RESUME

TEORI BAHASA DAN OTOMATA

DOSEN PENGAMPU:

Widya Darwin S.Pd,MPdt

OLEH:

Elsa Rahma Hidayani

21346006

Informatika

PROGRAM STUDI INFORMATIKA

JURUSAN ELEKTRONIKA

FAKULTAS TEKNIK

UNIVERSITAS NEGERI PADANG

2023

DAFTAR ISI

KONSEP DASAR TEORI BAHASA & OTOMATA

A. Tingkat bahasa pemograman.

1. Bahasa pemrograman tingkat rendah

Bahasa mesin atau kode mesin merupakan satu-satunya bahasa yang bisa di olah komputer secara langsung tanpa transformasi sebelumnya (kompilasi). Contoh bahasa pemrograman tingkat rendah :Bahasa mesin (machine language).

2. Bahasa pemrograman tingkat menengah

Bahasa tingkat menengah memberikan satu tingkat abstraksi di atas kode mesin. Bahasa assembly memiliki sedikit semantik atau spesifikasi formal, karena hanya pemetaan simbol yang dapat di baca manusia. Biasanya, satu instruksi mesin di wakili sebagai satu baris kode *assembly*. Assembler menghasilkan file objek yang bisa dihubungkan dengan file objek lain atau dimuat sendiri.

Contoh bahasa pemrograman tingkat menengah :Assembler.

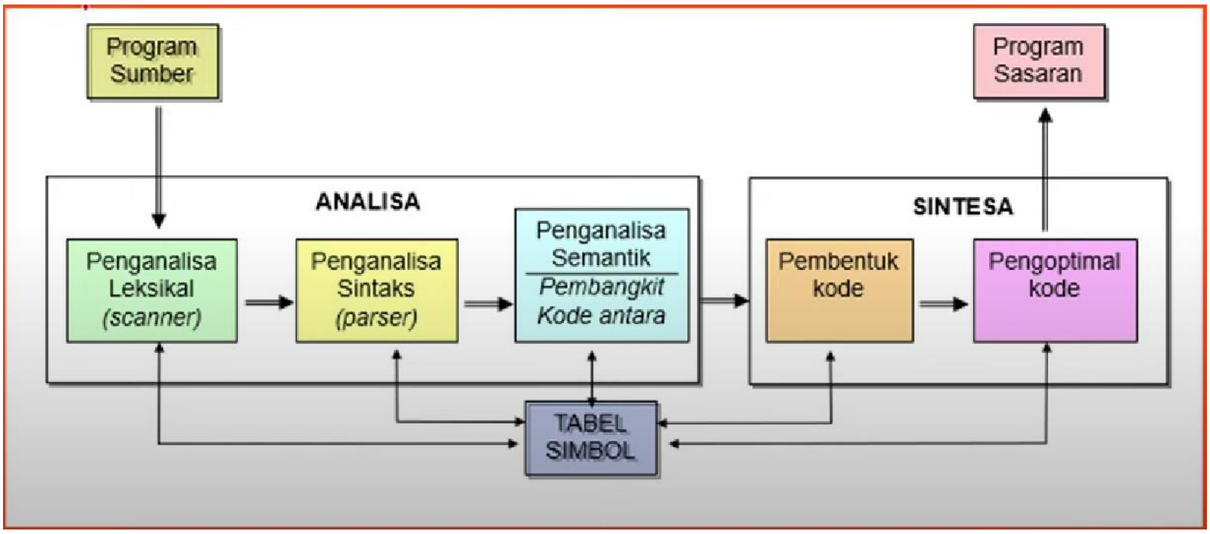
3. Bahasa pemrograman tingkat tinggi

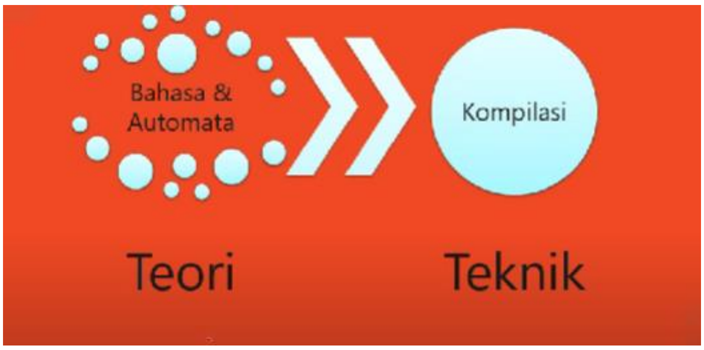
Contoh bahasa pemrograman tingkat tinggi :

C++ (Turbo C++)

Pascal (Turbo Pascal)

Proses Kompilasi



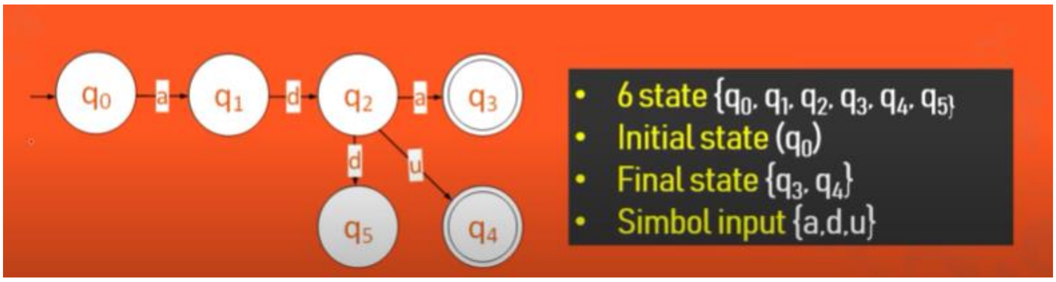


Teori Bahasa dan otomata adalah dasar dari Teknik kompilasi

Sebuah mesin yang hanya mengenali Bahasa mesin dapat memahami Bahasa pemograman tingkat tinggi karena ada compilator/penerjemah

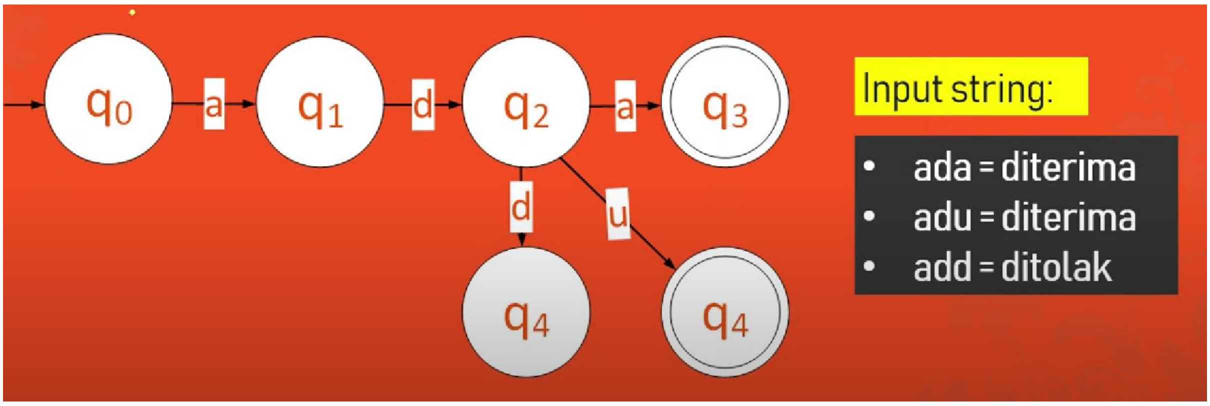
Teori bahasa membahas mengenai bahasa formal. Salah satunya adalah untuk kepentingan kompilator
Bahasa di dalam kamus adalah suatu sistem yang meliputi pengekspresian gagasan, fakta, konsep, termasuk sekumpulan simbol-simbol dan aturan untuk melakukan manipulasinya. Bahasa bisa juga disebut sebagai rangkaian 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.
Input pada mesin otomata dianggap sebagai Bahasa yang harus dikenali oleh mesin, mesin akan mengindikasikan apakah suat bahasa dapat diterima atau tidak.
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.

Contoh mesin otomata sederhana



String input diterima jika mencapai final state, selain itu ditolak
Pembacaan simbol pertama dimulai dari initial state
Perpindahan state berdasarkan simbol yang dibaca

Maka hasilnya akan menjadi



SIMBOL STRING DAN BAHASA

A. Apa yang dimaksud dengan symbol

Simbol adalah sebuah entitas abstrak (seperti halnya pengertian titik dalam geometri). Sebuah huruf atau sebuah angka adalah contoh simbol. •

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.

B. Apa yang dimaksud string

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.

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 hinpunan hingga (finite set) simbol-simbol.

Suatu sistem yang meliputi pengesksprесian gagasan. fakta, konsep, termasuk sekumpulan simbol-simbol dan aturan untuk melakukan manipulasinya.

Bahasa adalah Suatu sistem yang meiliputi pengekspresian gagasan, fakta konsep,termasuk sekumpulan simbol-simbol dan aturan untuk melakukan manipulasinya.Alphabet adalah himpunan berhingga dari simbol-simbol.

Bahasa juga disebut sebagai rangkaian simbol yang memiliki makna.

oSebuah bahasa adalah himpunan string-string dari simbolsimbol untuk suatualphabet.

oKarena sebuah bahasa adalah kumpulan dari string-string, maka kita bisamempunyai bahasa yang tidak mempunyai string-string, yaitu Bahasa kosongyang dinotasikan seperti menuliskan notasi himpunan kosong.

oInput pada mesin otomata dianggap sebagai Bahasa yang harus dikenali olehmesin.

oMesin akan mengindikasikan apakah suatu Bahasa dapat diterima atau tidak

TATA BAHASA HIRARKI CHOMSKY DAN ATURAN PRODUKS

A. Simbol terminal dan non terminal

Apa itu simbol terminal?

simbol terminal adalah simbol yang tidak dapat diturunkan lagi

arti bisa **diturunkan** yaitu misalkan simbol A yang dapat diturunkan menjadi bc

yang termasuk simbol terminal:

huruf kecil alfabet: a,b,c

simbol operator: + (tambah), – (kurang)

simbol tanda baca: , (koma) ! (tanda seru)

string yang tercetak tebal, misalnya: if, then dan else

Apa itu simbol NON terminal (Variabel)?

adalah simbol yang masih bisa diturunkan menjadi simbol terminal atau non terminal lainnya

yang termasuk simbol NON terminal (variabel):

huruf besar alfabet: A, B, C

huruf S sebagai simbol awal

String yang tercetak miring misal expr dan stmt

B. Aturan Produksi

Dalam teori bahasa dan otomata, aturan produksi dinyatakan dalam:

$\alpha \rightarrow \beta$

(dibaca: alpha menghasilkan atau menurunkan beta)

Dengan menerapkan aturan produksi, suatu tata bahasa bisa menghasilkan sejumlah string.

Himpunan semua string tersebut adalah bahasa yang didefinisikan oleh tata bahasa tersebut.

contoh aturan produksi:

$T \rightarrow a$

(dibaca: T menghasilkan a)

Contoh lain:

$E \rightarrow T \mid T + E$

(dibaca: E menghasilkan T atau E menghasilkan T + E)

C. Hirarki Chomsky

Grammar atau tata bahasa didefinisikan secara formal sebagai kumpulan dari himpunan himpunan variabel, simbol simbol terminal, simbol awal, yang dibatasi oleh aturan aturan produksi.

4 tingkatan tata bahasa menurut Chomsky:

a) Tipe 0 – Unrestricted Grammar
b) Tipe 1 – Context Sensitive Grammar
c) Tipe 2 – Context Free Grammar
d) Tipe 3 – Regular Grammar

a) Tipe 0 – Unrestricted Grammar

- Aturan produksinya tidak memiliki batasan
- Dalam aturan $\alpha \rightarrow \beta$ pada tipe 0:
    - alpha adalah string terminal dan non-terminal dengan setidaknya 1 non-terminal

- alpha tidak boleh kosong
- beta adalah rangkaian simbol terminal dan non – terminal
- α adalah (V + T)*V(V+T)*
- β adalah (V+T)*
- note:
- (tanda plus '+' dibaca atau)
- (tanda * (asterisk) yang berarti bisa tidak muncul atau bisa juga muncul hingga berkali kali (0 – n))

Tipe ini menghasilkan bahasa yang dikenali oleh mesin Turing. Bahasa ini juga dikenal dengan nama "Recursively Enumerable Languages".

jadi aturan yang perlu diingat adalah:

α adalah (V + T)*V(V+T)*
β adalah (V+T)*

Contoh penerapan aturan tipe 0:

S → ACaB (Terpenuhi, karena di ruas kiri ada non terminal, di ruas kanan ada terminal dan non-terminal)
Bc → acB (Terpenuhi, karena di ruas kiri ada non terminal, di ruas kanan ada terminal dan non-terminal)
CB → DB (Terpenuhi)
aD → Db (Terpenuhi)
Sab → ba (Terpenuhi)
A → S (Terpenuhi)

b) Tipe 1 – Context Sensitive Grammar

Aturan produksinya sama dengan tipe 0 namun dibatasi dengan aturan |a|≤|b| (jumlah simbol di ruas kiri harus lebih kecil atau sama dengan jumlah simbol di ruas kanan)
Aturan S → Ɛ dibolehkan jika S tidak muncul pada ruas kanan setiap aturan
Menghasilkan bahasa yang dikenali oleh Linear Bound Automata

jadi aturan yang perlu diingat adalah:

|a|≤|b|
a adalah (V+T)*V(V+T)*
b adalah (V+T)* (V+T) (V+T)*

Contoh penerapan:

S → AB (Terpenuhi)
AB → abcd (Terpenuhi, ukuran ruas kiri lebih kecil dari ruas kanan)
B → b (Terpenuhi)
aD → Db (Terpenuhi, ukuran kedua ruas sama)
aB → aa | aaAA (Terpenuhi)
Ac → Bbcc (Terpenuhi)

c) Tipe 2 – Context Free Grammar

Aturan produksi ini sama dengan tipe 0 namun a sisi kiri hanya boleh memiliki 1 variabel non terminal (|a| = 1) dan b tidak memiliki batasan
a adalah sebuah simbol non terminal
b dapat berupa rangkaian atau simbol terminal, non terminal atau epsilon
tipe ini menghasilkan bahasa yang dikenali oleh Non Deterministic Push Down Automata

jadi aturan yang perlu diingat adalah:

|a| = 1

a adalah V

b adalah (V+T)*

contoh:

S → X a (Terpenuhi, di ruas kiri hanya ada satu non terminal yaitu S dan di ruas kanan boleh terminal atau non terminal)

X → a (Terpenuhi)

X → aX (Terpenuhi)

X → abc (Terpenuhi)

X → Є (Terpenuhi)

S → AB (Terpenuhi)

A → a (Terpenuhi, di ruas kiri hanya ada satu non terminal dan di ruas kanan ada satu terminal)

B → b (Terpenuhi)

d) Tipe 3 – Regular Grammar

Aturan S → Ɛ dibolehkan jika S tidak muncul pada ruas kanan setiap aturan

α adalah sebuah simbol non terminal

β adalah simbol terminal atau simbol terminal dengan sebuah simbol variabel yang jika ada terletak pada posisi paling kanan

Menghasilkan bahasa yang dikenali oleh Finite State Automata

jadi perlu diingat:

α adalah V

β adalah T* atau T*V

Contoh:

X → Ɛ

X a | aY

Y → b

S → abB

B → cd

FINAL STATE AUTOMATA

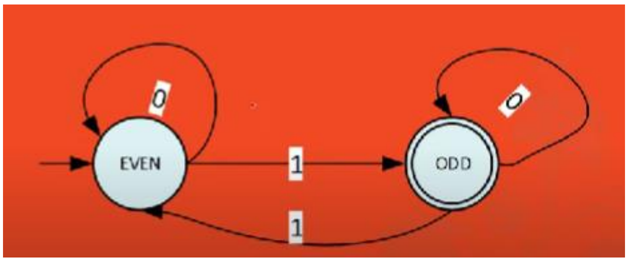
A. Apa itu FSA?

- FSA adalah mesin abstrak berupa sistem model matematika dengan masukan dan keluaran diskrit yang dapat mengenali bahasa paling sederhana
- mekanisme kerja FSA dapat diterapkan pada analisis leksikal, text editor, protocol dan komunikasi jaringan

contoh analisis leksikal:

- analisis leksikan adalah salah satu bagian dari proses penerjemahan
- code yang dituliskan akan dianalisis seperti yang mana variabel, operator, keyword

B. Arti bentuk symbol pada graf transisi FSA



Keterangan gambar:

- Initial state ditandai dengan busur tanpa asal
- Lingkaran menyatakan state
- Label pada lingkaran adalah nama state, yaitu EVEN dan ODD
- Busur adalah transisi / arah perpindahan state
- Label pada busur adalah simbol input, yaitu 0 dan 1
- Lingkaran ganda menyatakan final state, yaitu ODD

C. Pernyataan Final State Automata secara Formal

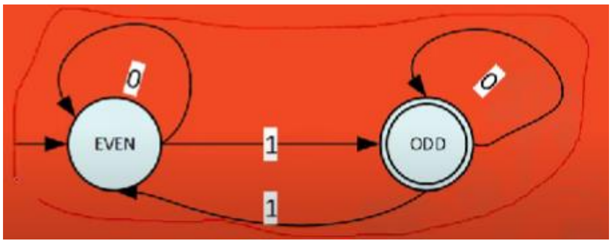
FSA dapat dinyatakan dalam 5 tupel atau M=(Q, Σ, δ, q0, F) dimana:

- Q adalah Himpunan state atau kedudukan
- Σ (dibaca: sigma) adalah Himpunan symbol input / masukan / abjad
  - Jika pada contoh sebelumnya, terdapat input / simbol yaitu 0 dan 1
- δ (dibaca: delta) adalah Fungsi transisi
- q0 (atau bisa juga S) adalah State awal q0. dimana q0 Q
- F adalah Himpunan State akhir (final) F ⊆ Q

Catatan:

- Jumlah state akhir bisa lebih dari satu

Contoh:



Q = {ODD, EVEN}

$\Sigma = \{0, 1\}$
$q0$ = EVEN
F = {ODD}
δ (Transisi):
δ(EVEN, 0) = EVEN State EVEN menerima inputan 0 ke EVEN
δ(EVEN, 1) = ODD
δ(ODD, 0) = ODD
δ(ODD, 1) = EVEN

catatan: Final state F = {ODD} berupa himpunan karena final state dapat berjumlah lebih dari satu

Jadi aturan ini menyesuaikan model mesinnya

Contoh ketika mesin digunakan:

Ketika mendapat input 1101, maka urutan state yang terjadi adalah EVEN1ODD1EVEN0EVEN1ODD
berakhir dengan state ODD, maka 1011 diterima mesin
Karena diterima maka dapat diketahui bahwa jumlah bit 1 nya adalah ganjil

Contoh Lain:

Ketika mendapat input 101, maka urutan state yang terjadi adalah EVEN1ODD0ODD1EVEN
berakhir dengan state EVEN, maka 101 ditolak mesin
Karena ditolak maka dapat diketahui nilai bit nya genap

Berdasarkan pendefinisian kemampuan merubah statenya, FSA dikelompokkan kedalam 2 jenis, yaitu:

Deterministic FSA
Non Deterministic FSA

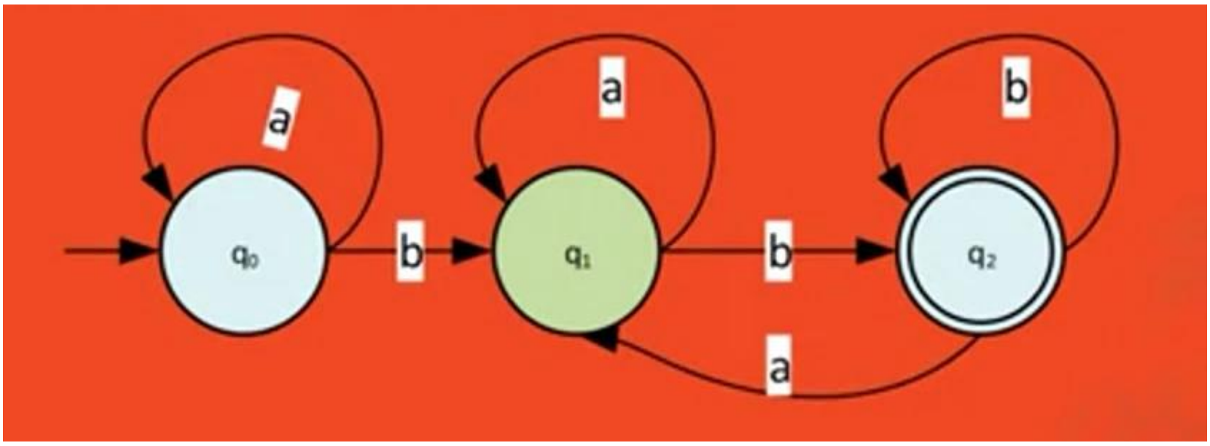
DETERMINISTIC FINAL STATE AUTOMATA

Berdasarkan pendefinisian kemampuan merubah statenya, FSA dikelompokkan ke dalam dua jenis

- Deterministic FSA
- Non- Deterministic FSA

Contoh 1:

Pada DFA, dari suatu state hanya ada tepat satu state berikutnya untuk setiap simbol masukan yang diterima



Q = {q0, q1, q2}

∑ = { a,b }

S = q0

F = {q2}

Fungsi Transisi

$$\delta(q_0 , a) = q_0 ; \quad \delta(q_0 , b) = q_1$$
$$\delta(q_1 , a) = q_1 ; \quad \delta(q_1 , b) = q_2$$
$$\delta(q_2 , a) = q_1 ; \quad \delta(q_2 , b) = q_2$$

Tabel transisi

δ	a	b
q_0	q_0	q_1
q_1	q_1	q_2
q_2	q_1	q_2

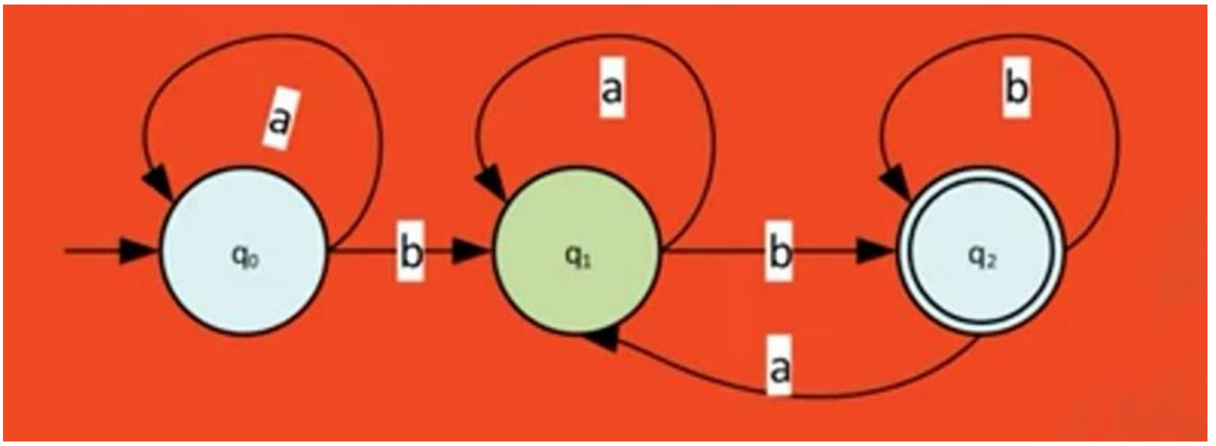
Kapan suatu string input dinyatakan diterima?

Suatu string x dinyatakan diterima,Bila δ (S,x) berada pada state akhir.

Bila M adalah sebuah bahasa FSA

M = (Q, ∑, δ, S, F) menerima bahasa yang disebut L(M) yang merupakan himpunan { x | δ(S,x) di dalam F }

Contoh 2:

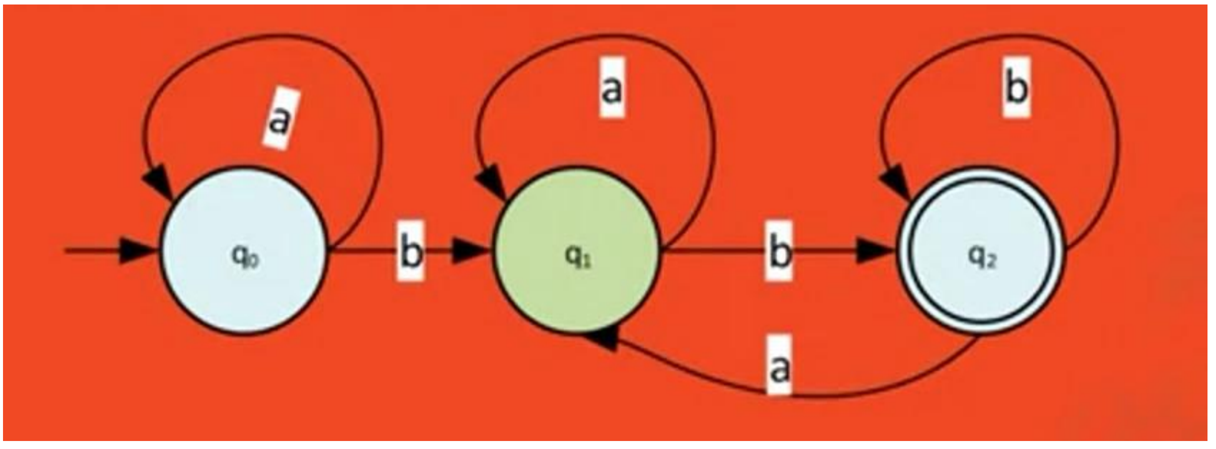


Jika pada gambar contoh 1 kita inputkan string 'abb'. Maka:

$$\delta(q_0 , abb) = \delta(q_0 , bb) = \delta(q_1 , b) = q_2$$

Karena q2 adalah state akhir maka 'abb' berada berada dalam L(M}

Contoh 3:

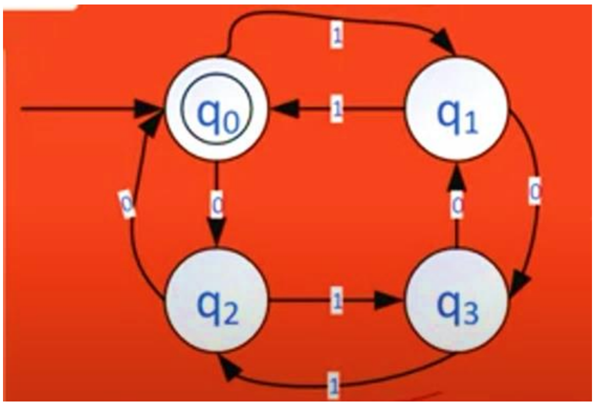


Jika pada gambar contoh 1 inputkan string 'baba'. Maka;

$$\delta(q_0 , baba) = \delta(q_1 , aba) = \delta(q_1 , ba) = \delta(q_2 , a) = q_1$$

Karena q1 bukan state akhir maka 'baba' tidak berada dalam L(M)

Contoh 4 – Pengecekan bit 0 dan 1 ganap



Tabel transisi

δ	0	1
q0	q2	q1
q1	q3	q0
q2	q0	q3
q3	q1	q2

Konfigurasi NFA berikut :

Q = {q0, q1, q2,q3}

∑ = { 0,1 }

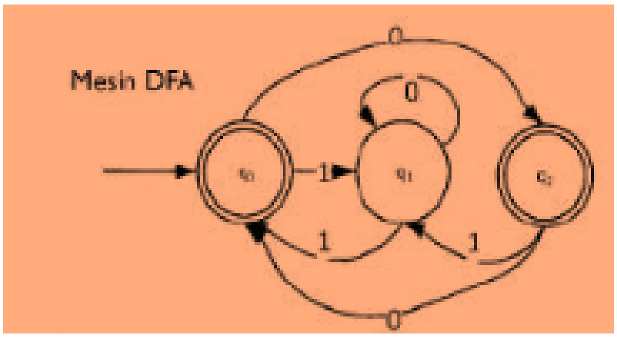
S = q0

F = {q0}

Secara formal dinyatakan sebagai berikut:

String	Status
0011	Diterima
00111	Ditolak
01010	Ditolak
010111	Diterima

Contoh 5



Tabel transisi:

δ	0	1
q_0	q_2	q_1
q_1	q_1	q_0
q_2	q_0	q_1

Secara formal dinyatakana sebagai berikut:

String	Status
01	Ditolak
10	Ditolak
1110	Ditolak
11	Diterima
101	Diterima
1100101	Diterima
0100011101	Ditolak

NON DETERMINISTIC FINITE STATE AUTOMATA

Kemungkinan transisinya ke lebih dari satu state. Dari suatu statebisa terdapat 0, 1 atau lebih transisi dengan label input yang sama.Perubahan state dapat terjadi secara spontan tanpa input (transisikosong).
Salah satu tracing-nya berakhir di state akhir, atau himpunan state setelah membaca string tersebut mengandung state akhir
FSA dinyatakan oleh 5 tuple :M = (Q , Σ , δ , S , F) dimana:
Q = Himpunan state / kedudukan
Σ = Himpunan symbol input
δ = Fungsi Transisi
q0 = State awal q0,
F = himpunan state akhir
* Catatann : Jumlah state akhir bisa lebih dari satu

A. NFA Dengan E-MOVE

Pada NFA dengan e-move dibolehkan berpindah state (tanpa membaca input)

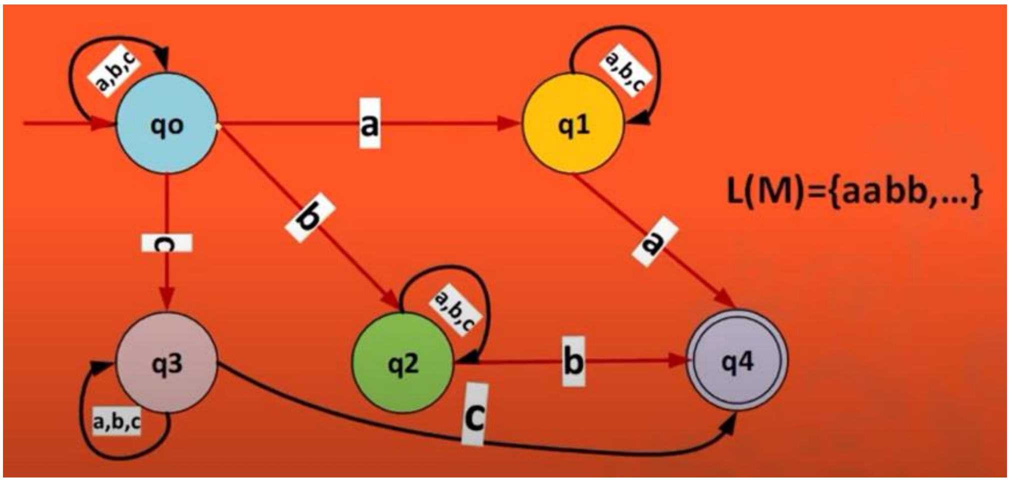
B. E-closure untuk NFA dengan e-move

* E-closure adalah himpunan state-state yang dapat dicapai dari suatu state tanpa membaca input
* E-closure (q0)
berarti bahwa himpunan yang dapat dicapai dari q0 tanpa membaca input

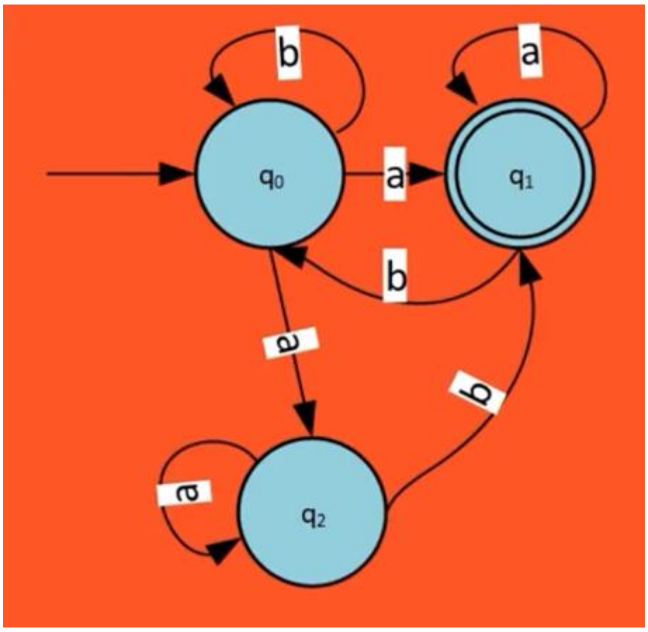
Contoh 1 :
FSA dinyatakan oleh 5 tuple :M = (Q , Σ , δ , S , F) dimana:
Q = {q0 , q1 , q2 , q3 , q4 }
Σ = {a,b,c}
S = q0
F = {q4}

Tabel Transisi :

δ	a	b	c
q0	{q0,q1}	{q0,q2}	{q0,q3}
q1	{q1,q4}	{q1}	{q1}
q2	{q2}	{q2,q4}	{q2}
q3	{q3}	{q3}	{q3,q4}
q4	ø	ø	ø



Contoh 2 :



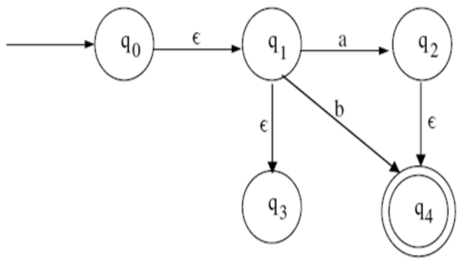
Gambar tersebut termasuk NFA karena jika q0 manerima inputan ‘a’ maka akan berpindah ke q1 atau q2. Tidak tepat satu berikutnya

δ	a	b
q_0	{q1, q2}	q_0
q_1	q_1	q_0
q_2	q_2	q_1

Pada NFA, String diterima jika setidaknya ada 1 dari semuakemungkinan transisi berakhir pada sebuah final sate. Harus mencobasemua kemungkinan.Pada NFA boleh terdapat transisi kosong. Simbol ε berarti tanpamasukan atau disebut transisi kosong. Terjadi perubahan state secara spontan

C. Ekuivalensi NFA dengan e-move

Kita dapat membuat suatu NFA tanpa e-move dari NFA dengan e-move yang ekuivalen

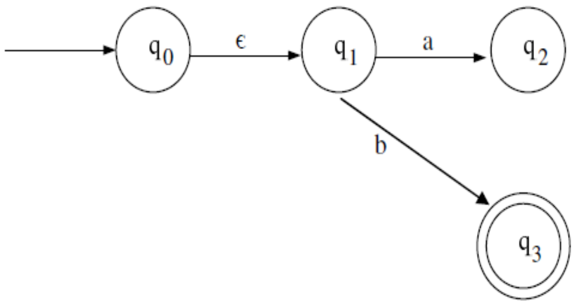


Dari gambar di atas, kita ketahui Є – Closure untuk setiap *state adalah sebagai berikut.*

- Є – Closure (q0) = { q0, q1, q3 }
- Є – Closure (q1) = { q1, q3 }
- Є – Closure (q2) = { q2, q4 }
- Є – Closure (q3) = { q3 }
- Є – Closure (q4) = { q4 }

- **Langkah Ekuivalensi NFA dengan e-move ke NFA tanpa e-move**

* Buatlah tabel tarnsisi dari NFA dengan e-move



Tabel Transisi

δ	a	b
q_0	Ø	Ø
q_1	$\{q_2\}$	$\{q_3\}$
q_2	Ø	Ø
q_3	Ø	Ø

* Tentukan ϵ-closure untuk setiap *state*
Є – Closure (q0) = { q0, q1 }
Є – Closure (q1) = { q1 }
Є – Closure (q2) = { q2 }
Є – Closure (q3) = { q3 }

* Carilah setiap fungsi transisi hasil dari pengubahan NFA ϵ – move ke NFA tanpa ϵ – move. Fungsi transisi itu ditandai dengan simbol δ'

$$\delta'(q_0, a) = \epsilon\_cl(\delta(\epsilon\_cl(q_0), a))$$
$$= \epsilon\_cl(q_2)$$
$$= \{ q_2 \}$$
$$\delta'(q_0, b) = \epsilon\_cl(\delta(\epsilon\_cl(q_0), b))$$
$$= \epsilon\_cl(q_3)$$
$$= \{ q_3 \}$$

$$\delta'(q_1, a) = \epsilon\_cl(\delta(\epsilon\_cl(q_1), a))$$
$$= \epsilon\_cl(q_2)$$
$$= \{ q_2 \}$$
$$\delta'(q_1, b) = \epsilon\_cl(\delta(\epsilon\_cl(q_1), b))$$
$$= \epsilon\_cl(q_2)$$
$$= \{ q_3 \}$$

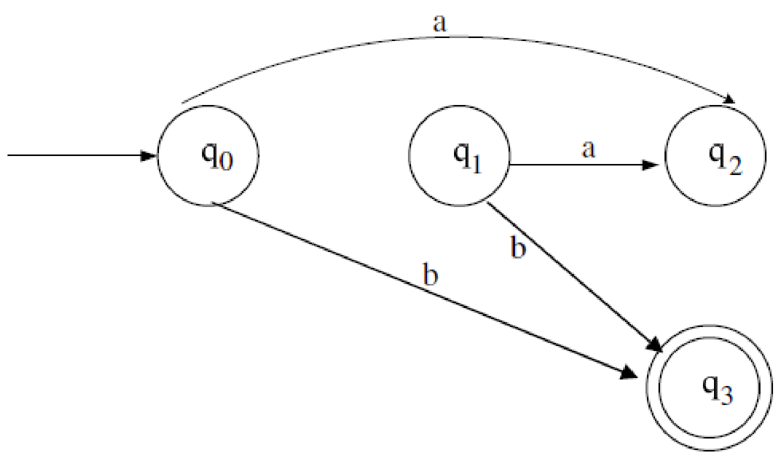
$$\delta'(q_2, a) = \epsilon\_cl(\delta(\epsilon\_cl(q_2), a))$$
$$= \epsilon\_cl(\varnothing)$$
$$= \varnothing$$
$$\delta'(q_2, b) = \epsilon\_cl(\delta(\epsilon\_cl(q_2), b))$$
$$= \epsilon\_cl(\varnothing)$$
$$= \varnothing$$

$$\delta'(q_3, a) = \epsilon\_cl(\delta(\epsilon\_cl(q_3), a))$$
$$= \epsilon\_cl(\varnothing)$$
$$= \varnothing$$

* Buatlah tabel transisi dari fungsi transisi yang telah dibuat pada langkah sebelumnya.

δ	a	b
q_0	$\{q_2\}$	$\{q_3\}$
q_1	$\{q_2\}$	$\{q_3\}$
q_2	Ø	Ø
q_3	Ø	Ø

* Kemudian, tentukanlah himpunan state akhir untuk NFA tanpa ϵ – move ini. Himpunan state akhir semula adalah {q3}. Karena tidak ada state lain yang ϵ – closurenya memuat q3 , maka himpunan state akhir sekarang tetap {q3}. Sehingga diperoleh diagram transisi sebagai berikut.



FINITE STATE TRANSDUCER (FST)

Input : Input berupa deretan karakter dan string
Output : Output berupa diterima atau tidak diterimanya suatu inputan

A. JENIS FINITE STATE TRANSDUCER (FST)

- Moore Machine
- Mealy Machine

B. Karakteristik FST

- FST terdiri dari dari sejumlah state tanpa final state
- (Mealy) -> State tersebut dihubungkan oleh transisi yang diberi label pasangan input/output
- (Moore) -> Output berasosiasi dengan state
- FST dimulai dengan initial state yang ditentukan dan melompat ke state yang berbeda, tergantung pada nilai input, sambil menghasilkan output sesuai dengan tabel transisinya

C. Moore Machine

Pada mesin moore, output akan berasosiasi state.
Didefenisikan dengan 6 tupel
•Q =Himpunan state
•£ =Himpunan symbol input
• =Fungsi transisi
•S=State awal S, dimana S Q
•Δ=Himpunan output
• =fungsi output untuk setiap state

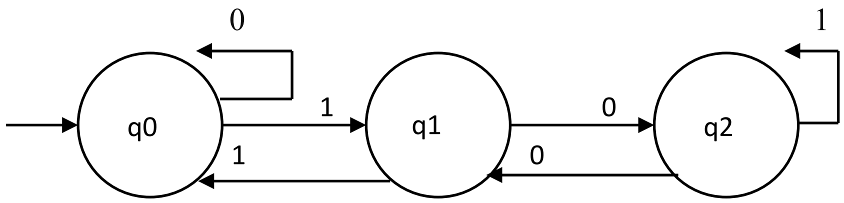
Contoh 1
Kita akan mencari nilai sisa pembagian (modulus) suatu bilangan dengan 3. Dimana input dinyatakan dalam biner.

Konfigurasi :
Q = {q0, q1, q2}
Σ = {0, 1} (input dalam biner)
S = q0
Δ = {0, 1, 2} (untuk outputnya pada kasus mod dengan 3 maka sisanya kemungkinan adalah 0,1,2)

λ(q0) = 0 λ(q1) = 1

λ(q2) = 2

Gambar Mesin Moore untuk modulus 3 :



Pembuktian :

- 5 mod 3 = ?

input 5 dalam biner 101

bila kita masukkan 101 kedalam mesin, urutan state yang dicapai adalah :

q0, q1, q2, q2

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 kedalam mesin, urutan state yang dicapai adalah : q0, q1, q2, q2, q1

State terakhir yang dicapai adalah q1, λ(q1) = 1 Maka 10 mod 3 = 1

- 12 mod 3 = ?

input 12 dalam biner 1100 bila kita masukkan 1100 kedalam mesin, urutan state yang dicapai adalah : q0, q1, q0, q0

State terakhir yang dicapai adalah q0, λ(q0) = 0 Maka 12 mod 3 = 0

D. Contaxt Free Grammar

Tata Bahasa Bebas Komteks atau CFG

•CFG atau Contaxt Free Grammar adalah tata bahasa formal dimana setiap aturan produksi adalah dalam bentuk a→B, dimana a adalah pemproduksi, dan B adalah hasil produkai

*Batasan Aturan Produksi

•Ruas kiri adalah sebuah simbol variabel

•Ruas kanan bisa berupa terminal, variable aturan E

•a E N (Non-terminal)

•B E (TUN) (Terminal atau Non-terminal)

•B boleh berisi E

*Contoh aturan Produksi CFG

•X→bY|Za

•Y→aY|B

•Z→bZ|E

•B→CDefg
•D→BcDe

E. CFG VS RE

•Keduanya merupakan suatu cara untuk menunjukan bagaimana menghasilkan suatu untai-untai dalam sebuah bahasa
•Pada saat menurunkan suatu string, symbol-symbol variable akan mewakili bagian-bagian yang belum diturunkan dari string tersebut
•Pada ER, bagian yang belum diturunlan selalu berada diujung
•PadavCFG,bagian yang belum diturunlan bisa berada dimana saja

Contoh aturan produksi ER
S→aE
E→A|B
A→aA|b
B→aE|b

Contoh aturan produksi CFG
X→bY|Za
Y→aY|B
Z→bZ|E
B→CDefg

*Parsing
Pohon atau tree adalah sebuah graph terhubung tidak sirkuler yang memiliki satu simpul(node)/ vertex yang disebut akar (root) dan dari situ kita memiliki lintasan setiap simpul
•Derivation Tree (Pohon Penurunan)
Derivation tree berguna untuk menggambarkan bagaimana memperoleh suatu string dengan cara menurunkan symbol-symbol variable menjadi symbol-symbol terminal. Hingga tidak ada lagi symbol variable yang bisa diturunkan
•Contoh:pohon penurunan"aabbb"
Aturan produksi
S→AB
A→aA|a
B→bB|b

Hasil penurunan
S→AB→aAV→aaB→aabB→aabb

•Proses penuruan / parsing
1.Lefmost Derivation
Simbol variabel yang paling kiri diturunkan terlebih dahulu
2.Reightmos

Simbol variable yang diturunkan terlebih dahulu
•Contoh proses penurunn / parsing
Lefmost Derivation
S→aAS→asbAS→aabAS→aabbaS→aabbaa
Reightmost Derivation
S→aAS→aAa→aSbAa→aSbbaa→aabbaa

Ambigiutas
Ambigius adalah kedwiartian atau maksa ganda
Ambigiutas terjadi jika terdapat lebih dari satu pohon penurunan yang berbeda untuk memperoleh suatu untai
Contoy ambigius
Aturan produksi
S→A|B
A→a
B→a
String yang dicari "a"
Cara 1
S→A→a
Cara 2
S→B→a
Aturan produksi
S→SbS
S→ScS
S→a
String yang dicari'abaca"
Cara
S→SbS→SbScS→SbSca→Sbaca→abaca

Kesimpulan
Ambiguitas dapat menimbulkan masalah pada bahasa-bahasa tertentu,baik pada bahasa alami maupun pada bahasa pemrograman
Bila suatu struktur bahasa memiliki lebih dari suatu dekomposisi,dan susunannya akan menentukan arti, mla artinya menjadi ambigu

PENGANTAR ATURAN PRODUKSI REKURSIF KIRI

Aturan Produksi yang rekursif memiliki ruas kanan (hasil produksi) yang memuat simbol variabel pada ruas kiri. Terdapat rekursif kanan dan rekursif kiri. Sebuah aturan produksi dalam bentuk:

A ◊ βA

* Merupakan aturan produksi yang rekursif kanan:

β(V∪T)*

atau kumpulan symbol variabel dan terminal

* Contoh aturan produksi yang rekursif kanan:

S ◊ dS

B ◊ adB

Aturan Produksi Rekursif Kiri

* Sebuah aturan produksi dalam bentuk:

A ◊ Aβ

* Merupakan aturan produksi yang rekursif kiri,
  contohnya:
  S ◊ Sd
  B ◊ Bad

* Produksi yang rekursif kanan menyebabkan pohon penurunan tumbuh ke kanan, sebaliknya produksi yang rekursif ke kiri menyebabkan pohon penurunan tumbuh ke kiri. Bisa dilihat pada pohon penurunan dari tata bahasa bebas konteks dengan aturan produksi:

S ◊ aAc

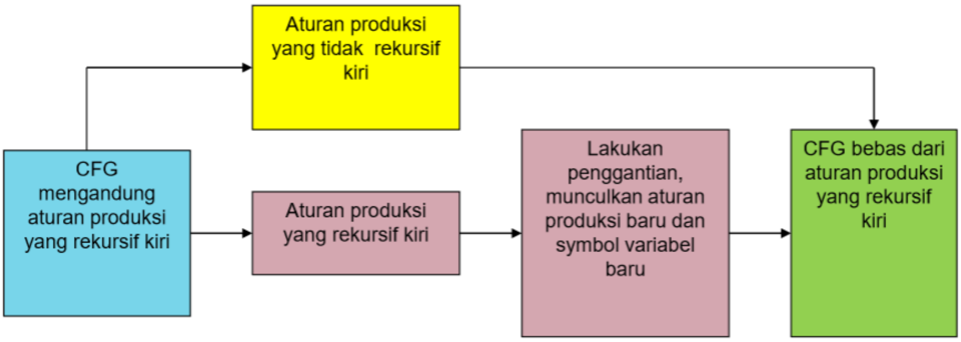
A ◊ Ab | ε

A. Penghilangan Rekursif Kiri

1. Tujuan Rekursif Kiri Dihapus

- Dalam banyak penerapan tata bahasa, rekursif kiri tak diinginkan.
- Untuk menghindari penurunan yang bisa mengakibatkan loop hilangkan sifat rekursif kiri dari aturan produksi
- Penghilangan rekursif kiri memungkinkan suatu tata bahasa bebas konteks diubah ke dalam bentuk normal Greibach

2. Tahapan Penghilangan Rekursif Kiri



a) Pisahkan aturan produksi yang rekursif kiri dan yang tidak
   - ❖ Aturan produksi yang rekursif kiri :
     $A \rightarrow A\alpha1 \mid A\alpha2 \mid A\alpha3 \mid \dots \mid A\alpha n$
   - ❖ Aturan produksi yang tidak rekursif kiri :
     $A \rightarrow \beta1 \mid \beta2 \mid \beta3 \mid \dots \mid \beta m$

b) **Tentukan simbol-simbol α1, α2, α3, … , αn dan β1, β2, β3, … , βm** dari setiap aturan produksi yang memiliki simbol ruas kiri yang sama

c) Lakukan penggantian aturan produksi yang rekursif kiri menjadi sebagai berikut:
   $A \rightarrow \beta1\ Z \mid \beta2\ Z \mid \beta3\ Z \mid \dots \mid \beta m\ Z$ $Z \rightarrow \alpha1 \mid \alpha2 \mid \alpha3 \mid \dots , \alpha n$ $Z \rightarrow \beta1 \mid \beta2 \mid \beta3 \mid \dots \mid \beta m$
   Penggantian diatas dilakukan untuk setiap aturan produksi dengan symbol ruas kiri yang sama. Bisa muncul symbol variabel baru Z1, Z2 dan seterusnya sesuai banyaknya variabel yang menghasilkan produksi yang rekursif kiri.

d) Hasil akhir berupa aturan produksi pengganti ditambah dengan aturan produksi semula yang tidak rekursif kiri

 # Contoh Kasus Penghilangan Rekursif Kiri

- Contoh 12.1
  Hilangkan rekursif kiri dari aturan produksi berikut!

S $\rightarrow$Sab | aSc | dd | ff | Sbd

Penyelesaian:

Aturan produksi rekursif kiri : S $\rightarrow$Sab | Sbd $\alpha 1$ = ab ; $\alpha 2$ = bd

Aturan produksi tidak rekursif kiri : S $\rightarrow$ aSc | dd | ff $\beta 1$ = aSc ; $\beta 2$ = dd ; $\beta 3$ = ff

Aturan produksi rekursif kiri, S $\rightarrow$Sab | Sbd digantikan oleh: S $\rightarrow$ aScZ1 | dd Z1 | ff Z1 Z1 $\rightarrow$ ab | bd Z1 $\rightarrow$ abZ1 | bdZ1

Hasil akhir setelah penghilangan rekursif kiri adalah : S $\rightarrow$ aSc | dd | ff

- Contoh 12.2

  Hilangkan rekursif kiri dari aturan produksi berikut!

  S $\rightarrow$Sab | Sb | cA A $\rightarrow$Aa | a | bd

  Penyelesaian:

  - Aturan produksi rekursif kiri, S $\rightarrow$Sab | Sb $\alpha 1$ = ab ; $\alpha 2$ = b
  - Aturan produksi rekursif kiri, A $\rightarrow$Aa $\alpha 1$ = a
  - Aturan produksi tidak rekursif kiri, S $\rightarrow$ cA $\beta 1$ = cA
  - Aturan produksi tidak rekursif kiri, A $\rightarrow$ a | bd $\beta 1$ = a ; $\beta 2$ = bd

Pergantian aturan produksi rekursif kiri:

S $\rightarrow$Sab | Sb digantikan oleh:

- S $\rightarrow$ caZ1
- Z1 $\rightarrow$ ab | b
- Z1 $\rightarrow$ abZ1 | bZ1

A $\rightarrow$Aa digantikan oleh :

- A $\rightarrow$ aZ2 | bdZ2
- Z2 $\rightarrow$ a
- Z2 $\rightarrow$ aZ2

Hasil akhir setelah penghilangan rekursif kiri adalah:

S $\rightarrow$ cA

A $\rightarrow$ a | bd

S $\rightarrow$ caZ1

Z1 $\rightarrow$ ab | b

Z1 $\rightarrow$ abZ1 | bZ1

A $\rightarrow$ aZ2 | bdZ2

Z2 $\rightarrow$ a

Z2 $\rightarrow$ aZ2

PENYEDERHANAAN ATURAN PRODUKSI CONTEXT FREE GRAMMAR

A. Tujuan Penyederhanaan

Melakukan pembatasan, sehingga tidak dihasilkan pohon penurunan yang memiliki kerumitan yang tak perlu atau aturan produksi yang tidak berarti

Misal

Diketahui suatu tata bahasa konteks:

S -> AB | a

A -> a

Kelemahan : Aturan produksi S -> AB tidak berarti karena B tidak memiliki penurunan

B. Teknik Penyederhanaan

a. Penghilangan Produksi ε

Produksi ε (Empty) adalah produksi dalam bentuk α -> ε atau bisa dianggap sebagai produksi kosong. Penghilangan produksi ε dilakukan dengan melakukan penggantian produksi yang memuat variable yang manuju produksi ε, atau biasa disebut nullable.

Contoh 1 :

Diketahui tata bahasa bebas konteks sebagai berikut :

S ->dA | Bd

A ->bc

A ->ε

B ->c

Sehingga aturan produksi setelah penyederhanaan : S ->dA | d | Bd

A ->bc

B ->c

Contoh 2 :

S -> bcAd

A ->ε

Pada kasus 1, A nullable serta A ->ε merupakan satu-satunya produksi dari A maka variable A bias ditiadakan.

Maka hasil penyederhanaan adalah : S ->bcd

Kasus 2 S ->bcAd

A ->bd | ε

Pada kasus 2, A nullable, tapi A ->ε bukan satu-satunya produksi dari A. Maka hasil penyederhanaan adalah : S ->bcAd | bcd A ->bd

b. Penghilangan Produksi Unit

Produksi unit adalah produksi yang ruas kiri dan kanan aturan produksinya hanya berupa satu simbol variable. (α = 1 N/V dan β = 1 N/V)

Dengan adanya bentuk produksi unit ini membuat tata bahasa memiliki kerumitan yang tidak perlu atau menambah panjang penurunan.

Penyederhanaan ini dilakukan dengan melakukan penggantian aturan produksi unit.

Contoh 1 :

Diketahui tata bahasa bebas konteks sebagai berikut :

S ->Sb
S ->C
C ->D
C ->ef
D ->dd
Sehingga aturan produksi setelah penyederhanaan :
S ->Sb | dd | ef
C ->dd | ef
D ->dd

Contoh 2 :
Diketahui tata bahasa bebas konteks sebagai berikut :
S ->A
S ->Aa
A ->B
B ->C
B ->b
C ->D
C ->ab
D ->b
Sehingga aturan produksi setelah penyederhanaan :
S ->A => S ->ab | b
S ->Aa
A ->B => A ->ab | b B ->ab B ->b
C ->b
C ->ab
D ->b

c. Penghilang Produksi Useless

Produksi yang memuat simbol variable yang tidak memiliki penurunan yang akan menghasilkan terminal-terminal seluruhnya (menuju terminal), produksi ini tidak berguna karena bila diturunkan tidak akan pernah selesai (masih ada simbol variable tersisa).

Produksi yang tidak akan pernah dicapai dengan penurunan apapun dari simbol awal, sehingga produksi itu redundan (berlebih).

Contoh 1 :
Diketahui tata bahasa bebas konteks sebagai berikut
S ->aSa | Abd | Bde
A ->Ada B ->BBB | a
Maka tata bahasa hasil penyederhanaan adalah :
S ->aSa | Bde
B ->BBB | a
Contoh 2 :
Diketahui tata bahasa bebas konteks :
S ->Aa | B

A ->ab | D
B ->b | E
C ->bb
E ->aEa
Maka tata bahasa hasil penyederhana an menjadi :
S ->Aa | B
A ->ab
B ->b

1. Gabungan Useless, Unit & ε

Urutannya sebagai berikut :

a. Hilangkan produksi ε
b. Hilangkan produksi unit
c. Hilangkan produksi useless

Contoh 1 :
Hilangkan produksi useless, unit dan empty dari tata bahasa bebas konteks berikut :
S ->a | aA | B | C
A ->aB | ε
B ->Aa
C ->cCD
D ->ddd

a. Hilangkan produksi ε

S ->a | aA | B | C
A ->aB | ε
B ->Aa
C ->cCD
D ->ddd
Hasilnya
S ->a | aA | B | C
A ->aB
B ->Aa | a
C ->cCD
D ->ddd

b. Hilangkan produksi unit

S ->a | aA| B | C
A ->aB
B ->Aa | a
C ->cCD
D ->ddd
Hasilnya
S ->a | aA| Aa | cCD
A ->aB
B ->Aa | a
C ->cCD
D ->ddd

c. Hilangkan produksi useless

S ->a | aA| Aa | ~~cCD~~

A ->aB

B ->Aa | a

~~C ->cCD~~

~~D ->ddd~~

Hasilnya

S ->a | aA| Aa

A ->aB

B ->Aa | a

CHOMSKY NORMAL FORM (CNF)

Merupakan salah satu bentuk normal yang sangat berguna untuk tata Bahasa bebas konteks (CFG)

Chomsky Normal Form (CNF) dapat dibentuk dari CFG yang telah mengalami penyederhanaan yaitu penghilangan produksi useless, unit dan kosong.

B. Bentuk Normal Chomsky

Syarat Konversi CFD ke CNF

Tidak memiliki produksi useless

Tidak memiliki Produksi unit

Tidak memiliki Produksi ε (Kosong)

Aturan Produksi CNF

Ruas kanan hanya boleh berupa sebuah simbol terminal atau dua buah simbol variable.

Jika terdapat lebih dari satu simbol terminal maka harus dilakukan penggantian dan juga jika terdapat lebih dari dua buah simbol variable maka harus dilakukan perubahan.

$\alpha \to \beta$

α = 1 Non-Terminal

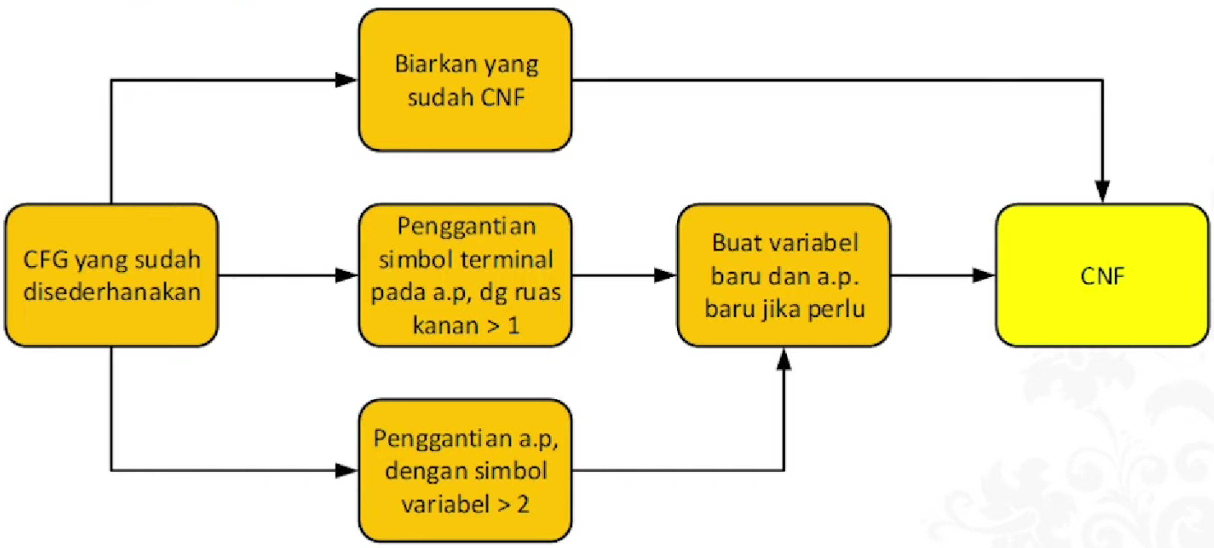
β = 1 Terminal atau 2 Non-Terminal

C. Pembentukan Bentuk Normal Chomsky

Langkah Secara Umum

a. Biarakan aturan produksi yang telah dalalm bentuk CNF
b. Lakukan penggantian aturan produksi yang ruas kanannya memuat symbol terminal dan panjang ruas kanan >1
c. Lakukan penggantian atiran produksi yang ruas kanannya memuat >2symbol variable
d. Penggantian-penggantian tersebut bisa dilakukan berkali kali sampai akhirnya semua aturan produksi dalam bentuk normal Chomsky
e. Selama dilakukan penggantian, kemungkinan kita akan memperoleh aturan-aturan produksi baru, dna juga memunculkan symbol-simbol variable baru.

Tahapan Pembentukan CNF



Contoh 1 :

Perhatikan aturan produksi CFG berikut, diasumsikan telah disederhanakan :

S → bA | aB

A → bAA | aS | a

B → aBB | bS | b

Ubahlah ke dalam bentuk normal Chomsky !!!

Jawab :

Aturan produksi yang sudah dalam bentuk normal Chomsky:

A → aB → bb.

Dilakukan penggantian aturan produksi yang belum bentuk normal Chomsky('=>' bisa dibaca berubah menjadi):

S → bA => S → P1A

S → aB => S → P2B

A → bAA => A → P1AA => A → P1P3

A → aS => A → P2S

B → aBB => B → P2BB => B → P2P4

B → bS => B → P1Sc.

Terbentuk aturan produksi dan simbol variabel baru:

P1 → b

P2 → a

P3 → AA

P4 → BBd.

Hasil akhir aturan produksi dalam bentuk normal Chomsky :

A → a

B → b

S → P1A

S → P2B

A → P1P3

A → P2S

B → P2P4

B → P1S

P1 → b

P2 → a

P3 → AA

P4 → BB

Contoh 2 :

Berikut adalah tata bahasa bebas konteks yang sudah disederhanakan.

S -> aB | CA

A -> a | bc B -> BC | Ab

C -> aB | b Dari aturan produksi diatas, aturan produksi yang sudah dalam bentuk normal Chomsky (CNF) adalah: S -> CA

A -> a

B -> BC

C -> b

Aturan produksi yang belum dalam CNF adalah:
S -> aB => S -> P1B
A -> bc =>A -> P2P3
B -> Ab => B -> AP2
C -> aB =>C -> P1B
Terbentu aturan produksi baru:
P1 -> a
P2 -> b
P3 -> c
Hasil ahir aturan produksi dalam CNF adalah:
S -> CA
A -> a
B -> BC
C -> b
S -> P1B
A -> P2P3
B -> AP2
C -> P1B
P1 -> a
P2 -> b
P3 -> c

- Algoritma CYK untuk CFG

Algoritma the Cocke-Younger-Kasami algorithm (CYK) merupakan algoritma parsing dan keanggotaan (Membership) untuk tata Bahasa Bebas Konteks. Algoritma ini diciptakan oleh J.Cocke.DH.Younger. dan T.Kasami. Syarat untuk menggunakan algoritma ini adalah tata bahasa harus sudah dalam bentuk Normal Chomsky
Simbol Algoritma CYK
n = panjang string yang akan diperiksa, misalnya 'ada', n = ladal =3
I menyatakan kolom ke-
J menyatakan baris ke-
Tahapan no (2) dan (3) untuk mengisi tabel baris pertama kolom 1-n
No (4), iterasi dari baris ke-2 sampai no (4)
No (5), iterasi untuk mengisi kolom 1 sampai (n-baris+1) pada suatu baris
No(7), inisialisasi Vij dengan Ø
No (8) dan (9), iterasi untuk memeriksa mana saja yang menjadi anggota Vij
Contoh 1 :
Diketahui tata bahasa bebas konteks:
S -> AB | BC
A ->BA | a
B -> CC | b
C -> AB | a
Periksa, apakah untai ‘baaba’ termasuk ke dalam bahasa tersebut.
Penyelesaian:
Syarat agar sebuah untai termasuk ke dalam tata bahasa tertentu adalah V1n

harus memuat simbol awal
i -> (1 – 5)
j -> 1 – 5)

Hasil pada tabel

	b	a	a	b	a
			i		
	1	2	3	4	5
1	B	A,C	A,C	B	A,C
2	S,A	B	S,C	S,A	
3	Ø	B	B		
4	Ø	S, A, C			
5	S, A, C				