2. TEORI PENUNJANG

2.1. Kecerdasan Buatan

Sejak awal mula, komputer adalah sebuah mesin yang hebat. Dimana komputer dapat memproses sejumlah besar data secara cepat dan akurat. Dewasa ini komputer tidak hanya menyimpan, memanipulasi dan memelihara data, tetapi makin bertambah dengan mengambil peranan dalam pengambilan keputusan. Dalam periode yang singkat, teknologi berpindah dari pemrosesan data, melalui era manipulasi informasi menjadi pemrosesan pemgetahuan. Bidang studi yang mempelajari pemrosesan pengetahuan ini adalah *Artificial Intelligence* atau kecerdasan buatan.

Tujuan utama dari *Artificial Intelligence* adalah membuat sebuah perangkat lunak yang dapat 'berpikir' seperti manusia. Dengan kata lain *Artificial Intelligence* atau kecerdasan buatan merupakan subbidang komputer yang mencurahkan perhatiannya pada usaha untuk menciptakan *hardware* dan *software* komputer untuk melakukan segala sesuatu seperti yang dilakukan manusia.

Artificial Intelligence atau kecerdasan buatan harus didasarkan pada prinsip – prinsip teori dan terapan yang menyangkut struktur data yang digunakan dalam representasi pengetahuan (*knowledge representation*), algoritma yang diperlukan dalam penerapan pengetahuan itu, serta teknik – teknik bahasa pemrograman yang dipakai dalam implementasinya.

Namun demikian, definisi diatas mungkin tidaklah terlalu baik, karena kenyataannya nama kecerdasan itu sendiri tidak didefinisikan dengan baik. Pertanyaan – pertanyaan mengenai : pengertian kecerdasan, hubungan kecerdasan dengan proses belajar dan kreativitas, dan pembuktian adanya mekanisme tertentu dalam kecerdasan, merupakan pertanyaan yang belum terjawab dan kesemuanya itu justru membantu untuk menggeluti metode problema dan solusi yang membentuk intisari kecerdasan buatan modern.

Perangkat lunak standar hanya dapat menyelesaikan persoalan yang telah diprogram secara spesifik. Jika suatu program standar perlu diubah untuk

menyesuaikan diri dengan suatu informasi baru, seluruh program harus dilihat satu persatu sampai didapat ruang optimal untuk menyisipkan perubahan – perubahan atau modifikasi tersebut. Cara seperti ini tidak hanya memboroskan waktu, namun juga dapat mengetahui bagian tertentu dari program itu yang menyebabkan terjadinya kesalahan.

Sebaliknya kecerdasan buatan dapat memungkinkan komputer untuk 'berpikir'. Dengan cara penyederhanaan program, kecerdasan buatan dapat menirukan proses belajar manusia sehingga informasi baru dapat diserap dan digunakan sebagai acuan di masa – masa yang akan datang.

Teknik yang digunakan dalam kecerdasan buatan memungkinkan dibuatnya sebuah program yang setiap bagiannya mengandung langkah – langkah *independen* dan dapat diidentifikasi dengan baik untuk dapat memecahkan sebuah atau sejumlah persoalan. Setiap potong bagian program adalah seperti sepotong informasi dalam pikiran manusia. Jika informasi tadi diabaikan, pikiran akan secara otomatis dapat mengatur cara kerjanya sehingga menyesuaikan diri dengan fakta atau informasi baru. Otak tidak perlu selalu mengingat setiap potongan informasi yang telah dipelajari. Hanya yang relevan dengan persoalan yang dihadapilah yang digunakan. Demikian pula kecerdasan buatan, setiap potong dari program *Artificial Intelligence*, dapat dimodifikasi tanpa mempengaruhi struktur seluruh programnya. Keluwesan ini dapat menghasilkan program yang semakin efisien dan mudah dipahami .

Program *Artificial Intelligence* modern umumnya berisi sejumlah komponen modular, aturan tingkah laku, yang tidak melakukan eksekusi secara kaku, namun justru memberikan respon sesuai dengan struktur persoalan tertentu yang dihadapinya pada suatu saat. Sistem semacam ini akan memiliki keluwesan yang sangat tinggi yang memungkinkan sejumlah program yang relatif kecil untuk menangani berbagai tingkah laku dalam rentang yang besar dengan problem dan situasi yang beragam.

Pada saat ini teknologi *Artificial Intelligence* atau kecerdasan buatan yang paling banyak berkembang ada tiga bidang pokok, yaitu :

a. Sistem Pakar (*Expert System*)

   Sistem pakar merupakan sistem dimana dapat diharapkan dapat berpikir seperti halnya seorang pakar dalam memecahkan masalah. Sistem pakar ini merupakan bidang *Artificial Intelligence* yang paling berkembang pesat.

b. Robotika (*Robotics*)

   Robot adalah sebuah peralatan elektronika yang dapat diprogram untuk melaksanakan tugas – tugas manusia. Tetapi tidak semua robot dianggap bagian *Artificial Intelligence*, sebab robot hanya dapat melakukan tindakan – tindakan yang telah diprogram sebelumnya. Sedangkan bagian kecerdasan dari robot memungkinkan untuk memberikan tanggapan dan menyesuaikan diri terhadap perubahan lingkungannya. Sebagai perbedaan antara mesin otomatis dengan robot yang cerdas adalah robot dapat merasakan lingkungannya dan menyesuaikan tindak tanduknya sebagai hasil dari informasi yang diperolehnya. Sehingga dapat dikatakan bahwa bidang ini terbatas hanya mempelajari bergeraknya sebuah robot dimana diharapkan dapat membantu manusia dalam pekerjaannya.

c. Pengolahan Bahasa Alami (*Natural Language Processing*)

   Teknologi pengolahan bahasa alami ini memberikan kemampuan kepada komputer untuk berkomunikasi dengan pemakainya dalam bahasa sehari – hari. Sehingga dapat dikatakan bidang ini mempelajari usaha untuk membuat komputer dapat mendengar, membaca atau berbicara seperti manusia.

2.2. Sistem Pakar Atau Expert System

Sistem pakar adalah sebuah perangkat lunak yang memasukkan suatu pengetahuan dan proses penalaran seorang pakar. Atau dengan kata lain, sistem pakar merupakan perangkat lunak yang berbasis pengetahuan yang menyediakan penyelesaian berkualitas pakar. Sehingga sistem pakar juga disebut sebagai sistem yang berbasis pengetahuan (*Knowledge Based System*).

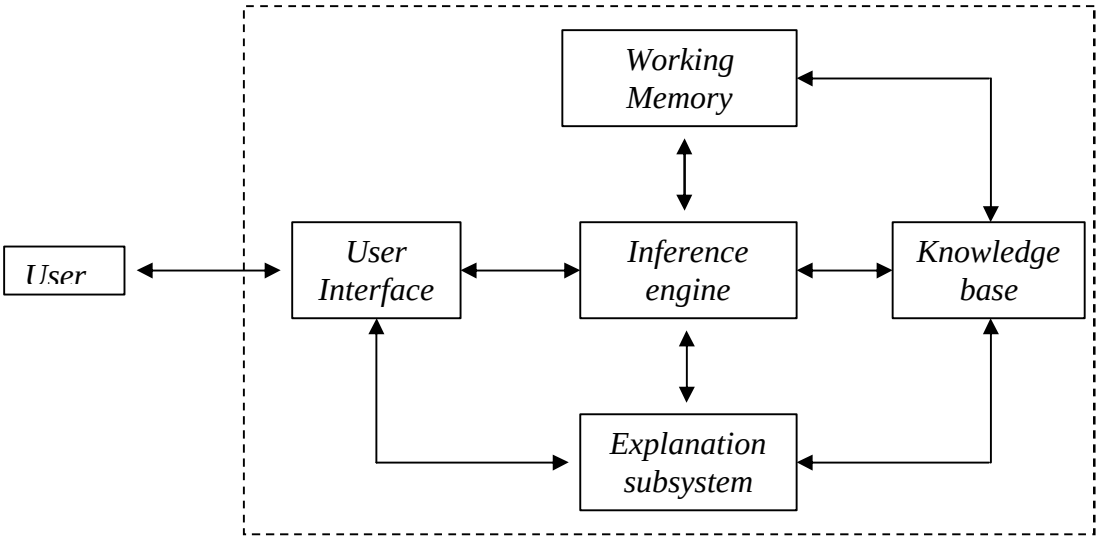
Ciri khusus dari sistem pakar adalah memisahkan *knowledge* dan kontrol. Dimana *knowledge* berada dalam *knowledge base*. Sedangkan kontrol berada dalam *inference engine*.

Pada kenyataannya, kualitas sebuah sistem pakar sering disamakan dengan *knowledge* yang diletakkan dalam *knowledge base*. Yang dimaksud dengan kualitas dari sebuah sistem pakar adalah kemampuan dari sistem pakar untuk menyamai kemampuan dari seorang pakar.

2.2.1. Konsep Dasar Sistem Pakar

Pada dasarnya konsep dasar dari sebuah sistem pakar adalah proses pemindahan keahlian yang dimiliki oleh seorang pakar ke dalam suatu sistem dimana sistem tersebut diharapkan dapat memiliki kemampuan untuk ‘berpikir’ seperti halnya seorang pakar. Orang yang bertugas untuk membuat sistem pakar disebut dengan *knowledge engineer*.

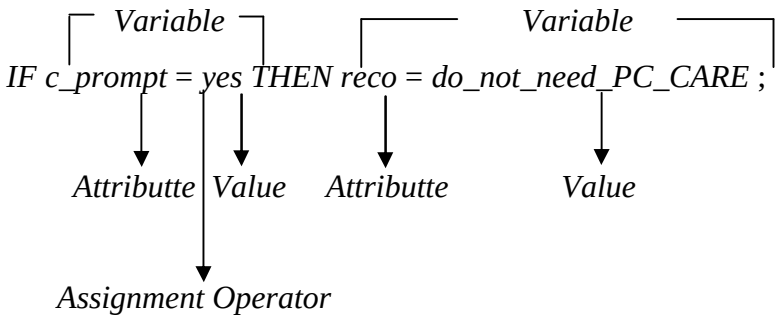
Sebuah sistem harus memiliki sejumlah komponen yang saling berinteraksi untuk mencapai tujuannya. Pada sistem pakar komponen utama yang harus ada yaitu : *knowledge base, inference engine* dan *user interface*. Sedangkan *explanation module* dan *working memory* merupakan komponen pendukungnya. Pada gambar 2.1 menunjukkan sebuah arsitektur sistem pakar dimana komponen – komponen pembentuknya saling berinteraksi.



Gambar 2.1. Arsitektur Sistem Pakar

2.2.1.1. Knowledge Base

Knowledge base yang merupakan jantung dari sistem pakar dimana keahlian seorang pakar yang telah diformat dalam bentuk *rule*. *Knowledge base* ini biasanya disimpan dalam bentuk *file* teks sehingga mudah diakses dan dimodifikasi. Untuk membentuk *knowledge base*, seorang pakar harus terlebih dahulu mengidentifikasi apa yang akan menjadi objek atau *goal* yang ingin dicapai. Setelah itu mengidentifikasi semua *attributte* beserta *value – value* yang dimiliki tiap *attributte*, baik untuk *attributte condition* maupun *conclusion*, yang mendukung objek atau *goal* yang ingin dicapai. Untuk membentuk *knowledge base*, *knowledge* yang didapat oleh seorang pakar dituliskan dalam format : *IF condition THEN conclusion.*



Gambar 2.2. Struktur Suatu Rule

Pada dasarnya, setiap *rule* dimulai dengan *keyword IF* dan diikuti dengan beberapa *condition* yang telah dievaluasi sebelumnya, kemudian diikuti *keyword THEN* dan diakhiri dengan suatu *conclusion. Condition – condition* dari suatu *rule* dapat diserikan dengan menggunakan *keyword AND* atau *OR*. *Keyword AND* artinya bahwa kedua *condition* yang mengapit *keyword* tersebut harus bernilai benar supaya *condition* dari *rule* tersebut terpenuhi.

Contoh penulusan r*ule* dengan k*eyword AND* :

IF c_prompt = no AND fan = no THEN ps = defective;

Keyword OR artinya bahwa salah satu atau kedua *condition* yang mengapit *keyword* tersebut harus bernilai benar. Selain itu kedua *condition* tersebut berisi fakta yang sama, tetapi nilai dari faktanya berbeda.

Contoh penulisan r*ule* dengan k*eyword OR* :

IF temperature = abnormal OR temperature = not_known THEN problem = serious;

Selain itu *keyword AND* dan *OR* dapat digunakan secara bersamaan dalam *condition*.

Contoh penulisan *rule* dengan *keyword AND dan OR* :

IF member_status = ok AND reason = new_case OR reason = follow_up_case
THEN support = level_1;

Jika semua *condition* dari suatu *rule* bernilai benar, maka *THEN clause* atau *conclusion* dari *rule* tersebut dicapai atau *fired.* Hal ini menyebabkan *attributte* dari *variable conclusion* yang terletak di sebelah kiri tanda sama dengan diisi dengan *value* dari *variable conclusion* yang terletak di sebelah kanan tanda sama dengan.

Contoh penulisan *knowledge base* :

IF engine_type = proof
THEN plane = C130;

IF engine_type = jet AND wing_position =low
THEN plane = B747;

IF engine_type = jet AND wing_position = high AND bulges = none
THEN plane = C5A;

IF engine_type = jet AND wing_position = high AND bulges = aft_of_wing
THEN plane = C141;

2.2.1.2. Working Memory

Working memory pada dasarnya merupakan sebuah perluasan dari *knowledge base*. *Working memory* digunakan untuk menyimpan *intermediate result*, yaitu ketika usaha untuk menyelesaikan setiap permasalahan, sistem menyimpannya dalam *working memory*.

2.2.1.3. Inference Engine

Sebagai komponen yang berfungsi sebagai pengolah pengetahuan, *inference engine* akan bernalar seperti otak manusia dengan melakukan sejumlah operasi atau menggabungkan berbagai fakta dan aturan yang telah disediakan di dalam *knowledge base* untuk mengembangkan dan menarik fakta – fakta baru, sehingga dapat memberikan konklusi dari pengetahuan dan masalah yang tersedia. *Inference engine* secara langsung melakukan pencarian ke dalam *knowledge base* untuk menemukan fakta atau *rule* yang dibutuhkan. Setelah proses *inference engine* selesai, maka sistem akan memberikan masukan atau jawaban kepada user.

Teknik *inference engine* untuk menemukan fakta atau *rule* yang dibutuhkan ada banyak metode, antara lain *forward chaining* (atau *data driven*) dan *backward chaining* (atau *goal driven*). Jika suatu *knowledge base* memiliki sedikit *condition* dan banyak *conclusion*, maka *forward chaining* merupakan proses penalaran terbaik. Dan sebaliknya, jika suatu *knowledge base* memiliki banyak *condition* dan sedikit *conclusion* maka sebaiknya proses penalarannya menggunakan *backward chaining*.

2.2.1.4. Forward Chaining

Forward chaining adalah proses penalaran dari sekumpulan data atau fakta menuju ke arah suatu kesimpulan yang telah dideskripsikan sebelumnya. *Inference engine* mulai dari *rule* pertama pada level pertama. Kemudian mengecek setiap *rule* apakah ada yang sudah *fired*. Jika belum ada rule yang di*fired*, *inference engine* mengecek semua *IF Condition*. Jika semua *condition* dari suatu *rule* terpenuhi semua maka *rule* tersebut di*fired*, kemudian *THEN Conclusion* disimpan dalam *Working Memory*, dan *inference engine* terus

mengevaluasi *rule* berikutnya sampai tidak ada *rule* yang dapat di*fired* lagi atau sudah mencapai *goal*.

Dengan *knowledge base* seperti pada contoh diatas, maka *inference engine* mulai dengan fakta atau kondisi *engine_type* dan menunggu *user* untuk me*respon* jawabannya. Setelah *user* me*respon*, misalnya *user* menjawab *jet*, maka *inference engine* mulai mencari *rule* mana saja yang mengandung fakta atau kondisi diatas pada bagian *IF condition*. Jika *inference engine* sudah menemukan *rule* tersebut, maka *inference engine* mengecek apakah *IF condition* dari *rule* tersebut sudah terpenuhi semua atau *IF condition* bernilai *true*. Jika belum, maka *rule* tersebut belum dapat di*fired*, maka *inference engine* memulai lagi dengan fakta atau kondisi berikutnya, yaitu *wing_position* dan menunggu *user* untuk me*respon* jawabannya. Ketika *user* me*respon*, misalnya *user* menjawab *low*, maka *inference engine* mulai mencari *rule* mana saja yang mengandung fakta atau kondisi diatas pada bagian *IF condition*. Dan jika *inference engine* sudah menemukan *rule* tersebut, maka *inference engine* mengecek apakah *IF condition* dari *rule* tersebut sudah terpenuhi semua atau *IF condition* bernilai *true*. Sewaktu *IF condition* bernilai *true*, maka *rule* tersebut dapat di*fired*, *attributte* dan *value* yang ada pada bagian *THEN conclusion* disimpan dalam *working memory*. Proses *inference engine* ini berulang sampai tidak ada lagi *rule* yang dapat di*fired* atau sudah mencapai *goal*.

Tehnik *forward chaining* ini sangat baik untuk diterapkan pada *knowledge* yang memiliki banyak *conclusion*. Aplikasi sistem pakar yang berkategori *planning* dan *design* biasanya menerapkan tehnik *forward chaining* ini.

Efisiensi tehnik *forward chaining* ini bergantung pada *knowledge base* dan *user interface* yang digunakan agar *user* dan sistem fokus pada masalah tersebut.

2.2.1.5. Backward Chaining

Backward chaining bekerja berlawanan arah dengan *forward chaining*. Daripada mengambil data dan menunggu apa yang akan keluar berikutnya, *backward chaining* mencoba untuk meng*fire* suatu *rule* secara langsung.

Langkah pertama *backward chaining* adalah mendaftar semua *rule* yang mengandung *attributte goal* pada bagian *THEN conclusion*. Kemudian *inference engine* akan mencoba meng*fire* setiap *rule* tersebut, sampai salah satu dari *rule* tersebut berhasil di*fired*. Untuk melakukan itu, *inference engine* mengambil *rule* pertama dalam daftar untuk mencoba meng*fire rule* tersebut. Untuk meng*fire rule* tersebut, maka *inference engine* perlu untuk mengetahui bahwa semua *if-clause* sudah ada dalam *working memory*. Diasumsikan bahwa tidak ada informasi yang terdapat dalam *working memory*. Ini menandakan bahwa tidak ada *if-clause* yang diketahui. Karena itu, *inference engine* mencoba dan sampai pada *if-clause* secara tidak langsung.

Dengan *knowledge base* seperti pada contoh *knowledge base* diatas, maka *inference engine* mendaftar semua *rule* yang bagian *THEN conclusion* mengandung *attributte goal*. Karena semua *rule* pada contoh *knowledge base* diatas mengandung *attributte goal*, maka *inference engine* akan mencoba untuk meng*fire rule* pertama. Untuk meng*fire rule* tersebut, maka *IF condition* harus bernilai benar. Fakta atau kondisi pertama dari *rule* tersebut adalah *engine_type*. Karena fakta atau kondisi ini belum ada dalam *working memory* maka perlu dikonsultasikan dengan *user*. Setelah *user* me*respon* dengan menjawab *jet*, maka *rule* pertama tidak dapat di*fired* karena fakta atau kondisi pertamanya bernilai salah. Kemudian *inference engine* mengambil *rule* kedua dalam daftar dan mencoba untuk meng-*fire*. Fakta atau kondisi pertama dari *rule* kedua adalah *engine_type*, dan fakta atau kondisi ini sudah ada dalam *working memory*, maka *inference engine* mengambil fakta atau kondisi berikutnya yaitu *wing_position*. Karena fakta atau kondisi ini belum ada dalam *working memory* maka perlu dikonsultasikan dengan *user*. Setelah *user* me*respon* dengan menjawab *low*, maka *rule* kedua dapat di*fired* karena fakta atau kondisi keduanya bernilai benar. Kemudian masukkan *attributte* dan *value* yang ada pada bagian *THEN conclusion*, yaitu *plane = B747* ke dalam *working memory*. Ketika meng*fire rule* tersebut, maka *inference engine* akan berhenti karena berhasil meng*fired* salah satu rule dalam daftar, kemudian memberitahukan *user* bahwa *goal* telah dicapai.

Tehnik *backward chaining* ini sangat baik untuk diterapkan pada *knowledge* yang memiliki sedikit *conclusion*. Aplikasi sistem pakar yang

berkategori *diagnose, control,* dan *monitoring,* biasanya menerapkan tehnik *backward chaining* ini.

2.2.1.6. User Interface

Setelah *knowledge base* terbentuk, maka dibuatlah bagian *user interface*. Bagian ini berlaku sebagai sebuah penghubung antara pemakai dengan sistem dengan cara menerima data ataupun memberikan masukan/jawaban tertentu dengan penalaran yang dimilikinya. Pada tingkat minimum, *user interface* ini berisi pesan pembukaan dan pesan penutupan, serta tampilnya beberapa pertanyaan selama konsultasi. Semua pesan, pertanyaan, dan jawaban yang muncul tergantung sepenuhnya dari *knowledge base* yang tersedia.

2.2.1.7. Explanation Subsystem

Bagian ini menjelaskan saat mana *user* perlu mengetahui apakah alasan diberikannya sebuah solusi. Sekali sistem telah menemukan sebuah solusi, *user* mungkin akan menanyakan suatu penjelasannya. Bagian ini juga secara konkrit membedakan sebuah sistem pakar dengan sistem aplikasi konvensional, karena pada pemrograman konvensional tidak menyediakan informasi tambahan mengapa atau dimana sebuah solusi diperoleh.

2.2.2. Kategori Aplikasi Sistem Pakar

Dewasa ini sistem pakar telah banyak dikembangkan dalam berbagai bidang mulai dari matematika, kedokteran, teknik, kimia, geologi, pertanian, pendidikan sampai hukum. Dari sejumlah aplikasi sistem pakar yang telah dikembangkan tersebut, sesungguhnya sistem pakar dikategorikan sebagai berikut ini :

a. Interpretation

Memberikan gambaran tentang sekumpulan data mentah yang biasanya diperoleh melalui sensor.

Contoh : pengenalan kata/ucapan, analisa citra/pengenalan pola, pembuatan peta dan analisa intelegensia seseorang.

b. Prediction

Memberikan kesimpulan mengenai akibat yang mungkin ditimbulkan dari sejumlah situasi yang diberikan.

Contoh : prakiraan cuaca, ramalan keuangan, ramalan panen.

c. Diagnosis

Menentukan penyebab kegagalan suatu sistem dalam situasi yang kompleks yang didasarkan pada observasi terhadap gejala – gejala yang dapat diamati.

Contoh : diagnosis penyebab suatu penyakit, kerusakan elektronik, analisa keuangan.

d. Design

Menentukan konfigurasi yang cocok dari komponen – komponen yang ada dalam sebuah sistem sehingga unjuk kerja yang memuaskan dapat diperoleh walaupun didalamnya terdapat sejumlah keterbatasan.

Contoh : layout circuit, penyusunan anggaran belanja, arsitektur rumah dan bangunan.

e. Planning

Mendapatkan urutan tindakan yang harus dilakukan untuk mencapai sasaran yang telah ditentukan sebelumnya dari suatu kondisi awal tertentu.

Contoh : perencanaan strategis dalam manajemen, penjadwalan proses.

f. Monitoring

Membandingkan perilaku yang dapat diamati pada suatu sistem dengan perilaku yang diharapkan atau ditentukan sebelumnya untuk mengenal lebih banyak variasi perilaku di dalamnya.

Contoh : manajemen pengawasan, pengendalian instalasi nuklir.

g. *Debugging* dan *Repair*

Penentuan dan implementasi perbaikan atau pertolongan pada kegagalan suatu sistem.

Contoh : uji coba software komputer, reparasi mesin.

h. Instruction

Mendeteksi dan memperbaiki kekurangan perilaku siswa dalam memahami suatu bidang ilmu tertentu.

Contoh : CAI untuk kuliah remedial, CAI untuk tutorial.

i. Classification

Menentukan kategori dari sejumlah kriteria yang diberikan.

Contoh : auditing, forecasting (peramalan), perencanaan program.

j. Control

Pengaturan perilaku kerja sistem dalam suatu lingkungan yang kompleks, yang termasuk didalamnya adalah penafsiran, prakiraan, pengawasan dan perbaikan perilaku kerja sistem tersebut.

Contoh : pengawasan jadwal penerbangan pesawat pada suatu bandara, pengawasan proses manufacturing, manajemen dalam pertempuran atau peperangan.

2.2.3. Representasi Pengetahuan Pada Sistem Pakar

Representasi pengetahuan merupakan suatu metode untuk mengkodekan pengetahuan dalam sebuah basis pengetahuan sistem pakar. Teknik – teknik representasi ini berbeda dengan representasi pada sebuah software konvensional, dimana sebuah program harus memproses data, tetapi sebuah sistem pakar harus memproses pengetahuan.

Representasi sebuah pengetahuan ada beberapa macam cara, diantaranya adalah sebagai berikut :

a. Production Rule/Basis Rule

Jenis representasi ini merupakan metode yang paling banyak dipakai dalam representasi pengetahuan, karena kesederhanaan strukturnya. Dimana dalam representasi ini *knowledge* yang didapat akan disajikan dalam bentuk aturan – aturan (*rules*) dalam format : *IF…THEN…*

b. Jaringan Semantik

Jaringan semantik merupakan model representasi berupa jaringan pengertian diantara beberapa fakta.

c. Frame

Frame adalah bentuk representasi pengetahuan yang bertujuan untuk memberikan definisi suatu objek tertentu dengan jalan menjelaskan segala sesuatu yang berkaitan dengan objek tersebut.

d. Script

Script adalah representasi pengetahuan secara terstruktur ang menjelaskan tentang urutan kejadian dalam suatu konteks yang spesifik.

2.3. Shell Sistem Pakar

Pengembangan *shell* sistem pakar seperti halnya pengembangan sebuah sistem aplikasi yang implementasinya dapat dilakukan dengan menggunakan bahasa pemrograman seperti *Pascal*, *Borland Delphi*, dan lain sebagainya.

Apabila implementasi sebuah sistem pakar dilakukan dengan menggunakan bahasa pemrograman, maka seluruh komponen sistem pakar mulai dari *knowledge base editor*, *inference engine* sampai ke *user interface* harus dibuat sendiri. Selain itu, implementasi dari sistem pakar dapat dilakukan dengan bahasa pemrograman yang memang khusus dirancang untuk pengembangan sebuah sistem pakar, misalnya *Prolog*.

Lain halnya jika *user* menggunakan *shell* sistem pakar untuk membangun suatu sistem pakar, maka kemampuan programming tidak begitu diperlukan. Hal ini karena dalam *shell* sistem pakar tersebut *knowledge base editor, inference engine* sampai ke *user interface* sudah tersedia, sehingga seorang *user* tidak perlu tahu bagaimana proses penalaran yang cukup sulit untuk menghasilkan rangkaian pertanyaan yang tepat pada sebuah konsultasi. Biasanya *user* hanya perlu membangun *knowledge base* sesuai dengan area permasalahan yang dihadapi, menentukan jenis *chaining* yang akan digunakan, *backward* atau *forward.*

Saat ini, *shell* sistem pakar menjadi cara yang paling banyak digunakan untuk mengembangkan sistem pakar skala kecil melalui mikro kompuer. Contoh *shell* sistem pakar yang terdapat di pasaran yaitu *VP Expert*. Besar kecilnya ukuran dilihat dari jumlah *rule* di dalamnya. Dibawah 100 *rule* biasanya masih disebut kecil, 100 – 200 *rule* disebut sedang dan lebih dari itu disebut ukuran besar. Sedangkan kompleksitas biasanya menunjukkan pada rumitnya hubungan antara masing – masing kelompok *rule.*

Suatu *shell* sistem pakar juga memiliki keterbatasan dan kelemahan. Karena *shell* sistem pakar ini tidak fleksibel, maka akan menjadi sangat sulit

untuk membangun suatu sistem pakar dibawah standart dari *shell* tersebut. *Shell* sistem pakar biasanya hanya digunakan untuk memecahkan masalah umum yang tidak begitu komplek , sehingga untuk menyelesaikan masalah khusus yang komplek sebaiknya menggunakan bahasa pemrograman. *Shell* sistem pakar merupakan *end user tools,* dan kegunaannya untuk mengatasi masalah yang berkaitan dengan *end user computing*.

2.4. Faktor Keyakinan

Dalam beberapa keadaan, sebuah fakta mungkin dipercayai benar, tetapi tidak dengan keyakinan yang sempurna. Dalam beberapa kasus, sebuah nilai mewakili tingkat dari kepercayaan yang dihubungkan dengan fakta/*rule*. Nilai ini disebut faktor keyakinan (*certainty factors*/CNF) atau derajad kepercayaan (*degree of belief*).

Faktor keyakinan adalah sebuah angka yang diberikan kepada suatu nilai yang menunjukkan tingkat keyakinan pada nilai tersebut dari pembuat/pakar atau pengguna. Dalam *VP Expert* nol menunjukkan tidak adanya keyakinan dan 100 menunjukkan keyakinan total pada suatu nilai. Program sistem pakar yang lain mungkin menggunakan skala yang berbeda, seperti –1 hingga +1, -100 hingga +100, dan lain sebagainya.

Faktor keyakinan bukan merupakan *probababilitas* statistik, meskipun sering dianggap sebagai *prosentase*. Faktor keyakinan dapat berdasarkan *intuisi subjektif*/kriteria *objektif* untuk menggambarkan bagaimana besarnya keyakinan seseorang akan suatu nilai adalah benar. Jika faktor keyakinan tidak digunakan, maka yang akan dipakai adalah 100 sebagai nilai standart. Pada contoh *knowledge base* dibawah ini memberikan gambaran bagaimana faktor keyakinan digunakan oleh pembuat/pakar. Dengan mengamati klausa *THEN* pada *rule*, terbukti bahwa pembuat/pakar memiliki tingkat keyakinan yang beragam terhadap konklusi – konklusi tersebut. hal ini bisa dilihat pada contoh basis pengetahuan dibawah ini. Angka – angka yang didahului kata kunci CNF menunjukkan tingkat keyakinan pembuat/pakar terhadap *rule* yang bersangkutan.

Contoh Basis Pengetahuan Dengan Faktor Keyakinan :

+

+

+

IF c_prompt = no AND fan = no THEN ps = defective CNF 80;
IF c_prompt = no AND fan = yes THEN ps = non_defective CNF 90;

+

+

+

2.5. Lexical Analyzer

Metode yang digunakan untuk membaca *knowledge base* yang telah disimpan dalam bentuk *file* teks, maka digunakan metode *lexical analysis. Lexical analysis* adalah suatu proses analisa yang membagi kalimat menjadi beberapa kata berdasarkan *pattern* tertentu. Tugas *lexical analysis* adalah membaca *file* teks dan menghasilkan sekumpulan *token*, kemudian dicocokkan dengan *pattern* yang diinginkan. Didalam *lexical analysis* terdapat dua bagian yang penting, yaitu scanner dan parser.

2.5.1. Scanner

Scanner menganalisa suatu *knowledge base* yang disimpan dalam bentuk *file* teks menjadi besaran leksik/*token*. Tugas *scanner* dapat dirangkum sebagai berikut :

- Merunut karakter demi karakter
- Mengenali besaran leksik
- Mentransformasi menjadi sebuah *token*
- Mengirimkan *token*
- Membuang / mengabaikan *blank*
- Menangani tabel simbol

Untuk mempercepat kerja dari *lexical analysis*, maka diperlukan *buffer pairs* untuk mengidentifikasi *token* dari tiap *input*. Teknik *buffer pairs* ini menggunakan suatu *buffer* untuk menampung tiap karakter dari *input* yang ada, kemudian menggunakan *pointer* untuk membaca isi dari *buffer*. Dari sinilah

lexical analysis bekerja untuk membagi *input* yang masuk menjadi kata atau tanda operator, kemudian menentukan *token* tiap bagian tersebut.

	I	F		c	_	p	r	o	m	p	t		…

Gambar 2.3. Buffer Pairs

Dan jika jumlah karakter yang ada dalam *input*an melebihi panjangnya *buffer*, maka *buffer* tersebut akan diisi sepenuhnya, kemudian *lexical analysis* mulai membaca sampai akhir dari *buffer*. Setelah itu, isi *buffer* akan dikosongkan dan diisi dengan sisa *input*an tadi, kemudian mulai dibaca lagi, dan begitu seterusnya.

2.5.2. Parser

Proses *parsing* adalah tahapan memeriksa kebenaran dan urutan kemunculan *token*. *Recursive descent parser* adalah salah satu cara untuk mengaplikasikan tata bahasa bebas konteks untuk melakukan analisa sintaksis suatu *knowledge base* yang tersimpan dalam bentuk *file* teks. Disini simbol terminal maupun simbol variabelnya sudah bukan sebuah karakter, tetapi berupa besaran leksik sebagai simbol terminalnya dan besaran sintaks sebagai simbol variabelnya. Ciri dari *recursive descent parser* yang menonjol adalah secara rekursif menurunkan semua variabel dari awal sampai bertemu terminal dan tidak pernah mengambil *token* secara mundur. Ciri lain dari *recursive descent parser* adalah sangat bergantung pada scanner dalam mengambil *token*.

Jika *lexical analysis* mendapat *input* seperti ini "*IF c_prompt = yes THEN reco = do_not_need_PC_CARE*; ", maka *lexical analysis* akan membagi *input* tersebut menjadi beberapa kata atau tanda operator, kemudian menentukan *token* dari setiap kata atau tanda operator, seperti terlihat pada tabel dibawah ini.

Tabel 2.1. Contoh Token Dan Lexeme

Token	***Lexeme***
t_if	IF
t_attributte	C_prompt
t_operator (Assignment Operator)	=
t_value	Yes
t_then	THEN
t_attributte	Reco
t_operator (Assignment Operator)	=
t_value	Do_not_need_PC_CARE
t_ttkm	;