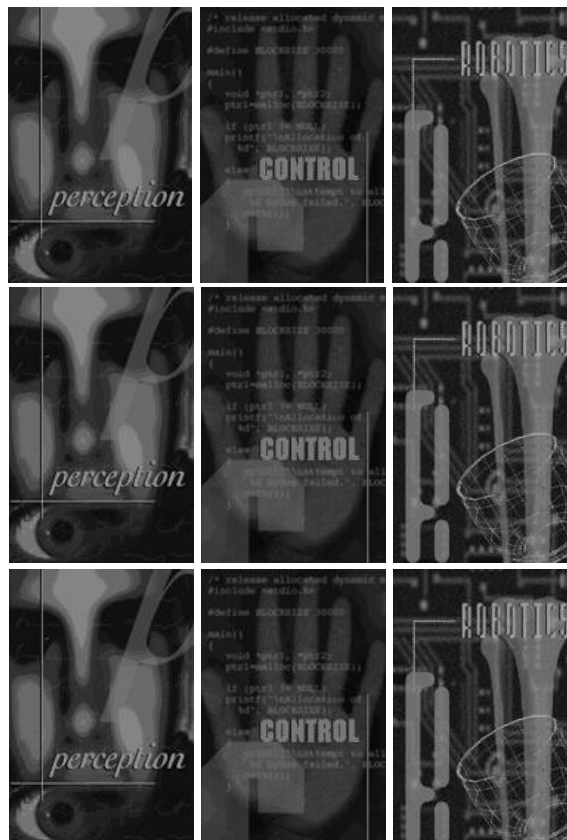


# Intelijensia Buatan

Suyanto, ST



**JURUSAN TEKNIK INFORMATIKA  
SEKOLAH TINGGI TEKNOLOGI TELKOM  
2002**

**Lembar Pengesahan**

# **Intelijensia Buatan**

**Oleh:  
Suyanto, ST**

**Telah diterima dan disahkan untuk dipergunakan sebagai  
Buku Ajar di STTTelkom**

**Bandung, Oktober 2002  
Ketua UPT Perpustakaan STTTelkom**

**(Danang Mursita, Drs., MT)**



## Kata Pengantar

Setelah setengah abad perkembangannya, disiplin ilmu intelijensia buatan belum menunjukkan hasil yang diharapkan. Masalah yang dihadapi adalah bagaimana merepresentasikan pengetahuan di dunia nyata menjadi basis pengetahuan (*knowledge base*) dalam *domain* komputer secara efektif dan efisien. Saat ini terdapat lima *logic, tools* matematika yang digunakan untuk membangun dan memanipulasi pengetahuan, yaitu *Propositional Logic, First Order Logic, Temporal Logic, Probability Theory*, dan *Fuzzy Logic*. Berbagai macam struktur pengetahuan diajukan, diantaranya *frame, scripts*, dan *conceptual dependency*. Teknik-teknik penyelesaian masalah (*problem solving*), dengan kelebihan dan kekurangannya masing-masing, telah dilakukan penelitian. Diantaranya adalah teknik *Searching, Reasoning, Planning*, dan *Adaptive Intelligence*. Tetapi, sampai saat ini struktur representasi pengetahuan dan teknik-teknik penyelesaian masalah tersebut belum banyak membantu menyelesaikan permasalahan yang kompleks dalam kehidupan manusia.

Buku Ajar berjudul Kecerdasan Buatan ini terdiri dari delapan bab, membahas teknik-teknik dan metode-metode dalam disiplin ilmu intelijensia buatan. Bagian pertama, bab 1, bab 2, dan bab 3, membahas ruang masalah dan teknik pencarian. Bagian ke dua, bab 4, membahas representasi pengetahuan menggunakan *Propositional logic* dan *First Order Logic* atau *Predicate Calculus*. Teknik penyelesaian masalah menggunakan *Planning* dibahas pada bab 5. Sistem intelijen adaptif (Jaringan Syaraf Tiruan, Algoritma Genetika, dan Sistem Fuzzy) dibahas secara teoritis dan contoh penerapannya pada bab 6. Bab 7 membahas Pengolahan Bahasa Alami (*Natural Language Processing*), sistem pengenalan bahasa alami yang ideal, permasalahan yang ada, dan metode-metode yang dapat digunakan. Pengetahuan kognitif, pengimplementasian teknik dan metode intelijensia buatan sebagai alat bantu dalam proses belajar manusia, dibahas dalam bab 8. Pada bab ini dibahas komponen-komponen yang terdapat dalam sistem pengajaran berbantuan komputer yang memiliki intelijensia.

Bandung, Oktober 2002

Penulis

## Bab 1

# Pendahuluan

Bab ini menjelaskan mengapa inteligensia buatan (*artificial intelligence*) menjadi subyek yang pantas dan berguna untuk dipelajari, dan di sini kita mencoba untuk memberikan definisi *inteligensia buatan* secara tepat sebelum kita mulai belajar *inteligensia buatan*.

Manusia mempunyai kapasitas mental dan pikiran yang sangat penting dalam kehidupannya. Bidang *inteligensia buatan* berusaha untuk memahami entitas-entitas cerdas. Satu alasan kenapa mempelajarinya adalah untuk lebih memahami diri kita sendiri. Tetapi tidak seperti filosofi dan psikologi, yang juga berhubungan dengan kecerdasan, AI berusaha membangun entitas-entitas cerdas sesuai dengan pemahaman manusia. Alasan lain mengapa mempelajari AI adalah bahwa entitas-entitas cerdas yang dibangun ini menarik dan berguna dalam kebenarannya sendiri. AI telah menghasilkan banyak produk yang berarti dan mengesankan bahkan pada tahap awal perkembangannya. Walaupun tidak seorangpun dapat memprediksi masa depannya secara detail, tetapi jelas bahwa komputer-komputer dengan kecerdasan setingkat manusia akan mempunyai pengaruh yang sangat besar pada kehidupan kita dan pada sebagian dari peradaban masa depan.

Perkembangan perangkat keras (*hardware*) saat ini sudah berada pada tingkat yang sangat tinggi. Perkembangan teknologi komputer yang kini masih ditunggu adalah dibuatnya komputer tanpa kawat (*wireless computer*). Dalam *wireless computer* data ditransfer sepenuhnya melalui berkas cahaya (*light beams*) yang dapat saling berpotongan tanpa terjadinya interferensi. Saat ini orang sudah mampu membuat processor yang sepenuhnya optikal. Suatu komputer optikal dalam bentuk awal (*rudimentary*), yang mampu mengerjakan matematika sederhana telah berhasil dibuat oleh para peneliti dari *University of Colorado* [CAR93]. Suatu komputer optikal yang beroperasi dengan kecepatan cahaya dan kemampuan komunikasi melalui ribuan jalur dalam serat optik akan meningkatkan kemampuan komputer yang luar biasa. Komputer yang demikian merupakan impian dan dambaan para peneliti. Namun dana yang sangat besar untuk pengembangan alat tersebut, saat ini, masih merupakan ganjalan utama dalam realisasinya. Suatu usaha yang realistis saat ini adalah mengembangkan komputer hibrida dengan mengkombinasikan teknologi elektronik dengan teknologi fotonik, yaitu dengan mengembangkan hubungan optik antara berbagai *electronic processor*, *memory units*, dan *communication ports*. Hal ini akan memberikan transfer data yang cepat dan sekaligus menghindarkan terjadinya *wiring bottle necks* yang banyak terjadi pada komputer elektronik saat ini. Banyak lagi yang diharapkan datang dari teknologi optoelektronik, namun yang sudah jelas dalam teknologi tersebut adalah perubahan radikal dalam cara memproses informasi. Unit informasi dalam komputer optik tidak lagi bit-bit seperti pada komputer elektronik, tetapi suatu *image* yang berisi miliaran bit.

Prosesor optik dapat menumpuk dua buah image dan dapat segera, dengan sangat cepat, mengenali perbedaannya. Hal ini penting dalam pengenalan pola dan bila dikombinasikan dengan lensa dapat berfungsi sebagai mata.

Melihat perkembangan perangkat keras komputer yang sangat pesat, maka muncullah pertanyaan: “Masih perlukan kita mempelajari *intelijensia buatan*?”. Saat ini, kita merasa masih perlu karena komputer teknologi fotonik yang diharapkan masih jauh dari rencana fabrikasi dan tentu saja harganya sangat tinggi. Sedangkan, saat ini, yang kita butuhkan adalah sistem berintelijensia dengan *resource* dan harga yang murah.

Selanjutnya, marilah kita mencoba memberikan definisi *intelijensia buatan*. Definisi *intelijensia buatan* dari delapan *textbook* terakhir terlihat pada gambar 1.1 di bawah ini.

<p>“<i>The exciting new effort to make computers think ... machines with minds, in the full and literal sense</i>” (Haugeland, 1985)</p> <p>“<i>[The automation of] activities that we associate with human thinking, activities such as decision making, problem solving, learning ...</i>” (Bellman, 1978)</p>	<p>“<i>The study of mental faculties through the use of computational models</i>” (Charniak and McDermott, 1985)</p> <p>“<i>The study of the computations that make it possible to perceive, reason, and act</i>” (Winston, 1992)</p>
<p>“<i>The art of creating machines that perform functions that require intelligence when performed by people</i>” (Kurzweil, 1990)</p> <p>“<i>The study of how to make computers do things which, at the moment, people do better</i>” (Rich and Knight, 1991).</p>	<p><i>A field of study that seeks to explain and emulate intelligent behavior in term of computational processes</i>” (Schalkoff, 1990)</p> <p>“<i>The branch of computer science that is concerned with the automation of intelligent behavior</i>” (Luger and Stubblefield, 1993)</p>

**Gambar 1.1** Definisi-definisi *intelijensia buatan* yang diorganisasikan ke dalam empat katagori:

<i>Systems that think like humans.</i>	<i>Systems that think rationally.</i>
<i>Systems that act like humans.</i>	<i>Systems that act rationally.</i>

***Thinking Humanly: The cognitive modeling approach.***

1. Melalui introspeksi: mencoba menangkap pemikiran-pemikiran kita sendiri pada saat kita berpikir. Tetapi seorang psikolog Barat mengatakan: “*how do you know that you understand?*”. Bagaimana anda sadar bahwa anda sedang sadar? Karena pada saat anda menyadari pemikiran anda, ternyata pemikiran tersebut sudah lewat dan digantikan kesadaran anda.
2. Melalui eksperimen-eksperimen psikologi.

***Acting Humanly: The Turing test approach.***

Alan Turing (1950) merancang suatu ujian bagi komputer berintelijensia apakah mampu mengelabui seorang manusia yang menginterogasi komputer tersebut melalui *teletype*. Jika *interrogator* tidak dapat membedakan apakah yang diinterogasi adalah manusia atau komputer, maka komputer berintelijensia tersebut lolos dari *Turing test*. Komputer tersebut perlu memiliki kemampuan: *Natural Language Processing, Knowledge Representation, Automated Reasoning, Machine Learning, Computer Vision, Robotics*. *Turing Test* sengaja menghindari interaksi fisik antara *interrogator* dan komputer karena simulasi fisik manusia tidak memerlukan intelijensia.

***Thinking Rationally: The laws of thought approach.***

Terdapat dua masalah dalam pendekatan ini, yaitu:

1. tidak mudah untuk membuat pengetahuan informal dan menyatakannya dalam *formal term* yang diperlukan oleh notasi logika, khususnya ketika pengetahuan tersebut memiliki kepastian kurang dari 100%.
2. Terdapat perbedaan besar antara dapat memecahkan masalah “dalam prinsip” dan dan juga memecahkannya dalam praktek.

***Acting Rationally: The rational agent approach.***

Membuat inferensi yang benar kadang-kadang merupakan bagian dari suatu *rational agent*, karena satu cara untuk melakukan aksi secara rasional adalah menalar secara logika untuk mendapatkan kesimpulan bahwa aksi yang diberikan akan mencapai tujuan, dan kemudian melakukan aksi atas kesimpulan tersebut. Dalam buku ajar ini, kita akan menggunakan pendekatan *rational agent*. Hal ini akan mebatasi bahasan kita pada teknik-teknik rasional pada *intelijensia buatan*. Sedangkan aksi dan pikiran manusia yang diluar rasio (refleks dan intuitif) belum dapat ditirukan oleh komputer.

## Bab 2

# Teknik Pencarian

Bab ini membahas bagaimana membuat ruang masalah untuk suatu masalah tertentu. Sebagian masalah mempunyai ruang masalah yang dapat diprediksi, sebagian lainnya tidak.

### 1.1 Pendefinisian Masalah Sebagai Pencarian Ruang Keadaan

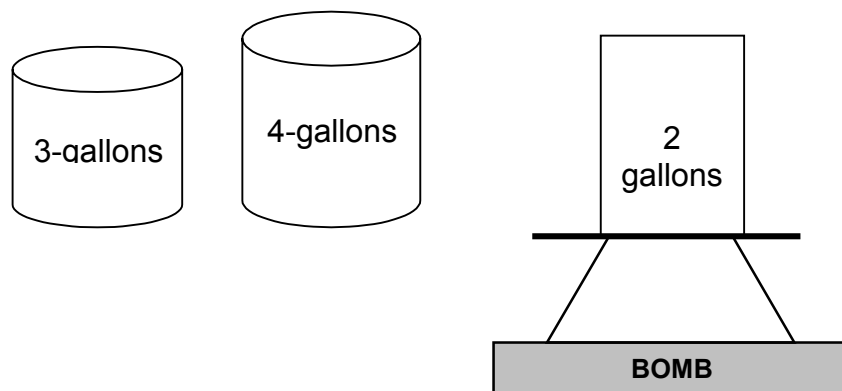
Masalah utama dalam membangun sistem berbasis AI adalah bagaimana mengkonversikan situasi yang diberikan ke dalam situasi lain yang diinginkan menggunakan sekumpulan operasi tertentu.

#### *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-operartor (aturan produksi) yang digunakan untuk memecahkan masalah terlihat pada gambar 2.1.



**Gambar 2.1** Suatu masalah: *Water jug Problem*.



1.  $(x,y)$   
If  $x < 4$   $\rightarrow (4,y)$  Isi penuh gelas 4 galon
2.  $(x,y)$   
If  $y < 3$   $\rightarrow (x,3)$  Isi penuh gelas 3 galon
3.  $(x,y)$   
If  $x > 0$   $\rightarrow (x-d,y)$  Buang sebagian air dari gelas 4 galon
4.  $(x,y)$   
If  $y > 0$   $\rightarrow (x,y-d)$  Buang sebagian air dari galon ukuran 3 galon
5.  $(x,y)$   
If  $x > 0$   $\rightarrow (0,y)$  Kosongkan gelas 4 galon
6.  $(x,y)$   
If  $y > 0$   $\rightarrow (x,0)$  Kosongkan gelas 3 galon
7.  $(x,y)$   
If  $x+y \geq 4$  and  $y > 0$   $\rightarrow (4,y-(4-x))$  Tuangkan air dari gelas 3 galon ke gelas 4 galon sampai gelas 4 galon penuh
8.  $(x,y)$   
If  $x+y \geq 3$  and  $x > 0$   $\rightarrow (x-(3-y),3)$  Tuangkan air dari gelas 4 galon ke gelas 3 galon sampai gelas 3 galon penuh
9.  $(x,y)$   
If  $x+y \leq 4$  and  $y > 0$   $\rightarrow (x+y,0)$  Tuangkan seluruh air dari gelas 3 galon ke gelas 4 galon
10.  $(x,y)$   
If  $x+y \leq 3$  and  $x > 0$   $\rightarrow (0,x+y)$  Tuangkan seluruh air dari gelas 4 galon ke gelas 3 galon
11.  $(0,2)$   $\rightarrow (2,0)$  Tuangkan 2 galon air dari gelas 3 galon ke gelas 4 galon
12.  $(2,y)$   $\rightarrow (0,y)$  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*.

## 1.2 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 dilakukan 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*).

### 2.2.1 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 sistematis. Beberapa strategi kontrol yang sistematis 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.

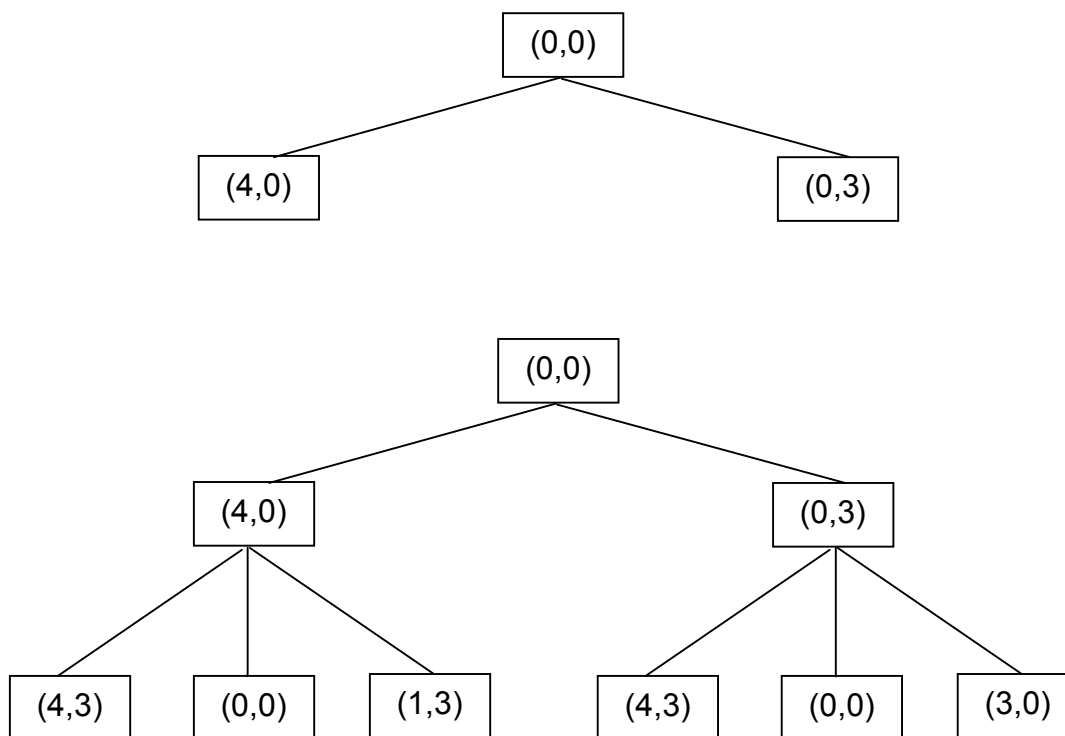
### 2.2.2 Strategi Pencarian

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?

## 1. 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 2.2). 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*.

## 2. Uniform Cost Search (UCS)

Konsepnya hampir sama dengan BFS, bedanya adalah bahwa BFS menggunakan urutan level dari yang paling rendah sampai yang paling tinggi. Sedangkan UCS menggunakan harga terendah yang dihitung berdasarkan harga dari node asal menuju ke node tersebut atau biasa dilambangkan sebagai  $g(n)$ . BFS adalah juga UCS jika harga  $g(n) = \text{DEPTH}(n)$ .

Syarat yang harus dipenuhi oleh pohon UCS:  $g(\text{SUCCESSOR}(n)) \geq g(n)$  untuk setiap node  $n$ . Jika syarat ini tidak dipenuhi maka UCS tidak bisa dipakai.



### 3. Depth-First Search (DFS)

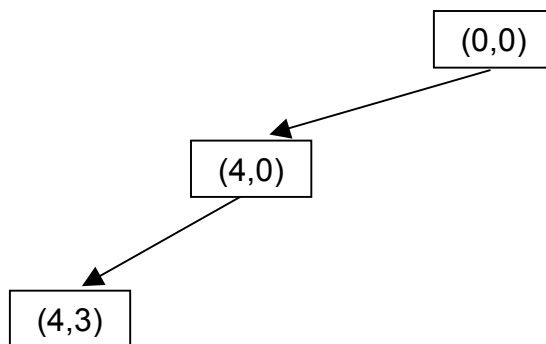
Pencarian dilakukan pada satu node dalam setiap level dari yang paling kiri. Jika pada level yang paling dalam, solusi belum ditemukan, maka pencarian dilanjutkan pada node sebelah kanan. Node yang kiri dapat dihapus dari memori. Jika pada level yang paling dalam tidak ditemukan solusi, maka pencarian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

Kelebihan DFS adalah:

- Pemakaian 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*.

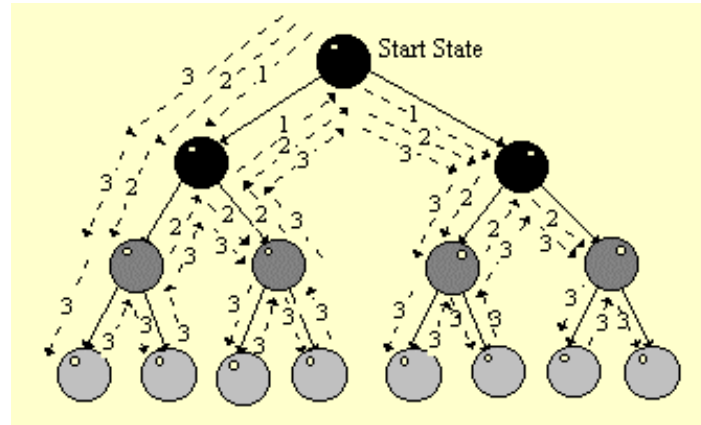
### 4. Depth-Limited Search (DLS)

Mengatasi kelemahan DFS (**tidak complete**) dengan membatasi kedalaman maksimum dari suatu jalur solusi. Tetapi harus diketahui atau ada batasan dari sistem tentang level maksimum. Jika batasan kedalaman terlalu kecil, DLS tidak **complete**.

### 5. Iterative-Deepening Depth-First Search (IDS)

Merupakan metode yang berusaha menggabungkan keuntungan BFS (**Complete** dan **Optimal**) dengan keuntungan DFS (**Space Complexity** yang rendah). Tetapi konsekuensinya adalah Time Complexity-nya menjadi tinggi. Perhatikan gambar 2.6. Pencarian dilakukan secara iteratif (menggunakan penelusuran DFS) dimulai dari

batasan level 1. Jika belum ditemukan solusi, maka dilakukan iterasi ke-2 dengan batasan level 2. Demikian seterusnya sampai ditemukan solusi. Jika solusi ditemukan maka tidak diperlukan proses *backtracking* (penelusuran balik untuk mendapatkan jalur yang diinginkan).

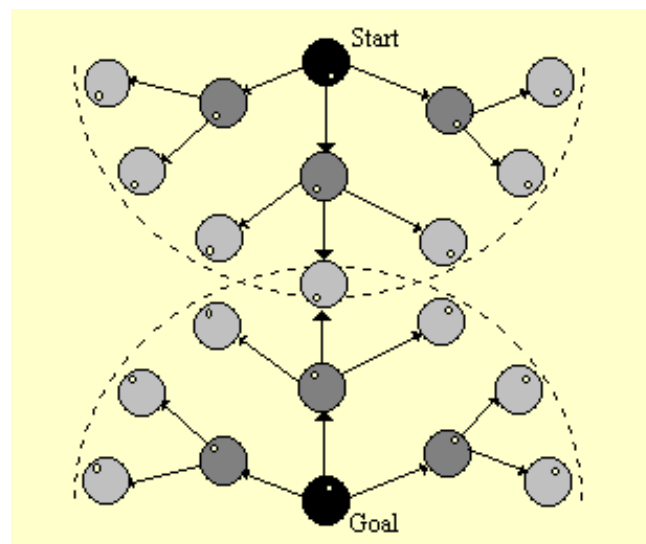


**Gambar 2.6** Penelusuran *Iterative Deepening DFS*.

## 6. Bi-Directional Search

Pada setiap iterasi, pencarian dilakukan dari dua arah: dari node asal (*start*) dan dari node tujuan (*goal*). Ketika dua arah pencarian membangkitkan node yang sama, maka solusi telah ditemukan, dengan cara menggabungkan kedua jalur yang bertemu. Ada beberapa masalah sebelum memutuskan untuk menggunakan strategi *Bi-directional Search*, yaitu:

- Bagaimana kalau terdapat beberapa node tujuan yang berbeda?
- Terdapat perhitungan yang tidak efisien untuk selalu mengecek apakah node baru yang dibangkitkan sudah pernah dibangkitkan oleh pencarian dari arah yang berlawanan.
- Bagaimana menentukan strategi pencarian untuk kedua arah tersebut? Misalnya dari arah sumber dan dari arah tujuan digunakan BFS.



**Gambar 2.7** Penelusuran *Bi-directional Search*.  
**Perbandingan Strategi Pencarian [RUS95]**

Criterion	Breadth First	Uniform Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	$b^d$	$b^d$	$b^m$	$b^l$	$b^d$	$b^{d/2}$
Space	$b^d$	$b^d$	$bm$	$bl$	$bd$	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

**Keterangan:**

$b$  : faktor pencabangan (the branching factor)

$d$  : kedalaman solusi (the depth of solution)

$m$  : kedalaman maksimum pohon pencarian (the maximum depth of the search tree)

$l$  : batasan kedalaman (the depth limit)

## Bab 3

# Pencarian Heuristik

Bab 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*.

### 3.1 *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 teknik *generate & test* adalah teknik yang layak. Sebagai contoh, pada teka-teki yang terdiri dari empat kubus segi enam, dengan masing-masing 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 masing-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 dibagikan 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.



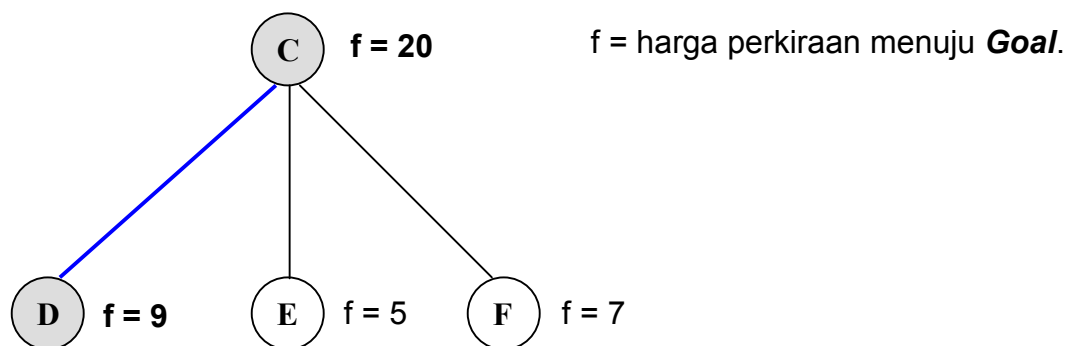
## 3.2 Hill Climbing

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 *heuristics*-nya adalah jarak antara lokasi sekarang dengan gedung yang tinggi dan *state* yang diperlukan adalah jarak yang terpendek.

### 3.2.1 Simple HC

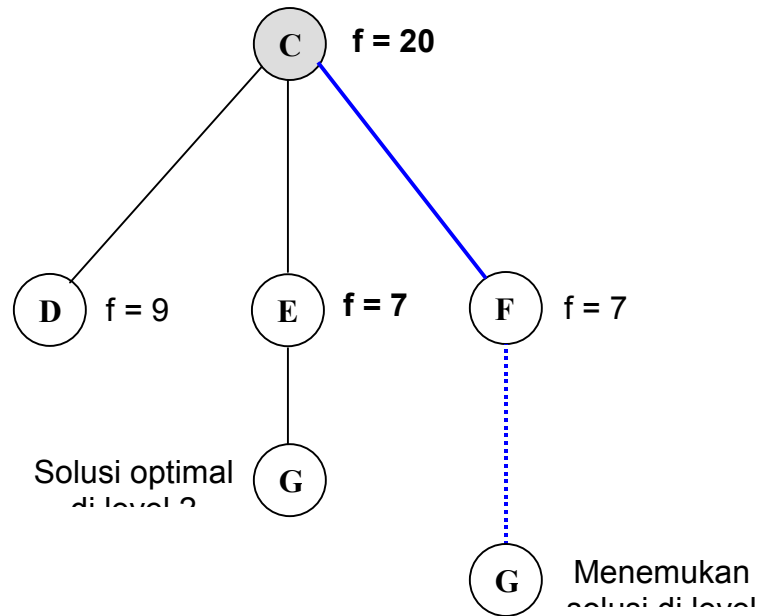
#### Algorithm: Simple HC

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*.

### 3.2.2 Steepest-Ascent HC



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

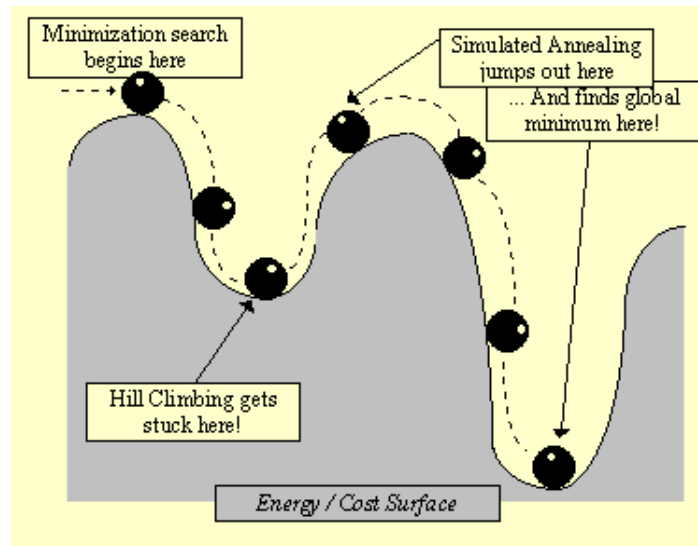
Pada gambar 3.2 di atas, terjadi ambiguitas dimana fungsi heuristik node E dan node F adalah sama. Misalkan dipilih F dan ternyata menemukan solusi di level 8. Padahal terdapat solusi lain yang lebih optimal di level 2. Hal ini dikatakan bahwa *Steepest-Ascent Hill Climbing* terjebak pada solusi lokal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

### 3.3 Simulated Annealing

- SA memanfaatkan analogi antara cara pendinginan dan pembekuan metal menjadi sebuah struktur crystal dengan energi yang minimal (proses penguatan) dan pencarian untuk *state* tujuan minimal dalam proses pencarian.
- Tidak seperti pendekatan HC (dengan jalan mengikuti sebuah turunan yang tetap dalam meminimisasi masalah), SA lebih banyak menjadi jebakan pada *local minima*.
- Pada Seperti diilustrasikan oleh gambar 3.3, *simulated annealing* berusaha keluar dari jebakan *minimum local*.



Gambar 3.3 *Simulated Annealing* berusaha keluar dari jebakan *minimum local*.

#### Random Search pada SA:

- Algoritma SA menggunakan pencarian acak yang tidak hanya menerima perubahan yang mengurangi fungsi energi (sebagai harapan : fungsi ini mungkin memberikan resiko perjalanan antara dua kota), tapi juga beberapa perubahan yang menambah fungsi nilai, kemudian SA ijin untuk melompat keluar dari local minima.
- Probabilitas dengan SA yang akan menerima sebuah penambahan dalam energi system DE adalah dinyatakan sebagai berikut:

$$p(\Delta E) = \frac{1}{e^{\Delta E/T}}$$

- Kenapa fungsi ini? Mengadopsi dari fisika, dimana fungsi ini merepresentasikan distribusi Boltzman dari energi dalam system termodinamik. Sehingga didapat

- persamaan probabilitas dari level energi yang diberikan dalam system pada temperatur  $T$ .
- Dugaan dari temperatur sistem adalah *intrinsic* pada proses SA. Dengan penurunan temperatur yang lambat dari awal system acak, kita mendorong elemen-elemen dari system disimpulkan dengan sebuah pesan, paling sedikit susunan energi. Dalam mencari proses terminal, sebuah pendinginan yang lambat dapat kemudian menuju sebuah *state* opsional.

Algoritma untuk mensimulasikan penguatan sedikit berbeda dengan prosedur *simple HC*, yaitu:

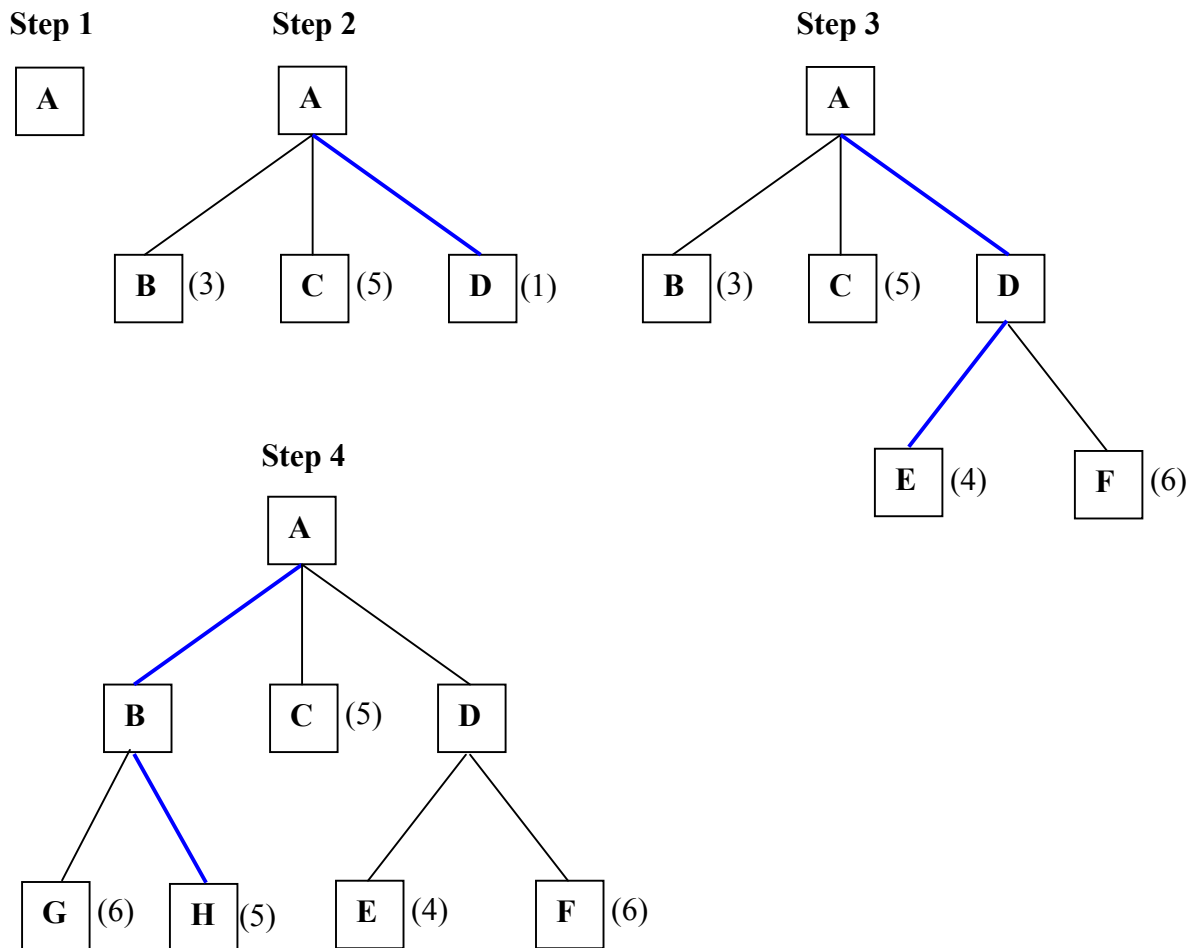
- Jadwal penguatan harus di-*maintain*.
- Pindah ke *state* yang lebih jelek mungkin diterima.
- Ini adalah ide yang baik untuk maintain, untuk menambahkan *current state*, *state* terbaik ditemukan terlalu jauh. Maka, jika *state* tujuan lebih jelek daripada *state* sebelumnya (karena kegagalan dalam penerimaan pemindahan *worse state*), *state* sebelumnya masih tersedia.

**Algoritma: *Simulated Annealing***

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. Initialize *BEST-SO-FAR* to the current state.
3. Initialize  $T$  according to the annealing schedule.
4. 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. Compute
 
$$\Delta E = (\text{value of } current) - (\text{value of new } state)$$
    - If the new state is a goal, then return it and quit.
    - If it is not a goal state but is better than the current state, then make it the current state. Also set *BEST-SO-FAR* to this new state.
    - If it is not better than the current state, then make it the current state with probability  $p'$  as defined above. This step is usually implemented by invoking a random number generator to produce a number in the range  $[0,1]$ . If that number is less than  $p'$ , then the move is accepted. Otherwise do nothing.
  - c) Revise  $T$  as necessary according to the annealing schedule.
5. Return *BEST-SO-FAR* as the answer.

### 3.4 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 dibangkitkan, 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 dibangkitkan 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

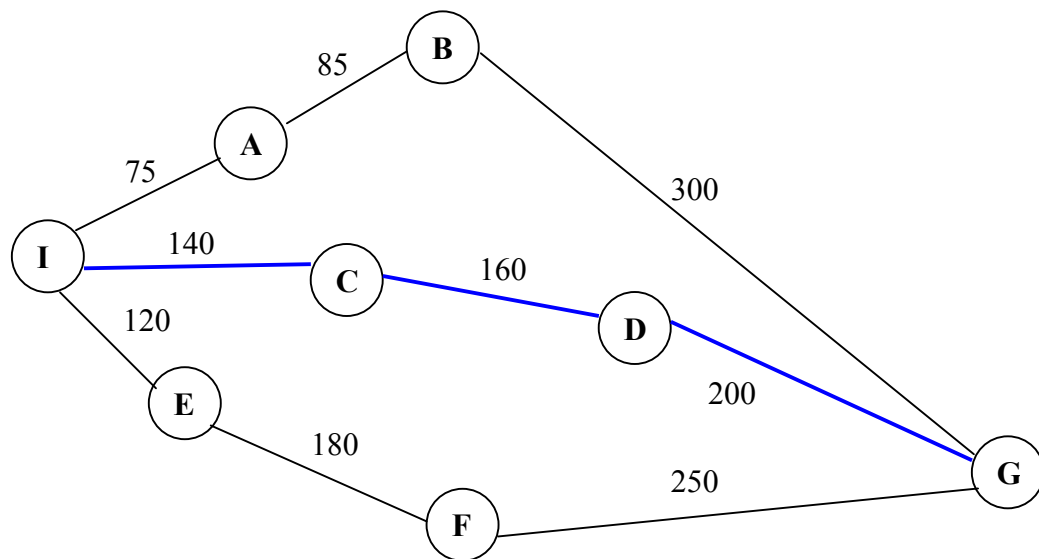
dievaluasi dan CLOSE untuk mengelola node-node yang pernah dibangkitkan dan sudah dievaluasi. Algoritma selengkapnya adalah sebagai berikut:

**Algoritma Best-First 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.

### 3.4.1 Greedy Search

Merupakan Best First Search dengan hanya mempertimbangkan harga perkiraan (*estimated cost*). Sedangkan harga sesungguhnya tidak digunakan. Sehingga solusi yang dihasilkan **tidak optimal**. Perhatikan contoh kasus di bawah ini.



**Gambar 3.5 Studi kasus 1:** pencarian jalur dalam suatu daerah yang direpresentasikan dalam suatu graph. Node menyatakan kota dan busur menyatakan jarak antar kota (harga sesungguhnya) dan  $h'(n)$  adalah harga perkiraan dari node  $n$  menuju node tujuan (G).

Dengan  $h'(n)$  = fungsi heuristik (jarak garis lurus dari node n menuju G).

I	A	B	C	D	E	F
400	360	280	300	180	400	200

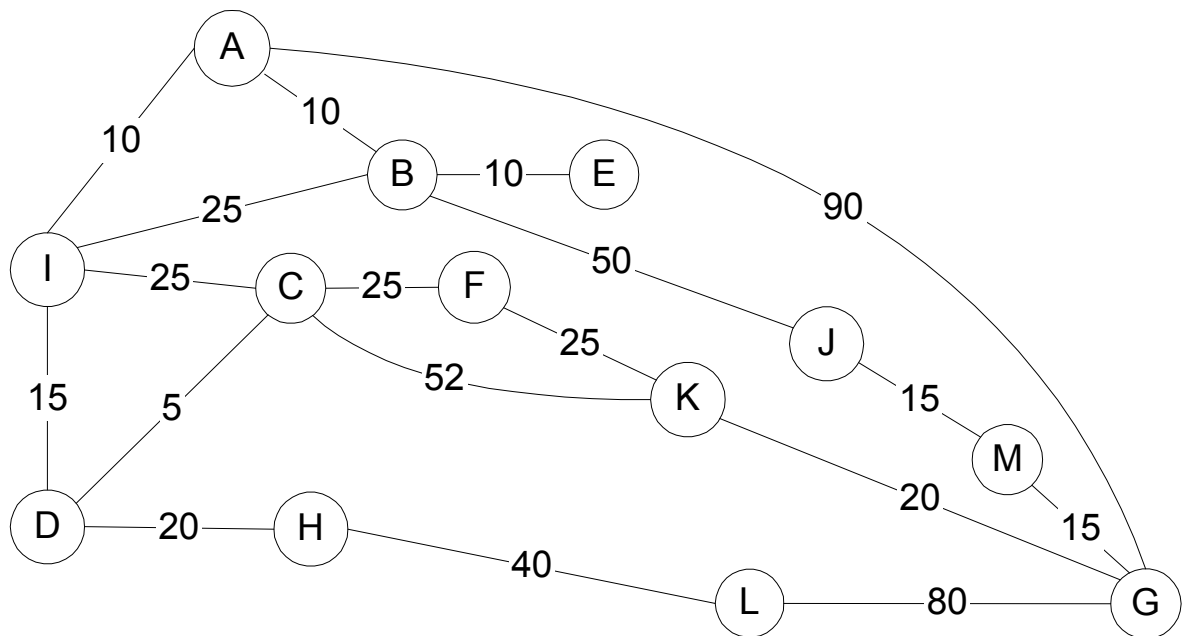
Pada kasus di atas, jalur yang terpilih adalah: I-C-D-G =  $140 + 160 + 200 = 500$ .

Padahal jalur terbaik adalah: I-A-B-G =  $75 + 85 + 300 = 460$ .

### 3.4.2 A Bintang (A\*)

Merupakan algoritma Best First Search dengan menggabungkan *Uniform Cost Search* dan *Greedy Search*. Harga yang dipertimbangkan didapat dari harga sesungguhnya ditambah dengan harga perkiraan. Dalam notasi matematika dituliskan:  $f'(n) = g(n) + h'(n)$ . Dengan penggunaan  $f'(n)$ , maka algoritma A\* adalah **Complete** dan **Optimal**. Penggunaan algoritma A\* untuk studi kasus 1 pada gambar 3.5 dihasilkan jalur yang paling optimal, yakni: I-A-B-G dengan jarak 460.

Pada algoritma ini juga digunakan dua senarai, yaitu: OPEN dan CLOSE. Untuk lebih memahami algoritma A\*, perhatikan studi kasus 2 pada gambar 3.6 berikut ini. Pada saat A sebagai *Best Node*, maka akan dibangkitkan B dan G. Karena B sudah berada di CLOSE maka akan dicek apakah parent dari B harus diganti atau tidak. Karena  $g(B)$  melalui A ( $10 + 10 = 20$ ) lebih kecil daripada  $g(B)$  melalui I (25), maka parent dari B harus diganti. Begitu juga dengan  $f'(B)$  yang tadinya 85 menjadi 80. Kemudian akan dilakukan propagasi ke semua suksesor B. Dalam hal ini adalah node E yang semula  $g(E) = 35$  menjadi 30 dan  $f'(E)$  yang semula 105 menjadi 100, dan node J yang semula  $g(J) = 75$  menjadi 70 dan  $f'(J)$  yang semula 95 menjadi 90.



Dengan  $h'(n)$  = fungsi heuristik (Jarak garis lurus dari node  $n$  menuju node  $G$ ) adalah sebagai berikut:

**Gambar 3.6 Studi kasus 2:** pencarian jalur dalam suatu daerah yang direpresentasikan dalam suatu graph. Node menyatakan kota dan busur menyatakan jarak antar kota (harga sesungguhnya)

Node	A	B	C	D	E	F	H	I	K	L	M
$h'(n)$	80	60	40	5	70	40	80	20	20	20	5



### Algoritma A\*:

```
1. OPEN = node asal
   CLOSE array kosong
    $g = 0$ 
    $f' = h'$ 

2. Ulangi sampai node tujuan ditemukan
   If OPEN kosong then
     Gagal
   Else
     BestNode = node yang ada di OPEN dengan  $f'$  minimal
     Pindahkan node terbaik tersebut dari OPEN ke CLOSE
     If BestNode = goal then
       Sukses
     Else
       Bangkitkan semua suksesor BestNode tapi jangan buat pointer
       Untuk setiap suksesor kerjakan:
       Hitung  $g(\text{suksesor}) = g(\text{BestNode}) + \text{actual cost}(\text{dari BestNode ke suksesor})$ 
       {Periksa suksesor}
       If suksesor ada di OPEN then {sudah pernah digenerate tapi belum diproses}
         OLD = isi OPEN tersebut
         Tambahkan OLD sebagai suksesor BestNode
         Buat pointer dari OLD ke BestNode
         Bandingkan nilai  $g(\text{OLD})$  dengan  $g(\text{isi OPEN})$ 
         If  $g(\text{OLD})$  lebih baik then
           Ubah parent isi OPEN ke BestNode
           Ubah nilai  $g$  dan  $f'$  pada isi OPEN
         End
       Else
         If suksesor ada di CLOSE then {sudah pernah digenerate dan sudah diproses}
           OLD = isi CLOSE
           Tambahkan OLD sebagai suksesor BestNode
           Bandingkan nilai  $g(\text{OLD})$  dengan  $g(\text{isi CLOSE})$ 
           If  $g(\text{OLD})$  lebih baik then
             Ubah parent isi CLOSE ke BestNode
             Ubah nilai  $g$  dan  $f'$  pada isi CLOSE
             Propagasi untuk semua suksesor OLD dengan penelusuran DFS dengan aturan:
             Ulangi sampai node suksesor tidak ada di OPEN atau node tidak punya suksesor
             If suksesor ada di OPEN then
               Propagasi diteruskan
             Else
               If nilai  $g$  via suksesor lebih baik then
                 Propagasi diteruskan
               Else
                 Propagasi dihentikan
             End
           End
         End
       Else {suksesor tidak ada di OPEN maupun CLOSE}
         Masukkan suksesor ke OPEN
         Tambahkan suksesor tersebut sebagai suksesor BestNode
         Hitung  $f' = g(\text{suksesor}) + h'(\text{suksesor})$ 
       End
     End
   End
End
```

### 3.5 Memory Bounded Search

Untuk masalah tertentu, dimana *memory* komputer (RAM) terbatas, algoritma A\* tidak mampu menyelesaikannya. Terdapat dua algoritma modifikasi A\*, untuk mengatasi masalah *memory*, yaitu:

#### 3.5.1 Iterative Deepening A\* (IDA\*)

Iterative Deepening Depth-First Search menggunakan batasan level (kedalaman) dalam setiap iterasi pencariannya. Di sini ide tersebut diterapkan pada algoritma A\* dengan membatasi fungsi heuristiknya. Proses pencarian pada setiap iterasi akan mengembalikan nilai *f-limit* yang baru yang akan digunakan sebagai batasan pencarian untuk iterasi berikutnya[RUS95]. Sama seperti A\*, IDA\* adalah **complete** dan **optimal**.

#### Algoritma IDA\*

**function** IDA\*(problem) **returns** a solution sequence

**inputs:** *problem*, a problem

**local variables:** *f-limit*, the current *f*-Cost limit  
*root*, a node

*root*  $\leftarrow$  MAKE-NODE(INITIAL-STATE[*problem*])

*f-limit*  $\leftarrow$  *f*-Cost(*root*)

**loop do**

*solution, f-limit*  $\leftarrow$  DFS-CONTOUR(*root, f-limit*)

**if** *solution* is non-null **then** return *solution*

**if** *f-limit* is INFINITE **then** return failure

**end**

**function** DFS-CONTOUR(*node, f-limit*) **returns** a solution sequence and a new *f*-Cost limit

**inputs:** *node*, a node

*f-limit*, the current *f*-Cost limit

**local variables:** *next-f*, the *f*-Cost limit for the next contour, initially INFINITE

**if** *f*-Cost[*node*] > *f-limit* **then** return null, *f*-Cost[*node*]

**if** GOAL-TEST[problem](STATE[*node*]) **then** return *node, f-limit*

**for** each node *s* in SUCCESSOR(*node*) **do**

*solution, new-f*  $\leftarrow$  DFS-CONTOUR(*s, f-limit*)

**if** *solution* is non-null **then** return *solution, f-limit*

*next-f*  $\leftarrow$  MIN(*next-f, new-f*)

**end**

**return** null, *next-f*

3.5.

2

SM

A\*

:

Si

mp

lifi

ed

Me

mo

ry

Bo

un

de

d

A\*

Berlawanan den

sua

tu

jalu

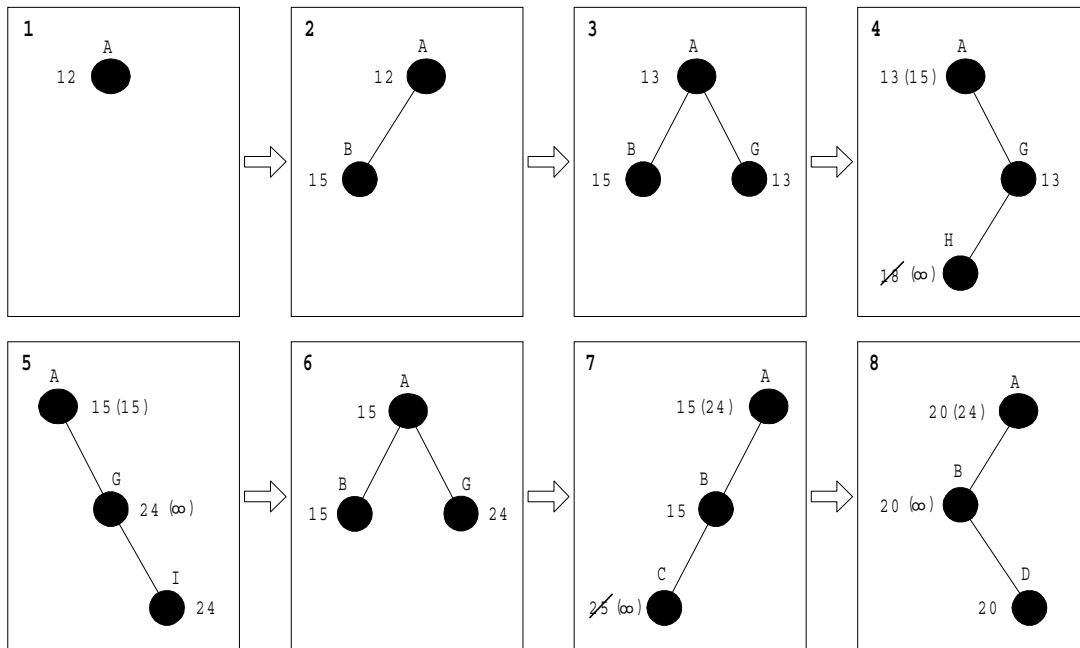
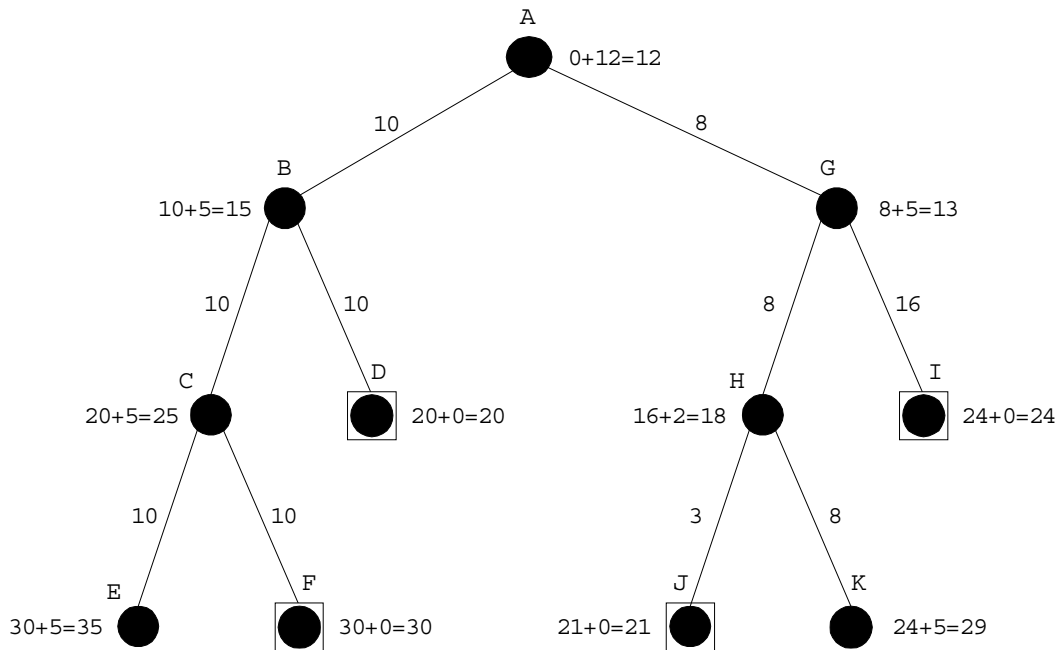
r

pali

ng

bag

us diantara jalur-jalur yang ada dalam batasan node-node tersebut. Perhatikan gambar 3.7 berikut ini.



**Gambar 3.7** Pencarian jalur dalam suatu graph menggunakan algoritma SMA\*.

Gambar 3.7 di atas menjelaskan bagaimana algoritma SMA\* melakukan pencarian jalur dengan dibatasi maksimum 3 node yang boleh tersimpan di memori. A adalah node asal, dan D, I, F, J adalah node-node yang bisa menjadi node tujuan. Dengan batasan 3 node, maka SMA\* dapat menemukan solusi yang paling optimal, yaitu A - B - D. Jika harga node J adalah 19 (lebih kecil dari harga node D (20)), maka SMA\* gagal menemukan solusi optimal. Kecuali jika jumlah node maksimum yang boleh tersimpan di memori adalah 4.

### Algoritma SMA\*

```

function SMA*(problem) returns a solution sequence
  inputs: problem, a problem
  local variables: Queue, a queue of nodes ordered by f-cost

  Queue  $\leftarrow$  MAKE-QUEUE({MAKE-NODE(INITIAL-STATE[problem])})
  loop do
    if Queue is empty then return failure
    n  $\leftarrow$  deepest least f-cost node in Queue
    if GOAL-TEST(n) then return success
    s  $\leftarrow$  NEXT-SUCCESSOR(n)
    if s is not a goal and is at maximum depth then
       $f(s) \leftarrow$  INFINITE
    else
       $f(s) \leftarrow$  MAX( $f(n)$ ,  $g(s) + h(s)$ )
    if all of n's successors have been generated then
      update n's f-cost and those of its ancestors if necessary
    if SUCCESSORS(n) all in memory then remove n from Queue
    if memory is full then
      delete shallowest, highest f-cost node in memory
      remove it from its parent's successor list
      insert its parent on Queue if necessary
    insert s in Queue
  end

```

### 3.6 Variasi A\*

Berhubungan dengan *bi-directional search*, maka dapat dilakukan sedikit variasi dari algoritma A\*. Kita akan membahas dua variasi algoritma A\*, yakni *Bi-directional A\** dan *Modified Bi-directional A\**.

#### 3.6.1 Bi-directional A\*

Ide dasarnya adalah melakukan pencarian menggunakan A\* dari dua arah, yakni dari arah node asal dan dari arah node tujuan. Pencarian dihentikan jika BestNode dari arah node asal telah berada dalam senarai CLOSE dari arah node tujuan. Kemudian dilakukan pengecekan apakah harus mengganti *parent*-nya BestNode tersebut dari arah node tujuan. Atau sebaliknya, pencarian dihentikan jika BestNode dari arah node tujuan telah berada dalam senarai CLOSE dari arah node asal.



#### Terminologi:

*Graph* berarah dilambangkan  $G = (V, E)$

$V$  : himpunan *node-node* dalam *graph* (Vertex).

$E$  : himpunan sisi-sisi yang menghubungkan *node-node* dalam  $V$ .

$e(u, v)$  : sisi-sisi dalam *graph*  $E$

$l(u, v)$  : panjang  $e(u, v)$  dalam  $E$ .

$p_s(u)$  : jarak antara *node*  $s$  dengan *node*  $u$  (nilai potensial *node*  $u$  terhadap *node*  $s$ ).

$r(s, d)$  : rute dari  $s$  (*source*) ke  $d$  (*destination*).

Fungsi *heuristic* pada A\* dapat dituliskan dalam bentuk lain:

$$f = l(s, u) + l(u, v) + h_s(v)$$

dimana:

$$g = l(s,u) + l(u,v)$$

$$h' = h_s(v)$$

Dengan  $h_s(v)$  adalah estimasi jarak dari node  $v$  ke node  $d$ .

### Algoritma Bi-directional A\*

1. Himpunan  $S, T \in \phi$ ; nilai  $s$  dan  $d$  tidak boleh  $+\infty$ ; tentukan  $p_s(s) = 0$  dan  $p_d(d) = 0$ .
2. Cari node  $u_s$  yang memiliki nilai  $p_s(v_s) + h_s(v_s)$  terkecil di  $S$  dan tambahkan  $u_s$  ke  $S$ . Jika  $u_s$  di  $D$ , lakukan langkah 7.
3. Untuk semua node  $v_s$  dimana sisi  $(u_s, v_s)$  di dalam  $E$ , jika  $p_s(u_s) + l(u_s, v_s)$  lebih kecil dari  $p_s(v_s)$ : ganti rute  $(s, v_s)$  dengan rute  $(s, u_s) +$  sisi  $(u_s, v_s)$  dan ganti nilai  $p_s(v_s) = p_s(u_s) + l(u_s, v_s)$  dan hapus  $v_s$  dari  $S$  jika  $v_s$  di dalam  $S$ .
4. Cari node  $u_d$  yang memiliki nilai  $p_d(v_d) + h_d(v_d)$  terkecil di  $D$  dan tambahkan  $u_d$  ke  $D$ . Jika  $u_d$  di  $S$ , lakukan langkah 7.
5. Untuk semua node  $v_d$  dimana sisi  $(v_d, u_d)$  di dalam  $E$ , jika  $l(v_d, u_d) + p_d(u_d)$  lebih kecil dari  $p_d(v_d)$ : ganti rute  $(v_d, d)$  dengan sisi  $(v_d, u_d) +$  rute  $(u_d, d)$  dan ganti nilai  $p_d(v_d) = l(v_d, u_d) + p_d(u_d)$  dan hapus  $v_d$  dari  $D$  jika  $v_d$  di dalam  $D$ .
6. Kembali ke langkah 2.
7. Cari sisi  $(u, v)$  dengan meminimalisasi  $p_s(u) + l(u, v) + p_d(v)$  dimana  $u$  di dalam  $S$  dan  $v$  di dalam  $D$ . Jika  $p_s(u) + l(u, v) + p_d(v) < p_s(u_s) + p_d(u_d)$  maka rute terpendek  $(s, d) = \text{rute}(s, u) + \text{sisi}(u, v) + \text{rute}(v, d)$  jika tidak, maka rute terpendek  $(s, d) = \text{rute}(s, u_s) + \text{rute}(u_d, d)$ .

### 3.6.2 Modified Bi-directional A\*

Berbeda dengan algoritma *Bi-directional A\**, algoritma *Modified Bi-directional A\** menggunakan fungsi heuristik sebagai berikut[IKE94]:

$$F = l(s, u) + l(u, v) + \frac{1}{2}[h_s(v) - h_s(u)] + \frac{1}{2}[h_t(u) - h_t(v)]$$

Secara detail, algoritmanya sama dengan algoritma *bi-directional A\** dengan perubahan pada langkah (2) dan (4):

- (2) Cari node  $u_s$  yang memiliki nilai  $p_s(v_s)$  terkecil di  $S$  dan tambahkan  $u_s$  ke  $S$ . Jika  $u_s$  di  $D$ , lakukan langkah 7.
- (4) Cari node  $u_d$  yang memiliki nilai  $p_d(v_d)$  terkecil di  $D$  dan tambahkan  $u_d$  ke  $D$ . Jika  $u_d$  di  $S$ , lakukan langkah 7.

Dan langkah (3) dan (5):

- (3) Untuk semua *node*  $v_s$  dimana sisi  $(u_s, v_s)$  di dalam  $E$ , jika  $p_s(u_s) + l'(u_s, v_s)$  lebih kecil dari  $p_s(v_s)$ : ganti rute  $(s, v_s)$  dengan rute  $(s, u_s) +$  sisi  $(u_s, v_s)$  dan ganti nilai  $p_s(v_s) = p_s(u_s) + l'(u_s, v_s)$  dan hapus  $v_s$  dari  $S$  jika  $v_s$  di dalam  $S$ .
- (4) Untuk semua *node*  $v_d$  dimana sisi  $(v_d, u_d)$  di dalam  $E$ , jika  $l'(v_d, u_d) + p_d(u_d)$  lebih kecil dari  $p_d(v_d)$ : ganti rute  $(v_d, d)$  dengan sisi  $(v_d, u_d) +$  rute  $(u_d, d)$  dan ganti nilai  $p_d(v_d) = l'(v_d, u_d) + p_d(u_d)$  dan hapus  $v_d$  dari  $D$  jika  $v_d$  di dalam  $D$ .

## 3.7 Problem Reduction

### 3.7.1 AND-OR Graphs

Struktur jenis lain, And-Or Graph (atau tree), digunakan untuk memperlihatkan solusi dari permasalahan yang dapat diselesaikan dengan mendekomposisi permasalahan tersebut menjadi sekumpulan masalah yang lebih kecil, dimana semuanya harus dapat diselesaikan. Pemecahan atau pereduksian ini membangkitkan *arcs* (busur) yang kita kenal dengan **AND arcs**. Satu busur AND akan menghasilkan beberapa nomor dari simpul setelahnya. Semuanya harus diselesaikan supaya pancaran menghasilkan solusi. Hanya saja pada OR graph, banyak pancaran akan muncul dari beberapa *node*, mengindikasikan beberapa jalan yang permasalahan bisa diselesaikan.

Untuk menemukan solusi dari sebuah AND-OR graph, kita memerlukan sebuah algoritma serupa dengan BFS tapi dengan kemampuan untuk menangani pancaran AND dengan tepat. Algoritma ini harus menemukan bagian dari simpul awal dari graph untuk menghimpun simpul-simpul yang menggambarkan *state* solusi. Perlu diketahui bahwa ini mungkin diperlukan untuk mendapatkan lebih dari satu solusi sejak setiap pancaran AND.

### 3.7.2 Constraint Satisfaction

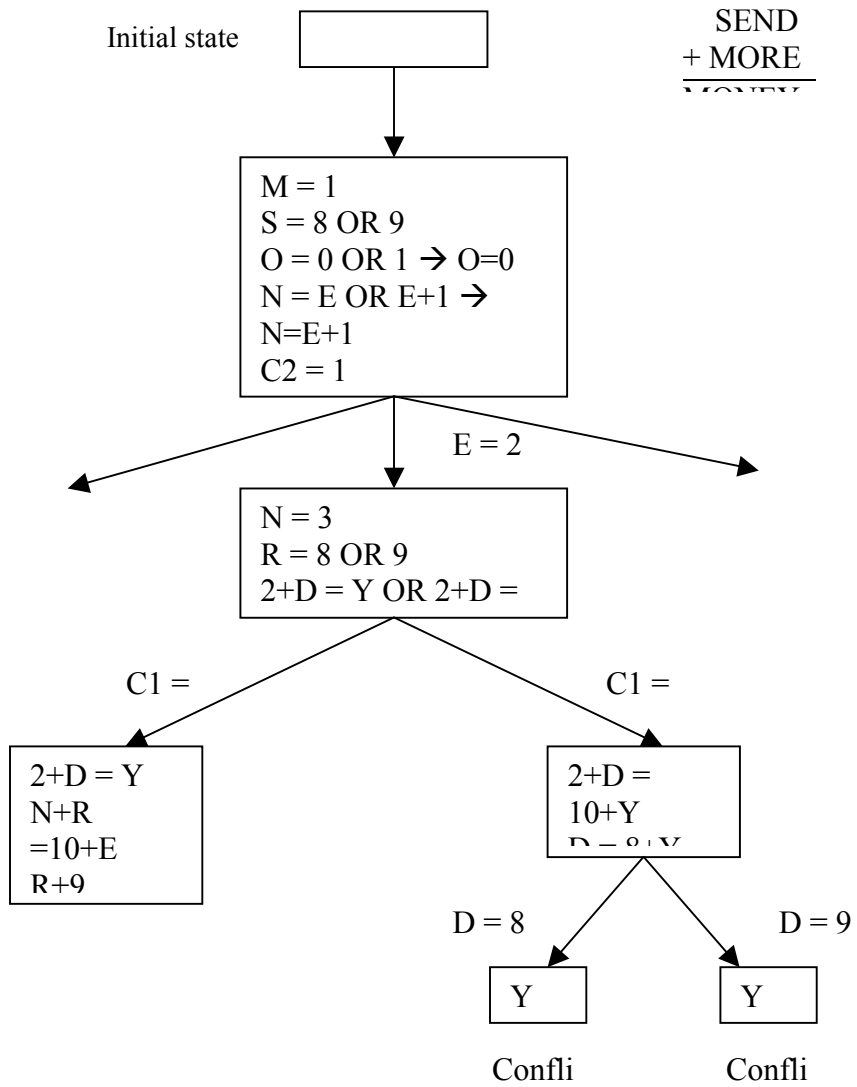
Adalah sebuah prosedur pencarian yang dioperasikan pada sekumpulan ruang batas. *Initial state* terdiri dari batas-batas yang benar-benar diberikan dalam deskripsi masalah.

#### Algorithm: Constraint Satisfaction

1. Propagate available constraints. To do this, first set *OPEN* to the set of all object that must have values assigned to them in a complete solution. Then do until an inconsistency is detected or until *OPEN* is empty:
  - a. Select an object *OB* from *OPEN*. Strengthen as much as possible the set of constraints that apply to *OB*.
  - b. If this set is different from the set that was assigned the last time *OB* examined or if this is the first time *OB* has been examined, then add to *OPEN* all object that share any constraints with *OB*.
  - c. Remove *OB* from *OPEN*.
2. If the union of the constraints discovered above defines a solution, then quit and report the solution.

3. If the union of the constraints discovered above defines a contradiction, then return failure.
4. If neither of the above occurs, then it is necessary to make a guess at something in order to proceed. To do this, loop until a solution is found or all possible solutions have been eliminated:
  - a. Select an object whose value is not yet determined and select a way of strengthening the constraints on that *object*.
  - b. Recursively invoke constraint satisfaction with the current set of constraints augmented by the strengthening constraint just selected.





### 3.7.3 Means-Ends Analysis (MEA)

Sejauh ini kita sudah melihat sekumpulan strategi pencarian *forward* dan *backward*, tetapi untuk sebuah problem yang diberikan, secara langsung atau harus dipilih. Bagaimanapun, gabungan dari dua arah tersebut adalah lebih tepat.

Pengertian dari *Means-End Analysis* adalah pusat pemrosesan disekitar pendeteksian perbedaan antara *current state* dan *goal state*. Sekalipun perbedaan diisolasi, sebuah operator yang dapat mengurangi perbedaan harus ditemukan. Tetapi kemungkinan operator tidak dapat diterapkan pada *current state*.

#### **Algoritma Means-Ends Analysis(*CURRENT*, *GOAL*):**

1. Compare *CURRENT* dengan *GOAL*. If there are no difference between them then return.
2. Otherwise, select the most important difference and reduce it by doing the following until success or failure is signaled:
  - a. Select an as yet untried operator *O* that is applicable to the current difference. If there are no such operators, then signal failure.
  - b. Attempt to apply *O* to *CURRENT*. Generate descriptions of two states: *O-START*, a state in which *O*'s preconditions are satisfied and *O-RESULT*, the state that would result if *O* were applied in *O-START*.
  - c. **If**  
(*FIRST-START*  $\leftarrow$  MEA(*CURRENT*, *O-START*))  
**and**  
(*LAST-PART*  $\leftarrow$  MEA(*O-RESULT*, *GOAL*))  
are successful, then signal success and return the result of concatenating *FIRST-START*, *O*, and *LAST-PART*.

## Bab 4

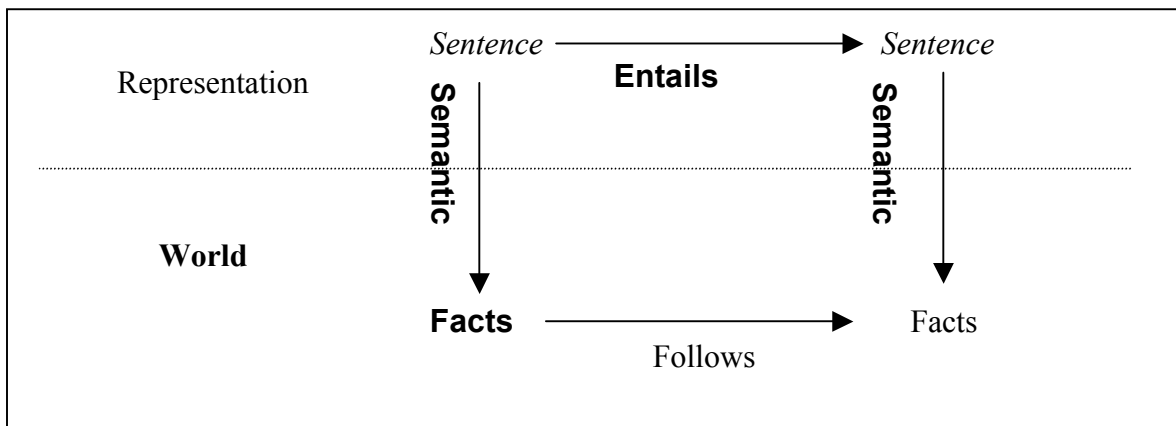
# Pengetahuan dan Penalaran

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).

```
function KB-Agent(percept) returns an action
static: KB, a Knowledge Base
        t, a counter, initially 0, indicating time

TELL(KB, MAKE-PERCEPT-SENTENCE(percept,t))
action ← ASK(KB, MAKE-ACTION-QUERY(t))
TELL(KB, MAKE-ACTION-SENTENCE(action,t))
t ← t + 1
```

**Gambar 4.1** A generic knowledge-based agents.



**Gambar 4.2** Hubungan antara *sentence* dan *facts* 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

## 4.1 Propositional Logic (Propositional Calculus)

*Sentence*  $\rightarrow$  *AtomicSentence* | *ComplexSentence*

*AtomicSentence*  $\rightarrow$  *True* | *False*

| *P* | *Q* | *R*

*ComplexSentence*  $\rightarrow$  (*Sentence*)

| *Sentence* *Connective* *Sentence*

|  $\neg$  *Sentence*

**Gambar 4.3** A BNF (Backus-Naur Form) Grammar of sentences in propositional logic

**1. Modus Ponens atau Implication-Elimination:**

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

**2. And-Elimination:**

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

**3. And-Introduction:**

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

**4. Or-Introduction:**

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

**5. Double-Negation-Elimination:**

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

**6. Unit Resolution:**

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

**7. Resolution:**

$$\alpha \vee \beta, \neg \alpha$$

$$\neg \beta \vee \gamma$$

**Gambar 4.4** Aturan inferensi dalam Logika Proposisi.

## Studi Kasus: The Wumpus World Environment

### 1. *Percept*

- Percept : [*stench, breeze, glitter, bump, scream*]
- [*stench, breeze, None, None, None*] = ada stench dan breeze, tetapi tidak ada glitter, bump, maupun scream.

### 2. *Action*

- **Move:** Turn Left 900, Turn Right 900, Straight.
- **Grab:** mengambil / merebut objek yang berada di kotak yang sama dimana agent berada.
- **Shoot:** memanah dengan arah lurus sesuai dengan arah agent menghadap.
- **Climb:** memanjat keluar dari gua.
- Agent akan mati jika memasuki kotak yang terdapat Wumpus atau Pit. Tetapi akan aman jika memasuki kotak yang di dalamnya terdapat Wumpus yang telah mati.

### 3. *Goal*

Menemukan emas dan membawanya kembali ke kotak start (1,1) secepat mungkin dengan jumlah *action* yang semimum mungkin, tanpa terbunuh. Sebagai hadiah, 1.000 point diberikan kepada *agent* jika berhasil keluar gua dengan membawa emas. Tetapi point -1 untuk setiap *action* yang dilakukan. Dan -10.000 jika *agent* terbunuh.

## An Agent for The Wumpus World

4	Stench		Breeze	<b>PIT</b>
3	<b>Wumpus</b> Stench	Breeze, Stench, Gold (glitter)	<b>PIT</b>	Breeze
2	Stench		Breeze	
1	<b>START</b> <b>Agent</b>	Breeze	<b>PIT</b>	Breeze
	1	2	3	4

## Knowledge Base (KB)

Pada setiap langkah, *percept* yang diterima *agent* dikonversi ke dalam *sentence* dan dimasukkan ke dalam KB bersama dengan beberapa *valid sentence* yang di-*entail* oleh *sentence* yang telah diterima sebelumnya. Misalkan simbol  $S_{1,2}$  menyatakan “Ada Stench di  $[1,2]$ ”. Setelah *agent* melakukan 3 *action*, maka di dalam KB terdapat:

$\neg S_{1,1}$	$\neg B_{1,1}$
$\neg S_{2,1}$	$B_{2,1}$
$S_{1,2}$	$\neg B_{1,2}$

**Background Knowledge** (beberapa pengetahuan tentang environment):

$$\begin{aligned} R_1 : \neg S_{1,1} &\Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,1} \\ R_2 : \neg S_{2,1} &\Rightarrow \neg W_{1,1} \wedge \neg W_{2,1} \wedge \neg W_{2,2} \wedge \neg W_{3,1} \\ R_3 : \neg S_{1,2} &\Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,2} \wedge \neg W_{1,3} \\ R_4 : S_{1,2} &\Rightarrow W_{1,3} \vee W_{1,2} \vee W_{2,2} \vee W_{1,1} \end{aligned}$$

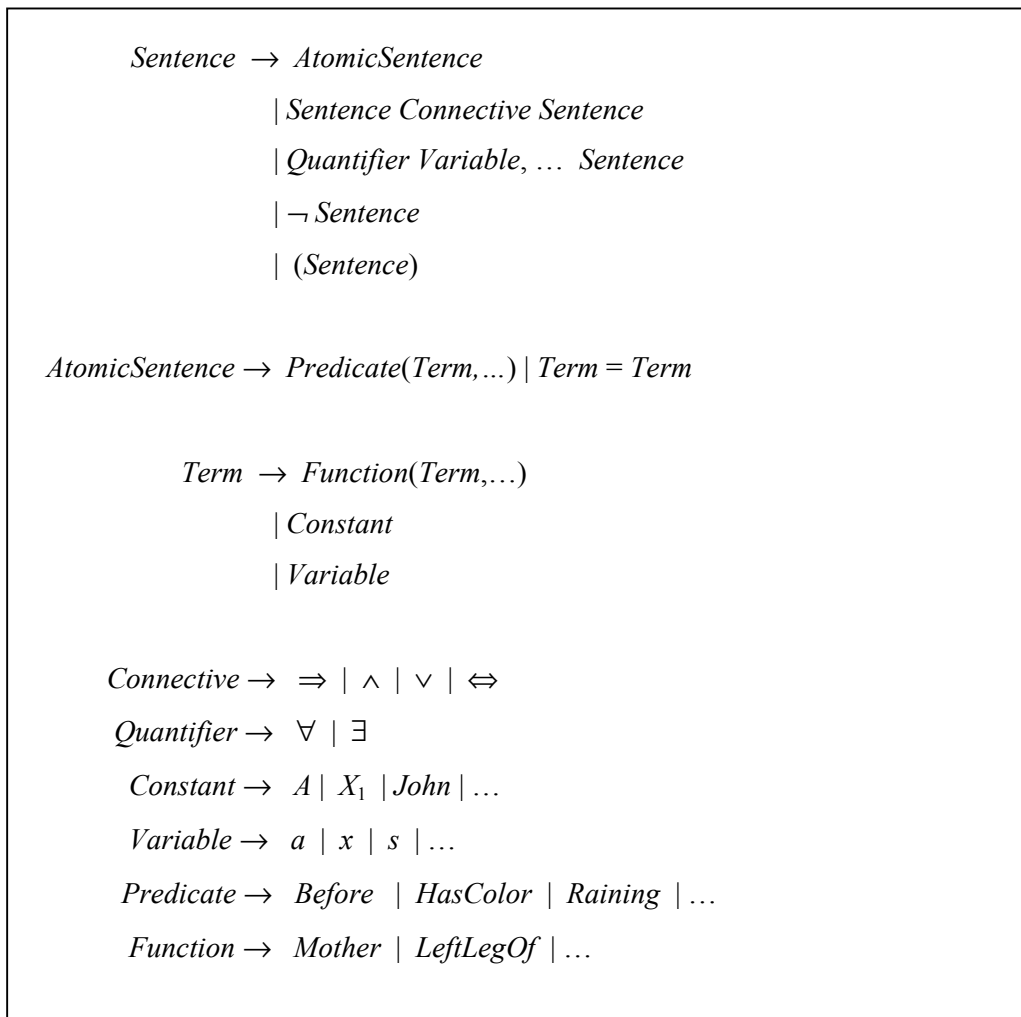
## Proses Inferensi Untuk Mencari Lokasi Wumpus

1. Lakukan **Modus Ponens** untuk  $\neg S_{1,1}$  dan *sentence*  $R_1$ , sehingga didapat:  
 $\neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,1}$
2. Lakukan **And-Elimination** terhadap hasil di atas, sehingga diperoleh tiga *sentence* :  
 $\neg W_{1,1} \quad \neg W_{1,2} \quad \neg W_{2,1}$
3. Lakukan **Modus Ponens** untuk  $\neg S_{2,1}$  dan *sentence*  $R_2$ . Kemudian **And-Elimination** terhadap hasil tersebut, sehingga didapat tiga *sentence* :  
 $\neg W_{1,1} \quad \neg W_{2,1} \quad \neg W_{2,2} \quad \neg W_{3,1}$
4. Lakukan **Modus Ponens** untuk  $S_{1,2}$  dan *sentence*  $R_4$ . sehingga didapat :  
 $W_{1,3} \vee W_{1,2} \vee W_{2,2} \vee W_{1,1}$
5. Lakukan **Unit Resolution**, dimana  $\alpha$  adalah  $W_{1,3} \vee W_{1,2} \vee W_{2,2}$ , dan  $\beta$  adalah  $W_{1,1}$  (dimana  $\neg W_{1,1}$  diturunkan dari langkah 2), sehingga didapat :  
 $W_{1,3} \vee W_{1,2} \vee W_{2,2}$
6. Lakukan **Unit Resolution**, dimana  $\alpha$  adalah  $W_{1,3} \vee W_{1,2}$ , dan  $\beta$  adalah  $W_{2,2}$  (dimana  $\neg W_{2,2}$  diturunkan dari langkah 3), sehingga didapat :  
 $W_{1,3} \vee W_{1,2}$
7. Lakukan **Unit Resolution**, dimana  $\alpha$  adalah  $W_{1,3}$  dan  $\beta$  adalah  $W_{1,2}$  (dimana  $\neg W_{1,2}$  diturunkan dari langkah 2), sehingga memberikan jawaban yang kita inginkan, yaitu bahwa Wumpus berada di kotak  $[1,3]$ :  
 $W_{1,3}$

## 4.2 First-Order Logic (Predicat Logic / Predicat Calculus)

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

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



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



### 4.2.1 Inferensi Dalam Kalkulus Predikat

#### 1. *Inference Rules Involving Quantifiers*

**SUBST( $\theta, \alpha$ ):** untuk menotasikan hasil dari pengaplikasian operasi substitusi  $\theta$  terhadap sentence  $\alpha$ .

$$\text{SUBST}(\{x/\text{Sam}, y/\text{Pam}\}, \text{Likes}(x,y)) = \text{Likes}(\text{Sam}, \text{Pam}).$$

**2. Universal Elimination:** untuk setiap sentence  $\alpha$ , variabel  $v$ , dan *ground term* (term yang tidak berisi variabel)  $g$  :

$$\frac{\forall v \ \alpha}{\text{SUBST}(\{v/g\}, \alpha)}$$

Dari  $\forall x \text{ Likes}(x, \text{IceCream})$ , dapat digunakan substitusi  $\{x/\text{Ben}\}$  dan melakukan inferensi bahwa  $\text{Likes}(\text{Ben}, \text{IceCream})$

**3. Existential Elimination:** untuk setiap sentence  $\alpha$ , variabel  $v$ , dan simbol konstanta  $k$  yang tidak tampak dimanapun di dalam basis pengetahuan:

$$\frac{\exists v \ \alpha}{\text{SUBST}(\{v/k\}, \alpha)}$$

Dari  $\exists x \text{ Kill}(x, \text{Victim})$ , kita dapat menyimpulkan  $\text{Kill}\{\text{Murderer}, \text{Victim}\}$ , selama *Murderer* tidak tampak dimanapun di dalam basis pengetahuan.

**4. Existential Introduction:** untuk setiap sentence  $\alpha$ , variabel  $v$  yang tidak terjadi pada  $\alpha$ , dan ground term  $g$  yang terjadi pada  $\alpha$ :

$$\frac{\alpha}{\exists v \text{ SUBST}(\{g/v\}, \alpha)}$$

Dari  $\text{Likes}(\text{Jerry}, \text{IceCream})$  kita dapat menyimpulkan  $\exists x \text{ Likes}(x, \text{IceCream})$ .

Contoh bagaimana menggunakan aturan inferensi dalam pembuktian.

*The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.*

Buktikan bahwa *West* adalah seorang penjahat atau *criminal*. Pertama representasikan fakta dalam kalkulus predikat, kemudian tunjukan bukti sebagai urutan aplikasi *inference rules*.

- $\forall x,y,z \text{ American}(x) \wedge \text{Weapon}(y) \wedge \text{Nation}(z) \wedge \text{Hostile}(z) \wedge \text{Sells}(x,z,y) \Rightarrow \text{Criminal}(x)$  (1)
- $\exists x \text{ Owns}(\text{Nono},x) \wedge \text{Missile}(x)$  (2)
- $\forall x \text{ Owns}(\text{Nono},x) \wedge \text{Missile}(x) \Rightarrow \text{Sells}(\text{West},\text{Nono},x)$  (3)
- $\forall x \text{ Missile}(x) \Rightarrow \text{Weapon}(x)$  (4)
- $\forall x \text{ Enemy}(x,\text{America}) \Rightarrow \text{Hostile}(x)$  (5)
- $\text{American}(\text{West})$  (6)
- $\text{Nation}(\text{Nono})$  (7)
- $\text{Enemy}(\text{Nono},\text{America})$  (8)
- $\text{Nation}(\text{America})$  (9)

### Proses pembuktian dengan pengaplikasian aturan inferensi kalkulus predikat

- Dari (2) dan Existential Elimination:  
 $\text{Owns}(\text{Nono},\text{M1}) \wedge \text{Missile}(\text{M1})$  (10)
- Dari (10) dan And-Elimination:  
 $\text{Owns}(\text{Nono},\text{M1})$  (11)  
 $\text{Missile}(\text{M1})$  (12)
- Dari (4) dan Universal Elimination:  
 $\text{Missile}(\text{M1}) \Rightarrow \text{Weapon}(\text{M1})$  (13)
- Dari (12), (13), dan Modus Ponens:  
 $\text{Weapon}(\text{M1})$  (14)
- Dari (3) dan Universal Elimination:  
 $\text{Owns}(\text{Nono},\text{M1}) \wedge \text{Missile}(\text{M1}) \Rightarrow \text{Sells}(\text{West},\text{Nono},\text{M1})$  (15)
- Dari (15), (10), dan Modus Ponens:  
 $\text{Sells}(\text{West},\text{Nono},\text{M1})$  (16)
- Dari (1) dan Universal Elimination (tiga kali):  
 $\text{American}(\text{West}) \wedge \text{Weapon}(\text{M1}) \wedge \text{Nation}(\text{Nono}) \wedge \text{Hostile}(\text{Nono})$

$$\wedge \text{Sells}(\text{West}, \text{Nono}, \text{M1}) \Rightarrow \text{Criminal}(\text{West}) \quad (17)$$

- Dari (5) dan Universal Elimination:  
 $\text{Enemy}(\text{Nono}, \text{America}) \Rightarrow \text{Hostile}(\text{Nono}) \quad (18)$

- Dari (8), (18), dan Modus Ponens:  
 $\text{Hostile}(\text{Nono}) \quad (19)$

- Dari (6), (7), (14), (16), (19), dan And-Introduction:  
 $\text{American}(\text{West}) \wedge \text{Weapon}(\text{M1}) \wedge \text{Nation}(\text{Nono})$   
 $\wedge \text{Hostile}(\text{Nono}) \wedge \text{Sells}(\text{West}, \text{Nono}, \text{M1}) \quad (20)$

- Dari (17), (20) dan Modus Ponens:  
 $\text{Criminal}(\text{West}) \quad (21)$

Proses pembuktian di atas dapat dilakukan menggunakan teknik **searching** dimana:

**Initial State** : Knowledge Base (sentences 1-9)

**Operators** : Applicable inference rules

**Goal test** : Knowledge Base yang berisi  $\text{Criminal}(\text{West})$

Tetapi diperlukan program yang **sangat pintar** untuk menemukan bukti tanpa menelusuri jalur yang salah.

#### 4.2.2 Generalized Modus Ponens

Generalisasi Modus Ponens memerlukan:

- And-Introduction
- Universal Elimination
- Modus Ponens

Contoh:

$\text{Missile}(\text{M1})$

$\text{Owns}(\text{Nono}, \text{M1})$

$\forall x \text{ Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Selles}(\text{West}, \text{Nono}, x)$

dan meng-*infer* dalam satu langkah suatu *sentence* baru, yaitu:

$\text{Sells}(\text{West}, \text{Nono}, \text{M1})$

dari tiga sentence di atas, anggaplah selain Nono, setiap negara mempunyai M1:

$\forall y \text{ Owns}(y, \text{M1})$

Untuk atomic sentences  $p_i$ ,  $p_i'$ , dan  $q$ , dimana terdapat substitution  $\theta$  sedemikian hingga  $\text{SUBST}(\theta, p_i') = \text{SUBST}(\theta, p_i)$ , untuk semua  $i$  :

$$\frac{p_1', p_2', \dots, p_n', (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

Terdapat  $n+1$  premise untuk aturan ini:  $n$  atomic sentences  $p_i'$  dan satu implikasi. Terdapat satu *conclusion* (kesimpulan): hasil pengaplikasian substitusi terhadap consequent  $q$ . Sebagai contoh dengan West dan missile:

$p_1'$ adalah Missile( $MI$ )	$p_1$ adalah Missile( $x$ )
$p_2'$ adalah Owns( $y, MI$ )	$p_2$ adalah Owns( $Nono, x$ )
$\theta$ adalah $\{x/MI, y/Nono\}$	$q$ adalah Sells( $West, Nono, x$ )
SUBST( $\theta, q$ ) adalah Sells( $West, Nono, MI$ )	

### Unification (Unifikasi)

Secara formal *unification* dituliskan:

$$\text{UNIFY}(p, q) = \theta \text{ dimana } \text{SUBST}(\theta, p) = \text{SUBST}(\theta, q)$$

$\theta$  disebut **unifier** dari dua sentences tersebut. Kita akan mengilustrasikan *unification* dalam konteks dari suatu contoh. Misalkan kita punya aturan

$\text{Knows}(\text{John}, x) \Rightarrow \text{Hates}(\text{John}, x)$  : yang berarti (“John hates everyone he knows”)

Misalkan knowledge base kita berisi sentences sebagai berikut :

$\text{Knows}(\text{John}, \text{Jane})$   
 $\text{Knows}(y, \text{Leonid})$   
 $\text{Knows}(y, \text{mother}(y))$   
 $\text{Knows}(x, \text{Elizabeth})$

Pelaksanaan unifikasi *premise* atau *antecedent* dari aturan terhadap setiap *sentence* dalam knowledge base akan memberikan :

$\text{UNIFY}(\text{Knows}(\text{John}, x), \text{Knows}(\text{John}, \text{Jane})) = \{x/\text{Jane}\}$   
 $\text{UNIFY}(\text{Knows}(\text{John}, x), \text{Knows}(y, \text{Leonid})) = \{x/\text{Leonid}, y/\text{John}\}$   
 $\text{UNIFY}(\text{Knows}(\text{John}, x), \text{Knows}(y, \text{mother}(y))) = \{y/\text{John}, x/\text{Mother}(\text{John})\}$   
 $\text{UNIFY}(\text{Knows}(\text{John}, x), \text{Knows}(x, \text{Elizabeth})) = \text{fail}$

Unifikasi terakhir *fail* (gagal) karena  $x$  tidak dapat bernilai *John* dan bernilai *Elizabeth* pada waktu yang sama. Tetapi secara intuitif, dari fakta-fakta bahwa John membenci setiap orang yang dia kenal dan setiap orang mengenal Elizabeth, kita akan dapat menginferensi bahwa John membenci Elizabeth. Tidak masalah jika sentence yang terdapat dalam knowledge base tersebut adalah  $\text{Knows}(x, \text{Elizabeth})$  atau  $\text{Knows}(y, \text{Elizabeth})$ .

$$\text{UNIFY}(\text{Knows}(\text{John}, x_1), \text{Knows}(x_2, \text{Elizabeth})) = \{x_1/\text{Elizabeth}, x_2/\text{John}\}$$

Tetapi jika terdapat suatu substitusi seperti di atas, maka terdapat tak berhingga substitusi yang lain, seperti:

$\text{UNIFY}(\text{Knows}(\text{John}, x), \text{Knows}(y, z)) = \{y/\text{John}, x/z\}$   
or  $\{y/\text{John}, x/z, w/\text{Freda}\}$   
or  $\{y/\text{John}, x/\text{John}, z/\text{John}\} \dots$

Kita tegaskan bahwa UNIFY mengembalikan **Most General Unifer** (MGU), yang merupakan substitusi yang membuat komitmen paling sedikit memuat variable.

### 4.2.3 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 ( $p_1 \wedge \dots \wedge p_n \Rightarrow q$ ) in KB such that for some i, UNIFY( $p_i, p$ ) =  $\theta$  succeeds do  
    FIND-AND-INFER(KB, [ $p_1, \dots, p_{i-1}, p_{i+1}, \dots, p_n$ ], q,  $\theta$ )  
  end
```

```
procedure FIND-AND-INFER(KB, premises, conclusion,  $\theta$ )  
  if premises = [ ] then  
    FORWARD-CHAIN(KB, SUBST( $\theta$ , conclusion))  
  else for each  $p'$  in KB such that UNIFY( $p'$ , SUBST( $\theta$ , FIRST(premises))) =  $\theta_2$  do  
    FIND-AND-INFER(KB, REST(premises), conclusion, COMPOSE( $\theta$ ,  $\theta_2$ ))
```

**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 simpanlah *conclusion*. Jika premise dapat dicocokkan dengan beberapa cara

```
function BACK-CHAIN(KB,q) returns a set of substitutions
```

```
  BACK-CHAIN-LIST(KB, [q], {})
```

```
function BACK-CHAIN-LIST(KB, qlist,  $\theta$ ) returns a set of substitutions
```

```
  inputs: KB, a knowledge base
```

```
          qlist, a list of conjuncts forming a query ( $\theta$  already applied)
```

```
           $\theta$ , the current substitution
```

```
  local variables: answers, a set of substitutions, initially empty
```

```
  if qlist is empty then return { $\theta$ }
```

```
   $q \leftarrow$  FIRST(qlist)
```

```
    for each  $q_i'$  in KB such that  $\theta_i \leftarrow$  UNIFY( $q'$ ,  $q_i'$ ) succeeds do
```

```
      Add COMPOSE( $\theta$ ,  $\theta_i$ ) to answers
```

```
    end
```

```
    for each ( $p_1 \wedge \dots \wedge p_n \Rightarrow q_i'$ ) in KB such that  $\theta_i \leftarrow$  UNIFY( $q$ ,  $q_i'$ ) succeeds do
```

```
      answers  $\leftarrow$  BACK-CHAIN-LIST(KB, SUBST( $\theta_i$ , [ $p_1, \dots, p_n$ ], q),  $\theta$ ),
```

**Gambar 4.7 Algoritma Inferensi Backward-Chaining.**

#### 4.2.4 Aturan Inferensi Resolusi (The Resolution Inference Rule)

Bentuk sederhana dari aturan inferensi resolusi untuk logika proposisi adalah:

$$\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma} \quad \text{atau ekuivalen dengan} \quad \frac{\neg\alpha \Rightarrow \beta, \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

- **Generalized Resolution (Disjunctions):** Untuk literal  $p_i$  dan  $q_i$ , dimana  $\text{UNIFY}(p_j, \neg q_k) = \theta$ :

$$\frac{p_1 \vee \dots p_j \dots \vee p_m \quad q_1 \vee \dots q_k \dots \vee q_n}{\text{SUBST}(\theta, (p_1 \vee \dots p_{j-1} \vee p_{j+1} \vee p_m \vee q_1 \vee \dots q_{k-1} \vee q_{k+1} \dots \vee q_n))}$$

- **Generalized Resolution (Implications):** Untuk atom  $p_i, q_i, r_i, s_i$  dimana  $\text{UNIFY}(p_j, q_k) = \theta$ :

$$\frac{p_1 \wedge \dots p_j \dots \wedge p_{n1} \Rightarrow r_1 \vee \dots \vee r_{n2} \quad s_1 \wedge \dots \wedge s_{n3} \Rightarrow q_1 \vee \dots q_k \dots \vee q_{n4}}{\text{SUBST}(\theta, (p_1 \wedge \dots p_{j-1} \wedge p_{j+1} \vee p_{n1} \wedge s_1 \wedge \dots s_{n3} \Rightarrow r_1 \vee \dots r_{n2} \vee q_1 \vee \dots q_{k-1} \vee q_{k+1} \dots \vee q_{n4}))}$$

Misalkan kita mempunyai Knowledge Base sebagai berikut:

$$\begin{aligned} \forall x \quad P(x) &\Rightarrow Q(x) \\ \forall x \quad \neg P(x) &\Rightarrow R(x) \\ \forall x \quad Q(x) &\Rightarrow S(x) \\ \forall x \quad R(x) &\Rightarrow S(x) \end{aligned}$$

Bagaimana menyimpulkan  $S(A)$  ?

Canonical forms (bentuk resmi) untuk Resolusi:

- **Conjunctive Normal Form (CNF):** secara implisit adalah disjunction (first rule).
- **Implicative Normal Form (INF):** berbeda dengan Horn form (karena di sebelah kanan implikasi bisa berupa disjunction).

Conjunctive Normal Form (CNF)	Implicative Normal Form (INF)
$\neg P(w) \vee Q(w)$	$P(w) \Rightarrow Q(w)$
$P(x) \vee R(x)$	$\text{True} \Rightarrow P(x) \vee R(x)$
$\neg Q(y) \vee S(y)$	$Q(y) \Rightarrow S(y)$
$\neg R(z) \vee S(z)$	$R(z) \Rightarrow S(z)$

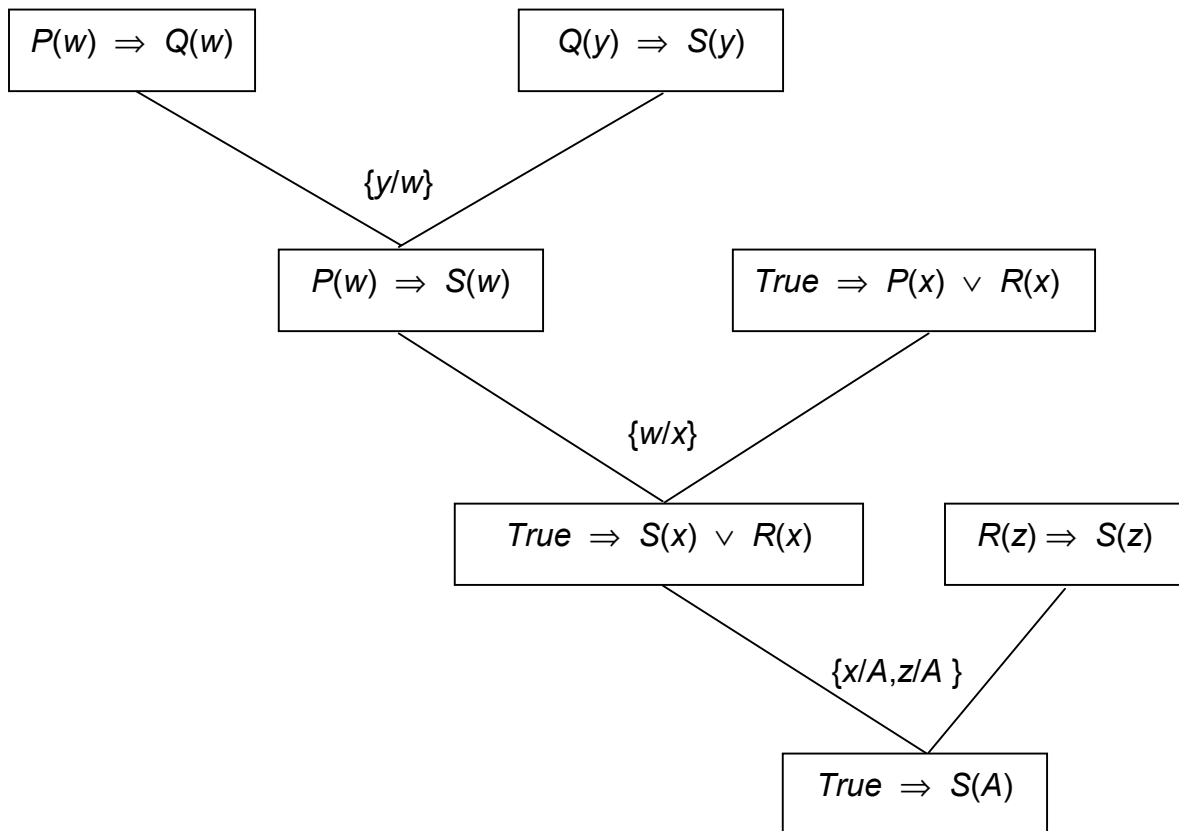
Dalam Resolusi lebih sering digunakan INF, karena lebih alami.

Resolusi adalah generalisasi dari modus ponens.

Kita dapat melihat bahwa modus ponens hanyalah *special case* dari Resolusi:

$$\frac{\alpha, \alpha \Rightarrow \beta}{\beta} \quad \text{adalah ekuivalen dengan} \quad \frac{True \Rightarrow \alpha, \alpha \Rightarrow \beta}{True \Rightarrow \beta}$$

Dari knowledge base di atas, perhatikan tiga langkah resolusi di bawah ini untuk menyimpulkan  $S(A)$ . dua sentence di atas merupakan *premise*, dan satu sentence di bawahnya merupakan *conclusion* atau *resolvent*.



Chaining dengan resolusi lebih kuat dibandingkan dengan Modus Ponens, tetapi masih belum lengkap. Resolusi tidak bisa membuktikan  $\neg P \vee P$  dari Knowledge Base yang masih kosong. Tidak ada sentence apapun bagi resolusi untuk diaplikasikan.

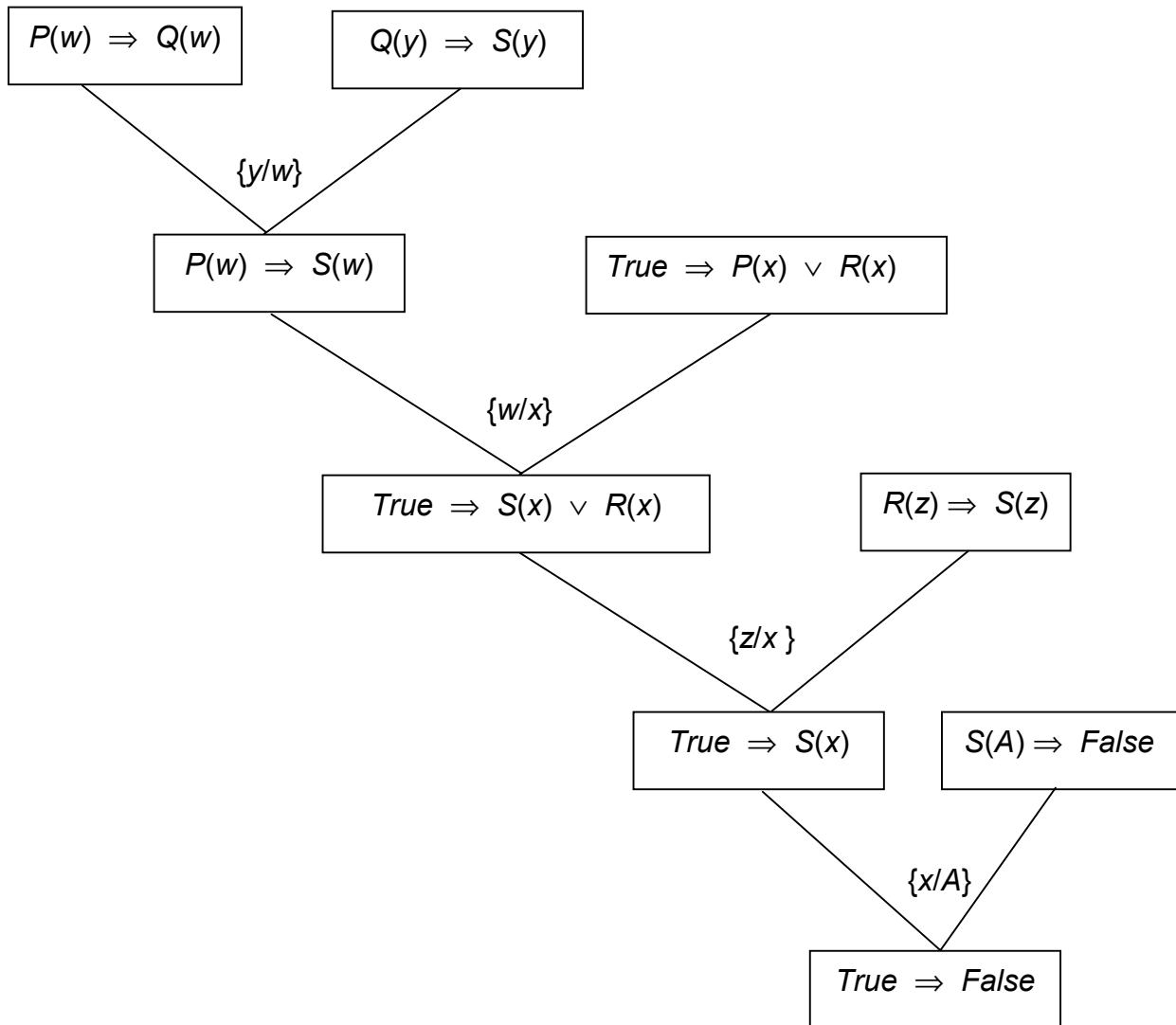
**Refutation (proof by contradiction and reductio ad absurdum):** Prosedur inferensi yang lengkap dengan menggunakan resolusi. Identya adalah bahwa untuk membuktikan  $P$ , asumsikan  $P$  adalah salah (yaitu dengan menambahkan  $\neg P$  ke dalam KB) dan membuktikan suatu kontradiksi.

$$(KB \wedge \neg P \Rightarrow False) \Leftrightarrow (KB \Rightarrow P)$$



Dari knowledge base di atas, perhatikan langkah-langkah **Refutation** di bawah ini untuk menyimpulkan  $S(A)$ . Kita tambahkan ke dalam KB suatu sentnse  $\neg S(A)$ , yang dalam bentuk INF adalah  $S(A) \Rightarrow False$ . Kemudian kita aplikasikan resolusi sampai menemukan suatu kontradiksi, yang dalam bentuk INF adalah  $True \Rightarrow False$ .

Pada refutation terdapat satu langkah tambahan, tetapi hal ini merupakan harga yang kecil untuk membayar keamanan dari metode pembuktian yang lengkap.



#### 4.2.5 Konversi ke Bentuk Normal (CNF dan INF)

1. **Eliminate Implication:** ganti semua implication dengan disjunction yang sesuai.  
 $p \Rightarrow q$  menjadi  $\neg p \vee q$
2. **Move  $\neg$  inwards:** negasi hanya dibolehkan pada atom-atom dalam *Conjunctive Normal Form* (CNF), dan tidak diijinkan pada semua *Implicative Normal Form*. Hilangkan negasi menggunakan hukum de Morgan.

$\neg(p \vee q)$	menjadi	$\neg p \wedge \neg q$
$\neg(p \wedge q)$	menjadi	$\neg p \vee \neg q$
$\neg \forall x p$	menjadi	$\exists x \neg p$
$\neg \exists x p$	menjadi	$\forall x \neg p$
$\neg \neg p$	menjadi	$p$
3. **Standardize variables:** untuk menghindari kebingungan.  
 $(\forall x P(x)) \vee (\exists x Q(x))$  menjadi  $(\forall x P(x)) \vee (\exists y Q(y))$
4. **Move quantifiers left:** pindahkan kuantifier ke kiri tanpa mengubah arti sentence.  
 $p \vee \forall x q$  menjadi  $\forall x p \vee q$  (karena  $p$  dijamin tidak berisi  $x$ )
5. **Skolemize:** skolemization adalah proses penghilangan *existential quantifier*.  
 $\forall x \text{ Person}(x) \Rightarrow \exists y \text{ Heart}(y) \wedge \text{Has}(x,y)$   
Jika hanya mengganti  $y$  dengan suatu konstanta,  $H$ , kita akan dapatkan:  
 $\forall x \text{ Person}(x) \Rightarrow \text{Heart}(H) \wedge \text{Has}(x,H)$   
yang berarti setiap orang mempunyai hati yang sama. Untuk itu diperlukan fungsi yang memetakan  $\text{Person}$  ke  $\text{Heart}$ :  
 $\forall x \text{ Person}(x) \Rightarrow \text{Heart}(F(x)) \wedge \text{Has}(x,F(x))$   
 $F$  (skolem function) adalah nama fungsi yang tidak terdapat di Knowledge Base.
6. **Distribute  $\wedge$  over  $\vee$ :**  
 $(a \wedge b) \vee c$  menjadi  $(a \vee c) \wedge (b \vee c)$
7. **Flatten nested conjunctions and disjunctions:**

$(a \vee b) \vee c$	menjadi	$(a \vee b \vee c)$
$(a \wedge b) \wedge c$	menjadi	$(a \wedge b \wedge c)$
8. **Convert disjunctions to implications:** untuk setiap conjunction, kumpulkan negative literals ke dalam satu list, dan positive literals ke dalam list yang lain, dan bangun suatu implikasi dari keduanya:  
 $(\neg a \vee \neg b \vee c \vee d)$  menjadi  $(a \wedge b \Rightarrow c \vee d)$

### 4.3 Declarative structure used in Knowledge Representation

1. First Order Logic / Predicate Logic / Predicate Calculus
  2. Production Rules
  3. Non-monotonic Systems
  4. Statistical Reasoning Systems
  5. Semantic Nets
  6. Frames System
  7. Conceptual Dependency
  8. Scripts
  9. CYC
- Weak Slot-and-Filler Structures**
- Strong Slot-and-Filler Structures**

#### 4.3.1 Non-monotonic Systems: ‘akal sehat’ manusia

Penambahan *statement* baru mungkin akan menyebabkan jumlah *statement* TRUE berkurang (menjadi FALSE).

**Contoh:** pengecualian

$$\begin{array}{l} p \text{ unless } q \Rightarrow r \\ r \Rightarrow s \end{array} \quad \{ \text{arti: } p \Rightarrow r \text{ jika } q \text{ FALSE} \}$$

Jika  $q$  FALSE maka dapat dilakukan aturan:  $p \Rightarrow r$ , lalu  $r \Rightarrow s$

Tapi begitu ada fakta baru  $q$  TRUE, maka  $r$  dan  $s$  tidak dapat diterima.

**Non-monotonic** dapat menyelesaikan masalah dengan informasi yang:

- tidak tepat
- dipertanyakan validitasnya
- tidak lengkap, dsb.

**Modus Ponens** : IF  $a, b, c$   
THEN  $d$

**Commonsense** : IF  $a, b, c$  assuming  $e, f, g$   
THEN  $d$   
UNLESS  $h, i, j$

#### 4.3.2 Statistical Reasoning Systems

Salah satu contohnya adalah Certainty Factor (CF)

$$CF(P1 \wedge P2) = \text{Min}(CF(P1), CF(P2))$$

$$CF(P1 \vee P2) = \text{Max}(CF(P1), CF(P2))$$

Contoh 1:

**IF** sukubunga = jatuh (CF = 0.6) **AND** pajak = berkurang (CF = 0.8)  
**THEN** saham = naik (CF aturan = 0.9)

CF premise:  $\min(0.6, 0.8) = 0.6$ .

CF aturan: 0.9

CF bahwa saham naik adalah  $0.6 * 0.9 = 0.54$

Contoh 2:

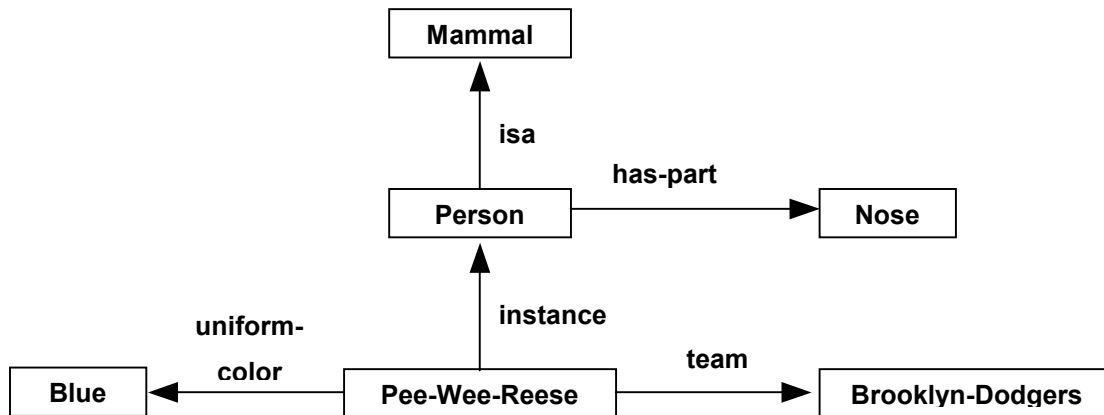
**Sentence 1: IF** A (CF = 0.3) **AND** B (CF = 0.6)  
**THEN** C (CF aturan = 0.5)

**Sentence 2: IF** D (CF = 0.4) **AND** E (CF = 0.7)  
**THEN** C (CF aturan = 0.9)

Sentence 1 **OR** sentence 2 =  $\text{Max}(\text{Min}(0.3, 0.6) * 0.5, \text{Min}(0.4, 0.7) * 0.9)$   
=  $\text{Max}(0.3 * 0.5, 0.4 * 0.9)$   
= 0.36

#### 4.3.3 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)`

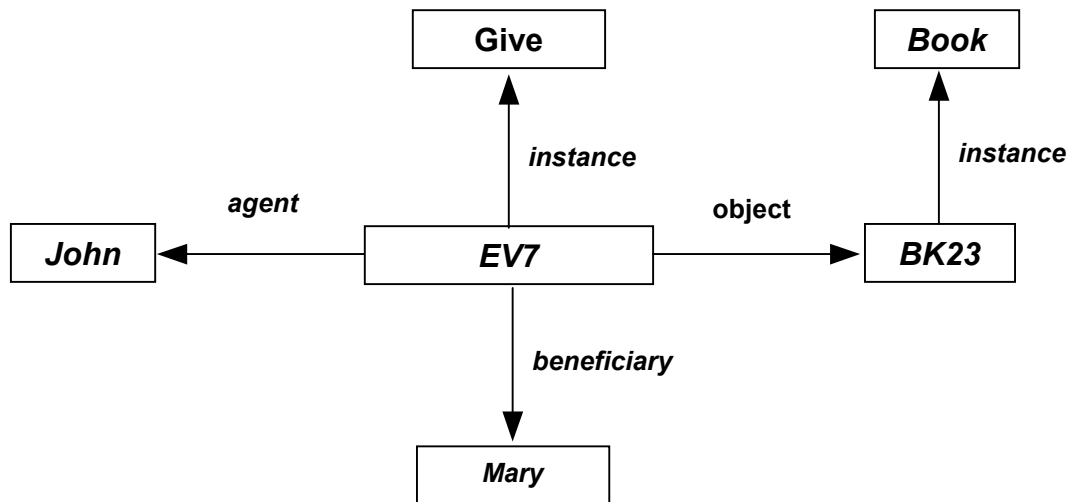
Dapat dilakukan **Inheritance** untuk menurunkan relasi tambahan:

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



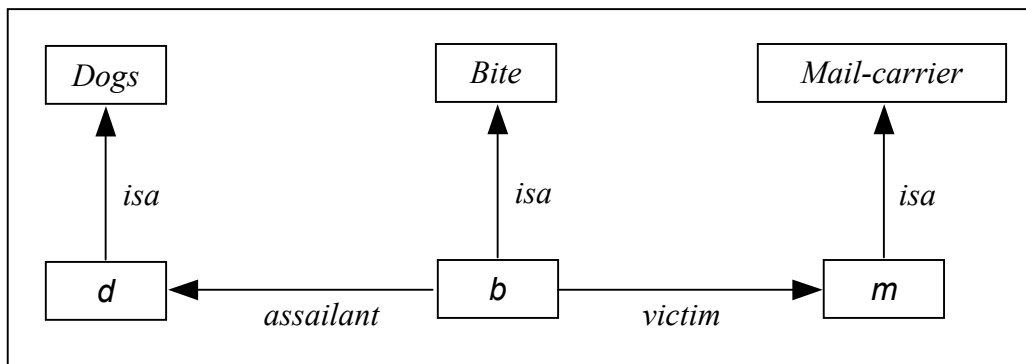
### Representasi Nonbinary Predicate

Jaringan Semantik untuk menggambarkan aspek dari kejadian tertentu.  
Sebagai contoh: "John gave the book to Mary".

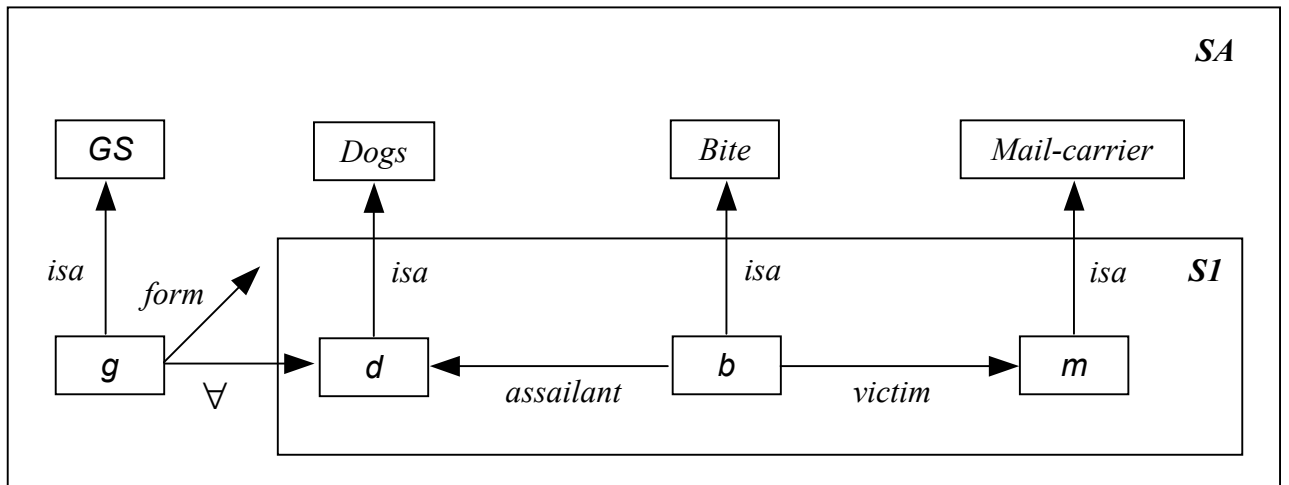


### Cara menggambarkan Quantifier dalam Semantic Nets

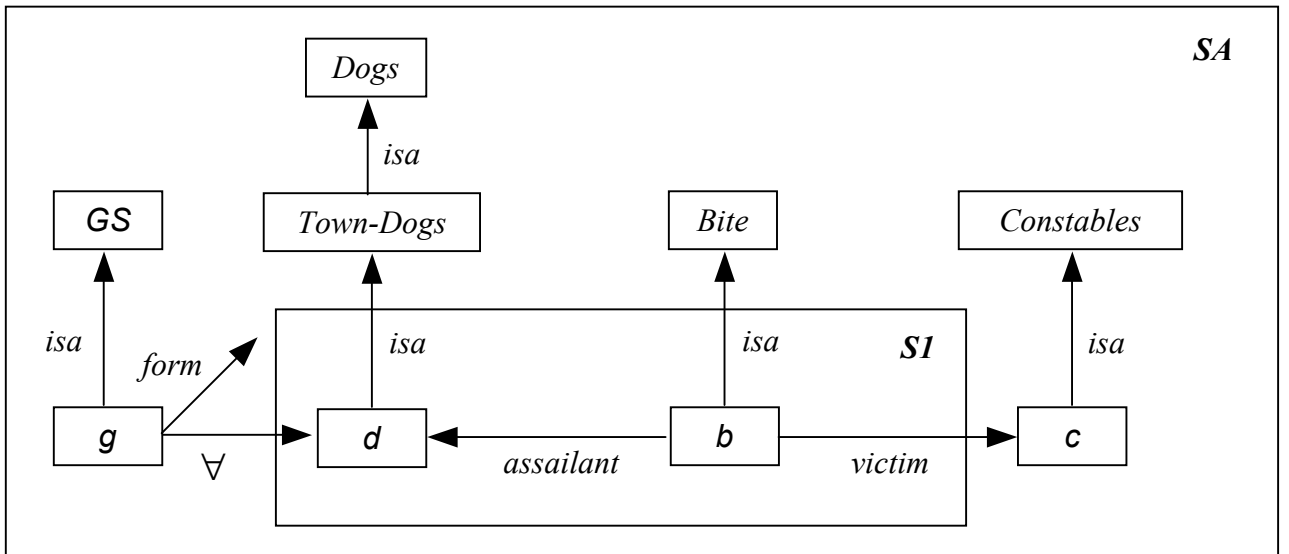
#### 1. The dog bit the mail carrier.



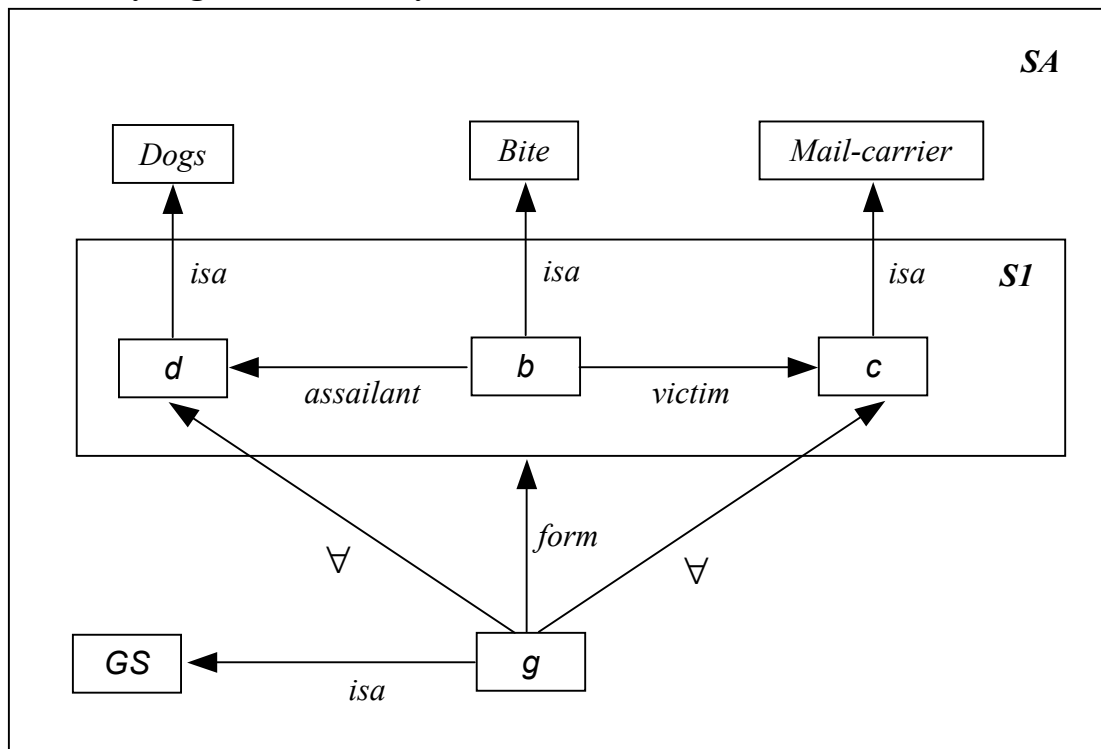
2. Every dog has bitten a mail carrier:  $\forall x \text{ Dog}(x) \Rightarrow \exists y \text{ Mail-carrier}(y) \wedge \text{Bite}(x,y)$ .



3. Every dog in town has bitten the constable (polisi penjaga).



#### 4. Every dog has bitten every mail carrier.



#### 4.3.4 Frames System

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

Nilai slot dapat berupa:

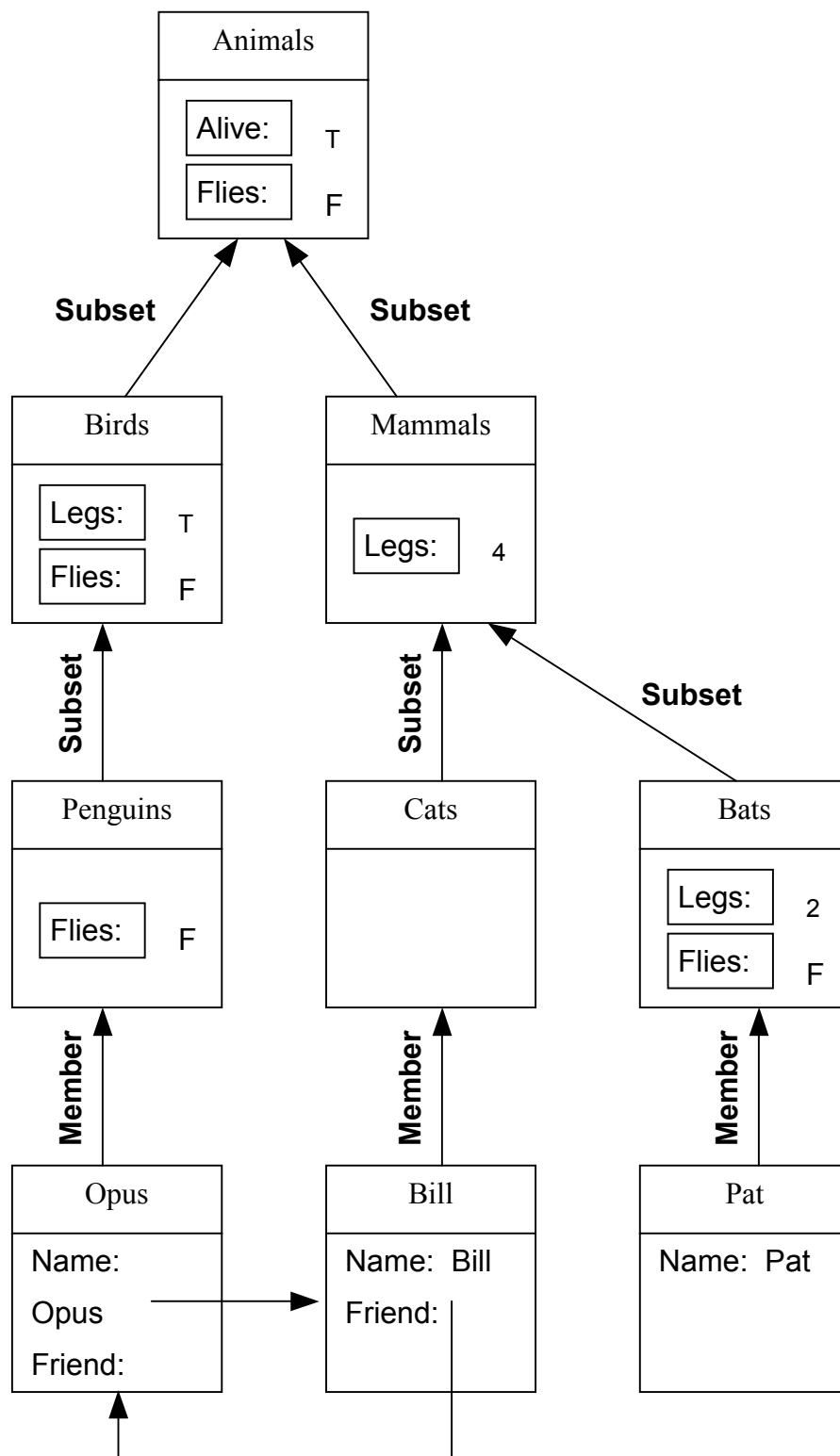
1. Identifikasi frame
2. relasi dengan frame lain (slotnya: isa, instance)
3. batasan nilai
4. nilai
5. default nilai (dapat diubah)
6. prosedur untuk mendapatkan nilai
7. prosedur yang dibangkitkan data (Data Driven): prosedur yang harus dilakukan jika nilai diubah, misalnya: periksa konsistensi.
8.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

Rel(Alive,Animals,T)

Rel(Flies,Animals,T)

Birds  $\subset$  Animals

Mammals  $\subset$  Animals

Rel(Flies,Birds,T)

Rel(Legs,Birds,2)

Rel(Legs,Mammals,4)

Penguins  $\subset$  Birds

Cats  $\subset$  Mammals

Bats  $\subset$  Mammals

Rel(Flies,Penguins,F)

Rel(Legs,Bats,2)

Rel(Flies,Bats,T)

Opus  $\in$  Penguins

Bill  $\in$  Cats

Pat  $\in$  Bats

Name(Opus,"Opus")

Name(Bill," Bill")

Friend(Opus,Bill)

Friend(Bill,Opus)

Name(Pat,"Pat")

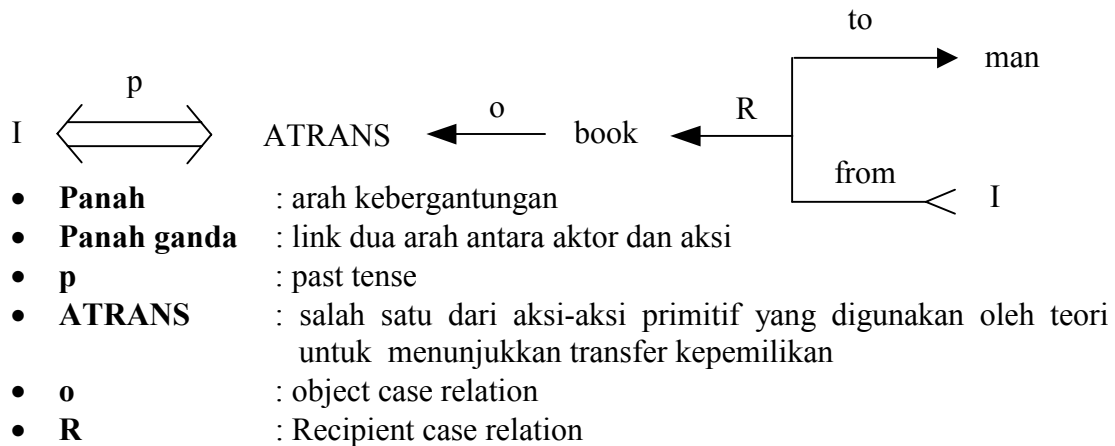
(b) Translation into FOL

#### 4.3.5 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.



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.


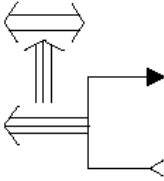
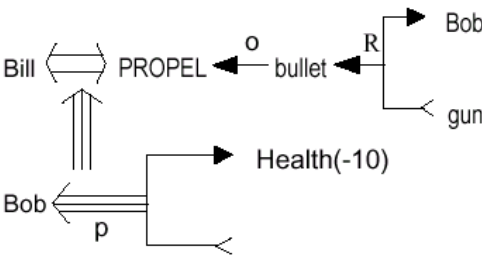
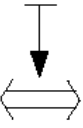
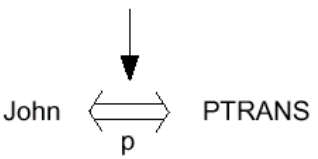

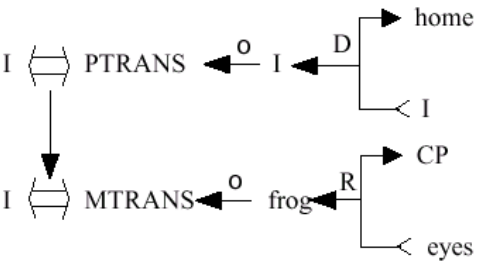

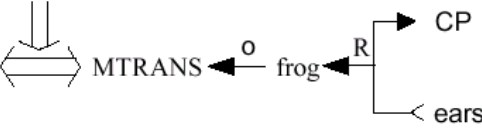
CD cocok untuk kalimat yang sederhana. Untuk primitif tingkat tinggi CD merepotkan.

**Misal:** “John bet Sam \$50 that the Mets would win the World Series”

Daftar ketergantungan antar konseptualisasi yang berhubungan dengan hubungan-hubungan semantik antar konsep-konsep yang mendasar adalah sebagai berikut:

1. Relasi antara aktor dan event yang disebabkan.		
PP $\longleftrightarrow$ ACT	John $\overset{p}{\longleftrightarrow}$ PTRANS	"John ran."
2. Relasi antara PP dan PA yang menyatakan deskripsi PP (misal dalam skala numerik).		
PP $\longleftrightarrow$ PA	John $\longleftrightarrow$ height (>average)	"John is tall."
3. Relasi antara dua PP, salah satunya milik himpunan yang didefinisikan oleh lainnya.		
PP $\longleftrightarrow$ PA	John $\longleftrightarrow$ doctor	"John is a doctor."
4. Relasi antara PP dan atribut yang menjadi predikatnya.		
PP $\uparrow$ PA	boy $\uparrow$ nice	"A nice boy."
5. Relasi antara dua PP, salah satunya mempunyai suatu jenis informasi khusus tentang yang lainnya. Selain poss-by (milik): LOC (lokasi), CONT (kandungan secara fisik).		
PP $\uparrow\uparrow$ PP	dog $\uparrow\uparrow$ Poss-by John	"John's dog."
6. Relasi antara ACT dan PP yang merupakan objek ACT. Anak panah ke ACT karena konteks khusus ACT menentukan arti relasi objek.		
ACT $\overset{o}{\leftarrow}$ PP	John $\overset{p}{\longleftrightarrow}$ PROPEL $\overset{o}{\leftarrow}$ cart	"John pushed the cart."

7. Relasi antara aksi dan sumber dan penerima aksi.		
		"John took the book from Mary."
8. Relasi antara aksi dan yang membentuknya. Alat harus konseptual lengkap, bukan sekedar objek fisik.		
		"John ate ice cream with a spoon."
9. Relasi antara aksi dan sumber fisik dan arah (destination).		
		"John fertilized the field."
10. Relasi antara PP dan kondisi awal & akhir.		
		"The plants grew."
11. Relasi antara dua konsep: yang satu merupakan penyebab yang lain. Pada contoh di bawah mendeskripsikan:		
<ul style="list-style-type: none"> <li>- penyebab aksi (a)</li> <li>- penyebab kondisi berubah (b)</li> </ul>		

<p>(a)</p>  <p>(b)</p> 		<p>"Bill shot Bob."</p>
12. Relasi antara konseptualisasi dan waktu terjadinya.		
	<p>yesterday</p> 	<p>"John ran yesterday."</p>
13. Relasi antara satu konsepsi dan lainnya yang terjadi lebih dulu.		
		<p>"While going home, I saw a frog."</p>
14. Relasi antara konseprual dan tempat terjadinya.		
<p>PP</p> 	<p>woods</p> 	<p>"I heard a frog in the woods."</p>

#### 4.3.6 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.

#### 4.3.7 CYC

Proyek pembangunan basis pengetahuan sangat besar untuk memuat pengetahuan akal sehat manusia (*commonsense*)

CYC memuat representasi untuk mendeskripsikan dunia nyata, mencakup: **event**, **object**, **attitude** etc.

Secara khusus memperhatikan masalah-masalah skala yang terjadi saat membangun KB dengan jutaan objek.

##### Motivasi pembangunan KB besar :

1. **Brittleness** (Kerapuhan)

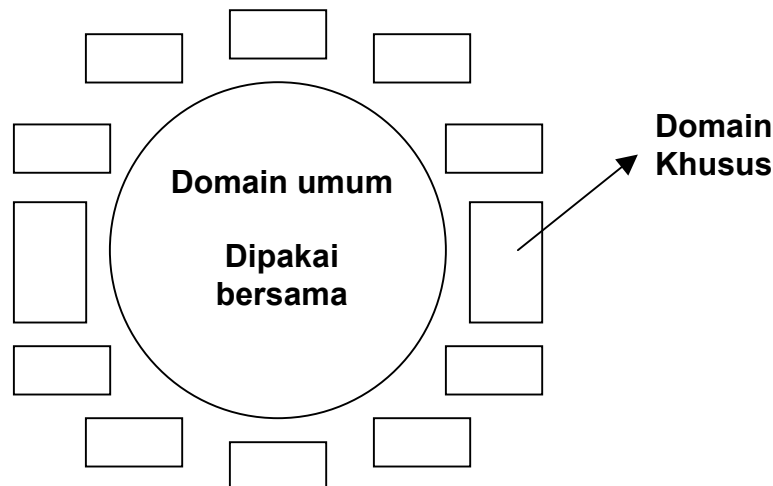
KB khusus biasanya rapuh, tidak mampu mengikuti situasi baru dan karena tidak lengkap maka tidak mampu menalar di luar domain.

2. **Form and Content** (Bentuk dan Isi)

Teknik-teknik untuk representasi dan penggunaan pengetahuan tidak mencukupi kebutuhan AI. Cara mengatasi adalah mengkodekan pengetahuan akal sehat yang besar dan mengamati letak kesulitan (kalau selama ini yang diamati adalah bentuk atau struktur, maka disini dicoba jika cakupan pengetahuan diperbesar). Ada yang mengatakan bahwa CYC adalah frame + logic.

3. **Shared Knowledge** (Pengetahuan yang dipakai bersama)

Sistem KB kecil biasanya menggunakan penyederhanaan asumsi tentang bagaimana merepresentasikan ruang, waktu, gerakan, struktur. Jika faktor-faktor tersebut dapat direpresentasikan pada level sangat tinggi sekaligus, maka sistem domain khusus dapat meraih perolehan dengan murah. Sistem-sistem yang memanfaatkan primitif-primitif yang sama secara terbagi dapat saling berkomunikasi dengan mudah.





## Bab 5

# Planning

Bab ini membahas teknik *planning* untuk menyelesaikan masalah-masalah yang dapat didekomposisi. Di sini kita akan melihat perbedaan teknik *planning* dengan teknik *searching*.

### 5.1 Menyelesaikan Masalah dengan Planning

*Planning* berbeda dengan *Search-Based Problem Solving* dalam hal representasi *goals*, *states*, dan *actions*, juga berbeda dalam representasi dan pembangunan urutan-urutan *action*. *Planning* berusaha untuk mengatasi kesulitan-kesulitan yang dialami dalam *Search-Based Problem Solving*.

*Planning* adalah suatu metode penyelesaian masalah dengan cara memecah masalah ke dalam sub-sub masalah yang lebih kecil, menyelesaikan sub-sub masalah satu demi satu, kemudian menggabungkan solusi-solusi dari sub-sub masalah tersebut menjadi sebuah solusi lengkap dengan tetap mengingat dan menangani interaksi yang ada antar sub masalah.

Pengujian keberfungsian suatu metode perencanaan dapat dilakukan pada suatu domain yang dinamakan **Dunia Balok** (*Blocks-World*). Dunia Balok dinilai cukup layak sebagai lahan pengujian karena tingkat kompleksitas permasalahan yang mungkin timbul di dalamnya. Di dalam Dunia Balok dikenal istilah *initial-state* dan *goal-state* yang masing-masing direpresentasikan oleh suatu komposisi dari sejumlah balok. Kemudian, ada satu set operator yang dapat diaplikasikan oleh sebuah tangan robot untuk memanipulasi balok. Permasalahan yang ada di dalam Dunia Balok adalah: Rangkaian operator seperti apa yang dapat mengubah *initial-state* menjadi *goal-state*? Rangkaian operator tersebut biasa disebut sebagai **Rencana Penyelesaian**.

Dua metode perencanaan yang cukup populer dan sudah pernah diuji pada Dunia Balok adalah **Goal-Stack-Planning** (GSP) dan **Constraint-Posting** (CP). GSP dan CP memiliki kelemahan dan keunggulan masing-masing. Dari segi kemudahan implementasi dan biaya komputasi, GSP lebih unggul dibanding CP. Sedangkan, dari segi efisiensi Rencana Penyelesaian yang dihasilkan, CP pada umumnya lebih unggul dibanding GSP.

Tetapi, dari seluruh kemungkinan permasalahan yang timbul pada Dunia Balok, meskipun GSP dan CP mampu menghasilkan rencana-rencana penyelesaian, namun rencana-rencana penyelesaian yang dihasilkan pada umumnya tidak efisien[RIC91]. Padahal, rencana penyelesaian yang efisien adalah salah satu hal yang penting, terutama pada sistem-sistem raksasa.

Pada domain tertentu, pemecahan masalah menggunakan komputer perlu dilakukan dengan cara bekerja pada bagian-bagian kecil dari masalah secara terpisah kemudian menggabungkan solusi-solusi per bagian kecil tersebut menjadi sebuah solusi lengkap dari masalah. Karena, jika hal-hal tersebut tidak dilakukan, jumlah kombinasi *state* dari

komponen masalah menjadi terlalu besar untuk dapat ditangani dipandang dari segi waktu yang tersedia.

Dua syarat untuk melakukan dekomposisi di atas adalah :

- Menghindari penghitungan ulang dari seluruh *state* masalah ketika terjadi perubahan dari suatu *state* ke *state* lainnya.
- Membagi masalah ke dalam beberapa sub masalah yang relatif lebih mudah untuk diselesaikan.

Penggunaan metode-metode yang terfokus pada cara mendekomposisi masalah ke dalam sub-sub masalah yang sesuai dan cara untuk mengingat dan menangani interaksi antar sub masalah ketika terjadi proses penyelesaian masalah tersebut diistilahkan dengan **Perencanaan**.

Pemecahan masalah dengan menggunakan perencanaan pada umumnya:

- Modus *Goal-Directed*, yaitu pencarian solusi dilakukan dari kondisi *goal-state* sampai ke kondisi *initial-state* yang dapat dicapai.
- Runut-Balik-Terpandu-Kebergantungan (*Dependency-Directed-Backtracking*) ketika menemukan jalan buntu.

Untuk berpindah dari satu *state* ke *state* lainnya, perencana diperkenankan untuk mengaplikasikan sejumlah operator. Sebuah sistem perencanaan pada umumnya perlu untuk mampu :

- Memilih operator
- Mengaplikasikan operator
- Mendeteksi ketika solusi telah tercapai
- Mendeteksi jalan-jalan buntu
- Mendeteksi ketika solusi yang hampir benar telah dicapai dan melakukan teknik khusus untuk membuat solusi tersebut menjadi benar

Pemilihan operator pada umumnya dilakukan dengan:

- Pertama, mengumpulkan kondisi pembeda *goal-state* dan *current -state* (*current -state* adalah suatu *state* yang menunjukkan kondisi saat ini).
- Kedua, mengidentifikasi operator yang dapat mengurangi perbedaan tersebut.

Pendeteksian bahwa solusi telah ditemukan dilakukan dengan cara menguji rangkaian operator yang telah dihasilkan. Jika rangkaian operator tersebut dapat mengubah kondisi *initial-state* menjadi *goal-state*, maka solusi telah ditemukan.

Pendeteksian jalan-jalan buntu dilakukan dengan: jika suatu rangkaian operator menghasilkan *state* yang diyakini bukan *state* antara *goal-state* dan *initial-state* atau jika kemajuan yang dicapai dianggap terlalu kecil atau menimbulkan permasalahan yang lebih sulit, maka rangkaian operator tersebut dianggap telah menemukan atau akan menuju jalan buntu.

Membuat solusi yang hampir benar menjadi benar dilakukan jika teknik yang dipakai dalam pencarian solusi hanya menghasilkan rangkaian solusi yang hampir benar. Perencana kemudian akan mengaplikasikan suatu metode tertentu untuk membuat solusi yang hampir benar tersebut menjadi benar.

Untuk menghindari pencatatan kondisi keseluruhan di setiap titik persimpangan (*node*) dalam pencarian solusi, pengaplikasian operator memerlukan tiga daftar predikat untuk mendeskripsikan perubahan kondisi, yaitu:

- **PRECONDITION** : predikat-predikat yang harus bernilai benar sebelum pengaplikasian operator
- **ADD** : predikat-predikat yang bernilai benar setelah pengaplikasian suatu operator.
- **DELETE** : predikat-predikat yang bernilai salah setelah pengaplikasian suatu operator

Ketiga daftar predikat tersebut selanjutnya disebut dengan **daftar-PAD**.

## 5.2 Dunia Balok

Dunia Balok adalah sebuah domain dengan karakteristik:

- Memiliki sebuah permukaan datar tempat menyimpan balok, umumnya disebut dengan meja.
- Memiliki sejumlah balok kotak yang berukuran sama.
- Memiliki sebuah tangan robot yang dapat memanipulasi balok.

Untuk memudahkan pendefinisian kondisi balok pada suatu *state* dalam Dunia Balok digunakan predikat-predikat berikut :

- **ON(A,B)** – Balok A menempel di atas balok B.
- **ONTABLE(A)** – Balok A berada di permukaan meja.
- **CLEAR(A)** – Tidak ada balok yang sedang menempel di atas balok A.

Sedangkan, untuk tangan robot digunakan :

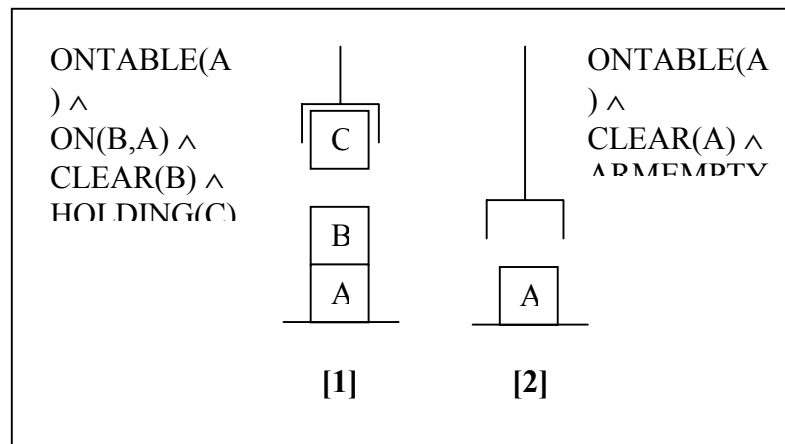
- **HOLDING(A)** – Tangan robot sedang memegang balok A.
- **ARMEMPTY** – Tangan robot tidak sedang memegang balok.

$$\begin{aligned} \forall x \text{ ONTABLE}(x) &\rightarrow \neg \text{HOLDING}(x) \wedge [\neg \exists y \text{ ON}(x,y)] \\ \forall x \text{ CLEAR}(x) &\rightarrow \neg \exists y \text{ ON}(y,x) \\ \forall x [\exists y \text{ ON}(x,y)] &\rightarrow \neg \text{HOLDING}(x) \wedge \neg \text{HOLDING}(y) \wedge \neg \text{ON}(y,x) \wedge [\neg \exists z \end{aligned}$$

**Notasi 5.1** Definisi Kondisi Ditinjau dari Sudut Pandang Balok

$$\exists x \text{ HOLDING}(x) \rightarrow \neg \text{ARMEMPTY} \wedge \neg \text{ONTABLE}(x) \wedge [\neg \exists y \text{ ON}(x,y) \vee \text{ON}(y,x)]$$

**Notasi 5.2** Definisi Kondisi Ditinjau dari Sudut Pandang Tangan Robot



**Gambar 5.1** Contoh dua buah *State*.

**Tabel 5.1** Daftar Operator untuk Tangan Robot

Operator	Hal yang Dilakukan
<i>STACK(A,B)</i>	Menempelkan balok A di atas balok B
<i>UNSTACK(A,B)</i>	Mengangkat balok A yang menempel di atas balok B
<i>PICKUP(A)</i>	Mengangkat balok A dari permukaan meja
<i>PUTDOWN(A)</i>	Menempelkan balok A di permukaan meja

<i>STACK(A,B)</i>
P : $HOLDING(A) \wedge CLEAR(B)$
A : $ON(A,B) \wedge ARMEMPTY$
D : $HOLDING(A) \wedge CLEAR(B)$
<i>UNSTACK(A,B)</i>
P : $ON(A,B) \wedge CLEAR(A) \wedge ARMEMPTY$
A : $HOLDING(A) \wedge CLEAR(B)$
D : $ON(A,B) \wedge ARMEMPTY$
<i>PICKUP(A)</i>
P : $ONTABLE(A) \wedge CLEAR(A) \wedge ARMEMPTY$
A : $HOLDING(A)$
D : $ONTABLE(A) \wedge ARMEMPTY$
<i>PUTDOWN(A)</i>
P : $HOLDING(A)$
A : $ONTABLE(A) \wedge ARMEMPTY$
D : $HOLDING(A)$
LEGEND :
P : <i>PRECONDITION</i>
A : ADD
D : DELETE

**Notasi 5.2** Daftar Operator dan Kondisi Balok Terkait untuk Tangan

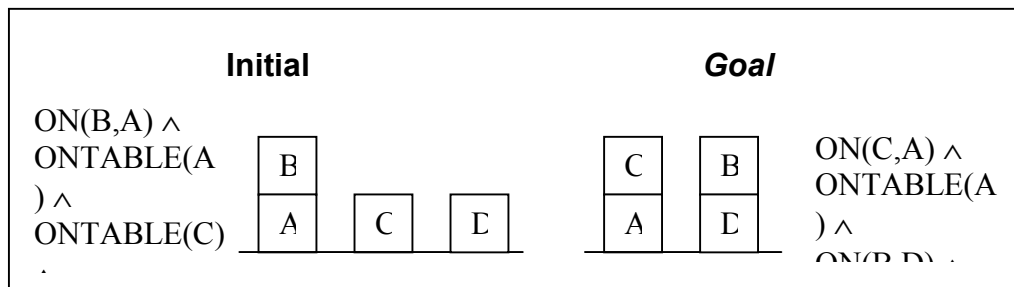
Permasalahan yang akan dicari rencana penyelesaiannya adalah permasalahan yang memenuhi spesifikasi berikut:

- Jumlah balok sama.
- Balok yang berada pada *initial-state* juga berada pada *goal-state*.
- Kondisi terdefinisi dengan benar.
- Tangan robot tidak sedang memegang balok.

### 5.3 Goal-Stack-Planning (GSP)

Dalam menyelesaikan sebuah masalah, GSP menggunakan sebuah *stack* untuk menampung kondisi-kondisi (kondisi *goal* dan kondisi-kondisi yang mungkin terjadi ketika pencarian solusi) dan operator-operator yang telah diajukan untuk memenuhi kondisi-kondisi tersebut. Juga, tergantung pada sebuah basis data yang menggambarkan *current -state* dan satu set operator yang dideskripsikan sebagai daftar predikat *PRECONDITION*, *ADD* dan *DELETE* (notasi 5.3) atau **daftar-PAD**.

Langkah pertama GSP dalam menyelesaikan sebuah masalah adalah menempatkan kondisi-kondisi *goal-state* pada *stack*. Kondisi-kondisi tersebut akan disimpan di dalam sebuah slot *stack*. Sebagai contoh, untuk masalah-1 (gambar 5.2), isi *stack* pada saat awal penyelesaian masalah dapat dilihat pada notasi 5.4. *Current -state* diisi dengan *initial-state*.



**Gambar 5.3** Masalah-1.

$$ON(C,A) \wedge ON(B,D) \wedge ONTABLE(A) \wedge$$

**Notasi 5.4** Isi *Stack*.

Langkah kedua, mengacu pada *current -state*, kondisi-kondisi *goal-state* yang belum tercapai dimasukkan ke dalam *stack*, masing-masing menempati sebuah *slot*. GSP tidak memiliki aturan khusus yang mengatur urutan pemasukan ke dalam *stack* dari kondisi-kondisi yang belum tercapai tersebut. Untuk masalah-1, ada dua kombinasi yang

mungkin terjadi. Kedua kombinasi tersebut dapat dilihat pada notasi 5.5. Pernyataan **OTAD** pada notasi 5.5 merupakan kependekan dari  $ONTABLE(A)$  dan  $ONTABLE(D)$ .

ON(C,A)	ON(B,D)
ON(B,D)	ON(C,A)
$ON(C,A) \wedge ON(B,D) \wedge OTAD \wedge ON(C,A) \wedge ON(B,D) \wedge OTAD$	

**Notasi 5.5** Isi *Stack*.

Langkah ketiga, slot terisi yang berada paling atas pada *stack* akan diperiksa. Hal-hal yang akan dilakukan bergantung pada kondisi slot tersebut. Kondisi yang mungkin terjadi pada slot tersebut adalah sebagai berikut:

- **Kondisi 1**

Jika slot berisi kondisi yang sudah memenuhi *current -state*, tetapi slot tidak terletak di dasar *stack* dan juga tidak terletak di atas slot yang berisi operator, maka isi slot akan di-pop dari *stack* dan pemeriksaan dilanjutkan pada slot berikutnya.

- **Kondisi 2**

Jika slot berisi kondisi yang belum memenuhi *current -state* maka isi slot akan di-pop dari *stack*. Kemudian, sebuah operator yang sesuai untuk mencapai kondisi tersebut akan dimasukkan ke dalam *stack*. Setelah itu, serangkaian kondisi yang dibutuhkan agar operator itu bisa diaplikasikan akan dimasukkan ke dalam sebuah slot *stack*. Selanjutnya, setiap kondisi dari rangkaian kondisi yang dibutuhkan operator agar dapat diaplikasikan tersebut akan dimasukkan ke dalam sebuah slot secara terurut, dimana kondisi yang harus dicapai paling akhir dimasukkan pertama kali.

- **Kondisi 3**

Jika slot berisi kondisi atau rangkaian kondisi dan slot tersebut berada di atas slot yang berisi operator, maka isi slot teratas dari *stack* tersebut akan di-pop. Kemudian, operator pada slot berikutnya akan di-pop dan dimasukkan ke dalam antrian operator dalam rencana penyelesaian dan *current -state* di-update dengan mengaplikasikan operator tersebut pada *current -state* berdasarkan daftar-PAD.

- **Kondisi 4**

Jika slot yang diperiksa adalah slot terdasar maka akan diuji kesamaan antara *current -state* dan *goal-state*. Jika sama (berarti *goal-state* telah tercapai) maka isi slot akan di-pop dan pencarian rencana penyelesaian dihentikan. Jika berbeda (*goal-state* belum tercapai) maka langkah ke dua diulangi. Jika kondisi yang terjadi bukan kondisi 4, setelah rangkaian tindakan yang bersesuaian dilakukan, langkah ketiga diulangi.

Sebagai contoh, jika isi *stack* yang tersusun sama dengan notasi 5.5[1], berikut urutan kemunculan kondisi isi *stack* dan *current -state*:

CLEAR(A)	
HOLDING(C)	
CLEAR(A)	^
HOLDING(C)	
<b>STACK(C,A)</b>	
ON(B,D)	
ON(C,A)    ON(D,D)	

**Notasi 5.6** Isi Stack.

ON(B,A)	
CLEAR(B)	
ARMEMPTY	
ON(B,A)	^    CLEAR(B)
ARMEMPTY	
<b>UNSTACK(B,A)</b>	
HOLDING(C)	
CLEAR(A) ^ HOLDING(C)	
<b>STACK(C,A)</b>	

**Notasi 5.7** Isi Stack.

ONTABLE(A) ^ ONTABLE(C) ^ ONTABLE(D) ^ HOLDING(B)
---

**Notasi 5.8** Current-state.

HOLDING(C)	
CLEAR(A)	^
HOLDING(C)	
<b>STACK(C,A)</b>	
ON(B,D)	

**Notasi 5.9** Isi Stack.

ONTABLE(C)	
CLEAR(C)	
ARMEMPTY	
ONTABLE(C)	^    CLEAR(C)
ARMEMPTY	
<b>PICKUP(C)</b>	
CLEAR(A) ^ HOLDING(C)	
<b>STACK(C,A)</b>	

**Notasi 5.10** Isi Stack.

CLEAR(D)	
HOLDING(B)	
CLEAR(D) ^ HOLDING(B)	
<b>STACK(B,D)</b>	
ONTABLE(C)	^    CLEAR(C)
ARMEMPTY	
<b>PICKUP(C)</b>	
CLEAR(A) ^ HOLDING(C)	
<b>STACK(C,A)</b>	

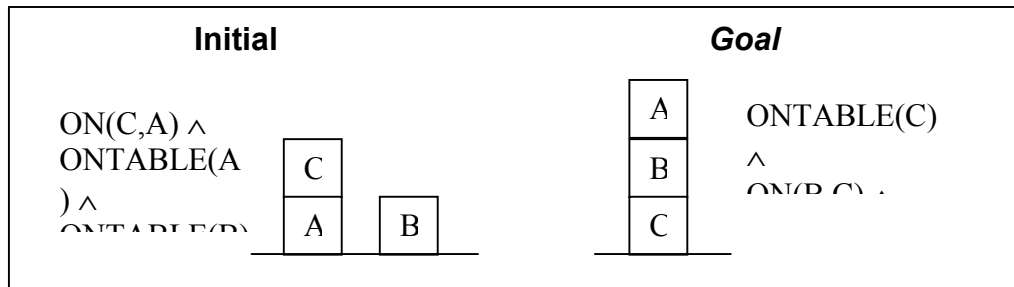
**Notasi 5.11** Isi Stack.

- |                 |
|-----------------|
| 1. UNSTACK(B,A) |
| 2. STACK(B,D)   |
| 3. PICKUP(C)    |
| 4. STACK(C,A)   |

**Notasi 5.12** Sebuah Rencana Penyelesaian.



Untuk masalah semudah masalah-1, GSP terbukti dapat menghasilkan rencana yang efisien (notasi 5.12) dan dinilai layak digunakan [RIC91].



**Gambar 5.3** Masalah 2.

Tetapi, ketika dihadapkan pada masalah yang sedikit lebih rumit, seperti masalah-2 (gambar 5.3), GSP mulai memperlihatkan keterbatasannya. Dengan asumsi urutan isi *stack* pada awal proses penyelesaian masalah seperti pada notasi 5.13, rencana penyelesaian yang akan dihasilkan oleh GSP dapat dilihat pada notasi 5.14 [RIC91].

ON(A,B)  
ON(B,C)  
ON(A,B)  $\wedge$

**Notasi 5.13** Isi  
*Stack*

1. UNSTACK(C,A)
2. PUTDOWN(C)
3. PICKUP(A)
4. STACK(A,B)
5. UNSTACK(A,B)
6. PUTDOWN(A)
7. PICKUP(B)
8. STACK(B,C)
9. PICKUP(A)

**Notasi 5.14** Sebuah Rencana  
Penyelesaian

Rencana penyelesaian pada notasi 5.14 bukan merupakan sebuah rencana penyelesaian yang efisien (terlihat dari adanya langkah ke-3 sampai dengan ke-6). Rencana penyelesaian yang efisien bagi masalah-2 dapat dilihat pada notasi 5.15.

1. UNSTACK(C,A)
2. PUTDOWN(C)
3. PICKUP(B)
4. STACK(B,C)
5. PICKUP(A)
6. STACK(A,B)

**Notasi 5.15** Sebuah Rencana  
Penyelesaian

$$\begin{array}{l} \text{ON(B,C)} \\ \text{ON(A,B)} \\ \text{ON(A,B)} \wedge \end{array}$$

**Notasi 5.16** Isi  
*Stack*

Penyebab ketidakefisienan tersebut adalah kondisi yang belum tercapai yang dipilih untuk diselesaikan lebih dulu bukanlah kondisi yang harus diselesaikan terlebih dahulu. GSP tidak memiliki metode khusus untuk mengurutkan pemasukan kondisi-kondisi yang belum tercapai ke dalam *stack* pada saat langkah pertama dilakukan. GSP akan menghasilkan rencana yang efisien jika urutan isi *stack* pada awal proses penyelesaian seperti notasi 5.16. Bisa dikatakan GSP menemui jalan buntu yang tidak disadari karena: meskipun ada langkah yang meng-undo langkah lainnya (*STACK(x,y)* di-undo oleh *UNSTACK(x,y)* dan *PICKUP(x)* di-undo oleh *PUTDOWN(x)*), seluruh langkah yang dibangkitkan tetap dipakai.

Karena ketidakefisienan tersebut, GSP dinilai tidak layak untuk menyelesaikan masalah yang sesulit ini, atau yang lebih sulit lagi [RIC91]. Selain itu, GSP juga mungkin menemui jalan buntu yang disadari pada saat proses pencarian rencana penyelesaian terjadi. Memasuki jalan buntu diketahui sebagai salah satu hal yang menyebabkan ketidakefisienan biaya komputasi. Karena GSP mungkin memasuki jalan buntu maka GSP diasumsikan tidak efisien dalam hal biaya komputasi.

**Algoritma global GSP**

1. Menempatkan seluruh kondisi *goal-state* pada *slot* terdasar dari *stack*.
2. Memasukkan setiap kondisi *goal-state* yang belum tercapai ke dalam sebuah *slot stack*.
3. Menghilangkan kondisi yang sudah dicapai dalam *stack*, mengganti kondisi yang belum dicapai dengan operator atau memindahkan operator yang sudah bisa diaplikasikan ke dalam rencana penyelesaian kemudian kembali mengulangi langkah ke-3. Atau, menguji kesamaan *current-state* dengan *goal-state* kemudian mengulangi langkah 2 jika berbeda dan berhenti jika sama.

GSP mungkin menemui jalan buntu yang disadari (meningkatkan biaya komputasi) ataupun yang tidak disadari (membuat rencana penyelesaian tidak efisien). Jalan buntu yang tidak disadari disebabkan karena GSP tidak memiliki aturan untuk mengurutkan pemasukan kondisi *goal-state* yang belum tercapai ke dalam *stack* (langkah ke-2).

**5.4 Constraint Posting (CP)**

Berbeda dengan GSP yang berusaha mencapai kondisi *goal-state* dengan secara terurut berusaha mencapai sebuah kondisi pada suatu saat dengan operator-operator yang

dibangkitkan pada rencana penyelesaian yang dihasilkan terurut untuk menyelesaikan kondisi pertama yang ingin dicapai, kedua dan seterusnya (*linear-planning*), CP bekerja secara paralel untuk mencapai kondisi-kondisi pada *goal-state* (non-linear-planning), berdasarkan kepada keyakinan bahwa: pada kebanyakan masalah, sub-sub masalah perlu untuk dikerjakan secara simultan [RIC91]. CP membangun rencana penyelesaian dengan secara bertahap menganalisa operator-operator, secara parsial mengurutkan operator-operator dan melekatkan variabel antar operator. Sehingga, pada suatu saat dalam proses penyelesaian masalah, bisa didapatkan sekumpulan operator meskipun cara yang jelas untuk mengurutkan operator-operator tersebut mungkin tidak dimiliki.

**Tabel 5.2** Fungsi-fungsi Pemandu pada CP (TWEAK)

Pemandu	Fungsi
Step-addition	Membuat langkah baru.
<i>Promotion</i>	Menempatkan suatu langkah sebelum langkah lainnya pada rencana penyelesaian akhir.
<i>Declobbering</i>	Menempatkan sebuah langkah s2 (mungkin baru) di antara dua langkah yang sudah ada, s1 dan s3, mengembalikan prekondisi dari s3 yang dihilangkan (atau di- <i>clobber</i> ) oleh s1.
<i>Simple-Establishment</i>	Menetapkan sebuah nilai ke dalam sebuah variabel, dalam rangka memastikan <i>precondition</i> untuk beberapa langkah.
<i>Separation</i>	Mencegah penetapan suatu nilai ke dalam suatu variabel.

Untuk memperjelas cara CP dalam menyelesaikan masalah, akan dipaparkan langkah-langkah yang dilakukan CP dalam menyelesaikan masalah-2 (gambar 5.3). Dua kondisi yang ingin dicapai adalah *ON(A,B)* dan *ON(B,C)*. *ON(A,B)* dapat dicapai dengan *STACK(A,B)* dan *ON(B,C)* dapat dicapai dengan *STACK(B,C)*, seperti diilustrasikan oleh notasi 5.17, tanda \* menunjukkan kondisi yang belum tercapai agar operator bisa dioperasikan, dan tanda  $\neg$  menunjukkan kondisi yang dihilangkan setelah operator diaplikasikan.

<u>CLEAR(B)</u>	<u>CLEAR(C)</u>
<u>*HOLDING(A)</u>	<u>*HOLDING(B)</u>
$\neg$ <u>STACK(A,B)</u>	$\neg$ <u>STACK(B,C)</u>
ARMEMPTY	ARMEMPTY
ON(A,B)	ON(B,C)
CLEAR(B)	CLEAR(C)

**Notasi 5.17** Penulisan Langkah.

Karena A dan B keduanya berada di permukaan meja, operator yang cocok untuk mencapai *HOLDING(A)* ialah dengan mengaplikasikan *PICKUP(A)*, dan *HOLDING(B)* dengan *PICKUP(B)* (notasi 5.18).

*CLEAR(A)	*CLEAR(B)
ONTABLE(A)	ONTABLE(B)
<u>*ARMEMPTY</u>	<u>*ARMEMPTY</u>
$\neg$ <u>PICKUP(A)</u>	$\neg$ <u>PICKUP(B)</u>
HOLDING(A)	HOLDING(B)

\**ARMEMPTY* muncul pada kedua langkah karena ada langkah lain yang memiliki *postcondition*  $\neg \text{ARMEMPTY}$ , \**CLEAR(A)* muncul karena pada kondisi *initial-state* *CLEAR(A)* belum terpenuhi, \**CLEAR(B)* muncul karena ada langkah lain yang memiliki *postcondition*  $\neg \text{CLEAR(B)}$ . Karena *PICKUP(A)* dibuat untuk memenuhi *precondition* dari *STACK(A,B)* dan *PICKUP(B)* dibuat untuk memenuhi *precondition* dari *STACK(B,C)*, dihasilkan apa yang ditampilkan pada notasi 5.19, baris pertama dan ke dua. Baris ke tiga ditujukan untuk memenuhi \**CLEAR(B)* karena *PICKUP(B)* memiliki *precondition* *CLEAR(B)* dan tidak dihilangkan setelahnya.

PICKUP(A)  $\leftarrow$  STACK(A,B)  
PICKUP(B)  $\leftarrow$  STACK(B,C)  
PICKUP(B)  $\leftarrow$  CLEAR(B)

#### Notasi 5.19 Operator-operator Yang Terurut

Kini ada tiga kondisi yang belum terpenuhi, \**ARMEMPTY* sebagai *precondition* *PICKUP(A)* dan *PICKUP(B)* kemudian \**CLEAR(A)* sebagai *precondition* *PICKUP(A)*. Katakanlah kita mencoba untuk mencapai \**ARMEMPTY* pada langkah *PICKUP(A)* dan *PICKUP(B)*. *PICKUP(A)* dan *PICKUP(B)* saling menyebabkan \**ARMEMPTY* muncul pada *precondition* masing-masing. Dengan menggunakan *promotion*, salah satu langkah bisa lebih didahulukan dibanding yang lainnya untuk memanfaatkan kondisi *ARMEMPTY* pada *initial-state*. Katakanlah kita mencoba mendahulukan *PICKUP(A)* dibandingkan *PICKUP(B)* (notasi 5.20, baris pertama).

PICKUP(A)  $\leftarrow$  PICKUP(B)  
PICKUP(A)  $\leftarrow$  PUTDOWN(A)  $\leftarrow$

#### Notasi 5.20 Operator-operator Yang Terurut

*Postcondition* *PICKUP(A)* meng-clobber *precondition* \**ARMEMPTY* dari *PICKUP(B)* dan untuk men-declobber-nya dapat dicoba dua cara. Pertama, menggunakan langkah yang telah ada, yaitu *STACK(A,B)*, tetapi *STACK(A,B)* telah dikondisikan untuk ada setelah *PICKUP(B)*, jadi *STACK(A,B)* tidak dapat dipakai. Ke dua, menggunakan langkah baru, yaitu *PUTDOWN(A)* (notasi 5.20, baris ke dua). Tetapi *PUTDOWN(A)* setelah *PICKUP(A)* mengembalikan kondisi A ke posisi sebelum *PICKUP(A)* dilaksanakan. Kedua cara tersebut tidak layak diaplikasikan karena pendefinisian

$PICKUP(A)$  mendahului  $PICKUP(B)$  tidak tepat (jalan buntu).  $PICKUP(A)$  mendahului  $PICKUP(B)$  harus dibatalkan dan dicoba sebaliknya (notasi 5.21, baris pertama).

$PICKUP(B) \leftarrow PICKUP(A)$ $PICKUP(B) \leftarrow STACK(B,C) \leftarrow$
--

#### Notasi 5.21 Penulisan Langkah.

*Precondition*  $ARMEMPTY$  pada  $PICKUP(A)$  yang di-clobber oleh *postcondition*  $\neg ARMEMPTY$  pada  $PICKUP(B)$  dapat di-declobber dengan langkah yang sudah ada, yaitu  $STACK(B,C)$  (notasi 5.21, baris ke dua). Kondisi yang belum tercapai kini hanya  $*CLEAR(A)$ .  $CLEAR(A)$  dapat dicapai dengan mengaplikasikan  $UNSTACK(x,A)$ , dengan  $x$  sama dengan  $C$  (lihat notasi 5.22).

$*CLEAR(x)$ $*ON(x,A)$ <u><math>*ARMEMPTY</math></u>  <del><math>UNSTACK(x,A)</math></del>  $HOLDING(C)$ $CLEAR(A)$ $\neg ON(x,A)$
--

#### Notasi 5.22 Penulisan Langkah.

Tetapi, langkah  $UNSTACK(C,A)$  memiliki tiga *precondition* yang belum tercapai. Namun hal tersebut dapat dihadapi dengan *promotion* (notasi 5.23)

$UNSTACK(x,A) \leftarrow STACK(B,C)$ $UNSTACK(x,A) \leftarrow PICKUP(A)$ $UNSTACK(x,A) \leftarrow PICKUP(B)$
--

#### Notasi 5.23 Operator-operator Yang Terurut Sebagian.

Pada notasi 5.23, baris pertama terjadi karena  $UNSTACK(C,A)$  harus dilakukan sebelum ada sesuatu di atas  $C$ . Baris ke dua terjadi karena  $PICKUP(A)$  baru dapat dilakukan jika tidak ada balok di atas  $A$ . Baris ke tiga terjadi karena operator  $UNSTACK$  ditempatkan lebih dahulu dibandingkan  $PICKUP$  jika hanya  $*ARMEMPTY$  yang menghubungkan keduanya. Dari seluruh fungsi pemandu, *step-addition* adalah yang paling rumit untuk diaplikasikan, karena harus dilakukan pengecekan apakah penambahan langkah yang baru akan meng-clobber langkah-langkah yang telah ada.  $PICKUP(B)$  membutuhkan  $ARMEMPTY$ , tetapi penambahan  $UNSTACK(C,A)$  meng-clobber  $ARMEMPTY$  tersebut, demikian pula untuk  $ARMEMPTY$  pada *precondition*  $PICKUP(A)$ . langkah yang sudah ada tidak ada yang cocok untuk melakukan declobbering tersebut sehingga dibutuhkan

langkah baru. Berdasarkan pada *goal-state*, *PUTDOWN(C)* adalah langkah yang baik (notasi 5.24).

HOLDING(C)
PUTDOWN(C)
ONTABLE(C)
ARMEMPTY

Setelah dilakukan *promotion* pada langkah-langkah yang ter-*clobber* oleh *step-addition* tersebut, kini seluruh *precondition* langkah telah terpenuhi. Rencana penyelesaian tinggal disusun dari langkah-langkah yang telah ada berdasarkan urutan yang telah ditentukan. Rencana penyelesaian yang dihasilkan sama dengan notasi 5.15.

Pada umumnya, CP menghasilkan rencana penyelesaian yang lebih efisien dibanding GSP. Tetapi, efisiensi itu harus dibayar dengan pengimplementasian metode yang lebih sulit dan biaya komputasi yang lebih tinggi dibandingkan dengan GSP.

Pernyataan biaya komputasi CP yang lebih tinggi dibandingkan GSP disimpulkan dari pemikiran mengenai :

- Setiap penambahan langkah baru pada CP membutuhkan pencarian pengaruh pada dan dari seluruh langkah yang sudah ada sebelumnya.
- Setiap pengurutan langkah baru pada CP membutuhkan perbandingan dengan seluruh urutan langkah yang sudah ada sebelumnya.

Pernyataan pengimplementasian CP yang lebih sulit dibandingkan GSP disimpulkan dari pemikiran sebagai pemrogram mengenai :

- Logika algoritma yang dibutuhkan oleh setiap fungsi pemandu yang diperlukan CP (lihat tabel 5.2).
- Pendefinisian segala variabel dan struktur data dari setiap variabel yang dibutuhkan CP untuk mendeskripsikan *state*, langkah, urutan langkah serta hubungan antar langkah dan urutan langkah yang dapat mendukung dan didukung oleh fungsi-fungsi pemandu.

Meskipun CP terlihat membutuhkan usaha pengimplementasian dan biaya komputasi yang lebih tinggi dibanding GSP, CP masih mungkin menghasilkan rencana yang tidak efisien.

### Algoritma Global CP

- 1) Inisialisasi *S* sebagai satu set *proposisi* pada *goal-state*.
- 2) Hilangkan beberapa *proposisi*-yang-belum-tercapai *P* dari *S*.
- 3) Capai *P* dengan menggunakan salah satu fungsi pemandu yang ada.
- 4) *Review* seluruh langkah dalam rencana untuk melihat *precondition* yang belum tercapai. Tambahkan *precondition* yang belum tercapai ke dalam *S*.
- 5) Jika *S* kosong, lengkapi rencana penyelesaian dengan mengkonversi urutan sebagian langkah menjadi urutan lengkap.

6) Jika tidak, kembali ke langkah ke-2.

## Bab 6

# Sistem Intelijensia Adaptif

Bab ini membahas tiga jenis sistem intelijensia adaptif, yaitu: *artificial neural network*, *genetic algorithm*, dan *fuzzy system*. Kita akan mempelajari karakteristik dan perbedaan dari ketiga sistem.

### 6.1 Jaringan Syaraf Tiruan (Artificial Neural Network)

Jaringan Syaraf Tiruan (JST) adalah prosesor tersebar paralel yang sangat besar (*massively paralel distributed processor*) yang memiliki kecenderungan untuk menyimpan pengetahuan yang bersifat pengalaman dan membuatnya siap untuk digunakan (Aleksander & Morton 1990).

**JST menyerupai otak manusia dalam dua hal, yaitu:**

1. Pengetahuan diperoleh jaringan melalui proses belajar.
2. Kekuatan hubungan antar sel syaraf (*neuron*) yang dikenal sebagai bobot-bobot sinaptik digunakan untuk menyimpan pengetahuan.

**JST mempunyai sifat dan kemampuan:**

1. Nonlinieritas (*Nonlinearity*)
2. Pemetaan Input-Output (*Input-Output Mapping*)
3. Adaptivitas (*Adaptivity*)
4. Respon Yang Jelas (*Evidential Response*)
5. Informasi Yang Sesuai Dengan Keadaan (*Contextual Information*)
6. Toleransi Kesalahan (*Fault Tolerance*)
7. Kemampuan Implementasi Pada VLSI (*VLSI Implementability*)
8. Keseragaman Analisis Dan Perancangan (*Unifomity of Analysis and Design*)
9. Analogi Sel Syaraf Biologi (*Neurobiological Analogy*)

#### 6.1.1 Model Sel Syaraf (Neuron)

Satu sel syaraf dapat dimodelkan secara matematis seperti diilustrasikan oleh gambar 6.1. Satu sel syaraf terdiri dari tiga bagian, yaitu: fungsi penjumlah (*summing function*), fungsi aktivasi (*activation function*), dan keluaran (*output*).

Secara matematis kita bisa menggambarkan sebuah *neuron k* dengan menuliskan pasangan persamaan sebagai berikut :

$$u_k = \sum_{j=1}^p w_{kj} x_j$$

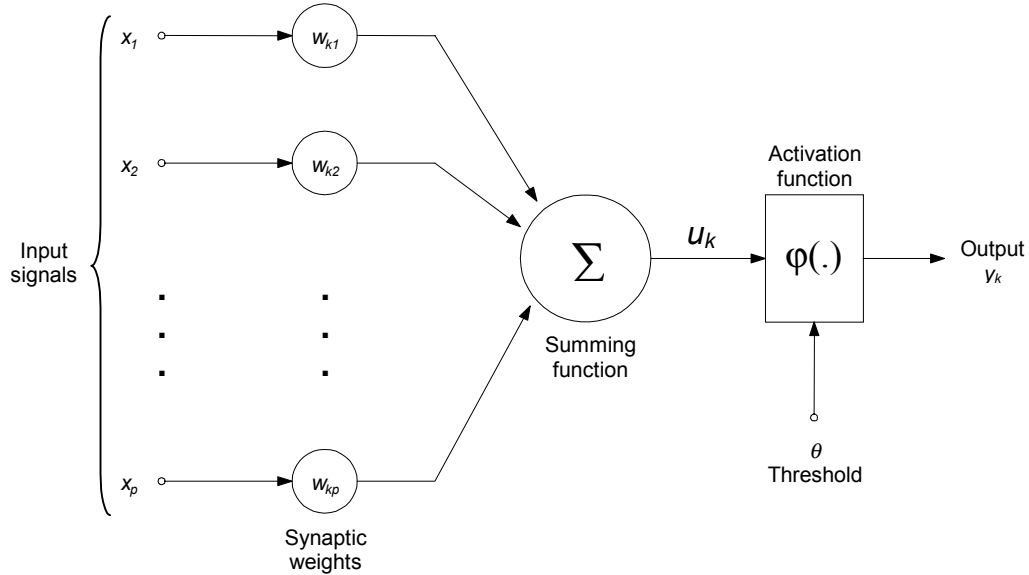
dan

$$y_k = \varphi(u_k - \theta_k)$$



dimana  $x_1, x_2, \dots, x_p$  adalah sinyal input;  $w_{k1}, w_{k2}, \dots, w_{kp}$  adalah bobot-bobot sinaptik dari *neuron k*;  $u_k$  adalah *linear combiner output*;  $\theta_k$  adalah threshold;  $\mu(.)$  adalah fungsi aktivasi; dan  $y_k$  adalah sinyal output dari *neuron*. Penggunaan threshold memberikan pengaruh adanya *affine transformation* terhadap output  $u_k$  dari linear combiner pada model gambar 1 sebagai berikut:

$$v_k = u_k - \theta_k$$



**Gambar 6.1** Model Matematis Nonlinier Dari Suatu *Neuron*[HAY94].

### 6.1.2 Fungsi Aktivasi

Terdapat berbagai macam fungsi aktivasi yang dapat digunakan tergantung karakteristik masalah yang akan diselesaikan. Tiga diantara fungsi aktivasi adalah sebagai berikut:

#### 1. Threshold Function

$$\phi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

#### 2. Piecewise-Linear Function

$$\varphi(v) = \begin{cases} 1 & v \geq \frac{1}{2} \\ v & \frac{1}{2} > v > -\frac{1}{2} \\ 0 & v \leq -\frac{1}{2} \end{cases}$$

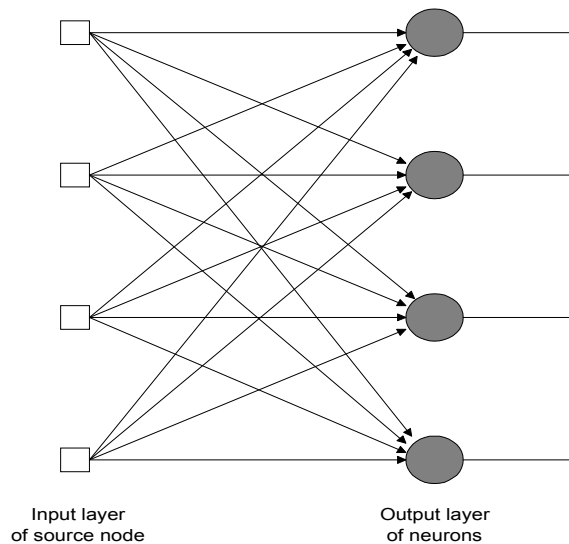
### 3. Sigmoid Function

$$\varphi(v) = \frac{1}{1 + \exp(-av)}$$

## 6.1.3 Arsitektur Jaringan

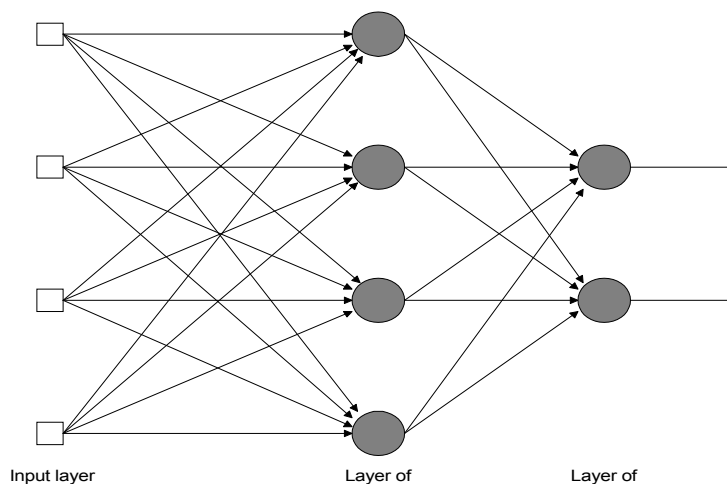
Pola dimana *neuron-neuron* pada JST disusun berhubungan erat dengan algoritma belajar yang digunakan untuk melatih jaringan.

### 1. Single-Layer Feedforward Networks

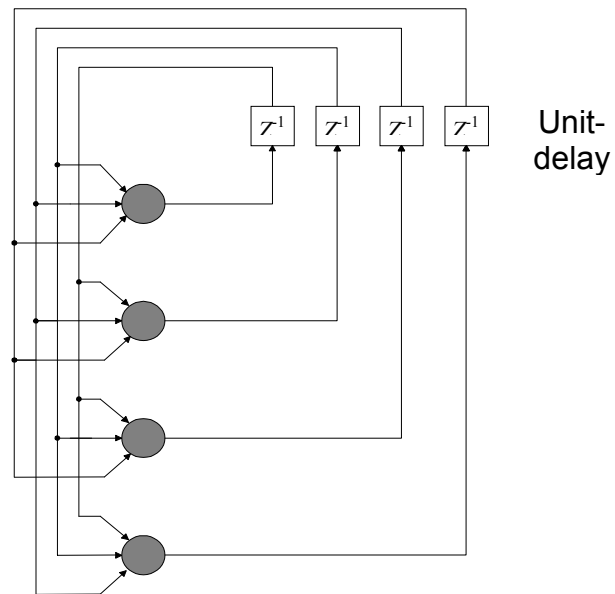


**Gambar 6.2** Feedforward Network dengan satu lapisan *neurons*

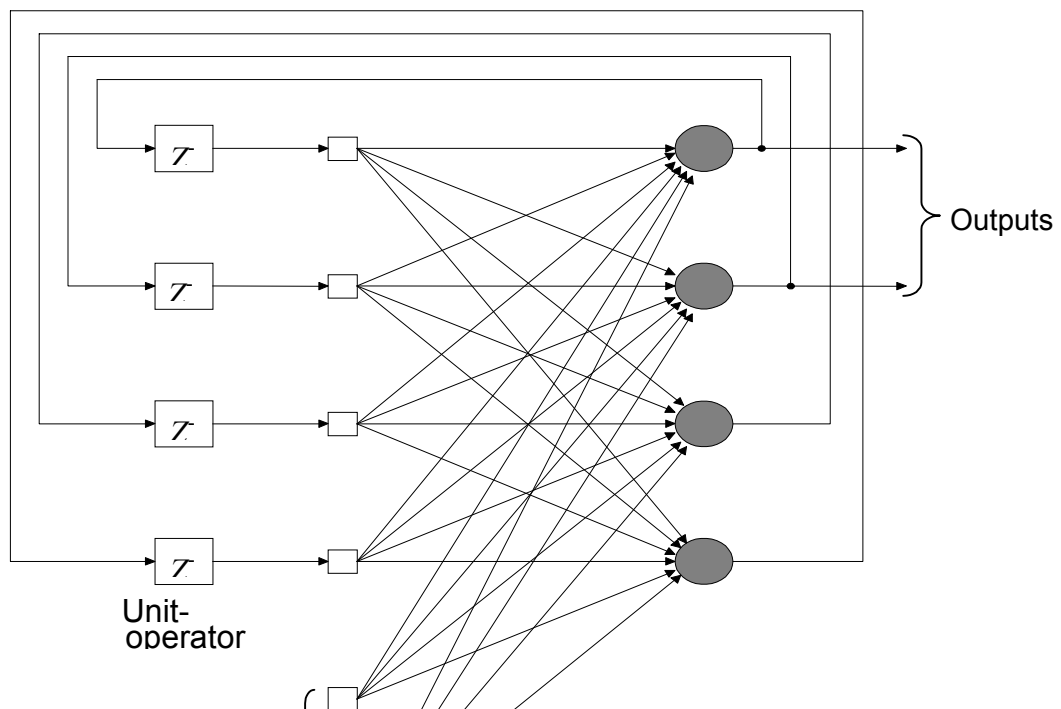
### 2. Multi-Layer Feedforward Networks



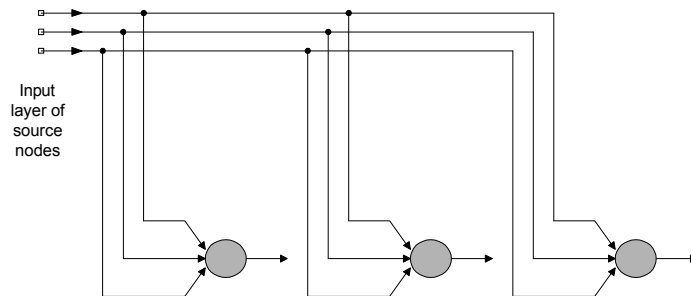
### 3. Recurrent Networks



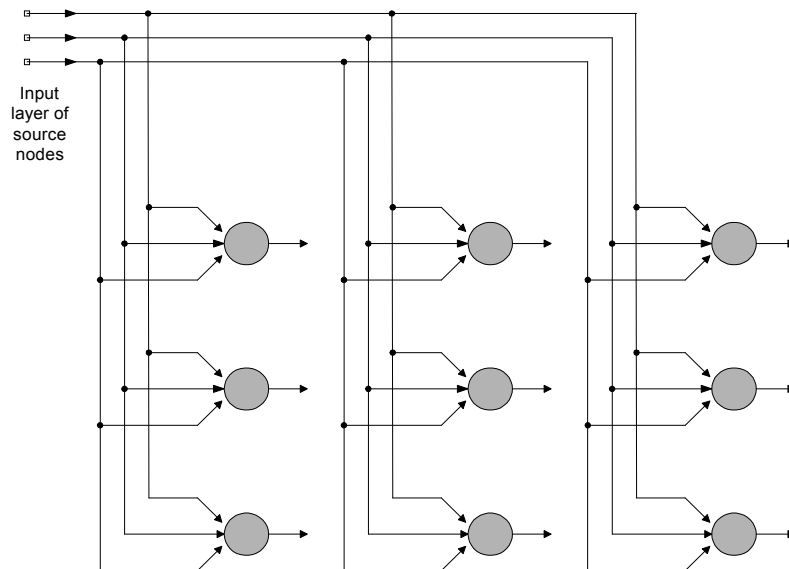
**Gambar 6.4** Recurrent network tanpa *self-feedback loop* dan tanpa *hidden neurons*.



#### 4. Lattice Structure



(a)



(b)

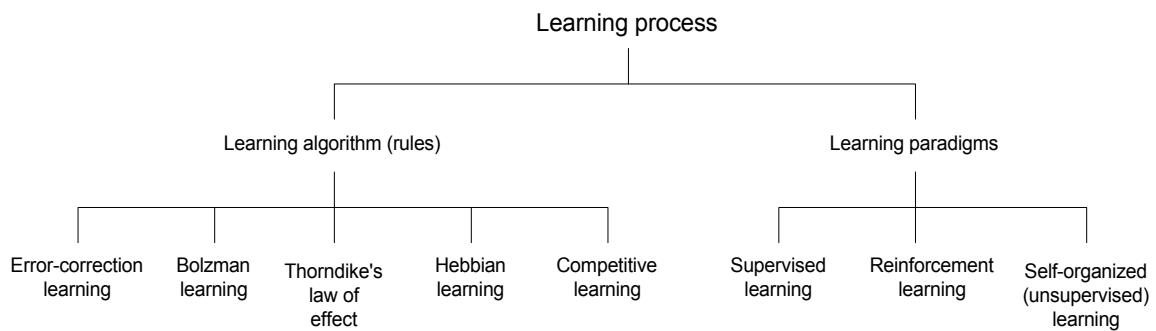
**Gambar 6.6** (a) *Lattice* satu dimensi dengan 3 *neurons*. (b) *Lattice* dua dimensi dengan 3 kali 3 *neurons*.

### 6.1.4 Proses Belajar

**Belajar:** suatu proses dimana parameter-parameter bebas JST diadaptasikan melalui suatu proses perangsangan berkelanjutan oleh lingkungan dimana jaringan berada.

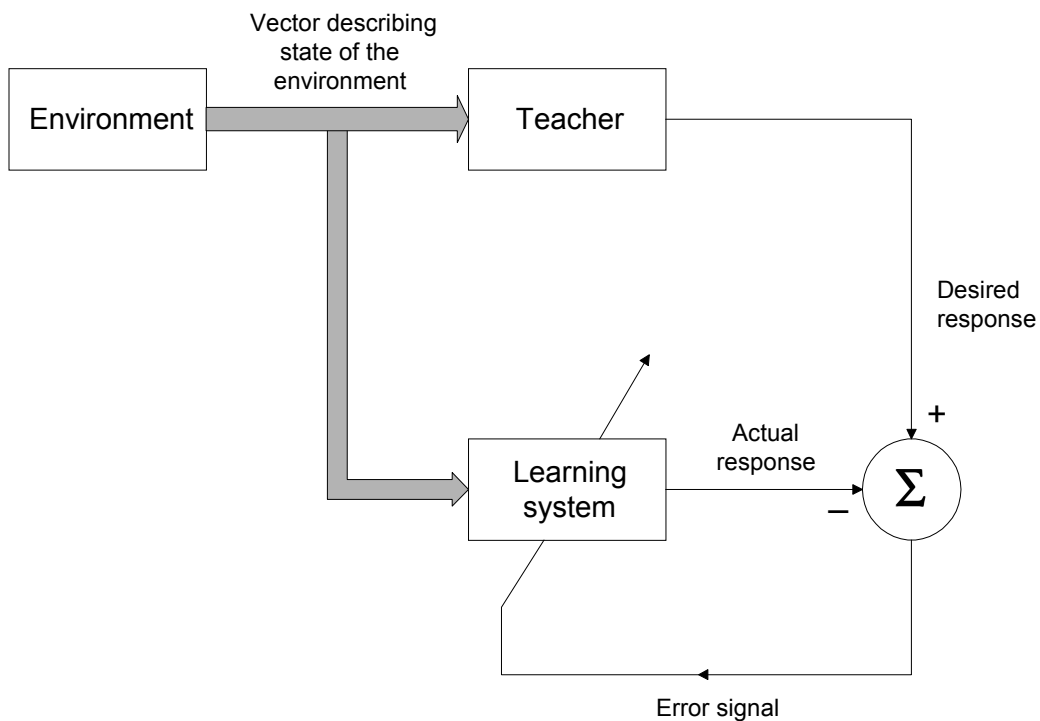
Definisi proses belajar ini menyebabkan urutan kejadian sebagai berikut:

1. JST dirangsang oleh lingkungan
2. JST mengubah dirinya sebagai hasil rangsangan ini.
3. JST memberikan respon dengan cara yang baru kepada lingkungan, disebabkan perubahan yang terjadi dalam struktur internalnya sendiri.



**Gambar 6.7** Taksonomi Proses Belajar.

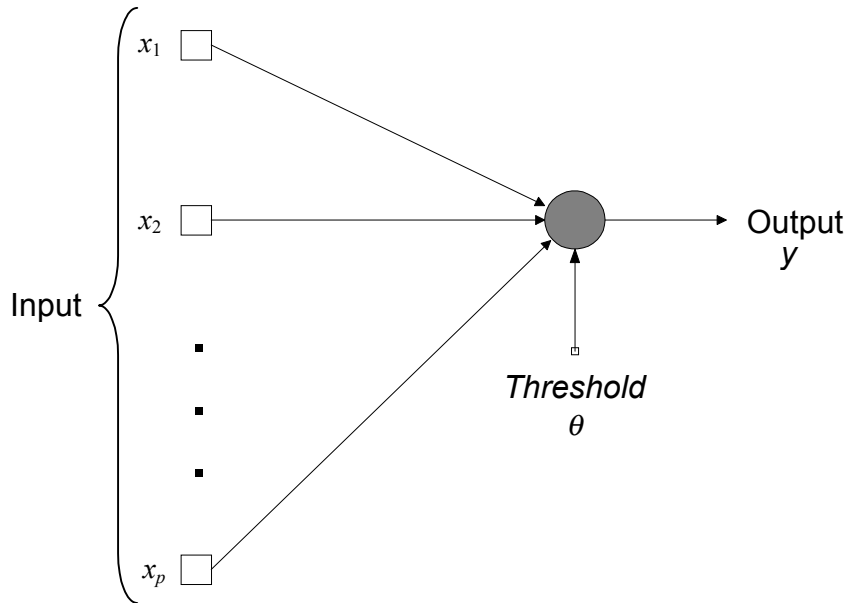
### Supervised *Learning* (Belajar Dengan Pengawasan)



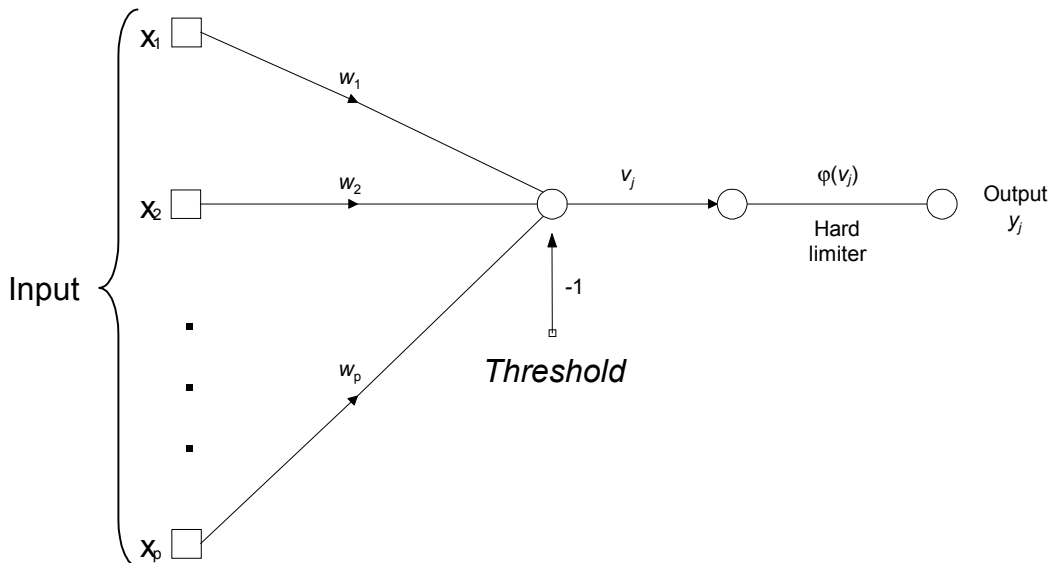


### 6.1.5 Perceptron

Perceptron adalah bentuk paling sederhana dari JST yang digunakan untuk pengklasifikasian jenis pola khusus yang biasa disebut *linearly separable* (pola-pola yang terletak pada sisi yang berlawanan pada suatu bidang).



**Gambar 6.8** Single-layer perceptron.

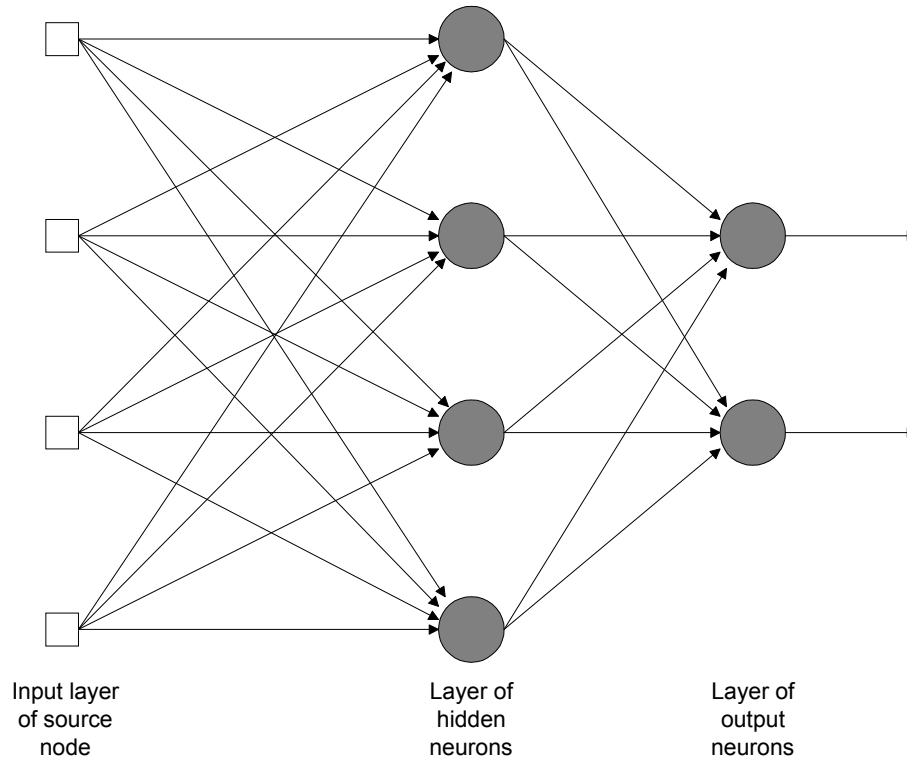


**Gambar 6.9** Graph aliran sinyal (*Signal-flow graph*) dari perceptron.

## 6.1.6 JST Dengan Supervised Learning

### 6.1.6.1 JST Propagasi Balik

JST Propagasi Balik merupakan model JST yang paling banyak digunakan dalam edukasi. Arsitektur dan proses belajar yang sederhana sangat memudahkan untuk dipelajari. Arsitektur JST Propagasi Balik diilustrasikan oleh gambar berikut:



**Gambar 6.10** ST Propagasi Balik dengan empat node pada input layer, satu hidden layer dengan empat node, dan dua node pada output layer.

#### Algoritma Pelatihan JST Propagasi Balik

1. Definisi masalah, misalkan matriks masukan ( $P$ ) dan matriks target ( $T$ ).
2. Inisialisasi, menentukan bentuk jaringan dan menetapkan nilai-nilai bobot sinaptik ( $W1$  dan  $W2$ ) dan *learning rate* ( $lr$ ).
3. Pelatihan Jaringan

- **Perhitungan Maju**

Keluaran dari *hidden layer* dan *output layer*:

$$A1 = \frac{1}{1 + e^{-\sum_{i=1}^m P_i * W1_{ji}}} \quad (1)$$



$$A2 = \frac{1}{1 + e^{-\sum_{j=1}^n A1_j * W2_{kj}}} \quad (2)$$

Galat ( $E$ ) dan *Sum Square Error* ( $SSE$ ) didefinisikan sebagai berikut:

$$E = T - A2 \quad (3)$$

$$SSE = \sum E^2 \quad (4)$$

- **Perhitungan Balik (Perbaikan Bobot-bobot Sinaptik)**

$$D2 = A2 * (1 - A2) * E \quad (5)$$

$$dW2 = dW2 + (lr * D2 * A1) \quad (6)$$

$$D1 = A1 * (1 - A1) * (W2 * D2) \quad (7)$$

$$dW1 = dW1 + (lr * D1 * P) \quad (8)$$

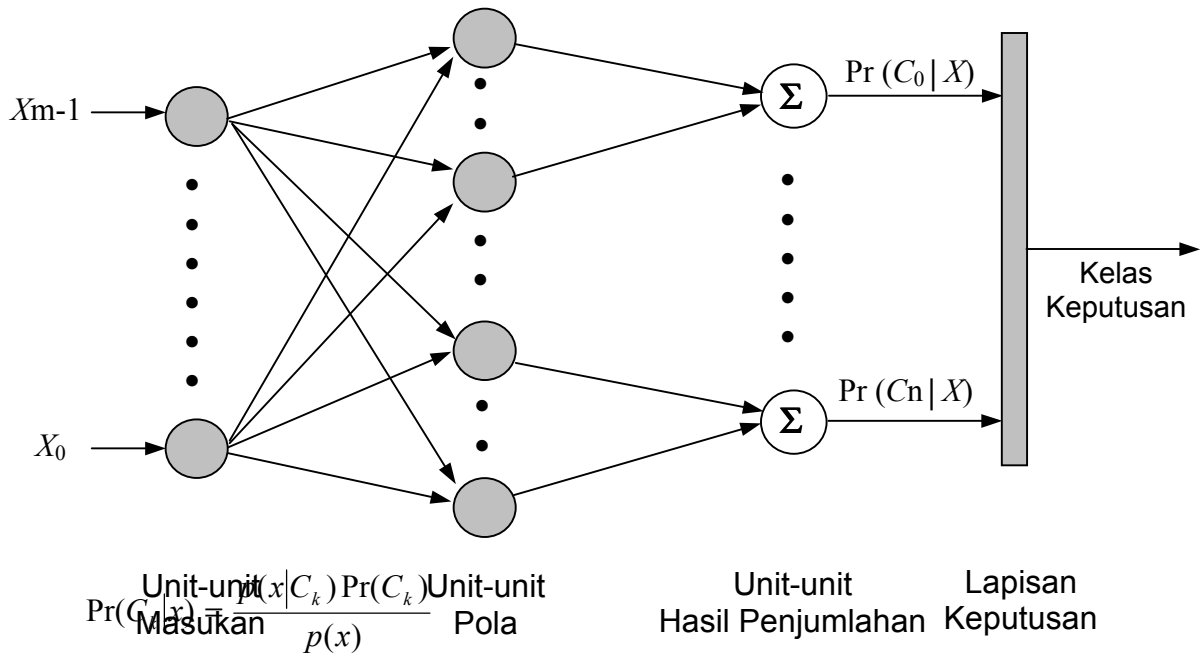
$$W2 = W2 + dW2 \quad (9)$$

$$W1 = W1 + dW1 \quad (10)$$

4. Langkah-langkah diatas adalah untuk satu kali siklus pelatihan (satu *epoch*). Proses pelatihan diulang sampai jumlah *epoch* tertentu atau telah tercapai *SSE* yang diinginkan.
5. Hasil akhir pelatihan jaringan adalah didapatkannya bobot-bobot  $W1$  dan  $W2$  yang kemudian disimpan untuk pengujian jaringan.

Pada prakteknya, perancangan arsitektur JST Propagasi Balik sangat tergantung pada masalah yang akan diselesaikan. Untuk himpunan masukan berdimensi besar atau jumlah kelas keluaran yang diinginkan besar, maka diperlukan jumlah *node* pada *hidden layer* yang lebih banyak. Atau diperlukan lebih dari satu *hidden layer*, tetapi tentu saja ada batas optimumnya untuk kedua parameter tersebut.

### 6.1.6.2 JST Probabilistik



$d(x) = C_k$  jika  $p(x | C_k) \Pr(C_k) > p(x | C_j) \Pr(C_j)$  untuk semua  $j \neq k$ .

$$p(x | C_k) \approx \frac{1}{(2\pi)^{m/2} \sigma_k^m |C_k|} \sum_{\rho_i \in C_k} \exp \left[ -\|x - w_i\|^2 / (2\sigma_k^2) \right]$$

(\* Tahap Pertama \*)

**for** setiap pola  $\rho_i$

**begin**

$w_i = \rho_i$ ;

Bentuk unit pola dengan masukan vektor bobot  $w_i$ ;

Hubungkan unit pola pada unit penjumlah untuk masing-masing kelas;

**end**;

Tentukan konstanta  $|C_k|$  untuk setiap unit penjumlah;

(\* Tahap ke dua \*)

**for** setiap pola  $\rho_i$

**begin**

$k = \text{kelas } \rho_i$ ;

Cari jarak,  $d_i$ , dengan pola terdekat pada kelas  $k$ ;

$d_{tot}[k] = d_{tot}[k] + d_i$ ;

**end**;

Algoritma Pelatihan JST Probabilistik.

Pemilihan konstanta  $g$ , agar jaringan memiliki akurasi pengklasifikasian yang tinggi diperoleh melalui percobaan, karena konstanta  $g$  dipengaruhi oleh jumlah kelas, dimensi pola latih, dan jumlah anggota himpunan pelatihan.

Perbedaan mendasar antara JST Propagasi Balik dengan JST Probabilistik adalah:

<b>JST Propagasi Balik</b>	<b>I. JST Probabilistik</b>
Jumlah <i>neuron</i> <b>tetap</b> , tetapi bobot-bobot sinaptiknya <b>berubah</b> .	Jumlah <i>neuron</i> <b>bertambah</b> sebanding dengan banyaknya vektor input, tetapi bobot-bobot sinaptiknya <b>tetap</b> .
Proses pelatihan memerlukan waktu yang lama saat iterasi pengubahan bobot sampai mencapai <i>steady state</i> .	JST Probabilistik memerlukan waktu yang sangat singkat, karena hanya dilakukan untuk satu tahap pelatihan saja.
Memerlukan memori yang kecil karena hanya menyimpan bobot-bobot sinaptik hasil pelatihan	Memerlukan memori yang besar sebanding dengan vektor pola latihnya.

## 6.2 Algoritma Genetika (Genetic Algorithm)

*Genetic Algorithm* (GA) adalah algoritma pencarian yang didasarkan pada mekanisme **seleksi alamiah** dan **genetika alamiah** [GOL89].

GA dimulai dengan serangkaian solusi awal (kromosom), yang disebut populasi. Populasi ini akan ber-evolusi menjadi populasi yang berbeda melalui serangkaian iterasi. Pada akhir iterasi, GA mengembalikan anggota populasi yang terbaik sebagai solusi untuk problem tersebut. Pada setiap iterasi (generasi), proses evolusi yang terjadi adalah sebagai berikut:

1. Dua anggota populasi (*parent*) dipilih berdasarkan pada suatu distribusi populasi. Kedua anggota ini kemudian dikombinasikan atau dikawinkan melalui operator **crossover** (pindah silang) untuk menghasilkan anak (*offspring*).
2. Dengan probabilitas yang rendah, anak ini akan mengalami perubahan oleh operator **mutasi**.
3. Apabila anak sesuai untuk populasi tersebut, suatu skema penggantian (*replacement scheme*) diterapkan untuk memilih anggota populasi yang akan digantikan oleh anak.
4. Proses ini terus berulang sampai dicapai kondisi tertentu, misalnya sampai jumlah iterasi tertentu.

**Kromosom** : serangkaian nilai gen.

**Gen** : mempunyai nilai berupa suatu alfabet  $S$ , misalnya  $\{0,1\}$ .

**Solusi** : direpresentasikan oleh suatu kromosom.

**Skema** : pola gen yang menggambarkan suatu subset dari string yang memiliki kesamaan pada posisi gen tertentu.

**Orde** : banyaknya simbol khusus pada suatu skema.

**Defining length** : jarak antara simbol khusus paling kiri dan paling kanan.

Misal:  $10^*$  adalah skema yang mewakili himpunan kromosom yang mempunyai nilai 1 pada bit pertama dan 0 pada bit kedua. Skema tersebut memiliki orde 2 dan *defining length*-nya adalah 1.

Suatu kromosom dengan panjang  $n$  dapat juga dipandang sebagai *instance* (anggota) dari  $2^n$  skema. Misal, kromosom 101 adalah suatu instance dari 8 skema, yaitu  $***$ ,  $1**$ ,  $*0*$ ,  $**1$ ,  $10^*$ ,  $1^*1$ ,  $*01$ , dan 101. Algoritma Genetika tidak secara eksplisit dalam menangani skema, namun konsep skema sangat penting dalam analisis perilaku Algoritma Genetika dalam pemecahan suatu masalah.

### 6.2.1 Sifat-Sifat Khusus GA [GOL89]:

1. GA bekerja dengan sebuah pengkodean dari kumpulan parameternya sendiri.
2. GA mencari dari sebuah populasi titik-titik, bukan dari sebuah titik tunggal.
3. GA hanya menggunakan informasi hasil (fungsi objektif), bukan menurunkan dari bantuan pengetahuan lain.
4. GA menggunakan kemungkinan aturan transisi, bukan aturan-aturan deterministik.

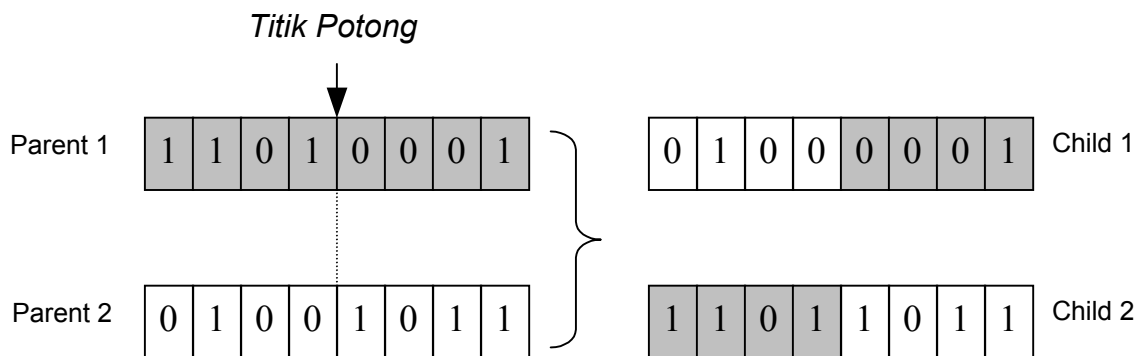
## 6.2.2 Operator-operator GA

### 6.2.2.1 Reproduksi

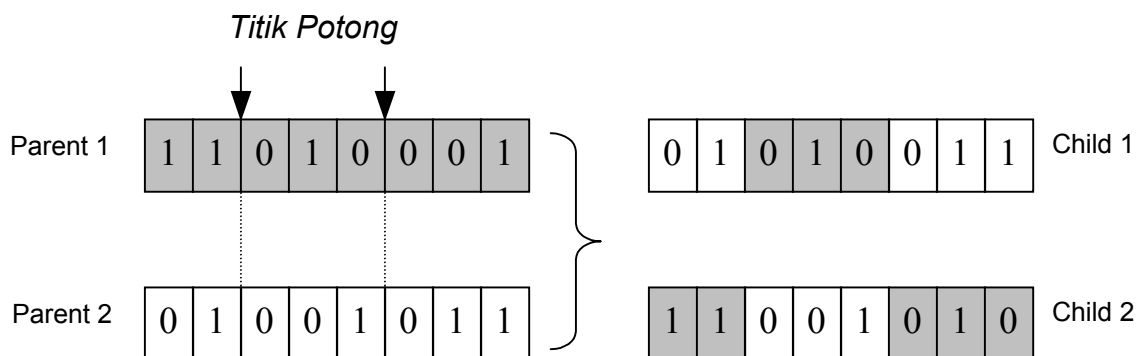
- Reproduksi merupakan sebuah proses dimana sebuah string individu disalin kembali menjadi individu-individu baru yang siap menjadi orang tua.
- Mekanisme reproduksi menggunakan **cross over** dan **mutasi**. Pemilihan individu orang tua bergantung pada nilai fungsi **fitness** dari individu-individu tersebut di dalam populasinya.
- Kegunaan dari pemilihan orang tua dalam GA adalah untuk memberikan kesempatan bereproduksi terhadap seluruh anggota populasi yang terbaik. Ada banyak cara untuk menyelenggarakan pemilihan orang tua, salah satunya adalah pemilihan orang tua **Roulette Wheel**.

### 6.2.2.2 Pindah Silang (cross over)

- **Pindah Silang Satu Titik**



- **Pindah Silang Dua Titik**



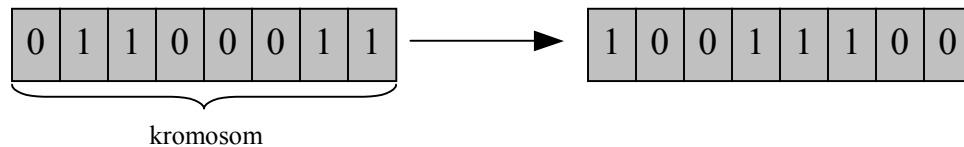
Operator pindah silang dapat dilakukan dengan lebih dari dua titik. Tetapi jumlah titik potong yang semakin banyak akan memperendah kualitas solusi yang didapatkan. Hal ini disebabkan operasi pindah silang terlalu sering merusak kromosom yang baik.

### 6.2.2.3 Mutasi

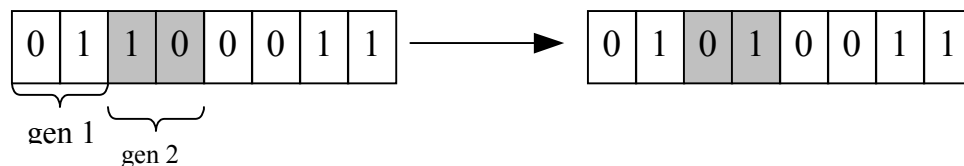
Mutasi diperlukan untuk mengembalikan informasi bit yang hilang akibat *cross over*. Mutasi diterapkan dengan probabilitas sangat kecil. Jika mutasi dilakukan terlalu sering, maka akan menghasilkan individu yang lemah karena konfigurasi bit pada kromosom yang unggul akan dirusak.

Berdasarkan bagian yang termutasi, proses mutasi dapat dibedakan atas tiga bagian:

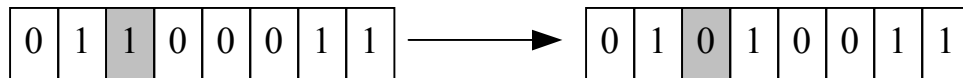
- Tingkat kromosom: semua gen dalam kromosom berubah.



- Tingkat gen: semua bit dalam satu gen akan berubah. Misal gen 2 yang mengalami mutasi.



- Tingkat bit: hanya satu bit yang berubah.



### 6.2.3 Algoritma Genetika Sederhana

Random populasi awal

**Repeat**

Pilih parent1 dan parent2 dari populasi;

Offspring = crossover(parent1,parent2);

**If** Mutable **then** Mutation(offspring);

**If** Suited(offspring) **then**

Replace (population,offspring);

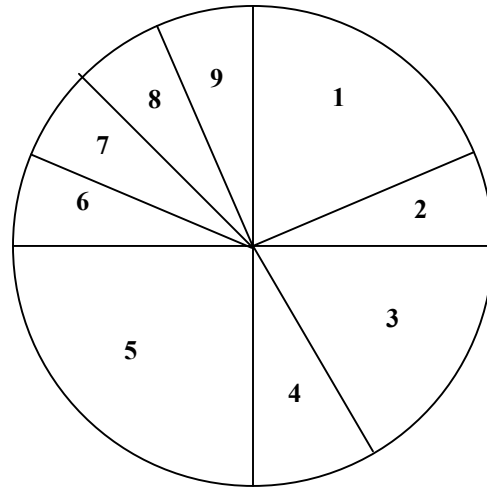
-- . . . . .

**Gambar 6.11** Struktur Umum Steady State GA.

#### 6.2.4 Skema Pemilihan Orang Tua

Pemilihan orang tua didasarkan pada *proportional selection scheme*, yaitu: Setiap kromosom dipilih sebagai *parent* berdasarkan suatu probabilitas yang proporsional terhadap nilai *fitness* masing-masing. Salah satu teknik yang sering digunakan adalah *Roulette Wheel*.

String	Nilai Fitness
S1	0.6
S2	0.2
S3	0.6
S4	0.2
S5	0.8
S6	0.2
S7	0.2
S8	0.2
S9	0.2
Jumlah	3.2



**Catatan:** Penentuan fungsi fitness bergantung pada masalah yang akan dioptimasi.

#### 6.2.5 Skema Penggantian Individu dalam Populasi

1. Selalu mengganti anggota populasi yang memiliki nilai fitness terkecil [GEN97].
2. Membandingkan anak dengan kedua orang tua (*preselection*). Apabila anak memiliki nilai fitness yang lebih baik daripada salah satu atau kedua orang tua, anak menggantikan orang tua yang memiliki nilai fitness terendah sebagai anggota populasi. Skema ini dapat menjaga keanekaragaman yang lebih baik dibanding skema sebelumnya. Dengan mengganti orang tua yang memiliki *Hamming Distance* (beda bit) yang lebih sedikit, penggantian tersebut dapat diharapkan tidak cepat menghilangkan keragaman.

#### 6.2.6 Kriteria Penghentian

1. Mengambil suatu nilai sebagai batas iterasi. Apabila batas iterasi tersebut dicapai, iterasi dihentikan dan solusi terbaik pada populasi dilaporkan sebagai solusi terbaik pencarian.
2. Menghitung kegagalan penggantian anggota populasi yang terjadi secara berurutan sampai jumlah tertentu. Hal ini berarti bahwa anggota-anggota populasi adalah kemungkinan-kemungkinan solusi yang terbaik dari semua kemungkinan solusi pada ruang pencarian yang ada. Sehingga iterasi lanjutan yang dilakukan tidak akan banyak membantu lebih jauh untuk mendapatkan solusi yang lebih baik.

### 6.2.7 Kelemahan Algoritma genetika [GOL89]:

- Lebih lama menemukan solusi karena memproses beberapa solusi sekaligus.
- GA dapat mengalami konvergensi yang terlalu cepat sehingga tidak ada jaminan solusi yang didapatkan adalah solusi optimal.

#### **Hipotesa *Building Blocks*:**

Operasi terhadap suatu kromosom dapat juga dipandang sebagai operasi implisit terhadap  $2^n$  skema. Sehingga dapat dikatakan bahwa selain memproses kromosom secara eksplisit, GA juga secara implisit memproses skema. Ini disebut sebagai *implicit parallelism* atau *intrinsic parallelism*.

#### ***Building Block Hypothesis* by Goldberg:**

GA secara implisit akan memilih skema berorde rendah dan memiliki nilai fitness tinggi, kemudian menggenerate skema-skema dengan orde yang lebih tinggi dan nilai fitness tinggi dari skema-skema berorde rendah tersebut melalui operasi pindah silang. Proses inilah yang menjadi kelebihan Algoritma Genetika [GOL89].

Agar skema-skema bernilai fitness tinggi dapat dikombinasikan satu sama lain menjadi skema berorde lebih tinggi, skema tersebut harus tetap terjaga selama proses pindah silang (tidak terpotong). Sehingga dapat dikatakan bahwa semakin pendek *defining length* dari suatu skema, kemungkinan skema tersebut terpotong oleh operasi pindah silang semakin kecil, demikian pula sebaliknya.



## 6.3 Fuzzy Set, Fuzzy Logic, dan Fuzzy System

### 6.3.1 Fuzzy Set dan Fuzzy Logic

**Crisp Set:** himpunan yang membedakan anggotanya dan non-anggotanya dengan batasan yang pasti.

**Misal:**  $A = \{x \mid x \text{ integer, } x > 2\}$   
Anggota  $A = \{3, 4, 5, \dots\}$

**Linguistic Variable:** suatu variable yang tidak dapat dipasangkan langsung dengan nilai dalam suatu range, karena range dapat berubah bergantung keadaan.

**Misal:** 1. “Hangat” =  $[1^0\text{C} - 5^0\text{C}]$  untuk winter  $[-5^0\text{C} - 5^0\text{C}]$   
2. “Hangat” =  $[14^0\text{C} - 16^0\text{C}]$  untuk Spring  $[12^0\text{C} - 18^0\text{C}]$

#### **Membership Function (Fungsi Keanggotaan)**

$\mu_A(x) = [0,1]$ , dimana  $x$  adalah suatu crisp set.

$\mu_A$  = Fungsi yang menyatakan keanggotaan suatu elemen dalam *fuzzy set*.

Fuzzy set = subset dari  $x$

Penentuan  $\mu$  sulit dilakukan, maka dibantu JST.

**Fuzzy Set :** anggotanya ditentukan oleh fungsi keanggotaan.

$x = \{5, 10, 20, 30, 40, 50, 60, 70, 80\}$  , merupakan crisp set umur dalam tahun.

Fuzzy set “balita”, “dewasa”, “muda”, dan “tua” adalah subset dari  $x$ .

Elemen	Balita	Dewasa	Muda	Tua
5	0	0	1	0
10	0	0	1	0
20	0	0.8	0.8	0.1
30	0	1	0.5	0.2
40	0	1	0.2	0.4
50	0	1	0.1	0.6
60	0	1	0	0.8
70	0	1	0	1
80	0	1	0	1

Balita =  $\{\}$

Dewasa =  $\{20, 30, 40, 50, 60, 70, 80\}$

$\mu_{Dewasa} = \{0.8, 1, 1, 1, 1, 1, 1\}$

Muda =  $\{5, 10, 20, 30, 40, 50\}$

$\mu_{Muda} = \{1, 1, 0.8, 0.5, 0.2, 0.1\}$

Tua =  $\{20, 30, 40, 50, 60, 70, 80\}$

$\mu_{Tua} = \{0.1, 0.2, 0.4, 0.6, 0.8, 1, 1\}$



## Cara menyatakan Fuzzy Set

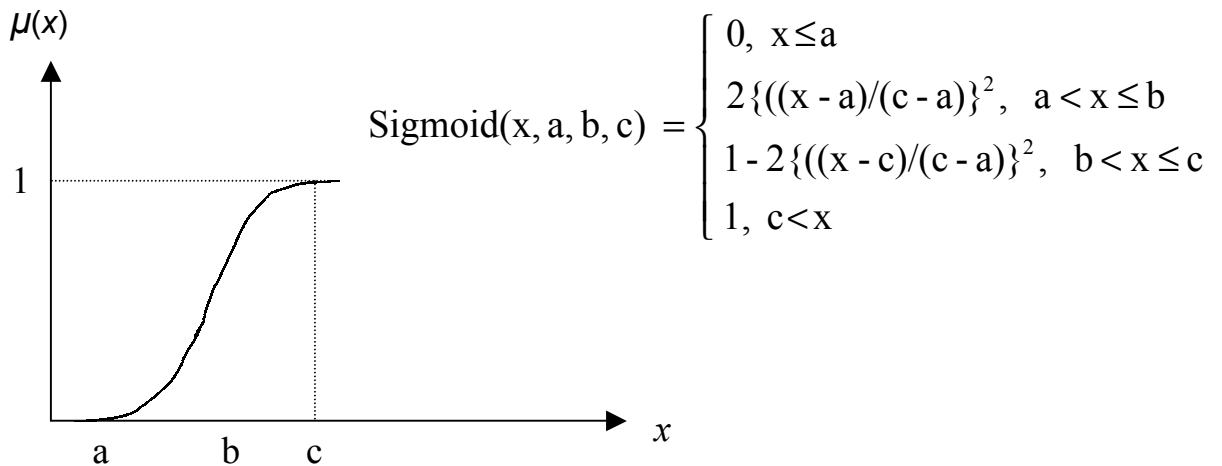
$$\begin{aligned} \text{Tua} &= \sum \frac{\mu_{\text{Tua}}}{e} \\ &= 0.1/20 + 0.2/30 + 0.4/40 + 0.6/50 + 0.8/60 + 1/70 + 1/80 \end{aligned}$$

atau

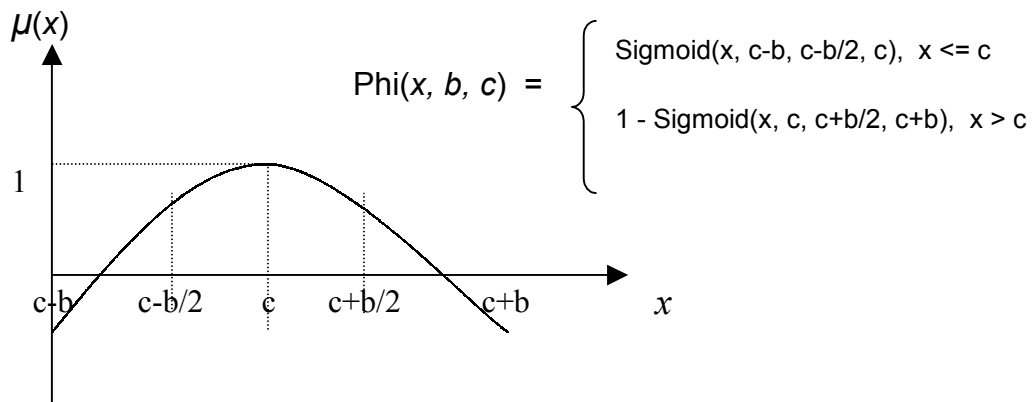
$$\text{Tua} = \{0.1/20, 0.2/30, 0.4/40, 0.6/50, 0.8/60, 1/70, 1/80\}$$

## Jenis-jenis Fungsi Keanggotaan:

### 1. Fungsi Sigmoid

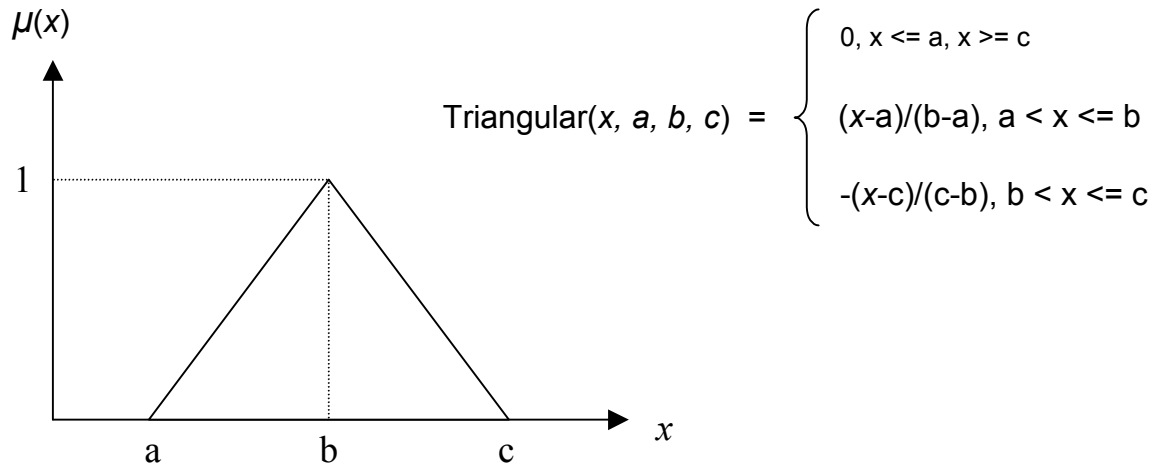


### 2. Fungsi Phi

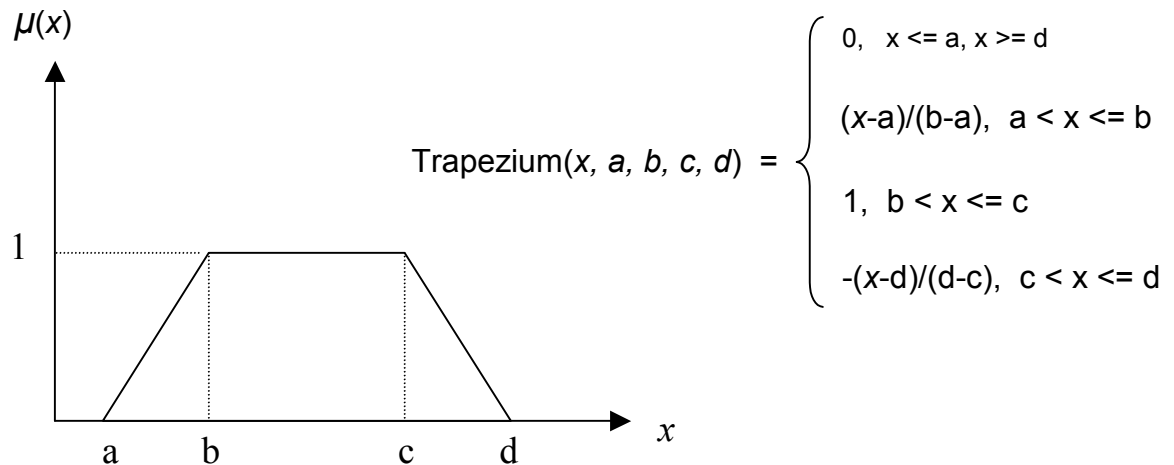




### 3. Fungsi Segitiga



### 4. Fungsi Trapezium



### Operasi Dasar Fuzzy

#### 1. Complement : $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$

$\mu_A(x)$  Tua = {0/5, 0/10, 0.1/20, 0.2/30, 0.4/40, 0.6/50, 0.8/60, 1/70, 1/80}

$\mu_{\bar{A}}(x)$  Tidak Tua = {1/5, 1/10, 0.9/20, 0.8/30, 0.6/40, 0.4/50, 0.2/60, 0/70, 0/80}

#### 2. Union (Disjunction) : $\mu_{A \cup B}(x) = \max[\mu_A(x), \mu_B(x)]$

$\mu_A(x)$  Muda = {1/5, 1/10, 0.8/20, 0.5/30, 0.2/40, 0.1/50, 0/60, 0/70, 0/80}

$\mu_A(x)$  Tua = {0/5, 0/10, 0.1/20, 0.2/30, 0.4/40, 0.6/50, 0.8/60, 1/70, 1/80}

$$\mu_A(x) \text{ Muda } \cup \text{ Tua} = \{1/5, 1/10, 0.8/20, 0.5/30, 0.4/40, 0.6/50, 0.8/60, 1/70, 1/80\}$$

3. **Intersection (Conjunction):**  $\mu_{A \cap B}(x) = \min[\mu_A(x), \mu_B(x)]$

$$\mu_A(x) \text{ Muda } \cap \text{ Tua} = \{0/5, 0/10, 0.1/20, 0.2/30, 0.2/40, 0.1/50, 0/60, 0/70, 0/80\}$$

**Hedge:** Modifikasi derajat keanggotaan (bersifat heuristik).

1. Penguat : sangat
2. Pelemah : agak
3. Komplemen : tidak
4. Penaksir : sekitar
5. Pembatas : di atas
6. Kuantifier : hampir semua

**Note:** Kata dalam Kalimat Fuzzy dibaca dari kiri ke kanan.

$\neg(\text{SANGAT}(\text{BERAT}))$  tidak sama dengan  $\text{SANGAT}(\neg\text{BERAT})$

### **Fuzzy Quantifier (Pengukur Kesamaran)**

Menggambarkan pendekatan pengukuran Fuzzy Logic.

Fuzzy Quantifier digunakan untuk **Disposisi** (pernyataan yang mengandung makna “Implied” dalam Fuzzy)

Contoh :

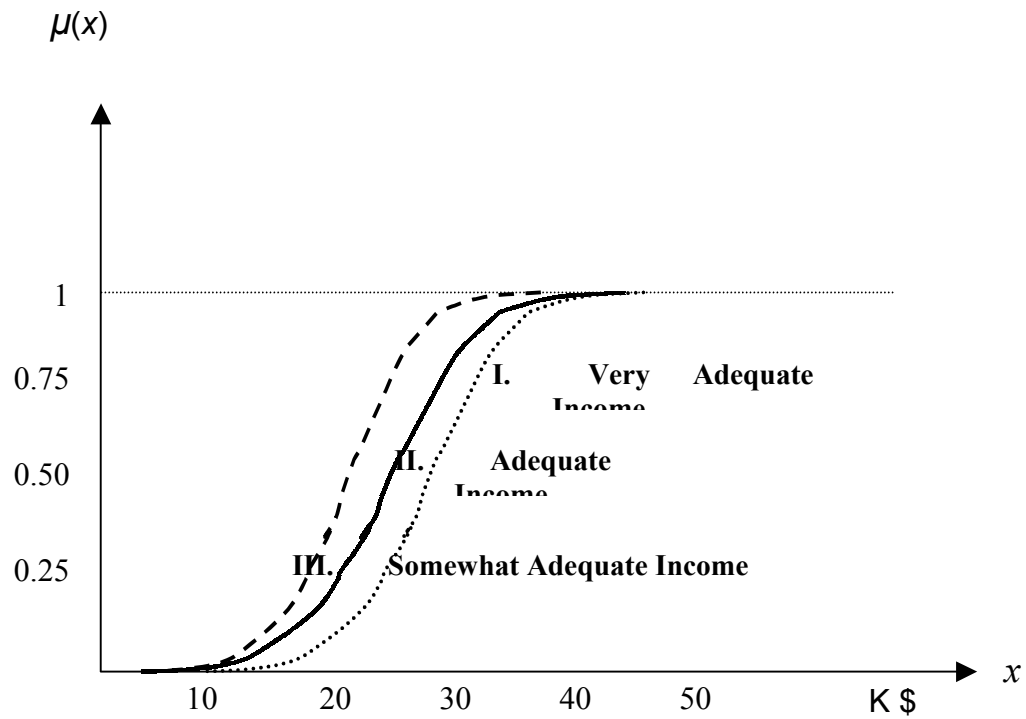
1. **Most** horses have tails.
2. **Very** old people.
3. **Approximately** adequate income.

**Concentrate:** Pertajaman (Quite, Very)

$$\mu^2 : \text{power } 2$$

**Dilate:** Perluasan (Somewhat)

$$\sqrt{\mu} : \text{power } \frac{1}{2}$$



**Adequate Income** =  $\{0/10, 0.125/20, 0.5/30, 0.75/35, 1/40, 1/50\}$

**Not Adequate Income** =  $\{1/10, 0.875/20, 0.5/30, 0.25/35, 0/40, 0/50\}$

**Very Adequate Income** =  $\{0/10, 0.16/20, 0.25/30, 0.56/35, 1/40, 1/50\}$

**Somewhat Adequate Income** =  $\{0/10, 0.35/20, 0.71/30, 0.87/35, 1/40, 1/50\}$

### ***Compositional Operator: o***

Approximately adequate income (y) = Adequate income (x) **o** Approximately equal income (y)

$$[x_1 \ x_2 \ \dots \ x_n] \begin{bmatrix} y_{11} & y_{12} & \dots & y_{1m} \\ y_{21} & y_{22} & \dots & y_{2m} \\ y_{n1} & y_{n2} & \dots & y_{nm} \end{bmatrix}$$

Approximately equal income  $(x,y) =$

$$\begin{array}{c}
 20k \\
 30k \\
 35k \\
 40k
 \end{array}
 \begin{array}{c}
 \begin{array}{c} 20k \quad 30k \quad 35k \quad 40k \end{array} \\
 \left[ \begin{array}{cccc}
 1 & 0.5 & 0 & 0 \\
 0.5 & 1 & 0.7 & 0.1 \\
 0 & 0.7 & 1 & 0.8 \\
 0 & 0.1 & 0.8 & 1
 \end{array} \right]
 \end{array}$$

Approximately Adequate income =

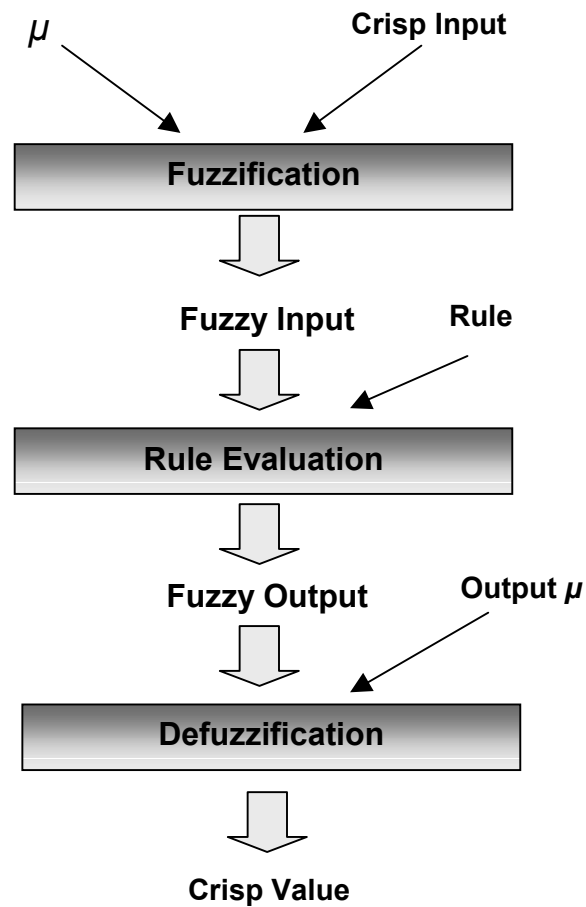
$$\begin{array}{c}
 [0.125 \quad 0.5 \quad 0.75 \quad 1] \\
 \left[ \begin{array}{cccc}
 1 & 0.5 & 0 & 0 \\
 0.5 & 1 & 0.7 & 0.1 \\
 0 & 0.7 & 1 & 0.8 \\
 0 & 0.1 & 0.8 & 1
 \end{array} \right]
 \end{array}$$

$$= \{0.5, 0.7, 0.8, 1\}$$

Jika user penghasilannya \$ 35,000, maka approximately adequate sama dengan 0.8.



### 6.3.2 Fuzzy-Rule-Based Systems



**Studi Kasus:** *Sprinkler Control System as a Fuzzy-Rule-Based System*

#### Antecedent 1 (Temperature)

Antecedent 2 (Moisture)		Cold	Cool	Normal	Warm	Hot
	Dry	L	L	L	L	L
	Moist	L	M	M	M	M
	Wet	S	S	S	S	S

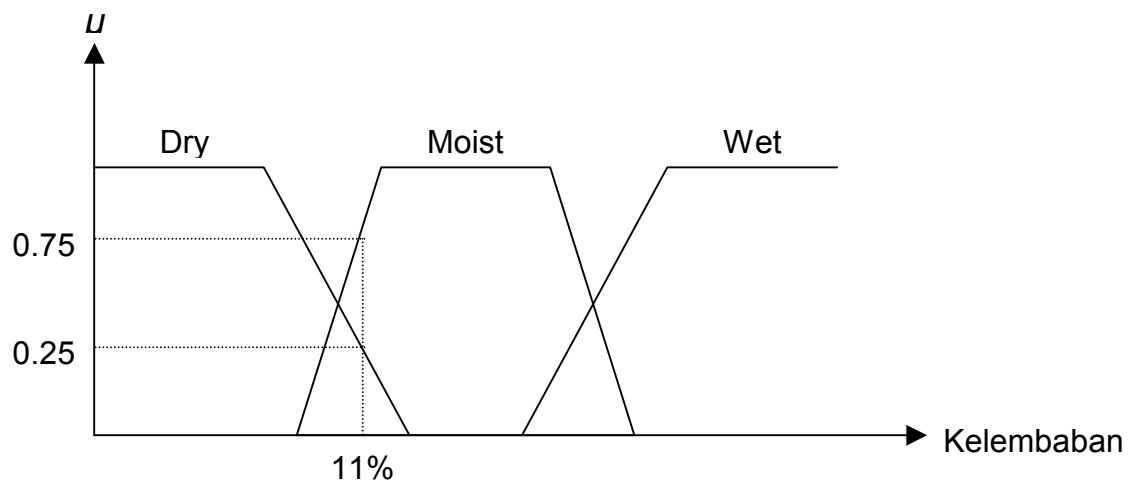
**Note:** L = Long, M = Medium, S = Short

### 6.3.2.1 Proses Fuzzyfication

- Membership Function untuk **Temperature** adalah **fungsi Phi**.
- Temperatur Udara 92°F menjadi Warm = 0.2  
Hot = 0.46

Label	Domain (°F)
Cold	30 – 47
Cool	40 – 70
Normal	60 – 84
Warm	75 – 98
Hot	90 – 110

- Membership Function untuk **Moisture** adalah **Trapezium**.
- Kelembaban 11% menjadi Dry = 0.25  
Moist = 0.75  
Wet = 0



### 6.3.2.2 Rule Evaluation

Fuzzy rules untuk Sprinkler (yang diukur **Moisture** dan **Temperature**)

- IF** Temperature is Hot (0.46)  $\wedge$  Soil is Dry (0.25)  
**THEN** watering duration is Long
- IF** Temperature is Hot (0.46)  $\wedge$  Soil is Moist (0.75)  
**THEN** watering duration is Medium
- IF** Temperature is Warm (0.2)  $\wedge$  Soil is Moist (0.75)  
**THEN** watering duration is Medium
- IF** Temperature is Warm (0.2)  $\wedge$  Soil is Dry (0.25)  
**THEN** watering duration is Long

**Fuzzy Output:** Water Duration is **Long (0.25)** dan **Medium (0.46)**

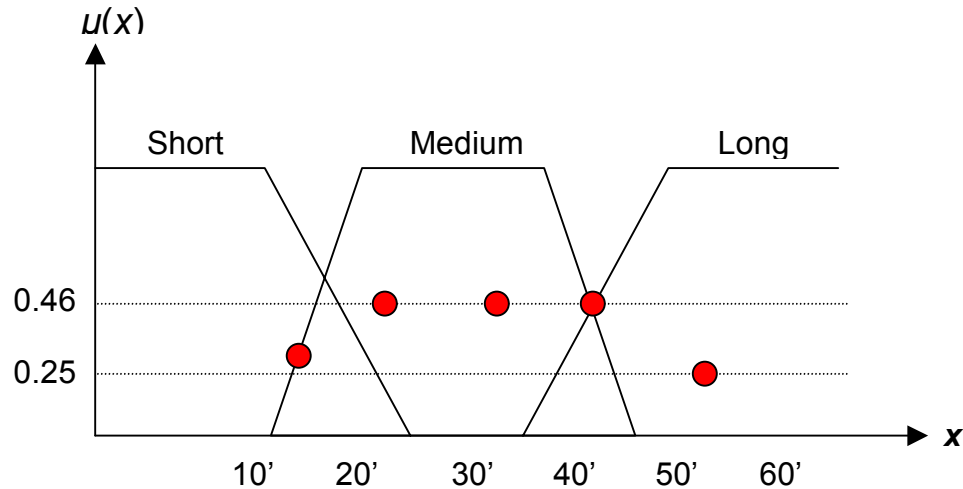
### 6.3.2.3 Proses Defuzzification

Metode yang digunakan: **Center of Gravity (CoG)** atau **Centroid Method**.

Fuzzy Output yang berupa  $\mu$  dipotongkan ke grafik = **Lamda Cut ( $\lambda$ -Cut)**

**Lamda Cut:** membatasi maximum kebenaran dari himpunan fuzzy /  $\mu$ .

Untuk setiap  $\mu_A$  berlaku  $\mu(x) = \min_A(\mu(x), \lambda\text{-Cut}_A)$



$$\text{CoG} = \frac{\int_a^b \mu(x) x \, dx}{\int_a^b \mu(x) \, dx}$$

$$= \frac{\sum_{x=a}^b \mu(x) x}{\sum_{x=a}^b \mu(x)}$$

**Center of Gravity (CoG)**

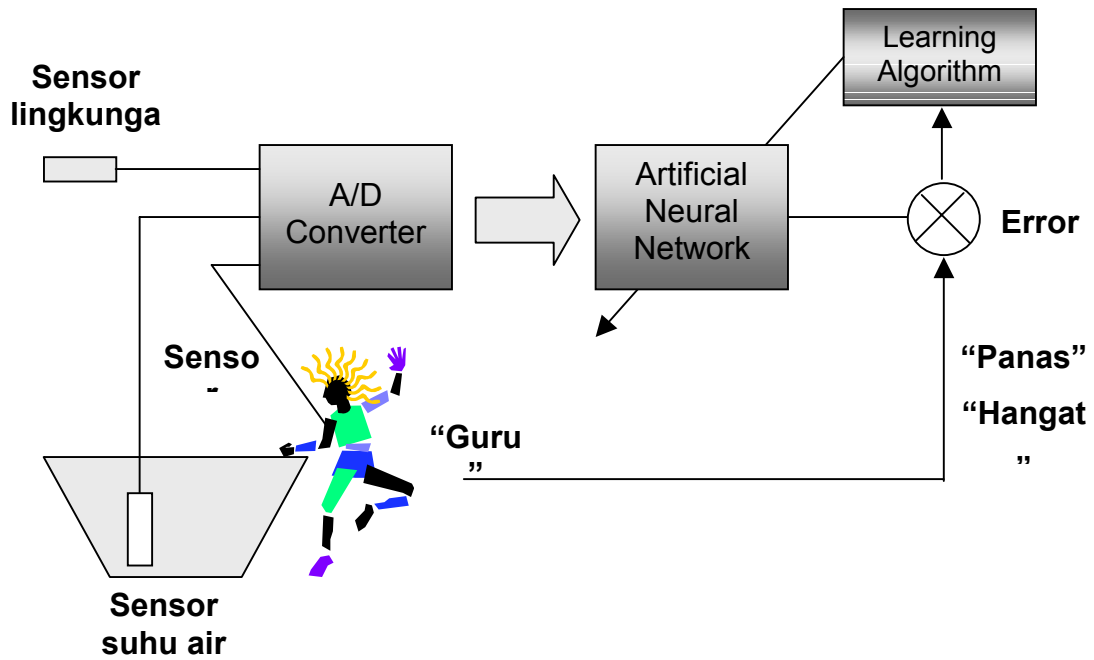
$$\begin{aligned} \text{CoG} &= \frac{((15 * 0.3) + (25 * 0.46) + (35 * 0.46) + (45 * 0.46) + (55 * 0.25))}{(0.3 + 0.46 + 0.46 + 0.46 + 0.25)} \\ &= 34.5' \end{aligned}$$

Tetapi jika menggunakan **integral** hasilnya adalah 38'

**Crisp Output**-nya adalah **Water Duration = 38 menit**.

Jadi untuk temperatur udara 92°F dan Kelembaban tanah 11%, maka sprinkle secara otomatis akan menyiram selama 38 menit.

**Contoh:** Aplikasi Adaptive Intelligent System adalah Sistem mengenali “Panas”, “Hangat”, dan “Dingin” Menggunakan JST dan Himpunan Fuzzy.





## Bab 7

# Pengolahan Bahasa Alami

Bab ini membahas beberapa masalah, strategi mesin penerjemah, dan aplikasi pengolahan bahasa alami lainnya. Pengolahan bahasa alami menggunakan Jaringan syaraf tiruan diusulkan dalam bentuk diagram blok.

**Bahasa:** suatu sistem untuk komunikasi yang terdiri atas **simbol** dan **aturan** yang digunakan untuk **mengeskpresikan ide, pikiran, dan perasaan**.

Masalah NLP dibagi menjadi dua, yaitu:

**1. Processing written text**

Menggunakan pengetahuan *lexical*, *syntactic*, dan *semantic* dari bahasa tersebut dan juga informasi dunia nyata yang diperlukan.

**2. Processing spoken language**

Menggunakan semua informasi yang diperlukan di atas ditambah pengetahuan tentang *fonologi* dan juga informasi yang cukup untuk mengatasi ambiguitas yang muncul dalam pengucapan.

**Lingkup NLP:**

1. Pemahaman Bahasa Alami
2. Mesin Penerjemah
3. Pemahaman Wacana
4. Intelligent Data Base
  - NLP + Knowledge Base
  - Dengan Fuzzy

**Karakteristik bahasa Alami:**

1. Deskripsi Tidak Lengkap
2. Ekspresi atau Emosi
3. Berkembang
4. Suatu fakta diekspresikan dengan banyak cara

Komponen-komponen di dalam proses pemahaman bahasa alami:

**1. Morphological Analysis**

Individual words dianalisis ke dalam komponen-komponennya, dan non-word tokens, seperti tanda baca, dipisahkan dari kata.

**2. Syntactic Analysis**

Deretan kata-kata ditransformasikan ke dalam struktur yang menunjukkan bagaimana kata-kata berhubungan antara satu dengan lainnya. Deretan kata yang tidak sesuai dengan aturan bahasa akan ditolak.

**3. Semantic Analysis**

- Memberikan arti terhadap struktur-struktur yang dibangun oleh syntactic analyzer.
4. **Discourse Integration**  
 Arti dari suatu kalimat tunggal mungkin bergantung pada kalimat sebelumnya dan mempengaruhi arti dari kalimat yang mengikutinya.
  5. **Pragmatic Analysis**  
 Struktur yang merepresentasikan apa yang dikatakan harus diinterpretasikan kembali untuk menentukan apa arti kalimat sesungguhnya. Sebagai contoh: "Do you know what time it is?" diinterpretasikan sebagai permintaan informasi waktu.

### Aplikasi Pengolahan Bahasa Alami:

#### 1. Psikologi

**ELIZA** : Meniru masukan + Respon yang sesuai (sudah disiapkan)

System : Halo, apa kabar?

User : Jelek.

System : Jelek?

User : Saya sedih.

System : Kamu sedih, kenapa?

User : Ayah saya membenci saya.

System : Ayah kamu membenci kamu? Ceritakan jika tidak keberatan...

#### 2. Basis Data

Parser untuk grammar dengan kecenderungan memiliki gangguan.

Show me xyz

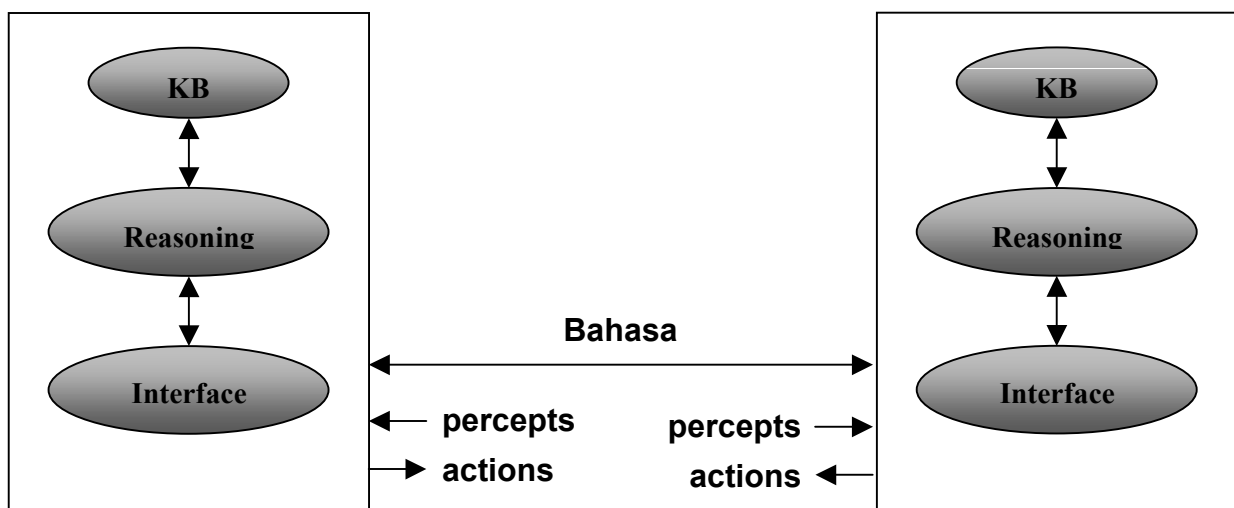
Show me all companies with stock > 100

Gile Bener, show me all companies with stock > 100

#### 3. Pengambilan Data dari Wacana

### Komunikasi dengan Bahasa

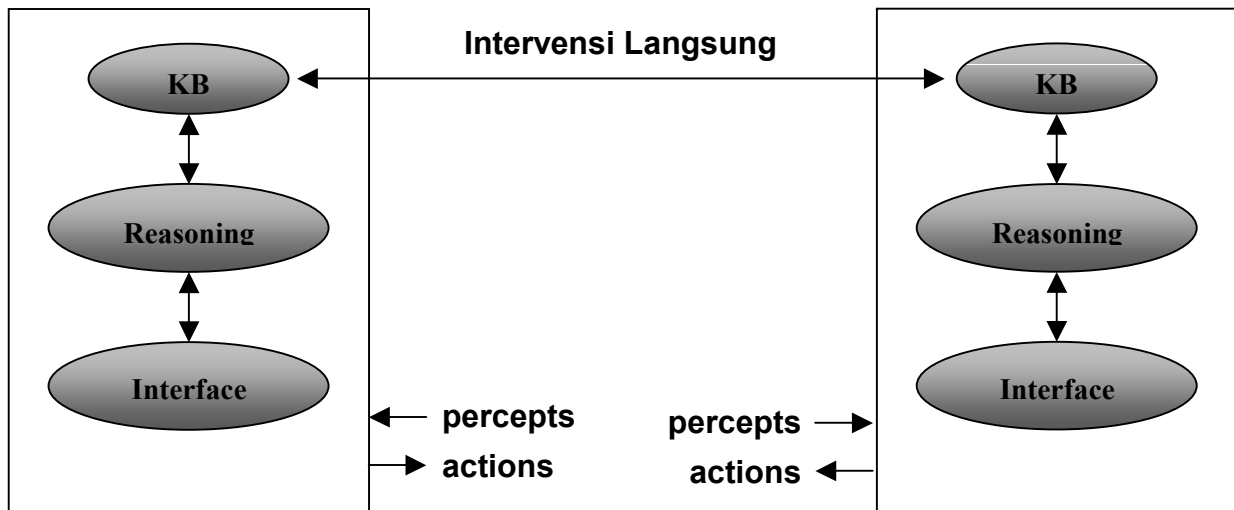
**Masalah**: menyamakan pemahaman atas perbedaan tiap KB agent.



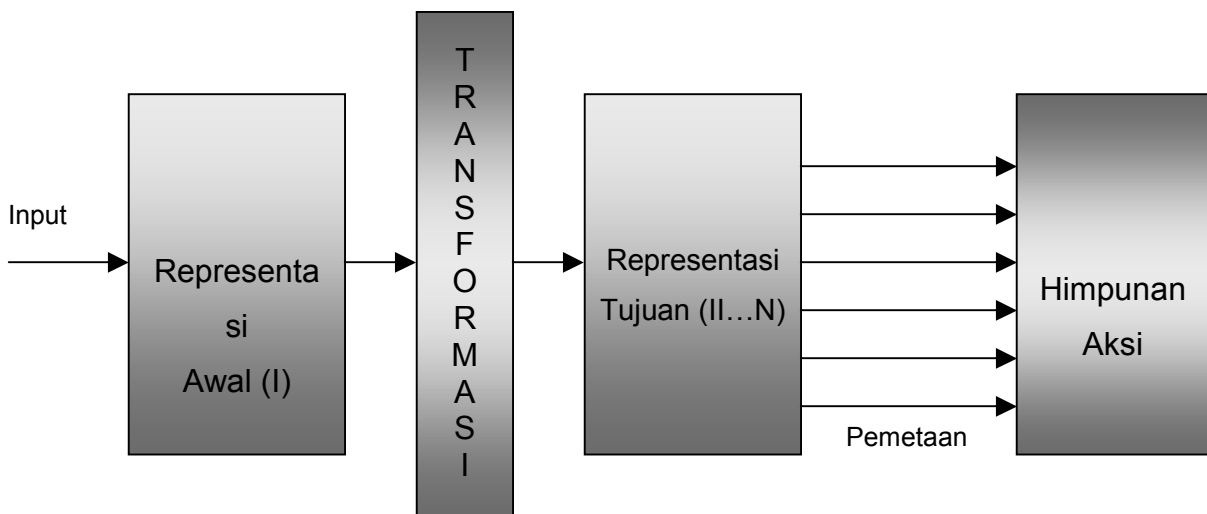




## Telepathic Communication



### 7.1. Pemahaman (*Understanding*)



#### Faktor Penyebab Masalah Pemahaman menjadi Sulit:

1. Kompleksitas representasi Tujuan  
Dalam ekstraksi informasi masukan sering dibutuhkan pengetahuan tambahan tentang Domain Pembicaraan (World)
2. Jenis-jenis pemetaan dan transformasi:
  - a. **1-1 (satu ke satu)**  

$$V_t = V_0 + a * t$$
 (rumus pasti yang hanya punya satu pengertian)
  - b. **1-N (satu ke banyak), biasanya perlu ditambah domain pembicaraan.**

lukisan itu bermutu tinggi.

Anak cantik itu tinggi hati

**c. N-1 (banyak ke satu), banyak cara untuk menyatakan sesuatu.**

Tell me about the last presidential election.

I'd like to see all the stories on the last presidential election.

I'm interested in the last presidential election.

**d. N-M (banyak ke banyak), banyak cara u/ menyatakan sesuatu yang sama.**

Orang itu kurang menyenangkan.

kurang menyenangkan : sombong = tinggi hati, angkuh, suka meremehkan

kurang menyenangkan : tidak ramah = tidak peduli, cuek, kasar.

3. Level Interaksi antara komponen-komponen Representasi Awal

a. Proses pemetaan tidak memperhatikan interaksi antar komponen

$$x := A * B + C * (D + \cos(Q)) * \sqrt{\sin(R)}$$

$$x := A * B + C * (D + \cos(Q)) * \sqrt{\cos(R)}$$

b. Proses pemetaan memperhatikan interaksi antar komponen

John saw the boy in the park **with a telescope**.

John saw the boy in the park **with a dog**.

John saw the boy in the park **with a statue**.

4. Kehadiran Noise dalam masukan.

Harus dapat membedakan input dan noise agar dapat mengekstrak dan membentuk proses pemahaman.

a. Speech

The cat scares all the birds away.

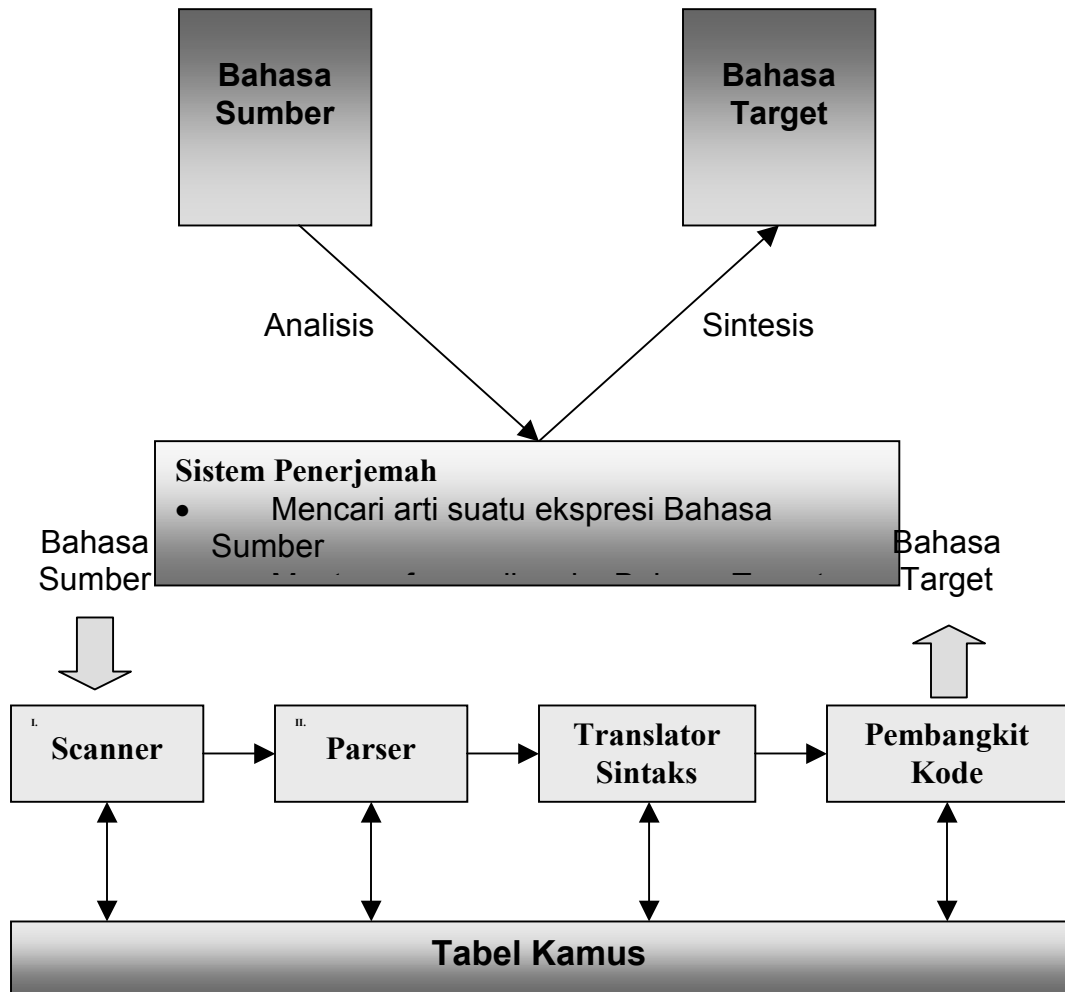
The cat's cares are few.

b. Written text

Noise Disposal Parser : Yang bukan kata kunci.

Kat-kata yang tidak bermakna.

## 7.2 Mesin Penerjemah

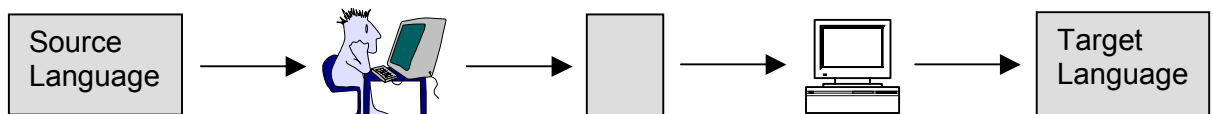


## 7.3 Jenis Mesin Penerjemah (Berdasarkan Tingkat Otomasi):

### 1. Alat Bantu Terjemah

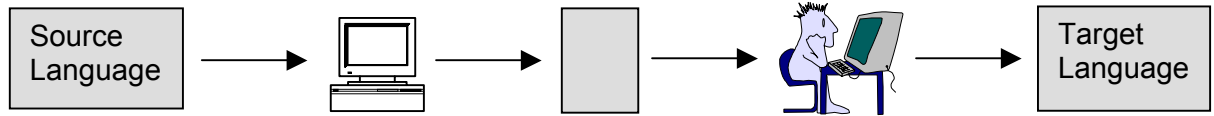


### 2. “Pre-Editing” mesin Penerjemah

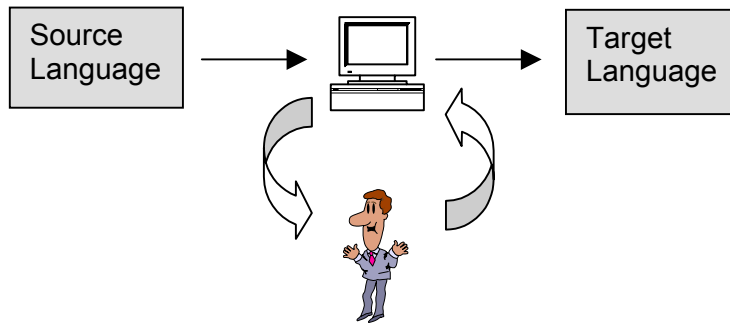




### 3. “Post-Editing” mesin Penerjemah



### 4. Sistem Penerjemah Interaktif



Memberikan pertanyaan yang perlu dijawab user untuk:

- *word-sense ambiguity*
- *referential ambiguity*

## 7.4 Strategi Sistem Penerjemah:

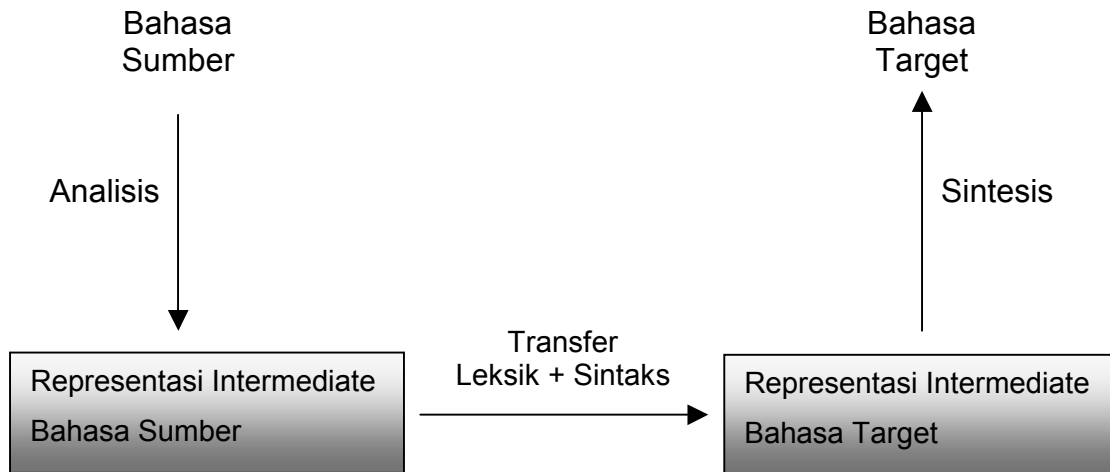
### 1. Strategi Penerjemahan Langsung (dirancang untuk dua bahasa spesifik)

Misal: mesin penerjemah SYSTRAN

#### Tahapan Proses:

- Peninjauan kamus Bahasa Sumber
- Analisa Morphologi
- Identifikasi homograph
- Identifikasi kata majemuk
- Identifikasi kata benda dan kata kerja
- Pemrosesan idiom
- Pemrosesan preposisi
- Identifikasi Subyek dan Predikat
- Identifikasi kerancuan sintaks
- Sintesa Morphologi Bahasa Target
- Pengaturan kembali kata dan frasa Bahasa Target

## 2. Strategi Transfer

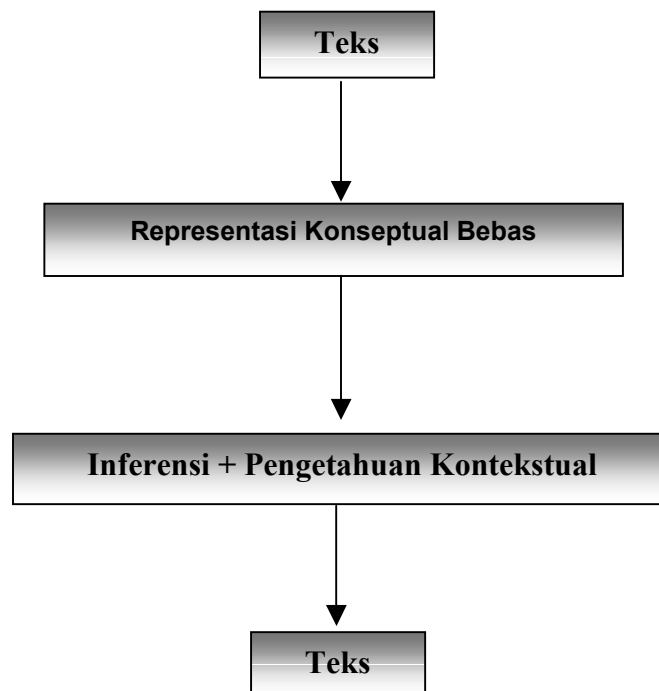
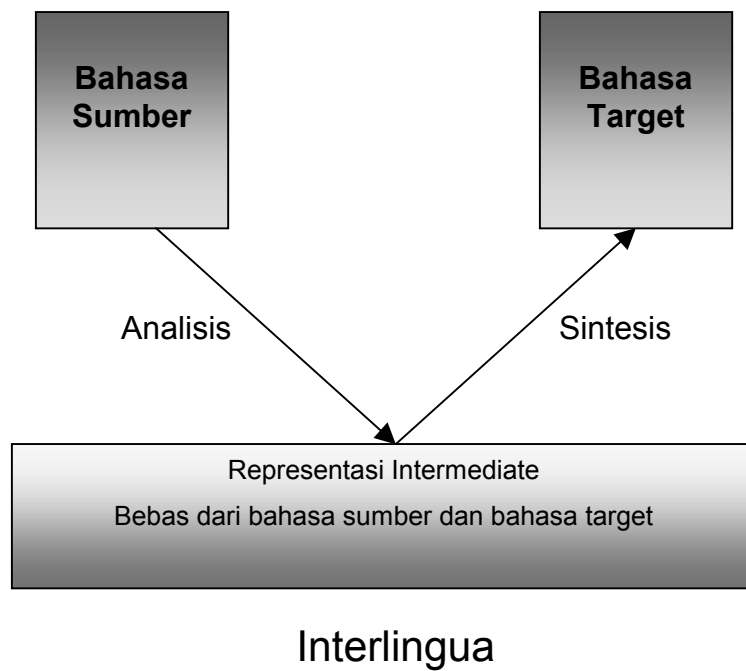


Kamus yang dibutuhkan:

- Kamus Bahasa Sumber
- Kamus Transfer Bilingual
- Kamus Bahasa Target

Tahap transfer butuh komponen **Bilingual Spesifik** untuk setiap Bahasa Sumber dan Bahasa Target

### 3. Strategi Interlingua



## 7.5 Mesin Penerjemah Menggunakan JST

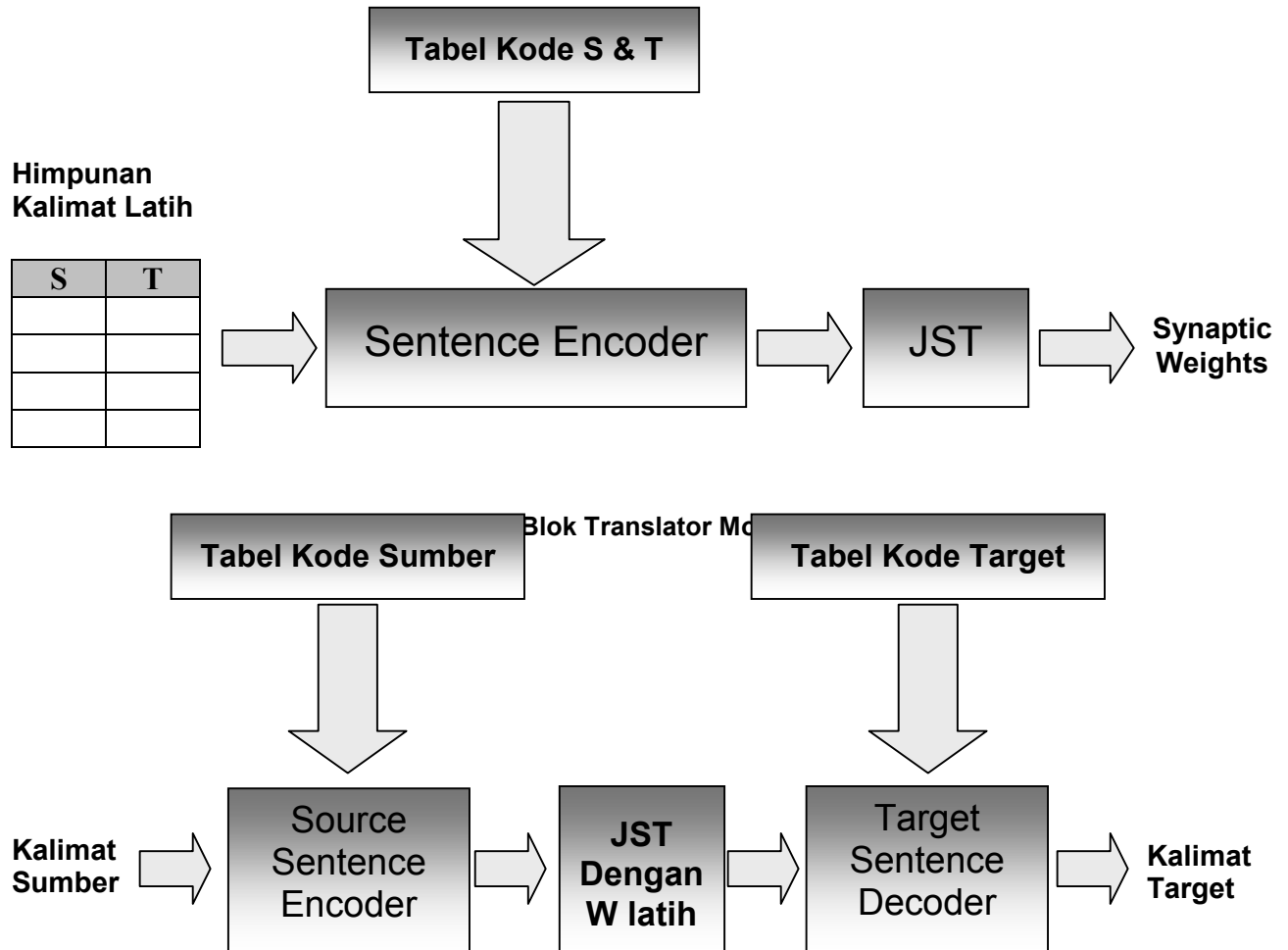
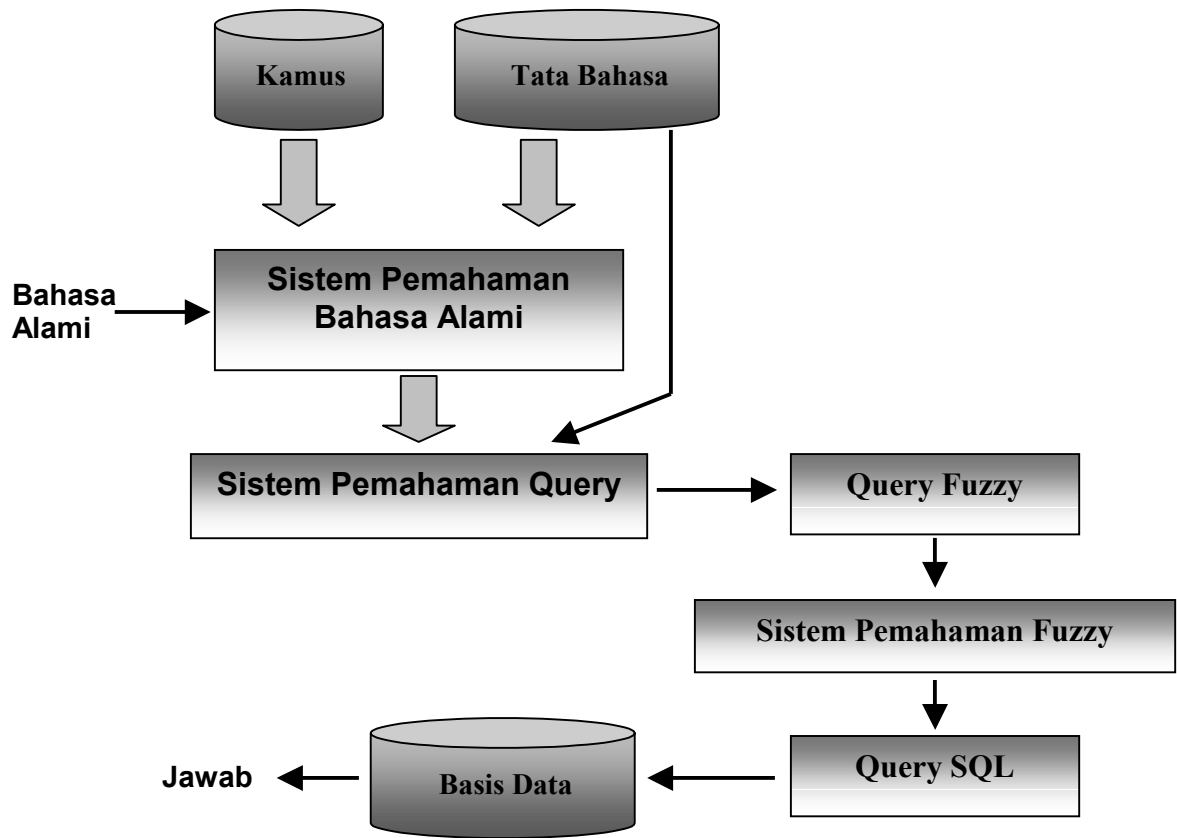


Diagram Blok Translator Mode Produksi.



## 7.6 Query menggunakan NLP dan Fuzzy



Contoh Kasus:

- Bahasa Alami:  
Tampilkan daftar harga barang yang penjualannya cukup tinggi.
- Query Fuzzy:  
SELECT Harga FROM Barang  
WHERE Penjualan equal Jual(Barang, Cukuptinggi)
- SQL:  
SELECT Harga FROM Barang  
WHERE Penjualan >= 10.000



## Bab 8

# Pengetahuan Kognitif

Bab ini membahas aplikasi *inteligensia buatan* dalam sistem pengajaran berbantuan komputer. Dalam hal ini diharapkan *inteligensia buatan* memberikan nilai lebih dalam sistem pengajaran berbantuan komputer tersebut.

### 8.1 Pengajaran Berbantuan Komputer (PBK)

Pengajaran Berbantuan Komputer (*Computer Aided Instruction*) terdiri atas komponen-komponen sebagai berikut:

1. **Hardware**, yaitu komputer dan piranti pendukungnya
2. **Software**, dapat berupa sistem operasi atau modul program komputer untuk merepresentasikan materi perangkat ajar.
3. **Brainware**, yaitu pembuat sistem, pengajar, atau siswa

**Perangkat ajar secara umum terdiri atas tiga elemen penting:**

1. **Modul domain materi**, berisi materi yang akan dipresentasikan kepada siswa
2. **Sistem pengendali pengajaran**, berkaitan dengan strategi penyampaian materi, sehingga presentasi menjadi terarah dan sistematis
3. **Antar muka pengajaran**.

**Perangkat ajar secara umum dibedakan menjadi empat kategori:**

- a. **Tutorial** (Penjelasan)  
Tipe perangkat ajar ini digunakan untuk menyampaikan suatu materi pengajaran.
- b. **Drill And Practise** (Latihan dan Praktek)  
Jenis ini digunakan untuk menguji tingkat pengetahuan siswa dan mempraktekkan pengetahuan mereka, sehingga pembuatannya disesuaikan dengan tingkat kemampuan masing-masing siswa.
- c. **Simulation** (Simulasi)  
Pada perangkat ajar simulasi siswa dihadapkan pada situasi yang mirip dengan kehidupan nyata. Aplikasi simulasi digunakan untuk mempelajari objek yang rumit dan melibatkan banyak besaran yang saling berhubungan yang seringkali siswa kesulitan mempelajarinya. Dunia nyata dipresentasikan dalam bentuk model dan kemudian dengan teknik simulasi siswa dapat mempelajari kelakuan sistem.
- d. **Games** (Permainan)  
Berdasarkan tujuan belajarnya jenis permainan dibagi menjadi dua tipe, yaitu:
  - **Permainan Instrinsik (*Intrinsic Games*)**  
Mempelajari aturan permainan dan keahlian dalam suatu permainan (*games*).
  - **Permainan Ekstrinsik (*Extrinsic Games*)**  
Permainan hanya sebagai perangkat tambahan sebagai fasilitas belajar dan membangkitkan motivasi siswa.

Perangkat ajar dapat diimplementasikan dalam tipe tertentu tergantung pada bidang pengajaran, sasaran yang ingin dicapai, dan siswa sebagai pemakai sistem. Perangkat ajar dapat diimplementasikan dalam berbagai bentuk. Pemilihannya tergantung pada materi yang akan dibahas, sebab antara materi dan alur pengajaran terdapat keterkaitan yang erat.

## **8.2 Bentuk-bentuk Perangkat ajar:**

### **1. Buku elektronik**

Memindahkan isi suatu buku ke komputer. Siswa dapat memilih materi yang akan dipelajarinya tanpa ada batasan dan prasyarat.

### **2. Frame**

Materi dan bahan evaluasi disusun secara sistematis, per modul dan mempunyai suatu sistem kendali pengajaran.

### **3. Perluasan PBK**

Merupakan bentuk frame yang diperluas dengan kemampuan membangkitkan alur pengajaran sesuai kemampuan siswa.

### **4. Pengajaran Berbantuan Komputer Cerdas**

Mengeksplorasi teknik-teknik kecerdasan buatan dalam pembangkitan alur pengajarannya. Sebagaimana prinsip dalam kecerdasan buatan, pada bentuk ini antara materi dengan alur pengajaran diharapkan tidak terdapat keterkaitan.

## **Arsitektur Proses Pengajaran**

Arsitektur Proses Pengajaran pada umumnya mengikuti alur sebagai berikut:

1. Presentasi materi
2. Evaluasi
3. Analisis Respons
4. Tanggapan (komentar) berdasarkan analisis
5. Tindak lanjut, alur yang boleh dipilih siswa (tergantung pembuatan sistem), yaitu:
  - a. Materi dilanjutkan dengan memberikan penjelasan
  - b. Jika terdapat kesalahan, ditampilkan materi perbaikan dan selanjutnya siswa hanya bebas memilih materi sebelumnya.

## **8.3 Perangkat Ajar Berintelegensia (PAB)**

Penerapan kecerdasan buatan dalam bidang pendidikan yang dalam bidang informatika dikenal dengan istilah *Intelligent Computer Assisted Instruction* (ICAI). Dalam referensi lain PAB sering juga disebut *Intelligent Tutoring System* (ITS).

**PAB adalah generasi baru dari sistem PBK yang mengintegrasikan :**

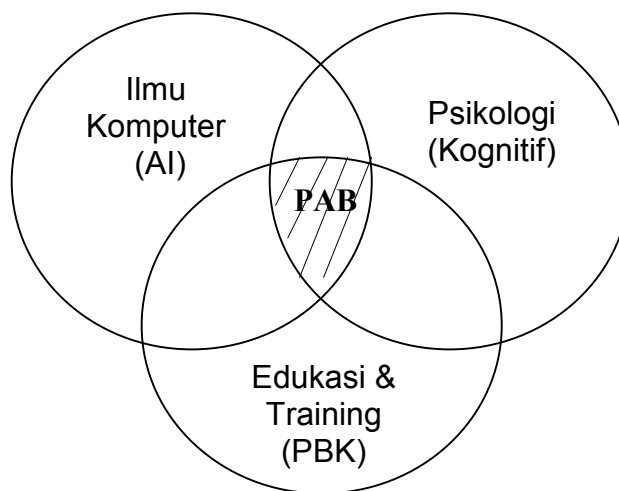
1. Materi pengajaran (*course material*)
2. Model siswa
3. Kemampuan diagnostik dalam suatu kerangka yang utuh.

PAB memakai teknik kecerdasan buatan untuk membangkitkan lingkungan pengajaran yang memungkinkan siswa mengikuti tahapan pengajaran secara alami, seperti belajar secara eksplorasi atau belajar dengan latihan (*learning by doing*).

Pengembangan PAB melibatkan tiga disiplin ilmu yang saling berinterseksi yaitu :

1. Ilmu komputer
2. Psikologi kognitif
3. Ilmu pendidikan.

Perkembangan penelitian dibidang PAB ditentukan oleh perkembangan ketiga disiplin ilmu tersebut. **Kualitas perancangan dan pembangunan** PAB juga tergantung dari sejauh mana pemahaman perancang memahami ketiga disiplin ilmu tersebut dan mengaplikasikannya ke dalam program.



**Gambar 8.1** Domain Perangkat Ajar Berintelijensia.

### 8.3.1 Komponen Pengajaran (Instruksional)

PAB ditinjau dari aspek pengajarannya, memiliki tiga komponen yaitu [GREG87]:

#### 1. Modul Kepakaran (*Expertise Module*)

Berisi pengetahuan domain yang akan diajarkan kepada siswa. Modul ini dipakai untuk membangkitkan penyajian materi dan mengevaluasi performansi hasil belajar siswa.

#### 2. Model Siswa

Model siswa dipakai untuk menilai kondisi pengetahuan siswa terhadap materi pengajaran. Dua pendekatan untuk mengetahui model siswa yaitu [GREG87] :

##### 2.1 Overlay Model

Pada pendekatan ini kondisi pengetahuan siswa merupakan **subset** dari basis pengetahuan pakar. Model siswa diperoleh dari hasil **perbandingan** antara performansi siswa dengan performansi seorang pakar pada domain materi dalam mengerjakan tugas yang sama.

##### 2.2 Buggy Model

Pada pendekatan ini model siswa dibentuk dengan mengevaluasi dan mendeteksi pemahaman-pemahaman siswa yang keliru (miskonsepsi). Domain materi dipresentasikan dengan aturan produksi dan daftar miskonsepsi dipresentasikan sebagai varian dan aturan.

Sumber-sumber informasi sebagai parameter input pembentukan model siswa:

1. Sistem mengobservasi perilaku siswa dalam memecahkan masalah
2. Sistem memberikan pertanyaan-pertanyaan secara langsung kepada siswa
3. Sistem membuat asumsi-asumsi berdasarkan pengalaman belajar siswa.
4. Sistem membuat asumsi-asumsi tingkat kesulitan pengajaran.

Sumber informasi 1 dan 2 adalah yang paling banyak dipakai dalam aplikasi PAB saat ini.

### 3. Modul Tutorial (*Tutoring Module*)

Memuat spesifikasi domain materi pengajaran dan strategi penyajiannya. Pengembangan PAB dapat memfokuskan hanya pada satu atau beberapa komponen pembangunnya dan tidak menonjolkan komponen lainnya.

#### 8.3.2 Paradigma Pengajaran

##### 1. Metode Socratic (*Mixed-Initiative Dialogues*)

Pada tipe ini program mengikutsertakan siswa dalam percakapan dua arah dan mencoba mengajar murid dengan memandu memecahkan masalah. Paradigma ini paling cocok untuk domain materi yang bersifat **konseptual** dan **prosedural**.

##### 2. Pembimbing (*Coaches*)

Layaknya seorang pelatih, PAB jenis ini mengobservasi performansi hasil belajar murid dan menyediakan saran-saran untuk memperbaiki performansi hasil belajarnya. Tipe ini paling cocok untuk program-program yang bersifat **pemecahan masalah** (seperti simulasi dan permainan).

##### 3. Tutor Diagnostik (*Diagnostic Tutor*)

Tipe ini berusaha mencari kelemahan dan kesalahan (*debugging*) dari hasil kerja murid. Proses *debugging* mengacu pada katalog kesalahan (*bug catalog*) yang memuat daftar kesalahan dan miskonsepsi yang biasanya dilakukan oleh murid dalam memecahkan masalah.

##### 4. Konsep Dunia Mikro (*Microworld Concept*)

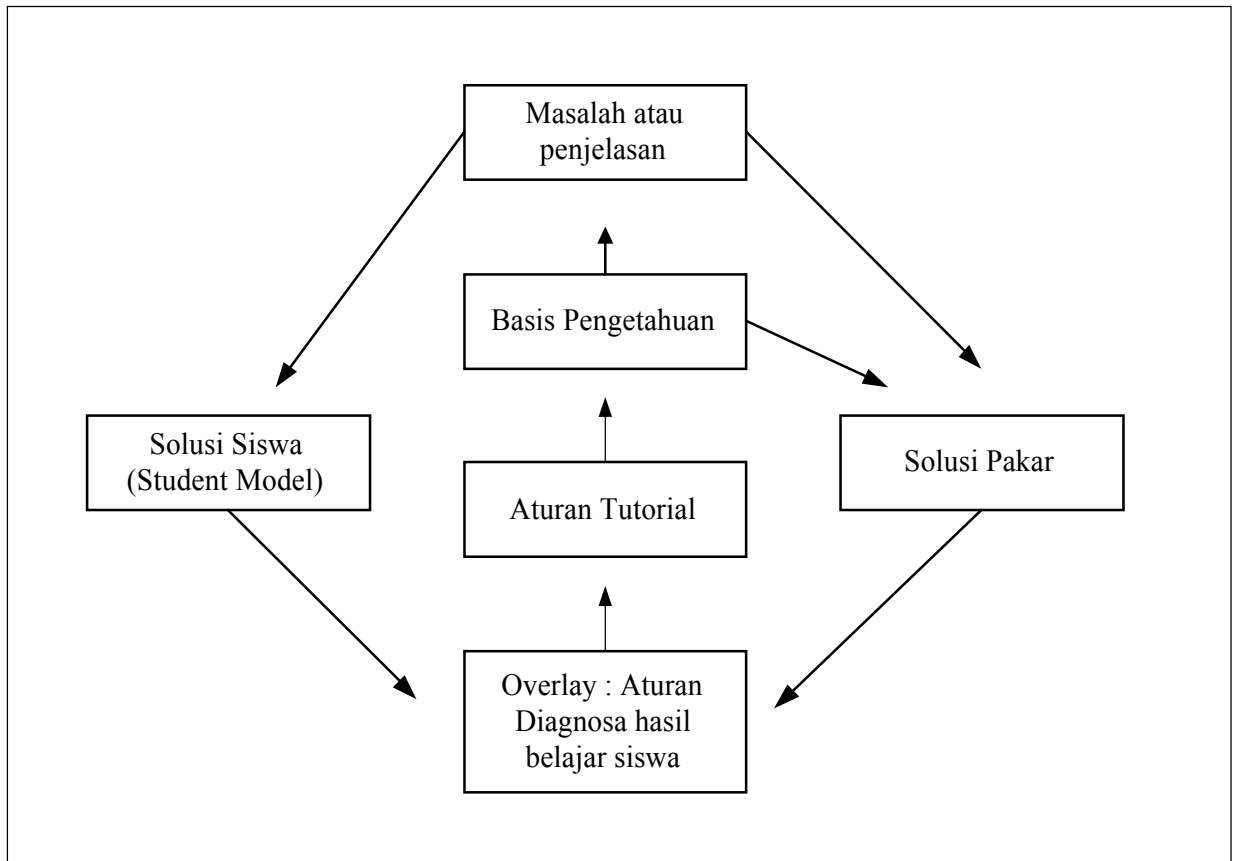
Menyediakan fasilitas yang memungkinkan siswa mengeksplorasi domain materi, misalnya dengan permainan dan simulasi. Tipe ini dipergunakan sebagai alat bantu komputasi pada domain-domain masalah yang eksak seperti geometri, fisika, matematika, atau musik.

##### 5. Sistem Pakar Artikulasi

Yaitu sistem pakar yang dapat menerangkan proses penalarannya dalam menghasilkan keputusan atau solusi. Sistem pakar bisa dipakai sebagai alat bantu dalam pekerjaan yang menyediakan fasilitas pelatihan keterampilan memecahkan masalah dan pembuat keputusan. Tipe ini potensial untuk diaplikasikan dalam domain training, yaitu aplikasi yang bertujuan pelatihan.

### Model Pengajaran Perangkat Ajar Berintelegensia

Karakteristik sistem PAB berbeda-beda tergantung dari **tujuan pembangunan sistem**, **karakteristik domain pengetahuan**, **karakteristik siswa** dan **metode teknis** yang dipakai. Pada dasarnya semua PAB memiliki model umum sbb:



**Gambar 8.2** Model Umum Perangkat Ajar Berintelijensia

Performansi siswa dievaluasi dengan membandingkannya dengan performansi pakar.

Hasil perbandingan dipakai untuk menginferensi penyebab-penyebab masalah siswa atau terjadinya pemahaman siswa yang salah (miskonsepsi) dan kemudian sistem mendefinisikan materi-materi yang dibutuhkan untuk mengatasi masalah-masalah tersebut.

Jika dibandingkan dengan realita pengajaran antara guru dan murid, model umum tersebut adalah penyederhanaan dari suatu sistem pengajaran yang lengkap, yaitu:

1. **Mekanisme evaluasi performansi siswa** diperoleh hanya dengan menganalisa respon siswa terhadap permasalahan dan pertanyaan yang diberikan. Evaluasi hasil respon ini diperlukan dalam mendiagnosa penyebab masalah dan sebagai acuan sistem menginferensi kebutuhan materi belajar tutorial.

2. Kebanyakan dari sistem PAB yang ada pada saat ini **mengabaikan banyak variabel potensial** yang seharusnya dipertimbangkan dalam proses diagnosa dan menentukan kebutuhan belajar.
3. **Kemampuan membangkitkan materi belajar terbatas** dan tergantung pada unit pengetahuan dan unit informasi yang ada pada *knowledge base*. Unit informasi dan pengetahuan dalam *knowledge base* direpresentasikan dalam bentuk level mikro (fakta, dan relasi antar fakta), sistem PAB tidak memiliki kemampuan memodifikasi dan memperbaiki level makro dari struktur pengetahuan (yaitu kurikulum materi).
4. **Prinsip dan strategi pengajaran direpresentasikan dalam aturan tutorial.** Strategi pengajaran direpresentasikan secara bebas yaitu sebagai modul tersendiri yang terpisah dan tidak terikat dengan domain pengetahuan. Pemisahan ini merupakan ciri khas suatu program yang berbasis kecerdasan buatan. Pemisahahn ini akan memudahkan dalam pemnbangunan sistem, implementasi dan modifikasi.
5. Kebanyakan dari sistem PAB yang ada pada saat ini **tidak memiliki kemampuan memperbaiki struktur *knowledge base*** maupun aturan-aturan diagnostik dan aturan tutorialnya.



## Daftar Pustaka

- [BES99] Besari, M. S, [1999], *Manusia, Ilmu Pengetahuan dan Komputer. Keynote speech* pada Seminar Nasional I Kecerdasan Komputasional, Universitas Indonesia Depok. Indonesia.
- [CAR93] Carey, J., Gross, N., Mc Williams, G, [1993], *The Light Fantastic*. Business Week.
- [FAU94] Fausett, Laurene. [1994]. *Fundamentals Of Neural Networks*, Prentice Hall, New Jersey.
- [GEN97] Gen, M. Cheng, R., [1997], *Genetic Algorithms & Engineering Design*, John Wiley & Sons, Inc. 1997.
- [GOL89] Goldberg, D.E, [1989], *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley Publishing Co.
- [GREG87] Kearsley, Greg, [1987], *Artificial Intelligence and Instruction: Applications and Methods*. Addison-Wesley Pub.Company.
- [HAY94] Haykin, Simon, [1994], *Neural Networks: A Comprehensive Foundation*. Macmillan Publishing Company: New York.
- [IKE94] Ikeda Takahiro, Min-Yao Hsu, Hiroshi Imai, Shigeki Nishimura, Takeo Hashimoto, Kenji Tenmoku, Kunihiko Mitoh, *A Fast Algorithm For Finding Better Routes by AI Search Techniques*. Proceeding Vehicle Navigation & Information System, 1994. Japan.
- [OJA82] Oja, E, [1982], *A Simplified Neuron Model as a Principal Component Analyzer*. Journal of Mathematical Biology 15, 267-273.
- [RIC91] Rich, Elaine, and Knight, Kevin, [1991], *Artificial Intelligence*. McGraw-Hill, Inc. second edition.
- [RUS95] Russel, Stuart, and Norvig, Peter, [1995], *Artificial Intelligence: A Modern Approach*. Prentice Hall International, Inc.
- [SAN89] Sanger, T.D, [1989], *Optimal Unsupervised Learning In A Single Layer Linear Feedforward Neural Network*. Neural Networks 12, 459-473.