BAB II
LANDASAN TEORI

2.1 Finite Automata

2.1.1 Definisi Finite Automata

Finite Automata adalah suatu pemodelan yang dapat digunakan terhadap berbagai *hardware* ataupun *software* yang terdiri dari sejumlah *state* hingga *state akhir* (*finite state*), sehingga suatu sistem dengan sekumpulan *resource* yang terbatas dapat diimplementasikan. *Finite Automata* melibatkan sejumlah *state* serta transisi antar *state* sebagai bentuk respon terhadap *input* yang diberikan. (Hopcroft, 2007). *Finite Automata* dapat juga disebut sebagai *finite-state machine* yang dapat diartikan sebagai suatu mesin komputasi abstrak dan merupakan suatu teknik komputasi yang sangat penting dalam pemrosesan pengucapan bahasa (*spoken language processing*) (Coleman, 2005).

2.1.2 Deterministic Finite Automata dan Nondeterministic Finite Automata

Secara garis besar, *Finite Automata* terbagi menjadi dua jenis, yaitu *Deterministic Finite Automata* (DFA) dan *Nondeterministic Finite Automata* (NFA) (Mozgovoy, 2010). Berikut adalah penjelasan dari kedua jenis *Finite Automata* tersebut.

2.1.3 Deterministic Finite Automata

Deterministic Finite Automata (DFA) adalah suatu representasi *Finite Automata* dimana sistem hanya dapat berada pada suatu *state* tunggal setelah diberikan rangkaian *input*. Istilah ”*deterministic*” sendiri mengacu pada fakta bahwa untuk setiap *input*, hanya terdapat satu *state* dimana sistem dapat

bertransisi dari *state*nya sekarang. Notasi dari DFA (disebut dengan *five tuple notation*) adalah sebagai berikut.

$$A = (Q, \Sigma, \delta, q_0, F)$$

Gambar 2.1 Gambar Notasi *Five Tuple*

Dengan masing – masing penjelasan setiap simbol adalah sebagai berikut.

Q = kumpulan *state* berhingga (*finite state*)

Σ = kumpulan simbol – simbol *input* berhingga

δ = fungsi transisi

q_0 = *state* awal (*start state*)

F = *state* akhir (*final state*)

Suatu *string* dikatakan "diterima" oleh DFA apabila setelah ditelusuri, *string* tersebut habis terbaca dan DFA berhenti di *state* akhir (*final state*). Istilah *language* digunakan untuk menunjuk kumpulan *string* yang dapat diterima oleh DFA.

Terdapat dua representasi lain yang dapat digunakan untuk mendefinisikan suatu DFA, yaitu diagram transisi *(transition diagrams*) dan tabel transisi (*transition table*).

a. Diagram transisi

Adalah suatu *graph* yang menggambarkan notasi *five tuple notation*, dengan rincian sebagai berikut.

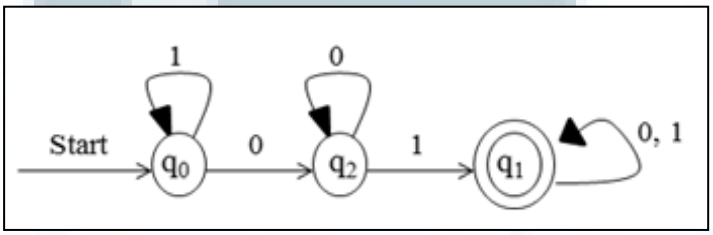
1) Untuk setiap *state* di Q direpresentasikan dengan sebuah *node.*

2) Untuk setiap state q di Q dan simbol input a dalam, sehingga terdapat fungsi transisi δ(q, a) = p, maka diagram transisi akan memiliki garis penghubung/arc dari node q ke node p dengan label a. Jika terdapat beberapa simbol *input* yang menyebabkan terjadinya transisi dari q ke p, maka diagram transisi dapat digambarkan memiliki satu arc dengan labelnya adalah simbol – simbol *input* tersebut.

3) Ada suatu anak panah ke *state* awal q_0 yang diberi label *Start*. Anak panah ini tidak berasal dari *node* manapun.

4) *Node – node* yang berada pada *state* penerima (yang terdapat dalam *state* akhir/F) ditandai dengan lingkaran ganda. *State* yang tidak terdapat dalam F ditandai dengan lingkaran tunggal.

Berikut adalah ilustrasi sederhana DFA yang menerima semua *string* dengan *substring* 01.

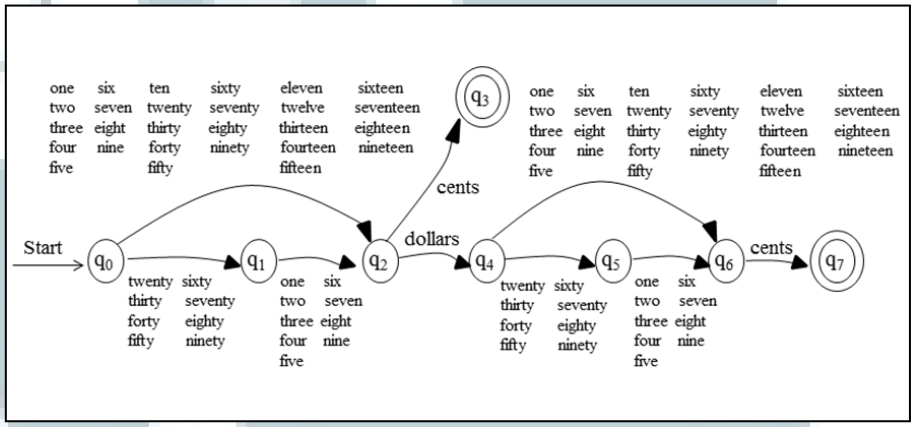


Gambar 2.2 Diagram Transisi DFA Penerima String Dengan *Substring* 01
(Sumber: Hopcroft, 2007)

Pada diagram tersebut, dapat dilihat bahwa untuk mencapai *state* akhir, dibutuhkan dua *substring* mutlak, yaitu "01", karena hanya *input* '0' yang akan menyebabkan transisi dari q_0 ke q_2, dan hanya *input* '1' yang akan menyebabkan transisi dari q_2 ke q_1 sebagai *state* akhir (dengan asumsi simbol *input* adalah 0 dan 1). Di *state* akhir sendiri (q_1), *string* akan tetap diterima apapun *input*-nya, karena

di *state* tersebut, tujuan dari DFA telah tercapai, yaitu menerima *string* dengan *substring* 01.

DFA juga dapat digunakan sebagai pemodelan kombinasi kata – kata. Berikut adalah contoh pemodelan DFA untuk kata – kata penyusun nilai mata uang Inggris.



Gambar 2.3 DFA untuk Kata – Kata Penyusun Nilai Mata Uang Inggris (Sumber: Jurafsky, 2009)

Diagram DFA tersebut menggambarkan kombinasi kata penyusun mata uang Inggris yang dibatasi pada *dollar* dan *cents*. Dapat dilihat bahwa pengucapan mata uang dapat berupa sekian *cent* saja atau sekian *dollars* sekian *cent*. Pecahan nilai mata uangnya sendiri dibatasi antara satu sampai dengan 99 dan digambarkan pada rangkaian *state* q_0 sampai dengan q_2 dan q_4 sampai dengan q_6. Transisi langsung antara q_0 ke q_2 dan antara q_4 ke q_6 berfungsi untuk menerima pecahan satu sampai 20 dan puluhan bulat (30, 40, 50, dan seterusnya hingga 90). Transisi q_0-q_1-q_2 dan q_4-q_5-q_6 berfungsi untuk menerima pecahan puluhan tidak bulat (21, 22, 23, 24, dan seterusnya hingga 99).

b. Tabel transisi

Tabel transisi adalah representasi tabular dari fungsi transisi δ yang membutuhkan dua argumen dan mengembalikan suatu nilai. Baris tabel menggambarkan *state* dan kolom tabel menggambarkan *input*-nya. Nilai isi dari baris *state* q dan kolom *input* a adalah *state* hasil dari δ(q, a). Berikut adalah contoh tabel transisi dari diagram transisi yang terdapat pada gambar 2.1

Tabel 2.1 Tabel Transisi DFA Penerima *String* dengan *Substring* "01"

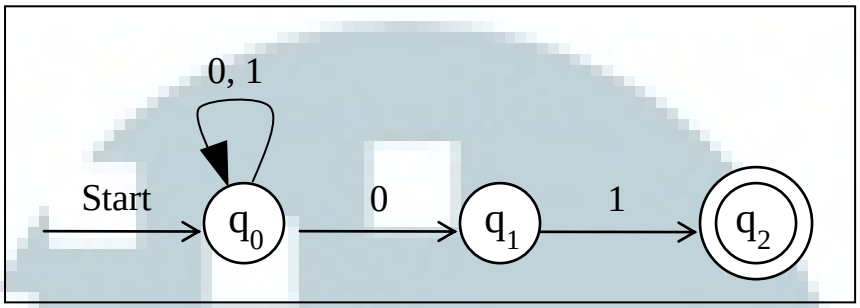
	0	1
→ q_0	q_2	q_0
*q_1	q_1	q_1
q_2	q_2	q_1

(Sumber: Hopcroft, 2007)

2.1.4 Nondeterministic Finite Automata

Nondeterministic Finite Automata (NFA) adalah representasi lain dari *Finite Automata* dimana suatu sistem mampu berada di beberapa *state* dalam satu waktu. Kemampuan ini seringkali diungkapkan sebagai kemampuan untuk "mengira – ngira" *input* yang diberikan. Sebagai contoh, ketika digunakan untuk mencari sekumpulan rangkaian karakter (misalnya: suatu kata kunci/*keyword*) dalam suatu *string* yang panjang, dapat dilakukan "perkiraan" bahwa kita sedang berada di awal salah satu huruf penyusun *string* tersebut dan menggunakan serangkaian *state* untuk melakukan pengecekan kemunculan *string* secara per karakter. Notasi yang digunakan untuk merepresentasikan NFA adalah *five tuple notation* begitu pula dengan diagram transisi dan tabel transisinya. Berikut adalah

contoh diagram transisi dan tabel transisi dari NFA yang menerima *string* berakhiran "01".



Gambar 2.4 Diagram Transisi NFA Penerima *String* Berakhiran 01
(Sumber: Hopcroft, 2007)

Pada diagram tersebut dapat dilihat bahwa di *state* awal masukkan 0 dapat menyebabkan transisi dari q_0 kembali ke q_0 ataupun ke q_1 sehingga tidak dapat dipastikan. Akan tetapi, dapat dipastikan bahwa agar suatu *string* diterima, maka 2 *substring* terakhir dari *string* tersebut haruslah 0 dan 1, terlepas dari sekian kombinasi *substring* sebelumnya.

Tabel 2.2 Tabel Transisi NFA Penerima *String* Berakhiran "01"

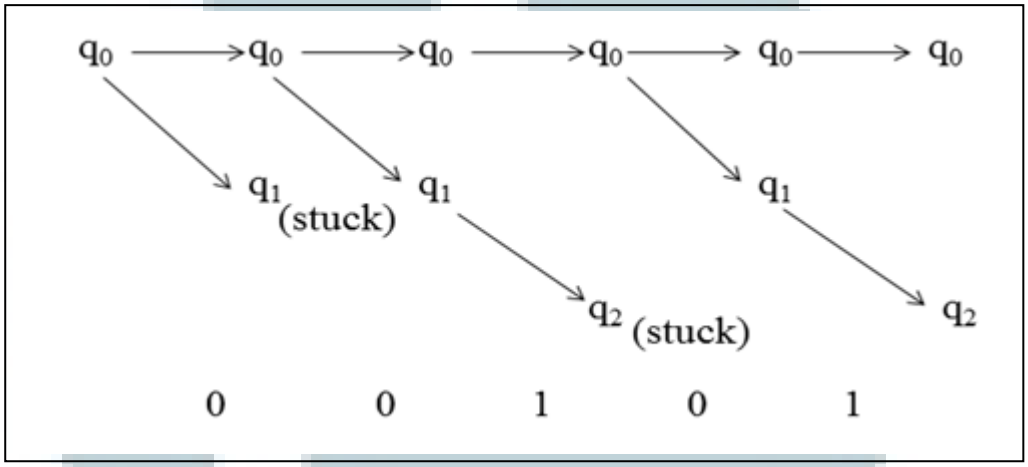
	0	1
→ q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$
*q_2	Ø	Ø

(Sumber: Hopcroft, 2007)

Pembangunan tabel transisi dari diagram transisi NFA sama dengan DFA, di mana baris tabel menggambarkan *state* dan kolom tabel menggambarkan inputnya. Nilai isi dari baris *state* q dan kolom *input* a adalah *state* hasil dari δ(q, a). Pada NFA, karena *state* hasil transisi bisa saja lebih dari satu, maka *state* –

state tersebut ditulis dalam kurung kurawal {}. Tanda Ø menggambarkan tidak ada *state* hasil transisi dari δ(q, a).

Proses pembacaan *string* dalam NFA pada gambar 2.4 (dengan contoh *input string* "00101") digambarkan dalam skema berikut.



Gambar 2.5 Skema Proses Pembacaan *String* "00101" Oleh NFA

Sumber : (Hopcroft,2007)

- Saat '0' pertama dibaca, NFA dapat berpindah ke *state* q_0 ataupun q_1, sehingga perpindahan akan terjadi ke kedua *state* tersebut.
- Lalu '0' kedua akan dibaca. *State* q_0 dapat berpindah kembali ke *state* q_0 ataupun q_1. Di sisi lain, *state* q_1 tidak memiliki transisi untuk '0', sehingga proses terhenti atau "dies"
- *Input* ketiga, yaitu '1' kemudian dibaca. *State* q_0 hanya akan berpindah ke *state* q_0, sedangkan *state* q_1 hanya akan berpindah ke *state* q_2. Pada proses ini (setelah pembacaan "001"), NFA akan berada pada *state* q_0 dan q_2. Karena *state* q_2 adalah *state* akhir, maka NFA akan menerima *string* "001".

- Tetapi proses pembacaan *string* tetap berjalan karena *input*-nya belum selesai dibaca. *Input* keempat, yaitu '0' menyebabkan *state* q_2 "*dies"*, sedangkan *state* q_0 berpindah ke *state* q_0 dan q_1.
- Input terakhir, yaitu '1' menyebabkan perpindahan dari q_0 ke q_0 dan q_1 ke q_2. Karena *state* q_2 merupakan *state* akhir dan input "00101" telah selesai dibaca, maka dapat disimpulkan *string* "00101" diterima.

2.1.5 Penerapan Finite Automata

Beberapa penerapan *Finite Automata*, khususnya dalam bidang *software* antara lain adalah sebagai berikut (Hopcroft, 2007).

1. *Software* untuk merancang dan melakukan pengecekan terhadap perilaku (*behavior*) sirkuit digital.

2. *Software compiler* yang berfungsi sebagai "*lexical analyzer*", yang memisahkan teks input ke dalam unit – unit logikal, seperti *identifiers*, kata kunci/*keywords*, dan tanda baca.

3. *Software* untuk melakukan *scanning* terhadap teks dalam jumlah banyak (misalnya halaman *Web* untuk mencari kata, frasa, ataupun kumpulan kata dengan pola – pola tertentu).

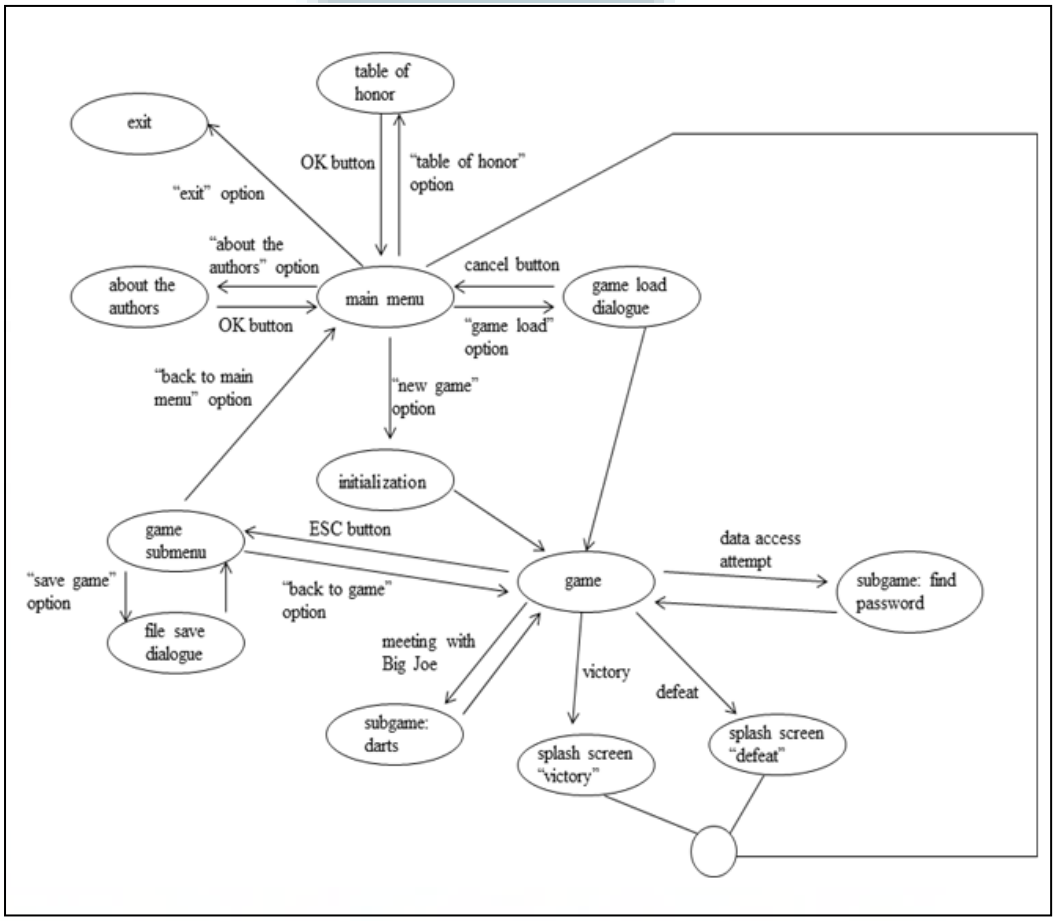
4. *Software* untuk melakukan verifikasi terhadap segala jenis sistem yang memiliki sejumlah *state* berbeda, seperti protokol komunikasi atau protokol keamanan pertukaran data.

Terdapat dua notasi penting yang juga memiliki peranan penting terhadap penerapan *Finite Automata*.

1. *Grammars*, yang merupakan pemodelan penting dalam merancang *software* yang memroses data yang memiliki struktur rekursif, misalnya ”*parser*” yang merupakan komponen *compiler* yang menangani struktur pemrograman yang bersifat rekursif, misalnya ekspresi aritmatik, kondisional, dan sebagainya. Sebagai contohnya adalah aturan E => E + E, dimana aturan tersebut dapat dibentuk dengan mengambil dua ekspresi apapun dan menghubungannya dengan tanda “+”.
2. *Regular Expressions* yang juga mendefinisikan struktur data, terutama teks/*string*. Contoh *Regular Expression* adalah UNIX *Style Regular Expression* dalam bentuk ‘[A-Z][a-z]*[ ][A-Z][A-Z]’ yang merepresentasikan suatu kata yang diawali huruf kapital dengan diikuti dengan spasi dan dua huruf kapital, misalnya nama suatu kota diikuti dengan negara (contoh: Ithaca NY).

Finite Automata dalam bidang pemrograman, juga dapat digunakan sebagai suatu pemodelan dalam struktur pemrograman. Setiap *state* tidak hanya berfungsi sebagai penanda keadaan sistem sekarang, tetapi juga berfungsi untuk menjalankan sejumlah operasi tertentu sesuai dengan *state* yang bersangkutan. Contoh penerapan *Finite Automata* dalam hal ini adalah untuk memodelkan suatu permainan *game* yang paling tidak memiliki beberapa *state* dasar, seperti memulai permainan (*start*), me*load* permainan (*load game*), menyimpan permainan (*save game*), dan *state* yang menandai akhir permainan (baik kemenangan/*victory*

ataupun kekalahan/*defeat*). Berikut adalah gambaran penerapan *Finite Automata* dalam memodelkan suatu permainan. (Mozgovoy, 2010).



Gambar 2.6 Deskripsi Game Menggunakan Finite Automata

(Sumber: Mozgovoy, 2010)

Gambar tersebut memperlihatkan struktur suatu permainan dalam bentuk *Finite Automata*, dimana simbol *input* direpresentasikan dalam tombol – tombol tertentu (OK atau *Cancel*) dan pilihan – pilihan/*option*. Setiap tombol dan pilihan akan membawa permainan dari suatu *state* ke *state* lain (terjadinya transisi), dimana dalam *state* hasil transisi tersebut akan dijalankan sejumlah operasi

tertentu, misalnya menampilkan informasi mengenai pembuat *game* pada *state "about the authors*", menggambar *sprite*, memroses masukkan dari *keyboard*, menampilkan pilihan utama/*main menu* dan menunggu pilihan dari pemain pada *state* "*main menu*", dan sebagainya. (Mozgovoy, 2010).

2.2 Jenis-jenis kalimat bahasa Inggris (*tenses*)

Ada tiga kelompok *tenses* yaitu *present tense*, *past tense* dan *future tense* setiap kelompok masing-masing memiliki empat *tenses*. *Tenses* sendiri memiliki pengertian berupa bentuk kalimat dalam bahasa Inggris yang dibuat berdasarkan peristiwa dan waktu berlangsungnya peristiwa yang diceritakan dalam suatu kalimat. Bahasa Indonesia tidak memiliki *tenses* (perubahan waktu) seperti dalam bahasa Inggris. Baik untuk kemarin, sekarang maupun besok, bentuk kata kerja yang digunakan dalam bahasa Indonesia tidak pernah berubah. (Hakim, 2002)

Berdasarkan pengertian tersebut, maka *english tenses* yang berjumlah 16 macam bisa dibagi menjadi dua bagian besar, yaitu bentuk waktu dan bentuk peristiwa.

1. Mengenal bentuk waktu

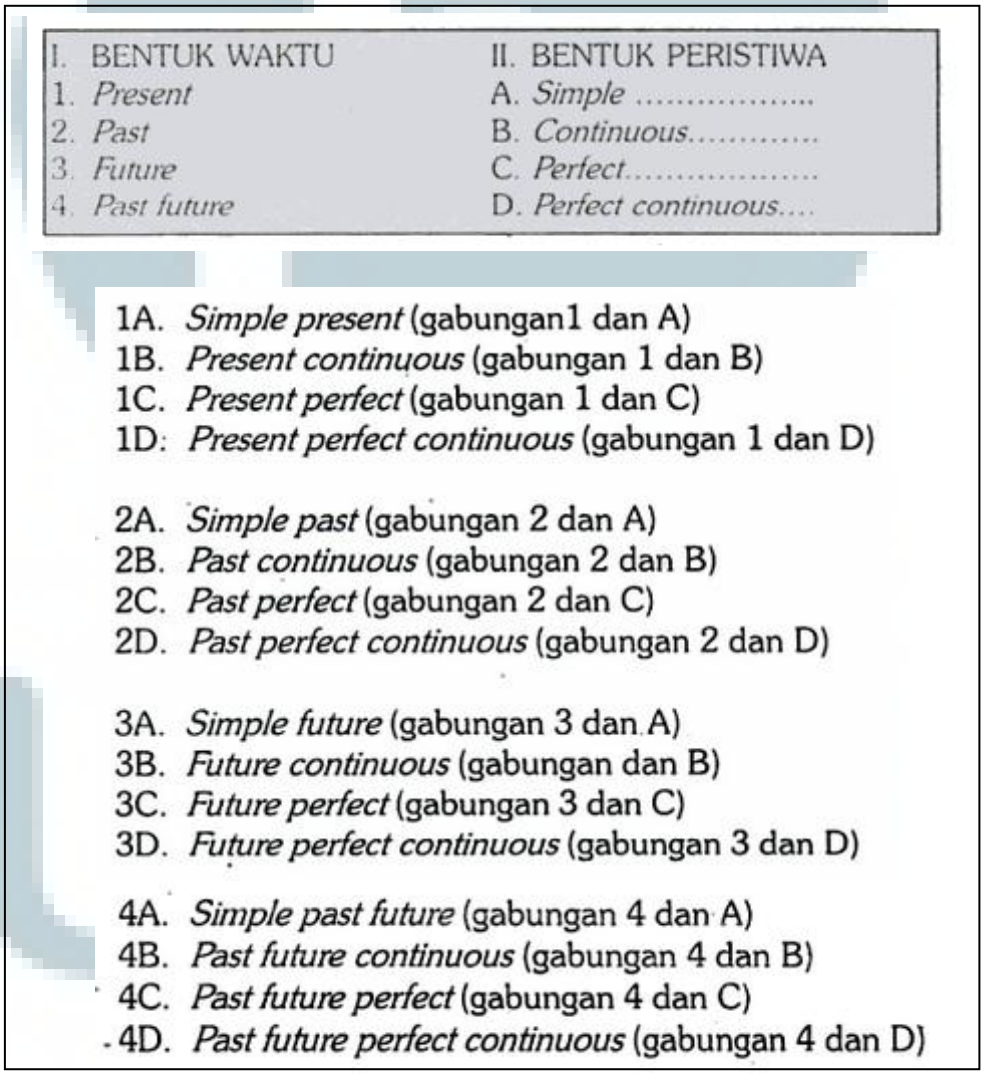
Bentuk waktu yang dapat ditemukan berdasarkan macam-macam waktu, yaitu :

- *present* (sekarang),
- *past* (lampau),
- *future* (yang akan datang),
- *past future* (gabungan masa lalu dan yang akan datang)

2. Mengenal bentuk peristiwa

Bentuk peristiwa bisa dilihat dari proses berlangsungnya peristiwa yang diceritakan dalam suatu kalimat, misalnya :

- *simple* (sederhana atau biasa dilakukan / terjadi berulang-ulang),
- *continous* (sedang dilakukan / sedang terjadi),
- *perfect* (sudah dilakukan / sudah terjadi)
- *perfect continous* (sudah dilakukan dan masih terjadi)



Gambar 2.7 Gambar Diagram *English Tenses*
(Hakim, 2002)

Untuk mendapatkan 16 nama *English tenses* dapat dilakukan dengan menggabungkan bentuk waktu dan bentuk peristiwa dalam diagram khusus seperti yang dilampirkan di atas. (Hakim, 2002)

2.2.1 Simple Present Tense

Simple present merupakan bentuk kalimat sederhana yang peristiwanya menunjukkan kebiasaan atau dilakukan secara berulang-ulang yang berhubungan dengan waku sekarang.

2.2.2 Present Continous Tense

Present Continous adalah bentuk kalimat yang peristiwanya sedang berlangsung pada waktu sekarang.

2.2.3 Present Perfect Tense

Present Perfect adalah bentuk kalimat yang peristiwanya sudah selesai dilakukan pada waktu sekarang (sudah selesai).

2.2.4 Present Perfect Continous Tense

Present Perfect Continous adalah adalah bentuk kalimat yang peristiwanya telah selesai di masa lampau atau peristiwanya telah dimulai dimasa lalu dan terus berlanjut sampai sekarang.

2.2.5 Simple Past Tense

Simple Past merupakan bentuk kalimat sederhana yang peristiwanya terjadi di masa lampau.

2.2.6 Past Continous Tense

Past Continous adalah bentuk kalimat yang peristiwanya sedang berlangsung pada waktu tertentu di masa lampau.

2.2.7 *Past Perfect Tense*

Past Perfect adalah bentuk kalimat yang peristiwanya sudah selesai dilakukan pada waktu lampau (sudah selesai di masa lampau) sebelum peristiwa lainnya terjadi.

2.2.8 *Past Perfect Continous Tense*

Past Perfect Continous adalah bentuk kalimat yang peristiwanya sudah terjadi dan sedang berlangsung pada waktu tertentu di masa lampau.

2.2.9 *Simple Future Tense*

Simple Future merupakan bentuk kalimat sederhana yang peristiwanya akan terjadi di masa depan, baik secara spontan maupun terencana.

2.2.10 *Future Continous Tense*

Future Continous adalah bentuk kalimat yang peristiwanya akan berlangsung pada waktu tertentu di masa depan.

2.2.11 *Future Perfect Tense*

Future Perfect adalah bentuk kalimat yang peristiwanya akan sudah selesai pada waktu tertentu di masa depan.

2.2.12 *Future Perfect Continous Tense*

Future Perfect Continous adalah adalah bentuk kalimat yang peristiwanya akan sudah terjadi selama waktu tertentu di masa depan.

2.2.13 *Simple Past Future Tense*

Simple Past Future merupakan bentuk kalimat sederhana yang meramalkan peristiwa akan terjadi di masa depan, pada saat berada di masa lalu.

2.2.14 Past Future Continous Tense

Past Future Continous adalah bentuk kalimat yang meramalkan peristiwa akan sedang terjadi di masa depan, pada saat berada di masa lalu.

2.2.15 Past Future Perfect Tense

Past Future Perfect adalah bentuk kalimat yang membicarakan suatu peristiwa akan telah dilakukan di masa lalu.

2.2.16 Past Future Perfect Continous Tense

Past Future Perfect Continous adalah adalah bentuk kalimat yang peristiwanya akan sudah terjadi selama waktu tertentu di masa lalu dan dapat diketahui hasilnya pada saat ini.

2.3 Tatanan Kata dan Contoh Kalimat Bahasa Inggris

Berikut adalah rumus dan contoh sederhana dari ke enam belas jenis *tenses: (*Amr, 2014)

2.3.1 *Simple Present Tense* (Waktu Sekarang Sederhana)

Rumus :

+ } S + V1

- } S + Do/does + not + V1

? } Do/does + S + V1 ?

Contoh :

(+) Sisca reads book everyday.

(-) Sisca does not read book everyday.

(?) Does Sisca read book everyday ?

2.3.2 *Present Continuous Tense* (Waktu Berlangsung Sekarang)

Rumus :

+ } S + tobe(is/am/are) + V1 + ing

- } S + tobe(is/am/are) + not + V1 + ing

? } tobe(is/am/are) + S + V1 + ing ?

Contoh :

(+) They are playing badminton now.

(–) They are not playing badminton now.

(?) Are they playing badminton now ?

2.3.3 *Present Perfect Tense* (Waktu Sempurna Sekarang)

Rumus :

+ } S + have/has + V3

– } S + have/has + not + V3

? } have/has + S + V3 ?

Contoh :

(+) You have eaten noodle.

(-) She has not been to Rome.

(?) Have you finished ?

2.3.4 *Present Perfect Continuous Tense* (Waktu Berlangsung Sempurna Sekarang)

Rumus :

+ } S + have/has + been + Ving

- } S + have/has + not + been + Ving

? } Have/has + S + been + Ving ?

Contoh :

(+) She has been going to Malang since evening.

(-) She hasn't been going to Malang since evening.

(?) Has she been going to Malang ?

2.3.5 *Simple Past Tense* (Waktu Lampau Sederhana)

Rumus :

+ } S + V2

- } S + Did + not + V1

? } Did + S + V1 ?

Contoh :

(+) I saw a good film last night.

(-) I didn't see a good film last night.

(?) Did I see a good film last night ?

2.3.6 *Past Continuous Tense* (Waktu Berlangsung Lampau)

Rumus :

+ } S + tobe (was/were) + Ving

- } S + tobe (was/were) + not + Ving

? } tobe (was/were) + S + Ving ?

Contoh :

(+) They were talking about sport when I met him.

(-) He wasn't watching television all afternoon last week.

(?) Was he watching television all afternoon last week ?

2.3.7 *Past Perfect Tense* (Waktu Sempurna Lampau)

Rumus :

+ } S + had + V3

- } S + had + not + V3

? } had + S + V3 ?

Contoh :

(+) The ship had left before I arrived.

(-) I hadn't painted my motorcycle.

(?) Had I painted my motor cycle ?

2.3.8 *Past Perfect Continuous Tense* (Waktu Berlangsung Sempurna Lampau)

Rumus :

+ } S + had + been + Ving

- } S + had + not + been + Ving

? } had + S + been + Ving ?

Contoh :

(+) Your father had been playing badminton.

(-) They hadn't been living there for two month.

(?) Had they been living there for two month ?

2.3.9 *Simple Future Tense* (Waktu Akan Datang Sederhana)

Rumus :

+ } S + will/shall + V1

- } S + will/shall + not + V1

? } will/shall + S + V1 ?

Contoh :

(+) He will meet girl friend by seven o'clock.

(-) We shall not at Nederland the day after tomorrow.

(?) Will he go to America next month ?

2.3.10 *Future Continuous* Tense (Waktu Berlangsung Akan Datang)

Rumus :

+ } S + will/shall + be + Ving

- } S + will/shall + not + be + Ving

? } will/shall + S + be + Ving ?

Contoh *:*

(+) I will be studying tomorrow night.

(-) I will not be writing a comic.

(?) Will I be writing a comic ?

2.3.11 *Future Perfect Tense* (Waktu Sempurna Akan Datang)

Rumus :

+ } S + will/shall + have + V3

- } S + will/shall + not + have + V3

? } will/shall + S + have + V3 ?

Contoh :

(+) You will have forgotten me.

(-) She will not have gone to school.

(?) Will you have arrived ?

2.3.12 *Future Perfect Continuous Tense* (Waktu Berlangsung Sempurna Akan Datang)

Rumus :

+ } S + will/shall + have + been + Ving

- } S + will/shall + not + have + been + Ving

? } will/shall + S + have + been + Ving ?

Contoh :

(+) He will have been listening music.

(-) I will haven't been reading a news paper.

(?) Will I have been reading a news paper ?

2.3.13 *Past Future Tense* (Waktu Akan Datang di Waktu Lampau)

Rumus :

+ } S + would + V1

- } S + would + not + V1

? } would + S + V1 ?

Contoh :

(+) They would buy a home the previous day.

(-) They wouldn't buy a home the previous day.

(?) Would he come ?

2.3.14 *Past Future Continuous Tense* (Waktu Akan Sedang Terjadi di waktu Lampau)

Rumus :

+ } S + would + be + Ving

- } S + would + not + be + Ving

? } Would + S + be + Ving ?

Contoh :

(+) They would be sleeping at 10 o'clock tomorrow.

(-) They would not be sleeping at 10 o'clock tomorrow.

(?) Would they be swimming at this time the following day ?

2.3.15 *Past Future Perfect Tense* (Waktu Akan Sudah Selesai di Waktu Lampau)

Rumus :

+ } S + would + have + V3

- } S + would + not + have + V3

? } Would + S + have + V3 ?

Contoh :

(+) Nonok would have studied math by the end of this week.

(-) He wouldn't have gone if he had met his darling.

(?) Would he have gone if he had met his darling ?

2.3.16 *Past Future Perfect Continuous Tense* (Waktu Yang Sudah Sedang Berlangsung pada Waktu Lampau)

Rumus :

+ } S + would + have + been + Ving

- } S + would + not + have + been + Ving

? } Would + S + have + been + Ving ?

Contoh :

(+) Rianawati would have been speaking English for two years.

(-) Rianawati wouldn't have been speaking English for two years.

(?) Would Rianawati have been walking here for seventeen years ?

2.4 Word Error Rate (WER)

Word Error Rate (WER) adalah sebuah perhitungan umum untuk menghitung performa dari sebuah *speech recognition system* atau *machine translation system*. Pengukuran WER dinyatakan dalam rumus berikut.

$$WER = \frac{S + D + I}{N}$$

atau

$$WER = \frac{S + D + I}{S + D + C}$$

Gambar 2.8 Rumus WER

di mana :

- S adalah jumlah dari *substitutions*,
- D adalah jumlah dari *deletions*,
- I adalah jumlah dari *insertions*,
- C adalah jumlah dari *corrects*,
- N adalah jumlah dari kata-kata yang direferensikan (diujicobakan) (N=S+D+C)

Perhitungan tersebut dilakukan dengan *first aligning* urutan kata yang dikenali dengan urutan kata yang direferensikan dengan menggunakan *dynamic string alignment.* Pemeriksaan dari perhitungan ini dapat dilihat melalui sebuah teori yang disebut *power law* yang menyebutkan korelasi antara hal-hal yang tidak dapat dimengerti dan membingungkan dengan *word error rate.* (Dietrich, 2002)

2.5 Cronbach's Alpha

Cronbach's Alpha adalah perhitungan yang biasanya untuk memperkirakan kepastian akan sesuatu hal, misalnya perhitungan *internal consistency*. Perhitungan ini dikembangkan oleh Kuder & Richardson (1937) untuk perhitungan *dichotomously scored data* (0 atau 1) dan kemudian digeneralisasi oleh Cronbach untuk perhitungan lainnya. perhitungan *Cronbach's basic equation* untuk *alpha* ditunjukkan dengan rumus dan skor sebagai berikut.

$$\alpha = \frac{n}{n-1}\left(1 - \frac{\Sigma Vi}{Vtest}\right)$$

Gambar 2.9 Rumus Cronbach alpha

Tabel 2.3 Tabel indeks skor Cronbach alpha

Cronbach's alpha	Internal consistency
α ≥ 0.9	Excellent (High-Stakes testing)
0.7 ≤ α < 0.9	Good (Low-Stakes testing)
0.6 ≤ α < 0.7	Acceptable
0.5 ≤ α < 0.6	Poor
α < 0.5	Unacceptable

keterangan:

n = jumlah pertanyaan

Vi = jumlah varian dari skor untuk masing-masing pertanyaan

Vtest = jumlah varian dari keseluruhan skor (bukan dalam %) untuk keseluruhan tes

Skor akhir dengan nilai *alpha* yang tinggi adalah baik, hal disebabkan oleh varian yang tinggi. Varian yang tinggi menunjukkan persebaran dari skor secara meluas, hal ini berarti orang-orang yang diajukan kuesioner dapat mengerti pertanyaan yang diajukan dengan mudah. Apabila sebuah tes memiliki varian yang rendah maka skor secara keseluruhan mendekati kebenaran, kecuali yang ditanyakan tidak mengerti, maka tes yang dilakukan menjadi tidak berarti.

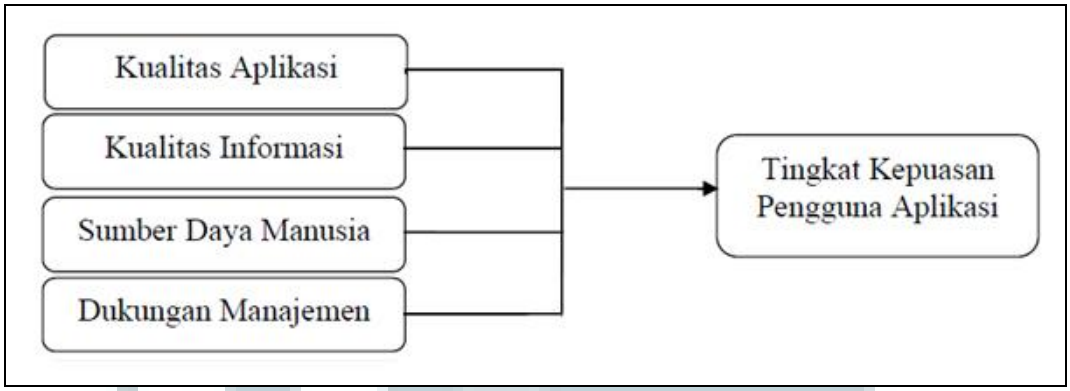
Dalam mengintepretasikan Cronbach alpha, ada lima konsep yang penting untuk diketahui :

1. Cronbach alpha menghasilkan perkiraan *internal consistency* dari hasil tes, di mana *alpha* tidak mengindikasikan kestabilan atau konsistensi hasil uji coba dari waktu ke waktu.
2. Cronbach alpha hanya cocok diterapkan pada uji coba yang bersifat *norm-referenced* dan *norm-referenced decisions*, tidak berlaku untuk *criterion-referenced tests* dan *criterion-referenced decisions.*
3. Setiap faktor lain yang memengaruhi konstanta, tes yang mempunyai distribusi skor yang normal memiliki nilai perkiraan Cronbach alpha yang tinggi dengan distribusi positif maupun negatif, sehingga *alpha* harus diintepretasikan dengan ditribusi yang dilibatkan di dalamnya.
4. Setiap faktor lain yang memengaruhi konstanta, Cronbach alpha akan menjadi lebih tinggi ketika tes yang dilakukan berjangka waktu lebih lama, sehingga *alpha* harus diintepretasikan dengan ditribusi yang dilibatkan di dalamnya.

5. *Standard Error Measurement* (SEM) adalah tambahan dari perkiraan *reliability statistic* yang dihitung dari perhitungan perkiraan yang lebih berguna dalam proses menghitung ketika digunakan dalam membuat keputusan dengan skor dari tes.

2.6 Tingkat Kepuasan Pengguna Aplikasi

Sebelum dilakukan penyebaran kuesioner penelitian, tentunya dibutuhkan sumber yang konkrit dalam penyusunan pertanyaan kuesioner yang akan diajukan. Kuesioner yang akan disebarkan tersebut tentunya sehubungan dengan aplikasi yang dibangun penulis dan bertujuan untuk mengukur tingkat kepuasan pengguna aplikasi. Kepuasan menurut Kotler (2003) merupakan fungsi dari persepsi atau kesan atas kinerja atau hasil suatu produk dan harapan. Jika kinerja atau hasil suatu produk berada di bawah harapan, maka pengguna akan merasa tidak puas. Jika kinerja atau hasil suatu produk memenuhi harapan, maka pengguna akan merasa puas. Jika kinerja atau hasil suatu produk melebihi harapan, maka pelanggan akan sangat puas atau senang. Tingkat kepuasan pengguna aplikasi mengacu pada sejauh mana pengguna aplikasi merasakan aplikasi yang digunakan mampu memenuhi harapan mereka (Al-Adaileh, 2009). Tingkat kepuasan pengguna aplikasi ini diukur dengan menggunakan indikator antara lain pemenuhan harapan dan kebutuhan pengguna, pencapaian tujuan pekerjaan, peningkatan kinerja, serta persepsi rasa puas menggunakan aplikasi. Berlandaskan pada hal-hal tersebut, penulis melampirkan pembagian aspek-aspek yang dimaksud sebagai berikut:



Gambar 2.10 Skema Tingkat Kepuasan Pengguna Aplikasi

2.6.1 Pengaruh Kualitas Aplikasi terhadap Tingkat Kepuasan Pengguna Aplikasi

Faktor kualitas aplikasi merupakan faktor utama dalam pengukuran tingkat kepuasan pengguna aplikasi. Hal ini dikarenakan keseluruhan siklus aplikasi diproses dengan menggunakan aplikasi hingga menghasilkan *output*. Jika aplikasi yang digunakan berkualitas maka akan mempengaruhi tingkat kepuasan penggunanya. Sesuai dengan penjelasan Kotler (2003), pengguna akan merasa puas apabila kinerja dari produk yang digunakan dapat memenuhi atau melebihi harapan pengguna.

2.6.2 Pengaruh Kualitas Informasi terhadap Tingkat Kepuasan Pengguna Aplikasi

Faktor kualitas informasi digunakan untuk mengukur kualitas *output* dari aplikasi yang digunakan dalam pengaruhnya terhadap tingkat kepuasan pengguna. Informasi yang berkualitas akan berguna bagi pihak-pihak yang berkepentingan dalam hal pengambilan keputusan. Mengingat dampaknya tersebut maka kualitas informasi akan berpengaruh terhadap tingkat kepuasan pengguna. Sesuai dengan

penjelasan Kotler (2003), pengguna akan merasa puas apabila hasil dari produk yang digunakan dapat memenuhi atau melebihi harapan pengguna.

2.6.3 Pengaruh Sumber Daya Manusia terhadap Tingkat Kepuasan Pengguna Aplikasi

Baroudi dan Orlikowski (1988) mendefinisikan kompetensi pengguna sebagai penilaian dari responden atas kualitas dari pelatihan yang disediakan. Untuk menunjang kesuksesan implementasi suatu sistem, pengguna sistem harus memiliki kemampuan teknis yang diperlukan dalam pengoperasian sistem tersebut. Menurut Al-Adaileh (2009) kemampuan teknis pengguna dapat diukur dengan indikator antara lain: tingkat pengetahuan dan penguasaan aplikasi, latar belakang pendidikan, serta pengalaman kerja pengguna.

2.6.4 Pengaruh Dukungan Manajemen terhadap Tingkat Kepuasan Pengguna Aplikasi

Tersedianya fasilitas sarana dan prasarana untuk menunjang kinerja pengguna aplikasi menjadi salah satu indikator nyata dari implementasi dukungan manajemen. Adam Mahmood (2000) dalam penelitiannya menggunakan variabel dukungan manajemen sebagai pengukur pengaruh dari faktor organisasi. Dukungan manajemen tersebut diukur dengan sikap positif dari pengguna, kesempatan pelatihan bagi pengguna dan persepsi sikap manajemen.