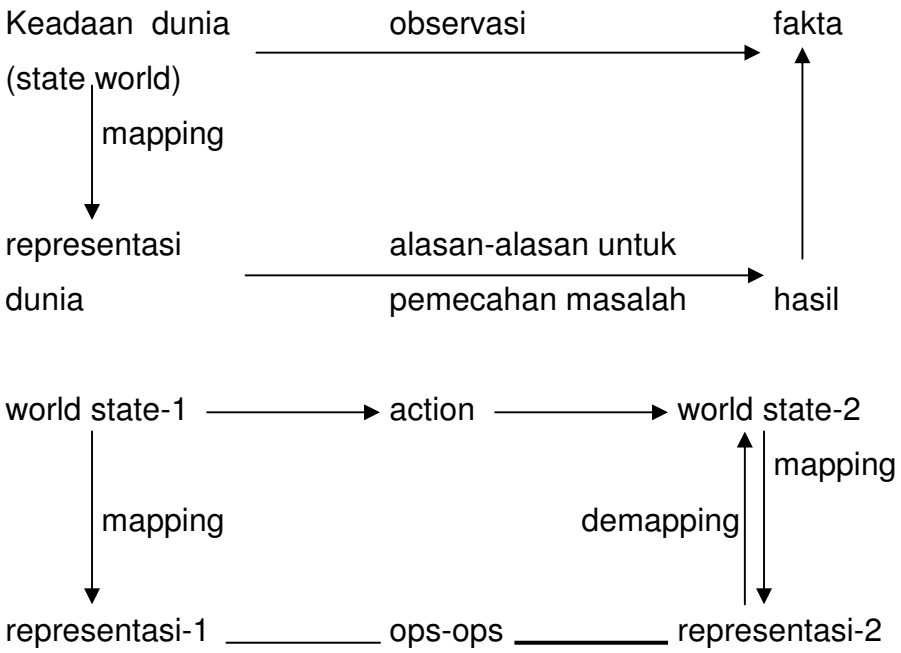
Knowledge adalah segala sesuatu yang digunakan oleh pemecah persoalan untuk memecahkan persoalan.

Understanding: kemampuan menggunakan *knowledge*(pengetahuan) kita untuk memproduksi (meramal) efek-efek dari suatu kegiatan (aksi) di dunia atau untuk memproduksi hasil dari suatu observasi.

Understanding Object X berarti :

Mempunyai pengetahuan (*knowledge*) tentang X:

- deskripsi dari X
- hak-hak / kewajiban X
- operasi-operasi terhadap X



Hal-hal yang harus dipertimbangkan atau ada dalam *knowledge*:

1. Objek :
   - deskripsi
   - hak-hak/kewajiban
   - operasi-operasi
2. Kejadian-kejadian
3. *Meta-knowledge* : *knowledge about knowledge* (mengetahui apa yang anda tahu atau tidak tahu).

   Misal: 100! --->kita dapat mengerjakan ini kalau cukup waktu dan kertas/pensil.
4. Prosedur-prosedur *knowledge (procedural Knowledge*) Prosedur-prosedur tentang bagaimana menyelesaikan persoalan dan bagaimana melaksanakan pekerjaan tersebut. (misal: bagaimana naik sepeda, bagaimana menerbangkan pesawat, dan sebagainya). Proses-proses *Knowledge*:
   - *Acquisition* : mendapatkan dan mengintegrasikan *knowledge*
   - *retrieval* : mencari *knowledge-knowledge* yang relevan
   - *interence/reasoning* : mencari *knowledge-knowledge* baru dari *knowledge* ke dalam *knowledge.*

     misal :
     1. Bessie adalah sapi.
     2. Sapi adalah binatang memamah biak.
     3. Memamah biak adalah mengunyah makanan dari perut

     Apakah Bessie mengunyah makanan dari perut?

   ◊ *Matching* : pola p1 serupa atau sama dengan pola p2

   2 macam *matching*:

   a. *identity matching* : p1 = p2
   b. *instance matching* :apakah p1 suatu keadaan/instance dari p2

Tipe-Tipe Representasi :

1. *Analogue representations* (representasi-analog) representasi internal secara struktur sama dengan sesuatu yang sedang direpresentasikan. Misal : *8 puzzle geometry theorem prover signal processing*

2. Table-table data :
   - *Relational Data Base*
   - *Samuel Rate Learning Experience*

3. *Logic* :
   - Bessie adalah sapi $\longrightarrow$ Bessie E sapi
   - semua sapi memamah biak

$$\forall x \, [\, x \in \text{sapi} \Rightarrow \text{kunyah} \, (x, \text{cud})]$$

4. *Production Rule* : if P1 and P2 and … Pn

then → *conclude* Pn+1
then → *do action* A

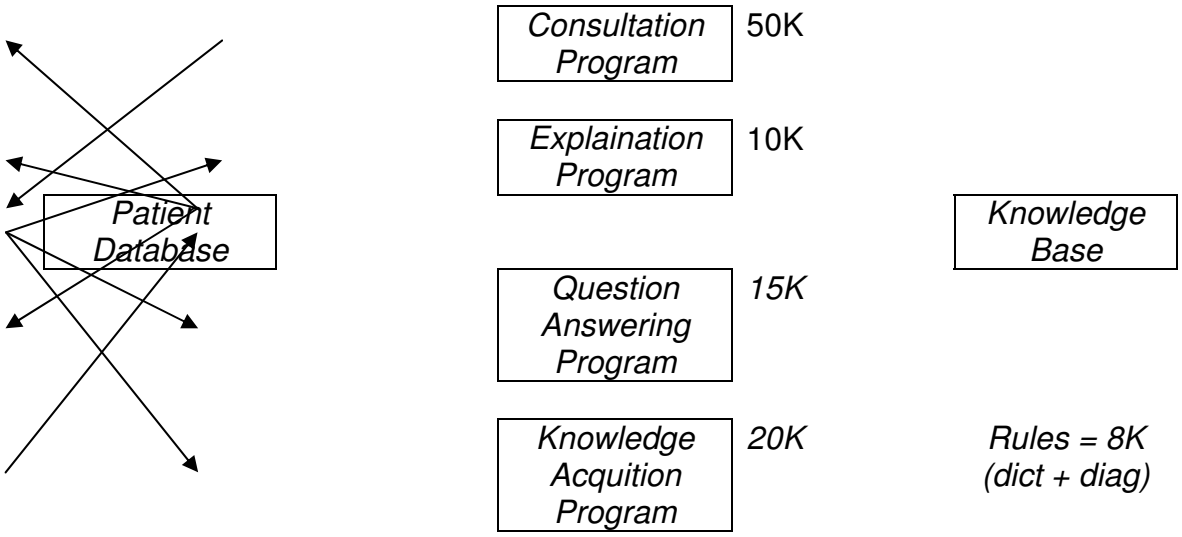
5. *Procedural Representations*:

```
Program chew-cud-test(x)
tentukan apakah/bila x adalah sapi
adalah binatang memamah biak
        bila ya return(true)
        else return(don't know)
```

6. Semantik *Network* : *Network Data Base*

7. Frames, Scripts

Contoh : MYCIN memiliki enam komponen :



Pengobatan Infeksi

Rules : 200 *production rules* → *Premise*, *Action*

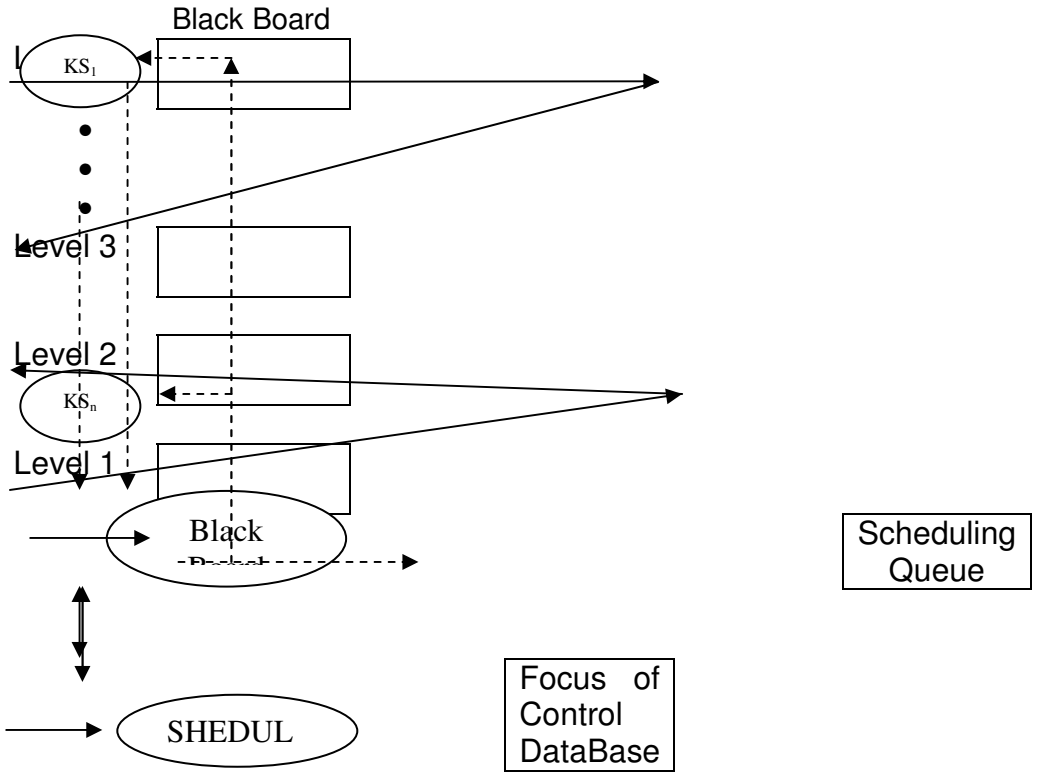
Misal: Premise ($and (same contxt infect primary bacterimia)
(membf contxt site stenilesites)
(same contxt portal GI))

Action (conclude contxt ident bacteroides tallys)

Bila: 1). infeksi adalah *primary bacterimia*, dan
2). sebab lain salah satu dari *sterilesitas*, dan
3). hasil atau efek organ tubuh adalah *Gastro Intestival*(GI) *tract*, kemudian kemungkinan identitas organ penyakit *bacteroides*

Knowledge System

Skema arsitektur Hearsay-II



Keterangan gambar skema arsitektur Hearsay-II:

KS : *Knowledge System*

: Database(DB)

: Modul-modul Program

Alur data

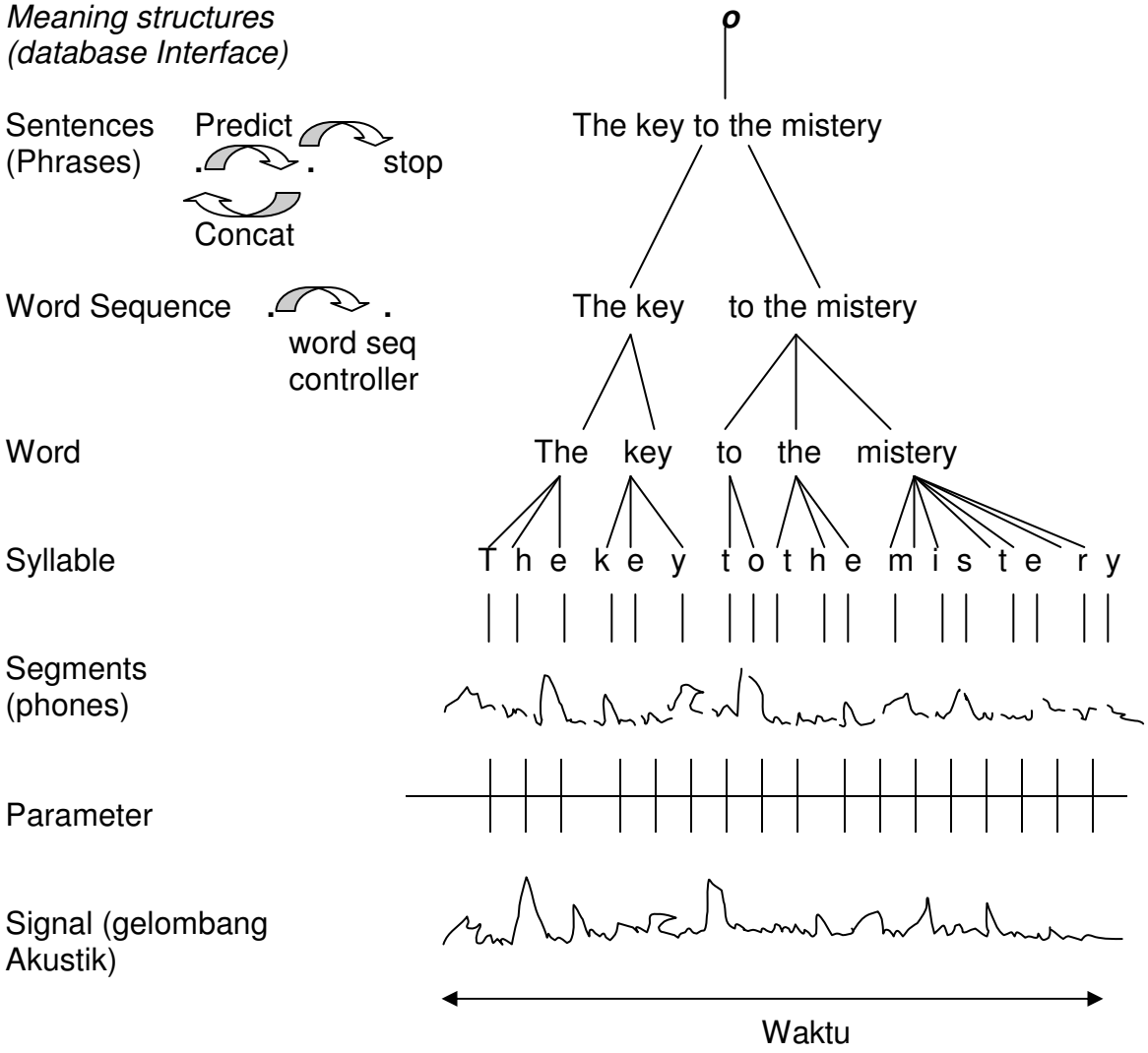
Pengendali Alur

Contoh lain dalam pengembangan *Knowledge System* :

- o HWIN (Hear What I Say)
- o SRI (Walk78, Walk80)
- o HARRY(Love78, Love80)

misal : buatkan struktur Black Board untuk "*The key to the mistery*"

STRUKTUR BLACK BOARD



BAB IX
Neural Network

Jaringan syaraf, bersama sistem pakar dan perangkat lunak, ternyata memberikan solusi persoalan dunia industri, telekomunikasi dan informasi dengan berbagai aplikasi masa kini maupun masa depan. Dalam bidang ilmu pengetahuan jaringan syaraf(*neural network*) sudah sejak lama dibicarakan banyak orang. Mulai dikenal akhir tahun 1940-an, jaringan syaraf masuk dalam blok perkembangan teknologi komputer. Meski begitu, anehnya perkembangan teknologi komputer itu pulalah yang jadi penghambat berkembangnya ilmu jaringan syaraf. Lihat saja, meski riset dan pengembangan teknologi komputer terus berjalan, jaringan syaraf kurang begitu mendapat perhatian.

Ternyata kini *neural network* dapat menjawab beberapa persoalan dunia telekomunikasi, industri maupun informasi, yang tak terlintas sebelumnya. Dan melihat prospeknya di masa depan, para ahli yang sangat fanatik pada model komputer digital konvensional, boleh jadi berbalik menyesali diri.

Jaringan syaraf adalah sistem pengolahan informasi yang didasari fisolofi struktur perilaku syaraf makhluk hidup. Dengan begitu, jaringan syaraf tak diprogram selayaknya mekanisme pada komputer digital konvensional. Begitu juga dari segi arsitekturnya. Dalam arsitekturnya, jaringan syaraf mempelajari bagaimana menghasilkan keluaran yang diinginkan pada saat diberikan sekumpulan masukan. Proses ini dilakukan secara internal, yaitu dengan memerintahkan sistem untuk mengidentifikasikan hubungan antar masukan kemudian mempelajari respon tersebut. Dengan metoda pensintesisan hubungan, jaringan syaraf dapat mengenal situasi yang sedang dan telah dijumpai sebelumnya.

Berbeda dengan proses internal, proses eksternal lebih tergantung pada aplikasinya. Sistem bisa menggunakan umpan balik eksternal atau sinyal tanggapan yang diinginkan, untuk membentuk prilaku jaringan. Ini disebut sebagai

supervised learning. Dengan cara lain, jaringan dapat membangkitkan sinyal tanggapan yang diinginkan sendiri dalam skenario *unsupervised learning*.

Latar belakang dikembangkannya *neural network*, itu karena pada pemrograman beberapa aplikasi seperti *image recognition*(pengenalan citra), *speech recognition*(pengenalan suara), *weather forecasting*(peramalan cuaca) ataupun permodelan tiga dimensi, tak dapat dengan mudah dan akurat diterapkan pada set instruksi komputer biasa. Atas dasar itu, maka diterapkan arsitektur komputer khusus yang dimodel berdasar otak manusia.

Analoginya, otak manusia terdiri dari ratusan milyar(1011) *neuron*. *Output* dari *neuron* akan menjadi input bagi puluhan *neuron* lain melalui tali penghubung, sinapsis. komputer jaringan syaraf tak diprogram seperti komputer digital biasa, namun harus dilatih pendesainnya. Tak juga seperti pemrograman sistem pakar (*expert system*) dengan serangkaian aturan serta basis data(*database*), jaringan syaraf diprogram untuk mempelajari tingkah laku yang diinginkan lingkungan.

Karena itu, kita dapat melihat bahwa jaringan syaraf mempunyai kelebihan memecahkan masalah teknis. Yaitu: pertama, jaringan syaraf Őtak perlu pemrograman tentang hubungan *input* dan *output*. Melainkan, akan mempelajari sendiri respon yang diinginkan dengan cara pelatihan. Ini sangat penting guna menghilangkan sebagian besar biaya pemrograman.

Kedua, jaringan syaraf dapat memperbaiki respon dengan belajar. Itu karena jaringan syaraf didesain untuk mengevaluasi dan beradaptasi terhadap kriteria-kriteria respon yang baru.

Sedang ketiga, karena jaringan syaraf bekerja sebagai penjumlah semua sinyal input, input tidak harus sama. Ini artinya, jaringan syaraf akan dapat mengenali seseorang meski orang tersebut sudah berbeda dengan saat dikenali pertama kali. Atau jaringan syaraf akan mengenali suatu kata, meski kata itu diucapkan oleh orang yang berbeda-beda. Semua ini tentunya sangat sulit dikerjakan oleh teknik komputer digital biasa.

9.1. Implementasi

Perkembangan *neural network* saat ini, cukup menggembirakan. Jaringan syaraf, bersama sistem pakar dan perangkat lunak, ternyata memberikan ásolusi

persoalan dunia industri, telekomunikasi dan industri dan informasi dengan berbagai aplikasi masa kini maupun masa depan.Untuk implementasi *neural network* pada telekomunikasi, diantaranya adalah pemampatan citra, pengolahan sinyal, pemfilteran derau dan *routing* trafik.

9.2. Pemampatan Citra

Telah dimanfaatkan banyak orang untuk menghasilkan pengkodean data citra yang efisien. Nilai intensitas(*gray level*) setiap elemen gambar(*pixel= picture element*) sebuah citra diperlihatkan secara khas menggunakan satu *byte memori* komputer. Biasanya citra tersebut terdiri dari kurang 256 x 256 *pixel*, sehingga untuk menampilkan sebuah citra secara digital diperlukan sekitar 65.000 *byte memory*.

Untuk menampilkan citra tersebut tidak hanya memerlukan sejumlah memori saja, namun juga masalah pengiriman data citra melalui pita transmisi yang terbatas seperti pada saluran telepon. Pemampatan citra mengacu pada pengubahan data citra ke bentuk tampilan berbeda yang hanya memerlukan sedikit memori, namun bentuk citra asal dapat direkonstruksi kembali. Sistem menggunakan tiga lapis jaringan syaraf yang telah dibangun dengan mengatur peta pengkodean dan peta rekonstruksi secara parallel. Sistem demikian diselesaikan dengan perbandingan pamampatan 8:1.

9.3. Pengolahan Sinyal

Dalam mengupas sistem pengolahan sinyal, dilakukan estimasi jaringan perambatan balik (*back propagation*) untuk melakukan prediksi serta permodelan simulasi. Dalam permodelan tersebut diperlihatkan bahwa deretan waktu chaotis, perambatan balik melampaui metoda polinomial prediktif dan linier konvensional dengan berbabunyi yang dimiliki dengan melakukan pendekatan untuk menghasilkan deret elemen secara matematis.

9.4. Pemfilteran Derau

Jaringan syaraf dapat juga digunakan untuk melakukan pemfilteran derau. Jaringan ini mampu mempertahankan struktur lebih baik dan lebih áseksama dibanding dengan filter-filter biasa yang hanya mampu menghilangkan derau saja.

9.5. Routing Trafic

Ini penting untuk sistem telekomunikasi. Pada *routing node to node* konvensional, akan ada usaha minimalisasi fungsi *loss*, yakni jumlah total *link*/hubungan dan waktu tunda. Fungsi *loss* dibuat agar mendasar sebagai trafik aktual mendekati kapasitas. Berdasarkan proposional untuk *delay* rata-rata per message pada sebuah hubungan, fungsi *loss infinity* ternyata lebih sulit dijalankan komputer.

Dengan kehadiran jaringan syaraf, *routing trafic* akan dapat meminimumkan parameter yang menghambat, seperti waktu tunda dan banyak hubungan yang harus dilalui. Untuk memperkecil *delay* yang terjadi, jaringan syaraf tidak memakai algoritma atau tabel *routing* seperti pada sistem konvensional. Oleh sebab itu pula jaringan ini disebut kelas pengolah informasi non algoritmis. Namun begitu, jaringan syaraf ini dapat dikelompokkan sebagai algoritma terdistribusi tanpa menggunakan tabel-tabel *routing*.

9.6. Tantangan Masa Depan

Perkembangan masalah yang makin kompleks di bidang telekomunikasi, industri dan informasi, menuntut kemampuan yang luar biasa terhadap kehandalan teknik perangkat lunak tradisional. Pada jaringan telekomunikasi, kemampuan algoritma jaringan syaraf dapat memecahkan masalah routing dan dapat mengantisipasi respon dinamis dari kondisi trafik yang diukur sistem jaringan yang telah diselidiki secara periodik.

Dengan rekayasa perangkat lunak, di masa datang diharapkan perangkat lunak konvensional, sistem pakar dan jaringan syaraf terintegrasi sehingga dihasilkan perangkat lunak baru yang dapat memaksimumkan efektifitas sistem jaringan telekomunikasi termasuk perangkat terminal dan peralatan sentral.

Sehingga nantinya, peralatan telekomunikasi mengerti kehendak pemakai untuk melakukan hubungan komunikasi, tanpa harus mengingatkan peripheralnya.

Untuk mewujudkan itu semua perlu dilakukan pengkajian dan penelitian menerus agar segera mengimplementasikan jaringan syaraf dalam bidang telekomunikasi.

DAFTAR PUSTAKA

1. Rich, Elaine, and Knight, Kevin, "*Artificial Intelligence*", 2nd Edition, Mc.Graw-Hill, New York, 1991.

2. Charniak, Eugene, and Mc.Demott, Drew, "*Introduction To Artificial Intelligence*", Second Edition, Addition-Wesley, 1985.

3. F. Luger, George, and A. Stubblefield, William, "*Artificial Intelligence: Structured and Strategies for Complex Problem Solving*", Second Edition, The Benjamin/Cumming Publishing Company, Inc., California, 1993.

4. J. Schalkoff, Robert, "*Artificial Intelligence: An Engineering Approach*", Mc.Graw-Hill, New York, 1990.

5. Dean, Thomas, Allen, James, and Aloimonos, yiannis, "*Artificial Intelligence: Theory and Practice"*, The Benjamin/Cumming Publishing Company, Inc., California, 1995.
6. L. Dym, Clive & E. Levitt, Raymond, "*Knowledge Based Systems in Engineering*", Mc. Graw-Hill, Singapore,1991.

7. Parsaye, Kamran & Chignell, Mark, "*Expert Systems For Experts*", John Wiley & Sons Inc, England,1988.

8. Schildt, Herbert, "*Artificial Intelligence Using C*", Mc.Graw-Hill, Singapore, 1987.

Tujuan Kuliah Sistem Berbasis Pengetahuan:

Memberikan pengetahuan kepada mahasiswa tentang ruang lingkup dan bidang penerapan dari AI, sehingga setelah mengikuti matakuliah ini, maka mahasiswa mempunyai pengetahuan tentang representasi internal, *vision, natural language processing, search*, logika, expert *systems*, robotik, dan *machine learning*.

Khususnya :

1. Membekali bentuk-bentuk sederhana dari AI.
2. Membekali konsep-konsep yang membedakan teknik-teknik AI dengan teknik-teknik pemecahan persoalan yang lain.
3. Membiasakan diri dengan alat-alat AI(bahasa pemrograman yang digunakan di AI).
4. Membiasakan diri dengan arsitektur khusus untuk AI.
5. Membuat catatan dari perdebatan utama dan penelitian dalam AI dan dalam area teman sejawat.