1.1. Pendahuluan

**Pengertian  Teori Bahasa**

Teori bahasa membicarakan tentang bahasa formal (*formal language*), terutama dalam kepentingan perancangan kompilator (*compiler*) dan pemroses naskah (*text processor*). Bahasa formal merupakan kumpulan kalimat. Semua kalimat dalam sebuah bahasa dibangkitkan oleh sebuah tata bahasa (*grammar*) yang sama. Sebuah bahasa formal bisa dibangkitkan oleh dua atau lebih tata bahasa berbeda. Dikatakan bahasa formal karena *grammar*diciptakan mendahului pembangkitan setiap kalimatnya. Bahasa manusia bersifat sebaliknya; *grammar*diciptakan untuk meresmikan kata-kata yang hidup di masyarakat. Dalam pembicaraan selanjutnya ‘bahasa formal’ akan disebut ‘bahasa’ saja.

**Pengertian Automata**

Automata adalah mesin abstrak yang dapat mengenali (*recognize*), menerima (*accept*), atau membangkitkan (*generate*) sebuah kalimat dalam bahasa tertentu. Automata berasal dari bahasa Yunani *automaton*, yang berarti sesuatu yang bekerja secara otomatis (mesin). Istilah *automaton* sebagai bentuk tunggal dan automata sebagai bentuk jamak. Teori Automata adalah teori tentang mesin abstrak yang :

* bekerja sekuensial
* menerima *input*
* mengeluarkan *output*

Pengertian mesin bukan hanya mesin elektronis/mekanis saja, melainkan segala sesuatu (termasuk perangkat lunak) yang memenuhi ketiga ciri di atas. Penggunaan automata pada perangkat lunak terutama pada pembuatan kompiler bahasa pemrograman. Secara garis besar ada dua fungsi automata dalam hubungannya dengan bahasa, yaitu :

1. Fungsi automata sebagai pengenal (*RECOGNIZER*) *string-string* dari suatu bahasa, dalam hal ini bahasa sebagai masukan dari automata
2. fungsi automata sebagai pembangkit (*GENERATOR*) *string-string* dari suatu bahasa, dalam hal ini bahasa sebagai keluaran dari automata

1.2. Konsep Teori Bahasa dan Automata

Teori bahasa dan otomata merupakan bagian dari teori komputasi pada ilmu komputer. Beberapa teori komputasi datang dari bahasa dan rekayasa sistem, terutama yang berbasiskan matematika. Dalam hal ini penekanannya adalah pada pemecahan masalah. Melalui contoh-contoh ilustrasi masalah dapat dikenali latar belakang dari suatu konsep dan hubungannya dengan definisi dan teorema yang ada.

Secara teoritis ilmu komputer diawali dari sejumlah berbeda disiplin ilmu;

1. Ahli biologi mempelajari *neural network*,
2. Insinyur elektro mengembangkan *switching* sebagai *tools* untuk mendesain perangkat keras,
3. Matematikawan bekerja berdasarkan logika, dan
4. Ahli bahasa menyelidiki tata bahasa untuk bahasa alami (*natural language*)

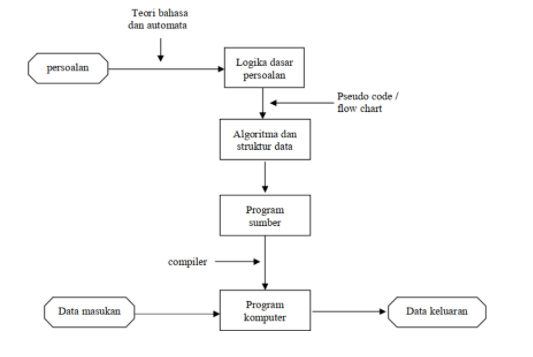
*Finite state automata* dan ekspresi reguler awal dikembangkan berdasarkan pemikiran *neural network* dan *switching circuit*. *Finite state* automata merupakan *tools* yang sangat berguna dalam perancangan suatu penganalisa leksikal (*lexical analyzer*) yang berguna dalam mengelompokkan karakter-karakter ke dalam token-token sebagai unit terkecil dalam mengenali pola.

Teori bahasa merupakan suatu gagasan mendasar dalam komputasi yang menjadi tools untuk mengenali persoalan. Gagasan dasar tersebut dimodel dengan suatu simbol-simbol yang merepresentasikan juga suatu fungsi dari komputer digital.

Teori bahasa pada awalnya lebih diarahkan untuk mengenali suatu tata bahasa dan dapat mendefinisikan spesifikasi formal dari tata bahasa tersebut. Sehingga pada akhirnya dapat didefinisikan langkah-langkah algoritmik dalam pemrosesan tata bahasa.

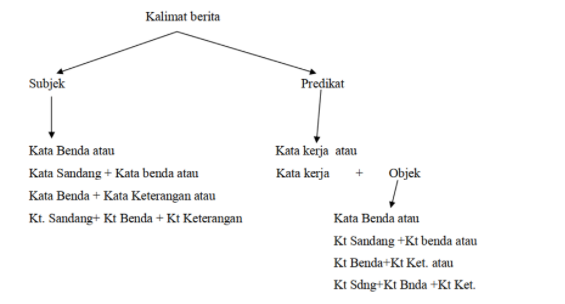
1.3. Teori Bahasa dan Otomata dalam Ilmu Komputer

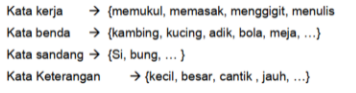
Teori memberikan konsep dan prinsip yang menolong untuk memahami perilaku dari suatu persoalan yang berkorelasi dengan teori tersebut. Bidang ilmu komputer meliputi topik yang luas, dari perancangan mesin sampai pemrograman. Disamping perbedaan yang ada, terdapat keseragaman prinsip-prinsip umum yang dipakai. Untuk mempelajari prinsip-prinsip dasar tersebut, kita mengkonstruksi suatu mesin otomata sebagai model abstrak dari komputer dan komputasi. Model ini memiliki fungsi-fungsi yang penting dan umum pada perangkat keras dan perangkat lunak komputer. Meskipun model tersebut sederhana untuk diterapkan langsung pada dunia nyata, keuntungan yang diperoleh dari mempelajarinya adalah memberikan landasan untuk basis dari suatu pengembangan algoritma. Pendekatan ini, juga diterapkan pada ilmu sains lainnya.



1.4. Konsep Tata Bahasa

Penulisan suatu kalimat dalam sebuah bahasa, akan mengikuti suatu aturan tertentu yang berlaku pada bahasa tersebut. Aturan tersebut dikenal sebagai Tata Bahasa (*grammar*). Sebagai contoh dalam bahasa Indonesia, penulisan sebuah kalimat berita akan mengikuti aturan SP (Subjek Predikat), atau lebih detilnya aturan penulisan suatu kalimat berita adalah :





Dengan mengikuti aturan tata bahasa tersebut, dapat direkonstruksi suatu kalimat berita sebagai berikut :

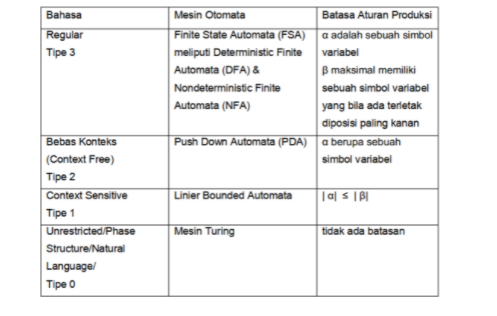
1. Adik menulis
2. Kucing menggigit tikus
3. Si kambing cantik memakan sayuran segar

Periksalah kalimat berikut apakah memenuhi aturan tersebut :

* Si tikus jorok mengejar kucing galak
* Si bola besar menendang kambing jelek

## 1.5. Klasifikasi Tata Bahasa

Tata bahasa (*grammar*) bisa didefinisikan secara formal sebagai kumpulan dari himpunan-himpunan variabel, simbol-simbol terminal, simbol awal yang dibatasi oleh aturan-aturan produksi. Pada tahun 1959 seorang ahli bernama Noam Chomsky melakukan penggolongan tingkatan bahasa menjadi empat, yang disebut dengan Hirarki Chomsky. Penggolongan tersebut bisa dilihat pada tabel berikut:



## 2.1. Sejarah Automata

Automata atau Otomata bermula sebelum komputer ada, yaitu pada teori di bidang sistem logika matematika atau formal. Ilmuwan **David Hilbert** telah mencoba menciptakan **algoritma umum untuk melakukan pembuktian (seluruh) persoalan matematika secara otomatis**, yaitu mampu menentukan salah benarnya sembarang proposisi (pernyataan mengenai hal-hal yang dapat dinilai benar atau salah) matematika.

Tahun 1931, KurtGdel mempublikasikan teori ketidaklengkapan dimana ia membuktikan prosedur/ algoritma yang dikehendaki David Hilbert tersebut tidak akan pernah ada. **KurtGdel** membangun rumus di kalkulus predikat yang diterapkan pada bilangan bulat yang memiliki pernyataan-pernyataan definisi yang tidak dapat dibuktikan maupun dibantah di dalam sistem logika yang mungkin dibangun manusia.

Formalisasi argumen teorema ketidaklengkapan KurtGdel ini berikut penjelasan dan formalisasi selanjutnya dari prosedur efektif secara intuisi merupakan salah satu pencapaian intelektual terbesar abad 20, yaitu abad dimana formalisasi berkembang dengan pesat.

Pengembangan teori otomata, komputasi dan teori bahasa berikutnya difasilitasi perkembangan bidang psyco-linguistic. Bidang psyco-linguistic berupaya menjawab pertanyan-pertanyan berikut :

* Apakah bahasa secara umum?
* Bagaimana manusia mengembangkan bahasa?
* Bagaimana manusia memahami bahasa?
* Bagaimana manusia mengajarkan bahasa ke anak-anaknya?
* Apa gagasan-gagasan yang dapat dinyatakan dan bagaimana caranya?
* Bagaimana manusia membangun kalimat-kalimat dari gagasan-gagasan yang berada dipikirannya ?

Sekitar tahun 1950-an, Noam Chomsky menciptakan model matematika sebagai sarana untuk mendeskripsikan bahasa serta menjawab pertanyaan-pertanyaan di atas. Saat ini dimulai pendalaman bidang bahasa computer.

Perbedaan antara bahasa komputer dan bahasa manusia adalah sampai sekarang belum diketahuinya bagaimana cara manusia mengartikan bahasa, sementara dengan pasti dapat mengartikan bahasa pada komputer.

**Noam Chomsky** mengemukakan perangkat format disebut grammar untuk memodelkan properti-properti bahasa. Tata bahasa (grammer) bisa didefinisikan secara, formal sebagai kumpulan dari himpunan-himpunan variabel, simbol-simbol, terminal, simbol awal, yang dibatasi oleh aturan- aturan produksi. Tingkat bahasa dapat digolongkan menjadi empat, yaitu :

**1. Bahasa : Regular type 3**

* Mesin otomata : Finite State Otomata (FSA) meliputi deterministic finite automata dan non deterministic finite automata.
* Batasan aturan produksi : adalah sebuah simbol variabel maksimal memiliki sebuah simbol variabel yang bila terletak di posisi paling kanan.

**2.Bahasa : Bebas konteks/context free /type 2**

* Mesin otomata : Push down automata (PDA)
* Batasan aturan produksi : Berupa sebuah simbol variabel.

**3.Bahasa : Context sensitive/type 1**

* Mesin otomata : Linier bounded automata
* Batasan aturan produksi :-

**4.Bahasa : Unrestricted /phase /natural language/type 0**

* Mesin otomata : Mesin turing
* Batasan aturan produksi : Tidak ada batasan

Konsep mind map : bagaimana sebuah komputer dapat memahami perintah yang kita perintah kan kedevice melalui bahasa program, namun dengan proses yang masih manusiawi pada saat menjalankan sebuah program ( konsep expert nya bagaimana sebuah google translate terjadi di komputer seperti itulah awalan dari sebuah teori bahasa dan automata

## 2.2. Definisi Automata

**Teori Bahasa**

* Teori bahasa membicarakan bahasa formal (*formal language*), terutama untuk kepentingan perancangan kompilator (*compiler*) dan pemroses naskah (*text processor*).
* Bahasa formal adalah kumpulan *kalimat*. Semua kalimat dalam sebuah bahasa dibangkitkan oleh sebuah tata bahasa (*grammar*) yang sama.
* Sebuah bahasa formal bisa dibangkitkan oleh dua atau lebih tata bahasa berbeda.
* Dikatakan bahasa formal karena grammar diciptakan mendahului pembangkitan setiap kalimatnya.
* Bahasa Natural/manusia bersifat sebaliknya; grammar diciptakan untuk meresmikan kata-kata yang hidup di masyarakat. Dalam pembicaraan selanjutnya ‘bahasa formal’ akan disebut ‘bahasa’ saja.

**Otomata (*Automata*)**

* Otomata adalah mesin abstrak yang dapat mengenali (*recognize*), menerima (*accept*), atau membangkitkan (*generate*) sebuah kalimat dalam bahasa tertentu.

## 2.3. Istilah dalam Teori Bahasa dan Automata

* Simbol adalah sebuah entitas abstrak (seperti halnya pengertian *titik*dalam geometri). Sebuah huruf atau sebuah angka adalah contoh simbol.
* String adalah deretan terbatas (*finite*) simbol-simbol. Sebagai contoh, jika *a*, *b*, dan *c* adalah tiga buah simbol maka *abcb*adalah sebuah string yang dibangun dari ketiga simbol tersebut.
* Jika *w*adalah sebuah string maka panjang string dinyatakan sebagai |*w*| dan didefinisikan sebagai cacahan (banyaknya) simbol yang menyusun string tersebut. Sebagai contoh, jika *w* = *abcb*maka |*w*|= 4.
* String hampa adalah sebuah string dengan nol buah simbol. String hampa dinyatakan dengan simbol ε (atau ^) sehingga |ε|= 0. String hampa dapat dipandang sebagai simbol hampa karena keduanya tersusun dari nol buah simbol.
* Alfabet adalah himpunan hingga (*finite set*) simbol-simbol.

## 2.4. Operasi Dasar String

Diberikan dua string : *x*= *abc*, dan *y*= *123*

* **Prefik**string *w*adalah string yang dihasilkan dari string *w*dengan menghilangkan *nol* atau lebih simbol-simbol paling belakang dari string *w*tersebut.
  + Contoh : *abc*, *ab*, *a*, dan ε adalah semua Prefix(*x*)
* **ProperPrefix**string *w*adalah string yang dihasilkan dari string *w*dengan menghilangkan*satu*atau lebih simbol-simbol paling belakang dari string *w*tersebut.
  + Contoh : *ab*, *a*, dan ε adalah semua ProperPrefix(*x*)
* **Postfix**(atau Sufix) string *w*adalah string yang dihasilkan dari string *w*dengan menghilangkan *nol*atau lebih simbol-simbol paling depan dari string *w*tersebut.
  + Contoh : *abc*, *bc*, *c*, dan ε adalah semua Postfix(*x*)
* **ProperPostfix**(atau PoperSufix) string *w*adalah string yang dihasilkan dari string *w* dengan menghilangkan *satu*atau lebih simbol-simbol paling depan dari string *w* tersebut.
  + Contoh : *bc*, *c*, dan ε adalah semua ProperPostfix(*x*).
* **Head string***w*adalah simbol paling depan dari string *w*.
  + Contoh : *a*adalah Head(*x*)
* **Tail**string *w*adalah string yang dihasilkan dari string *w*dengan menghilangkan simbol paling depan dari string *w*tersebut.
  + Contoh : *bc*adalah Tail(*x*)
  + *w*adalah string yang dihasilkan dari string *w*dengan menghilangkan *nol*atau lebih simbol-simbol paling depan dan/atau simbol-simbol paling belakang dari string *w* tersebut.
  + Contoh : *abc*, *ab*, *bc*, *a*, *b*, *c*, dan ε adalah semua Substring(*x*)
* **ProperSubstring**string *w*adalah string yang dihasilkan dari string *w*dengan menghilangkan *satu*atau lebih simbol-simbol paling depan dan/atau simbol-simbol paling belakang dari string *w*tersebut.
  + Contoh : *ab*, *bc*, *a*, *b*, *c*, dan ε adalah semua Substring(*x*)
* **Subsequence**string *w*adalah string yang dihasilkan dari string *w*dengan menghilangkan *nol*atau lebih simbol-simbol dari string *w*tersebut.
  + Contoh : *abc*, *ab*, *bc*, *ac*, *a*, *b*, *c*, dan ε adalah semua Subsequence(*x*)
* **ProperSubsequence**string *w*adalah string yang dihasilkan dari string *w*dengan menghilangkan *satu*atau lebih simbol-simbol dari string *w*tersebut.
  + Contoh : *ab*, *bc*, *ac*, *a*, *b*, *c*, dan ε adalah semua Subsequence(*x*)
* **Concatenation**adalah penyambungan dua buah string. Operator concatenation adalah*concate*atau tanpa lambang apapun.
  + Contoh : concate(*xy*) = *xy*= *abc123*
* **Alternation**adalah pilihan satu di antara dua buah string. Operator alternation adalah*alternate*atau | |.
  + Contoh : alternate(*xy*) = *x*|*y*= *abc*atau *123*
* **Kleene**Closure : *x*\* = ε|*x*|*xx*|*xxx*|… = ε|*x*|*x*2 |*x*3 |…
  + Positive Closure : *x*+ = *x*|*xx*|*xxx*|… = *x*|*x*2 |*x*3 |…

## 2.5. Sifat Dasar Operasi String

* Tidak selalu berlaku : *x*= Prefix(*x*)Postfix(*x*)
* Selalu berlaku : *x*= Head(*x*)Tail(*x*)
* Tidak selalu berlaku : Prefix(*x*) = Postfix(*x*) atau Prefix(*x*) ≠ Postfix(*x*)
* Selalu berlaku : ProperPrefix(*x*) ≠ ProperPostfix(*x*)
* Selalu berlaku : Head(*x*) ≠ Tail(*x*)
* Setiap Prefix(*x*), ProperPrefix(*x*), Postfix(*x*), ProperPostfix(*x*), Head(*x*), dan Tail(*x*) adalah Substring(*x*), tetapi tidak sebaliknya
* Setiap Substring(*x*) adalah Subsequence(*x*), tetapi tidak sebaliknya
* Dua sifat aljabar concatenation :
  + Operasi concatenation bersifat asosiatif : *x*(*yz*) = (*xy*)*z*
  + Elemen identitas operasi concatenation adalah ε : ε*x*= *x*ε = *x*
* Tiga sifat aljabar alternation :
  + Operasi alternation bersifat komutatif : *x*|*y*= *y*|*x*
  + Operasi alternation bersifat asosiatif : *x*|(*y*|*z*) = (*x*|*y*)|*z*
  + Elemen identitas operasi alternation adalah dirinya sendiri : *x*|*x*= *x*
* Sifat distributif concatenation terhadap alternation : *x*(*y*|*z*) = *xy*|*xz*
* Beberapa kesamaan :
  + Kesamaan ke-1 : (*x*\*)\* = (*x*\*)
  + Kesamaan ke-2 : ε|*x*+ = *x*+ |ε = *x*\*
  + Kesamaan ke-3 : (*x*|*y*)\* = ε|*x*|*y*|*xx*|*yy*|*xy*|*yx*|… = semua string yang merupakan concatenation dari nol atau lebih *x*, *y*, atau keduanya.

## 2.6. Contoh Terapan Teori Otomata

Contoh Penerapan Teori Bahasa Otomata

Model switch on/off digambarkan sebagai berikut:

**Contoh 1:**

Model tersebut mengingat apakah *switch*berada dalam state ”*on*” atau state ”*off*”. Model memungkinkan user untuk menekan tombol yang memiliki pengaruh berbeda tergantung pada keadaan *switch*:

* *switch*berada dalam *state*“off” maka setelah tombol ditekan *state*berubah menjadi “on”.
* Jika *switch*berada dalam *state*“on” maka setelah tombol ditekan *state*berubah menjadi “off”.

Model pada Gambar 1 dapat dipandang sebagai model *finite automata*sederhana.

Dalam *finite automata*, ***state***dinyatakan oleh lingkaran dan dalam Contoh 1 *state*diberi nama “on” dan “off”. ***Arc***(Peta Hubungan Kerja Kegiatan) diantara *state*diberi label “input “ yang menyatakan pengaruh eksternal pada sistem. Dalam Contoh 1 kedua *arc*diberi label ‘push” yang menyatakan *user* menekan tombol tertentu. Salah satu *state*dinyatakan sebagai ***start state***atau ***initial state***yang merupakan *state* dimana sistem berada dalam keadaan awal. Dalam Contoh *start state*adalah off. Dalam pembahasan selanjutnya, *start state*ditunjukan oleh kata *start*dan panah menuju *start state*tersebut. Dalam Gambar 1 *state*on dinyatakan sebagai ***final***atau ***accepting***state.

Dalam *state*tersebut, peralatan yang sedang dikontrol oleh *switch*akan beroperasi. Dalam pembahasan selanjutnya, *final State*dinyatakan dalam lingkaran ganda.

**Contoh 2:**

*Finite automaton*berikut dapat dinyatakan sebagai bagian dari *lexical analyzer*.

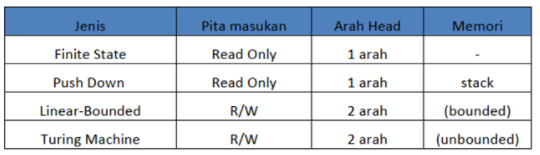
Tugas dari *automaton*tersebut adalah mengenali *keyword*“then” sehingga diperlukan lima*state*masing-masing menyatakan posisi yang berbeda dalam kata “then” yang telah dicapai sejauh ini. Posisi ini berhubungan dengan prefix dari kata yang berkisar dari kata string kosong (tidak ada kata yang dikenali sejauh ini) sampai dengan kata lengkap. Dalam Gambar 2, input dinyatakan oleh huruf. *Start state*merupakan string kosong, dan setiap *state* memiliki transisi pada huruf selanjutnya dari kata then ke *state*yang menyatakan prefix selanjutnya yang lebih besar. *State*yang diberi nama “then” dimasuki ketika input mengeja kata “then”. Karena fungsi dari model dalam Gambar 2 adalah mengenali kata then, maka*state*“then”dinyatakan sebagai *accepting state*.

## 2.7. Sifat-Sifat Automata

Automata adalah suatu mesin sekuensial (otomatis), yang menerima input (dari pita masukan) dan mengeluarkan output, keduanya dalam bentuk diskrit. Automata mempunyai sifat-sifat :

* Kelakuan mesin bergantung pada rangkaian masukan yang diterima mesin tersebut.
* Setiap saat, mesin dapat berada pada satu status tertentu dan dapat berpindah ke status baru karena adanya perubahan input.
* Rangkaian input (diskrit) pada mesin automata dapat dianggap sebagai bahasa yang harus “dikenali” oleh sebuah automata. Setelah pembacaan input selesai, mesin automata kemudian membuat “keputusan”.

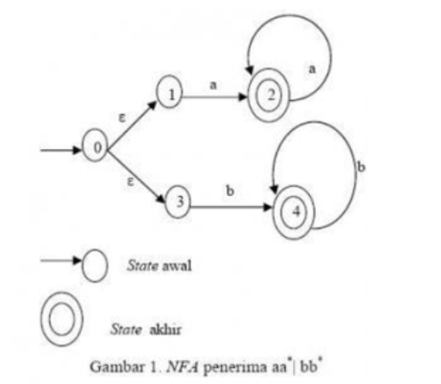
Jenis- jenis automata :



Pada bahasan ini jenis automata yang akan dipakai adalah Finite State Automata (FSA). FSA adalah mesin yang dapat mengenali kelas bahasa reguler dan memiliki sifat-sifat :

1. Pita masukan (input tape) berisi rangkaian simbol (string) yang berasal dari himpunan simbol / alfabet.
2. Setiap kali setelah membaca satu karakter, posisi read head akan berada pada symbol berikutnya.
3. Setiap saat, FSA berada pada status tertentu
4. Banyaknya status yang berlaku bagi FSA adalah berhingga.
   * Suatu FSA didefenisikan sebagai F = (Q, S, q0, d, F) dengan
   * Q = himpunan state (keadaan)
   * ∑ = himpunan input
   * q0 e Q adalah keadaan awal
   * &= Q x S .. Q adalah tabel transisi
   * F = keadaan akhir

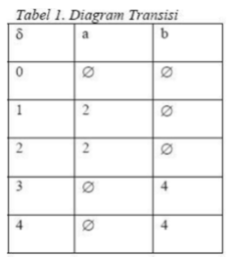
Suatu NFA dapat direpresentasikan dalam bentuk bagan sebagai suatu graf yang diberi label dan disebut dengan graf transisi. Dalam graf transisi ini nodal adalah state dan label dari sisi menyatakan fungsi transisi, contoh Graf transisi NFA dapat dilihat pada gambar dibawah.



Gambar diatas mempunyai defenisis formal sebagai berikut :

Q = {0, 1, 2, 3, 4}  
∑ = {a,b}  
q0 = 0  
F = {2, 4}

&= diagram transisi dapat dilihat pada tabel dibawah



## 3.1. Istilah Penting dalam Bahasa dan Automata

* Dalam pembicaraan *grammar*, anggota alfabet dinamakan simbol terminal atau *token.*
* Kalimat adalah deretan hingga simbol-simbol terminal.
* Bahasa adalah himpunan kalimat-kalimat. Anggota bahasa bisa tak hingga kalimat.
* Simbol-simbol berikut adalah simbol terminal:

1. huruf kecil awal alfabet, misalnya : a, b, c
2. simbol operator, misalnya : +, -, dan ×
3. simbol tanda baca, misalnya : (, ), dan ;
4. string yang tercetak tebal, misalnya : **if**, **then**, dan **else**.

* Simbol-simbol berikut adalah simbol non terminal:

1. huruf besar awal alfabet, misalnya : A, B, C
2. huruf S sebagai simbol awal
3. string yang tercetak miring, misalnya : *expr* dan *stmt*.

* Huruf besar akhir alfabet melambangkan simbol terminal atau non terminal, misalnya : X, Y, Z.
* Huruf kecil akhir alfabet melambangkan string yang tersusun atas simbol-simbol terminal, misalnya : x, y, z.
* Huruf yunani melambangkan *string* yang tersusun atas simbol-simbol terminal atau simbol-simbol non terminal atau campuran keduanya, misalnya : α, β, dan γ.
* Sebuah produksi dilambangkan sebagai α → β, artinya : dalam sebuah derivasi dapat dilakukan penggantian simbol α dengan simbol β.
* Simbol α dalam produksi berbentuk α → β disebut ruas kiri produksi sedangkan simbol β disebut ruas kanan produksi.
* Derivasi adalah proses pembentukan sebuah kalimat atau sentensial. Sebuah derivasi dilambangkan sebagai : α ⇒ β.
* Sentensial adalah string yang tersusun atas simbol-simbol terminal atau simbol-simbol non terminal atau campuran keduanya.
* Kalimat adalah string yang tersusun atas simbol-simbol terminal. Jelaslah bahwa kalimat adalah kasus khusus dari sentensial.
* Pengertian terminal berasal dari kata *terminate* (berakhir), maksudnya derivasi berakhir jika sentensial yang dihasilkan adalah sebuah kalimat (yang tersusun atas simbol-simbol terminal itu).
* Pengertian non terminal berasal dari kata *not terminate* (belum/tidak berakhir), maksudnya derivasi belum/tidak berakhir jika sentensial yang dihasilkan mengandung simbol non terminal.

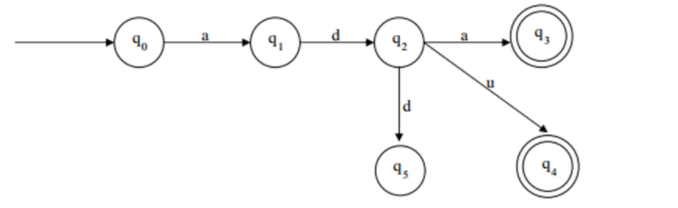
## 3.2. Pengertian Bahasa

Bahasa di dalam kamus adalah suatu sistem yang meliputi pengekspresian gagasan, fakta, konsep, termasuk sekumpulan simbol-simbol dan aturan untuk melakukan manipulasinya. Bahasa bisa juga disebut sebagai rangkaian simbol-simbol yang mempunyai makna.

Otomata merupakan suatu sistem yang terdiri atas sejumlah berhingga *state*, di mana *state*menyatakan informasi mengenai *input*. Otomata juga dianggap sebagai mesin otomatis (bukan mesin fisik) yang merupakan suatu model matematika dari suatu sistem yang menerima input dan menghasilkan output, serta terdiri dari sejumlah berhingga *state*.

Hubungan di antara bahasa dan otomata adalah bahasa dijadikan sebagai *input*oleh suatu mesin otomata, selanjutnya mesin otomata akan membuat keputusan yang mengindikasikan apakah *input* itu diterima atau tidak.

Misalnya, kita memiliki sebuah mesin sederhana yang menerima *input*kata dalam bahasa Indonesia, hal ini bisa dilihat pada gambar berikut ini.



Pada gambar di atas, bila mesin mendapat *string input*berikut.

1. ada : diterima
2. adu : diterima
3. add : ditolak

Sebuah *string input*diterima bila mencapai *state*akhir/*final state*yang disana digambarkan dengan lingkaran ganda. Mesin ini memiliki 6 *state*, {q0, q1, q2, q3, q4, q5}, yang mana adalah himpunan *state* yang ada pada mesin itu. *State*awal dari mesin adalah q0 . {q3, q4}adalah himpunan *state*akhir/*final*. Sedangkan himpunan simbol *input*adalah {a, d, u}.

**Tata Bahasa Regular**

Aturan :

1. Simbol pada sebelah kiri harus berupa sebuah simbol variabel
2. Simbol pada sebelah kanan maksimal hanya memiliki sebuah simbol variabel dan bila ada terletak di posisi paling kanan

Contoh :

1. A → b (Diterima)
2. a → B (Ditolak, karena simbol pada sebelah kiri harus berupa sebuah simbol variabel)
3. A → B (Diterima)
4. A → bC (Diterima)
5. A → Bc (Ditolak, karena simbol variabel pada sebelah kanan harus berada pada posisi paling kanan)
6. A → bcD (Diterima)
7. A → bCD (Ditolak, karena simbol pada sebelah kanan maksimal hanya memiliki sebuah simbol variabel)
8. Ab → c (Ditolak, karena simbol pada sebelah kiri harus berupa sebuah simbol variabel)

Tentukan apakah produksi-produksi berikut memenuhi aturan tata bahasa Regular

1. A → b
2. B → bdB
3. B → C
4. B → bC
5. B → Ad
6. B → bcdef
7. B → bcdefg
8. A → aSa
9. A → aSS
10. A → є
11. Ad → dB

**Tata Bahasa Bebas Konteks**

Aturan : Simbol pada Sebelah kiri harus berupa sebuah simbol variabel

Contoh :

1. A → b (Diterima)
2. A → B (Diterima)
3. A → bC (Diterima)
4. A → Bc (Diterima)
5. A → BcD (Diterima)
6. A → AAA (Diterima)
7. a → b (Ditolak, karena simbol pada sebelah kiri harus berupa sebuah simbol variabel)
8. Ab → c (Ditolak, karena simbol pada sebelah kiri harus berupa sebuah symbol variabel)
9. AB → c (Ditolak, karena simbol pada sebelah kiri harus berupa sebuah symbol variabel)

Tentukan apakah aturan produksi-produksi berikut memenuhi aturan tata bahasa bebas konteks.

1. A → aSa
2. A → Ace
3. A → ab
4. A → є
5. B → bcdef
6. B → bcdefG
7. A → aSa
8. A → aSS
9. A → BCDEF
10. Ad → dB
11. A → AAAAA
12. d → A

**Tata Bahasa *Context Sensitive***

Aturan :

* Simbol pada sebelah kiri harus minimal ada sebuah simbol variabel
* Jumlah simbol pada ruas sebelah kiri harus lebih kecil atau sama dengan jumlah simbol pada ruas kanan

Contoh :

1. A → bc (Diterima)
2. Ab → cd (Diterima)
3. AB → CD (Diterima)
4. ABC → DE (Ditolak, karena jumlah simbol pada ruas sebelah kiri lebih bayak dari jumlah simbol pada ruas kanan)
5. Ab → cDe (Diterima)
6. bA → cd (Diterima)
7. a → b (Ditolak, karena simbol pada sebelah kiri harus minimal ada sebuah symbol variabel)

Tentukan apakah produksi-produksi berikut memenuhi aturan tata bahasa *context sensitive*.

1. B → bcdefG
2. A → aSa
3. A → aSS
4. A → BCDEF
5. Ad → dB
6. A → є
7. AB → є
8. ad → b
9. ad → є
10. abC → DE
11. abcDef → ghijkl
12. AB → cde
13. AAA → BBB

**Tata Bahasa *Unrestricted***

Aturan : Simbol pada sebelah kiri harus minimal ada sebuah simbol variabel

Contoh :

1. Abcdef → g (Diterima)
2. aBCdE → GHIJKL (Diterima)
3. abcdef → GHIJKL (Ditolak, karena simbol pada sebelah kiri tidak ada sebuah symbol variabel)

Tentukan apakah produksi-produksi berikut memenuhi aturan tata bahasa *unrestricted*.

1. A → є
2. AB → є
3. ad → b
4. ad → є
5. abC → DE
6. AB → cde
7. e → a
8. ABCDEFG → h
9. bA → CDEFGH

## 3.3. Grammar

Suatu kumpulan aturan (*production*) yang menentukan urut-urutan karakter. Suatu formal *grammar* adalah *grammar*biasa yang ditentukan dengan menggunakan notasi yang ketat. Dalam Bahasa Pemrograman ditentukan  2 jenis *Grammar*:

**1) Primary Grammar**

Disebut juga *phrase-structure grammar*, menspesifikasikan bagian utama pada kompilator dan interpreter yang disebut parser. *Grammar*ini menspesifikasikan bagaimana kata-kata dalam bahasa pemrograman dapat tergabung dan membentuk  program yang valid secara sintaks. Parsing adalah istilah pada bahasa yang menggambarkan proses analisis sebuah kalimat bahasa menurut *grammar*-nya.

**2) Secondary Grammar**

Umumnya digunakan untuk menspesifikasikan bentuk  yang benar, *spelling*dari kata-kata pada bahasa komputer à disebut dengan *grammar*leksikal. Bagian dari kompilator yang menganalisis kata-kata secara individu pada *input*program disebut *scanner*.

Ada dua kelas *grammar*yang berguna untuk teknologi *compiler*, yaitu:

**1) EBNF Grammar**

* EExtended Backus-Naus Form
* Metalanguage: Bahasa yang digunakan untuk mendeskripsikan bahasa lain.
* Menggunakan notasi matematis, ::=, <, >, |, \*, +, {, }, [, ] disebut Metasymbol.
* Suatu bahasa yang dideskripsikan dalam EBNF merupakan suatu kumpulan aturan.

**2) Regular Grammar**

* Regular Grammar  disebut juga Regular Language.
* Digunakan pada Teori Bahasa Otomata (Mesin abstrack yang menggunakan model matematika, tetapi menggunakan matematika yang benar-benar berbeda dengan matematika klasik dan kalkulus)
* Bahasa  pemrograman yang menggunakan aturan sintaktik bahasa regular ini antara lain: Javascript & Perl.

Dalam hierarki-bahasa  Chomsky, grammar regular adalah grammar paling restriktif yang dapat membangkitkan sebuah kalimat. Dalam hierarki tersebut, grammar regular mempunyai kemampuan pembangkitan kalimat yang sangat minimal karena:

1. Sisi kiri hanya boleh berisi sebuah nonterminal,
2. Sisi kanan dalam setiap aturan produksinya hanya boleh berisi satu nonterminal dan posisinya hanya boleh berada di akhir atau sisi kanan rangkaian.

Deskripsi yang lebih sederhana, parser untuk  grammar ini tidak dapat mengetahui konteks pemunculan nonterminal  yang tengah dideﬁnisikan, dan hanya dapat melihat  symbol terminal yang ada tepat didepannya.

* Terminal Symbols, Alphabet dan Strings
  + Alphabet : representasi letter yang ditulis dalam huruf kecil (a,b,c,…z)
  + Terminal Simbol (T) : individual karakter, termasuk di dalamnya adalah alphabet {a,b,….z, 0,1,…9}
  + Strings : urutan simbol terbatas (sequence finite symbol). contoh : aba, ab12z, axy operasi yang dapat dilakukan pada strings : substrings, concatenation.
* Non Terminal (NT) : kumpulan string alphabeth yang ada pada formal language.
  + Non Terminal direpresentasikan dalam huruf  besar (A,B,….) dan diapit dengan tanda <..> untuk membedakannya dengan string.
* Konsep T dan NT direpresentasikan dalam Produksi.
  + Bentuk Umum Produksi   NT ::= string T atau NT
* Derivasi & Reduksi
  + Derivasi merupakan uraian dari suatu Produksi.
  + Reduksi proses perubahan string ke dalam NT sehingga menjadi suatu *grammar production*.

**Klasifikasi Grammar**

Chomsky (1963) mengklasifikasikan Grammar menjadi 4 kategori :

1. **Grammar Type-0**

* Disebut Phrase Structure Grammar atau grammar tidak terbatas.
* Bentuknya : a ::= ß.
  + Dimana, a dan ß dapat berupa T atau NT, yang dapat saling bersubstitusi, karenanya tidak relevan untuk bahasa pemrograman.

1. **Grammar Type-1**

* ·Disebut Context Sensitive Grammar (CSG), produksi dari grammar ini adalah derivasi/reduksi dari particular  string yang hanya terdapat pada particular context
* Bentuknya :
  + a Aß ::= apß
  + Dimana, p menggantikan A untuk menutupi string a dan ß .
* Grammar ini-pun tidak relevan untuk bahasa pemrograman.

1. **Grammar Type-2**

* Disebut Context Free Grammar (CFG), grammar ini tidak membutuhkan context pengenal atau derivasi
  + Bentuknya : A ::= p
* Grammar ini digunakan pada ALGOL-60 dan PASCAL

1. **Grammar Type-**3

* Disebut Linier atau Regular Grammar, dimana dapat berupa single terminal simbol dan terminal  simbol atau kebalikannya
* Bentuknya :
  + A ::= t B | t  atau A ::= B t | t
* Bentuk ini banyak dijumpai pada bahasa pemrograman.

**CFG (Context Free Grammar)**

* CFG ditemukan untuk membantu menspesifikasikan bahasa manusia dan ternyata sangat cocok untuk mendefinisikan bahasa komputer, menyederhanakan penerjemahan bahasa komputer dan aplikasinya
* CFG digunakan untuk menspesifikasikan struktur sintaks bahasa pemrograman serta beragam basis data.
* CFG adalah sekumpulan berhingga variabel yang juga disebut nonterminal atau kategori sintaks, dimana masing-masing mempresentasikan bahasa. Bahasa-bahasa yang direpresentasikan dengan variabel-variabel itu yang dideskripsikan secara rekursif dalam bentukan lain dan symbol-simbol primitif disebut terminal. Aturan-aturan yang berhubungan dengan variabel-variabel itu disebut produksi.

## 3.4. Contoh Analisa Penentuan Type Grammar

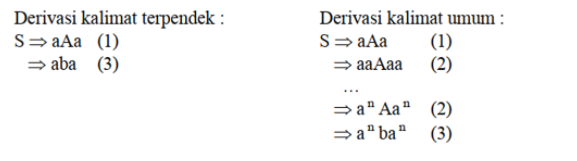
1. Grammar G1 dengan Q1 = {S → aB, B → bB, B → b}. Ruas kiri semua produksinya terdiri dari sebuah VN maka G1 kemungkinan tipe CFG atau RG. Selanjutnya karena semua ruas kanannya terdiri dari sebuah VT atau string VT VN maka G1 adalah RG.
2. Grammar G2 dengan Q2 = {S → Ba, B → Bb, B → b}. Ruas kiri semua produksinya terdiri dari sebuah VN maka G2 kemungkinan tipe CFG atau RG. Selanjutnya karena semua ruas kanannya terdiri dari sebuah VT atau string VN VT maka G2 adalah RG.
3. Grammar G3 dengan Q3 = {S → Ba, B → bB, B → b}. Ruas kiri semua produksinya terdiri dari sebuah VN maka G3 kemungkinan tipe CFG atau RG. Selanjutnya karena ruas kanannya mengandung string VT VN (yaitu bB) dan juga string VN VT (Ba) maka G3 bukan RG, dengan kata lain G3 adalah CFG.
4. Grammar G4 dengan Q4 = {S → aAb, B → aB}. Ruas kiri semua produksinya terdiri dari sebuah VN maka G4 kemungkinan tipe CFG atau RG. Selanjutnya karena ruas kanannya mengandung string yang panjangnya lebih dari 2 (yaitu aAb) maka G4 bukan RG, dengan kata lain G 4 adalah CFG.
5. Grammar G5 dengan Q5 = {S → aA, S → aB, aAb → aBCb}. Ruas kirinya mengandung string yang panjangnya lebih dari 1 (yaitu aAb) maka G5 kemungkinan tipe CSG atau UG. Selanjutnya karena semua ruas kirinya lebih pendek atau sama dengan ruas kananya maka G5 adalah CSG.
6. Grammar G6 dengan Q6 = {aS → ab, SAc → bc}. Ruas kirinya mengandung string yang panjangnya lebih dari 1 maka G6 kemungkinan tipe CSG atau UG. Selanjutnya karena terdapat ruas kirinya yang lebih panjang daripada ruas kananya (yaitu SAc) maka G6 adalah UG.

## 3.5. Derivasi Kalimat dan Penentuan Bahasa

Tentukan bahasa dari masing-masing grammar berikut :

* G1 dengan Q1 = {1. S → aAa, 2. A → aAa, 3. A → b}.

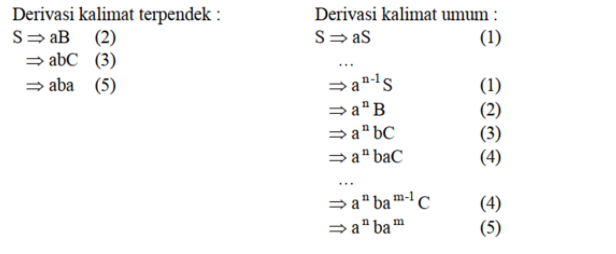
Jawab :



Dari pola kedua kalimat disimpulkan : L1 (G1 ) = { an ban | n ≥ 1}

* G2 dengan Q2 = {1. S → aS, 2. S → aB, 3. B → bC, 4. C → aC, 5. C → a}.

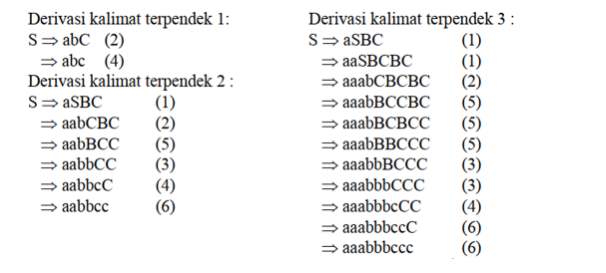
Jawab :



Dari pola kedua kalimat disimpulkan : L2 (G2 ) = { an bam | n ≥ 1, m ≥ 1}

* G3 dengan Q 3 = {1. S → aSBC, 2. S → abC, 3. bB → bb, 4. bC → bc, 5. CB → BC, 6. cC → cc}.

Jawab :



Dari pola ketiga kalimat disimpulkan : L3 (G3 ) = { an bn cn | n ≥ 1}

## 4.1. Definisi Finite State Automata (FSA)

***Finite State Automata* (FSA)** adalahmesin abstrak berupa sistem model matematika dengan masukan dan keluaran diskrit yang dapat mengenali bahasa paling sederhana (bahasa reguler / tipe 3).*Finite State Automata* berisi model matematika yang dapat menerima *input*dan mengeluarkan *output*. ***Finite Automata***adalah mesin komputasi, yang merupakan mesin abstrak bukan mesin fisik, namun memadai untuk diimplementasikan secara nyata. FSA memiliki *state*berhingga banyaknya dan dapat berpindah dari satu ke yang lainnya sesuai dengan *input*-an dan fungsi transisi.

**Contoh :**

Seorang petani dengan seekor serigala, kambing dan seikat rumput berada pada suatu sisi sungai. Tersedia hanya sebuah perahu kecil yang hanya dapat dimuati dengan petani tersebut dengan salah satu serigala, kambing atau rumput. Petani tersebut harus menyeberangkan ketiga bawaannya kesisi lain sungai. Tetapi, jika petani meninggalkan serigala dan kambing pada suatu saat, maka kambing akan dimakan serigala. Begitu pula jika kambing ditinggalkan dengan rumput, maka rumput akan dimakan oleh kambing. Mungkinkah ditemukan suatu cara untuk melintasi sungai tanpa menyebabkan kambing atau rumput dimakan?.

Contoh sistem dengan *state*berhingga :

* Sistem *elevator*
* Mesin penjual minuman kaleng (*vending machine*)
* Pengatur lampu lalu lintas
* *Circuit switching*di komputer dan telekomunikasi
* *Lexical Analyzer*
* *Neuron Nets*

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

Q = himpunan *state*/kedudukan  
Σ = himpunan simbol *input*/masukan/abjad  
δ = fungsi transisi  
S = *state*awal/kedudukan awal (*initial state*)  
F = himpunan *state* akhir

*Finite State Automata* yang memiliki tepat satu *state*berikutnya untuk setiap simbol masukan yang diterima disebut *Deterministic Finite Automata*.

**Karakteristik *Finite Automata***

1. Setiap *Finite Automata* memiliki keadaan dan transisi yang terbatas.
2. Transisi dari satu keadaan ke keadaan lainnya dapat bersifat deterministik atau non- deterministik.
3. Setiap *Finite Automata* selalu memiliki keadaan awal.
4. *Finite Automata* dapat memiliki lebih dari satu keadaan akhir. Jika setelah pemrosesan seluruh *string*, keadaan akhir dicapai, artinya otomata menerima *string*tersebut.

**Setiap FSA memiliki:**

**1. Himpunan berhingga (*finite*) status (*state*)**

* Satu buah status sebagai status awal (*initial state*), biasa dinyatakan q0.
* Beberapa buah status sebagai status akhir (*final state*).

**2. Himpunan berhingga simbol masukan**

**3. Fungsi transisi**

Menentukan status berikutnya dari setiap pasang status dan sebuah simbol masukan.

**Cara Kerja *Finite State Automata***

*Finite State Automata*bekerja dengan cara mesin membaca memori masukan berupa *tape*(pita) yaitu 1 karakter tiap saat (dari kiri ke kanan) menggunakan *head*baca yang dikendalikan oleh kotak kendali *state*berhingga dimana pada mesin terdapat sejumlah *state*berhingga.

*Finite Automata*selalu dalam kondisi yang disebut *state*awal (*initial state*) pada saat *Finite Automata*mulai membaca *tape*. Perubahan *state*terjadi pada mesin ketika sebuah karakter berikutnya dibaca. Ketika *head*telah sampai pada akhir *tape*dan kondisi yang ditemui adalah *state*akhir, maka *string* yang terdapat pada *tape*dikatakan diterima *Finite Automata* (*String-string* merupakan milik bahasa bila diterima *Finite Automata* bahasa tersebut).

***Finite State Diagram*(FSD)**

*Finite State Automata*dapat dimodelkan dengan ***Finite State Diagram*(FSD)**dapat juga disebut *State Transition Diagram*. Sistem transisi adalah sistem yang tingkah lakunya disajikan dalam bentuk keadaan-keadaan (*states*). Sistem tersebut dapat bergerak dari *state*yang satu ke *state*lainnya sesuai dengan *input*yang diberikan padanya. Fungsi Transisi (δ) adalah representasi matematis atas transisi keadaan.

S = himpunan alfabet.  
Q = himpunan keadaan-keadaan.  
δ = Q x S →→ Q

*Finite State Diagram*terdiri dari:

**1. Lingkaran menyatakan *state***

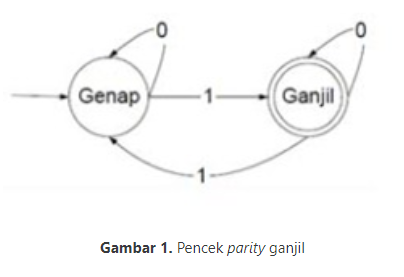
Lingkaran diberi label sesuai dengan nama *state*tersebut. Adapun pembagian lingkaran adalah:

* Lingkaran bergaris tunggal berarti *state*sementara
* Lingkaran bergaris ganda berarti *state*akhir

**2. Anak Panah menyatakan transisi yang terjadi.**

Label di anak panah menyatakan simbol yang membuat transisi dari 1 *state*ke *state*lain. 1 anak panah diberi label *start*untuk menyatakan awal mula transisi dilakukan.

**Contoh FSA**: Pencek *parity*ganjil

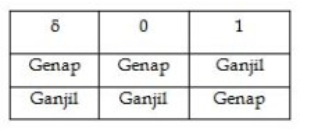


Misalnya *input*: 1101

* Genap 1 Ganjil 1 Genap 0 Genap 1 Ganjil : diterima mesin

Misalnya *input*: 1100

* Genap 1 Ganjil 1 Genap 0 Genap 0 Genap : ditolak mesin



Dari contoh diatas, maka:

Q = {Genap, Ganjil}  
Σ = {0,1}  
S = Genap  
F = {Ganjil }

atau

δ(Genap,0) = Genap  
δ(Genap,1) = Ganjil  
δ(Ganjil,0) = Ganjil  
δ(Ganjil,1) = Genap

Sebuah FSA dibentuk dari lingkaran yang menyatakan *state*:

* Label pada lingkaran adalah nama *state*
* Busur menyatakan transisi/perpindahan
* Label pada busur yaitu symbol *input*
* Lingkaran yang didahului sebuah busur tanpa label menyatakan *state*awal
* Lingkaran ganda menyatakan *state*akhir/*final*.

Jadi : sebuah mesin otomata dapat dinyatakan dalam diagram transisi, fungsi transisi dan tabel transisi.

## 4.2. Aturan Produksi Bahasa Regular

Sebuah otomata berhingga menspesifikasikan sebuah bahasa sebagai himpunan semua untai yang menggerakkannya dari *state*awal ke salah satu dari *state*yang diterimanya (himpunan *state*akhir). Otomata berhingga pada gambar 3 menerima ekspresi regular:

a(a\* U b\*)b

Selain dengan ekspresi regular, kita dapat mengkonstruksi aturan-aturan produksi untuk suatu tata bahasa regular. Kita ingat juga batasan aturan produksi untuk bahasa regular:

a →→ b

a adalah sebuah symbol variabel

b maksimal memiliki sebuah symbol variabel yang terletak di paling kanan bila ada. (bisa dibaca : a menghasilkan b), dimana a atau bisa berupa symbol terminal atau symbol non-terminal/variabel.

Simbol variabel/non-terminal adalah symbol yang masih bisa diturunkan, sedang symbol terminal sudah tidak bisa diturunkan lagi. Simbol terminal biasanya dinyatakan dengan huruf kecil, misal : a,b,c. Simbol nonterminal/variabel biasanya dinyatakan dengan huruf besar, missal A,B,C. Suatu tata bahasa (*grammar*) didefinisikan dengan 4 tupel (G={V,T,P,S}), dimana:

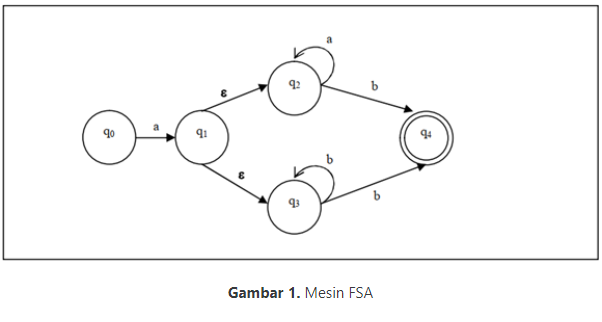
V = himpunan symbol variabel/non terminal

T = himpunan symbol terminal

P = kumpulan aturan produksi

S = symbol awal

Selanjutnya akan kita lihat bagaimana membuat kumpulan aturan produksi untuk suatu *finite state automata*.



**Gambar 1.** Mesin FSA

## 4.3. Mengkonstruksi Aturan Produksi dari Suatu Finite State Automata

Dalam mengkonstruksi aturan produksi tata bahasa regular dari suatu *finite state automata*, perlu kita ingat yang menjadi perhatian kita adalah *state-state* yang bisa menuju ke *state*akhir. Mesin *finite state automata* di gambar 3 memiliki *input*‘a’ dan ‘b’. Simbol ‘a’ dan ‘b’akan menjadi symbol terminal pada aturan produksi yang akan kita bentuk. Misalnya kita tentukan symbol awal adalah S. Kita identikkan S dengan *state*awal q0. Dari q0 mendapat *input*a menjadi q1. Bisa kita tuliskan sebagai aturan produksi:

S →→ aE

Dimana E kita identikkan dengan q1 (sebenarnya lebih tepatnya adalah bagian yang belum terbangkitkan mulai dari *state*q1). Kita bisa menambahkan symbol variabel baru setiap kali kita perlukan. Dari q1 mendapat transisi ε (tanpa menerima input) ke q2 dan q3. Kita tuliskan:

E →→ A

E →→ B

Bila kita identikkan q2 sebagai A, dan q3 sebagai B. Dari q2 mendapat input a tetap ke q2, dan dari q3 mendapat input b tetap ke q3, bisa kita tuliskan:

A →→ aA

B →→ bB

Selanjutnya kita lihat dari q2 mendapat input b ke q4, dan dari q3 mendapat input b ke q4, sementara q4 state akhir dan dari q4 tidak ada lagi busur keluar, maka bisa kita tuliskan:

A →→ b

B →→ b

Kumpulan aturan produksi yang kita peroleh bisa kita tuliskan sebagai berikut:

S →→ aE

E →→ A | B

A →→ aA | B

B →→ bB | b

Ingat ‘|’ berarti atau. Secara formal tata bahasa yang diperoleh dari otomata pada gambar 1:

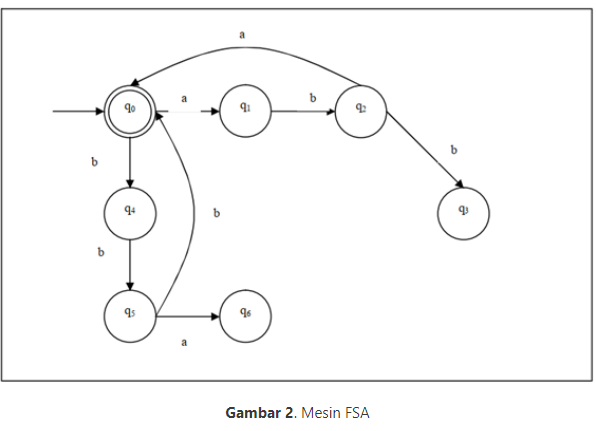
V= {S,E,A,B}

T = {a,b}

P = {S →→ aE, E →→ A | B, A →→ aA | B, B →→ bB | b}

S = S

Kita lihat contoh lain pada gambar 2



**Gambar 2**. Mesin FSA

Kita bisa mengkonstruksi aturan produksi untuk otomata tersebut:

T = {a,b}

S = S

Kumpulan aturan produksinya kita buat sebagai berikut:

S →→ aA | bB

(identikkan S untuk q0, A untuk q1, B untuk q4)

A →→ bC

(identikkan C untuk q2)

C →→ aS

(dari q2 mendapat input a ke q0)

(dari q3 tidak ada transisi keluar dan bukan *state*akhir maka transisi ke q3 kita abaikan)

B →→ bD

(identikkan D untk q5)

D →→ bS

(dari q5 mendapat *input*b ke q0)

(dari q6 tidak ada transisi keluar dan bukan *state*akhir maka transisi ke q6 kita abaikan)

Pada kasus ini kita lihat q0 sebagai *state*akhir masih memiliki transisi keluar, maka untuk menandakannya sebagai *state*akhir kita buat:

S →→ ε

\*Bedakan dengan kasus gambar 1 dimana *state*akhir q4 tidak memiliki transisi keluar. Maka diperoleh:

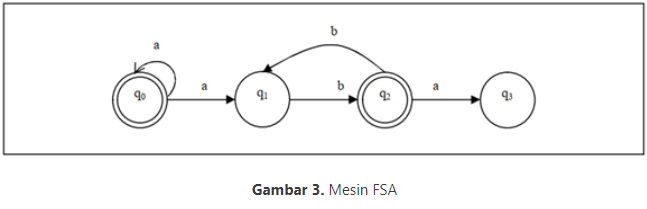
V = {S,A,B,C,D}

P = {S →→ aA|bB|ε, A →→ BC, B →→ bD, C →→ aS, D →→ bS}

Sebenarnya aturan produksi diatas masih dapat disederhanakan menjadi:

P = { S →→ aA|bB|ε, A →→ BaS, B →→ bbS}

Sehingga mereduksi jumlah symbol variabel yang diperlukan:



**Gambar 3.**Mesin FSA

Contoh lain, dari gambar 3 kita bisa membuat aturan produksinya sebagai berikut:

S →→ aA | aS

A →→ bB

B →→ bA | ε

## 4.4. Finite State Automata untuk Suatu Tata Bahasa Regular

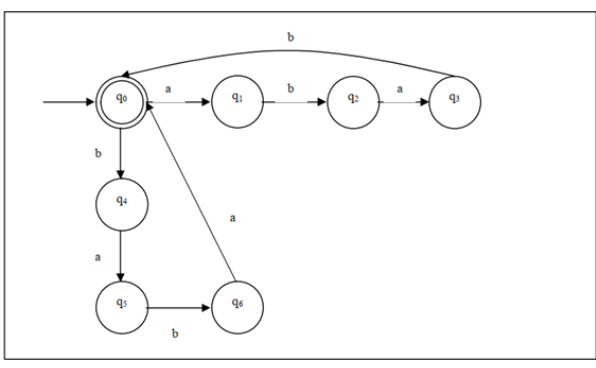
Bila sebelumnya dari suatu diagram transisi *finite state automata*kita bisa membuat aturan-aturan produksinya.

Misalkan terdapat tata bahasa regular dengan aturan produksi:

S →→ aB| bA |ε

A →→ abaS

B →→ babS



**Gambar 4.***Finite Otomata dari suatu regular grammar*

Bisa Anda lihat hasilnya pada gambar 4. S akan berkorelasi dengan q0, A dengan q4, dan B dengan q1. Sementara S →→ ε menandakan q0 termasuk state akhir.

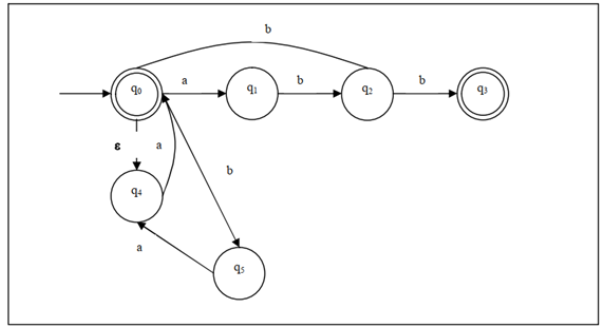
Contoh lain akan dibuat diagram transisi untuk tata bahasa regular:

S →→ abA | B |baB|ε

A →→ bS | b

B →→ aS

Hasilnya bisa kita lihat pada gambar 5. Kita lihat S akan berkorelasi dengan q0, A dengan q2, B dengan q4. Sementara S →→ ε menandakan qo *state*akhir. Dari S →→ B, maka kita bisa buat transisi ε dari q0 ke q4. Dari A →→ kita tentukan state q3 termasuk *state*akhir.



**Gambar 5***. Finite otomata dari suatu regular grammar*

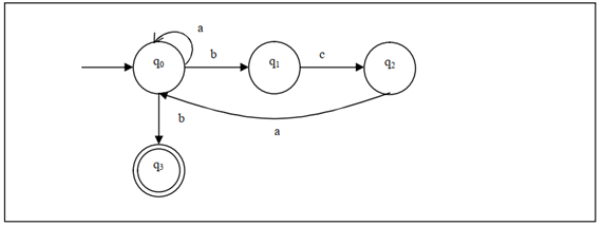
Contoh lain, akan dibuat diagram transisi untuk tata bahasa regular.

S →→ aS | bB |b

B →→ cC

C →→ aS

Bisa anda lihat hasilnya pada gambar 6 akan berkorelasi dengan q0, B dengan q1, C dengan q2. Kita lihat S →→ b, maka kita buat state akhir q3.



**Gambar 6.** Finite Otomata dari suatu regular grammar

## 4.5. Mesin Moore

Suatu keterbatasan dari *finite state automata*yang sudah kita pelajari selama ini keputusannya terbatas pada diterima atau ditolak. otomata tersebut biasa disebut sebagai *accepter*, dalam hal ini *finite state accepter*. Kita bisa mengkonstruksi sebuah *finite state automata*yang memiliki keputusan beberapa keluaran/*output*, dalam hal ini otomata tersebut akan dikenal sebagai *transducer*. Pada mesin Moore, *output*akan berasosiasi dengan *state*. Mesin Moore didefinisikan dalam 6 (enam) tupel, M = (Q, ∑, δ, S, Δ, λ), dimana:

Q = himpunan state

∑ = himpunan symbol input

δ = fungsi transisi

S = state awal, S ∈∈ Q

Δ = himpunan output

λ = fungsi output untuk setiap state

\*Perhatikan: komponen *state Final*dari *Deterministic Finite Automata*dihilangkan, karena disini keputusan dimunculkan sebagai *output*. Kita lihat contoh penerapan dari Mesin Moore. Misal kita ingin memperoleh sisa pembagian (modulus) suatu bilangan dengan 3. Dimana *input*dinyatakan dalam biner. Mesin Moore yang bersesuaian bisa dilihat pada gambar 1. Konfigurasi mesin sebagai berikut:

Q = {q0,q1,q2}

∑ = {0,1} (input dalam biner)

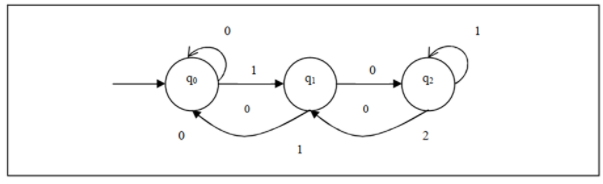
Δ = {0,1,2} (untuk *output*-nya pada kasus mod dengan 3 maka sisanya kemungkinan adalah (0,1,2)

S = q0

λ (q0) = 0

λ (q1) =1

λ (q2) =2



**Gambar 1**. Mesin Moore untuk modulus 3

Misalkan saja

* *5 mod 3*= ?

*input*5 dalam biner 101

bila kita masukkan 101 ke dalam mesin, urutan *state*yang dicapai: q0,q1,q2,q3

Perhatikan *state*terakhir yang dicapai adalah q2, λ (q2) =2, maka *5 mod 3*= 2

* *10 mod 3*=?

*input*10 dalam biner 1010

bila kita masukkan 1010 ke dalam mesin, urutan *state*yang dicapai: q0,q1,q2,q2, q1

λ (q1) =1, maka *10 mod 3*= 1

## 4.6. Mesin Mealy

Bila output pada mesin Moore berasosiasi dengan state, maka output pada Mesin Mealy akan berasosiasi dengan transisi. Mesin Mealy sendiri didefinisikan dalam 6 tupel, M = (Q, ∑, δ, S, Δ, λ), dimana:

Q = himpunan state

∑ = himpunan symbol input

δ = fungsi transisi

S = state awal, S ∈∈ Q

Δ = himpunan output

λ = fungsi output untuk setiap transisi

Contoh penerapan Mesin Mealy kita lihat pada gambar2.

Mesin itu akan mengeluarkan output apakah menerima (Y) atau menolak (T), suatu masukan. Dimana mesin akan mengeluarkan output ‘Y’ bila menerima untai yang memiliki akhiran 2 simbol berturutan yang sama, atau secara formal dalam ekspresi regular:

(0+1)\*(00+11)

Contoh input yang diterima :

01011, 01100, 1010100, 10110100, 00, 11, 100, 011, 000, 111

Konfigurasi dari Mesin Mealy tersebut:

Q = {q0,q1,q2}

∑ = {0,1}

Δ = {Y,T}

S = q0

λ (q0,0) = T

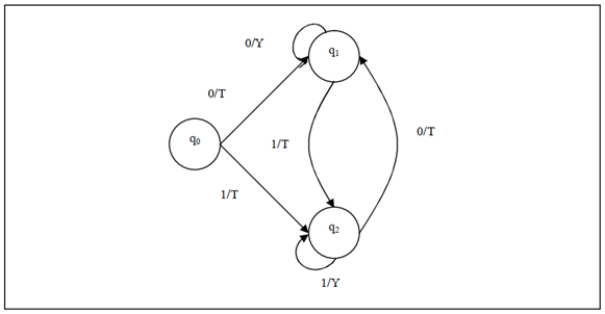
λ (q0,1) = T

λ (q1,0) = Y

λ (q1,1) = T

λ (q2,0) = T

λ (q2,1) =Y



**Gambar 2.** Mesin Mealy

## 4.7. Ekivalensi Mesin Moore dan Mesin Mealy

Dari suatu Mesin Moore dapat dibuat Mesin Mealy yang ekivalen, begitu juga sebaliknya. Untuk mesin Mealy pada gambar 2 dapat kita buat Mesin Moore yang ekivalen yaitu gambar 3. Bisa kita lihat state pada mesin Moore dibentuk dari kombinasi state pada Mealy dan banyaknya output. Karena jumlah state Mealy = 3, dan jumlah output = 2, maka jumlah state pada Moore yang ekivalen = 6.

Bisa dilihat konfigurasi Mesin Moore yang dibentuk:

Q = {q0Y, q0T, q1Y,q1T, q2Y,q2T}

∑ = {0,1}

Δ = {Y,T}

S = q0

λ (q0Y) = Y

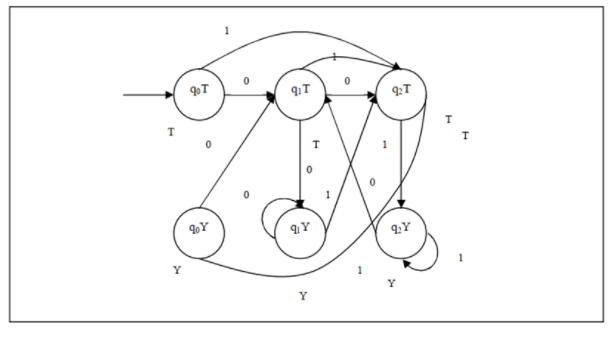
λ (q0T) = T

λ (q1Y) = Y

λ (q1T) = T

λ (q2Y) = Y

λ (q2T) = T



**Gambar 3.**Mesin Moore yang ekivalen dengan gambar 2

Bila kita perhatikan dari gambar diatas, state q0Y dapat dihapus karena tidak ada busur yang mengarah ke state tersebut. Untuk memperoleh ekivalensi mesin Mealy dari suatu mesin Moore caranya lebih mudah, cukup dengan menambahkan label output ke setiap transisi dan menghapus label output pada setiap state. Kita lihat gambar 4 merupakan mesin Mealy yang ekivalen dengan mesin Moore pada gambar 1. Konfigurasi Mesin Mealy tersebut:

Q = {q0, q1, q2}

∑ = {0,1}

Δ = {0,1,2}

S = q0

λ (q0,0) = 0

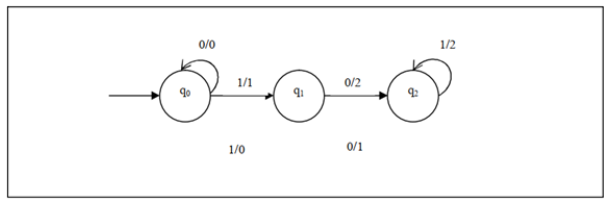
λ (q0,1) = 1

λ (q1,0) = 2

λ (q1,1) = 0

λ (q2,0) = 1

λ (q2,1) =2



**Gambar 4***. Mesin Mealy yang ekivalen dengan gambar1*

## 5.1. Pendahuluan

**Definisi**

Otomata Hingga (*Finite State Automata*) adalah :

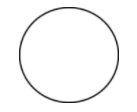
1. Model matematika yang dapat menerima input dan mengeluarkan output
2. Memiliki state yang berhingga banyaknya dan dapat berpindah dari satu state ke state lainnya berdasar input dan fungsi transisi
3. Tidak memiliki tempat penyimpanan/memory, hanya bisa mengingat state terkini
4. Mekanisme kerja dapat diaplikasikan pada elevator, text editor, analisa leksikal, pencek parity.

Otomata Hingga dinyatakan oleh 5-tupel atau M = (**Q, ∑, δ, S, F**)

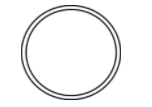
**Q** = himpunan kedudukan (*state*)  
**∑** = alfabet / himpunasn simbol *input*  
**δ**= fungsi transisi =**Q** x **∑**  
**S** = kedudukan (state) awal, **S** ∈∈ **Q**

**F** = kedudukan (state) akhir, F ⊆⊆** Q**

S dilambangkan dengan :

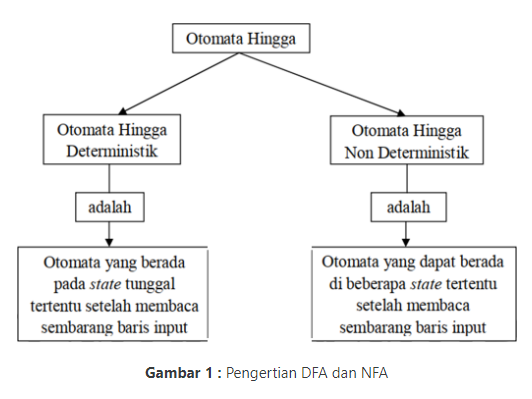


F dilambangkan dengan :



Setiap otomaton:

* mempunyai tepat satu S
* mempunyai satu F atau lebih



## 5.2. Pengertian DFA

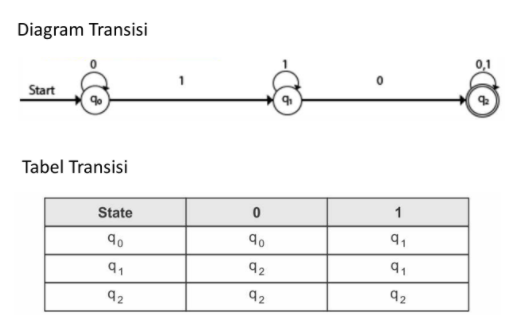
**Deterministic Finite Automata (DFA)** menerima masukan (input) yang hanya memiliki 1 busur keluar. Deterministic Finite Automata (DFA) sering dikenal juga sebagai Deterministic Finite-State Machine (DFSM) dan Deterministic Finite-State Automaton (DFSA). DFA diperkenalkan oleh **Warren McCulloch** dan **Walter Pitts** sebagai peneliti pertama yang memperkenalkan konsep yang mirip dengan finite automata di tahun 1943.

DFA sendiri merupakan finite automata dengan memiliki 5 tuple yang direpresentasikan sebagai berikut:

**Q** = himpunan state, contohnya {q0, q1, q2}  
**Σ** = input alphabet, contohnya {a, b}  
**δ** = fungsi transisi  
**q0** = state awal  
**F** = state akhir

**Contoh DFA**

Q = {q0, q1, q2}  
∑ = {0, 1}  
q0 = {q0}  
F = {q2}



**Gambar 2 :** Contoh DFA

## 5.3. Pengertian NFA

**Non-Deterministic Finite Automata (NFA)** menerima masukan (input) dengan memiliki lebih dari 1 busur keluar atau bahkan tidak memiliki busur keluar. Non-Deterministic Finite Automata (NFA) sering dikenal juga sebagai Non-Deterministic Finite-State Machine (NFSM) dan Non-Deterministic Finite-State Automaton (NFSA). NFA diperkenalkan pada tahun 1959 oleh **Michael O. Rabin** dan **Dana Scott**.

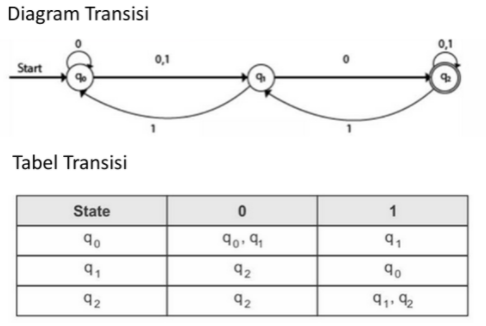
NFA sendiri merupakan finite automata dengan memiliki 5 tuple yang direpresentasikan sebagai berikut:

**Q =** himpunan state, contohnya {q0, q1, q2}  
**Σ** = input alphabet, contohnya {a, b}  
**δ** = fungsi transisi  
**q0=** state awal  
**F** = state akhir

**Contoh NFA**

Q = {q0, q1, q2}  
∑ = {0, 1}  
q0 = {q0}

F = {q2}



**Gambar 3 :** Contoh NFA

## 5.4. Perbedaan DFA dan NFA

Pada intinya, paling mendasar perbedaan DFA dan NFA, jika DFA apabila state diberi input, maka state akan selalu tepat menuju 1 state. Berbeda dengan NFA, jika state diberi input, mungkin saja bisa menuju ke beberapa state selanjutnya. Sementara itu, berikut ini beberapa perbedaan DFA dan NFA yang disajikan dalam tabel:

**DFA**

* DFA tidak dapat menggunakan transisi string kosong (empty string)
* DFA dipahami sebagai sebuah mesin
* DFA untuk state selanjutnya bisa ditetapkan dengan jelas
* DFA lebih sulit dibuat
* Waktu yang dibutuhkan untuk mengeksekusi string input lebih sedikit
* Semua DFA merupakan NFA
* DFA membutuhkan lebih banyak ruang (space)

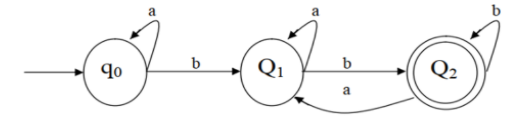
**NFA**

* NFA dapat menggunakan transisi string kosong (empty string)
* NFA dipahami sebagai beberapa mesin kecil yang melakukan komputasi di waktu bersamaan
* NFA untuk state selanjutnya mempunyai banyak kemungkinan
* NFA lebih mudah dibuat
* Waktu yang dibutuhkan untuk mengeksekusi string input lebih banyak
* Tidak semua NFA adalah DFA
* NFA membutuhkan lebih sedikit ruang (space)

## 5.5. Contoh Sederhana

Pada otomata berikut, tentukan apakah string‘abb’, dan ‘baba’ berada dalam L(M).

**Penyelesaian:**



δ(q0 , abb) = δ(q0 , bb) = δ(q1 , b) = q2  
Karena q2 adalah state akhir maka ‘abb’ berada dalam L(M)

δ(q0 , baba) = δ(q1 , aba) = δ(q1 , ba) = δ(q2 , a) = q1  
Karena q1 bukan state akhir maka ‘baba’ tidak berada dalam L(M)

## 6.1. Pengertian FSA

***Finite State Automata* (FSA)** adalahmesin abstrak berupa sistem model matematika dengan masukan dan keluaran diskrit yang dapat mengenali bahasa paling sederhana (bahasa reguler/tipe 3).*Finite State Automata* berisi model matematika yang dapat menerima *input*dan mengeluarkan *output*. ***Finite Automata***adalah mesin komputasi, yang merupakan mesin abstrak bukan mesin fisik, namun memadai untuk diimplementasikan secara nyata. FSA memiliki *state*berhingga banyaknya dan dapat berpindah dari satu ke yang lainnya sesuai dengan *input*-an dan fungsi transisi.

**Contoh :**

Seorang petani dengan seekor serigala, kambing dan seikat rumput berada pada suatu sisi sungai. Tersedia hanya sebuah perahu kecil yang hanya dapat dimuati dengan petani tersebut dengan salah satu serigala, kambing atau rumput. Petani tersebut harus menyeberangkan ketiga bawaannya kesisi lain sungai. Tetapi, jika petani meninggalkan serigala dan kambing pada suatu saat, maka kambing akan dimakan serigala. Begitu pula jika kambing ditinggalkan dengan rumput, maka rumput akan dimakan oleh kambing. Mungkinkah ditemukan suatu cara untuk melintasi sungai tanpa menyebabkan kambing atau rumput dimakan?.

Contoh sistem dengan *state*berhingga :

* Sistem *elevator*
* Mesin penjual minuman kaleng (*vending machine*)
* Pengatur lampu lalu lintas
* *Circuit switching*di komputer dan telekomunikasi
* *Lexical Analyzer*
* *Neuron Nets*

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

Q = himpunan *state*/kedudukan  
Σ = himpunan simbol *input*/masukan/abjad  
δ = fungsi transisi  
S = *state*awal/kedudukan awal (*initial state*)  
F = himpunan *state* akhir

*Finite State Automata* yang memiliki tepat satu *state*berikutnya untuk setiap simbol masukan yang diterima disebut *Deterministic Finite Automata*.

**Karakteristik *Finite Automata***

1. Setiap *Finite Automata* memiliki keadaan dan transisi yang terbatas.
2. Transisi dari satu keadaan ke keadaan lainnya dapat bersifat deterministik atau non- deterministik.
3. Setiap *Finite Automata* selalu memiliki keadaan awal.
4. *Finite Automata* dapat memiliki lebih dari satu keadaan akhir. Jika setelah pemrosesan seluruh *string*, keadaan akhir dicapai, artinya otomata menerima *string*tersebut.

**Setiap FSA memiliki:**

**1. Himpunan berhingga (*finite*) status (*state*)**

* Satu buah status sebagai status awal (*initial state*), biasa dinyatakan q0.
* Beberapa buah status sebagai status akhir (*final state*).

**2. Himpunan berhingga simbol masukan**

**3. Fungsi transisi**

Menentukan status berikutnya dari setiap pasang status dan sebuah simbol masukan.

**Cara Kerja *Finite State Automata***

*Finite State Automata*bekerja dengan cara mesin membaca memori masukan berupa *tape*(pita) yaitu 1 karakter tiap saat (dari kiri ke kanan) menggunakan *head*baca yang dikendalikan oleh kotak kendali *state*berhingga dimana pada mesin terdapat sejumlah *state*berhingga.

*Finite Automata*selalu dalam kondisi yang disebut *state*awal (*initial state*) pada saat *Finite Automata*mulai membaca *tape*. Perubahan *state*terjadi pada mesin ketika sebuah karakter berikutnya dibaca. Ketika *head*telah sampai pada akhir *tape*dan kondisi yang ditemui adalah *state*akhir, maka *string* yang terdapat pada *tape*dikatakan diterima *Finite Automata* (*String-string* merupakan milik bahasa bila diterima *Finite Automata* bahasa tersebut).

***Finite State Diagram*(FSD)**

*Finite State Automata*dapat dimodelkan dengan ***Finite State Diagram*(FSD)**dapat juga disebut *State Transition Diagram*. Sistem transisi adalah sistem yang tingkah lakunya disajikan dalam bentuk keadaan-keadaan (*states*). Sistem tersebut dapat bergerak dari *state*yang satu ke *state*lainnya sesuai dengan *input*yang diberikan padanya. Fungsi Transisi (δ) adalah representasi matematis atas transisi keadaan.

S = himpunan alfabet.  
Q = himpunan keadaan-keadaan.  
δ = Q x S →→ Q

*Finite State Diagram*terdiri dari:

**1. Lingkaran menyatakan *state***

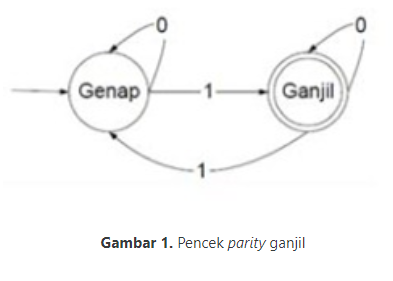
Lingkaran diberi label sesuai dengan nama *state*tersebut. Adapun pembagian lingkaran adalah:

* Lingkaran bergaris tunggal berarti *state*sementara
* Lingkaran bergaris ganda berarti *state*akhir

**2. Anak Panah menyatakan transisi yang terjadi.**

Label di anak panah menyatakan simbol yang membuat transisi dari 1 *state*ke *state*lain. 1 anak panah diberi label *start*untuk menyatakan awal mula transisi dilakukan.

**Contoh FSA**: Pencek *parity*ganjil



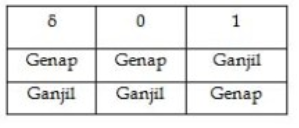
**Gambar 1.** Pencek *parity*ganjil

Misalnya *input*: 1101

* Genap 1 Ganjil 1 Genap 0 Genap 1 Ganjil : diterima mesin

Misalnya *input*: 1100

* Genap 1 Ganjil 1 Genap 0 Genap 0 Genap : ditolak mesin



Dari contoh diatas, maka:

Q = {Genap, Ganjil}  
Σ = {0,1}  
S = Genap  
F = {Ganjil }

atau

δ(Genap,0) = Genap  
δ(Genap,1) = Ganjil  
δ(Ganjil,0) = Ganjil  
δ(Ganjil,1) = Genap

Sebuah FSA dibentuk dari lingkaran yang menyatakan *state*:

* Label pada lingkaran adalah nama *state*
* Busur menyatakan transisi/perpindahan
* Label pada busur yaitu symbol *input*
* Lingkaran yang didahului sebuah busur tanpa label menyatakan *state*awal
* Lingkaran ganda menyatakan *state*akhir/*final*.

Jadi : sebuah mesin otomata dapat dinyatakan dalam diagram transisi, fungsi transisi dan tabel transisi.

## 6.2. Pengertian DFA

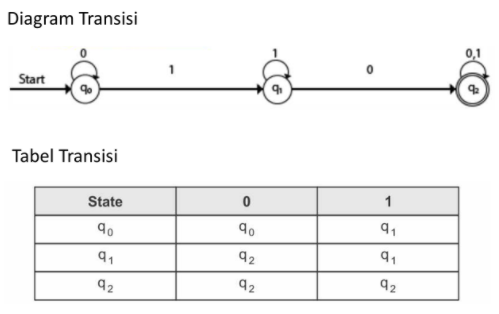
**Deterministic Finite Automata (DFA)** jenis FSA yang menerima masukan (input) yang hanya memiliki 1 busur keluar. Deterministic Finite Automata (DFA) sering dikenal juga sebagai Deterministic Finite-State Machine (DFSM) dan Deterministic Finite-State Automaton (DFSA). DFA diperkenalkan oleh **Warren McCulloch** dan **Walter Pitts** sebagai peneliti pertama yang memperkenalkan konsep yang mirip dengan finite automata di tahun 1943.

DFA sendiri merupakan finite automata dengan memiliki 5 tuple yang direpresentasikan sebagai berikut:

**Q** = himpunan state, contohnya {q0, q1, q2}  
**Σ** = input alphabet, contohnya {a, b}  
**δ** = fungsi transisi  
**q0** = state awal  
**F** = state akhir

**Contoh DFA :**

Q = {q0, q1, q2}  
∑ = {0, 1}  
q0 = {q0}  
F = {q2}



**Gambar 2 :** Contoh DFA

## 6.3. Pengertian NFA

**Non-Deterministic Finite Automata (NFA)** merupakan jenis FSA yang menerima masukan (input) dengan memiliki lebih dari 1 busur keluar atau bahkan tidak memiliki busur keluar. Non-Deterministic Finite Automata (NFA) sering dikenal juga sebagai Non-Deterministic Finite-State Machine (NFSM) dan Non-Deterministic Finite-State Automaton (NFSA). NFA diperkenalkan pada tahun 1959 oleh **Michael O. Rabin** dan **Dana Scott**.

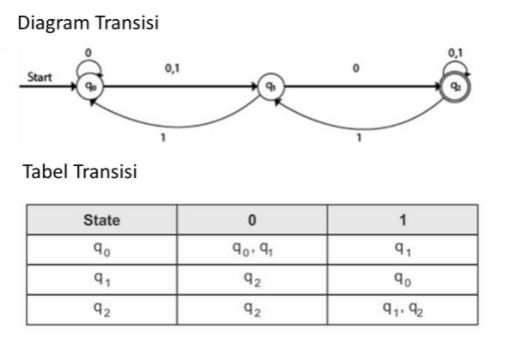
NFA sendiri merupakan finite automata dengan memiliki 5 tuple yang direpresentasikan sebagai berikut:

**Q =** himpunan state, contohnya {q0, q1, q2}  
**Σ** = input alphabet, contohnya {a, b}  
**δ** = fungsi transisi  
**q0=** state awal  
**F** = state akhir

**Contoh NFA :**

Q = {q0, q1, q2}  
∑ = {0, 1}  
q0 = {q0}

F = {q2}

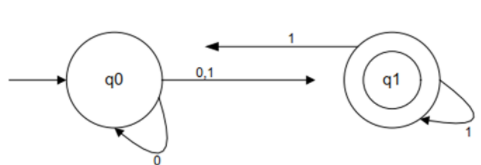


**Gambar 3 :** Contoh NFA

## 6.4. Ekivalen NFA ke DFA

Dari sebuah NFA dapat dibuat bentuk DFA-nya yang ekivalen (bersesuaian). Ekivalen disini artinya mampu memproduksi atau menerima bahasa yang sama. Adapun tahap pembuatan DFA yang ekivalen dari suatu NFA adalah sebagai berikut:

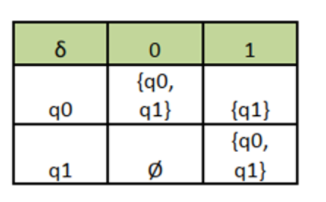
Contoh Diketahui NFA sebagai berikut.



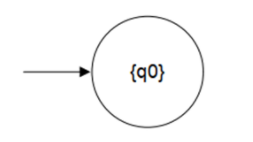
Konfigurasi NFA contoh di atas secara formal adalah sebagai berikut :

* Q = {q0, q1}
* Σ = {0, 1}
* S = q0
* F = {q1}

Tabel transisinya :



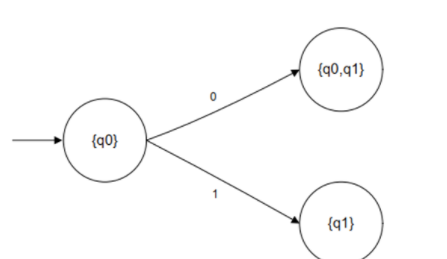
Kita mulai dengan state {q0}



Telusuri state berikutnya :

* δ (q0, 0) = {q0, q1}
* δ (q0, 1) = {q1}

Hasilnya :



Selanjutnya telusuri untuk setiap state baru yang terbentuk :

* δ (q1, 0) = Ø
* δ (q1, 1) = {q0, q1}
* δ ({q0,q1}, 0) = {q0, q1} adalah hasil gabungan dari

    δ (q0, 0) = {q0, q1}dengan δ (q1, 0) = Ø

* δ ({q0,q1}, 1) = {q0, q1} adalah hasil gabungan dari

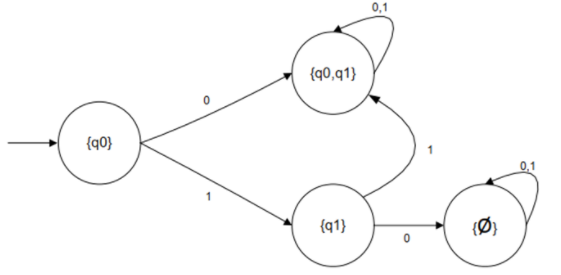
    δ (q0, 1) = {q1}dengan δ (q1, 0) = {q0, q1}

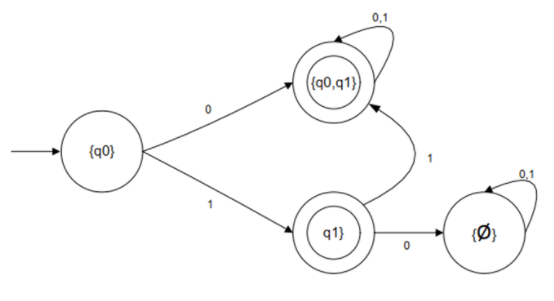
Hasilnya :

Selanjutnya telusuri state baru yang terbentuk :

* δ (Ø, 0) = Ø
* δ (Ø, 0) = Ø

Hasilnya :

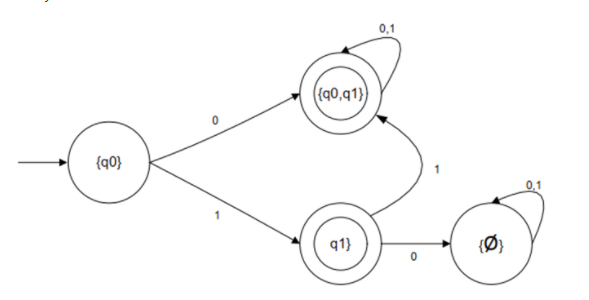




Selanjutnya kita ingat bahwa F = {q1}, maka himpunan state akhir (F) sekarang adalah semua yang mengandung state q1.

F = {{q1}, {q0, q1}}

Hasilnya :



## 6.5. Ekivalen DFA dan FSA

Ekivalensi DFA dan NFA :

* Suatu DFA dapat dipandang sebagai kasus khusus (subset) dari NFA.
* Jelas bahwa kelas bahasa yang diterima oleh DFA juga akan diterima oleh DFA
* Namun ternyata DFA juga dapat mensimulasikan NFA; yaitu untuk setiap NFA kita dapat membuat DFA yang ekivalen.
* Dapat dibuktikan bahwa DFA dan NFA adalah ekivalen, sehingga dapat disebut FA saja.

Simulasi NFA oleh DFA :

* Cara simulasi NFA oleh DFA adalah dengan membuat state DFA berkorespondensi dengan set state di NFA
* DFA yang dibentuk mencatat semua state yang mungkin pada NFA setelah membaca input tertentu

**Pembuktian :**

* Teorema: Jika L adalah himpunan yang diterima oleh NFA maka ada sebuah DFA yang menerima L
* Misalnya sebuah NFA M = (Q, Σ, q0, δ, A) dan DFA M’ = (Q’, Σ’, q0’, δ’, A’). State pada M’ adalah semua subhimpunan dari himpunan state M, yaitu Q’ = 2Q. M’ akan mencatat dalam statenya semua state M yang mungkin pada waktu tertentu.
* F’ adalah himpunan semua state di Q’ yang mengandung final state dari M.
* Elemen Q’ akan dinyatakan sebagai [q1, q2, …, qi] dimana q1, q2, …, qi ada di Q – [q1, q2, …, qi] adakah satu state dalam DFA yang berkorespondensi dengan suatu himpunan state di NFA – q0’ = [q0]
* δ’([q1, q2, …., qi], a) = [p1, p2, …, pj] jika dan hanya jika δ’({q1, q2, …., qi}, a) = {p1,p2, …, pj}
* Aplikasi δ’ terhadap elemen [q1, q2, …, qi] dari Q’ dihitung dengan mengaplikasikan δ terhadap setiap q1, q2, …, qi dan membuat gabungannya (unionnya), Gabungan tersebut digunakan untuk membuat set state baru p1, p2…pj. Himpunan baru ini memiliki representasi [p1, p2, …, pj] di Q’, dan elemen tersebut adalah nilai dari δ’([q1, q2, …., qi], a)
* Dengan menggunakan induksi, dapat dibuktikan bahwa: δ’(q0’, x) = [q1, q2, …, qi] jika dan hanya jika δ(q0, x) = {q1, q2, …, qi}. Jadi, dapat dibuat sebuah mesin DFA yang menerima bahasa yang sama dengan yang diterima oleh sebuah mesin NFA.
* Contoh : permainan catur, banyak alternatif pada suatu posisi tertentu -> nondeterministic
* Non deterministik dapat menyelesaikan problem tanpa backtrack, namun dapat diekuivalensikan ke DFA.

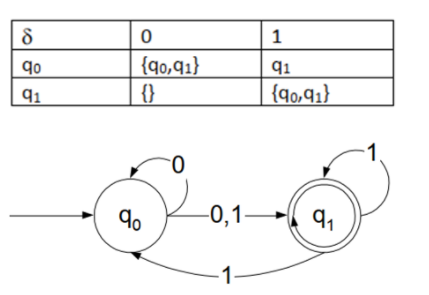
## 6.6. Contoh Permasalahan

**Algoritma**

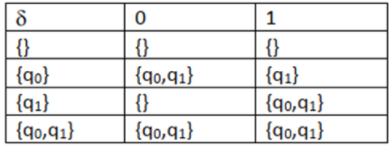
1. Buat semua state yang merupakan subset dari state semula. jumlah state menjadi 2Q
2. Telusuri transisi state–state yang baru terbentuk, dari diagram transisi.
3. Tentukan state awal : {q0}
4. Tentukan state akhir adalah state yang elemennya mengandung state akhir.
5. Reduksi state yang tak tercapai oleh state awal.

Contoh Ubahlah NFA berikut menjadi DFA

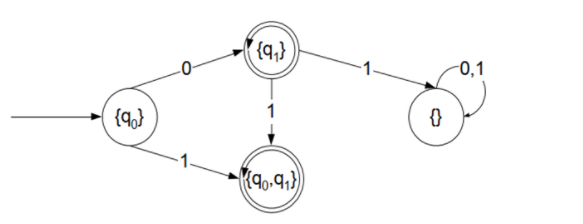
M={{q0,q1}, {0,1}, d, q0,{q1}} dengan tabel transisi



* State yang akan dibentuk : {}, {q0} {q1},{q0,q1}
* Telusuri state



* State awal : {q0}
* State akhir yang mengandung q1, yaitu {q1},{q0,q1}



* Reduksi state yang tak tercapai oleh state awal.

## 7.1. Pendahuluan

FSA hanya memberikan status keluaran berupa indikasi biner “diterima” atau “ditolak” terhadap *string* masukan. Dibutuhkan mesin *finite state* lain yang menghasilkan keluaran bukan biner tapi suatu simbol alfabet lain. *Finite State Transducer*(FST) adalah mesin yang menerima *string*masukan dan menerjemahkannya menjadi *string*keluaran.

FSA : *accepter*, dapat menerima atau tidak.

FSA dengan *output*: transducer

Pendekatan perancangan FST:

* FST yang keluarannya diasosiasikan dengan suatu status, disebut mesin Moore.
* FST yang keluarannya diasosiasikan dengan suatu transisi, disebut mesin Mealy.

## 7.2. Mesin Moore

Mesin Moore dinyatakan dengan 6-*tuple*(Q, Σ, Δ, δ, λ, q0), dimana :

Q: himpunan berhingga status  
Σ: himpunan berhingga simbol alfabet  
Δ: himpunan simbol keluaran (alfabet keluaran)  
δ : fungsi transisi yang memetakan Q x Σ ke Q  
λ: fungsi yang memetakan Q ke Δ, memberikan keluaran yang diasosiasikan dengan tiap status.  
q0: status awal, anggota Q.

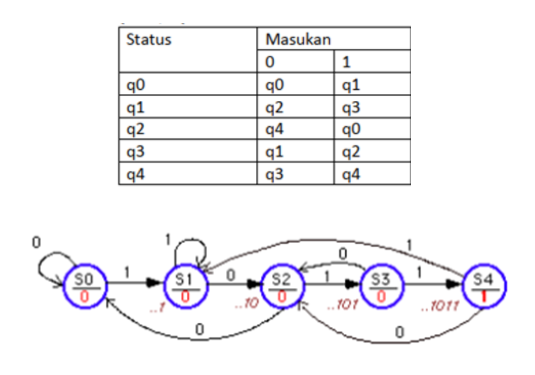
Keluaran mesin Moore terhadap masukan a1 a2…an≥n adalah λ(q0) λ(q1)… λ(qn) dimana q0, q1,…, qn adalah barisan status sedemikian sehingga δ(qi-1,ai) = qi untuk 1 ≤ i≤ n.  
Jika *string*masukan ε, mesin Moore memberikan keluaran λ(q0).

**Contoh :**

Mesin Moore yang menghasilkan keluaran modulo 5 dari suatu bilangan bulat positif biner adalah : (Q, Σ, Δ, δ, λ, q0) dimana :

Q = {q0, q1, q2, q3, q4}  
Σ = {0,1}  
Δ = {0,1,2,3,4}  
λ = Q → Δ, yaitu λ (q0) = j untuk j = 0,1,2,3,4

Q x Σ → Q didefinisikan sbb:



## 7.3. Mesin Mealy

Mesin Mealy dinyatakan dengan 6-*tuple*(Q, Σ, Δ, δ, λ, q0), dimana:

Q: himpunan berhingga status  
Σ: himpunan berhingga simbol alfabet  
Δ: himpunan simbol keluaran (alfabet keluaran)  
δ : fungsi transisi yang memetakan Q x Σ ke Q  
λ: fungsi yang memetakan Q x Σ ke Δ, λ(q,a) memberikan keluaran yang di asosiasikan dengan transisi dari q terhadap symbol keluaran a  
q0: status awal, anggota Q

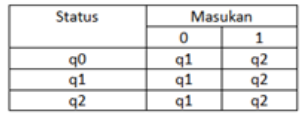
Keluaran mesin Mealy terhadap masukan a1 a2…an ≥ n adalah λ(q0,a1) λ(q0,a1) λ(q1,a2) … λ(qn-1,an) dimana q0, q1,…,qn-1 adalah barisan status sedemikian sehingga δ(qi-1,ai) = qi untuk 1 ≤ I ≤ n. Jika string masukan ε, mesin Mealy memberikan keluaran ε.

**Contoh :**

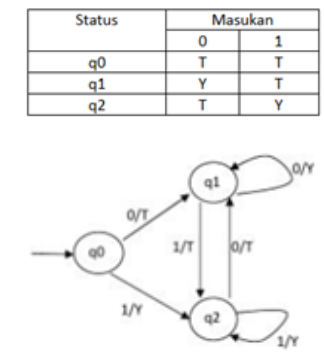
Mesin Mealy yang menerima bahasa himpunan *string*dari alfabet {0,1} yang dua simbol akhirnya sama adalah: (Q, Σ, Δ, δ, λ, q0) dimana:

Q = {q0, q1, q2}  
Σ = {0,1}  
Δ = {y,n}

δ = Q x Σ → Q



λ = Q x Σ → Δ



## 7.4. Mesin Moore (Lanjutan)

M = (Q,Σ,δ,S,∆,λ)

* Q : himpunan *state*
* Σ: himpunan simbol *input*
* δ: fungsi transisi
* S : *state*awal S ∈ Q
* ∆: himpunan *output*
* λ: fungsi *output*untuk setiap *state*

Contoh mesin moore untuk memperoleh modulus 3 pada suatu bilangan biner:

* M = (Q,Σ,δ,S,∆,λ)
* Q : q0,q1,q2
* Σ: [0,1]
* S : q0
* ∆: [0,1,2]
* λ(q0) =0
* λ(q1) =1
* λ(q2) =2

Prinsip:

* jika i diikuti dengan 0, maka hasilnya 2i

    1012=5                           10102= 2\*5 =10

* jika i diikuti dengan 1, maka hasilnya 2i+1

    1012=5                           10112= 2\*5+1 =11

* jika i/3 mempunyai sisa p, maka untuk *input*berikutnya bernilai 0 maka

    2i/3 mempunyai sisa 2p mod 3

    untuk p=0 maka 2p mod 3 = 0

    untuk p=1 maka 2p mod 3 = 2

    untuk p=2 maka 2p mod 3 = 1

* jika i/3 mempunyai sisa p, maka untuk *input*berikutnya bernilai 1 maka

    (2i+1)/3 mempunyai sisa (2p+1) mod 3

    untuk p=0 maka (2p+1) mod 3 = 1

    untuk p=1 maka (2p+1) mod 3 = 0

    untuk p=2 maka (2p+1) mod 3 = 2

Sehingga didapat mesin FSA sbb :

**Contoh :**

* *input*5 (1012), *state*terakhir q2/2 , 5 mod 3 = 2
* *input*10 (10102), *state*terakhir q1/1 , 10 mod 3 = 1

Mesin Mealy

* M = (Q,Σ,δ,S,Δ,λ)
* Q : himpunan *state*
* Σ : himpunan simbol *input*
* δ : fungsi transisi
* S : state awal S ∈Q
* Δ : himpunan *output*
* λ : fungsi *output*untuk setiap **transisi**

Contoh mesin Mealy untuk mendeteksi ekspresi reguler

(0+1)\*(00+11)

**Jawab :**

M = (Q,Σ,δ,S,Δ,λ)  
Q : q0,q1,q2  
Σ : [0,1]  
S : q0  
Δ : [0,1,2]  
λ(q0,0) =T  
λ(q0,1) =T  
λ(q1,0) =Y  
λ(q1,1) =T  
λ(q2,0) =T

λ(q2,1) =Y

**Ekuivalensi mesin Moore dengan mesin Mealy**

* Mesin Moore ke mesin Mealy

    Jml *state*= jml *state*sebelum \* jml *output*

**Mesin Mealy ke mesin Moore**

* Menambah label *output*pada transisi
* Menghapus label *output*pada *state*

## 7.5. Ekivalensi Mesin Moore dan Mesin Mealy

Jika diberikan mesin Moore maka kita dapat membuat mesin Mealy dan sebaliknya.  
Diberikan :

* mesin Moore M1 = (Q, Σ, Δ, δ, λ, q0)
* mesin Mealy M2 = (Q, Σ, Δ, δ, λ’, q0)

Maka didefinisikan λ’(q,a) = λ(δ (q,a)) untuk semua q di dalam Q, a di dalam Σ, dan b di dalam Δ.

jika diberikan :

* mesin Mealy M1 = (Q, Σ, Δ, δ, λ, q0)
* mesin Moore M2 = (Q, Σ, Δ, δ’, λ’, \*q0 ,b0+)

Mesin Moore M2 yg ekivalen dengan M1 dibuat dengan memecah setiap status dari M1 menjadi sejumlah |Q| x |Δ| status yg berbeda pada M2.

Maka didefinisikan :

* δ’(\*q,b+,a)=\*δ(q,a), λ(q,a)]
* λ’(q,b) = b

## 9.1. Pendahuluan

Automata merupakan suatu sistem yang terdiri atas sejumlah berhingga *state*, dimana *state* menyatakan informasi mengenai *input*yang lalu, dan dapat juga dianggap sebagai memori mesin. *Input*pada mesin automata dianggap sebagai bahasa regular yang harus dikenali oleh mesin. Selanjutnya mesin automata membuat keputusan yang mengindikasikan apakah *input*itu diterima atau ditolak. Representasi suatu bahasa regular pada automata lebih lazim diwakili dengan menggunakan ekspresi regular.

*Finite state automata* dan ekspresi regular awalnya dikembangkan berdasarkan pemikiran *neural network*dan*switching circuit*. *Finite state automata* merupakan *tool*yang sangat berguna dalam perancangan *lexical analyzer*, yaitu bagian dari kompilator yang mengelompokkan karakter-karakter ke dalam *token*, yang berupa unit terkecil seperti nama, variabel dan *keyword*. Dalam sistem penulisan kompilator secara otomatis akan mentransformasikan ekspresi regular ke dalam *finite state automata* untuk dipakai sebagai penganalisa leksikal. *Finite state automata*dan ekspresi regular dipakai juga dalam :

1. *text editor*,
2. *pattern-matching*,
3. sejumlah pemrosesan teks, dan
4. program *file-searching*, serta sebagai
5. konsep matematis untuk aplikasi disiplin ilmu lain seperti logika.

Suatu bahasa pemrograman harus didefinisikan secara tepat. Spesifikasi dari sebuah bahasa pemrograman meliputi:

1. himpunan simbol-simbol (alphabet) yang bisa dipakai untuk membentuk program yang benar,
2. himpunan program yang benar secara sintaktik, dan
3. makna dari program tersebut.

## 9.1. Pendahuluan

Automata merupakan suatu sistem yang terdiri atas sejumlah berhingga *state*, dimana *state* menyatakan informasi mengenai *input*yang lalu, dan dapat juga dianggap sebagai memori mesin. *Input*pada mesin automata dianggap sebagai bahasa regular yang harus dikenali oleh mesin. Selanjutnya mesin automata membuat keputusan yang mengindikasikan apakah *input*itu diterima atau ditolak. Representasi suatu bahasa regular pada automata lebih lazim diwakili dengan menggunakan ekspresi regular.

*Finite state automata* dan ekspresi regular awalnya dikembangkan berdasarkan pemikiran *neural network*dan*switching circuit*. *Finite state automata* merupakan *tool*yang sangat berguna dalam perancangan *lexical analyzer*, yaitu bagian dari kompilator yang mengelompokkan karakter-karakter ke dalam *token*, yang berupa unit terkecil seperti nama, variabel dan *keyword*. Dalam sistem penulisan kompilator secara otomatis akan mentransformasikan ekspresi regular ke dalam *finite state automata* untuk dipakai sebagai penganalisa leksikal. *Finite state automata*dan ekspresi regular dipakai juga dalam :

1. *text editor*,
2. *pattern-matching*,
3. sejumlah pemrosesan teks, dan
4. program *file-searching*, serta sebagai
5. konsep matematis untuk aplikasi disiplin ilmu lain seperti logika.

Suatu bahasa pemrograman harus didefinisikan secara tepat. Spesifikasi dari sebuah bahasa pemrograman meliputi:

1. himpunan simbol-simbol (alphabet) yang bisa dipakai untuk membentuk program yang benar,
2. himpunan program yang benar secara sintaktik, dan
3. makna dari program tersebut.

## 9.3. Ekspresi Regular pada suatu Finite State Automata

**Definisi 3.**

Suatu finite state automata (FSA) adalah suatu 5-tuple (Q, ∑, qo , δ , A ) dengan Q adalah finite set, ∑ adalah finite alphabet untuk input simbol, qo ∈∈ Q adalah state awal, δ adalah fungsi transisi dari Q x ∑ →→ Q , dan A ⊆⊆ Q adalah himpunan target state.

Untuk sembarang q ∈∈ Q dan a ∈∈ ∑ , δ (q , a ) sebagai fungsi transisi yang akan memindahkan state q ke state lain dengan input a.

**Definisi 4.**

Misalkan M = (Q, ∑ , qo , δ , A ) adalah FSA, fungsi δ \* : Q x ∑ →→ Q didefinisikan dengan  
i. sembarang q ∈∈ Q , δ\* (q , Λ) = q  
ii. sembarang y ∈∈ ∑\* , a ∈∈ ∑ , dan q ∈∈ Q , δ\* (q , ya ) →→ δ (δ\* (q , y ) , a)

**Contoh**

Misalkan M = (Q, ∑ , qo , δ , A ) adalah suatu FSA dengan diagram transisi

dengan menggunakan definisi di atas, tentukan penyelesaian dari δ\* (q , abc ) !

Penyelesaian :

δ\* (q , abc ) = δ (δ\* (q , ab ), c )  
= δ (δ (δ\* (q , a ) , b ), c )  
= δ (δ (δ\* (q , Λa ) , b ), c )  
= δ (δ (δ (δ\* (q , Λ) , a ) , b ), c )  
= δ (δ (δ (q , a ) , b ), c )  
= δ (δ (q1 , b ), c )  
= δ (q2 , c )  
= q3

## 9.4. Penerapan ER

Misalkan M = (Q, ∑ , qo , δ , A ) adalah FSA. Suatu string x ∈∈ ∑\* diterima oleh M jika δ\* (qo ,x) ∈∈ A. Bahasa yang diterima oleh M adalah himpunan

L (M ) = { x ∈∈ ∑\* | x diterima oleh M }

Jika L adalah sembarang bahasa atas ∑ , maka L dikatakan diterima oleh M jika dan hanya jika L = L (M).

**Contoh**

Diberikan FSA yang dideskripsikan dengan diagram berikut :

Tunjukkan bahwa string 23 diterima sebagai bahasa oleh M.

**Penyelesaian :**

Dari diagram tersebut diketahui bahwa Q = {a , b , c } , ∑ = {0 , 1} , qo = a , A = {a , c}. Bilangan desimal 23 terlebih dulu dikonversi ke biner menjadi, 23 = 10111.

δ\* (a , 10111) = δ (δ\* (a , 1011),1)  
= δ (δ (δ\* (a ,101),1),1)  
= δ (δ (δ (δ\* (a , 10),1),1),1)  
= δ (δ (δ (δ (δ\* (a , 1),0),1),1),1)  
= δ (δ (δ (δ (δ\* (a , Λ1),0),1),1),1)  
= δ (δ (δ (δ (δ (δ\* (a , Λ),1),0),1),1),1)  
= δ (δ (δ (δ (δ (a ,1),0),1),1),1)  
= δ (δ (δ (δ (b ,0),1),1),1)  
= δ (δ (δ (a ,1),1),1)  
= δ (δ (b ,1),1)  
= δ (c ,1)  
= c

dari uraian tersebut didapat δ\* (a , 10111) = c ∈∈ A , dengan demikian string 23 diterima oleh FSA.

Ekspresi regular sebagai suatu pola untuk string merupakan pembangun (generator) dari bahasa. Berikut diberikan contoh permasalahannya. Misalkan diberikan FSA sebagai berikut :

Tentukan ekspresi regularnya.

**Penyelesaian.**

▪ Λ ∈∈ L karena δ\* (a, Λ) = a ∈∈ A sehingga Λ diterima oleh L.  
▪ 0\* ∈∈ L karena δ\* (a, 0\*) = a ∈∈ A sehingga 0\* diterima oleh L.  
▪ (10\*1(01)\*1\*)\* ⊆⊆ L  
δ\* (a, 10\*1(01)\*1\*10\*1(01)\*1\*)  
= δ\* (c, 10\*1(01)\*1\*)  
= δ\* (c, 0\*1(01)\*1\*) ; terdapat dua kemungkinan 0\* | Λ dan 0\* = Λ

Untuk 0\* ≠ Λ  
δ\* (c, 0\*1(01)\*1\*)  
= δ\* (b, 1(01)\*1\*)  
= δ\* (c, (01)\*1\*)  
= c

Untuk 0\* = Λ  
δ\* (c, 0\*1(01)\*1\*)  
= δ\* (c, 1(01)\*1\*)  
= δ\* (c, (01)\*1\*)  
= c

Dari uraian tersebut, dihasilkan bahasa L = { 0\* U (10\*1(01)\*1\* } dengan ekspresi regularnya adalah r = 0\* + (10\*1(01)\*1\*.

## 9.5. Rangkuman Materi

Finite state automata dan ekspresi regular awalnya dikembangkan berdasarkan pemikiran neural network dan switching circuit. Finite state automata merupakan tool yang sangat berguna dalam perancangan lexical analyzer, yaitu bagian dari kompilator yang mengelompokkan karakter-karakter ke dalam token, yang berupa unit terkecil seperti nama, variabel dan keyword. Dalam sistem penulisan kompilator secara otomatis akan mentransformasikan ekspresi regular ke dalam finite state automata untuk dipakai sebagai penganalisa leksikal. Finite state automata dan ekspresi regular dipakai juga dalam :

1. text editor,
2. pattern-matching,
3. sejumlah pemrosesan teks, dan
4. program file-searching, serta sebagai
5. konsep matematis untuk aplikasi disiplin ilmu lain seperti logika.

Suatu bahasa pemrograman harus didefinisikan secara tepat. Spesifikasi dari sebuah bahasa pemrograman meliputi:

1. himpunan simbol-simbol (alphabet) yang bisa dipakai untuk membentuk program yang benar,
2. himpunan program yang benar secara sintaktik, dan
3. makna dari program tersebut.

## 10.1. Pendahuluan

Bahasa disebut reguler jika terdapat FSA yang dapat menerimanya. Bahasa reguler dinyatakan secara sederhana dengan ekspresi reguler/*regular expression* (RE).

**Contoh penerapan**: *searching string* pada file

**Hubungan RE dan NFA**: Setiap RE ada satu NFA dengan ε-move yang ekuivalen.  
Konversi bentuk dari ekspresi regular menjadi bentuk DFA:  
RE -> NFA dengan ε Move -> DFA

**Definisi ekspresi reguler**  
Jika Σ merupakan himpunan simbol, maka

* ∅ , ε , dan a ∈ Σ adalah ekspresi reguler dasar
* Jika r dan t masing masing merupakan ekspresi reguler maka komposisi berikut merupakan ekspresi reguler :

**Contoh ekspresi reguler**

* (0+1)\* : himpunan seluruh string yang dapat dibentuk dari simbol ‘0’ dan ‘1’
* (0+1)\*00(0+1)\* : himpunan string biner yang mengandung paling sedikit satu substring ‘00’
* (0+1)\*00 : himpunan string biner yang diakhiri dengan ‘00’

Apabila r adalah RE, maka L(r) adalah bahasa reguler yang dibentuk menggunakan ekspresi reguler r.

## 10.2. Sifat Bahasa Reguler

a) Tertutup terhadap operasi himpunan sederhana

Jika L1 dan L2 adalah bahasa reguler, maka L1∪L2, L1∩L2, L1L2, ~(L1) dan L1\* adalah bahasa reguler juga

b) Tertutup terhadap *homomorphic image*

* Jika L1 adalah bahasa reguler, maka homomorphic image h(L1) adalah bahasa reguler juga.
* Dimisalkan Σ dan Γ adalah alfabet, maka fungsi homomorphic dinyatakan dengan h : Σ → Γ
* Jika w = a1 a2 ... an, maka h(w) = h(a1) h(a2 ) ... h(an)
* Jika L adalah bahasa pada Σ maka homomorphic image bahasa L adalah h(L)= { h(w) | w∈L}

**Contoh:**

Dimisalkan Σ = {a,b} dan Γ = {a,b,c} dan didefinisikan h(a) = ab dan h(b) =bbc

homomorphic image bahasa L = {aa,aba } adalah :

**Jawab:** h(L)= { abab, abbbcab}

## 10.3. Konversi Ekspresi Reguler ke FSA

**Contoh :**Tentukan FSA untuk ekspresi reguler r = 0(1 | 23)\*

## 10.4. DFA dan Tatabahasa Reguler

**Tatabahasa Linier kiri dan linier kanan**

* Suatu tatabahasa G(T,N,S,P) disebut linier kiri jika seluruh aturan produksinya berbentuk A → xB dengan A, B ∈ N dan x ∈ T\*
* Suatu tatabahasa G(T,N,S,P) disebut linier kanan jika seluruh aturan produksinya berbentuk A → Bx dengan A, B ∈ N dan x ∈ T\*

Tatabahasa reguler bisa bersifat linier kiri atau linier kanan.

Contoh :

* Tatabahasa G = { {a,b}, {S}, S, P } dengan aturan produksi P adalah :
  + S → abS |a adalah tatabahasa linier kanan /regular
* Tatabahasa G = {{a,b}, {S, S1,S2}, S, P } dengan aturan produksi P
  + adalah S → S1ab, S1→ S1ab | S2, S2→ a
  + adalah tatabahasa linier kiri /reguler
* Tatabahasa G = {{a,b}, {S, A, B}, S, P } dengan aturan produksi P
  + adalah S → A, A → aB | ε, B → Ab, adalah bukan tatabahasa regular.

**Konversi DFA ke tatabahasa linier**

Setiap DFA dapat diubah menjadi tatabahasa yang memiliki aturan produksi yang linier. Aturan pengubahan ini adalah sebagai berikut :

* setiap transisi status δ(A,a)=B diubah menjadi aturan produksi A → aB
* setiap status akhir P diubah menjadi aturan produksi Pε→

Contoh FSA berikut :

Tatabahasa linier untuk FSA tersebut yaitu :

* G = ({a,b}, {S,S1,S2,S3}, S, P ) dengan aturan produksi P adalah :
* S → S1, S1 → aS2, S1 → bS3, S3 → ε

**Konversi tatabahasa linier ke DFA**

* setiap aturan produksi A → aB diubah menjadi transisi status δ(A,a)=B
* setiap aturan produksi A → a diubah menjadi δ(A,a)=SF
* untuk a ∈ T\* dengan |a|>1 dapat dibuat state tambahan
* setiap aturan produksi A → B diubah menjadi δ(A,ε)=B

Contoh: Tata bahasa G = ({a,b}, {V0,V1}, V0, P ) dengan P :

* V0 →aV1
* V1 →abV0 | b

Mesin FSA nya menjadi :