2. TEORI PENUNJANG

2.1. *Ringtone*

Ringtone merupakan salah satu fasilitas yang terdapat pada *handphone*. Fungsi daripada *ringtone* adalah untuk memberitahukan pemilik *handphone* jika ada panggilan yang masuk. *Ringtone* dibagi menjadi dua macam yaitu *monophonic* dan *polyphonic*. *Monophonic* mempunyai arti bahwa suara alat musik yang dapat dibunyikan secara bersamaan hanya ada satu jenis suara alat musik saja, sedangkan *polyphonic* lebih dikenal dengan enambelas *polyphonic* mempunyai arti bahwa suara alat musik yang dapat dibunyikan secara bersamaan jenisnya ada enambelas macam alat musik, antara lain alat musik biola, bass, terompet dan sebagainya sehingga suara yang dihasilkan lebih indah dan merdu menyerupai suara orkestra.

Pada model *handphone* tertentu antara lain Nokia 3210, Siemens C35 dan Siemens M35 mempunyai jumlah nada yang terbatas yaitu maksimum sebanyak limapuluh buah nada yang berarti suara atau *ringtone* yang dimasukkan ke dalam model *handphone* tersebut harus memiliki jumlah nada sebanyak limapuluh buah dan tidak boleh lebih.

Setiap vendor mempunyai *format* penulisan *ringtone* yang berbeda, antara lain :

1. Siemens :

   [Nada] [(Tanda Kromatis)] [(Oktav)] [(Ketukan Nada)], dimana :

Nada	: c, d, e, f, g, a, h, c', p
Tanda Kromatis	: # (kres) disimbolkan dengan 'is'
Oktav	: 1, 2, 3, 4
Ketukan Nada	: 3/1, 2/1, 1/1, 1/2, 1/4, 1/8, 1/16 dalam program ketukan 3/1 & 2/1 tidak digunakan karena pada dasarnya *ringtone* yang ada hanya menggunakan ketukan mulai dari 1/1 sampai dengan 1/16

2. Nokia :

   [(Ketukan Nada)] [(Spesial Ketukan Nada)] [(Tanda Kromatis)] [Nada] [(Oktav)], dimana :

   Ketukan Nada : 1, 2, 4, 8, 16, 32

   Spesial Ketukan Nada : .

   Tanda Kromatis : # (kres)

   Nada : c, d, e, f, g, a, b, c', -

   Oktav : 1, 2, 3

3. Ericsson:

   [(Oktav)] [(Tanda Kromatis)] [Nada & (Ketukan Nada)], dimana :

   Oktav : Terdiri dari 2 oktav (oktav 2 disimbolkan tanpa menggunakan tanda "+", oktav 3 disimbolkan dengan menggunakan tanda "+")

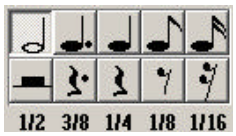
   Tanda Kromatis : b (mol), # (kres) dalam program hanya menggunakan tanda kromatis # (kres)

   Nada & Ketukan Nada : c, d, e, f, g, a, b, c', p

   Ketukan nada disimbolkan menggunakan huruf besar dan huruf kecil, contoh : C mempunyai ketukan 1/8, c mempunyai ketukan 1/16

4. Samsung :

   Sama dengan *format* nokia (kompatibel dengan nokia), hanya saja di tampilkan dalam *format* not balok, dengan simbol not balok seperti pada gambar berikut :



Gambar 2.1. Nada dan Ketukan Nada Pada Samsung

Gambar 2.2. Tanda Kromatis Pada Samsung

2.2. Teori Dasar Musik

Setiap notasi dalam musik memiliki aturan peletakan, *format* dan nilai sendiri-sendiri, baik dalam not balok maupun not angka. Ada yang bernilai 1/2, 1/4, 1/8, 1/16 dan 1/32.

2.2.1. Not Balok

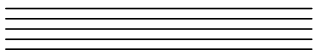
Untuk menuliskan notasi musik dalam not balok, ada dua hal penting yang harus diperhatikan, pertama adalah lima garis lurus berjajar mendatar secara teratur. Kedua adalah bulatan-bulatan bertangkai.

2.2.1.1. Paranada

Yang pertama, adalah lima garis lurus yang berjajar mendatar dan berjarak sama, yang disebut paranada. Kegunaan paranada ini ialah untuk menempatkan not-not, sesuai dengan sifat-sifat nada yang dilambangkannya. Not yang dalam paranada ditulis rendah melambangkan nada yang rendah pula.

Ruang diantara garis-garis paranada dipergunakan juga untuk menuliskan not. Ruang diantara garis-garis itu dinamakan spasi. Not-not tersebut dituliskan, baik pada garis maupun spasi. Not yang ditulis pada garis disebut not garis. Not yang ditulis pada spasi disebut not spasi.

Dengan demikian, sebuah paranada dapat digunakan untuk menulis sedikitnya lima tingkat not garis dan empat tingkat not spasi.



Gambar 2.3. Paranada

Dalam keterangan selanjutnya mungkin akan disebutkan garis kedua, atau spasi ketiga. Nomor ini pada paranada selalu dimulai dari bawah.

2.2.1.2 Bentuk Not

Yang kedua, adalah bulatan-bulatan bertangkai. Dalam notasi balok, bulatan tersebutlah yang disebut not. Untuk penulisan not ke dalam paranada ada beberapa aturan :

1. Jika kepala not terletak diatas garis ketiga, tangkai not harus menuju ke bawah.
2. Jika kepala not terletak dibawah garis ketiga, tangkai not harus menuju ke atas.
3. Jika kepala not terletak pada garis ketiga, tangkai not dapat menuju ke atas atau ke bawah. Dalam program tugas akhir ini, tangkai not dibuat menuju kebawah.

Contoh bentuk dari not-not tersebut sebagai berikut :

Not 1/4 =  Not 1/8 = 

Not 1/16 =  Not 1/32 = 

Untuk nilai notasi yang naik setengah atau turun setengah, bisa digunakan tanda kres (#) atau mol (b). Pada not balok, bila suatu not diberi tanda kres atau mol, maka semua not yang dibelakangnya yang berada pada garis atau spasi yang sama dan berada dalam satu ruas birama akan ikut naik sesuai not yang diberi tanda tersebut. Untuk mengembalikannya perlu diberikan tanda normal.

Tanda titik atau dot (**.**) pada belakang not berarti menambah nilai 1/2 dari not yang di depannya. Contoh :

Not 3/8 = 

2.2.1.3 Kunci G dan Nama Not

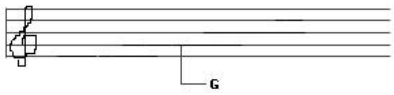
Jumlah nada yang dapat didengar, sebenarnya sangat banyak. Namun demikian, musik hanya mengambil sebagian saja untuk diolah secara indah. Sebagai contoh, sebuah nada mempunyai frekuensi 440 Hz yang dipakai dalam musik. Nada-nada lain dengan frekuensi 441, 442, 443 dan seterusnya tidak semua dipakai. Baru nada dengan frekuensi 466 Hz yang dipakai sebagai nada terdekatnya.

Contoh lain dilihat pada piano. Nada terendah pada piano kurang lebih 27 Hz, sedangkan nada tertinggi sekitar 4100 Hz. Tapi dari jarak sejauh itu, kurang lebih hanya 88 nada saja yang diwujudkan dalam bilah-bilah nada piano dan nada-nada inilah yang dapat dituliskan dalam notasi not balok.

Not-not balok diberi nama dengan huruf abjad. Yang dipakai hanya tujuh huruf yaitu A sampai dengan G. Diatas not G atau dibawah not A, tujuh nama

pokok not tersebut diulangi lagi. Yang menunjukkan tingkat nada adalah paranada. Not yang dalam paranada tertulis di atas, berarti lebih tinggi tingkatannya dari not yang tertulis di bawah. Garis-garis dan spasi paranada ini menentukan nama pokok not. Not yang terletak pada garis G akan bernama not G. Not yang terletak pada spasi A akan bernama not A.

Untuk mengetahui nama garis-garis dan spasi-spasi paranada, dipakai kunci. Dalam bagian ini akan dikenalkan kunci G. Kunci ini selalu dituliskan pada awal paranada. Dengan adanya kunci G ini, maka letak not G dapat diketahui, yaitu pada garis kedua paranada. Dengan demikian semua not yang terletak pada garis kedua bernama not G. Berikut ini adalah gambar dari kunci G, yang harus diletakkan pada setiap awal dari paranada.



Gambar 2.4. Kunci G

Dibawah garis kedua adalah spasi kesatu. Dibawah not G adalah not F. Jadi spasi kesatu adalah tempat untuk not F, demikian seterusnya.

2.2.1.4. Nada Dasar

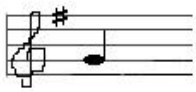
Dalam kehidupan sehari-hari, sering dijumpai orang menyanyikan nada-nada musik dengan pengucapan do, re, mi, fa, sol, la dan si. Dalam not angka, pengucapan ini ditulis dengan angka-angka yaitu 1, 2, 3, 4, 5, 6 dan 7. Susunan semacam ini dalam musik disebut tangga nada. Cara pengucapan tadi disebut pengucapan system doremisasi atau solmisasi.

Nada pertama dari tangga nada diatas adalah nada do, atau not angka 1. Nada atau not tersebut selanjutnya disebut nada dasar. Jelasnya, nada dasar adalah nada yang dipakai sebagai dasar dari sebuah tangga nada. Dalam not balok, untuk mengetahui nada dasar diperlukan tanda mula. Salah satu tanda mula adalah tanda mula kres (#). Nada dasar ditentukan oleh jumlah kres. Jika jumlah kresnya berbeda, berbeda pula nada dasarnya.

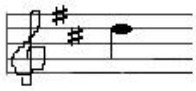
Susunan dan letak kres dalam paranada sudah ditentukan. Jadi tiap kres dari tanda mula terletak pada garis atau spasi yang tetap.

- Tanda mula kres satu, bernada dasar G
- Tanda mula kres dua, bernada dasar D
- Tanda mula kres tiga, bernada dasar A
- Tanda mula kres empat, bernada dasar E

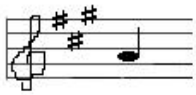
Sebagai contoh, jika pada paranada terdapat tanda mula kres berjumlah satu, maka berarti not do terletak pada garis kedua, yaitu not G. Spasi yang kedua diatas garis tersebut untuk not re, atau not A, garis ketiga untuk not mi atau not B dan seterusnya. Jika dalam penulisan lagu atau *ringtone* tidak tertulis tanda mulanya, berarti lagu atau *ringtone* tersebut memakai tanda mula natural, atau dengan nada dasar C. Dalam program yang dibuat dalam tugas akhir ini hanya digunakan tanda mula natural, atau dengan nada dasar C.



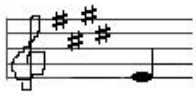
Gambar 2.5. Nada Dasar G (G = Do)



Gambar 2.6. Nada Dasar D (D = Do)



Gambar 2.7. Nada Dasar A (A = Do)



Gambar 2.8. Nada Dasar E (E = Do)

Selain tanda mula kres, ada juga tanda mula mol (b), untuk tanda mol, nilainya :

- Tanda mula satu mol, bernada dasar F
- Tanda mula dua mol, bernada dasar Bes

- Tanda mula tiga mol, bernada dasar Es
- Tanda mula empat mol, bernada dasar As

2.3. Teori Bahasa dan Otomata

Teori bahasa dan otomata merupakan model dan gagasan mendasar mengenai komputasi. Secara teoritis ilmu komputer diawali dari sejumlah berbeda disiplin ilmu : ahli biologi mempelajari *neural network*, insinyur elektro mengembangkan *switching* sebagai *tool* untuk mendesain *hardware*, matematikawan bekerja mendasarkan logika, dan ahli bahasa menyelidiki tata bahasa untuk *natural language*.

Finite state automata dan ekspresi regular awalnya dikembangkan berdasar pemikiran *neural network* dan *switching circuit. Finite state automata* dipakai dalam *text editor*, *pattern matching*, sejumlah pemrosesan teks, dan program *file searching*.

2.3.1. Konsep Bahasa dan Otomata

Sebuah simbol adalah suatu entitas abstrak yang tidak didefinisikan secara formal, seperti halnya ‘titik’ dan ‘garis’ pada geometri tidak didefinisikan. Huruf dan digit adalah contoh dari simbol yang sering dipakai. Sebuah *string* (kata/untai) adalah suatu deretan berhingga dari simbol-simbol. Sebagai contoh : ‘a’,’b’,’c’ adalah simbol-simbol, dan ‘abcd’ adalah suatu *string*. Panjang *string* adalah jumlah simbol yang membentuk *string* tersebut, contoh : ‘abad’ panjangnya 4, ‘asdfgh’ panjangnya 6.

Sebuah *string* kosong, biasanya dinyatakan dengan ε, didefinisikan panjangnya = 0, atau $|\varepsilon| = 0$ (simbol untuk ε dapat dinyatakan juga dengan λ). Suatu *alphabet* adalah himpunan berhingga dari simbol-simbol.

Definisi bahasa di dalam kamus adalah suatu sistem yang meliputi pengekspresian gagasan, fakta, konsep, termasuk sekumpulan simbol-simbol dan aturan untuk melakukan manipulasinya. Sebuah bahasa adalah himpunan *string-string* dari simbol-simbol untuk suatu *alphabet*. Sebuah bahasa adalah kumpulan dari *string-string*, sebuah bahasa juga bisa tidak terdiri dari *string-string*, yaitu bahasa kosong, yang dinotasikan seperti himpunan kosong, ∅. Bahasa kosong,

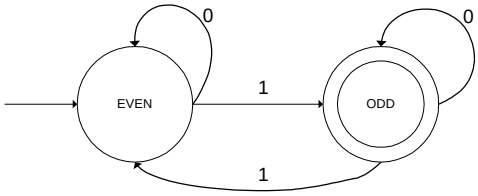
berbeda dengan bahasa yang terdiri dari *string* kosong {ε}. Bahasa bisa juga disebut sebagai rangkaian simbol-simbol yang mempunyai makna.

Otomata adalah suatu bentuk yang memiliki fungsi-fungsi dari komputer digital. Menerima *input*, menghasilkan *output*, bisa memiliki penyimpanan sementara, dan mampu membuat keputusan dalam mentransformasikan *input* ke *output*. Sebuah bahasa formal adalah suatu abstraksi terdiri dari himpunan simbol-simbol dan aturan-aturan yang mana simbol-simbol tersebut bisa dikombinasikan ke dalam entitas yang disebut kalimat.

Otomata merupakan suatu sistem yang terdiri atas sejumlah *state* yang berhingga , dimana *state* menyatakan informasi mengenai *input* yang diberikan, dan dapat pula dianggap sebagai memori mesin. *Input* pada mesin otomata dianggap sebagai bahasa yang harus dikenali oleh mesin. Selanjutnya mesin otomata membuat keputusan yang mengindikasikan apakah *input* itu diterima atau tidak. Sehingga mesin otomata dapat dipakai untuk menghasilkan bahasa yang aturannya ditentukan oleh aturan bahasa tersebut.

2.3.2. *Finite State Automata*

Finite state automata merupakan mesin otomata dari bahasa regular. Suatu *finite state automata* memiliki *state* yang banyaknya berhingga, dan dapat berpindah-pindah dari suatu *state* ke *state* lain. Perubahan *state* ini dinyatakan oleh fungsi transisi. Jenis otomata ini tidak memiliki tempat penyimpanan, sehingga kemampuan ‘mengingatnya’ terbatas.



Gambar 2.9. Mesin Otomata Untuk Memeriksa Pariti Ganjil

Sumber : Teori Bahasa dan Otomata Firrar Utdirartatmo J&J Learning Yogyakarta

Pada *finite state automata*, arti dari bentuk-bentuk seperti yang ada pada gambar 2.9 sebagai berikut :

1. Lingkaran menyatakan *state*/kedudukan
2. Label pada lingkaran adalah nama *state* tersebut
3. Busur menyatakan transisi yaitu perpindahan kedudukan/*state*
4. Label pada busur adalah simbol *input*
5. Lingkaran didahului sebuah busur tanpa label menyatakan *state* awal
6. Lingkaran ganda menyatakan *state* akhir/*final*

Gambar seperti gambar 2.9 biasa disebut sebagai *graph* transisi, diagram transisi atau diagram keadaan (*state*).

State akhir pada umumnya dapat berjumlah lebih dari satu. Istilah *state* akhir (*final state*) tidak berarti komputasi berhenti (*halt*) begitu *state* akhir tercapai. *State* akhir hanya menyatakan kedudukan-kedudukan (*state*) tertentu sebagai kedudukan-kedudukan yang diterima (*accepting state*).

Secara formal *finite state automata* dinyatakan oleh 5 tupel atau M = (Q, Σ, δ, S, F), dimana :

Q = himpunan *state*/kedudukan

Σ = himpunan simbol *input*/masukan/abjad

δ = fungsi transisi

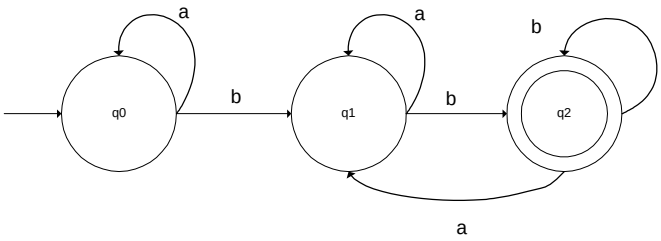
S = *state* awal/kedudukan awal (inisial *state*), $S \in Q$

F = himpunan *state* akhir, $F \subseteq Q$

Finite state automata berdasar pada pendefinisian kemampuan berubah *state-state*-nya bisa dikelompokkan ke dalam deterministik maupun non-deterministik.

2.3.2.1. *Deterministic Finite Automata* (DFA)

Pada Otomata Berhingga Deterministik/*Deterministic Finite Automata* (DFA), dari suatu *state* ada tepat satu *state* berikutnya untuk setiap simbol masukan yang diterima.



Gambar 2.10. Mesin DFA

Sumber : Teori Bahasa dan Otomata Firrar Utdirartatmo J&J Learning Yogyakarta

Konfigurasi *Deterministic Finite Automata* pada gambar 2.10 secara formal dinyatakan sebagai berikut :

Q = {q0, q1, q2}

∑ = {a,b}

A = q0

F = {q2}

Fungsi transisi yang ada sebagai berikut :

δ(q0,a) = q0

δ(q0,b) = q1

δ(q1,a) = q1

δ(q1,b) = q2

δ(q2,a) = q1

δ(q2,b) = q2

Biasanya fungsi transisi ini disajikan dalam sebuah tabel transisi. Tabel transisi tersebut menunjukkan *state-state* berikutnya untuk kombinasi *state-state* dan *input*. Tabel transisi dari fungsi transisi diatas sebagai berikut :

Tabel 2.1. Transisi Mesin DFA

Sumber : Teori Bahasa dan Otomata Firrar Utdirartatmo J&J Learning Yogyakarta

δ	a	B
q0	q0	q1
q1	q1	q2
q2	q1	q2

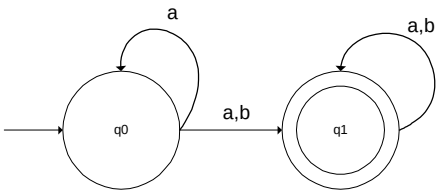
Pada tabel 2.1 terlihat bahwa ada sebuah *state* berikutnya yang unik untuk setiap pasangan *state-input*. Jadi untuk sebuah *state* dan *input* yang berlaku, dapat ditentukan tepat satu *state* berikutnya. Pada *Deterministic Finite Automata*, δ merupakan sebuah fungsi yang harus terdefinisi untuk semua pasangan *state-input* yang ada di dalam Q X Ó. Sehingga apapun *state* saat itu (*current state*) atau *input*-nya, selalu terdapat satu dan hanya satu *state* berikutnya. *State* berikutnya itu sepenuhnya ditentukan oleh informasi yang ada di dalam pasangan *state-input*.

Suatu *string* x dinyatakan diterima bila δ(S,x) berada pada *state* akhir. Biasanya secara formal dikatakan bila M adalah sebuah *finite state automata*, M=(Q, Ó, δ, S, F), menerima bahasa yang disebut L(M), yang merupakan himpunan {x | δ(S,x) di dalam F} (L : bisa dianggap kependekan dari '*language'*).

Dari suatu gambar/diagram transisi dapat dibuat tabel transisinya, begitu pula sebaliknya.

2.3.2.2. *Non-deterministic Finite Automata* (NFA)

Pada *Non-deterministic Finite Automata* (NFA) dari suatu *state* bisa terdapat 0, 1 atau lebih busur keluar (transisi) berlabel simbol *input* yang sama. *Non-deterministic Finite Automata* didefinisikan juga dengan lima (5) tupel M = (Q, Ó, δ, S, F), dengan arti yang serupa pada *Deterministic Finite Automata*. Perbedaan antara *Deterministic Finite Automata* dan *Non-deterministic Finite Automata* terletak pada fungsi transisinya, dimana untuk setiap pasangan *state-input*, kita bisa memiliki nol (0) atau lebih pilihan untuk *state* berikutnya.



Gambar 2.11. Mesin NFA

Sumber : Teori Bahasa dan Otomata Firrar Utdirartatmo J&J Learning Yogyakarta

Pada gambar 2.11, dari *state* q0 terdapat dua busur keluar yang berlabel *input* 'a'. Dari *state* q0 bila mendapat *input* 'a' bisa berpindah ke *state* q0 atau q1,

yang secara formal dinyatakan δ(q0,a) = {q0,q1} maka otomata ini disebut non-deterministik (tidak pasti arahnya).

Tabel 2.2. Transisi Mesin NFA

Sumber : Teori Bahasa dan Otomata Firrar Utdirartatmo J&J Learning Yogyakarta

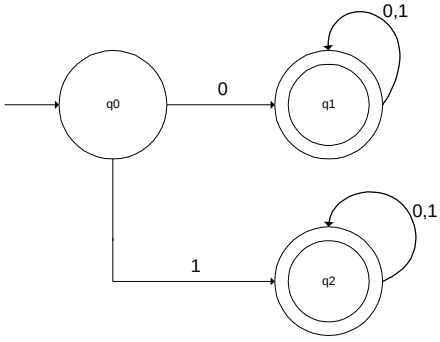
δ	A	B
q0	{q0,q1}	{q1}
q1	{q1}	{q1}

Suatu *string* diterima oleh *Non-deterministic Finite Automata* bila terdapat suatu urutan transisi sehubungan dengan *input string* tersebut dari *state* awal menuju *state* akhir.

Seperti pada *Deterministic Finite Automata*, pada *Non-deterministic Finite Automata* dapat dibuat diagram transisi dari tabel transisinya begitu pula sebaliknya.

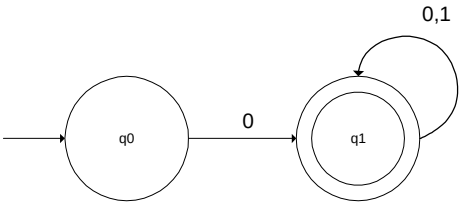
2.3.2.3. Tahapan Pengubahan NFA ke DFA

Dari sebuah mesin *Non-deterministic Finite Automata* dapat dibuat mesin *Deterministic Finite Automata*-nya yang ekivalen (bersesuaian). Ekivalen disini artinya mampu menerima bahasa yang sama. Pada gambar berikut ini, mesin DFA dan mesin NFA dapat menerima bahasa yang sama, yang dalam ekspresi regular = 0 (0 ∪ 1)* :



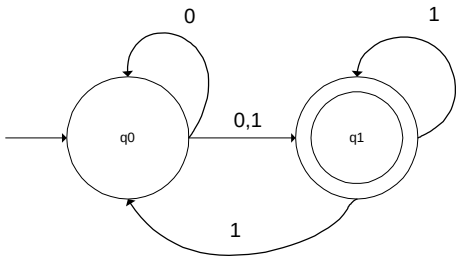
Gambar 2.12. Mesin DFA 0 (0 ∪ 1)*

Sumber : Teori Bahasa dan Otomata Firrar Utdirartatmo J&J Learning Yogyakarta



Gambar 2.13. Mesin NFA 0 (0 ∪ 1)*

Sumber : Teori Bahasa dan Otomata Firrar Utdirartatmo J&J Learning Yogyakarta



Gambar 2.14. Mesin Otomata NFA

Sumber : Teori Bahasa dan Otomata Firrar Utdirartatmo J&J Learning Yogyakarta

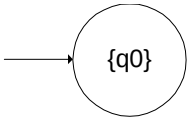
Untuk membuat mesin *Deterministic Finite Automata* dari mesin *Non-deterministic Finite Automata* pada gambar 2.14, yang dilakukan pertama kali adalah membuat tabel transisi NFA tersebut. Bila diketahui Σ = {0,1}, maka table transisinya adalah :

Tabel 2.3. Transisi NFA Σ = {0,1}

Sumber : Teori Bahasa dan Otomata Firrar Utdirartatmo J&J Learning Yogyakarta

δ	0	1
q0	{q0,q1}	{q1}
q1	∅	{q0,q1}

Dengan adanya tabel transisi tersebut akan mempermudah langkah selanjutnya. Dimulai dari *state* awal, kemudian mengikuti transisinya untuk membentuk *state-state* baru, untuk setiap *state* yang terbentuk diikuti lagi transisinya sampai ter-‘cover’ semua. Contoh : Dimulai dengan *state* awal q0, lihat pada gambar 2.15.



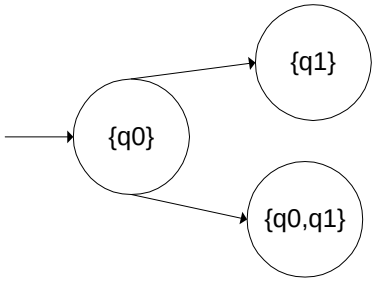
Gambar 2.15. *State* Awal

Sumber : Teori Bahasa dan Otomata Firrar Utdirartatmo J&J Learning Yogyakarta

Selanjutnya telusuri *state* berikutnya yang diperoleh dengan memanfaatkan tabel transisinya :

1. *State* {q0} bila memperoleh *input* 0 menjadi *state* {q0,q1}
2. *State* {q0} bila memperoleh *input* 1 menjadi *state* {q1}

Lihat hasilnya pada gambar 2.16.



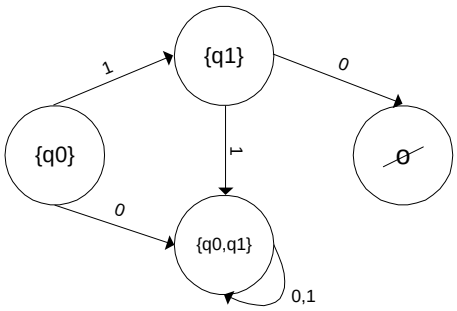
Gambar 2.16. Hasil dari Penelusuran {q0}

Sumber : Teori Bahasa dan Otomata Firrar Utdirartatmo J&J Learning Yogyakarta

Pada gambar setiap *state* kita tuliskan sebagai himpunan *state*. Selanjutnya telusuri *state-state* baru yang terbentuk :

1. *state* {q1} bila memperoleh *input* 0 menjadi *state* ∅
2. *state* {q1} bila memperoleh *input* 1 menjadi *state* {q0,q1}
3. *state* {q0,q1} bila memperoleh *input* 0 menjadi *state* {q0,q1}, ini diperoleh dari $\delta(\{q0,q1\},0) = \{q0,q1\}$ digabung dengan $\delta(q1,0) = \emptyset$, maka hasilnya $\delta(\{q0,q1\},0) = \{q0,q1\}$
4. *state* {q0,q1} bila memperoleh *input* 1 menjadi *state* {q0,q1}, ini diperoleh dari $\delta(q0,1) = \{q1\}$ digabung dengan $\delta(q1,1) = \{q0,q1\}$, maka hasilnya $\delta(\{q0,q1\},1) = \{q0,q1\}$

state yang sama cukup ditulis sekali. Lihat hasilnya pada gambar 2.17.

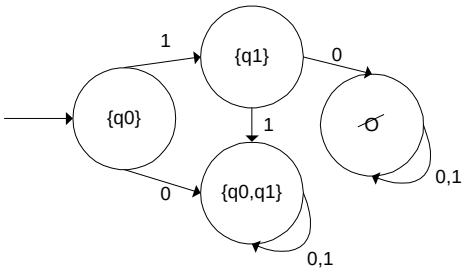


Gambar 2.17. Hasil Setelah Penelusuran {q1} dan {q0,q1}

Sumber : Teori Bahasa dan Otomata Firrar Utdirartatmo J&J Learning Yogyakarta

State q1 menerima *input* 0 menjadi *state* ∅, di sini ∅ digambarkan juga sebagai sebuah *state*.

Semua *state* sudah ditelusuri, tinggal *state* ∅. *State* ∅ menerima *input* 0 atau 1 menjadi *state* ∅, atau $\delta(\emptyset,0) = \emptyset$ dan $\delta(\emptyset,1) = \emptyset$. Lihat hasilnya pada gambar 2.18.

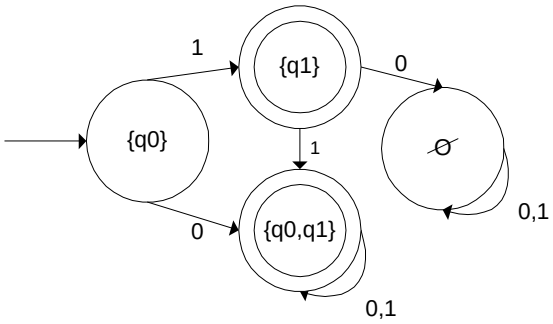


Gambar 2.18. Hasil Setelah Semua Ditelusuri

Sumber : Teori Bahasa dan Otomata Firrar Utdirartatmo J&J Learning Yogyakarta

Pada mesin *Non-deterministic Finite Automata*, himpunan *state* akhir adalah {q1}, maka pada *Deterministic Finite Automata* hasil perubahannya *state-state* akhir adalah semua *state* yang mengandung {q1}. Maka *state* akhirnya sekarang adalah *state* {q1} dan *state* {q0,q1}, atau secara formal : F = {{q1}, {q0,q1}}

Sehingga *Deterministic Finite Automata* hasil ekivalensi dengan *Non-deterministic Finite Automata* pada gambar 2.14 dapat dilihat pada gambar 2.19.



Gambar 2.19. Mesin DFA yang Ekivalen dengan NFA Pada Gambar 2.14

Sumber : Teori Bahasa dan Otomata Firrar Utdirartatmo J&J Learning Yogyakarta.

Kedua otomata tersebut dapat diperiksa apakah ekivalen atau tidak. Untuk membuktikannya, perlu diperlihatkan bahwa suatu bahasa yang diterima oleh *Non-deterministic Finite Automata* juga diterima oleh *Deterministic Finite Automata* ekivalennya tersebut. Bila diketahui *Non-deterministic Finite Automata* semula (gambar 2.14.) menerima *string* '001', maka seharusnya *Deterministic Finite Automata* pada gambar 2.19 juga menerima *string* tersebut.

2.4. *Wave File*

Wave file pada dasarnya tidak jauh berbeda dengan *file* suara VOC. Keduanya merupakan hasil rekaman secara digital. Perbedaan yang sangat menyolok adalah pada *header file*nya. Dari *header* tersebut akan didapatkan informasi tentang bagaimana proses perekaman, menggunakan mode *mono/stereo*, menggunakan berapa *bit* dan sebagainya. *Wave file* itu sendiri tidak berdiri sendiri, karena dari pihak Microsoft selaku pembuat sistem membuat standard RIFF sebagai bagian induknya. Jadi secara tidak langsung *wave* merupakan bagian daripada Microsoft's RIFF yang berfungsi sebagai penyimpanan *multimedia file*. Sebuah *file* dimulai dengan sebuah *file header*, diikuti oleh serangkaian data *chunk*.

2.4.1. RIFF (*Resource Interchange File Format*)

RIFF pada dasarnya bukanlah sebuah *format file* yang baru, RIFF biasanya digunakan untuk aplikasi yang berbasis *windows*. RIFF juga telah banyak digunakan untuk *format* standard di beberapa aplikasi *windows*. Dapat juga digunakan untuk mengonversi berbagai *format file* ke dalam *format* RIFF. Sebagai contohnya sebuah *file* midi dapat dikonversikan menjadi RIFF midi dengan cara menambahkan struktur RIFF yaitu berupa RIFF *chunk* ke dalam *file* midi. Pada dasarnya RIFF dapat dibagi menjadi beberapa bagian pokok seperti :

1. RIFF *chunk*

   Berguna untuk memberitahu bahwa sebuah *file* adalah *file* RIFF

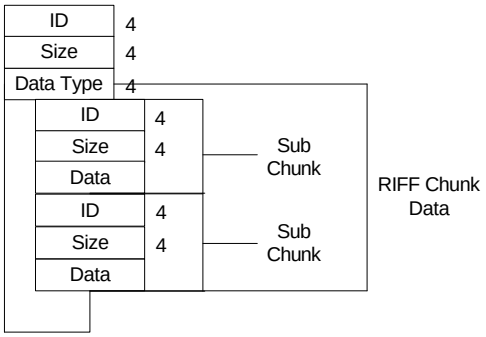
2. *List chunk*

   Berguna untuk memberikan tambahan informasi yang mungkin diperlukan seperti nama perusahaan, nama pembuat, tanggal pembuatan, penjelasan tentang isi data suara dan sebagainya.

3. *Sub chunk*

   Berguna untuk memberikan tambahan informasi untuk *chunk* utama apabila datanya sudah penuh.

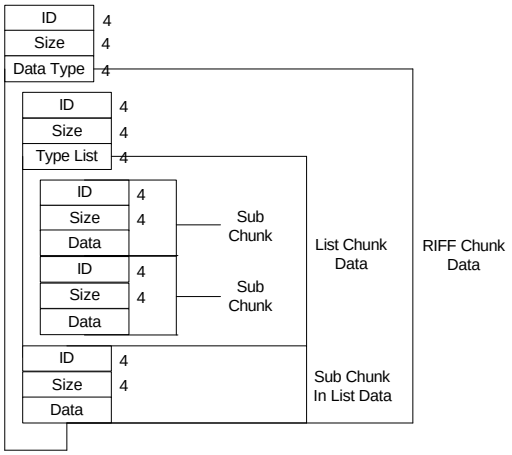
Hubungan dari ketiga *chunk* tersebut dapat dilihat pada gambar 2. 20.



Gambar 2.20. Struktur RIFF *Chunk*

Sumber : Studi Analisis Format File Suara & Implementasi Program dengan Sound Blaster 4-E/127 1997 Sentot STTS Surabaya

Sedangkan hubungan antara RIFF *chunk* dengan *list chunk* dapat dilihat pada gambar 2.21.



Gambar 2.21. Struktur RIFF *Chunk* dengan *List Chunk*

Sumber : Studi Analisis Format File Suara & Implementasi Program dengan Sound Blaster 4-E/127 1997 Sentot STTS Surabaya

2.4.1.1. RIFF *Chunk*

Pada awal *file* RIFF harus berisi RIFF *chunk* dan berikutnya dapat berisi satu atau lebih *sub chunk*. 4 *byte* pertama pada sebuah *sub chunk* akan berisi identitas *format* data yang disimpan. Adapun *format* data yang dapat diakses oleh Microsoft Windows melalui tipe *file* RIFF adalah sebagai berikut :

Tabel 2.4. *Format* Data Tipe *File* RIFF

Sumber : Studi Analisis Format File Suara & Implementasi Program dengan Sound Blaster 4-E/127 1997 Sentot STTS Surabaya

Tipe File	Tipe Data	Extension File
Wave Form Audio File	Wave	.WAV
Audio Video Interleaved File	AVI	.AVI
Midi File	RMID	.RMI
Device Independent Bitmap	RDIP	.RDI
Palette File	PAL	.PAL

Suatu *file* RIFF terdiri dari bagian-bagian yang kecil yang biasanya disebut dengan *chunk*. Sebuah *chunk* itu sendiri akan terdiri dari 4 karakter ASCII yang berfungsi sebagai identitas, 4 *bytes* berikutnya yang berfungsi sebagai besar *chunk* dan yang terakhir adalah sebuah *array* yang berisi data daripada *chunk* itu

sendiri. Baik untuk RIFF *chunk*, *list chunk* maupun *sub chunk* memiliki struktur yang sama.

2.4.1.2. *List chunk*

Di dalam sebuah RIFF *chunk* dapat ditambahkan sebuah *list chunk*. Adapun struktur daripada *list chunk* adalah 4 karakter ASCII yang pertama berfungsi untuk identitas, 4 *byte* berikutnya berisi tentang besar daripada data dan yang terakhir adalah data itu sendiri. 4 *byte* pertama daripada data suara adalah tipe *list* dan pada saat ini dari pihak Microsoft hanya memiliki 1 buah tipe *list* yang biasa disebut “INFO”. Didalam “INFO” itu sendiri masih terdapat *sub chunk - sub chunk* yang lain. Adapun struktur daripada “INFO” pada umumnya adalah struktur *chunk* untuk *wave file*.

2.4.1.2.1. Struktur *Wave File*

- Wformat tag

  Berfungsi untuk mengintepretasikan posisi dari data suara dalam data *chunk*. Untuk *wave file* sampai saat ini masih menggunakan *format* PCM (*Pulse Code Modulation*). Didalam aplikasi yang dipakai biasa memakai konstanta wave_format_PCM. Struktur *wave file* selengkapnya beserta struktur untuk PCM adalah :

  [ChunkID] [ChunkSize] [Format]

  - [ChunkID] → 4 byte

    Berisi karakter R, I, F dan F

  - [ChunkSize] → 4 byte

    Berisi panjang data selain [ChunkID] dan [ChunkSize]

  - [Format]

    Chunk data dari *file*. Mempunyai bentuk sebagai berikut :

    [Format] = [FormatID] [FormatChunk] [DataChunk]

- [FormatID] → 4 byte

  Berisi data berupa karakter W,A,V dan E

- [FormatChunk]

  Berisi data yang menentukan *format* dari data yang terdapat dalam data *chunk.* Mempunyai bentuk sebagai berikut :

  [FormatChunk] = [SubChunk1ID] [SubChunk1Size]
  [AudioFormat] [NumChannels]
  [SampleRate] [ByteRate] [BlockAlign]
  [BitsPerSample]

  - [SubChunk1ID] → 4 byte

    Berisi data berupa karakter f,m,t dan spasi yang menandai bahwa blok tersebut merupakan *format chunk*

  - [SubChunk1Size] → 4 byte

    Adalah panjang dari data berikutnya dalam *format chunk*

  - [AudioFormat] → 2 byte

    Menunjukkan kategori dari *format wave file*, biasanya menggunakan PCM (*Pulse Code Modulation*) format yaitu 01[1]

  - [NumChannels] → 2 byte

    Untuk menentukan suara yang dihasilkan *stereo* atau *mono*, 1 untuk *mono* dan 2 untuk *stereo*

  - [SampleRate] → 4 byte

    Menunjukkan *sampling rate* atau frekuensi yang digunakan dalam proses perekaman data suara. Frekuensi yang

disediakan untuk data suara adalah 11025 Hz, 22050 Hz atau 44100 Hz.

- [ByteRate] → 4 byte

  Menunjukkan rata-rata jumlah *byte* perdetik data yang harus dikirimkan. Hal ini sangat berguna dalam program karena digunakan untuk menentukan ukuran dari *buffer* yang dipakai untuk memainkan data suara. Cara Menghitungnya sebagai berikut :

  [ByteRate] = NumChannels X SampleRate X
  (BitsPerSample/8)

- [BlockAlign] → 2 byte

  Menunjukkan jumlah *byte* yang digunakan untuk membentuk sebuah sample tunggal. Cara menghitungnya sebagai berikut :

  [BlockAlign] = NumChannels X (BitsPerSample/8)

  ⊙ [BitsPerSample] → 2 byte

  Berisi nilai yang menentukan berapa *bit* yang digunakan dalam sebuah sample, yaitu delapan *bit* atau enambelas *bit* per *sample*

▪ [DataChunk]

Berisi data sesungguhnya dari *wave file*, formatnya sesuai dengan [FormatChunk]. Mempunyai bentuk sebagai berikut :

[DataChunk] = [SubChunk2ID] [SubChunk2Size] [Data]

- [SubChunk2ID] → 4 byte

Berisi karakter d,a,t dan a yang menandai data *chunk*

- [SubChunk2Size] → 4 byte

  Menunjukkan panjang data yang akan mengikuti

- [Data]

  Data *waveform* yang sebenarnya