Pengantar Kecerdasan Buatan(AI)

Sub-topik:

1. Konsep dasar dan pengertian AI
2. Asumsi Dasar AI
3. Perbedaan antara Pemrograman AI dan konvensional
4. Bidang-bidang aplikasi AI

Pengertian AI :

1. Suatu cara yang sederhana untuk membuat komputer dapat "berpikir" secara inteligent
2. Bagian dari ilmu komputer yang mempelajari perancangan sistem komputer yang inteligent, yaitu suatu sistem yang meperlihatkan karakteristik yang ada pada tingkah laku manusia, seperti mengerti suatu bahasa, mempelajari,mempertimbangkan dan memecahkan suatu masalah.
3. Suatu studi bagaimana membuat komputer dapat mengerjakan sesuatu, yang pada saat ini, orang dapat mengerjakan lebih baik
4. Bidang ilmu komputer yang memungkinkannya untuk memahami, bernalar dan bertindak.

Tujuan AI:

1. Untuk mengembangkan metode dan sistem untuk menyelesaikan masalah,masalah yang biasa diselesaikan melalui aktifivitas intelektual manusia, misalnya pengolahan citra,perencanaan, peramalan dan lain-lain, meningkatkan kinerja sistem informasi yang berbasis komputer.
2. Untuk meningkatkan pengertian/pemahaman kita pada bagaimana otak manusia bekerja

Arah AI:

1. Mengembangkan metode dan sistem untuk menyelesaikan masalah AI tanpa mengikuti cara manusia menyelesaikannya (sistem pakar / expert systems)
2. Mengembangkan metode dan sistem untuk menyelesaikan masalah AI melalui pemodelan cara berpikirnya manusia, atau cara bekerjanya otak manusia (neural networks).

Paradigma AI:

symbolic or sub-symbolic (connectionist)

Bidang-bidang Aplikasi AI

Menurut Elaine Rich : Masalah pada AI (Task Domain) :

Task Keduniaan (*Mundane Task*)

* *Perception: Vision, Speech Recognition*
* *Natural Language: Understanding, Generation, Translation*
* *Commonsense Reasoning*
* *Robot Control*

Task Formal (Formal Task)

* Games (Chess, Backgamon, checkers,Go)
* Mathematics (Geometry, Logic, Integral Calculus, Proving properties of programs)

Task Ahli (Expert Tasks)

* Engineering (Design,Fault Finding, Manufacturing Planning)
* *Scientific Analysis, Medical Diagnosis, Financial Analysis*

Asumsi Dasar AI

Jantung Penelitian AI adalah : *Physical Symbol System Hypothesis.*

Physical Symbol System terdiri dari himpunan entitas yang dinamakan simbol, berpola fisik yang dapat menjadi komponen dari entitas tipe lain yang dinamakan Ekspresi (struktur simbol)

Struktur simbol terdiri dari sejumlah *instant* (*Token*) dari simbol-simbol yang berhubungan pada beberapa cara fisik. Selain struktur tersebut, sistem juga berisi koleksi proses-proses yang beroperasi pada ekspresi, untuk menghasilkan ekspresi lain : proses pembuatan (*create*), modifikasi, reproduksi, dan penghancuran (destruksi).

Jadi PSS adalah mesin yang memproduksi suatu koleksi penyusunan struktur simbol. Sistem seperti itu terdapat dalam suatu objek dunia yang lebih luas dari hanya ekspresi simbolik itu sendiri.

Perbedaan antara Pemrograman AI dan Konvensional

AI	**Komputasi Konvensional**
Representasi dan Manipulasi simbol	Algoritama
Memberitahu komputer tentang suatu masalah	Memerintah komputer untuk menyelesaikan masalah
Komputer diberi pengetahuan dan kemampuan inferensi	Memberi data kepada komputer dan program

Pemrograman AI :

- Bila terjadi perubahan dalam program, maka tidak mengganggu seluruh “Facts” yang tersimpan dalam “Otak” (layaknya pikiran manusia/seperti informasi yang terdapat pada pikiran manusia)
- Independen
- Dapat Dimodifikasi tanpa mempengaruhi struktur kesluruhan program
- Fleksibel → efisien dan mudah untuk dimengerti

II. Penyelesaian Masalah berdasarkan teknik AI

Empat hal untuk membangun sistem atau memecahkan masalah tertentu :

1. Definisikan masalah dengan jelas
2. Analisis masalah
3. Kumpulkan dan representasikan knowledge
4. Pilih teknik pemecah masalah terbaik dan gunakan untuk masalah tertentu

Mendefinisikan Masalah sebagai "State Space Search" (SSS)

Misalnya permainan catur , maka SSS nya adalah :
Menspesifikasikan posisi awal dari papan catur
Peraturan (rules) yang mendefinisikan langkah-langkah yang legal
Posisi papan yang merepresentasikan pemenang dari satu sisi atau sisi lainnya.

Tujuan (*Goal*) dari permainan adalah : memenangkan permainan.

Pendefinisian Masalah Sebagai Pencarian Ruang Keadaan

Masalah utama dalam membangun sistem berbasis AI adalah bagaimana mengkonversikan situasi yang diberikan ke dalam situasi lain yang diinginkan menggunakan sekumpulan operasi tertentu.

A Water Jug Problem

Anda diberi dua buah gelas, yang satu ukuran 4 galon dan yang lain 3 galon. Kedua gelas tidak memiliki skala ukuran. Terdapat pompa yang dapat digunakan untuk mengisi gelas dengan air. Bagaimana anda mendapatkan tepat 2 galon air di dalam gelas 4 ukuran galon?
Ruang masalah untuk masalah di atas dapat digambarkan sebagai himpunan pasangan bilangan bulat (x,y) yang terurut, sedemikian hingga x = 0, 1, 2, 3, atau 4 dan y = 0, 1, 2, atau 3; x menyatakan jumlah air dalam gelas ukuran 4 galon, dan y menyatakan jumlah air dalam gelas ukuran 3 galon. Keadaan mula-mula adalah (0,0). *State* tujuan adalah

(2,n) untuk setiap nilai n.

Operator-opeartor (aturan produksi) yang digunakan untuk memecahkan masalah terlihat pada gambar 2.2.

1.	(x,y) If x < 4	→	(4,y)	Isi penuh gelas 4 galon
2.	(x,y) If y < 3	→	(x,3)	Isi penuh gelas 3 galon
3.	(x,y) If x > 0	→	(x-d,y)	Buang sebagian air dari gelas 4 galon
4.	(x,y) If y > 0	→	(x,y-d)	Buang sebagian air dari galon ukuran 3 galon
5.	(x,y) If x > 0	→	(0,y)	Kosongkan gelas 4 galon
6.	(x,y) If y > 0	→	(x,0)	Kosongkan gelas 3 galon
7.	(x,y) If x+y ≥4 and y > 0	→	(4,y-(4-x))	Tuangkan air dari gelas 3 galon ke gelas 4 galon sampai gelas 4 galon penuh
8.	(x,y) If x+y ≥3 and x > 0	→	(x-(3-y),3)	Tuangkan air dari gelas 4 galon ke gelas 3 galon sampai gelas 3 galon penuh
9.	(x,y)	→	(x+y,0)	Tuangkan seluruh air dari

	If x+y ≤4 and y > 0			gelas 3 galon ke gelas 4 galon
10.	(x,y) If x+y ≤3 and x > 0	→	(0,x+y)	Tuangkan seluruh air dari gelas 4 galon ke gelas 3 galon
11.	(0,2)	→	(2,0)	Tuangkan 2 galon air dari gelas 3 galon ke gelas 4 galon
12.	(2,y)	→	(0,y)	Buang 2 galon dalam gelas 4 galon sampai habis.

Gambar 2.2 Aturan produksi untuk *Water Jug Problem*.

Jumlah galon dalam gelas 4 galon	**Jumlah galon dalam gelas 3 galon**	**Aturan yang dilakukan**
0	0	-
0	3	2
3	0	9
3	3	2
4	2	7
0	2	5 atau 12
2	0	9 atau 11

Gambar 2.3 Suatu solusi untuk *Water Jug Problem*.

Karakteristik Masalah Dalam AI :

- Apakah masalahnya dapat didekomposisi menjadi himpunan sub masalah yang (hampir) independen lebih kecil atau lebih mudah ?
- Dapatkah langkah penyelesaian diacuhkan paling tidak dibatalkan ketika dapat dibuktikan hal tersebut tidak bijaksana ?
- Apakah universe masalahnya dapat diprediksi ?
- Apakah solusi yang baik dari masalah tertentu jelas tanpa membandingkan dengan seluruh solusi lain yang mungkin ?
- Apakah solusi yang diinginkan sebuah keadaaan dari dunia atau sebuah jalur dari keadaan ?
- Apa peran dari pengetahuan ?
- Apakah pekerjaan memerlukan interakasi dengan manusia ?

Sistem Produksi

Sistem produksi terdiri dari:

- **Himpunan aturan**, masing-masing terdiri dari sisi kiri (pola) yang menentukan kemampuan aplikasi dari aturan tersebut dan sisi kanan yang menggambarkan operasi yang dilalukan jika aturan dilaksanakan.
- Satu atau lebih **pengetahuan** atau basis data yang berisi informasi apapun untuk tugas tertentu. Beberapa bagian basis data bisa permanen, dan bagian yang lain bisa hanya merupakan solusi untuk masalah saat ini. Informasi dalam basis data ini disusun secara tepat.
- **Strategi kontrol** yang menspesifikasikan urutan dimana aturan akan dibandingkan dengan basis data dan menspesifikasikan cara pemecahan masalah yang timbul ketika beberapa aturan sesuai sekaligus pada waktu yang sama.
- **A rule applier** (*pengaplikasi aturan*).

Strategi Kontrol

Syarat-syarat strategi kontrol:

- ***cause motion.*** Perhatikan kembali *water jug problem*. Jika kita mengimplementasikan strategi kontrol sederhana dengan selalu memilih aturan pertama pada daftar 12 aturan yang telah dibuat, maka kita tidak akan pernah memecahkan masalah. Strategi kontrol yang tidak menyebabkan ***motion*** tidak akan pernah mencapai solusi.
- ***Systematic.*** Strategi kontrol sederhana yang lain untuk *water jug problem*: pada setiap siklus, pilih secara random aturan-aturan yang dapat diaplikasikan. Strategi ini lebih baik dari yang pertama, karena menyebabkan *motion*. Pada akhirnya strategi tersebut akan mencapai solusi. Tetapi mungkin kita akan mengunjungi beberapa *state* yang sama selama proses tersebut dan mungkin menggunakan lebih banyak langkah dari jumlah langkah yang diperlukan. Hal ini disebabkan strategi kontrol tersebut tidak sistematik. Beberapa strategi kontrol yang sistematik telah diusulkan, yang biasa disebut sebagai metoda-metoda dalam teknik *searching*. Di bab ini, akan dibahas enam metoda, yaitu *Breadth First Search*, *Uniform Cost Search*, *Depth First Search*, *Depth-Limited Search*, *Iterative-Deepening Depth-First Search*, dan *Bi-directional search*. Masing-masing metoda tersebut mempunyai karakteristik yang berbeda.

Strategi Pencarian

Terdapat empat kriteria dalam strategi pencarian, yaitu:

- **Completeness:** Apakah strategi tersebut menjamin penemuan solusi jika solusinya memang ada?
- **Time complexity:** Berapa lama waktu yang diperlukan?
- **Space complexity:** Berapa banyak memori yang diperlukan?
- **Optimality:** Apakah strategi tersebut menemukan solusi yang paling baik jika terdapat beberapa solusi berbeda pada permasalahan yang ada?

Depth-First Search (DFS)

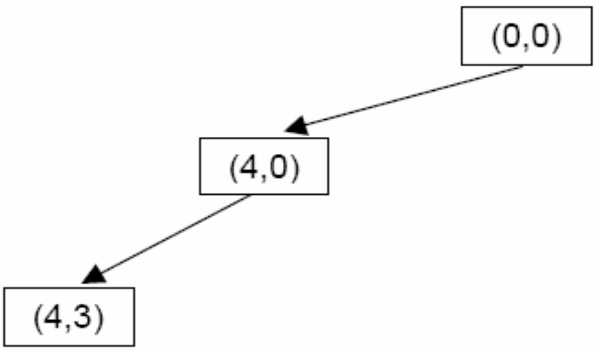
Pencarian dilakukan pada satu node dalam setiap level dari yang paling kiri. Jika pada level yang paling dalam, solusi belum ditemukan, maka pencarian dilanjutkan pada node sebelah kanan. Node yang kiri dapat dihapus dari memori. Jika pada level yang paling dalam tidak ditemukan solusi, maka pencarian dilanjutkan pada level sebelumnya. Demikian seterusnya sampai ditemukan solusi. Jika solusi ditemukan maka tidak diperlukan proses *backtracking* (penelusuran balik untuk mendapatkan jalur yang dinginkan).

Kelebihan DFS adalah:

- Pemakain memori hanya sedikit, berbeda jauh dengan BFS yang harus menyimpan semua node yang pernah dibangkitkan.
- Jika solusi yang dicari berada pada level yang dalam dan paling kiri, maka DFS akan menemukannya secara cepat.

Kelemahan DFS adalah:

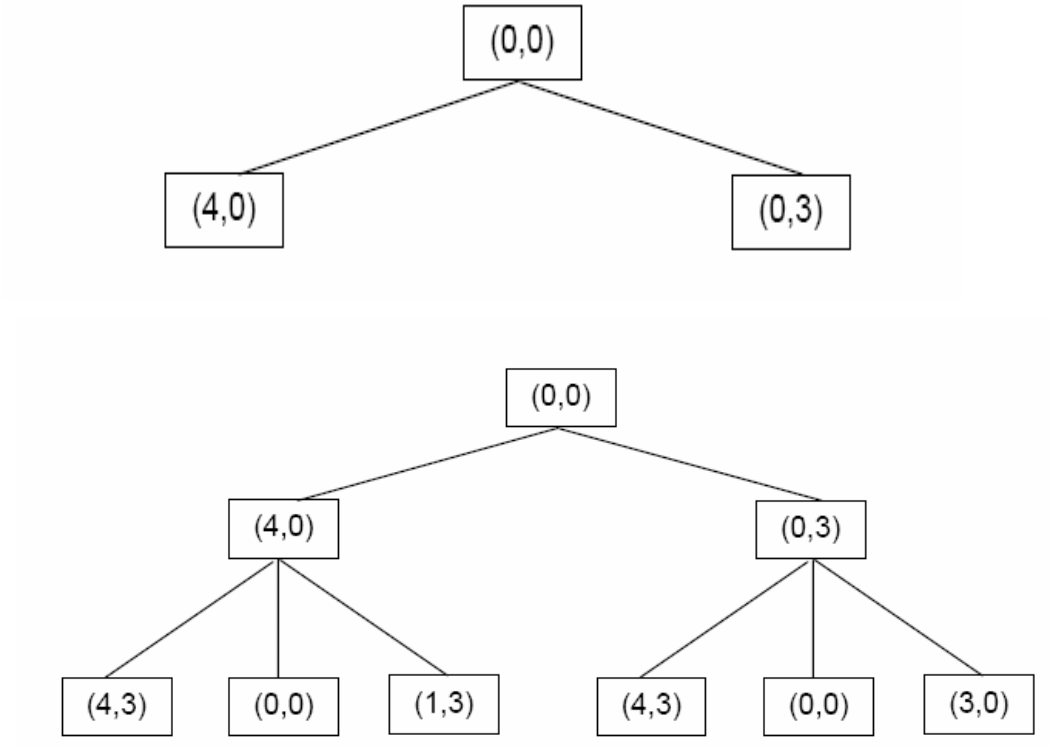
- Jika pohon yang dibangkitkan mempunyai level yang dalam (tak terhingga), maka tidak ada jaminan untuk menemukan solusi (**Tidak *Complete***).
- Jika terdapat lebih dari satu solusi yang sama tetapi berada pada level yang berbeda, maka pada DFS tidak ada jaminan untuk menemukan solusi yang paling baik (**Tidak *Optimal***).



Gambar 2.5 Penelusuran *Depth First Search* untuk *Water Jug Problem*.

***Breadth-First Search* (BFS)**

Pencarian dilakukan pada semua node dalam setiap level secara berurutan dari kiri ke kanan. Jika pada satu level belum ditemukan solusi, maka pencarian dilanjutkan pada level berikutnya. Demikian seterusnya sampai ditemukan solusi. Dengan strategi ini, maka dapat dijamin bahwa solusi yang ditemukan adalah yang paling baik (***Optimal***). Tetapi BFS harus menyimpan semua node yang pernah dibangkitkan. Hal ini harus dilakukan untuk penelusuran balik jika solusi sudah ditemukan. Gambar 2.4 mengilustrasikan pembangkitan pohon BFS untuk masalah *Water Jug*. Pembangkitan suksesor dari suatu node bergantung pada urutan dari Aturan Produksi yang dibuat (lihat gambar 2.3). Jika urutan dari aturan 4 ditukar dengan aturan 5, maka pohon BFS yang dibangkitkan juga akan berubah.



Gambar 2.4 Pohon *Breadth First Search* untuk *Water Jug Problem*.

Berikut ini membahas metoda-metode yang terdapat dalam teknik pencarian yang berdasarkan pada panduan (*Heuristic Search*), yaitu *Generate and Test, Simple Hill Climbing, Steepest-Ascent Hill Climbing, Simulated Annealing, Best First Search,Greedy Search, A Star (A*), Problem Reduction, Constraint Satisfaction,* dan *Means-Ends Analysis*.

Generate-and-Test

Metode *Generate*-and-*Test* adalah metode yang paling sederhana dalam pencarian *heuristic*. Jika pembangkitan *possible solution* dikerjakan secara sistematis, maka prosedur akan mencari solusinya, jika ada. Tetapi jika ruang masalahnya sangat luas, mungkin memerlukan waktu yang sangat lama.

Algoritma *Generate*-and-*Test* adalah prosedur DFS karena solusi harus dibangkitkan secara lengkap sebelum dilakukan *test*. Algoritma ini berbentuk sistematis, pencarian sederhana yang mendalam dari ruang permasalahan. *Generate & test* juga dapat dilakukan dengan pembangkitan solusi secara acak, tetapi tidak ada jaminan solusinya akan ditemukan.

Algorithm: *Generate*-and-*Test*

1. Generate a possible solution. For some problems, this means generating a particular point in the problem space. For others, it means generating a path from a start state.
2. Test to see if this is actually a solution by comparing the chosen point or endpoint of the chosen path to the set of acceptable goal states.
3. If a solution has been found, quit. Otherwise, return to step 1.

Contoh kasus:

Untuk permasalahan sederhana maka tehnik *generate & test* adalah tehnik yang layak. Sebagai contoh, pada teka-teki yang terdiri dari empat kubus segi enam, dengan masingmasing sisi dari setiap kubus dicat dengan 4 warna. Solusi dari teka-teki terdiri dari susunan kubus dalam beberapa baris yang semuanya empat sisi dari satu blok baris yang menunjukkan nasing-masing warna. Masalah ini dapat diselesaikan dengan manusia

dalam beberapa menit secara sistematis dan lengkap dengan mencoba semua kemungkinan. Ini bisa diselesaikan dengan lebih cepat menggunakan prosedur *generate & test*. Pandangan sekilas pada empat blok yang tampak bahwa masih ada lagi, katakanlah bagian merah dari warna-warna lain yang ada. Sehingga ketika menempatkan blok dengan beberapa bagian merah, ini akan menjadi ide yang baik untuk digunakan jika sebagian darinya sebisa mungkin dibagian luar. Sebagian yang lain sebisa mungkin harus ditempatkan pada blok berikutnya. Menggunakan aturan ini, banyak konfigurasi diperlukan tanpa di-*explore* dan sebuah solusi dapat ditemukan lebih cepat.

Heuristic Beam Search

Hill Climbing

Hill Climbing berbeda *Generate*-and-*Test*, yaitu pada *feedback* dari prosedur *test* untuk membantu pembangkit menentukan yang langsung dipindahkan dalam ruang pencarian. Dalam prosedur *Generate & test* , respon fungsi pengujian hanya **ya** atau **tidak**. Tapi jika pengujian ditambahkan dengan atauran fungsi-fungsi yang menyediakan estimasi dari bagaimana mendekati *state* yang diberikan ke *state* tujuan, prosedur pembangkit dapat mengeksplorasi ini sebagaimana ditunjukkan di bawah. HC sering digunakan jika terdapat fungsi *heuristic* yang baik untuk mengevaluasi *state*. Sebagai contoh, anda berada di sebuah kota yang tidak dikenal, tanpa peta dan anda ingin menuju ke pusat kota. Cara sederhana adalah gedung yang tinggi. Fungsi *heuristic*s-nya adalah jarak antara lokasi sekarang dengan gedung yang tinggi dan *state* yang diperlukan adalah jarak yang terpendek.

Simple HC

Algorithm: Simple HC

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until there are no new operators left to be applied in the current state:

a). Select an operator that has not yet been applied to the current state and apply it to

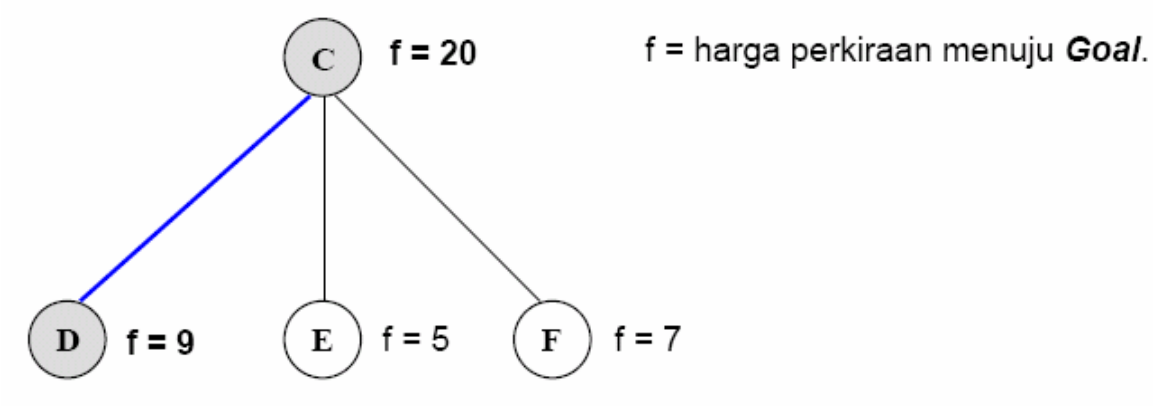
produce a new state.

b). Evaluate the new state:

(i) If it is a goal state, then return it and quit.

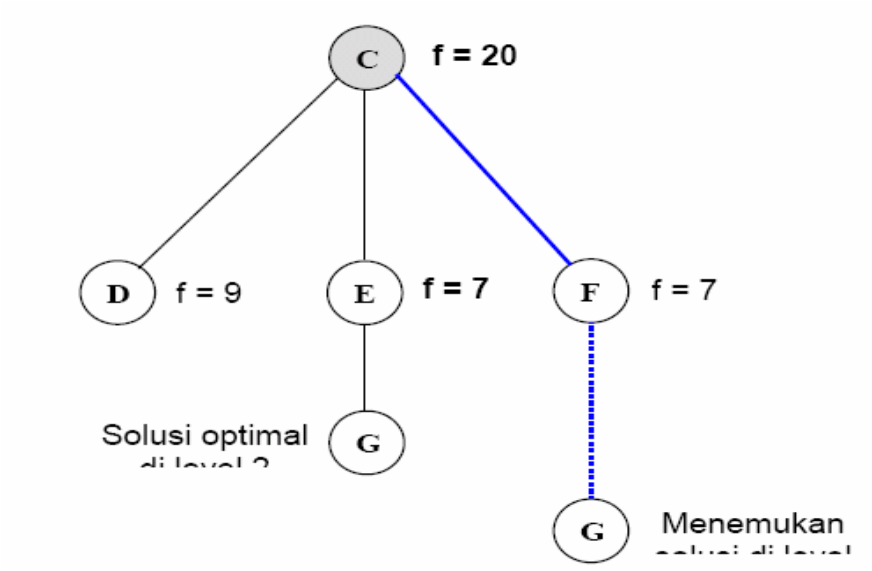
(ii) If it is not a goal state but it is better than the current state, then make it the current state.

(iii) If it is not better than the current state, then continue in the loop.



Gambar 3.1 Pencarian jalur menggunakan *Simple Hill Climbing*.

Steepest-Ascent HC



Gambar 3.2 Pencarian jalur menggunakan *Steepest-Ascent Hill Climbing*.

Pada gambar 3.2 di atas, terjadi ambiguitas dimana fungsi heuristik node E dan node F adalah sama. Misalkan dipilih F dan ternyata menemukan solusi di level 8. Padahal terdapat solusi lain yang lebih optimal di level 2. Hal ini dikatakan bahwa *Steepest-Ascent Hill Climbing* terjebak pada solusi lokal (*local minima*).

Algoritma Steepest-Ascent HC:

1. Evaluate initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until a complete iteration produces no change to current state:

a). Let *SUCC* be a state such that any possible successor of the current state will be better than *SUCC*.

b). For each operator that applies to the current state do:

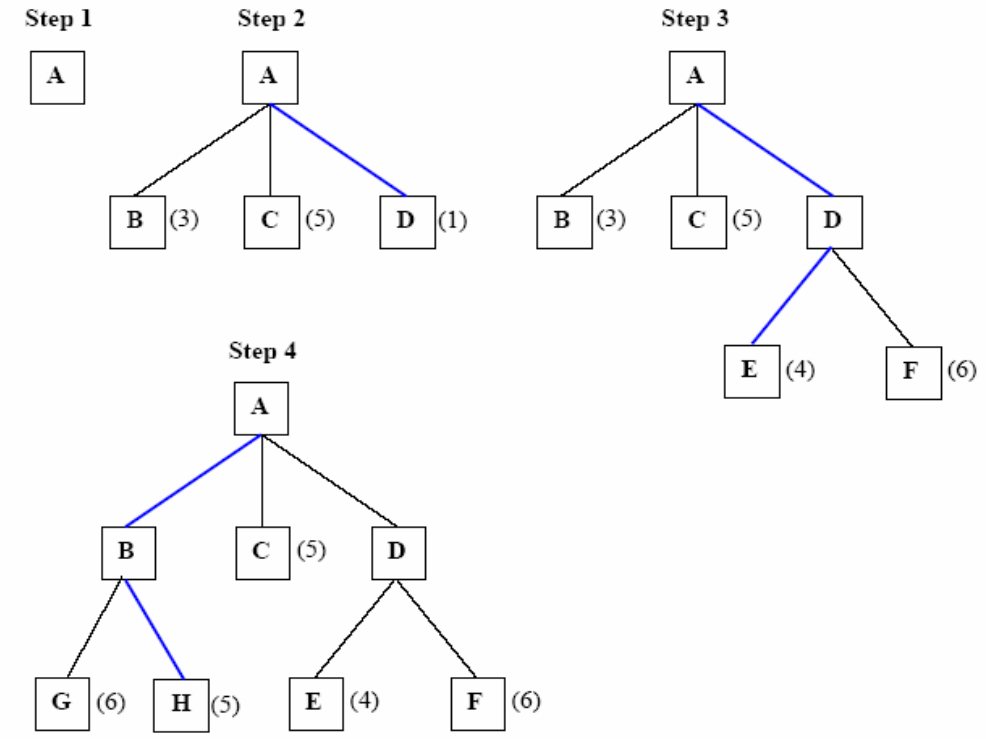
(i) Apply the operator and generate a new state.

(ii) Evaluate the new state. If it is a goal state, return it and quit. If not, compare it to *SUCC*. If it is better, then set *SUCC* to this state. If it is not better, leave *SUCC* alone.

c). If the *SUCC* is better than current state, then set current state to *SUCC*.

Best-First Search

Merupakan metode yang membangkitkan suksesor dengan mempertimbangkan harga (didapat dari fungsi heuristik tertentu) dari setiap node, bukan dari aturan baku seperti DFS maupun BFS. Gambar 3.4 mengilustrasikan langkah-langkah yang dilakukan oleh algoritma *Best First Search*. Pertama kali, dibangkitkan node A. Kemudian semua suksesor A dibangkitan, dan dicari harga paling minimal. Pada langkah 2, node D terpilih karena harganya paling rendah, yakni 1. Langkah 3, semua suksesor D dibangkitkan, kemudian harganya akan dibandingkan dengan harga node B dan C. Ternyata harga node B paling kecil dibandingkan harga node C, E, dan F. Sehingga B terpilih dan selanjutnya akan dibangkitkab semua suksesor B. Demikian seterusnya sampai ditemukan node Tujuan.



Gambar 3.4 Langkah-langkah yang dilakukan oleh algoritma *Best First Search*.

Untuk mengimplementasikan algoritma pencarian ini, diperlukan dua buah senarai, yaitu: OPEN untuk mengelola node-node yang pernah dibangkitkan tetapi belum dievaluasi dan CLOSE untuk mengelola node-node yang pernah dibangkitkan dan sudah dievaluasi. Algoritma selengkapnya adalah sebagai berikut:

Algoritma *Best-Fisrt Search*:

1. Start with *OPEN* containing just the initial state.
2. Until a goal 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, evaluate it, add it to *OPEN*, and record its parent.

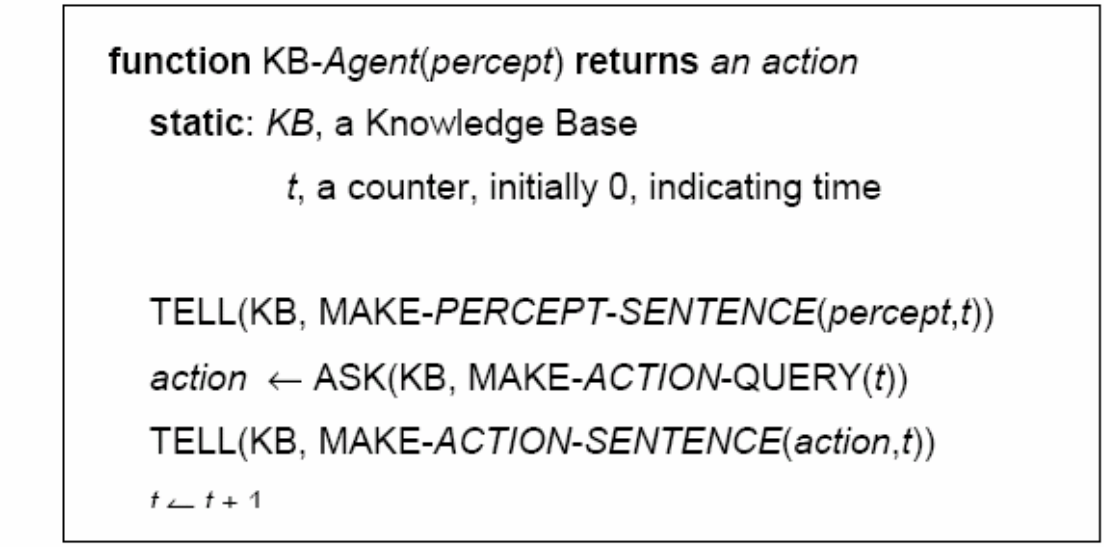
ii. If it has been generated, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.

Minggu ke 5,6, dan 7

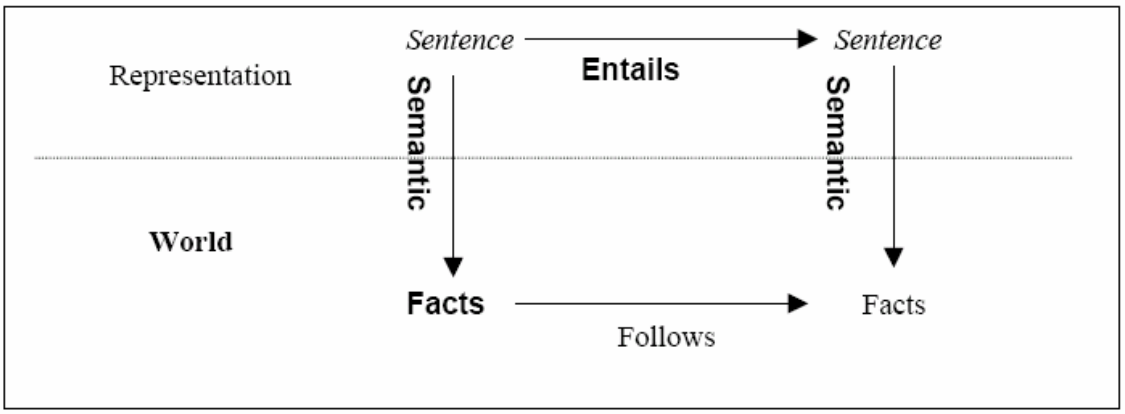
Representasi Pengetahuan

Pengetahuan dan Penalaran

Representasi pengetahuan adalah hal penting dalam *intelijensia buatan*. Di sini kita akan membahas dua *mathematical tools* untuk merepresentasikan pengetahuan, yaitu *propositional logic* (logika proposisi) dan *first order logic* (kalkulus predikat).



Gambar 4.1 *A generic knowledge-based agents*.



Gambar 4.2 Hubungan antara *sentence* dan *fatcs* yang disediakan oleh semantik bahasa.

Tabel 4.1 Pembagian formal language

Formal Language	**Apa yang ada di dunia nyata**	**Apa yang dipercaya *agent*tentang fakta**
Propositional logic	Facts	True/false/unknown
First-order logic	Facts, objects, relations	True/false/unknown
Temporal logic	Facts, objects, relations, times	True/false/unknown
Probability theory	Facts	Degree of believe 0…1
Fuzzy logic	Degree of truth	Degree of believe 0…1

Propositional Logic (Propositional Calculus)

Sentence → *AtomicSentence* | *ComplexSentence*

AtomicSentence → *True* | *False*
| P | Q | R

ComplexSentence → (*Sentence*)
| *Sentence Connective Sentence*
| ¬ *Sentence*

Gambar 4.3 A BNF (Backus-Naur Form) *Grammar of sentences inProportional Logic*

1. Modus Ponens atau Implication-Elimination:

$$\frac{\alpha \Rightarrow \beta,\ \alpha}{\beta}$$

2. And-Elimination:

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i}$$

3. And-Introduction:

$$\frac{\alpha_1, \alpha_2, \ldots, \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}$$

4. Or-Introduction:

$$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n}$$

5. Double-Negation-Elimination:

$$\frac{\neg\neg\alpha}{\alpha}$$

6. Unit Resolution:

$$\frac{\alpha \vee \beta,\ \neg\beta}{\alpha}$$

7. Resolution:

Gambar 4.4 Aturan inferensi dalam Logika Proposisi.

First-Order Logic (Predicat Logic / Predicat Calculus)

1. **Objects:** sesuatu dengan identitas individual (people, houses, colors, …)
2. **Properties:** sifat yang membedakannya dari object yang lain (red, circle, …)
3. **Relations:** hubungan antar object (brother of, bigger than, part of, ...)
4. **Functions:** relation yang mempunyai satu nilai (father of, best friend, …)

Contoh: **One plus two equals three.**

$$
\begin{aligned}
Sentence &\rightarrow AtomicSentence \\
&\mid Sentence\ Connective\ Sentence \\
&\mid Quantifier\ Variable, \ldots\ Sentence \\
&\mid \neg\ Sentence \\
&\mid (Sentence) \\
\\
AtomicSentence &\rightarrow Predicate(Term, \ldots) \mid Term = Term \\
\\
Term &\rightarrow Function(Term, \ldots) \\
&\mid Constant \\
&\mid Variable \\
\\
Connective &\rightarrow \Rightarrow \mid \wedge \mid \vee \mid \Leftrightarrow \\
Quantifier &\rightarrow \forall \mid \exists \\
Constant &\rightarrow A \mid X_1 \mid John \mid \ldots \\
Variable &\rightarrow a \mid x \mid s \mid \ldots \\
Predicate &\rightarrow Before \mid HasColor \mid Raining \mid \ldots \\
Function &\rightarrow Mother \mid LeftLegOf \mid \ldots
\end{aligned}
$$

Gambar 4.5 The Syntax of First-Order Logic (with equality) in BNF (Backus-Naur Form)

Konsep Dasar Representasi Pengetahuan

Representasi Pengetahuan berdasarkan LOGIKA

Proportional Logic (Zero Order Logic)

Predicate Logic (First Order Logic)

Representasi Pengetahuan berdasarkan *RULES*

Pengetahuan Prosedural vs Deklaratif

Logic Programming

Production Rules

Forward dan *Backward Reasoning Matching*

Forward and Backward Chaining

procedure FORWARD-CHAIN(*KB*,*p*)
  if there is a sentence in *KB* that is renaming of *p* **then return**
  Add *p* to *KB*
  for each $(p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ **in** *KB* such that for some *i*, UNIFY(p_i,*p*) = θ succeeds **do**
    FIND-AND-INFER(*KB*, [p_1, …, p_{i-1}, p_{i+1}, …, p_n],*q*,θ)
  end

---

procedure FIND-AND-INFER(*KB*, *premises*,*conclusion*,θ)
  if *premises* = [ ] **then**
    FORWARD-CHAIN(*KB*, SUBST(θ,*conclusion*))
  else for each *p′* **in** *KB* such that UNIFY(*p′*, SUBST(θ,FIRST(*premises*))) = θ_2 **do**
    FIND-AND-INFER(*KB*, REST(*premises*), *conclusion*, COMPOSE(θ,θ_2))

Gambar 4.6 Algoritma Inferensi Forward-Chaining.

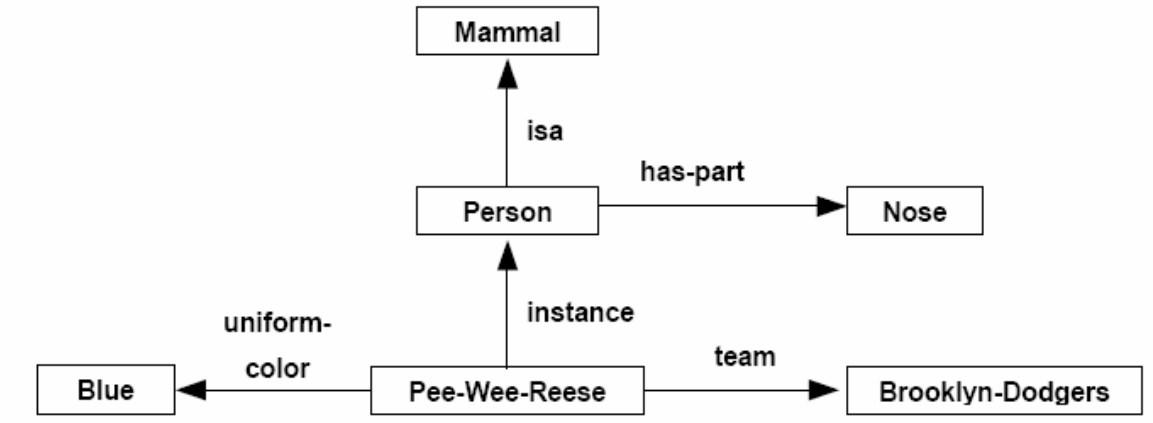
Semua *sentence* yang dapat diinferensi dari *sentence* p dimasukkan ke KB. Jika p baru, pertimbangkan setiap implikasi yang mempunyai premise yang sesuai dengan p. Untuk setiap implikasi seperti itu, jika semua *premise* yang tersisa berada dalam KB, maka simpulkan conclusion Jika premise dapat dicocokkan dengan beberapa cara

Representasi Pengetahuan Berdasarkan *Slot and Filler Structures*

Semantic Nets

Semantic Nets (Jaringan Semantik)

Terdapat relasi yang penting untuk inferensi, seperti *isa* dan *instance*.



Dalam Kalkulus Predikat dinyatakan (Binary predicate):

isa(Person, Mammal)

instance(Pee-Wee-Reese, Person)

team(Pee-Wee-Reese, Brooklyn-Dodgers)

uniform-color(Pee-Wee-Reese, Blue)

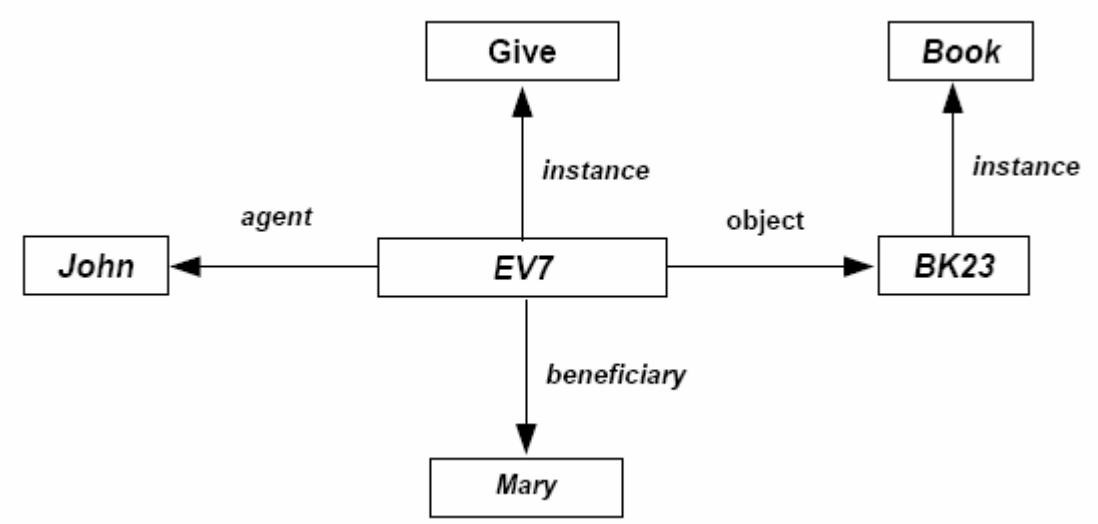
Dapat dilakukan **Inheritance** untuk menurunkan relasi tambahan:

Has-part(Pee-Wee-Reese, Nose)

Representasi Nonbinary Predicate

Jaringan Semantik untuk menggambarkan aspek dari kejadian tertentu.

Sebagai contoh: “John gave the book to Mary”**.**



Frames

Frames System

Kumpulan atribut (slot) dan nilai atribut yang mendeskripsikan suatu entitas.

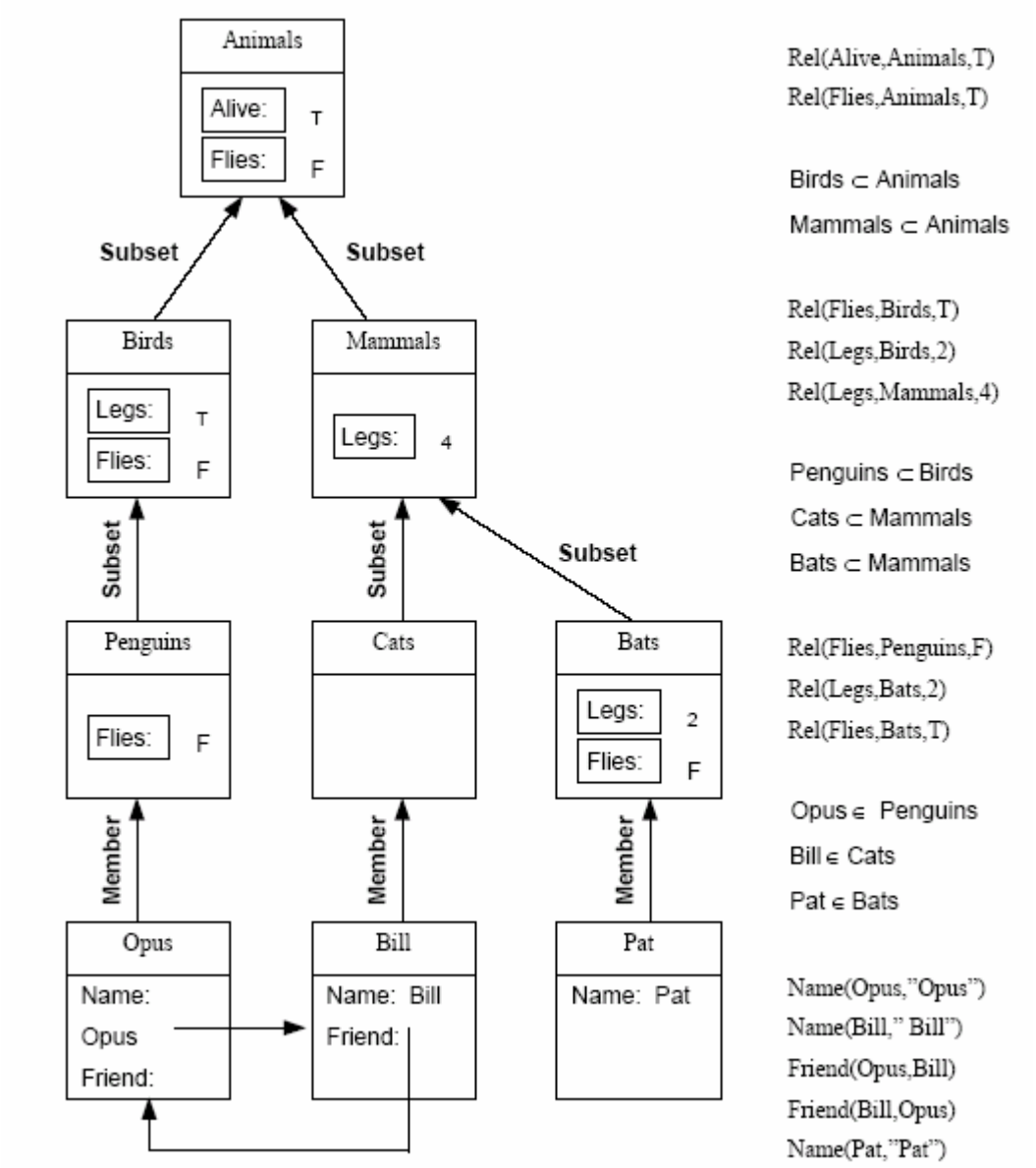
Nilai slot dapat berupa:

1. Identifikasi frame
2. relasi dengan frame lain (slotnya: isa, instance)
3. batasan nilai
4. nilai
5. default nilai (dapat diubah)
6. prosedur untuk mendapatkan nilai
7. prosedur yang dibangkitkan data (Data Driven): prosedur yang harus dilakukan jika nilai diubah, misalnya: periksa konsistensi.
8. kosong: untuk ditelusuri pada subclass-nya

Jenis Frame: Kelas dan Contoh (Instance)

Atribut Kelas:

1. Atribut tentang kelas itu sendiri.
2. Atribut yang harus diturunkan pada setiap elemen dalam himpunan.



(a) A Frame-Based Knowledge Base **(b) Translation into FOL**

Scripts

Merupakan representasi struktur yang mendeskripsikan aliran kejadian dalam konteks tertentu. Dimaksudkan untuk mengorganisasikan CD dalam situasi tertentu.

Komponen:

1. Entry condition : kondisi awal
2. Result : kondisi akhir
3. Props : yang harus ada
4. Roles : aksi yang dibangun tiap individu
5. Track : variasi spesifik pada pola yang lebih umum
6. Scenes : potongan-potongan "adegan" dalam Script

Keuntungan:

1. Mampu memprediksi event yang tidak disebutkan secara eksplisit.
2. Menyediakan cara pembangunan interpretasi tunggal dari sekumpulan observasi.
3. Mampu memfokuskan perhatian pada event yang "tidak biasa".

Contoh 1:

John pergi ke restaurant kemarin malam. Dia memesan steak. Saat membayar, dia menyadari uangnya kurang. Dia cepat pulang, karena hujan mulai turun.

Question: Apakah John makan malam?

(Dijawab dengan mengaktifkan Script restaurant)

Dari soal, urutan kejadian normal, sehingga pasti script restaurant berjalan normal, jadi John pasti melewati tahap makan.

Contoh 2:

Susan makan siang di luar. Dia duduk di meja dan memanggil pelayan. Pelayan memberikan menu dan Susan memesan hamburger.

Question: Mengapa pelayan memberikan menu?

Script mengandung dua jawaban:

- karena Susan meminta (backward)

- agar Susan dapat menentukan apa yang ingin dimakannya (Forward)

Contoh 3:

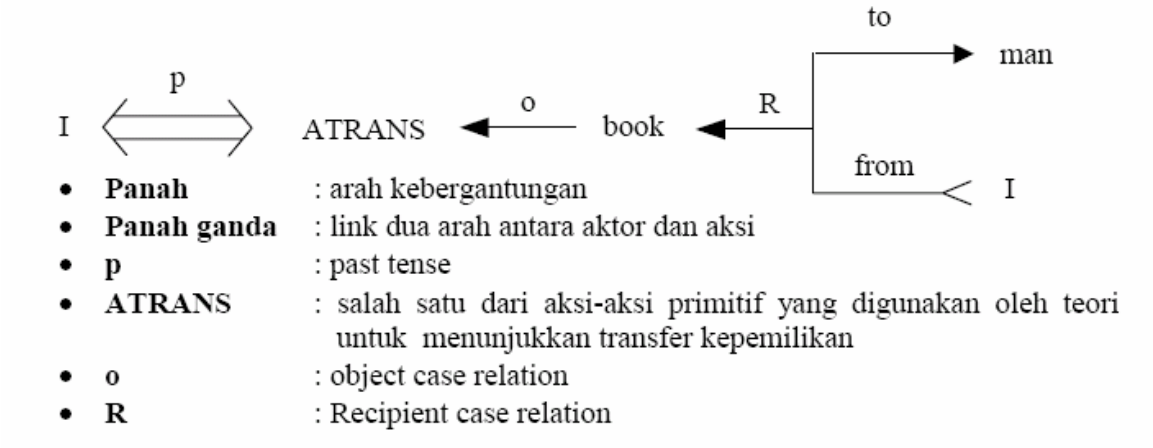
John pergi ke restaurant. Dia ditunjukkan mejanya. John memesan steak ukuran besar. Dia duduk dan menunggu lama. John marah dan pergi.

Conceptual Dependency (CD)

Merupakan *Strong Slot-and-Filler structure* karena menambahkan gagasan khusus tentang: apa tipe objek dan relasi yang diijinkan.

CD : Teori untuk merepresentasikan pengetahuan tentang **kejadian** yang terkandung dalam **kalimat** bahasa natural. Dengan catatan: menggambarkan penalaran kalimat dan tidak bergantung bahasa apa.

Contoh: I gave the man a book.



- **Panah** : arah kebergantungan
- **Panah ganda** : link dua arah antara aktor dan aksi
- **p** : past tense
- **ATRANS** : salah satu dari aksi-aksi primitif yang digunakan oleh teori untuk menunjukkan transfer kepemilikan
- **o** : object case relation
- **R** : Recipient case relation

Dalam CD, representasi aksi dibangun dari himpunan aksi primitif, yaitu:

ATRANS	Transfer of abstract relationship (e.g., give)
PTRANS	Transfer of physical location of an object (e.g., go)
PROPEL	Application of the physical force to an object (e.g., push)
MOVE	Movement of a body part by its owner (e.g., kick)

GRASP	Grasping of an object by an actor (e.g., clutch)
INGEST	Ingestion of an object by an animal (e.g., eat)
EXPEL	Expulsion of something from the body of an animal (e.g., cry)
MTRANS	Transfer of mental information (e.g., go)
MBUILD	Building new information out of old (e.g., decide)
SPEAK	Production of sound (e.g., say)
ATTEND	Focusing of a sense organ toward a stimulus (e.g., listen)

Terdapat 4 katagori konseptual primitif yang dapat dibangun, yaitu:

- **ACTS** : aksi
- **PPs** : objek / gambaran prosedur
- **AAs** : peubah aksi (pendukung aksi)
- **PAs** : peubah PPs (pendukung gambaran)

Tenses:

p	Past
f	Future
t	Transition
ts	Start transition
tf	Finished transition
k	Continuing
?	Interrogative
/	Negative
nil	Present
delta	Timeless
c	Conditional

CD tidak bisa membedakan yang alurnya sama. Misalnya : give, take, steal, donate.

CD cocok untuk kalimat yang sederhana. Untuk primitif tingkat tinggi CD merepotkan.

Misal: "John bet Sam $50 that the Mets would win the World Series"

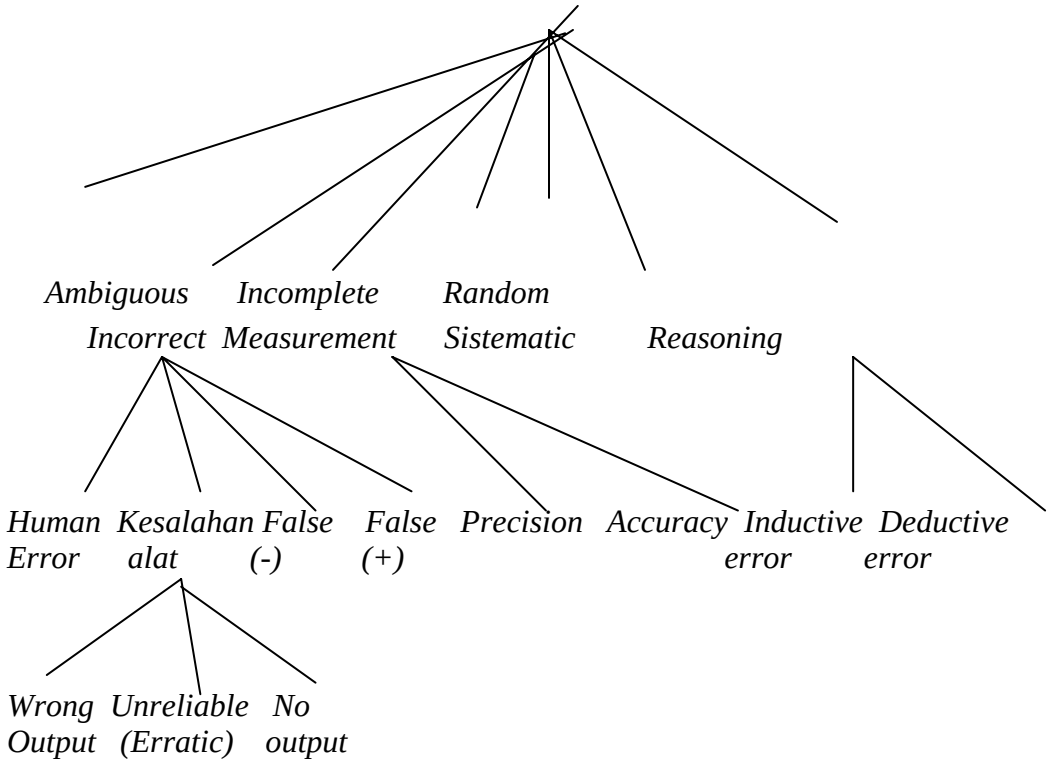
Minggu ke 9

PEMBERIAN ALASAN DI BAWAH KETIDAKPASTIAN

KETIDAKPASTIAN

- Disebut juga dg kekurangan informasi yg memadai untuk mengambil keputusan
- Probability klasik, bayesian prob, Hartley teory, Shannon teory, Dempster-Shafer teory, Zadeh's fuzzy teory
- Contoh yg berhubungan dg ketidakpastian : MYCIN, PROSPECTOR

TIPE *ERROR*/KESALAHAN



Keterangan :

- Ambiguous : kesalahan yg diinpretasikan lebih dari 1 cara
- Incomplete : informasi ada hilang
- Incorrect : informasi salah yang disebabkan manusia (kesalahan membaca data, peletakan informasi & peralatan)
- False Negative : penolakan hipotesa jika benar

- *False Positive* : penerimaan hipotesa jika tidak benar
- *Hipotesa* adalah sebuah asumsi yang akan di-test
- *Precision* : dalam milimeter, 10 X lebih teliti daripada centimeter, berhubungan dg bagaimana kebenaran itu diketahui/baik (*how well the truth is known*)
- *Accuracy* : dalam centimeter, berhubungan dg kebenaran (*the truth*)
- *Unreliability* : jika peralatan pengukuran mensuplay fakta yg tidak dipercaya.
- *Random* : fluktuai nilai
- *Systematic* : tidak acak tetapi karena bias mis pembacaan kalibrasi.

ERRORS DAN INDUKSI

Proses induksi merupakan kebalikan dari deduksi

DEDUKSI : umum ke khusus

Contoh : *All men are mortal*
Socrates is a man
∴ Socrates is mortal

INDUKSI : khusus ke umum

Contoh : *My disk drive has never crashed*
∵ It will never crash

Argumen induksi tidak pernah dapat dibuktikan, kecuali untuk induksi matematika. Argumen induksi hanya dapat menyatakan bahwa kesimpulan tersebut adalah benar

PROBABILITY KLASIK

- *Probability* merupakan cara kuantitas yang berhubungan dengan ketidakpastian
- Teori *probability* diperkenalkan pada abad 17 oleh penjudi Perancis dan pertama kali diajukan oleh Pascal dan Fermat (1654)
- Probabilitas Klasik disebut juga dengan **a priori probability** karena berhubungan dengan *game* atau sistem.
- **Formula fundamental probabilitas klasik**
  - W = jumlah kemenangan
- **P = W / N** N = jumlah kemungkinan kejadian yg
  - sama pada percobaan

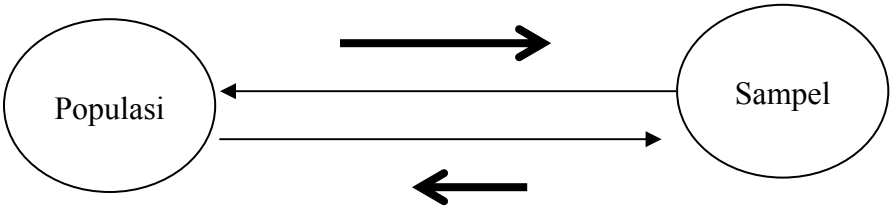
Contoh: Sebuah dadu dilemparkan 1X maka ada 6 kemungkinan

P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1/6

Jika percobaan diulang lagi maka akan menghasilkan yg sama (*Deterministic*), jika tidak *non-deterministic* (acak)

***Probability* kehilangan (Kalah)**

Q = (N –W) /N = 1 – P



EKSPERIMENTAL DAN PROBABILITAS SUBJECTIF

- **Ekperimental probability** kebalikan dari a priori yaitu *posteriori probability* atau *posterior probability* yaitu menentukan probabilitas suatu kejadian P(E).

  F(E) = frek kejadian

  $$P(E) = \lim_{N \to \sim} \frac{f(E)}{N}$$

  N = banyaknya kejadian

- *Subjective probability* berhubungan dengan kejadian yang tidak dapat direproduksi dan tidak mempunyai basis teori sejarah dimana untuk diramalkan (bukan berdasarkan aksioma)

PROBABILITAS GABUNGAN

- Kejadian dapat dihitung dari sample spacenya.
  Contoh : probabilitas perputaran dadu
  A = {2,4,6} B = {3,6}
  $A \cap B = \{6\}$
  $$P(A \cap B) = \frac{n(A \cap B)}{n(s)} = \frac{1}{6}$$
  n = angka elemen dalam set
  s = *sample space*

- **Independent events :** kejadian yg masing-masing tidak saling mempengaruhi. Untuk 2 kejadian bebas A dan B, probabilitasnya merupakan produk dari probabilitas individual.

- Kejadian A dan B disebut ***pairwise independent***
  $P(A \cap B) = P(A)\ P(B)$

- ***Stochastically independent event*** : Jika dan hanya jika formula diatas benar.
  Formula **mutual independence** N events mambutuhkan 2^N persamaan yagng dapat dipenuhi :
  $P(A^*_1 \cap A^*_2 \ldots\ldots \cap A^*_N) = P(A^*_1)\ P(A^*_2) \ldots P(A^*_N)$

Contoh :

$P(A \cap B \cap C) = P(A)\,P(B)\,P(C)$
$P(A \cap B \cap C') = P(A)\,P(B)\,P(C')$
$P(A \cap B' \cap C) = P(A)\,P(B')\,P(C)$ dst

- Untuk Gabungan $P(A \cup B)$
  1. **$P(A \cup B) = \dfrac{n(A) + n(B)}{n(S)} = P(A) + P(B)$**

     → hasilnya akan terlalu besar jika set overlap
     → untuk set disjoint

  2. **$P(A \cup B) = P(A) + P(B) - P(A \cap B)$**

     Atau

     $P(A \cup B \cup C) = P(A)+P(B)+P(C)-P(A\cap B)-P(A\cap C)$
     $\quad - P(B\cap C) + P(A \cap B \cap C)$

     → disebut *additive law*

PROBABILITAS KONDISIONAL

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \quad \text{for } P(B) \neq 0$$

$P(A \mid B)$ = Probabilitas kondisoinal

P(B) = probabilitas a priori

Jika probabilitas a priori digunakan dalam probabilitas kondisional maka disebut **unconditional/absolute probability**

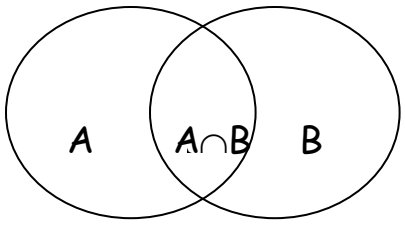
Jika probabilitas a priori digunakan dalam probabilitas kondisional maka disebut **unconditional/absolute probability**

Contoh

$$P(A) = \frac{n(A)}{n(S)} = \frac{4}{8}$$
$$P(B) = \frac{n(B)}{n(S)} = \frac{6}{8}$$

Jika diketahui kejadian B telah terjadi, maka *sample space* yang dikurangi hanya B
n(S) = 6

$$P(A \mid B) = \frac{n(A \cap B)}{n(B)} = \frac{2}{6}$$



Minggu ke 11

Konsep Dasar CLIPS

ELEMEN POKOK PADA CLIPS
1. fact-list: memori global untuk data
2. knowledge-base: berisi seluruh aturan/ rules
3. inference-engine: mengontrol keseluruhan pembuatan

CLIPS : (C Language Integrated Production System)
- dibuat di NASA/ John Space Center
- merupakan rangkaian forward bahasa berbasis aturan (rule-based) yang mempunyai inference dan representasi kemampuan.

NOTASI
Untuk kata dan karakter, yang tidak diapit dengan pasangan simbol < > , [ ] , { }
Contoh : (example)

[ ] : menunjukan OPTIONAL
(example [1]) berarti (example) ATAU (example 1)

< >: harus digantikan dengan argumen yang terdapat di dalamnya
(example <integer>) berarti (example 1) ATAU (example 5)

<< >>: deskripsi digantikan dengan nol/lebih kemunculan dari nilai yang ditentukan.

<<integer>> dapat diganti dg 1 dll atau tidak sama sekali.

<<<integer>>> ekuivalen dg <integer> <<integer>>

{ } : {all, none, somej berarti pilih salah satu saja, all ATAU none ATAU some

FIELDS

- Sekumpulan token. Token : sekumpulan karakter yang mempunyai arti khusus bagi CLIPS yaitu untuk membaca input dari keyboard dan file untuk mengeksekusi perintah dan me-load program.

Ada 3 tipe fields

1. Word

- Diawali dengan karakter ASCII yang dapat dicetak, diikuti dengan karakter lain yang diijinkan.
- Sebuah word tidak boleh diawali dengan sekelompok karakter berikut: < l & $? + - ( ) ;
- Word tidak boleh berisi salah satu karakter berikut < l & ( ) ; (dianggap sbg delimiter untuk mengakhiri word)
- Contoh : emergency
  emergency-fire
  kita_sama_sama
  ! ? # $
- CLIPS bersifat Case-Sensitive, membedakan huruf besar atau kecil.

2. String

✓ Diawali dan diakhiri dengan tanda petik dua (")
✓ Contoh : "emergency" , "< - ( ) + "
✓ Spasi dianggap **sebagai delimeter dalam CLIPS** untuk memisahkan field-field (seperti word) dan token lainnya.
✓ Spasi dianggap sebagai bagian dari string.
✓ Contoh : "fire" "fire " " fire"
✓ Jika akan ditempatkan dalam **string (dianggap** sebagai suatu string), maka tanda petik tersebut dimasukkan ke dalam suatu string dengan menggunakan \ **(backslash)**.
✓ Contoh : ""fire"" → "\"fire\""
"\fire\" → ???????

3. Number (numeric field)

✓ Floating point number
✓ Terdiri atas 3 bagian : simbol (+ atau -), nilai (terdiri dari 1 atau lebih digit dg sebuah optional desimal point), eksponen (e/E diikuti dg + atau – optional dengan 1 atau lebih digit)
✓ Contoh : 1 1.5 .7 +3 -5 3.6e10

F*AKTA*

➢ Fakta terdiri dari satu atau lebih field, yang diapit dengan tanda kurung ( ).

- Contoh : (single-field)
  (two field)
  (name "John Doe")
  (emergency-fire) (emergency fire)

FACT TEMPLATES

Template adalah kumpulan fakta yang mempunyai relasi sama. Sebuah template untuk fakta ditunjukkan dengan nama relasi diikuti dengan satu/lebih item umum/khusus. Contoh.:

(emergency <type>)

(emergency fire) ATAU (emergency flood) dsb.
Contoh :
(action activate springkle system) →
(action activate fire-alarm) →
(action notify fire-department) →
(action shutdown-electrical-power) →
(computer-components cpu disk-drive terminal) →

Fact template ini hanya digunakan sebagai dokumentasi, bukan sebuah perintah di CLIPS.

MASUK & KELUAR CLIPS

- ✓ Promptnya: CLIPS> perintah yang dimasukkan secara langsung pada CLIPS disebut model "top level".
- ✓ Keluar dari CLIPS dengan (exit)
- ✓ Setiap perintah yang dimasukkan ke CLIPS harus mempunyai tanda kurung. kin' dan kanan yang seimbang.
- ✓ Perintah-perintah yang dirnasukkan dengan menggunakan tanda kurung yang diakhiri dengan RETURN.

A> CLIPS (return/ enter)
CLIPS (V4.20 4/29/88)
CLIPS>(exit) (enter)
A>

ADDING & REMOVING FACTS

Menambah dan Menghapus Facts Fakta dapat disimpan da!am fact-list. Penambahan dengan menggunakan perintah :

(assert <<< fact > >>)

Contoh :

CLIPS>(assert (emergency fire)) (enter)
CLIPS>

Untuk menampilkan fact-list dengan perintah

(facts)

Contoh

CLIPS>(facts) (enter)
f - 1 (emergency fire)
CLIPS>

f-1 : fact identifier/fact index (bisa tidak terurut).
Sintaks perintah facts lengkapnya

(facts [<start> [<end> [<max>]]])

Menghilangkan fakta dari faktanya disebut retraction, perintahnya

(retract <<<fact-index>>>)

Contoh :
CLIPS> **(assert (emergency fire)** ↵
(emergency flood)) ↵
CLIPS> (facts) ↵
Output :
f-1 (emergency fire)
f-2 (emergency flood)
CLIPS> (retract 2) ↵
CLIPS> (retract 1 2) ↵
Jika yang di-retract tidak ada lagi maka message-error :

Fact <fact-index> does not exist

Komponen-Komponen Sebuah Rule (aturan)
Agar tujuan tercapai, sebuah SP harus mempunyai aturan disamping fakta. Rule dapat diketikkan langsung pada CLIPS atau di-load-kan dari file yang diketikkan dengan menggunakan sebuah editor.
Contoh

```
(defrule fire-emergency "An example rule"
    (emergency fire)
    =>
    (assert (action activate-springkler-system)))
```

IF the emergency is a fire
THEN the action is to activate the springkler system

Format umumnya

```
(defrule <rule name> [<optional comment>]
        <<patterns>>            ; LHS (Left Hand Side)  - IF
        =>
        <<actions>>)    ; RHS (Right Hand Side)  -THEN
```

✓ Bisa ditambah dengan komentar, diawali dengan **;** dan diakhiri dengan Return.

```
; Rule header
(defrule fire-emergency “An example rule”
   ; Patterns
   (emergency fire)
   ; THEN arrow
   =>
   ; Actions
   (assert (action activate-springkler-system)))
```

- ✓ Header terdiri dari 3 bagian :
  1. Nama rule
  2. Patterns/kondisi elemen yg terdiri dari 1 atau lebih field
  3. Action
- ✓ Jika rule sesuai dengan fakta, rule diaktifkan dan diletakkan di *"agenda"*, yaitu kumpulan rule-rule yang aktif.
- ✓ Jika sebuah rule yang diaktifkan mempunyai pattern, maka pattern khusuc-(initial-fact) ditambahkan sebagai pattern rule tsb.
- ✓ Mungkin juga sebuah rule tidak mempunyai action tetapi dapat tetap diaktifkan.
- ✓ Istilah fire berarti CLIPS mengeksekusi action sebuah rule dari agenda, Pelaksanaan rule pada agenda dan fire rule-nya tergantung pada prioritas tertinggi disebut **salience**.

AGENDA & PENGEKSEKUSIAN

Program CLIPS dapat dieksekusi dengan perintah

(run [<limit>])

<limit> : maks.jumlah rule yang di-fire

- ✓ Jika tidak ada <limit> atau -1 maka rule akan di-fire hingga t1dak ada rule lain di agenda. Sebaliknya, pelaksanaan (eksekusi) akan berakhir sebanyak <limit> rule yang di-fire.
- ✓ Rule hanya dapat diaktifkan oleh fakta yang di-assert

Mengaktifkan Rule

Jika fakta telah ada di fact-list, pemasukan sebuah fakta baru yang sama tidak ada pengaruhnya. Agar dapat dimasukkan, fakta yang lama harus dilhapus. Jadi tidak ada duplikasi fakta.

Daftar dari rule dapat ditampilkan dengan perintah

(agenda)

contoh : CLIPS> **(agenda)** ↵

```
0   fire-emergency: f-2
CLIPS>
```

Rule dan Refraksi

Dengan perintah (run) menyebabkan. rule di-fire dan menghasilkan output,

CLIPS> **(run)** ↵
1 rule fired ; jumlah rule yg di-fire
Run time is 0.00167 seconds ; tdk semua mesin ada
CLIPS>

CLIPS diprogram dengan karakteristik *sel neuron*. Setelah neuron mengirimkan impuls syaraf (fire), sejumlah stimulasi tidak akan di-fire untuk beberapa saat, disebut *refraksi/bias*.

PERINTAH – PERINTAH DENGAN RULE

Untuk menampilkan daftar rule dalam CLIPS

(rules)

Contoh : CLIPS> **(rules)** ↵
fire-emergency

Untuk menampilkan representasi teks dari rule

(pprule <rule-name>)

pprule = pretty print rule

CLIPS> **(pprule fire-emergency)** ↵
(defrule fire-emergency "An example Rule"
(emergency fire)
=>
(assert (action activate-springkler-system)))
CLIPS>

Me-load rule dari file : **(load <file-name>)**

(load "B:fire.clp")

(load "B:\\usr\\clips\\fire.clp”)

Menyimpan rule ke file : **(save <file-name>)**

(save "B:fire.clp")

KOMENTAR RULE

Sebuah komentar di CLIPS adalah suatu teks yang diawali dengan semicolon (;) dan diakhiri dengan return.

Pernanggilan (load) rule dari sebuah file pada CLIPS, akan menghilangkan setiap kornentar.

PERINTAH PRINTOUT

RHS dapat digunakan untuk pencetakan informasi dengan perintah

(printout <logical-name> <<print-items>>)

<logical-name> : tujuan output

Contoh :

(defrule fire-emergency
(emergency fire)

=>
(printout t "Activate the springkler system" crlf))

t memerintahkan CLIPS untuk mengirimkan output ke standard output device, Umumnya terminal, tetapi mungkin ke modem atau printer. ***Crlf = carriage return line feed***
PENGGUNAAN MULTIPLE RULES
SP dalam prakteknya dapat terdiri dari beratus-ratus atau ribuan rule.

TUGAS
(defrule class-A-fire-emergency
(emergency fire)
(fire-class A)
=>
(printout t "Activate springkier system" crlf))

(defrule class-B-fire-emergency
(emergency fire)
(fire-class B)
=>
(printout t "Use carbon dioxide extinguisher" crlf))

Sejumlah pattern dapat dimasukkan dalam rule. Yang penting adalah rule dapat diletakkan di agenda Jika seluruh pattern sesuai dengan fakta (LOGIKA AND)

BENTUK DEFFACTS

Sekumpulan fakta yang menunjukkan pengetahuan awal (initial knowledge) ditunjukkan dengan perintah

(deffacts <defact name> [<optional comment>]
<<facts>>)

Contoh :
(deffacts status "Some facts about the emergency"
(emergency fire)
(fire-class A))

Fakta-fakta dalam deffacts dimasukkan dengan perintah :
(reset)

Contoh : anggap deffacts status telah dimasukkan,
CLIPS> **(reset)** ↵

CLIPS> (facts) ↵
f-0 (initial-fact) ; otomatis
f-1 (emegency fire)
f-2 (fire-class A)

✓ ***CLIPS secara otomatis akan mendefinisikan***
(deffacts initial-fact (initial-fact))

- ✓ Meskipun pernyataan deffacts belum didefinisikan, reset akan memasukkan fakta (initial-fact)
- ✓ **(reset)** akan menghapus semua rule yang diaktifkan dari agenda dan seluruh fakta dalam fakta

Perintah lainnya : (list deffacts)
(ppdeffact <deffact-name>)

CLIPS> **(list deffacts)** ↵
initial-fact
status
CLIPS> **(ppdeffact status)** ↵ (return)
(deffacts status "Some facts about the emergency"
(emergency fire)
(fire-class A))

REMOVE DEFFACTS DAN RULES
Perintah menghapus dalam CLIPS :
(undeffacts <deffact-name>)

Untuk mengembalikan pernyataan undeffacts setelah di-undeffacts, perintah deffacts harus dimasukkan kembali.

Perintah menghapus dg mnyeleksinya :
(excise <rule-name>)

Perintah menghapus seluruh informasi yang berada pada CLIPS:
(clear)

PERINTAH WATCH
Watch dalam CLIPS digunakan untuk debugging program. Sintaksnya
(watch {facts, rules, activations, all})

Jika fact di-watch maka CLIPS secara otomatis mencetak pesan yang menunjukkan bahwa sebuah update terjadi.

Contoh :

CLIPS> (reset) ↵
CLIPS> (watch facts) ↵
CLIPS> **(assert (emergency fire)) ↵**
==> f-1 (emergency fire)
CLIPS> **(assert (emergency flood)) ↵**
==> f-2 (emergency flood)
CLIPS>**(retract 2) ↵**
<= = f-2 (emergency flood)

Perintah untuk mengembalikan (mematikan) watch

(unwatch (facts, rules, activations, all))

PERINTAH MATCHES (Perintah debug)

Digunakan untuk mencocokkan pattern dalarn rule dengan faktanya. Pattern yang tidak cocok menyebabkan rule tidak dapat diaktifkan. Sintaksnya

(matches <rule-narne>)

PERINTAH SET-BREAK

Digunakan untuk menghentikan eksekusi sebuah rule yang sebelum di-fire disebut breakpoint. Sintaks :

(set-break <rule-name>)

Perintah untuk menampilkan seluruh breakpoint

(show-breaks)

Perintah untuk menghapus breakpoint :

(remove-break [<rule-name])

Mingggu ke 12

PENGENALAN SISTEM PAKAR

Pengertian Sistem Pakar (*Expert System*)

- Membuat S/W Expert Systems → prog. Sebagai penasehat/konsultan pakar
- Dapat mengumpulkan dan penyimpan pengetahuan seorang/beberapa orang pakar ke dalam komp. → u/ semua orang yang memerlukan
- Tidak u/ menggantikan kedudukan seorang pakar ttp u/ memasyarakatkan pengetahuan & pengalamaan pakar tsb.
- Memungkinkan orang lain meningkatkan produktivitas, memperbaiki kualitas keputusan dll.

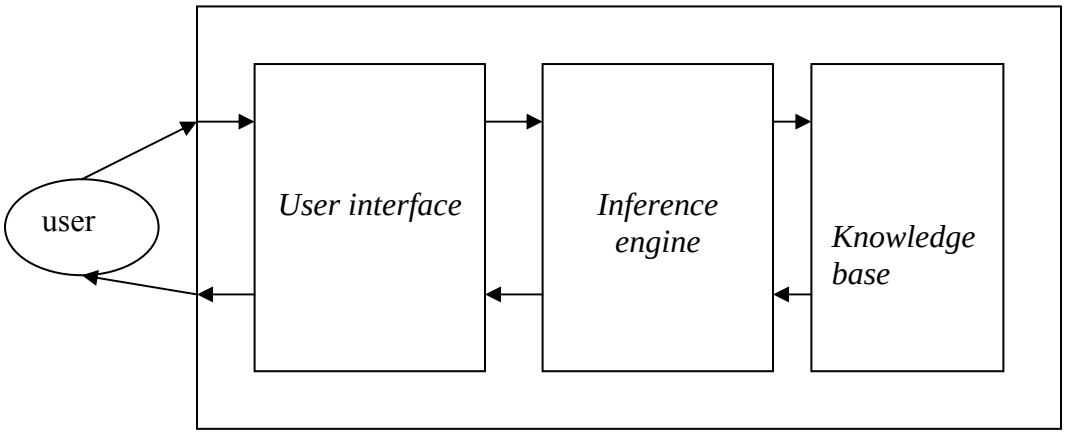


Diagram blok *Expert Systems* (umum)

- *Knowledge base* berisi semua fakta, ide, hubungan
- Motor inferensi bertugas untuk menganalisis pengetahuan dan menarik kesimpulan berdasarkan *knowledge base.*
- S/W *user interface* berfungsi sebagai media pemasukan pengetahuan ke dalam (KB)

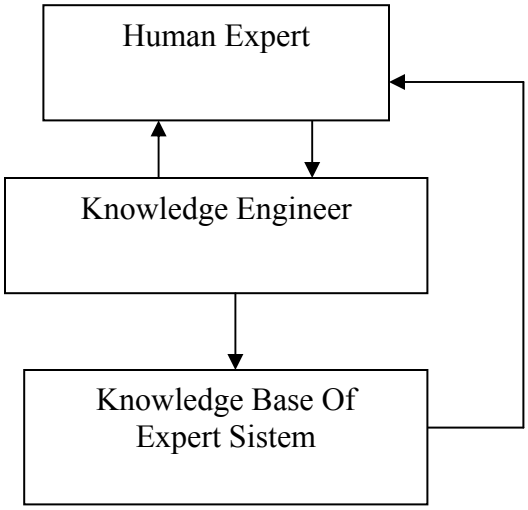
Keuntungan / Kelebihan Sistem Pakar

- Availability-bertambah
- Cost-rendah
- Danger-reduced
- Performance
- Multiple expertise
- Reability-bertambah
- Explanation
- Response-cepat
- Steady, unemotional and complete response
- Intelligent tutor
- Intelligent dB

KONSEP UMUM SISTEM PAKAR (SP)

*Salah satu metode representasi pengetahuan:
IF….. THEN

*Proses pembuatan SP → *knowledge engineering* yang dilakukan oleh *knowledge engineer.* Selain itu *domain expert* dan *end user.*



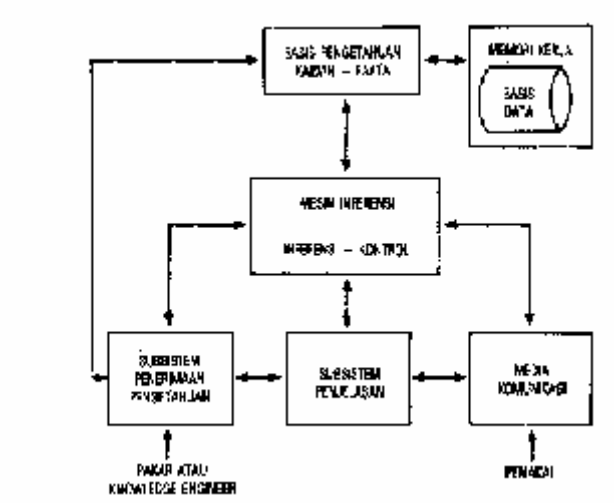
Perkembangan Sistem Pakar

* Tugas *knowledge engineer* adalah memilih S/W & H/W u/ pembuatan SP, membantu mengambil pengetahuan yg dibutuhkan dari pakar domain, serta implementasi pengetahuan pada basis pengetahuan yg benar & efisien

*Tugas pakar domain : meyediakan pengetahuan ttg bid problem yg dihadapi, memahami teknik-teknik pemecahan problema yg dipakai.

*Batasan praktis dari beberapa SP → *casual knowledge*

*SP lebih mudah untuk memprogram dg *Shallow knowledge yaitu berdasarkan pada pengalaman dan pengetahuan heuristik* .



Keterangan

- Basis Pengetahuan
  - inti prog SP
  - representasi pengetahuan dari seorang pakar.
  - Macam-macam ……
- Mesin Inferensi
  - mekanisme fungsi berfikir dan pola-pola penalaran sistem yg digunakan pakar
  - menganalisa suatu maslah tertentu
  - mencari jawaban atau solusi yg terbaik.
  - Ada 2 pelacakan backward & forward chaining

KARAKTERISTIK SP

- High Performance
- Adequate response time
- Good reliability
- Understanable
- Flexibility

PENGEMBANGAN TEKNOLOGI SP

- Akar SP pada banyak disiplin ilmu *"cognitive science"* yaitu study bagaimana orang memikirkan dlm pemecahan masalah. *"cognitive processor"* yaitu menemukan aturan yg akan diaktifkan.

SP YANG TERKENAL

1. MYCIN
   - Dirancang oleh Edward Feigenbaum (Universitas Stanford) th '70 an
   - SP medical yg dpt mendiagnosa infeksi bakteri & rekomendasi pengobatan antibiotik
2. DENDRAL
   - SP struktur molekular & kimia
3. PROSPECTOR
   - Membantu ahli geologi yg mencari & menemukan biji deposit (mineral& batu-batuan)
   - Didesign oleh Sheffield Research Institute, akhir '70an
4. XCON (R1)
   - SP konfigurasi sistem komputer dasar
   - Dikembangkan oleh Digital Equipment Corporation (DEC) dan Carnegie Mellon Universitas (CMU), akhir '70 an
   - Untuk sistem komputer DEC VAC 11 1780
5. DELTA
   - Didesign & dikembangkan oleh General Electric Company
   - SP personal maintenance dg mesin lokomotif listrik diesel.
6. YESMVS
   - Didesign oleh IBM awal th '80an

- Membantu operator komputer & mengontrol sistem operasi MVS (multiple virtual storage)

7. ACE
   - Didesign & dikembangkan oleh AT&T Bell Lab awal th ‘80an
   - SP troubleshooting pd sistem kabel telpon

KLASIFIKASI APLIKASI SP

1. CONTROL
   - Aplikasi komputer yg sangat umum
   - Ada 2 jenis kontrol : loop terbuka & tertutup
2. DEGUGGING
   - Proses mencari kesalahan & memperbaiki solusi.
3. DESIGN
   - Pengumpulan informasi mengenai spesifikasi sistem & produk tertentu
   - Untuk merancang sirkit elektronik, bangunan, dan rumah.
4. DIAGNOSIS
   - Untuk mendiagnosa produk atau sistem yg sudah tdk berfungsi.

5. INSTRUKSIONAL
   - Untuk membantu dalam proses belajar mengajar

6. INTERPRETASI
   - Membantu seorang dlm menafsir & memahami situasi/perspektif suatu peristiwa.
   - Contoh : analisa intelegensia, daya tahan, citra dan sinyal

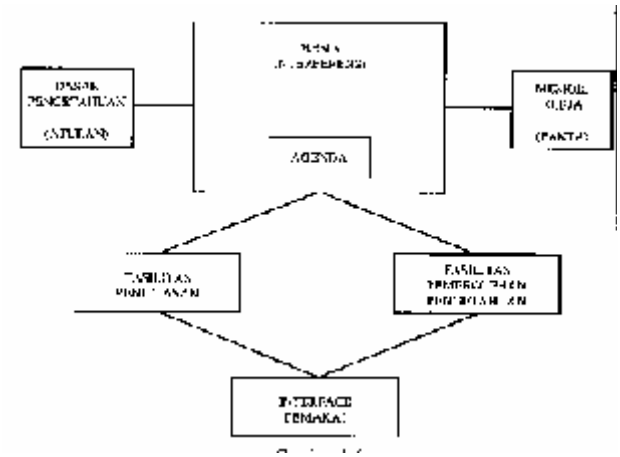
7. PLANNING
   - Merumuskan metode, penataan yg dapat mendekatkan pd tujuan.
   - Contoh : proyek manajemen, taktik & strategi militer, pemrograman robot

8. PREDIKSI
   - Meramalkan apa yg terjadi di masa yg akan datang.

9. REPARASI
   - Memperbaiki barang yg rusak ke keadaan semula

10. KONFIGURASI

ELEMEN SP



1. User Interface --- kom antara user & SP
2. Explanation Facility --- pemberian alasan pd user
3. Working Memori
4. Inference Engine --- penentuan aturan yg hrs dipenuhi, prioritas aturan yg tercukupi, & prioritas yg tertinggi
5. Agenda --- daftar yg diprioritaskan dari aturan (4)
6. Fasilitas Pemrolehan Pengetahuan --- cara otomatis bagi pemakai untuk memasukkan pengetahuan dlm sistem.

Rangkaian Forward (Forward chaining)
→ merupakan pemberi alasan dari fakta untuk kesimpulan hasil dari fakta
Contoh :

Jika kita melihat bahwa hari ini akan turun hujan sebelum pergi (nyata)

Maka kita harus membawa payung (kesimpulan)

Mis : Programan OPS5, CLIPS

Rangkaian Backward (Backward chaining)
→ Pemberian alasan sebaliknya dari hipotesa, kesimpulan potensial dibuktikan, pada fakta yg mendukung hipotesa

Contoh:

Jika kita tidak melihat keluar dan seseorang masuk dg sepatu basah dan payung.
Hipotesa kita adalah bahwa hari hujan

Mis : EMYCIN

SISTEM PRODUKSI

- Salah satu type SP yg paling terkenal adalah system yg berdasarkan pd aturan.
- Alasannya :
  1. *Modular nature*
  2. *Explanation facility*
  3. *Similarity to the human cognitive process*

- POST

  Idenya :
  - System matematika & logika merupakan set aturan sederhana untuk menentukan bagaimana mengubah 1 string simbol ke dlm simbol lainnya.
  - Yaitu dg input string, kejadian sebelumnya,
  - ALGORITMA MARKOV
    - Merupakan kelompok produksi yg terorder yg diterapkan untuk prioritas ke input string.
    - Algoritma akan berakhir dg baik jika:
      (1). Produksi terakhir tidak dapat diterapkan pada string
      (2). Suatu produksi yg berakhir dg periode diterapkan.
      - Jika input string GABKAB

System produksi AB → HIJ
Maka hasil akhir GHIJKHIJ

- *Karakter ^ → string nol*
- Mis A → ^ artinya menghilangkan seluruh kejadian karakter A dlm suatu string
- *Karakter tunggal a,b,c,……*
- Mis AxB → BxA artinya mengubah karakter A dan B
- Huruf Yunani α,β

Contoh : Memindahkan huruf pertama string input ke akhir

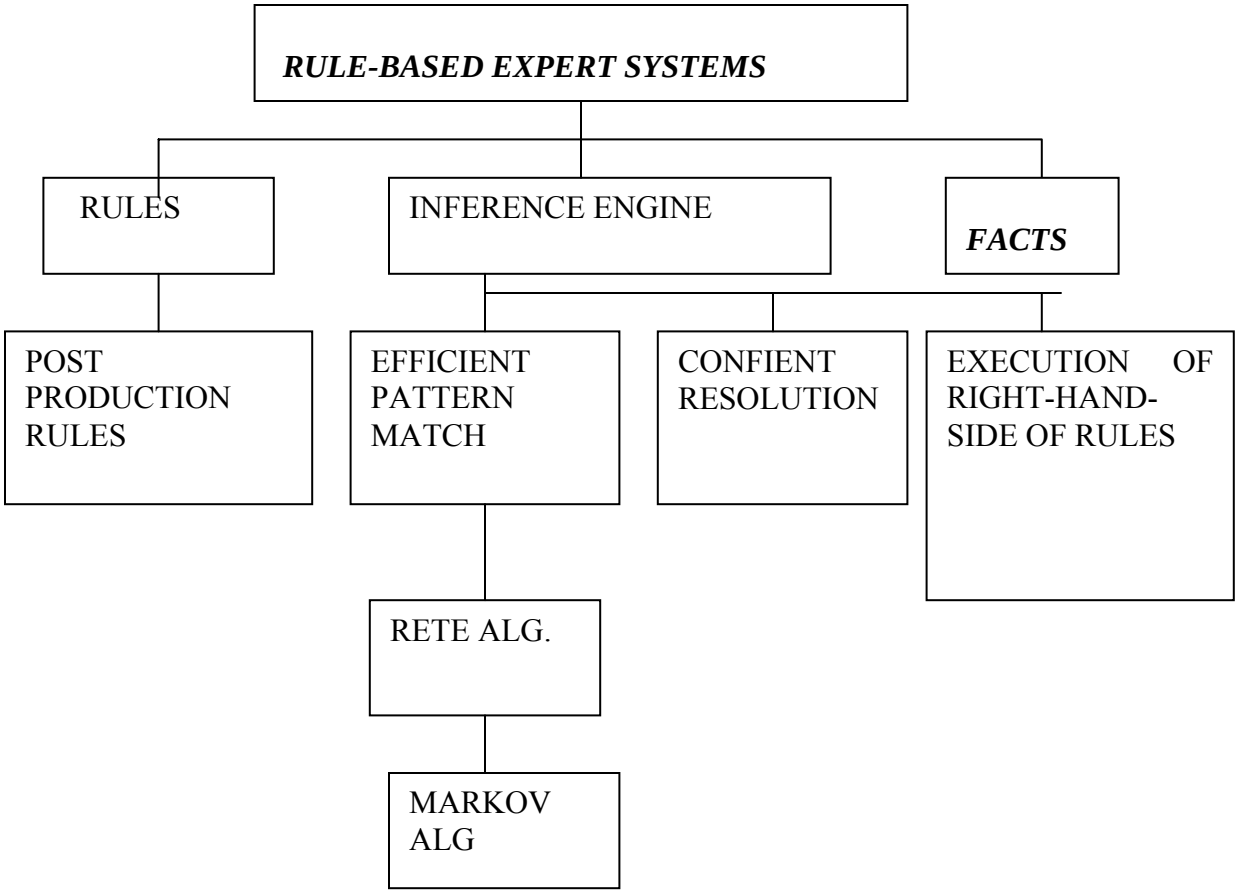
Aturan 1. αxy → yαx
2. α → ^
3. ^ → α

Input ABC

Aturan	Sukses atau Gagal	String
1	G	ABC
2	G	ABC
3	S	αABC
1	S	BαAC
1	S	BCαA
1	G	BCαA
2	S	BCA

➢ ALGORITMA RETE

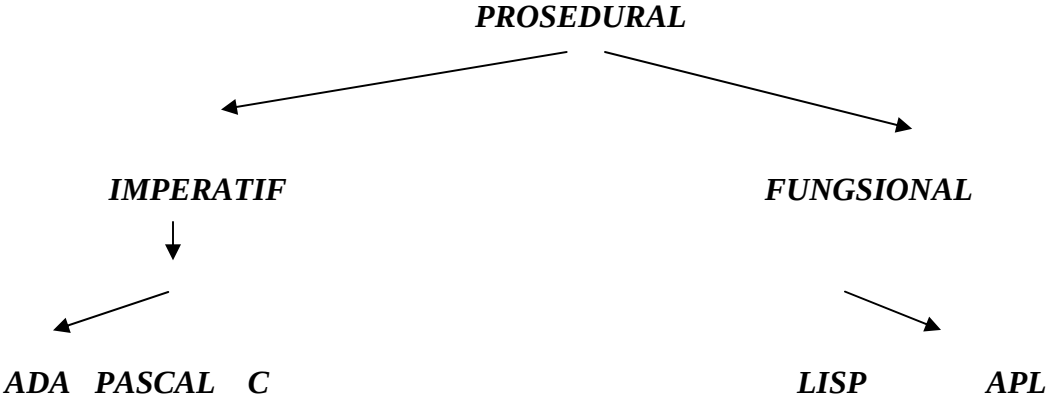
- Pada algoritma Markov diterapkan aturan/baris prioritas lebih tinggi.
- Masalah timbul jika system mempunyai aturan/baris yang banyak, maka tidak akan efisien.
- Solusinya adalah *algoritma Rete* yang dikembangkan oleh **Charles L.F** di **Carnegie-Mellon University (1979)**
- Yaitu algoritma yg mengetahui tentang seluruh aturan/baris seluruh sistem dan dapat menerapkan suatu baris tanpa harus mencoba setiap baris tanpa berangkai (mencari perubahan dalam gabungan setiap cycle)
- Merupakan gabungan pola yang sangat cepat, yang mendapatkan kecepatannya dengan menyimpan informasi tentang baris dalam jaringan.



SP YANG BERSADARKAN ATURAN MODEREN

KLASIFIKASI PARADIGMA PEMROGRAMAN

1. PARADIGMA PROSEDURAL

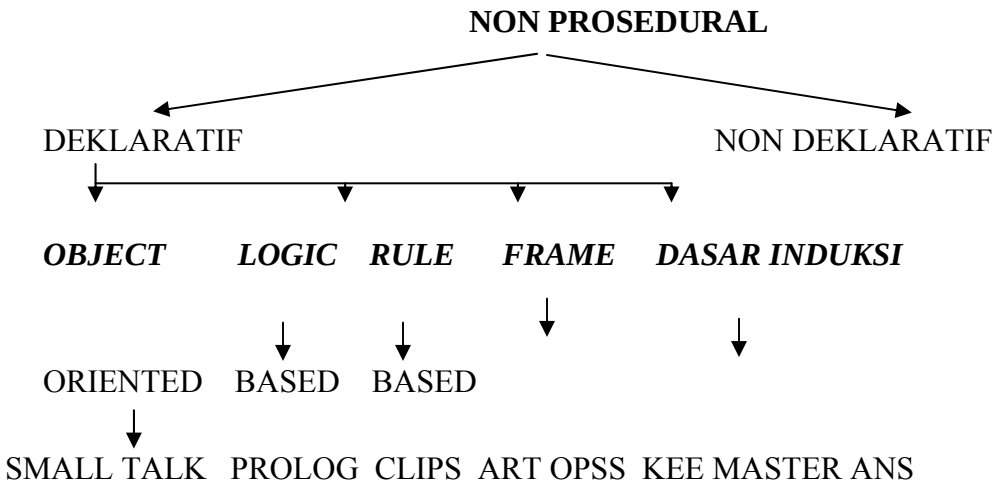


ALGORITMA adalah metode untuk pemecahan masalah dalam sejumlah tahap/langkah tertentu.

- ✓ Implementasi algoritma dalam suatu program disebut *program prosedural.*
- ✓ Pemrograman algoritma (prosedural) dan konvensional untuk program type non-AI.
- ✓ Sinonim untuk pemrograman prosedural adalah prog. Sequential.
- ✓ Pada pemrograman prosedural programmer harus menentukan sesungguhnya bagaimana pemecahan masalah harus di-code-kan.

- ✓ Pembuat code adalah pemrograman non prosedural.

2. PARADIGMA NON PROSEDURAL



- ✓ Penekanan pemrograman Non prosedural adalah penentuan apa yg akan diselesaikan dan membiarkan system menentukan bagaimana menyusunnya.

- ➢ PEMROGRAMAN DEKLARATIF
  Memisahkan tujuan dari metode yg digunakan untuk mencapai tujuan.

- ➢ PEMROGRAMAN OBJECT ORIENTED
  Ide : membuat dsign program dg mempertimbangkan data yg digunakan dalam

  program sebagai objek dan mengimplemnetasikan operasi pada objek tersebut.

PEMROGRAMAN LOGIKA Pembuktian teori logika dg Logic Theorirt Program

(Newell & Simon) pada Darmouth Conference A.I (1956)

Rangkaian backward dapat digunakan untuk mengekspresikan pengetahuan dalam representasi deklaratif maupun kontrol proses pemberian alasan.

Keuntungannya : pembuatannya dapat diproses secara paralel yaitu jika ada beberapa processor dapat bekerja secara simultan.

EXPERT SYSTEM

- ✓ Disebut *pemrograman deklaratif krn programmer tdk menetukan bagaimana prog. hrs mendapatkan tujuannya pada level algoritma*