BAB 2

TINJAUAN PUSTAKA

2.1 Profil Perusahaan

2.1.1 Sejarah

Klinik dr Iin Hartinah berdiri pada tahun 2009. Klinik ini Berada di Jl. Karang Baru No 12 kecamatan Palimanan Kabupaten Cirebon. Klinik dr Iin Hartinah berawal bernama klinik Sehat Bersama karena Banyak pasien lebih mengenal dr Iin Hatinah Sebagai Dokter Prateknya maka berubah nama menjadi Klinik dr Iin Hartinah.

Pendirian klinikdr Iin Hartinah sendiri sudah mendapatkan beberapa izin dan persetujuan dari pihak terkait, yaitu :

1. Surat Izin Penyelenggaraan Klinik Umum dari Dinas Kesehatan Cirebon Nomor : 445/73-10/K.UMUM/DKK/III/2010
2. Surat Izin Apotik dari Dinas Kesehatan Cirebon Nomor : 442/90-10/APT/DKK/IX/2011

Klinik dr Iin Hartinah memilik banyak pasien, setiap harinya ada 20 sampai 50 pasien yang ingin berobat atau berkonsultasi dengan dokter. Untuk menangani pasien yang banyak Klinik dr Iin Hartinah memiliki tenaga medis: 1 dokter, 2 perawat dan 1 apoteker.

2.1.2 Visi Misi

Visi:

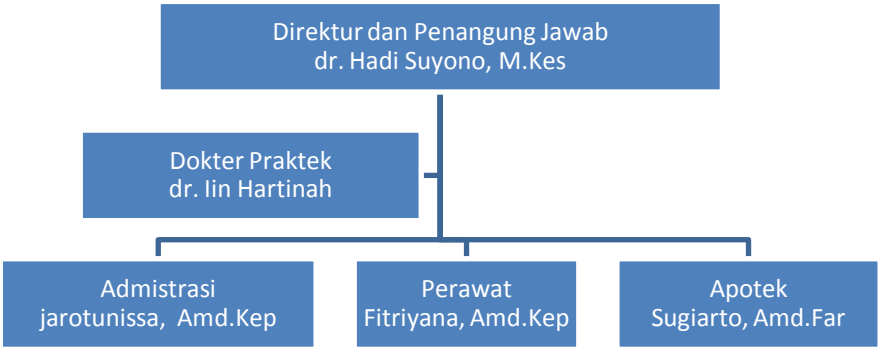
"Memberikan pelayanan kesehatan yang prima kepada masyarakat secara benar, bermutu dan professional dengan pendekatan kekeluargaan"

Misi:

1. Memberikan pelayanan kesehatan yang profesional kepada seluruh lapisan masyarakat
2. Menyediakan sarana dan pelayanan kesehatan tingkat dasar yang berbasis kompetensi
3. Meningkatkan derajat kesehatan masyarakat melalui pelayanan yang bermutu dengan biaya terjangkau
4. Mengembangkan kemitraan dengan masyarakat

2.1.3 Struktur Organisasi

STRUKTUR ORGANISASI KLINIK DR IIN HARTINAH



Gambar 2.1 Struktur Organisasi Klinik Dr Iin Hartinah

2.1.4 Job Desk

A. Direktur dan Penangung Jawab

1. Mengatur dan bertanggung jawab terhadapsemua yang ada di klinik
2. Menyusun perencanaan, kebutuhan dan operasional
3. Mengevaluasi kegiatan pelayanan kesehatan di klinik
4. Melakukan monitoring internal dan penilaian berkala pada seluruh kinerja
5. Membuat SOP, Sistem, Protap dan kebijakan

B. Dokter Praktek

1. Melakukan pemeriksaan, diagnosis dan terapi terhadap keluhan spesifik sesuai bidang keahliannya
2. Memberikan pelayanan konsultasi terhadap pasien
3. Memberikan resep terhadap pasien
4. Memberikan penyuluhan terhadap pasien
5. Memberikan pengobatan, penangan dan terapi terhadap yg membutuhkan pasien

C. Admistrasi

1. Mekakukan pendataan terhadap pasien
2. Mengelola pelaksanaan kegiatan di lingkungan tatausaha, keuangan
3. Membantu kepala klinik dalam pengawasan tata tertib
4. Melakukan evaluasi berkala terhadap fasilitas yang diperlukan, seperti alat-alat medis,dan kebutuhan operasional lainnya
5. Membuat laporan mingguan, bulanan, tahunan

D. Perawat

1. Bertugas di pelayanan darurat
2. pembantu dokter selama memeriksa pasien
3. Menjamin terlaksanannya proses pelayanan kesehatanyang efektif dan efisien
4. Melaksanakan koordinasi medisdan penunjang dengan tim yang lain

E. Apotek

1. Memeriksa dan menganalisis hasil diagnosa dokter
2. Membaca dan menganalisis hasil resep dari dokter yg diberikan dokter
3. Mengadakan kesediaan dan regulasi obat-obatan dengan bekerja sama dengan bidang logistik

2.1.5 Logo



Gambar 2.2 Logo Klinik

2.2 Landasan Teori

Landasan teori merupakan penjelasan berbagai konsep dasar dan teoriteori yang berkaitan dalam pembangunan aplikasi *chatbot* dokter *virtual* dengan pendekatan *Natural Language Processing.*

2.2.1 Kecerdasan Buatan

Kecerdasan Buatan adalah salah satu cabang ilmu pengetahuan berhubungan dengan pemanfaatan mesin untuk memecahkan persoalan yang rumit dengan cara yang lebih manusiawi. Hal Ini biasanya dilakukan dengan mengikutI atau mencontoh karakteristik dan analogi berpikir dari kecerdasan atau *Inteligensia* manusia, dan menerapkannya sebagai algoritma yang dikenal oleh komputer. Dengan suatu pendekatan yang kurang lebih *fleksibel* dan efisien dapat diambil tergantung dari keperluan, yang mempengaruhi bagaimana wujud dari perilaku kecerdasan buatan. AI biasanya dihubungkan dengan Ilmu Komputer, akan tetapi juga terkait erat dengan bidang lain seperti Matematika, Psikologi, Pengamatan, Biologi, Filosofi, dan yang lainnya. Kemampuan untuk mengkombinasikan pengetahuan dari semua bidang ini pada akhirnya akan bermanfaat bagi kemajuan dalam upaya menciptakan suatu kecerdasan buatan.

Pengertian lain dari kecerdasan buatan adalah bagian ilmu komputer yang membuat agar mesin komputer dapat melakukan pekerjaan seperti dan sebaik yang dilakukan manusia. Pada awal diciptakannya, komputer hanya difungsikan sebagai alat hitung saja. Namun seiring dengan perkembangan jaman, maka peran komputer semakin mendominasi kehidupan manusia. Komputer tidak lagi hanya digunakan sebagai alat hitung, lebih dari itu, komputer diharapkan untuk dapat diberdayakan untuk mengerjakan segala sesuatu yang bisa dikerjakan oleh manusia.

Menurut beberapa ahli kecerdasan buatan didefinisikan sebagai berikut

1. Menurut Rich and knight [1991] [2] : ”Kecerdasan buatan (*artificial intellegence*) merupakan sebuah studi tentang bagaimana membuat komputer melakukan hal-hal yang pada saat ini dapat dilakukan lebih baik oleh manusia”.
2. Menurut Andri Kristanto (2003) : Kecerdasan buatan merupakan bagian dari ilmu pengetahuan komputer yang khusus ditujukan dalam perancangan otomatisasi tingah laku cerdas dalam sistem kecerdasan komputer. Kecerdasan buatan dilihat dari berbagai sudut pandang adalah sebagai berikut:

1. Sudut pandang Kecerdasan (*Intellegence*)
   Kecerdasan buatan adalah bagaimana membuat mesin yang ”cerdas” dan dapat melakukan hal-hal yang sebelumnya dapat dilakukan oleh manusia.
2. Sudut pandang Penelitian
   Studi bagaimana membuat agar komputer dapat melakukan sesuatu sebaik yang dilakukan oleh manusia.

Pengertian lain dari kecerdasan buatan adalah bagian ilmu komputer yang membuat agar mesin komputer dapat melakukan pekerjaan seperti dan sebaik yang dilakukan manusia. Pada awal diciptakannya, komputer hanya difungsikan sebagai alat hitung saja. Namun seiring dengan perkembangan jaman, maka peran komputer semakin mendominasi kehidupan manusia. Komputer tidak lagi hanya digunakan sebagai alat hitung, lebih dari itu, komputer diharapkan untuk dapat diberdayakan untuk mengerjakan segala sesuatu yang bisa dikerjakan oleh manusia.

Agar komputer bisa bertindak seperti dan sebaik manusia, maka komputer juga harus diberi bekal pengetahuan dan mempunyai kemampuan untuk menalar. Untuk itu AI akan mencoba untuk memberikan beberapa metoda untuk membekali komputer dengan kedua komponen tersebut agar komputer bisa menjadi mesin pintar.

Dari beberapa perspektif, AI dapat dipandang sebagai:

1) Dari perspektif kecerdasan, AI adalah bagaimana membuat mesin yang cerdas dan dapat melakukan hal-hal yang sebelumnya hanya dapat dilakukan manusia.

2) Dari perspektif bisnis, AI adalah sekelompok alat bantu *(tools)* yang berdayaguna dan metodologi yang menggunakan alat-alat bantu tersebut untuk menyelesaikan masalah-masalah bisnis.

3) Dari perspektif pemrograman, AI meliputi studi tentang pemrograman simbolik, pemecahan masalah, dan proses pencarian *(search)*.

4) Dari perspektif penelitian

a) Riset tentang AI dimulai pada awal tahun 1960-an, percobaan pertama adalahmembuat program permainan catur, membuktikan teori, dan *general problem solving*.

b) *Artificial intelligence* adalah nama pada akar dari studi area.

Ada dua hal yang sangat mendasar mengenai penelitian-penelitian AI, yaitu *knowledge representation* (representasi pengetahuan) dan *search* (pelacakan). Para peneliti AI terus mengembangkan berbagai jenis teknik baru dalam menangani sejumlah permasalahan yang tergolong ke dalam AI seperti *vision* dan percakapan, pemrosesan bahasa alami, dan permasalahan khusus seperti diagnosa medis.

2.2.2 Konsep Kecerdasan Buatan

Ada beberapa konsep yang harus dipahami dalam kecerdasan buatan, diantaranya (kusrini, 2009) [4] :

1. *Turing Test* – Metode Pengujian Kecerdasan

   Turing Test merupakan sebuah metode pengujian kecerdasan yang dibuat oleh Alan Turing. Proses uji ini melibatkan seorang penanya (manusia) dan dua obyek yang ditanyai. Yang satu adalah seorang manusia dan yang satunya adalah sebuah mesin yang akan diuji. Penanya tidak dapat melihat langsung kepada obyek yang ditanyai. Penanya diminta untuk membedakan mana jawaban komputer dan mana jawaban manusia berdasarkan jawaban kedua obyek tersebut. Jika penanya tidak dapat membedakan mana jawaban mesin dan mana jawaban manusia maka *Turing* berpendapat bahwa mesin yang diuji tersebut dapat diasumsikan Cerdas.

2. Pemrosesan Simbolik

   Komputer semula didesain untuk memproses bilangan atau angka-angka (pemrosesan *numerik*). Sementara manusia dalam berfikir dan menyelesaikan masalah lebih bersifat simbolik, tidak didasarkan pada sejumlah rumus atau melakukan

komputasi matematika. Sifat penting dari AI adalah bahwa AI merupakan bagian dari ilmu komputer yang melakukan proses secara simbolik dan *non-algoritmik* dalam penyelesaian masalah.

3. Heuristic

   Istilah *heuristic* diambil dari bahasa Yunani yang berarti menemukan. Heuristic merupakan suatu *strategi* untuk melakukan proses pencarian (*search*) ruang problem secara selektif, yang memandu proses pencarian yang kita lakukan disepanjang jalur yang memiliki kemungkinan sukses paling besar.

4. Penarikan Kesimpulan (*Inferencing*)

   AI mencoba membuat mesin memiliki kemampuan berfikir atau mempertimbangkan (*reasoning*). Kemampuan berfikir (*reasoning*) termasuk didalamnya proses penarikan kesimpulan (*inferencing*) berdasarkan fakta-fakta dan aturan dengan menggunakan metode *heuristic* atau pencarian lainnya.

5. Pencocokan Pola (*Pattern Matching*)

   AI bekerja dengan metode pencocokan pola (*pattern matching*) yang berusaha untuk menjelaskan objek, kejadian (*event*) atau proses, dalam hubungan logika atau komputasional.

2.2.3 Lingkup Utama Kecerdasan Buatan

Lingkup utama kecerdasan buatan adalah :

1. Sistem pakar (*Expert System*). Komputer digunakan sebagai saran untuk menyimpan pengetahuan para pakar. Dengan demikian komputer akan memiliki keahlian untuk menyelesaikan masalah dengan meniru keahlian yang dimiliki para pakar
2. Pengolahan bahasa alami (*Natural Language Processing*). Dengan pengolahan bahasa alami ini diharapkan *user* mampu berkomunikasi dengan komputer dengan menggunakan bahasa sehari-hari.

3. Pengenalan ucapan (*Pattern Recognition*). Melalui pengenalan ucapan diharapkan manusia mampu berkomunikasi dengan komputer dengan menggunakan suara.
4. Robotika dan Sistem *sensor*.
5. *Computer vision*, mencoba untuk dapat mengintrepetasikan gambar atau objek-objek tampak melalui komputer.
6. *Intelligent Computer aid Instruction*. Komputer dapat digunakan sebagai tutor yang dapat melatih dan mengajar.

2.2.4 *Pattern Matching*

Pattern Matching atau Pencocokan Pola adalah suatu metode yang digunakan untuk mencocokan suatu pola tertentu (kumpulan huruf) dengan suatu kumpulan kata (*teks*) atau string. Pada bidang sains komputer metode *pattern matching* sangat banyak digunakan antara lain Editor *Teks*, Mesin Pencari *Web*, Analisis Gambar dan lain-lain. *String* dapat kita asumsikan sebagai kumpulan dari beberapa karakter yang membentuk suatu kesatuan.

2.2.5 *Brote force*

Algoritma *Brute Force* merupakan algoritma pencocokan *string* dengan melakukan pencarian semua kemunculan string pattern yang berada di *string* teks.

Cara Kerja Algoritma Brute Force :

1. Mula-mula pattern dicocokan pada awal teks.
2. Dengan bergerak dari kiri ke kanan, bandingkan setiap karakter di dalam pattern dengan karakter yang bersesuaian di dalam teks sampai :
   a. Semua karakter yang dibandingkan cocok atau sama (pencarian berhasil), atau
   b. Dijumpai sebuah ketidakcocokan karakter (pencarian belum berhasil)
3. Bila pattern belum ditemukan kecocokannya dan teks belum habis, geser pattern satu karakter ke kanan dan ulangi langkah 2.

Persoalan pencarian *string* dirumuskan sebagai berikut :

1. Teks (*text)*, yaitu (*long*) string yang panjangnya n karakter
2. *Pattern*, yaitu *string* dengan m karakter (m < n) yang akan dicari di dalam teks.

Contoh cara kerja algoritma Brute Force ini adalah sebagai berikut :

Teks : AABD CAECCA ABDABCABC BBABCABE

Pattern : ABCABE

	TEKS																												
	A	A	B	D		C	A	E	C	C	A		A	B	D	A	B	C	A	B	C		B	B	A	B	C	A	B	E
1	A	B	C	A	B	E																							
2		A	B	C	A	B	E																						
3			A	B	C	A	B	E																					
4				A	B	C	A	B	E																				
5					A	B	C	A	B	E																			
6						A	B	C	A	B	E																		
7							A	B	C	A	B	E																	
8								A	B	C	A	B	E																
9									A	B	C	A	B	E															
10										A	B	C	A	B	E														
11											A	B	C	A	B	E													
12												A	B	C	A	B	E												
13													A	B	C	A	B	E											
14														A	B	C	A	B	E										
15															A	B	C	A	B	E									
16																A	B	C	A	B	E								
17																	A	B	C	A	B	E							
18																		A	B	C	A	B	E						
19																			A	B	C	A	B	E					
20																				A	B	C	A	B	E				
21																					A	B	C	A	B	E			
22																						A	B	C	A	B	E		
23																							A	B	C	A	B	E	
24																								A	B	C	A	B	E
25																									A	B	C	A	B	E

Gambar 2.3 Contoh Algoritma Brute Force

2.2.6 *Fuzzy String Matching*

Fuzzy string matching merupakan pencocokan pola *string* secara samar, maksudnya pencocokan *string* dimana *string* yang dicocokan memilik kemiripan dimana keduanya memiliki susunan karakter yang berbeda (mungkin jumlah string atau urutanya) tetapi *string string* tersebut memiliki kemiripan, baik kemiripan tekstuan / penulisan (*approximete string matching*) atau kemiripan ucapan (*phonetic string matching*). Metode *fuzzy string matching* diarahkan untuk mencari nilai dari beberapa *string* yang mendekati dan tidak hanya mengasilkan cocok atau tidak cocok

Pencocokan string (*fuzzy string matching*) secara garis besar dapat dibedakan menjadi dua yaitu:

A. *Exact string matching*, merupakan pencocokan string secara tepat dengan susunan karakter dalam *string* yang dicocokkan memiliki jumlah maupun urutan karakter dalam *string* yang sama.

Contoh : kata sehat akan menunjukkan kecocokan hanya dengan kata sehat.

B. *Inexact string matching*, merupakan pencocokan *string* secara samar, maksudnya pencocokan string dimana *string* yang dicocokkan memiliki kemiripan dimana keduanya memiliki susunan karakter yang berbeda (mungkin jumlah atau urutannya) tetapi *string-string* tersebut memiliki kemiripan baik kemiripan tekstual/penulisan (*approximate string matching*) atau kemiripan ucapan (*phonetic string matching*). *Inexact string matching* masih dapat dibagi lagi menjadi dua yaitu:

1. Pencocokan string berdasarkan kemiripan penulisan (*approximate string matching*)

merupakan pencocokan string dengan dasar kemiripan dari segi penulisannya (jumlah

karakter, susunan karakter dalam dokumen). Tingkat kemiripan ditentukan dengan jauh tidaknya beda penulisan dua buah *string* yang dibandingkan tersebut dan nilai tingkat kemiripan ini ditentukan oleh pemrogram *(programmer*). Contoh: rabu dengan rabo, memiliki jumlah karakter yang sama tetapi ada dua karakter yang berbeda. Jika perbedaan satu karakter ini dapat ditoleransi sebagai sebuah kesalahan penulisan maka satu *string* tersebut dikatakan cocok dengan tingat nilai kecocokan bisa disuaikan.

2. Pencocokan *string* berdasarkan kemiripan ucapan (*phonetic string matching*) merupakan pencocokan string dengan dasar kemiripan dari segi pengucapannya meskipun ada perbedaan penulisan dua *string* yang dibandingkan tersebut. Contoh jumat dengan jum”at dari tulisan berbeda tetapi dalam pengucapannya mirip sehingga dua string tersebut dianggap cocok. Contoh yang lain adalah *step*, dengan *steppe*, *sttep, stepp, stepe. Exact string matching* bermanfaat jika pengguna ingin mencari *string* dalam dokumen yang sama persis dengan *string* masukan. Tetapi jika pengguna menginginkan pencarian *string* yang mendekati dengan *string*

masukan atau terjadi kesalahan penulisan *string* masukan maupun dokumen objek pencarian, maka *inexact string matching* yang bermanfaat. Dari contoh yang diberikan, sebenarnya *aphonetic string matching* juga dapat dimanfaatkan untuk *approximate string matching* dengan batasandua *string* yang dicocokkan masih memiliki kemiripan ucapan. *Phonetic string matching* sering juga dimanfaatkan untuk *approximate string matching* karena *phonetic string matching* lebih mudah diimplementasikan. *Phonetic string matching* banyak digunakan dalam bahasa Inggris karena dalam bahasa Inggris terdapat perbedaan antara penulisan dan pengucapan.

Langkah umum kerja *fuzzy string matching* dalam mencocokkan kata meliputi:

1. Menerima input berupa dua *string* yang akan dicocokkan
2. Mengubah dua string masukan menjadi dua buah kode biner (*binery code*). Biner kode adalah Sistem bilangan biner atau sistem bilangan basis dua adalah sebuah sistem penulisan angka dengan menggunakan dua simbol yaitu 0 dan 1. Sistem bilangan biner modern ditemukan oleh Gottfried Wilhelm Leibniz pada abad ke-17. Sistem bilangan ini merupakan dasar dari semua sistem bilangan berbasis digital. Dari sistem biner, kita dapat mengkonversinya ke sistem bilangan Oktal atau Hexadesimal. Sistem ini juga dapat kita sebut dengan istilah *bit*, atau *Binary Digit*. Pengelompokan biner dalam komputer selalu berjumlah 8, dengan istilah 1 Byte/bit. Dalam istilah komputer, 1 Byte = 8 bit. Kode-kode rancang bangun komputer, seperti ASCII, *American Standard Code for Information Interchange* menggunakan sistem peng-*kode*-an 1 *Byte*.
3. Membandingkan dua buah kode biner yang dihasilkan, jika kode biner sama maka dua string dianggap cocok dengan tingat nilai kecocokan yang sudah ditentukan, jika kode biner berbeda maka dua *string* dianggap tidak cocok.

Contoh

1.

String yang di masukan user

S	E	M	B	U	H



String yang ada di data base

S	E	M	B	U	H

Nilai Penalti Minimum = $\frac{1}{Jumlah\ Karakter\ String\ "Pola"}$

Nilai Penalti Minimum = $\frac{1}{6}$

Jarak = abs (lokasi karakter string – lokasi karakter *string* "pola")

Untuk karakter "S", jarak = 1-1 = 0;

Untuk karakter "E", jarak = 2-2 = 0;

Untuk karakter "M", jarak = 3-3 = 0;

Untuk karakter "B", jarak = 4-4 = 0;

Untuk karakter "U", jarak = 5-5 = 0;

Untuk karakter "H", jarak = 6-6 = 0;

Nilai Penalti = jarak x nilai penalti minimum

Untuk karakter "S", Nilai Penalti = 0 x (1/6) = 0;

Untuk karakter "E", Nilai Penalti = 0 x (1/6) = 0;

Untuk karakter "M", Nilai Penalti = 0 x (1/6) = 0;

Untuk karakter "B", Nilai Penalti = 0 x (1/6) = 0;

Untuk karakter "U", Nilai Penalti = 0 x (1/6) = 0;

Untuk karakter "H", Nilai Penalti = 0 x (1/6) = 0;

Nilai Persentase = $\frac{\sum_{n=1}^{x} 1 - nilai\ penalti}{jumlah\ karakter\ string\ "pola"}$ **x 100%**

Nilai Persentase = $\frac{(1-1)+(1-1)+(1-1)+(1-1)+(1-1)+(1-1)}{6}$ **x 100=100%**

Jika tingkat kococokan diatas 70% diangap cocok

2.

String yang di masukan user

M	A	K	A	N



String yang ada di data base

M	I	N	U	M

Nilai Penalti Minimum = $\frac{1}{Jumlah\ Karakter\ String\ "Pola"}$

Nilai Penalti Minimum = $\frac{1}{5}$

Jarak = abs (lokasi karakter string – lokasi karakter *string* "pola")

Untuk karakter "M" (1), jarak = 1-1 = 0;

Untuk karakter "M"(2), jarak = 1-5 = 4;

Untuk karakter "N", jarak = 5-3 = 0;

Nilai Penalti = jarak x nilai penalti minimum

Untuk karakter "M" (1), Nilai Penalti = 0 x (1/5) = 0;

Untuk karakter "M" (2), Nilai Penalti = 4 x (1/5) = 0,8;

Untuk karakter "N", Nilai Penalti = 2 x (1/5) = 0,4;

Nilai Persentase = $\frac{\sum_{n=1}^{x} 1-nilai\ penalti}{jumlah\ karakter\ string\ "pola"}$ **x 100%**

Nilai Persentase = $\frac{(1-1)+(1-0{,}8)+(1-0{,}4)}{5}$ **x 100=36%**

Jika tingkat kococokan kurang dari 70% diangap tidak cocok

Dari beberapa percobaan yang dilakukan didapatkan bahwa angka 70% itu, angka yang paling untuk sebagai persamaan nilai yang cocok.

Bahasa dalam bentuk tulisan sebenarnya terdiri atas simbol-simbol satuan yang jika dikombinasikan akan mempunyai arti yang berbeda-beda. Simbol-simbol yang bisa dipergunakan dalam sebuah bahasa tentunya terbatas jumlahnya, yang membentuk sebuah himpunan dan disebut sebagai abjad *(alphabet*). Kadangkala digunakan istilah karakter yang maknanya sama dengan simbol. Deretan karakter membentuk *string*. Bahasa (*language*) didefinisikan sebagai himpunan semua *string* yang dapat dibentuk dari suatu abjad. Kaidah/aturan pembentukan kata/kalimat disebut tata bahasa (*grammar*). Jika x adalah suatu string maka |x| adalah panjang x yaitu jumlah simbol yang terdapat dalam x. String kosong, dinotasikan dengan ε, adalah *string* dengan panjang 0 (*string* yang tidak mempunyai simbol di dalamnya).

Karena bahasa adalah sebuah himpunan dari *string*, maka untuk mendefinisikan suatu bahasa bisa dilakukan dengan menuliskan semua *string* yang menjadi anggotanya. Kita tidak bisa melakukannya jika jumlah *string* yang menjadi anggota bahasa tersebut banyak sekali atau bahkan tidak berhingga. Pada Teori Bahasa Formal, hal ini dilakukan dengan mendefinisikan tata bahasanya. Tata Bahasa G = (T,N,S,P), di mana

- T adalah himpunan berhingga simbol-simbol terminal
- N adalah himpunan berhinggasimbol-simbol non terminal
- S adalah simbol awal, $S \in N$
- P adalah himpunan berhingga aturan produksi yang setiap elemennya berbentuk $\alpha \rightarrow \beta$, $\alpha, \beta \in (T \cup N)+$, α harus berisi minimal 1 simbol non terminal.

Sentential form adalah semua *string* yang dapat diturunkan dari simbol awal S dengan menggunakan aturan produksi P. Kalimat (*sentence*) adalah sentential form yang tidak mengandung simbol non terminal. Bahasa yang dihasilkan dari G dinotasikan dengan L(G), yaitu himpunan kalimat yang dapat diturunkan dari S dengan menggunakan P.

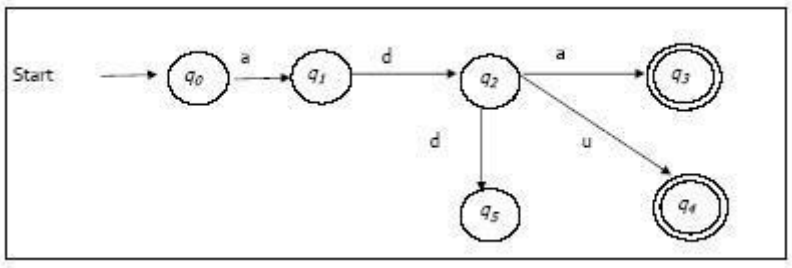
Untuk mengenali *string-string* dari suatu bahasa, akan dimodelkan sebuah mesin otomatis yang memiliki komponen sebagai berikut :

a. pita masukan, yang menyimpan string masukan yang akan dikenali;
b. kepala pita (*tape head*), untuk membaca/menulis ke pita masukan;
c. *Finite State Controller* (FSC), yang berisi status-status dan aturan-aturan yang mengatur langkah yang dilakukan oleh automaton berdasarkan status setiap saat dan simbol masukan yang sedang dibaca oleh kepala pita;
d. pengingat (*memory*), untuk tempat penyimpanan dan pemrosesan sementara Automaton pengenal, setelah membaca *string* masukan dan melakukan langkah langkah pemrosesan yang diperlukan, akan mengeluarkan keputusan apakah string tersebut dikenali atau tidak.

Konfigurasi adalah suatu mekanisme untuk menggambarkan keadaan suatu mesin pengenal , yang terdiri atas :

a. status FSC
b. isi pita masukan dan posisi kepala pita
c. isi pengingat

Mesin pengenal bersifat *deterministik* bila dalam setiap konfigurasi hanya ada satu kemungkinan yang dapat dilakukan mesin, jika tidak mesin pengenal bersifat *non deterministik*. Contoh mesin automata :



Gambar 2.4 Contoh Mesin Automata

- Sebuah string input diterima bila mencapai state akhir/final state yang digambarkan dengan lingkaran ganda.
- Pada Gambar 2.4 diatas, mesin mendapat string input berikut :
  – ada : diterima

- adu : diterima
- add : ditolak

- Mesin diatas memiliki 6 state {qo, q1, q2, q3, q4, q5}.
- State awal : q0, State akhir : {q3, q4}.
- Himpunan simbol input : {a,d,u}.

2.2.7 Aplikasi Kecerdasan Buatan

Kecerdasan buatan berupa aplikasi-aplikasi yang dapat diterapkan pada komputer besar (*main frame*), komputer mikro, atau *personal computer* (*PC*).[3]

Beberapa aplikasi kecerdasan buatan diantaranya sebagai berikut :

1. *General problem solving*

   Adalah suatu tahapan proses atau langkah-langkah yang berurutan untuk mencapai suatu tujuan, dengan cara melacak dan mengkombinasikan berbagai cara atau metode sehingga menghasilkan solusi terbaik.

2. *Expert system* atau sistem pakar

   Adalah suatu program yang bertindak sebagai penasehat atau konsultan pintar dengan mengambil pengetahuan yang disimpan dalam domain tertentu. Seorang pemakai yang belum berpengalaman dalam *mendiagnosa* suatu masalah dapat memecahkan masalah yang sulit dan mengambil keputusan dengan benar.

3. *Vision*

   Merupakan aplikasi yang dapat mengenali gambar yang diterimanya dari kamera, *scanner*, dan alat *input* lainnya dengan cara mencocokan dan melacak gambar apa yang diterimanya melalui kamera atau *scanner* sebagai masukan.

4. *Natural Language Processing* atau pengolahan bahasa alami

   Merupakan salah satu cabang dari *artificial intellegence* (AI) yang berhubungan dengan pemrosesan bahasa alami oleh komputer yang mencakup metode-metode *speech recognition*, *speech synthesizer* (*text-to-speech*), *parsing*, penterjemahan

bahasa, QA *system*, dan kemampuan bahasa alami lainnya pada sebuah mesin atau komputer.

5. *Robotics*

   Pada aplikasi robotics kecerdasan buatan bertindak seolah-olah melakukan pekerjaan fisik yang biasa dikerjakan manusia, bahkan robot dapat melakuakn pekerjaan yang tidak dapat dilakukan oleh manusia atau berbahaya bagi manusia. Semua tindakan dirancang dan disusun dalam satu urutan atau algoritma tertentu. Dengan bantuan *robot*, pekerjaan yang berulang kali dan rumit dapat dilakukan dengan baik.

6. *Education*

   Merupakan aplikasi kecerdasan buatan dalam bidang pendidikan yang bertindak sebagai partner bagi pelajar atau mahasiswa dalam mempelajari suatu bidang.

2.2.8 Tujuan Kecerdasan Buatan

Tujuan dari kecerdasan buatan menurut Winston dan Prendergast :

1. Membuat mesin menjadi lebih pintar (tujuan utama)
2. Memahami apa itu kecerdasan (tujuan ilmiah)
3. Membuat mesin lebih bermanfaat (tujuan entrepreneurial)

Dua bagian utama yang dibutuhkan untuk aplikasi kecerdasan buatan yaitu

1. Basis Pengetahuan (*Knowledge Base*) berisi fakta-fakta, teori, pemikiran dan hubungan antara satu dengan lainnya.
2. Motor Inferensi (*Inference Engine*) adalah kemampuan menarik kesimpulan berdasarkan pengalaman.

2.2.9 Bahasa Alami (*Natural Language*)

Pada prinsipnya bahasa alami adalah suatu bentuk *representasi* dari suatu pesan yang ingin dikomunikasikan antar manusia. Bentuk utama representasinya adalah berupa suara/ucapan (*spoken language*), tetapi sering pula dinyatakan dalam bentuk tulisan.

Bahasa dapat dibedakan menjadi Bahasa Alami dan Bahasa Buatan. Bahasa alami adalah bahasa yang biasa digunakan untuk berkomunikasi antar manusia, misalnya bahasa Indonesia, Sunda, Jawa, Melayu, Inggris, China, Jepang, dan sebagainya.

Bahasa buatan adalah bahasa yang dibuat secara khusus untuk memenuhi kebutuhan tertentu, misalnya bahasa pemodelan atau bahasa pemrograman komputer.

Chomsky adalah orang yang pertama kali merepresentasikan bahasa sebagai rangkaian simbol. *Chomsky* berhasil memperlihatkan bahwa bahasa apapun dapat direpresentasikan dengan suatu cara yang *universal*. Pemikiran *Chomsky* yang merepresentasikan bahasa sebagai kumpulan simbol-simbol dan aturan yang mengatur susunan simbol-simbol tersebut telah membuka peluang untuk melakukan pemrosesan bahasa secara simbolik dengan teknologi komputer, sehingga melahirkan bidang ilmu *Natural Language Processing (NLP).*

Linguistik adalah bidang ilmu yang khusus mengkaji tentang bagaimana bahasa distrukturkan dan digunakan. Bidang ilmu ini salah satunya membahas tentang perbendaharaan kata dan *leksikon.*

Perbendaharaan kata adalah sekumpulan kata-kata dan *frase-frase* yang digunakan dalam bahasa tertentu. Sebagai bagian dari pengkajian bahasa, *linguis* mendefinisikan semua kata-kata dan *frase-frase* yang digunakan secara umum kemudian mengorganisasikannya ke dalam sebuah *leksikon. Leksikon* adalah kamus yang mendaftar kata-kata bahasa itu secara alfabet.

2.2.10 Pengertian *Natural Language Processing*

Pengolahan bahasa alami (*NLP*) adalah bidang ilmu komputer dan *linguistik* berkaitan dengan interaksi antara komputer dan manusia. Seluler generasi bahasa sistem komputer yang mengubah informasi dari database ke dalam bahasa manusia yang dapat dibaca.

Natural language processing, biasanya disingkat dengan *NLP*, mencoba membuat komputer mampu memahami suatu perintah yang dituliskan dalam bentuk bahasa sehari-hari dan diharapkan komputer juga merespon dalam bahasa yang mirip dengan bahasa natural. Setelah komputer bisa memahami perintah

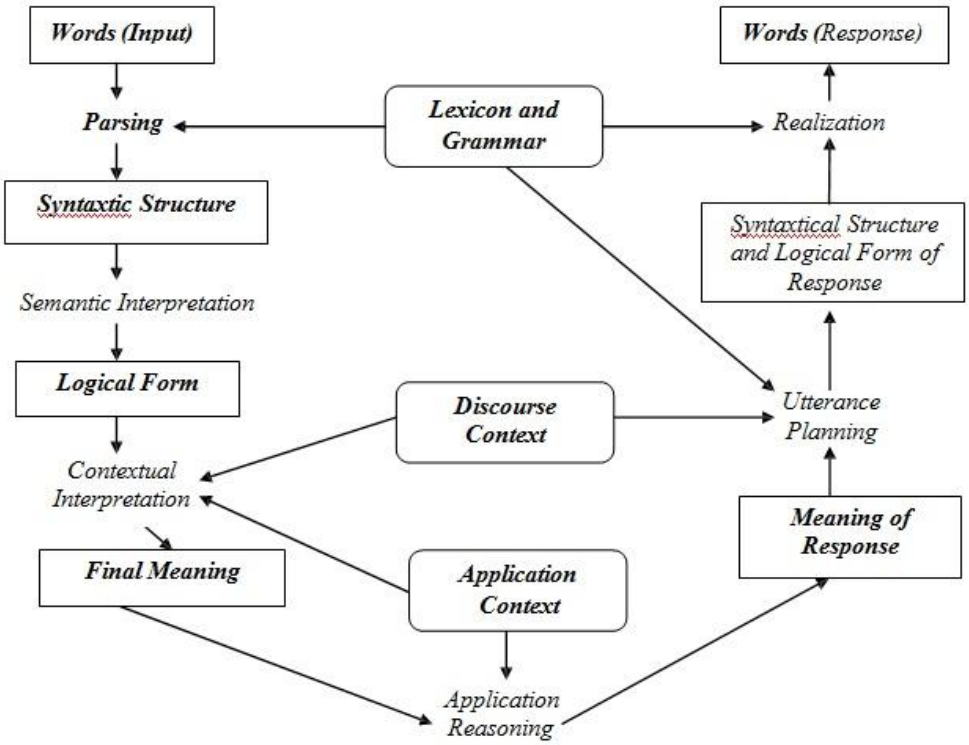
dalam bahasa *natural,* maka diharapakan sistem komputer juga dapat memberikan respon dalam bahasa *natural* pula.

Sebuah sistem *natural language* harus memperhatikan pengetahuan terhadap bahasa itu sendiri baik dari segi kata yang digunakan bagaimana kata-kata tersebut digabung untuk menghasilkan suatu kalimat, apa arti suatu kata, apa fungsi sebuah kata dalam sebuah kalimat dan sebagainya. *Natural Language processor* tidak memperdulikan bagaimana suatu kalimat diinputkan ke komputer. Tugasnya adalah mengekstrak informasi dari kalimat. Inti dari sistem *NLP* adalah *parser*. *Parser* adalah bagian dari program atau *system* yang membaca setiap kalimat, kata demi kata, untuk menentukan "*what is what*".

NLP tidak bertujuan untuk mentransformasikan bahasa yang diterima dalam bentuk teks atau suara menjadi data digital dan/atau sebaliknya pula; melainkan bertujuan untuk memahami arti dari kalimat yang diberikan dalam bahasa alami dan memberikan respon yang sesuai, misalnya dengan melakukan suatu aksi tertentu atau menampilkan data tertentu.

Untuk mencapai tujuan ini dibutuhkan tiga tahap proses. Proses yang pertama ialah *parsing* atau analisa *sintaksis* yang memeriksa kebenaran struktur kalimat berdasarkan suatu *grammar* (tata bahasa) dan *lexicon* (kosa kata) tertentu. Proses kedua ialah *semantic interpretation atauinterpretasi* semantik yang bertujuan untuk merepresentasikan arti dari kalimat secara context-independent untuk keperluan lebih

lanjut. Sedangkan proses ketiga ialah *contextual* interpretation atau interpretasi kontekstual yang bertujuan untuk merepresentasikan arti secara *context dependent* dan menentukan maksud dari penggunaan kalimat. Gambaran sebuah organisasi sistem NLP dapat dilihat pada gambar 2.5



Gambar 2.5 Organisasi sebuah sistem NLP

Jenis aplikasi yang bisa dibuat pada bidang bidang *natural language* adalah *text - based application dan dialogue - based applications*.

1. *Text - based application*

   Text -based application ini adalah Mencakup segala macam aplikasi yang melakukan proses terhadap text tertulis seperti misalnya buku, berita di surat kabar, e-mail dan lain sebagainya. Contoh penggunaan dari *text - based application* ini adalah :

   a. Mencari topik tertentu dari buku yang ada pada perpustakaan.
   b. Memberikan respon atas *input* yang diberikan.
   c. Mencari isi dari surat atau e-mail.
   d. Menterjemahkan dokumen dari satu bahasa ke bahasa yang lain.

2. *Dialogue - based application*

   Dialogue - based application idealnya pedekatan ini melibatkan bahasa lisan atau pengenalan suara, akan tetapi bidang ini juga memasukkan interaksi dengan cara memasukkan teks pertanyaan melalui *keyboard.* Aplikasi yang sering ditemui untuk bidang ini adalah :

   a. Sistem tanya jawab, dimana *natural language* digunakan dalam mendapatkan informasi dari suatu database.
   b. Sistem otomatis pelayanan melalui telepon.
   c. Kontrol suara pada perlatan sistem.
   d. Sistem *problem solving* yang membantu untuk melakukan penyelesaian masalah yang umum dihadapi dalam suatu pekerjaan.

2.2.11 *Question Answering System*

Question answering system (QA system) adalah sistem yang mengijinkan *user* menyatakan kebutuhan informasinya dalam bentuk natural language question (pertanyaan dalam bahasa alami), dan mengembalikan kutipan teks singkat atau bahkan *frase* sebagai jawaban.

Ketersediaan sumber informasi yang luas dan bervariasi, serta adanya perkembangan yang pesat dari teknik *Natural Language Processing* (NLP), *Information Extraction* (IE), dan Information Retrieval (IR) sangat mempengaruhi perkembangan dari QA system.

Umumnya arsitektur *question answering* system yang berbasis teks dibangun atas enam tahapan proses, yaitu analisis pertanyaan, preprocessing koleksi dokumen, pemilihan kandidat dokumen, analisis kandidat dokumen, ekstraksi jawaban, dan pemberian respon.

Aplikasi-aplikasi QA *system* (baik yang dapat diakses melalui internet maupun tidak) yang dikembangkan dengan ciri khasnya masing-masing memiliki urutan proses yang tidak jauh berbeda satu dengan lainnya. Jawaban yang dikembalikan oleh sebuah QA *system* sebagai respon terhadap pertanyaan perlu dievaluasi untuk menilai performansi sistem.

2.2.12 *Scanner* (Analisis Leksikal)

Analisis Leksikal *(Scanner)* merupakan antarmuka antara kode program sumber dan analisa *sintaktik* (*parser*). Atau dalam pengertiannya adalah sebuah proses yang mendahului parsing sebuah rangkaian karakter. *Scanner* melakukan pemeriksaan karakter per karakter pada teks masukan, memecah sumber program menjadi bagian-bagian disebut *Token*. Proses *parsing* akan lebih mudah dilakukan bila inputnya sudah berupa token.

Analisis *leksikal* membuat pekerjaan membuat sebuah parser jadi lebih mudah daripada membangun nama setiap fungsi dan variabel dari karakter-karakter yang menyusunnya, dengan analisis leksikal *parser* cukup hanya berurusan dengan sekumpulan token dan nilai *sintaksis* masing-masing.

Terlepas dari efisiensi pemrograman yang dapat dicapai dengan penggunaannya, proses kerja analisis *leksikal* yang membaca lebih dari sekali setiap karakter dari input yang diberikan menjadikan penganalisa *leksikal* sebagai sub-sistem yang paling intensif melakukan komputasi, terutama bila digunakan dalam sebuah kompilator.

Kompilator adalah sebuah program yang membaca suatu program yang ditulis dalam suatu bahasa sumber (*source language*) dan menterjemahkannya ke dalam suatu bahasa sasaran (*target language*). Dalam penguraian struktur kalimat, penganalisa leksikal menganalisa setiap kata dalam kalimat, kemudian menentukan jenis kelas katanya.

Hasil dari penganalisa leksikal ini digunakan oleh penganalisa *sintaks* yang akan memeriksa urutan simbol-simbol kelas kata tersebut dalam kalimat. Analisa kata dalam kalimat ini dilakukan oleh penganalisa leksikal berdasarkan kecocokan kata dengan aturan-aturan leksikal berupa ekspresi regular yang sudah didefinisikan.

Tugas dari *scanner* adalah sebagai berikut :

1. Melakukan pembacaan kode sumber dengan merunut karakter demi karakter
2. Mengenali besaran leksik
3. Mentransformasi menjadi sebuah *token* dan menentukan jenis *token*-nya.

4. Mengirimkan *token*
5. Membuang/mengabaikan *blank* dan komentar dalam program
6. Menangani kesalahan
7. Menangani tabel simbol

Di dalam aplikasi NLP sistem cerdas yang akan dibuat, yang dimaksud dengan program sumber yang diolah oleh *scanner* adalah berupa kalimat input dari pengguna dalam bentuk sms

Ketika *scanner* menerima input berupa stream karakter kemudian memilah menjadi satuan leksik, satuan leksik tersebut terdiri atas simbolsimbol satuan yang jika dikombinasikan akan mempunyai arti yang berbedabeda. Simbol-simbol yang bisa dipergunakan dalam sebuah bahasa tentunya terbatas jumlahnya, yang membentuk sebuah himpunan dan disebut sebagai abjad (*alphabet).*

Tata bahasa (*grammatika*) adalah sekumpulan dari himpunan variabel-variabel, simbol-simbol terminal, simbol non-terminal, simbol awal yang dibatasi oleh aturan-aturan produksi. Aturan produksi adalah pusat dari tata bahasa yang menspesifikasikan bagaimana suatu tata bahasa melakukan transformasi suatu string ke bentuk lainnya.

Dalam pembicaraan *grammar,* anggota *alfabet* dinamakan simbol terminal atau *token.* Kalimat adalah *string* yang tersusun atas simbol-simbol terminal. Bahasa adalah himpunan kalimat-kalimat. Anggota bahasa bisa berupa tak berhingga hingga kalimat.

Simbol-simbol berikut adalah *symbol* terminal :

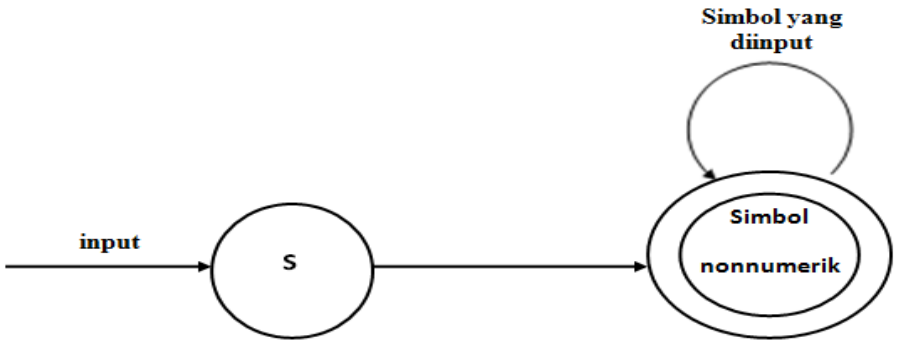
1. huruf kecil alphabet, misalnya :a, b, c
2. symbol operator, misalnya : +, -, dan ‘
3. symbol tanda baca, misalnya : (,), dan ;

Sedangkan simbol-simbol berikut adalah simbol non terminal :

1. huruf besar awal alphabet, misalnya : A, B, C

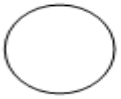
2. huruf S sebagai simbol awal

Pada saat *scanner* membaca *input*, *tools* yang digunakan untuk menggambarkan perpindahan dari posisi satu ke posisi lainnya adalah diagram transisi. Diagram transisi dapat dilihat pada gambar 2.6

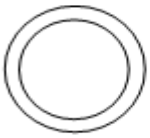


Gambar 2.6 Diagram transisi

Keterangan :


: state/keadaan awal input suatu kalimat


: *looping*/perulangan pembacaan simbol


: state/keadaan akhir suatu kalimat

2.2.13 Parsing

Proses *parsing* tidak hanya dapat dilakukan dalam proses *information retrieval*, melainkan juga pada bidang lain seperti pada pembuatan sebuah *compiler* dan bahasa alami. Sebelumnya perlu diketahui arti dari istilah *parser* yaitu program yang melakukan proses parsing.

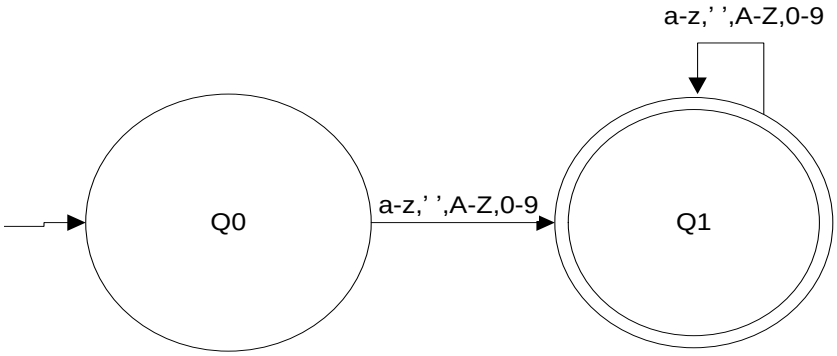
Untuk pemrosesan, dokumen dipisahkan menjadi unit-unit yang lebih kecil misalnya berupa kata, frasa atau kalimat. Unit pemrosesan tersebut disebut sebagai token. *Parsing* merujuk pada proses pengenalan *token* yang terdapat dalam rangkaian *teks*.

Proses *parsing* (penguraian kalimat) juga merupakan proses yang dilakukan untuk menterjemahkan masukan dari pengguna agar dapat dimengerti oleh sistem. Secara *default*, seluruh kalimat masukan pengguna yang dimasukan akan dianggap sebagai kata-kata yang harus ada pada data yang akan dicari. Misal pengguna memasukan kalimat sebagai berikut :

Penggunaan bahasa alami

Maka sistem akan melakukan pencarian data pada dokumen yang memiliki kata *bahasa* atau *alami.*Artinya jika terdapat dokumen yang mengandung kata *bahasa inggris* maka dokumen tersebut dianggap sesuai dengan yang dicari pengguna, karena mengandung kata *bahasa.*Hal ini membuat pengguna yang ingin mencari informasi mengenai *bahasa alami* akan mendapatkan hasil yang tidak sesuai dengan yang diinginkan.

Jika pengguna ingin melakukan proses pencarian yang lebih spesifik, agar informasi yang didapat lebih tepat sasaran, maka pengguna haru mengikuti aturan proses penguraian kalimat yang dimiliki sistem dapat dilihat pada gambar 2.7



Gambar 2.7 ***Finite State Automata*** **Proses** ***Parsing***

Konfigurasi FSA diatas secara formal dinyatakan sebagai berikut :

Q = {Q0, Q1}

∑ = {a-z,' ',A-Z,0-9}

S = Q0

F = {Q1}

Fungsi transisi yang ada sebagai berikut :

δ(Q0, a-z|' '|A-Z|0-9) = Q1

δ(Q1, a-z|’ ‘|A-Z|0-9) = Q1

Fungsi tersebut bisa disajikan dan dilihat dalam tabel 2.1 berikut :

Tabel 2.1 Tabel Transisi

δ	a-z|’ ‘|A-Z|0-9
Q0	Q1
Q1	Q1

Dari diagram state gambar 2.7 , didapat aturan produksi sebagai berikut:

S ->aA..zA| AA..ZA | 0A-9A|<space>A|a..Z|A..Z|0..9|<space>

A ->aA..zA| AA..ZA |0A-9A|<space>A|a..Z|A..Z|0..9|<space>

Dimana secara formal dapat ditulis :

V = {S, A}

T = {a-z,’ ‘ A-Z,0-9}

P = {S->aA..zA|AA..ZA|0A..9A|<space>A|a..Z|A..Z|0..9|<space>,

A->aA..zA| AA..ZA |0A..9A|<space>A|a..Z|A..Z|0..9|<space>}

S = S

Aturan produksi diatas menggunakan *left to right* sehingga jika pengguna memasukan pertanyaan “hai juga”, maka proses akan dilakukan dari depan (*left*) ke belakang (*right*), seperti berikut: :

S ->hA

S ->haA

S ->haiA

S ->haiA

S ->hai jA

S ->hai juA

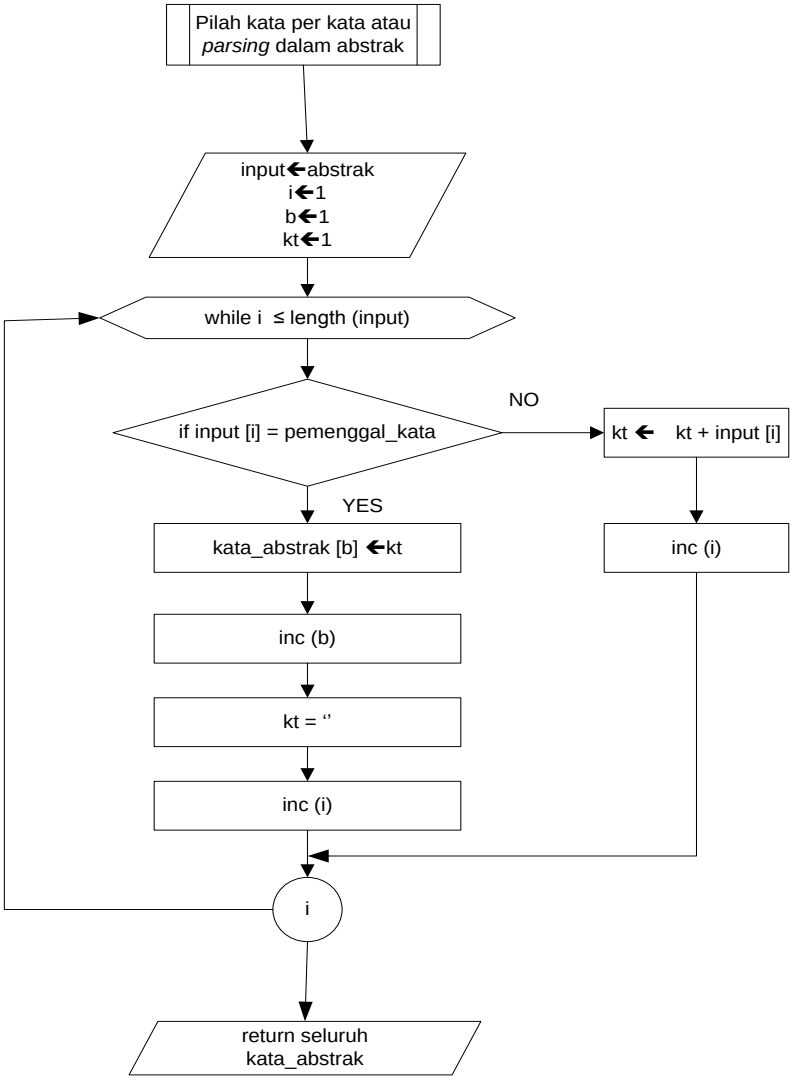
S ->hai jugA

S ->hai jugaA

S ->hai juga

Pada proses di atas setiap kalimat atau kata akan dipecah menjadi perbagian karakter dalam bentuk array. *Flowchart* proses *parsing abstak*

dapat dilihat pada gambar 2.8 , proses ini digunakan untuk memotong - motong dokumen kata per kata dan menyimpannya pada *array* kata *abstrak.*



Gambar 2.8 *Flowchart Parsing*

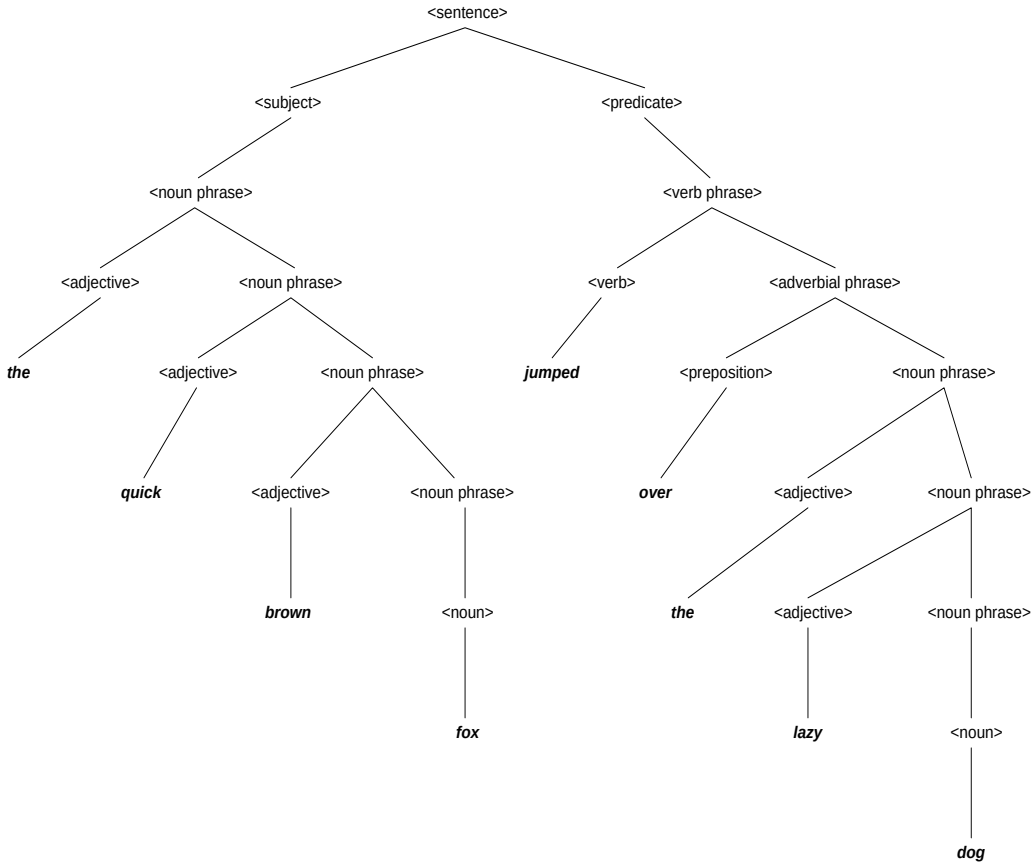
2.2.14 Pohon Sintaks

Pohon *(tree)* adalah suatu graph terhubung tidak sirkuler, yang memiliki satu simpul (*node*)/*vertex* disebut akar (*root*) dan dari situ memiliki lintasan ke setiap simpul.

Pohon sintaks/pohon penurunan (*syntax tree/derivation tree/ parse tree*) berguna untuk menggambarkan bagaimana memperoleh suatu *string* dengan cara menurunkan simbol-simbol variabel menjadi simbol-simbol

terminal. Setiap simbol variabel akan diturunkan menjadi *terminal*, sampai tidak ada yang belum tergantikan Contoh dapat dilihat pada gambar 2.9 , dimana contoh sebuah *tree* yang menguraikan kalimat dalam bahasa Inggris.

The quick brown fox jumped over the lazy dog



Gambar 2.9 Pohon Sintaks

Proses penurunan atau *parsing* bisa dilakukan dengan cara:

1. Penurunan terkiri (*leftmost derivation*): simbol variabel terkiri yang diperluas terlebih dulu
2. Penurunan terkanan (*rightmost derivation*): simbol variabel terkanan yang diperluas terlebih dulu.

2.2.15 *Text Mining*

Text mining merupakan salah satu aplikasi dari data mining. *Text mining* juga sering disebut sebagai *Text Data Mining* (TDM) dan *knowladge Discovery in Textual Databases* (KDT) . Text mining merupakan proses mengesktrak *petterns dan knowladge* yang bersifat menarik dan nontrivial (penting) dari dokumen-dokumen teks. Pada intinya proses kerja text mining sama dengan proses kerja data mining pada umumnya hanya saja data yang di mining merupakan *text database*s.[4]

Data teks akan diproses menjadi data numerik agar dapat dilakukan proses lebih lanjut. Sehingga dalam *text mining* ada istilah *preprocessing* data, yaitu proses pendahulu yang diterapkan terhadap data teks yang bertujuan untuk menghasilkan data numerik.

Pada proses *preprosesing* merupakan tahap dimana deskripsi di tangani untuk dapat siap diproses memasuki tahap *text mining.* Tahap-tahap tersebut adalah:

1. *Parsing/ Tokenizing*
2. *Stopwords Removal/ Filtering*
3. *Stemming*
4. *Anayizing*

2.2.16 *Tokenizing*

Tokenizing adalah sebuah proses yang dilakukan seseorang untuk menjadikan sebuah kalimat menjadi lebih bermakna atau berada dengan cara memecah kalimat tersebut menjadi kata-kata atau *frase-frase ("Parsing")*. *Parsing* di dalam pembuatan aplikasi *text mining* ini merupakan proses penguraian deskripsi yang semula berupa kalimat-kalimat berisi kata-kata dan tanda pemisah antara kata seperti titik(.), koma(,), spasi dan tanda pemisah lain menjadi kata-kata saja baik itu berupa kata-kata penting maupun kata-kata tak penting. Secara sederhana proses *parsing* ini terlihat sebagai proses pengambilan kata jika ketemu tanda spasi namun pada kenyataannya tidak sesederhana itu.

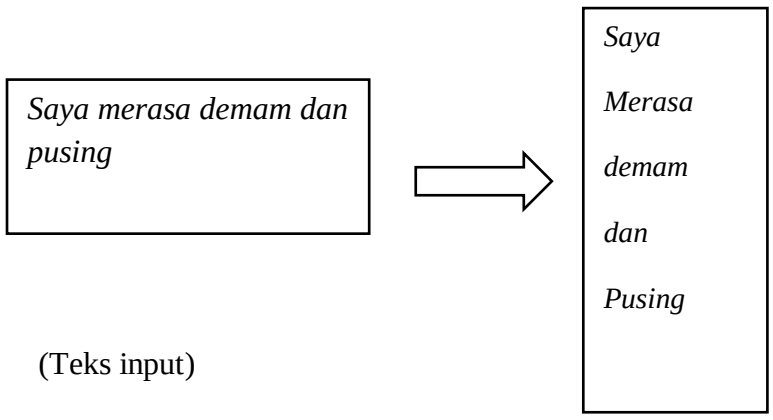
Proses parsing tidak hanya dapat dilakukan dalam proses information *retrieval*, melainkan juga pada bidang lain seperti pada pembuatan sebuah

compiler dan bahasa alami. Sebelumnya perlu diketahui arti dari istilah *parser* yaitu program yang melakukan proses parsing.

Untuk pemrosesan, dokumen dipisahkan menjadi unit-unit yang lebih kecil misalnya berupa kata, frasa atau kalimat. Unit pemrosesan tersebut disebut sebagai *token. Parsing* merujuk pada proses pengenalan token yang terdapat dalam rangkaian teks.

Proses *parsing* (penguraian kalimat) juga merupakan proses yang dilakukan untuk menterjemahkan masukan dari pengguna agar dapat dimengerti oleh sistem. Secara *default*, seluruh kalimat masukan pengguna yang dimasukan akan dianggap sebagai kata-kata yang harus ada pada data yang akan dicari. Contoh tahap ini dapat dilihat pada gambar 2.10

Gambaran contoh tahap *tokenizing* dapat dilihat pada gambar 2.10.

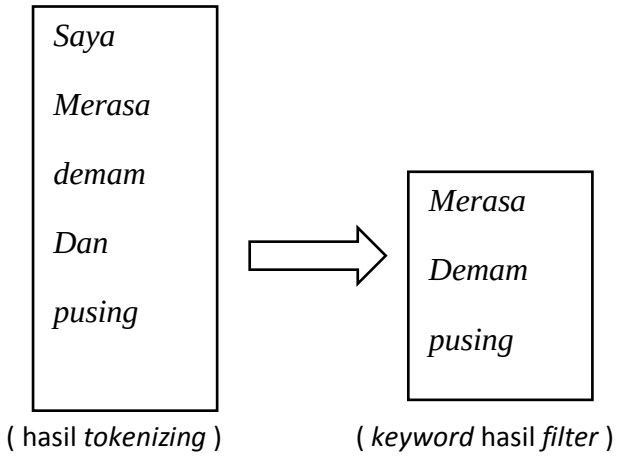


Gambar 2.10 Tahap *Tokenizing*

2.2.17 *Stopword Removal / Filtering*

Kebanyakan bahasa resmi di berbagai negara memiliki kata fungsi dan kata sambung seperti artikel dan preposisi yang hampir selalu muncul pada dokumen-dokumen teks. Biasanya kata-kata ini memiliki arti yang lebih di dalam memenuhi kebutuhan seorang *searcher* di dalam mencari informasi. Kata-kata tersebut (misalnya *a, an, the on* pada bahasa inggris) disebut sebagai *stopwords*. Di dalam bahasa Indonesia *stopwords* dapat disebut sebagai kata tidak penting misalnya "di", "oleh", "pada", "sebuah", "karena". Sebelum proses *stopwords removal* dilakukan, terlebih dulu dibuat daftar stopwords (stoplist). Preposisi, kata hubung dan partikel biasanya merupakan kandidat *stoplist*.

Stopwords removal merupakan proses penghilangan kata tidak penting pada deskripsi melalui pengecekan kata-kata hasil *parsing* deskripsi apakah termasuk di dalam daftar kata tidak penting *(stoplist)* atau tidak. Jika termasuk di dalam *stoplist* maka kata-kata tersebut akan di-*remove* dari deskripsi sehingga kata-kata yang tersisa di dalam deskripsi di anggap sebagai kata-kata penting atau *keywords*. Tahap *filtering* dapat dilihat pada gambar 2.11

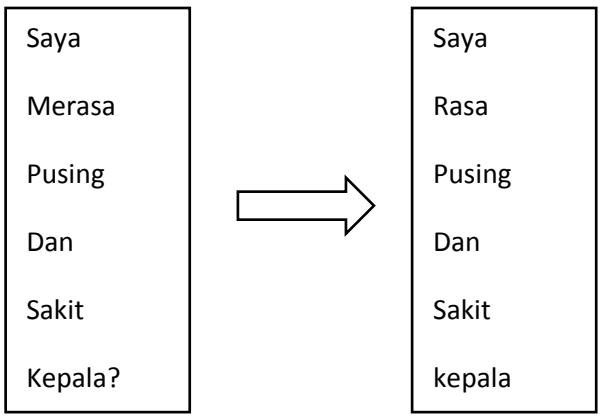


Gambar 2.11 Tahap *Filtering*

2.2.18 Stemming

Stemming adalah proses pemetaan dan penguraian berbagai bentuk (*variants*) dari suatu kata menjadi bentuk kata dasarnya (*stem*). Proses ini juga disebut sebagai conflation. Proses stemming secara luas sudah digunakan di dalam *Information retrieval* (pencarian informasi) untuk meningkatkan kualitas informasi yang didapatkan. Kualitas informasi yang dimaksud misalnya untuk mendapatkan hubungan antara varian kata yang satu dengan yang lainnya. Sebagai contoh kata "diculik", "menculik" (melakukan tindakan menculik) dan "penculik" (orang yang menculik) yang semula mengandung arti yang berbeda dapat di-stem menjadi sebuah kata "culik" yang memiliki arti yang sama sehingga kata-kata diatas saling berhubungan.

Selain itu *stemming* juga dapat digunakan untuk mengurangi ukuran dari suatu ukuran index file. Misalnya dalam suatu *deskripsi* terdapat varian kata "memberikan", "diberikan", "memberi" dan "diberi" hanya memiliki akar kata (*stem*) yaitu "beri". Ukuran *file* daftar index yang semula berjumlah lima record akan di-*reduce* sehingga menjadi satu record saja. Tahap *stemming* dapat dilihat pada gambar 2.12



Gambar 2.12 Tahap *stemming*

Confix Stripping (CS) stemmer adalah metode *stemming* untuk Bahasa Indonesia yang diperkenalkan oleh Jelita Asian. *Stemmer* ini merupakan pengembangan dari metode *stemming* untuk Bahasa Indonesia yang diperkenalkan oleh Nazief dan Adriani (1996). Algoritma stemming Nazief dan Adriani ini dikembangkan berdasarkan pada aturan morfologi Bahasa Indonesia yang mengelompokkan dan mengenkapsulasi imbuhan-imbuhan, termasuk di dalamnya adalah awalan (*prefix)*, sisipan 17 *(infix)*, akhiran *(suffix)* dan gabungan awalan-akhiran (*confixes).*

Algoritma ini menggunakan kamus kata dasar dan mendukung *recoding*, yakni penyusunan kembali kata-kata yang mengalami proses *stemming* berlebih. Kombinasi awalan dan akhiran yang dilarang dapat dilihat pada tabel 2.2.

Tabel 2.2 Kombinasi Awalan-Akhiran yang dilarang

Awalan (*prefix)*	Akhiran (*suffix)* yang tidak diperbolehkan
be-	-i
di-	-an
ke-	-i, -kan
me-	-an
se-	-i, -kan
te-	-an

Algoritma *stemmer* Nazief dan Adriani mengelompokkan imbuhan ke dalam beberapa kategori sebagai berikut:

1. *Inflection suffixes* yakni kelompok-kelompok akhiran yang tidak merubah bentuk kata dasar. Sebagai contoh, kata “duduk” yang diberikan

akhiran "-lah" akan menjadi "duduklah". Kelompok ini dapat dibagi menjadi dua :

a. Particle (P) atau partikel, yakni termasuk di dalamnya "-lah", "-kah", "-tah", dan "-pun".
b. *Possessive Pronoun* (PP) atau kata ganti kepunyaan, termasuk di dalamnya adalah "-ku" , "-mu", dan "-nya".

2. *Derivation Suffixes (*DS) yakni kumpulan akhiran asli Bahasa Indonesia yang secara langsung ditambahkan pada kata dasar. Termasuk di dalamnya adalah akhiran "-i", "-kan", dan "-an".

3. *Derivation Prefixes* (DP) yakni kumpulan awalan yang dapat langsung diberikan pada kata dasar murni, atau pada kata dasar yang sudah mendapatkan penambahan sampai dengan 2 awalan. Termasuk di dalamnya adalah awalan yang dapat bermorfologi ("me-", "be-", "pe-", dan "te-") serta awalan yang tidak bermorfologi ("di-", "ke-" dan "se-").

Berdasarkan pengklasifikasian imbuhan-imbuhan di atas, maka bentuk kata berimbuhan dalam Bahasa Indonesia dapat dimodelkan sebagai berikut:

[DP + [DP + [DP +]]] Kata Dasar [[+ DS][+ PP][+ P]]

Dengan batasan-batasan sebagai berikut :

a. Tidak semua kombinasi diperbolehkan, sebagai contoh pada kata yang diberi awalan "di-", maka penambahan akhiran "-an" tidak diperkenankan. Kombinasi-kombinasi imbuhan yang tidak diperbolehkan dapat dilihat pada Tabel 2.1.
b. Penggunaan imbuhan yang sama secara berulang tidak diperkenankan.
c. Jika suatu kata hanya terdiri dari satu atau dua huruf, maka proses *stemming* tidak dilakukan.
d. Penambahan suatu awalan tertentu dapat mengubah bentuk asli kata dasar, ataupun awalan yang telah diberikan sebelumnya pada kata dasar bersangkutan (bermorfologi). Sebagai contoh, awalan "me-" dapat berubah menjadi "meng-", "men-", "meny-", dan "mem-". Oleh

karena itu, diperlukan suatu aturan yang mampu mengatasi masalah *morfologi* ini.

Pada dasarnya, algoritma *confix stripping stemmer* dikembangkan dari algoritma *stemming* yang dibuat oleh Nazief dan Adriani, dengan beberapa penambahan aturan tertentu yang telah terbukti mampu meningkatkan kinerja stemmer tersebut. Algoritma stemmer yang diperkenalkan Nazief dan Adriani didefinisikan sebagai berikut:

1. Kata yang hendak di-stemming dicari terlebih dahulu pada kamus. Jika kata ditemukan dalam kamus, berarti kata tersebut sudah berbentuk kata dasar, jika tidak maka tahap selanjutnya dilakukan.
2. Hilangkan *inflectional particle* P (“-lah”, “-kah”, “-tah”, “-pun”) dan kata ganti kepunyaan atau *possessive pronoun* PP (“-ku”, “-mu”, “-nya”).
3. Hilangkan *derivation suffixes* DS (“-i”, “-kan”, atau “-an”).
4. Hilangkan derivation prefixes DP {“di-”,“ke-”,“se-”,“me-”,“be-”,“pe”, “te-”} dengan iterasi maksimum adalah 3 kali:
   a. Langkah 4 berhenti jika:

      Terjadi kombinasi awalan dan akhiran yang terlarang seperti pada Tabel 2.1.

      Awalan yang dideteksi saat ini sama dengan awalan yang dihilangkan sebelumnya.

      Tiga awalan telah dihilangkan.

   b. Identifikasikan tipe awalan dan hilangkan. Awalan ada dua tipe:

      Standar: “di-”, “ke-”, “se-” yang dapat langsung dihilangkan dari kata.

      Kompleks: “me-”, “be-”, “pe”, “te-” adalah tipe-tipe awalan yang dapat bermorfologi sesuai kata dasar yang mengikutinya. Oleh karena itu, gunakan aturan pada Tabel 2.2 untuk mendapatkan pemenggalan yang tepat.

   c. Cari kata yang telah dihilangkan awalannya ini di dalam kamus. Apabila tidak ditemukan, maka langkah 4 diulangi kembali. Apabila ditemukan, maka keseluruhan proses dihentikan.
5. Apabila setelah langkah 4 kata dasar masih belum ditemukan, maka proses recoding dilakukan dengan mengacu pada aturan pada Tabel 2.2.

Recoding dilakukan dengan menambahkan karakter recoding di awal kata yang dipenggal. Pada Tabel 2.2, karakter recoding adalah huruf kecil setelah tanda hubung (‘-’) dan terkadang berada sebelum tanda kurung.

6. Jika semua langkah gagal, maka input kata yang diuji pada algoritma ini dianggap sebagai kata dasar.

Dilihat segi *performa, stemmer* ini mampu memberikan hasil yang lebih baik jika dibandingkan dengan algoritma *stemmer* untuk Bahasa Indonesia seperti yang dikembangkan oleh Arifin dan Setiono (2002) , Vega (2001), Ahmad (1996), dan Idris (2001). Hal ini wajar, mengingat algoritma ini merupakan algoritma yang paling kompleks. Jelita Asian kemudian mengembangkan algoritma CS stemmer, dengan menambahkan beberapa perbaikan yang bertujuan untuk meningkatkan hasil stemming yang diperoleh. Perbaikan yang ditambahkannya adalah sebagai berikut:

1. Menggunakan kamus kata dasar yang lebih lengkap.
2. Memodifikasi dan menambahkan aturan pemenggalan untuk tipe awalan yang kompleks (memodifikasi aturan pada Tabel 2.2 sesuai modifikasi pada Tabel 2.3, dan menambahkan aturan pada Tabel 2.4 ke Tabel 2.2).
3. Menambahkan aturan *stemming* untuk kata ulang dan bentuk jamak, semisal pada “buku-buku” yang seharusnya di-*stemming* menjadi “buku”. Caranya, adalah dengan melakukan pemisahan menjadi dua sub-kata, yang masing-masing di-stemming. Apabila stemming memberikan kata dasar yang sama, maka output kata dasarnya adalah hasil stemming tersebut. Namun apabila hasil stemming 2 sub-kata ini berbeda, maka dapat disimpulkan bahwa input adalah kata ulang semu, dan tidak memiliki bentuk kata dasar lagi.
4. Mengubah urutan *stemming* untuk beberapa kasus tertentu. Algoritma *stemmer* Nazief dan Adriani akan menghilangkan akhiran terlebih dahulu, baru diikuti penghilangan awalan. Menurut Jelita Asian, cara ini tidak selalu berhasil pada beberapa kata. Oleh karena itu, diberikan beberapa aturan yang akan mengubah urutan stemming, dimana penghilangan awalan dilakukan terlebih dahulu, lalu diikuti

penghilangan akhiran. Aturan ini disebut rule precedence, dan berlaku jika suatu kata memiliki pasangan awalan-akhiran "be-lah", "be-an", "me-i", "di-i", "pe-i", atau "te-i".

Pada Tabel 2.3, 2.4, dan 2.5, simbol C merupakan konsonan, simbol V menandakan vokal, simbol A merupakan vokal atau konsonan, dan simbol P merepresentasikan partikel atau fragmen dari suatu kata, misalnya "er".

Tabel 2.3 Aturan Pemenggalan Awalan Stemmer Nazief dan Andriani

Aturan	Format Kata	Pemenggalan
1	berV…	ber-V…| ber-rV…
2	berCAP…	ber-CAP…dimana C!='r' & P!='er'
3	berCAerV…	ber-CaerV…dimana C!='r'
4	Belajar	bel-ajar
5	beC_1erC_2 …	be-C_1erC_2… dimana C_1!={'r'|'l'}
6	terV…	ter-V…| te-rV…
7	terCerV…	ter-CerV… dimana C!='r'
8	terCP…	ter-CP… dimana C!='r' dan P!='er'
9	teC_1erC_2…	te-C_1erC_2… dimana C_1!='r'
10	me{l|r|w|y}V…	me-{l|r|w|y}V…
11	mem{b|f|v}…	mem-{b|f|v}…
12	mempe{r|l}…	mem-pe…

13	mem{rV|V}…	me-m{rV|V}… | me-p{rV|V}…
14	men{c|d|j|z}…	men-{c|d|j|z}…
15	menV…	me-nV… | me-Tv
16	meng{g|h|q}…	meng-{g|h|q}…
17	mengV…	meng-V… | meng-kV…
18	menyV…	meny-sV…
19	mempV…	mem-pV… dimana V!='e'
20	pe{w|y}V…	pe-{w|y}V…
21	perV…	per-V…| pe-rV…
23	perCAP…	per-CAP… dimana C!='r' dan P!='er'
24	perCAerV…	per-CAerV… dimana C!='r'
25	pem{b|f|V}…	Pem-{b|f|V}…
26	pem{rV|V}…	Pe-m{rV|V}… | pe-p{rV|V}…
27	pen{c|d|j|z}…	pen-{c|d|j|z}…
28	penV…	pe-nV… | pe-tV…
29	peng{g|h|q}…	peng-{g|h|q}…
30	pengV…	peng-V… | peng-kV…
31	penyV…	peny-sV…
32	pelV…	pe-lV… kecuali "pelajar" yang menghasilkan "ajar"
33	peCerV…	per-erV… dimana

		C!={r|w|y|l|m|n}
34	peCP…	pe-CP… dimana C!={r|w|y|l|m|n} dan P!='er'

Tabel 2.4 Modifikasi Aturan pada Tabel 2.3

Aturan	Format Kata	Pemenggalan
12	mempe…	mem-pe…
16	meng{g|h|q|k}…	meng-{g|h|q|k}…

Tabel 2.5 Tambahan Aturan untuk Tabel 2.3

Aturan	Format Kata	Pemenggalan
35	terC_1erC_2 …	ter-C_1erC_2… dimana C_1!='r'
36	peC_1erC_2 …	pe-C_1erC_2… dimana C_1!={r|w|y|l|m|n}

Algoritma *confix stripping* (CS) stemmer yang dikembangkan Jelita Asian bekerja sebagai berikut :

1. Cek rule *precedence*, apabila bernilai benar maka lakukan penghilangan awalan terlebih dahulu. Apabila bernilai salah, maka penghilangan akhiran dilakukan terlebih dahulu.
2. Lakukan recoding apabila diperlukan.
3. Cek apakah terdapat tanda hubung ('-') yang menandakan bahwa input kata tersebut adalah kata ulang. Jika benar, maka lakukan proses *stemming* pada potongan kata di sebelah kiri dan kanan tanda hubung tersebut. Apabila *stemming* dua kata ini memberikan hasil yang sama,

maka kata dasar kata ulang tersebut adalah hasil stemming yang didapatkan.

4. Jika ketiga proses di atas gagal, maka input kata yang di-stemming dianggap sebagai kata dasar. Pada setiap perpindahan langkah, dilakukan proses pencarian output stemming ke kamus. Apabila ditemukan di kamus, maka algoritma ini berhenti. Berikut adalah contoh proses *stemming* pada kata "menangkapnya" dengan menggunakan *confix stripping stemmer*:
    1) *Rule precedence* bernilai *false* karena tidak ditemukan kombinasi awalan dan akhiran yang tidak diperbolehkan.Pemenggalan akhiran dilakukan terlebih dahulu, dan menyisakan kata "menangkap".
    2) Pemenggalan awalan menyisakan kata "nangkap".
    3) Karena "nangkap" tidak terdapat di kamus, maka lakukan *recoding* dengan mengganti 'n' dengan 't' (aturan 15 pada Tabel 2.3).
    4) Kata "tangkap" ada di kamus, oleh karena itu proses berhenti. Kata dasar "menangkapnya" adalah "tangkap".

Sayangnya, setelah melakukan beberapa analisa dan percobaan kecil, ditemukan beberapa contoh kata yang tidak dapat di-*stemming*. Analisa atas beberapa kata yang gagal di-*stemming* tersebut adalah sebagai berikut:

1) Kurangnya aturan pemenggalan awalan untuk kata-kata dengan format "mem+p...". Hal ini terjadi pada kata "mempromosikan", "memproteksi", dan "memprediksi".
2) Kurangnya aturan pemenggalan awalan untuk kata-kata dengan format "men+s...". Hal ini terjadi pada kata "mensyaratkan", dan "mensyukuri".
3) Kurang relevannya aturan 17 untuk pemenggalan awalan pada kata-kata dengan format "menge+kata dasar", seperti pada kata "mengerem".

4) Kurang relevannya aturan 30 untuk pemenggalan awalan pada kata-kata dengan format "penge+kata dasar", seperti pada kata "pengeboman".
5) Kurangnya aturan pemenggalan awalan untuk kata-kata dengan format "peng+k..." seperti pada kata "pengkajian".
6) Adanya elemen pada beberapa kata dasar yang menyerupai suatu imbuhan. Kata-kata seperti "pelanggan", "perpolitikan", dan "pelaku" gagal di-stemming karena akhiran "-an", "-kan" dan "-ku" seharusnya tidak dihilangkan.

Berdasarkan kegagalan-kegagalan ini, maka untuk memperbaiki kesalahan algoritma CS *stemmer* dengan menambah beberapa perbaikan, diantaranya adalah sebagai berikut:

1) Merevisi aturan 19 pada Tabel 2.3 agar stemming berhasil pada kata-kata dengan *format* "mem+p...". Revisi aturan ini dapat dilihat pada Tabel 2.6.
2) Merevisi aturan 14 pada Tabel 2.3 agar stemming berhasil pada kata-kata dengan format "men+s...". Revisi aturan ini dapat dilihat pada Tabel 2.6.
3) Merivisi aturan 17 pada Tabel 2.3 agar stemming berhasil pada kata-kata dengan format "menge+...". Revisi aturan ini dapat dilihat pada Tabel 2.6.
4) Merivisi aturan 30 pada Tabel 2.3 agar stemming berhasil pada kata-kata dengan format "penge+...". Revisi aturan ini dapat dilihat pada Tabel 2.6.
5) Merevisi aturan 29 pada Tabel 2.3 agar stemming berhasil pada kata-kata dengan format "peng+k...". Revisi aturan ini dapat dilihat pada Tabel 2.6.
6) Menambahkan suatu algoritma tambahan untuk mengatasi kesalahan pemenggalan akhiran yang seharusnya tidak dilakukan. Algoritma ini disebut loopPengembalianAkhiran, dan dilakukan apabila proses recoding gagal.

Algoritma loopPengembalianAkhiran dideskripsikan sebagai berikut:

1. Kembalikan seluruh awalan yang telah dihilangkan sebelumnya, sehingga menghasilkan model kata seperti berikut: [DP+[DP+[DP]]] + Kata Dasar. Pemenggalan awalan dilanjutkan dengan proses pencarian di kamus

kemudian dilakukan pada kata yang telah dikembalikan menjadi model tersebut.

2. Kembalikan akhiran sesuai dengan urutan model. Ini berarti bahwa pengembalian dimulai dari DS (“-i”, “-kan”, “-an”), lalu PP(“-ku”, “-mu”, “-nya”), dan terakhir adalah P (“-lah”, “-kah”, “-tah”, “-pun”). Untuk setiap pengembalian, lakukan langkah 3) hingga 5) berikut. Khusus untuk akhiran “-kan”, pengembalian pertama dimulai dengan “k”, baru kemudian dilanjutkan dengan “an”.
3. Lakukan pengecekan di kamus. Apabila ditemukan, proses dihentikan. Apabila gagal, maka lakukan proses pemenggalan awalan berdasarkan aturan pada Tabel 2.3 (dengan revisi Tabel 2.6).
4. Lakukan *recoding* apabila diperlukan.
5. Apabila pengecekan di kamus tetap gagal setelah *recoding*, maka awalan-awalan yang telah dihilangkan dikembalikan lagi.

Tabel 2.6 Revisi untuk Tabel 2.3

Aturan	Format Kata	Pemenggalan
14	men{c|d|j|s|z}…	men-{c|d|j|s|z}…
17	mengV…	meng-V… |meng-kV…| (mengV-…jika V=’e’)
19	mempA…	mem-pA… dengan A!=’e’
29	pengC…	peng-C…
30	pengV…	peng-V… |peng-kV… | (pengV-…jika V=’e’)

2.2.19 *Analyzing*

Tahap *analyzing* merupakan tahap penentuan seberapa jauh keterhubungan antara kata-kata dengan dokumen yang ada

2.2.20 *Parser* (Analisis Sintaksis)

Parser atau *syntactic analyzer* pada kompilator bahasa pemrograman berfungsi untuk memeriksa kebenaran kemunculan setiap token.

Pada QA *system*, fungsi dari parser ini agak berbeda karena *token* yang akan diolah semua memiliki tipe yang sama yaitu berupa kata (*word*). Urutan kemunculan token yang berupa kata-kata tersebut akan diolah dengan mengacu pada *brain file* agar didapatkan makna kalimat yang sesungguhnya. Dengan kata lain, tahap analisa semantik terjadi di bagian brain file. Kemampuan dari *parser* untuk mengolah token dan bekerja sama dengan *brain file* inilah yang paling menentukan tingkat kecerdasan dari sebuah *chatbot*.

2.2.21 *Chatbot*

Chatbot adalah sebuah program komputer yang dirancang untuk mensimulasikan sebuah percakapan atau komunikasi yang interaktif kepada *user* (manusia) melalui bentuk *teks*, suara, dan atau *visual*.[7] Percakapan yang terjadi antara komputer dengan manusia merupakan bentuk respon dari program yang telah dideklarasikan pada database program pada komputer. Respon yang dihasilkan merupakan hasil pemindaian kata kunci pada inputan user dan menghasilkan respon balasan yang dianggap paling cocok, atau pola kata-kata yang dianggap paling mendekati, dari *database* tentunya. Dalam bahasa sehari-harinya *ChatBot* merupakan Aplikasi atau Program komputer yang dirancang untuk meniru manusia itu sendiri, batasan yang diambil dari *Chatbot* adalah mampu meniru komunikasi manusia. Jadi jika manusia sedang bercakap-cakap dengan program ini, seakan-akan ada dua pribadi manusia yang saling berkomunikasi. Nyatanya tidak, manusia berkomunikasi dengan *robot*. *Robot* sudah dirancang untuk merespon segala jenis pertanyaan dan pernyataan yang diinputkan oleh manusia (*user*). Hal ini terjadi karena sebelumnya sudah dideklarasikan pada *database*, berupa *entitas-entitas* kata, pola kalimat, dan berbagai jenis pernyataan dan pertanyaan.

Chatbot biasa digunakan dalam IRC (*Internet Chat Groups*) di mana robot tersebut melakukan berbagai fungsi seperti menyapa partisipan baru dan memonitor penggunaan kata-kata yang tidak pantas. Sekarang ini, robot dapat mengumpulkan informasi, mengolah informasi tersebut, dan memberikan respon terhadap pengguna dan *event* yang terjadi di internet. Beberapa jenis robot bekerja secara *silent* di internet tanpa sepengetahuan kita namun ada pula robot yang bekerja secara interaktif yang dapat kita ajak berbicara yaitu *chatbot.*

Chatbot adalah sebuah simulator percakapan yang berupa program komputer yang dapat berdialog dengan penggunanya dalam bahasa alami. Karena *chatbot* hanya sebuah program, dan bukan robot (*chatbot* tidak memiliki tubuh dan tidak memiliki mulut sehingga tidak dapat berbicara seperti manusia), maka yang dimaksud dengan dialog antara manusia sebagai pengguna dengan *chatbot* dilakukan dengan cara mengetik apa yang akan dibicarakan dan *chatbot* akan memberikan *respon*. Orang yang membuat dan mengembangkan program *chatbot* disebut *bot master*.

Chatbot yang pertama adalah ELIZA yang dibuat pada tahun 1964 sampai 1966 oleh Professor Joseph Weizenbaum di MIT (*Massachusetts Institute of Technology*), dengan tujuan untuk mempelajari komunikasi *natural language* antara manusia dengan mesin. Eliza bertindak seolah-olah dia adalah seorang psikolog yang dapat menjawab pertanyaan-pertanyaan dari pasien dengan jawaban yang cukup masuk akal atau menjawabnya dengan pertanyaan balik.

ELIZA adalah pelopor atau dapat disebut sebagai nenek moyangnya *chatbot*. ELIZA (juga dikenal sebagai Dr. Eliza) terkenal sebagai program *chat* yang "memiliki profesi" sebagai seorang psikiater. ELIZA mensimulasikan percakapan antara seorang psikiater dengan pasiennya menggunakan metode biasa yang dapat mencerminkan perasaan pasien dengan mengajukan pertanyaan-pertanyaan seperti: "*How do you ...*", *"Why do you feel like ...*", *"What do you think about ...*". Program ini akan mencari pola kata-kata tertentu pada input yang diberikan oleh pengguna, dan kemudian memberikan output yang sesuai.

ELIZA dapat memberikan respon yang tepat hanya jika pernyaataan pengguna ditulis dalam ejaan yang benar. Meskipun begitu, pengguna dapat berbicara secara santai dengan ELIZA.

ALICE (*Artificial Linguistic Internet Computer Entity*) adalah sebuah program *chatbot* yang diciptakan oleh Dr. Richard Wallace. ALICE mampu menciptakan suasana seperti komunikasi antara dua manusia seperti biasanya, ALICE bahkan dapat berbohong, menyebarkan gosip, dan mengetahui banyak kejadian yang terjadi di dunia nyata.

Ada banyak lagi chatbot lain yang menarik seperti Erin the Bartender, Madame Celeste, Mega HAL, dan lain-lain.

2.2.22 Penggunaan *Brainfile* Pada *Chatbot*

Brain file merupakan otak dari chat bot itu sendiri yang menentukan bagaimana cara chat bot berpikir dan akan memberikan respon. *Brain file* biasanya berupa *file plain text*. *Brain file* berfungsi sebagaimana tabel informasi pada kompilator bahasa pemrograman tingkat tinggi. Di dalam *brain file* inilah disimpan semua kosakata, kepribadian, dan pengetahuan (*knowledge*) dari *chatbot*. Semakin banyak pengetahuan yang dimiliki *chatbot* maka akan semakin besar ukuran *file* dari *brain file* tersebut.

2.2.23 Perangkat Lunak

Perangkat Lunak (*software*) merupakan suatu program yang dibuat oleh pembuat program untuk menjalankan perangkat keras komputer. Perangkat Lunak adalah program yang berisi kumpulan *instruksi* untuk melakukan proses pengolahan data. *Software* sebagai penghubung antara manusia sebagai pengguna dengan perangkat keras komputer, berfungsi menerjemahkan bahasa manusia ke dalam bahasa mesin sehingga perangkat keras komputer memahami keinginan pengguna dan menjalankan *instruksi* yang diberikan dan selanjutnya memberikan hasil yang diinginkan oleh manusia tersebut.

Perangkat lunak terdiri dari kode-kode yang dibuat atau ditulis dalam bahasa pemrograman tertentu. Banyak bahasa pemrograman yang dapat dijadikan alat bantu untuk merancang sebuah perangkat lunak antara lain bahasa PASCAL, C, C++, BASIC, Java, HTML, PHP, JavaScript, AIML, dan bahkan tersedia *tool* yang berbasis *grafis* seperti Visual Basic, Visual C++, Borland Delphi, dan masih banyak lagi tool dan bahasa pemrograman lainnya.

2.2.24 Teknologi Web

Web adalah sebutan bagi sekelompok halaman web (*web page*), yang umumnya merupakan bagian dari suatu nama domain (*domain name*) atau subdomain di *World Wide* Web (WWW) di Internet. WWW terdiri dari seluruh situs web yang tersedia kepada publik.

Halaman-halaman sebuah situs web diakses dari sebuah URL yang menjadi "akar"(*root*), yang disebut homepage dan biasanya disimpan dalam *server* yang sama. Web memiliki kemampuan link yang sangat bagus. Keistimewaan inilah yang telah menjadikan Web sebagai *service* yang paling cepat pertumbuhannya. Web mengizinkan pemberian *highlight* pada kata atau gambar dalam sebuah dokumen untuk dihubungkan ke media lain seperti dokumen, frase, *movie clip*, atau *file* suara.

Tipe yang paling populer adalah dokumen hypertext yang disusun menurut bahasa khusus, seperti *Hypertext Markup Language* (HTML) sedangkan untuk interaktif web atau *dynamic page* terdapat banyak pilihan, salah satunya adalah PHP *Hypertext Preprocessor* (PHP).

Pada prinsipnya web bekerja dengan cara menampilkan *file* PHP yang berasal dari server web pada program client, yaitu *browser web*. Program browser pada client mengirim permintaan kepada *server* web, yang kemudian dikirimkan oleh *server* dalam bentuk Format HTML. *File* PHP berisi instruksi-instruksi yang diperlukan untuk membentuk *dynamic page*. Perintah-perintah PHP ini kemudian diterjemahkan oleh *browser* sehingga isi informasinya dapat ditampilkan secara *visual* kepada pengguna di layar komputer. Pada penggunaannya, mekanisme web juga dapat diakses melalui *Local Area Network* (LAN) sehingga tidak perlu terhubung dengan internet.

2.2.25 Pengertian HTML

HTML (*Hyper Text Markup Language*) adalah sebuah bahasa markup yang digunakan untuk membuat sebuah halaman *web* dan menampilkan berbagai informasi di dalam sebuah browser Internet. Bermula dari sebuah bahasa yang sebelumnya banyak digunakan di dunia penerbitan dan percetakan yang disebut dengan *SGML* (*Standard Generalized Markup Language*), *HTML* adalah sebuah standar yang digunakan secara luas untuk menampilkan halaman *web*. HTML saat

ini merupakan standar Internet yang didefinisikan dan dikendalikan penggunaannya oleh *World Wide Web Consortium (W3C).*

HTML berupa kode-kode tag yang menginstruksikan browser untuk menghasilkan tampilan sesuai dengan yang diinginkan. Sebuah *file* yang merupakan *file HTML* dapat dibuka dengan menggunakan *browser web* seperti Mozilla Firefox atau Microsoft Internet Explorer. *HTML* juga dapat dikenali oleh aplikasi pembuka *email* ataupun dari *PDA* dan program lain yang memiliki kemampuan *browser.*

HTML dokumen tersebut mirip dengan dokumen teks biasa, hanya dalam dokumen ini sebuah teks bisa memuat instruksi yang ditandai dengan kode atau lebih dikenal dengan *TAG* tertentu. Sebagai contoh jika ingin membuat teks ditampilkan menjadi tebal seperti: TAMPIL TEBAL, maka penulisannya dilakukan dengan cara: TAMPIL TEBAL. Tanda digunakan untuk mengaktifkan instruksi cetak tebal, diikuti oleh teks yang ingin ditebalkan, dan diakhiri dengan tanda untuk menonaktifkan cetak tebal tersebut.

2.2.26 Pengertian PHP

PHP pertama kali dibuat oleh Rasmus Lerdorf pada tahun 1995. Pada waktu itu *PHP* bernama FI (Form Interpreted). Pada saat tersebut *PHP* adalah sekumpulan script yang digunakan untuk mengolah data *form* dari *web.*

Web Server yang mendukung php dapat ditemukan dimana - mana dari mulai IIS sampai dengan apache, dengan *configurasi* yang relatif mudah. Dalam sisi pengembangan lebih mudah, karena banyaknya *milis - milis* dan *developer* yang siap membantu dalam pengembangan. Dalam sisi pemahamanan, *php* adalah bahasa *scripting* yang paling mudah karena referensi yang banyak.

PHP adalah bahasa *open source* yang dapat digunakan di berbagai mesin (linux, unix, windows) dan dapat dijalankan secara runtime melalui *console* serta juga dapat menjalankan perintah-perintah *system.* [5]

Sistem manajemen database yang sering digunakan bersama PHP adalah MySQL. Namun PHP juga mendukung system manajemen Database Oracle, Microsoft Acces, Interbase, d-Base, PostgreSQL dan sebagainya.

PHP membuat proses pengembangan aplikasi menjadi mudah

karena kelebihan-kelebihannya yaitu:

1. *Script developer* (kode program) terintegrasi dengan file HTML, sehingga dapat berkonsentrasi langsung pada penampilan dokumen *web*nya.
2. Tidak ada proses *compiling* dan *linking*.
3. Berorientasi objek (*Object Oriented*).
4. Sintaksis pemogramannya mudah dipelajari, sangat menyerupai C dan Perl.
5. Integrasi yang sangat luas ke berbagai server database.

Script PHP berkedudukan sebagai tag dalam bahsa HTML. Suatu skrip akan dikenali sebagai script PHP bila diapit oleh tanda :

a. <?php……?>
b. <?.....?>
c. <script language=”PHP”>……</script>

Script yang dibuat dengan PHP disimpan dengan nama *file* dan diikuti dengan ekstensi *.php, misalnya : contoh.php. Bila script PHP diakses melalui *komputer local* maka file PHP disimpan di folder htdocs di local *web server*.

2.2.27 *Object Oriented Programming* (OOP)

Object-Oriented Programming (OOP) adalah sebuah pendekatan untuk pengembangan / development suatu *software* dimana dalam struktur *software* tersebut didasarkan kepada interaksi object dalam penyelesaian suatu proses/tugas. Interaksi tersebut mengambil *form* dari pesan-pesan dan mengirimkannya kembali antar *object* tersebut. *Object* akan merespon pesan tersebut menjadi sebuah tindakan / *action* atau metode.

Object-oriented programs terdiri dari *objects* yang berinteraksi satu sama lainnya untuk menyelesaikan sebuah tugas. Seperti dunia nyata, *users* dari *software* programs dilibatkan dari logika proses untuk menyelesaikan tugas. Adapun karakteristik yang dimiliki OOP yaitu :

1. *Objects*

Object adalah sebuah *structure* yang menggabungkan data dan prosedur untuk bekerja bersama-sama.

2. *Abstraction*

   Ketika manusia berinteraksi dengan *object-object* di dunia ini, manusia sering hanya konsentrasi dengan sebuah bagian dari *properties*nya. Ketika membangun *objects* dalam aplikasi OOP, adalah penting untuk menggabungkan konsep *abstraction* ini. Jika membangun aplikasi *shipping*, maka harus membangun object produk dengan atribut seperti ukuran dan berat. Warna adalah contoh informasi yang tidak ada hubungannya dan harus dibuang. Tetapi ketika akan membangun *order entry application*, warna menjadi penting dan harus termasuk atribut *object* produk.

3. *Encapsulation*

   Ciri penting lainnya dari OOP adalah *encapsulation. Encapsulation* adalah sebuah proses dimana tidak ada akses langsung ke data yang diberikan, bahkan *hidden*. Jika ingin mendapat data, harus berinteraksi dengan *object* yang bertanggung jawab atas data tersebut.

4. *Polymorphism*

   Polymorphisms adalah kemampuan 2 buah object yang berbeda untuk merespon pesan permintaan yang sama dalam suatu cara yang unik. Dalam OOP, diterapkan tipe polymorphism melalui proses yang disebut *overloading*. Dapat dilakukan dalam *implementasi* metode yang berbeda pada sebuah *object* yang mempunyai nama yang sama.

5. *Inheritance*

   Penggunaan *inheritance* dalam OOP untuk mengklasifikasikan *objects* dalam program sesuai karakteristik umum dan fungsinya.

2.3 Metode Alat Analisis Yang digunakan

2.3.1 Flowchart

Flowchart merupakan gambaran dalam bentuk diagram alir dari *algoritma-algoritma* dalam suatu program, yang menyatakan arah alur program

tersebut. Hal tersebut memungkinkan untuk memecah proses menjadi kejadian-kejadian individual atau aktifitas untuk menunjukan secara singkat hubungan diantaranya. Konstruksi *flowchart* memungkinkan pengertian lebih baik kepada proses dan pengertian yang lebih baik terhadap proses akan membawa kepada perbaikan pengembangan suatu sistem. Adapun simbol-simbol yang digunakan dalam *flowchart*, dapat dilihat dalam daftar simbol.

2.3.2 Unified Modeling Language (UML)

UML (Unified Modeling Language) adalah sebuah bahasa untuk menetukan, visualisasi, kontruksi, dan mendokumentasikan *artifact* bagian dari informasi yang digunakan atau dihasilkan dalam suatu proses pembuatan perangkat lunak. *Artifact* dapat berupa model, deskripsi atau perangkat lunak) dari system perangkat lunak, seperti pada pemodelan bisnis dan *system non* perangkat lunak lainnya [5].

UML merupakan suatu kumpulan teknik terbaik yang telah terbukti sukses dalam memodelkan *system* yang besar dan kompleks. UML tidak hanya digunakan dalam proses pemodelan perangkat lunak, namun hampir dalam semua bidang yang membutuhkan pemodelan.

Bagian-bagian utama dari UML adalah *view*, diagram, model *elemen*t, dan *general mechanism*.

1. *View*

   View digunakan untuk melihat sistem yang dimodelkan dari beberapa aspek yang berbeda. *View* bukan melihat grafik, tapi merupakan suatu abstraksi yang berisi sejumlah diagram. Beberapa jenis *view* dalam UML antara lain: *use case view, logical view, component view, concurrency view, dan deployment view.*

a) Use case view

Mendeskripsikan fungsionalitas sistem yang seharusnya dilakukan sesuai yang diinginkan *external actors*. *Actor* yang berinteraksi dengan sistem dapat berupa user atau sistem lainnya. *View* ini digambarkan dalam *use case diagrams*

dan kadang-kadang dengan *activity diagrams*. *View* ini digunakan terutama untuk pelanggan, perancang (*designer*), pengembang (*developer*), dan penguji sistem (*tester*).

b) Logical view

Mendeskripsikan bagaimana fungsionalitas dari sistem, struktur statis (*class, object,dan relationship*) dan kolaborasi dinamis yang terjadi ketika *object* mengirim pesan ke *object* lain dalam suatu fungsi tertentu. *View* ini digambarkan dalam class diagrams untuk struktur statis dan dalam *state, sequence, collaboration, dan activity diagram* untuk model dinamisnya. View ini digunakan untuk perancang (*designer*) dan pengembang (*developer*).

c) Component view

Mendeskripsikan implementasi dan ketergantungan modul. Komponen yang merupakan tipe lainnya dari code module diperlihatkan dengan struktur dan ketergantungannya juga alokasi sumber daya komponen dan informasi administrative lainnya. View ini digambarkan dalam *component view* dan digunakan untuk pengembang (*developer*).

d) Concurrency view

Membagi sistem ke dalam proses dan prosesor. *View* ini digambarkan dalam diagram dinamis *(state, sequence, collaboration, dan activity diagrams)* dan diagram implementasi *(component dan deployment diagrams)* serta digunakan untuk pengembang (*developer*), pengintegrasi (*integrator*), dan penguji (*tester*).

e) Deployment view

Mendeskripsikan fisik dari sistem seperti komputer dan perangkat (*nodes*) dan bagaimana hubungannya dengan lainnya. *View* ini digambarkan dalam *deployment diagrams* dan digunakan untuk pengembang (*developer*), pengintegrasi (*integrator*), dan penguji (*tester*).

2. Diagram

Diagram berbentuk grafik yang menunjukkan simbol elemen model yang disusun untuk mengilustrasikan bagian atau aspek tertentu dari sistem. Sebuah

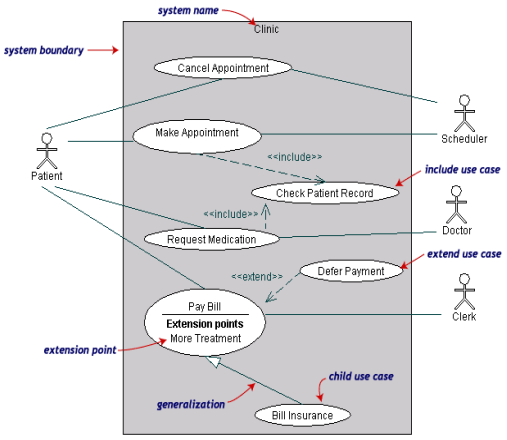
diagram merupakan bagian dari suatu *view* tertentu dan ketika digambarkan biasanya dialokasikan untuk *view* tertentu. Adapun jenis diagram antara lain :

a) *Use Case Diagram*

Use case adalah abstraksi dari interaksi antara sistem dan actor. *Use case* bekerja dengan cara mendeskripsikan tipe interaksi antara *user* sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem dipakai. *Use case* merupakan konstruksi untuk mendeskripsikan bagaimana sistem akan terlihat di mata *user*. Sedangkan *use case* diagram memfasilitasi komunikasi diantara analis dan pengguna serta antara analis dan *client*.

Use case diagram dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem. Contoh *use case diagram* dapat dilihat pada gambar 2.13

.



Gambar 2.13 *Use Case Diagram*

b) *Class Diagram*

Class adalah dekripsi kelompok obyek-obyek dengan *property*, perilaku (operasi) dan relasi yang sama. Sehingga dengan adanya *class* diagram dapat memberikan pandangan global atas sebuah sistem. Hal tersebut tercermin dari class- class yang ada. Sebuah sistem biasanya mempunyai beberapa *class*

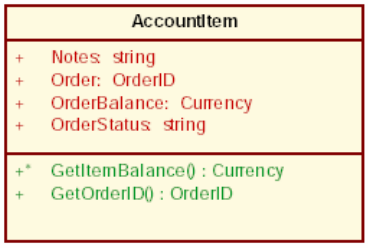
diagram. *Class* diagram sangat membantu dalam visualisasi struktur kelas dari suatu sistem. *Class* memiliki tiga area pokok :

1. Nama (dan *stereotype*)
2. Atribut
3. Metoda

Atribut dan metoda dapat memiliki salah satu sifat berikut :

a. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan.
b. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya.
c. *Public*, dapat dipanggil oleh siapa saja.

Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class*. Contoh *class diagram* dapat dilihat pada gambar 2.14



Gambar 2.14 *Class Diagram*

c) *Component Diagram*

Component software merupakan bagian fisik dari sebuah sistem, karena menetap di komputer tidak berada di benak para analis. Komponen merupakan implementasi *software* dari sebuah atau lebih *class*. Komponen dapat berupa *source code*, komponen biner, atau *executable component*. Sebuah komponen berisi informasi tentang *logic class* atau *class* yang diimplementasikan sehingga membuat pemetaan dari *logical view* ke *component view*. Sehingga component diagram merepresentasikan dunia nyata yaitu *component software* yang mengandung *component, interface dan relationship*.

d) *Deployment Diagram*

Menggambarkan tata letak sebuah sistem secara fisik, menampakkan bagian-bagian *software* yang berjalan pada bagian-bagian *hardware*, menunjukkan hubungan komputer dengan perangkat (*nodes*) satu sama lain dan jenis hubungannya. Di dalam *nodes*, *executeable component* dan *object* yang dialokasikan untuk memperlihatkan unit perangkat lunak yang dieksekusi oleh *node* tertentu dan ketergantungan komponen.

e) *State Diagram*

Menggambarkan semua *state* (kondisi) yang dimiliki oleh suatu *object* dari suatu *class* dan keadaan yang menyebabkan *state* berubah. Kejadian dapat berupa *object* lain yang mengirim pesan. *State class* tidak digambarkan untuk semua *class*, hanya yang mempunyai sejumlah *state* yang terdefinisi dengan baik dan kondisi *class* berubah oleh *state* yang berbeda.

f) *Sequence Diagram*

Sequence diagram digunakan untuk menggambarkan perilaku pada sebuah *scenario*. Kegunaannya untuk menunjukkan rangkaian pesan yang dikirim antara *object* juga interaksi antara *object*, sesuatu yang terjadi pada titik tertentu dalam eksekusi sistem.

g) *Collaboration Diagram*

Menggambarkan kolaborasi dinamis seperti *sequence* diagrams. Dalam menunjukkan pertukaran pesan, *collaboration* diagrams menggambarkan *object* dan hubungannya (mengacu ke konteks). Jika penekannya pada waktu atau urutan gunakan *sequence* diagrams, tapi jika penekanannya pada konteks gunakan *collaboration* diagram.

h) *Activity Diagram*

Menggambarkan rangkaian aliran dari aktivitas, digunakan untuk mendeskripsikan aktifitas yang dibentuk dalam suatu operasi sehingga dapat juga digunakan untuk aktifitas lainnya seperti *use case* atau interaksi.

2.3.3 Database

Database atau Basis Data adalah sekumpulan data yang saling terhubung satu dengan yang lainnya atau sekumpulan *table* yang saling terhubung satu

dengan yang lainnya.dan fungsi dari database adalah menyimpan suatu data pada *table-tabel* dan dikumpulkan menjadi satu dengan *database. Database* juga bisa di umpamakan sebagai sebuah rumah dengan beberapa kamar-kamar dan sebuah *property* seperti almari meja belajar tempat tidur itu bisa di sebut dengan data *querynya.*

Basis data terdiri dari 2 kata, yaitu basis dan data. Basis dapat diartikan sebagai tempat berkumpul, sedangkan data adalah representasi fakta dunia nyata yang mewakili suatu objek seperti manusia, barang, hewan, peristiwa, dan sebagainya yang direkam dalam bentuk angka, huruf, symbol, teks, gambar, bunyi atau kombinasinya.

Ada beberapa operasi dasar yang dapat dilakukan berkenaan dengan basis data, antara lain:

1. Pembuatan basis data baru (*create database*)
2. Penghapusan basis data (*drop database*)
3. Pembuatan file/tabel baru ke suatu basis data (*create table*)
4. Penghapusan file/tabel dari suatu basis data (*drop table*)
5. Penambahan/pengisian data baru ke sebuah file/tabel *(insert*)
6. Pengambilan/pencarian data dari sebuah file/tabel (*retrieve/search*)
7. Pengubahan data dari sebuah file/tabel (*update*)
8. Penghapusan data dari sebuah file/tabel (*delete*)

Istilah-istilah dalam database :

1. *Entity*

   Entity adalah orang, tempat, kejadian atau konsep yang informasinya direkam. Misalnya pada kampus terdapat *entity* mahasiswa, matakuliah, dosen, nilai test dan lain-lain.

2. *Atribute*

   Setiap *entity* mempunyai atribute atau sebutan untuk mewakili suatu *entity*. Seorang mahasiswa dapat dilihat dari atributenya, misalnya nim, nama, alamat, jenis kelamin dan lain-lain. Atribute juga disebut sebagai data elemen, data *field,* data item.

3. *Data value* (nilai atau isi data)

Data value adalah data *actual* atau informasi yang disimpan pada tiap data elemen atau atribute. Atribute nama mahasiswa menunjukan tempat dimana informasi nama mahasiswa disimpan, sedang data *value* adalah dali, andi, merupakan isi data nama mahasiswa tersebut.

4. *Record* (tupel)

   Record yaitu kumpulan elemen-elemen yang saling berkaitan menginformasikan tentang suatu entity secara lengkap. Satu *record* mewakili satu data atau informasi tentang seseorang misalnya, nomor induk mahasiswa, nama, alamat, jenis kelamin dan seterusnya.

5. *File*

   File adalah kumpulan *record-record* sejenis yang mempunyai panjang elemen yang

   sama, atribute yang sama, namun berbeda-beda data *valuenya*.

6. *Database*

   Database adalah kumpulan *file-file* yang mempunyai kaitan antara satu *file* dengan *file* yang lain sehingga membentuk satu bangunan data untuk menginformasikan satu perusahaan, instansi dalam batasan tertentu.

7. *Database Management System* (DBMS)

   Database Management System (DBMS) adalah program untuk pengelolaan disebut DBMS. Database adalah kumpulan datanya, sedangkan program pengelolanya berdiri sendiri dalam satu paket program yang berfungsi untuk membaca data, mengisi data, menghapus data serta melaporkan data dalam *database.*

2.3.4 Pengertian MYSQL

MySQL adalah sebuah perangkat lunak Pembuat database yang bersifat terbuka atau open source dan berjalan disemua platform baik Linux maupun Sistem Windows, MySQL merupakan program pengakses *database* yang bersifat network sehingga dapat digunakan untuk aplikasi *Multi User* (Pengguna Banyak).

MySQL adalah *Relational Database Management System* (*RDBMS*) yang didistribusikan secara gratis dibawah lisensi *GPL* (*General Public License*). [8] Dimana setiap orang bebas untuk menggunakan MySQL, namun tidak boleh dijadikan produk turunan yang bersifat komersial.

MySQL dapat dikatakan lebih unggul dibandingkan *database server* lainnya dalam *query* data. Hal ini terbukti untuk *query* yang dilakukan oleh single user, kecepatan *query* MySQL bisa sepuluh kali lebih cepat dari PostgreSQL dan lima kali lebih cepat dibandingkan *Interbase.*

2.3.5 *Web Browser*

Web Browser adalah suatu program atau *software* yang digunakan untuk menjelajahi internet atau untuk mencari informasi dari suatu web yang tersimpan didalam komputer. Awalnya, *web browser* berorientasi pada teks dan belum dapat menampilkan gambar. Namun, web browser sekarang tidak hanya menampilkan gambar dan teks saja, tetapi juga memutar file multimedia seperti video dan suara. *Web browser* juga dapat mengirim dan menerima email, mengelola HTML, sebagai input dan menjadikan halaman web sebagai hasil output yang informative.

Dengan menggunakan *web browser*, para pengguna internet dapat mengakses berbagai informasi yang terdapat di internet dengan mudah.

Fungsi *Web Browser* adalah untuk menampilkan dan melakukan interaksi dengan dukumen-dokumen yang disediakan *oleh web server.*

Contoh *Web Browser* :

1. Mozilla Firefox yaitu dibuat oleh mozilla corporation, firefox adalah salah satu web browser open source yang dibangun dengan Gecko *layout engine*. Tak hanya handal firefox juga didukung oleh sejumlah *Add-ons* yang dapat diinstall terpisah yang memungkinkan pengguna melakukan sesuai dengan kegunaan *Add-ons* tersebut.
2. Internet Exporer yaitu *Web browser* besutan Microsoft Corporation biasanya dikenal dengan nama pendek IE, sejak 1995 IE mulai di masukan sebagai *default sotware* pada saat instalasi Sistem Operasi Windows, sejak tulisan ini dibuat IE belum lama ini meluncurkan versi IE8.
3. Safari yaitu Dibuat oleh Apple Inc, perusahaan yang juga memproduksi komputer Macintosh, iPod, dan juga iPhone. dibangun dengan *browser engine* WebKit, WebKit juga adalah *browser engine* pertama yang lulus *test* Acid3.

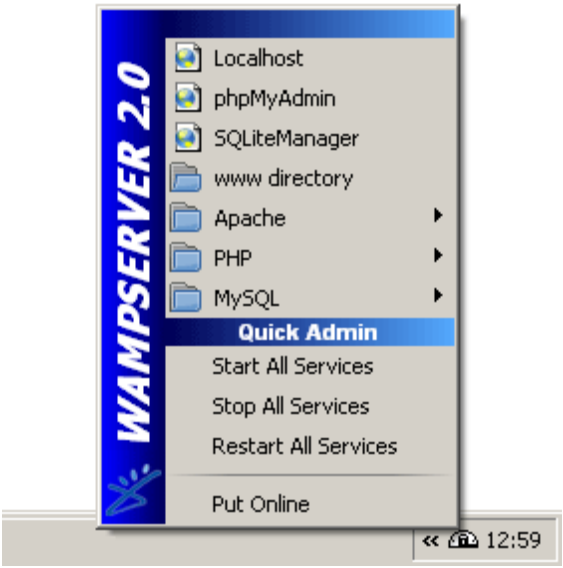
4. Flock yaitu Flock adalab *web browser* yang dibangun dengan *code* mozilla frefox yang web browser ini khususkan menyediakan *social networking* dan Web 2.0. Flock didesain untuk memudahkan aktivitas online pengguna internet mengatur beberapa *social networking, web mail, news feeds* dan *blogs* yang mereka miliki. Dengan Mengunakan Flock mereka dapat dengan mudah menjelajah, berbagi, dan menikmati content maupun menjalin hubungan di situs pertemanan yang mereka inginkan.
5. Opera yaitu Opera dikembangkan oleh Opera Software company adalah salah satu Web Browser dan juga Internet Suite. Jika firefox punya *Add-ons*, Opera punya "Opera Widgets", sebuah aplikasi web kecil yang dijalankan bersamaan dengan Opera yang mempunyai kegunaan tertentu, layaknya *Add-ons* firefox.

2.3.6 Wampserver

Wampserver adalah paket *web server* yang bekerja secara pada localhost yang dibuat secara independen dan di instal pada sistem operasi Windows.

WAMP adalah singkatan dari Windows and the principal components of *the package* : Apache, MySQL and PHP (or Perl or Python).

Apache adalah *Web server,* MySQL adalah database, PHP adalah bahasa *scripting* yang dapat memanipulasi informasi yang dibuat di database dan menghasilkan halaman web dinamis konten setiap waktu diminta oleh browser. Program lain juga dapat dimasukkan dalam paket, seperti php MyAdmin yang menyediakan antarmuka pengguna grafis untuk manajer database MySQL, atau bahasa scripting Python alternatif atau Perl.



Gambar 2.15 Wampserver

2.3.7 Pengertian Macromedia Dreamweaver

Pengertian Macromedia Dreamweaver adalah sebuah *software* HTML editor profesional yang digunakan untuk mendesain secara visual dan mengelola situs *web* maupun halaman *web*. Bilamana kita menyukai untuk berurusan dengan kode-kode HTML secara manual atau lebih menyukai bekerja dengan lingkungan secara *visual* dalam melakukan *editing*, Dreamweaver membuatnya menjadi lebih mudah dengan menyediakan *tools* yang sangat berguna dalam peningkatan kemampuan dan pengalaman kita dalam mendesain *web*.

Dreamweaver 8 dalam hal ini digunakan untuk *web desain*. Dreamweaver 8 mengikutsertakan banyak *tools* untuk kode-kode dalam halaman *web* beserta fasilitas-fasilitasnya, antara lain : Referensi HTML, CSS dan Javascript, Javascript debugger, dan editor kode (tampilan kode dan *Code inspector*) yang mengizinkan kita mengedit kode Javascript, XML, dan dokumen teks lain secara langsung dalam Dreamweaver.