**MODUL PERKULIAHAN**

**EDISI 1 (Kurikulum 2010)**

**Teori Bahasa dan Automata**



Tim Penulis :

Mira Kania Sabariah, M.T.

Utami Dewi Widianti,S.Kom

**PROGRAM STUDI TEKNIK INFORMATIKA**

**UNIVERSITAS KOMPUTER INDONESIA**

**BANDUNG**

**2010**

**DAFTAR ISI**

Daftar Isi …………………………………………………………………………………………………………………. 2

Bab 1 Pendahuluan………………………………………………………………………...……………………………… 3

Bab 2 Pengelompokan Bahasa Sesuai Hirarky Chomsky………………………………………………… 15

Bab 3 Finite State Automata..………………………………………………………………………………………… 19

Bab 4 Ekspresi Regular dan Hubungannya dengan FSA………………………………………………….. 28

Bab 5 Aturan Produksi untuk Suatu FSA dan FSA dengan Output…………………………………... 31

Bab 6 Penyederhanaan Tata Bahasa Bebas Konteks.…………………………………………………….. 42

Bab 7 Bentuk Normal Chomsky…..………………………………………………………………………………… 53

Bab 8 Penghilangan Left Linear Grammer…………………………………………………………………….. 59

Daftar Pustaka ……………………………………………………………………………………………………………….. 65

**1**

 **Pendahuluan**

|  |
| --- |
| JUMLAH PERTEMUAN : 1 PERTEMUANTUJUAN INSTRUKSIONAL KHUSUS :1. Mengerti mengenai konsep dasar tata bahasa dan automata,sehingga dapat memahami secara definitif mengenai bahasa, tata bahasa dan automata.
2. Mengerti prinsip kerja teori himpunan,graph dan tree secara individual
 |

**Materi :**

**Pendahuluan**

* Komputer digunakan sebagai alat bantu untuk menyelesaikan pekerjaan(*task*). Dua pertanyaan yang seringkali muncul sehubungan dengan pekerjaan tersebut adalah :
	1. Dapatkah pekerjaan tersebut dilaksanakan oleh komputer?
	2. Bila jawabannya dapat, bagaimana cara komputer melaksanakan pekerjaan tersebut?

Model komputasi berguna untuk menjawab kedua pertanyaan tersebut diatas.

* Terdapat tiga jenis struktur yang digunakan dalam model komputasi :
1. Grammar (tata bahasa)

Digunakan untuk membangkitkan kalimat dari suatu bahasa dan untuk memeriksa apakah suatu kalimat termasuk di dalam bahasa tersebut.

1. *Finite State Machine*(FSM)

Digunakan untuk mengenali kalimat dari suatu bahasa dan untuk menghasilkan keluaran sesuai dengan spesifikasi yang digunakan.

Konsep tatabahasa dan FSM keduanya saling berkaitan, dan antara keduanya dapat saling ditransformasikan satu sama lain.

1. Mesin Turing

Merupakan pengenal bahasa yang lebih kompleks, yang dalam pengenalannya mesin ini dilengkapi dengan memori. Mesin turing dapat mengenali kalimat yang tidak dapat dikenali oleh FSM sekalipun. Mesin turing adalah model komputasi yang paling umum serta dapat melakukan pekerjaan apa saja yang dapat dilaksanakan oleh komputer.

**I.1. Bahasa Alami dan Bahasa Formal**

* Menurut kamus  *Webster*, bahasa adalah ”*the body of words and methods of combining words used and understood by a considerable community*”.
* Sedangkan menurut Kamus Besar Bahasa Indonesia(KBBI), bahasa adalah :
1. Sistem lambang bunyi berartikulasi (yang dihasilkan oleh alat-alat ucap) yang bersifat sewenang-wenang dan konvensional yang dipakai sebagai alat komunikasi untuk melahirkan perasaan dan pikiran.
2. Perkataan-perkataan yang dipakai oleh suatu bangsa.
* Menurut kedua definisi diatas, ungkapan dari suatu bahasa adalah kalimat atau perkataan. Kalimat dalam sebuah bahasa dibentuk dengan menggabungkan satu atau lebih kata-kata.
* Dalam pembentukan kalimat, perhatian tertuju pada sintaks dan bentuk kalimat, bukan pada semantik atau makna kalimatnya. Sebagai contoh :

**Elang menyambar ayam**

**Adik makan dengan lahap**

Adalah kalimat yang sah sesuai dengan kaidah tata bahasa dari Bahasa Indonesia. Contoh kalimat berikut :

**Rudi menggigit anjing**

Juga benar secara sintaks meskipun maknanya secara sense tidak ada. Sedangkan kalimat berikut :

**dengan cepat ayah matematika menutup**

bukan kalimat yang sah secara sintaks karena tidak mengikuti kaidah tata bahasa.

* Bahasa Alami adalah bahasa komunikasi yang digunakan oleh manusia, yaitu bahasa ucap seperti Bahasa Indonesia, Inggris, dll. Sintaks bahasa alami sangat rumit dan kita tidak mungkin menspesifikasikan semua aturan sintaksnya.
* Bahasa Formal adalah bahasa yang kaidah sintaksnya dapat dispesifikasikan secara matematis dengan baik. Kaidah bahasa dalam teori bahasa formal tidak hanya bermanfaat untuk studi linguistik bahasa alami seperti penterjemahan secara otomatis dari suatu bahasa ke bahasa lain, tetapi juga berguna untuk studi bahasa pemrograman.

**I.2. Alfabet, *String*, dan Bahasa**

* Di dalam teori bahasa formal, bahasa didefinisikan secara matematis. Sebelum membahas lebih lanjut mengenai definisi bahasa, kita harus mengerti beberapa terminologi(istilah) yang digunakan disini.
* **Alfabet** (adalah *vocabulary*) adalah himpunan terbatas simbol.

**Contoh :**

* 1. Alfabet Latin {A, B, C, .., Z)
	2. Alfabet Yunani {α,β,γ,..,ω}
	3. Alfabet Biner, {0,1}

* **String** adalah barisan(*sequence*) yang disusun oleh simbol-simbol alfabet. Sebuah *string* dengan panjang n (n >= 1) yang dibentuk dari alfabet A disusun oleh barisan n simbol :

**a1a2a3,..an, aI Є A**

istilah lain untuk *string* adalah kalimat atau *word*.

* + Panjang *string x* dituliskan sebagai ІxІ. *String* kosong (*null* *string* atau *empty sentence* (ε)) adalah barisan simbol dengan panjang 0 dan tidak berisi simbol apapun.
	+ Jika A adalah alfabet, maka An menyatakan himpunan semua *string* dengan panjang n yang dibentuk dari himpunan A.
	+ A\* adalah himpunan semua rangkaian simbol dari himpunan A yang terdiri dari 0 simbol (*string* kosong), satu simbol, dua simbol, tiga simbol,.., atau dinyatakan sebagai :

**A\* = A0 U A1 U A2 U...**

Himpunan A\* disebut ***transitive closure* atau *Kleen Closure*.**

* + A+  ( *positive closure* ) untuk menyatakan himpunan V\* - Є. Jadi, A+ diperoleh dari A\* dengan mengeluarkan string kosong, atau :

**A+ = A1 U A2 U A3 U...**

* + Contoh :

Misalkan A={0,1} adalah sebuah alphabet yang disusun dari simbol 0 dan 1. Maka A\* :

A0={ ε }

 A1={ 0,1 }

 A2={ 00,01,10,11 }, dan seterusnya

 Apabila A+ :

 A1={ 0,1 }

 A2={ 00,01,10,11 }, dan seterusnya

* **Definisi bahasa(*language*)**

Jika A adalah alfabet, maka bahasa (pada alfabet A) adalah bagian dari dari himpunan A\*. Sebagai contoh, misalkan {a,b,c}, maka himpunan-himpunan berikut ini merupakan bahasa pada alfabet A :

L1 = {a, aa, bc, ac, abc, cab}

L2 = {aba, aabaa}

L3 = { }

L4 = {aicbi І i>=1}

(Keterangan : pada L4 ai berarti barisan *i* buah *a*. Jadi aicbi berarti kalimat yang disusun oleh *i* buah *a* diikuti dengan *c*, diikuti dengan *i* buah *b*)

I**.3 Tata Bahasa**

* Jika diberikan spesifikasi suatu bahasa, ada dua masalah utama yang berkaitan dengan masalah tersebut :
1. Bagaimana membangkitkan satu atau lebih kalimat didalam bahasa tersebut ?
	* Masalah pembangkitan (*Generation*)
2. Bagaimana menentukan apakah suatu kalimat termasuk di dalam bahasa tersebut atau tidak?
* Masalah Pengenalan (*Recognition*)
* Aturan yang digunakan untuk membangkitkan atau mengenali kalimat di dalam suatu bahasa dinamakan tata bahasa(*grammar*).
* **Contoh :** Tata bahasa dibawah ini adalah sebagian dari tata bahasa untuk membangkitkan atau mengenali kalimat di dalam Bahasa Inggris :
1. Sebuah **sentence** dibentuk dari **noun phrase** diikuti dengan **verb phrase.**
2. Sebuah **sentence** dibentuk dari **noun phrase** diikuti dengan **verb phrase** dan **noun phrase.**
3. dll
* Kita dapat menuliskan semua kaidah tata bahasa di atas secara ringkas sebagai berikut :

<sentence> → <noun phrase> <verb phrase>

<sentence> → <noun phrase> <verb phrase> <noun phrase>

<noun phrase> → <adjective><noun phrase>

<noun phrase> → <adjective><singular noun >

<verb phrase > → <singular verb> <adverb>

<verb phrase > → <singular verb>

<adjective> → *the*

<adjective> → *a*

<adjective> → *little*

<singular noun> → *boy*

<singular noun> → *dog*

<singular verb> → *runs*

<singular verb> → *bites*

<adverb> → *quickly*

* Tanda panah (→) pada kaidah di atas menyatakan bahwa item di sebelah kiri panah dapat diganti dengan item di sebelah kanan panah.
* Proses pembangkitan kalimat *the little boy runs quickly* dapat diturunkan sebagai berikut :

<sentence> → <noun phrase> <verb phrase>

* + <adjective> <noun phrase> <verb phrase>
	+ *the* <noun phrase> <verb phrase>
	+ *the* <adjective> <singular noun > <verb phrase>
	+ *the little* <singular noun > <verb phrase>
	+ *the little boy* <verb phrase>
	+ *the little boy*  <singular verb> <adverb>
	+ *the little boy runs* <adverb>
	+ *the little boy runs quickly*
* Proses pembangkitan kalimat diatas juga dapat digambarkan dalam suatu diagram pohon- yang disebut *pohon parsing*.
* Unsur-unsur tata bahasa :

Secara formal, suatu tata bahasa terdiri dari empat unsur :

1. Himpunan berhingga **terminal T**
2. Himpunan berhingga **non –terminal N**
3. Himpunan berhingga **aturan produksi P**
4. **Simbol awal S є N**
* Penjelasan :
1. Simbol terminal didalam *T* adalah symbol yang dapat digunakan untuk membuat kalimat/string di dalam bahasa itu. Simbol terminal biasanya dilambangkan dengan huruf kecil, seperti a,b,c,… atau angka 0,1,..

Contoh : *T* = {a,b,c}

1. Simbol non terminal di dalam *N* adalah lambang antara yang digunakan untuk mendeskripsikan struktur kalimat. Simbol non-terminal biasanya dilambangkan dengan hruf besar, seperti A,B,C,..

Contoh : *N* = {S,A,B}

1. Produksi adalah kaidah/aturan tata bahasa yang mengatur cara pembentukan kalimat di dalam bahasa.

Setiap aturan dituliskan dalam bentuk :

**α→β**

yang dalam hal ini, **α dan β**  adalah string yang dibentuk dari himpunan *T U N*, dan **α**  harus berisi paling sedikit satu simbol non-terminal.

1. Simbol awal adalah simbol non-terminal khusus yang mengawali pembangkitan kalimat di dalam bahasa tersebut.
* Keempat unsur tata bahasa tersebut sering dilambangkan dengan :

***G* = (*T,N,P,S*)**

Bahasa yang dihasilkan oleh *G*  ditulis sebagai L(*G*), yaitu himpunan *string* yang dapat diturunkan dan simbol awal *S* dengan menerapkan aturan produksi yang terdapat di dalam *P.*

* Contoh :

Tinjau tata bahasa ***G* = (*T,N,P,S*)**, dengan ***T={a,b}, N={S,A}, P={S*→aA**,***S*→b, *A*→aa*}***

Apakah *L(G)* dari Tata bahasa *G* ?

Dengan mencoba semua aturan produksi dari simbol awal *S* sebagai berikut :

 ( i ) ***S* → aA**

 **→ aaa**

 ( i ) ***S* → b**

maka hanya terdapat dua *string* yang dapat dibangkitkan, yaitu *b* dan *aaa*, tidak ada yang lain. Jadi *L(G) = {b, aaa}*.

**1.4 Kelas Tata Bahasa dan kelasa Bahasa**

* Berdasarkan bentuk aturan produksi α→β , tata bahasa formal dibagi ke dalam empat kelas (jenis). Keempat kelas itu disebut juga dengan *Hirarki Chomsky*, sebagaimana diklasifikasikan oleh Noam Chomsky.
* Kelas bahasa ditentukan oleh kelas bahasanya. Jadi, bahasa kelas *i* adalah bahasa yang dapat dispesifikasi oleh tata bahasa kelas *i*.
* Di dalam pembahasan berikut, kita menggunakan A dan B untuk menyatakan non-terminal, a dan b untuk menyatakan terminal, dan α dan β untuk menyatakan string yang dibentuk dari simbol terminal dan simbol non terminal.
* Keempat kelas tata bahasa tersebut adalah sebagai berikut :
	1. **Kelas tata bahasa regular (regular grammar)**
		+ Merupakan kelas tata bahasa jenis-3. Kelas ini dicirikan jika semua aturan produksinya berbentuk :

α→β

dimana : α Є N, β Є T U N dengan ketentuan sebagai berupa terminal atau sebuah terminal diikuti dengan sebuah non-terminal. Contoh : A → a ; A → aA

* + - Bahasa yang dispesifikasi oleh tata bahasa ini disebut **bahasa regular** (regular language)
		- Mesin yang mengenal **bahasa regular** adalah ***Finite State Automata.***
	1. **Kelas tata bahasa bebas konteks (context free grammar)**
		+ Merupakan kelas tata bahasa jenis-2. Kelas ini dicirikan jika semua aturan produksinya berbentuk :

α→β

dimana :

α Є N, dengan ketentuan sebuah non-terminal

β Є T U N.

ІαІ <= ІβІ

* + - Contoh : A → aBB ; A → aA ; B→bA
		- Mesin yang mengenal **bahasa bebas konteks** adalah ***Push Down Automata (PDA).***
	1. **Kelas tata bahasa peka-konteks (context sensitive grammar)**
		+ Merupakan kelas tata bahasa jenis-1. Kelas ini dicirikan jika semua aturan produksinya berbentuk :

α→β

dimana :

α Є T U N

β Є T U N

ІαІ <= ІβІ

* + - Contoh : S → aSBC ; S → aBC ; CB→BC ; bB→bb ; bC→bc; cC→cc
		- Mesin yang mengenal **bahasa peka-konteks** adalah ***Linear Bounded Automata (LBA).***
	1. **Kelas tata bahasa tanpa-pembatasan (*urestricted grammar*)**

Merupakan kelas tata bahasa jenis-0. Kelas ini tidak memiliki pembatasan untuk aturan produksinya

* + - Contoh : aS → aSBC ; S → aBC ; CbB→BC ; bB→bb ; bC→bc; cC→ccB
		- Mesin yang mengenal **bahasa tanpa-pembatasan** adalah **Mesin Turing (*Turing Machine*).**

Hubungan setiap bahasa digambarkan dalam diagram venn berikut :



**2**

 **Pengelompokan Bahasa Sesuai Hirarky Chomsky**

|  |
| --- |
| JUMLAH PERTEMUAN : 1 PERTEMUANTUJUAN INSTRUKSIONAL KHUSUS :1. Memahami berbagai klasifikasi tingkatan bahasa sesuai dengan Hirarky Chomsky
2. Mengetahui setiap jenis mesin automata sesuai dengan definisinya, sehingga dapat membedakan untuk setiap jenis mesin automata
3. Memahami aturan produksi untuk masing-masing level bahasa yaitu deskripsi batasan α dan β dalam aturan produksi
 |

**Materi :**

**Pengelompokan Bahasa sesuai Hirarky Chomsky**

Tata Bahasa (grammer) 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 pengelompokkan tingkatan bahasa menjadi empat, yang disebut dengan Hirarki chomsky. Pengelompokkan tersebut dapat dilihat pada tabel berikut :

|  |  |  |
| --- | --- | --- |
| **Bahasa** | **Mesin Automata** | **Aturan Produksi** |
| Regular (Tipe 3) | FSA meliputi DFA, NFA | α adalah sebuah simbol variabel.β maksimal memiliki sebuah simbol variabel yang bila ada terletak diposisi paling kanan. |
| Bebas Konteks (Tipe 2) | Push Down Automata | α berupa sebuah simbol variable |
| Context Sensitive (Tipe 1) | Linear Bounded Automata | | α | ≤ | β | |
| Natural Language (Tipe 0) | Mesin Turing | Tidak ada batasan |

Disini semua aturan produksi dalam bentuk α 🡪 β ( ‘dibaca α menghasilkan β’ ), dimana : α menyatakan simbol-simbol pada ruas kiri aturan produksi dan β menyatakan simbol-simbol pada ruas kanan aturan produksi (atau hasil produksi). Simbol-simbol tersebut dapat berupa simbol terminal (non variabel) atau simbol variabel. Simbol terminal adalah simbol-simbol yang tidak dapat diturunkan kembali, sedangkan simbol variabel adalah simbol yang masih bisa diturunkan kembali. Dalam penulisannya, simbol-simbol variabel digunakan huruf kapital (‘A’, ‘B’ dll), sedangkan simbol-simbol terminal digunakan huruf kecil (‘a’, ‘b’, dll).

**1.2 Level-level Bahasa**

Dalam hirarki chomsky, terdapat empat level bahasa yaitu level 0 (Natural Language), level 1 (Contex sensitive), level 2 (Bebas Konteks), dan level 3 (Regular). Dari setiap level tersebut terdapat aturan-aturan produksi yang membatasinya. Contoh-contoh aturan produksi untuk setiap level bahasa :

1. Level 0 (Natural Language)

Bahasa manusia termasuk kedalam tipe ini, dimana tidak ada batasan untuk aturan produksinya.

Contoh :

* 1. Abc 🡪 aa
	2. Bc 🡪 aBaB
	3. CaB 🡪 aaBaaBa
1. Level 1 (Contex sensitive)

Panjang string di ruas kiri (α) ≤ panjang string ruas kanan (β ).

Contoh :

* 1. Ab 🡪 aBa
	2. cD 🡪 aB
	3. dEd 🡪 FabCa
1. Level 2 (Bebas Konteks)

α adalah sebuah simbol variabel, dan batasannya bertambah bahwa ruas kiri harus tetap satu simbol variabel.

Contoh :

* 1. A 🡪 aa
	2. B 🡪 aBD
	3. C 🡪 FGaa
1. Level 3 (Regular)

Batasannya bertambah lagi, dimana ruas kanan maksimal memiliki sebuah simbol variabel yang terletak paling kanan. Artinya bisa memiliki simbol terminal dengan jumlah tidak dibatasi, tetapi bila terdapat simbol variabel maka simbol variabel tersebut hanya berjumlah satu (1) dan terletak paling kanan.

Contoh :

* 1. A 🡪 aa
	2. B 🡪 aaB
	3. C 🡪 aaaaa

**Latihan:**

1. Pilihlah dari sekian aturan produksi dibawah ini kedalam level-level bahasa yang ada:
	1. E 🡪 fgH
	2. E 🡪 HgB
	3. EF 🡪 a
	4. A🡪 aeGab
2. Abc 🡪 dEF
3. D 🡪 Ddd
4. A 🡪 AAA
5. FGH 🡪 abc
6. Buatlah alasan-alasan untuk setiap jawaban dari soal diatas !

**3**

 **Finite State Automata (FSA)**

|  |
| --- |
| JUMLAH PERTEMUAN : 3 PERTEMUANTUJUAN INSTRUKSIONAL KHUSUS :1. Mengetahui pengertian *Finite State Automata* (FSA),mengenali elemen penyusun berupa Tupel FSA dan membuat diagram *Finite State Automata* (FSA)
2. Memahami klasifikasi FSA dan membuat gambaran dalam bentuk diagram untuk masing-masing jenis.
3. Memahami perbedaan jenis FSA dan membuat konfigurasi masing-masing jenis
4. Memahami dalam membuat diagram NFA, NFA-ε dan DFA serta melakukan proses ekivalensi
 |

**Materi :**

**Finite State Automata (FSA)**

Finite state automata (FSA) bukanlah mesin fisik tetapi suatu model matematika dari suatu sistem yang menerima input dan output. FSA merupakan mesin automata dari bahasa regular (tipe 3). Suatu FSA memiliki state yang banyaknya berhingga, dan dapat berpindah dari suatu state ke state lain. Perubahan state dinyatakan oleh fungsi transisi.

Suatu FSA secara formal dinyatakan oleh 5 (lima) tupel M = (Q, Σ, δ, S, F) dimana :

Q = Himpunan state / kedudukan

Σ = Himpunan simbol input / masukan

δ = Fungsi transisi

S = State awal / kedudukan awal

F = Himpunan state akhir

FSA berdasar pada pendefinisian kemampuan berubah state-statenya bisa dikelompokkan kedalam Deterministic Finite Automata dan Non Deterministic Finite Automata.

**3.1 Deterministic Finite Automata (DFA)**

Pada DFA dari suatu state ada tepat satu state berikutnya untuk setiap simbol input (masukan) yang di terima.

a

q0

b

q1

a

b

q2

a

b

Contoh 1

Konfigurasi DFA contoh 1 secara formal adalah sebagai berikut :

Q = {q0, q1, q2}

Σ = {a, b}

S = q0

F = {q2}

Fungsi-fungsi transisinya sebagai berikut :

δ (q0, a) = q0, δ (q0, b) = q1,

δ (q1, a) = q1, δ (q1, b) = q2,

δ (q2, a) = q1, δ (q2, b) = q2.

Jika disajikan dalam tabel transisi :

|  |  |  |
| --- | --- | --- |
| δ | a | b |
| q0 | q0 | q1 |
| q1 | q1 | q2 |
| q2 | q1 | q2 |

\* Dalam DFA, selalu dan pasti terdapat satu state berikutnya untuk setiap pasangan state-input.

**3.2 Non** **Deterministic Finite Automata (NFA)**

Pada NFA dari suatu state bisa terdapat nol (0), satu (1), atau lebih busur keluar (transisis) berlabel simbol yang sama. Jadi setiap pasangna state-input, kita bisa memiliki 0 atau lebih pilihan untuk state berikutnya.

a

q0

a,b

q1

a, b

Contoh 2

Pada NFA contoh 2 diatas terdapat dua busur keluar berlabel input ‘a’. Dari state q0 bila mendapat input ‘a’ bisa berpindah ke state q0 atau q1 yang secara formal dinyatakan : δ (q0, a) = {q0, q1}

Konfigurasi NFA contoh 2 secara formal adalah sebagai berikut :

Q = {q0, q1 }

Σ = {a, b}

S = q0

F = {q1}

Fungsi-fungsi transisinya sebagai berikut :

δ (q0, a) = {q0,q1}, δ (q0, b) = q1,

δ (q1, a) = q1, δ (q1, b) = q1,

Jika disajikan dalam tabel transisi :

|  |  |  |
| --- | --- | --- |
| δ | a | b |
| q0 | {q0,q1} | {q1} |
| q1 | {q1} | {q1} |

\* Perhatikan : Dalam cara penulisan state hasil transisi pada tabel transisi untuk NFA, digunakan kurung kurawal ‘{‘ dan ‘}’karena hasil transisisnya merupakan suatu himpunan state.

\* Pada bagian ini dapat dilihat NFA dengan lebih dari satu pilihan state berikutnya.

Contoh 3

b

q0

a

q1

a

Konfigurasi NFA contoh 3 secara formal adalah sebagai berikut :

Q = {q0, q1 }

Σ = {a, b}

S = q0

F = {q1}

Fungsi-fungsi transisinya sebagai berikut :

δ (q0, a) = q1, δ (q0, b) = q0,

δ (q1, a) = q0, δ (q1, b) = Ø,

Jika disajikan dalam tabel transisi :

|  |  |  |
| --- | --- | --- |
| δ | a | b |
| q0 | {q1} | {q0} |
| q1 | {q0} | Ø |

**3.3 Ekivalensi Non Determnistic Finite Automata dengan Deterministic Finite Automata**

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 4 Diketahui NFA sebagai berikut

0

q0

0,1

q1

1

1

Konfigurasi NFA contoh 4 secara formal adalah sebagai berikut :

Q = {q0, q1 }

Σ = {0, 1}

S = q0

F = {q1}

Tabel transisinya :

|  |  |  |
| --- | --- | --- |
| δ | 0 | 1 |
| q0 | {q0, q1} | {q1} |
| q1 | Ø | {q0, q1} |

* Kita mulai dengan state {q0}

{q0}

Telusuri state berikutnya :

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

Hasilnya :

{q0}

{q0, q1}

0

{q1}

1

* 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 :

{q0}

{q0, q1}

0

{q1}

1

0

 { Ø }

1

0,1

* Selanjutnya telusuri state baru yang terbentuk :
	+ - * δ (Ø, 0) = Ø
			* δ (Ø, 0) = Ø

Hasilnya :

{q0}

0

{q1}

1

0

1

{q0, q1}

0,1

 { Ø }

0,1

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

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

Hasilnya :

{q0}

0

{q1}

1

0

1

{q0, q1}

0,1

 { Ø }

0,1

**Latihan :**

Buatlah DFA yang ekivalen !

1. Diketahui NFA sebagai berikut :

 q0

a,b

 a

 q1

Dengan tabel transisi sebagai berikut :

|  |  |  |
| --- | --- | --- |
| δ | 0 | 1 |
| q0 | {q0, q1} | {q1} |
| q1 | Ø | Ø |

1. Diketahui NFA sebagai berikut :

 q0

Σ = {0, 1},

F = {q0},

S = q0.

**4**

 **Ekspresi Regular dan Hubungannya dengan FSA**

|  |
| --- |
| JUMLAH PERTEMUAN : 1 PERTEMUANTUJUAN INSTRUKSIONAL KHUSUS :1. Memahami bentuk dan rumusan notasi ekspresi regular
2. Memahami hubungan ekspresi regular dengan FSA
3. Membangun ER dari diagram FSA yang diketahui atau sebaliknya.
 |

**Materi :**

**Ekspresi Regular dan Hubungannya dengan FSA**

* Sebuah bahasa dinyatakan regular jika terdapat *finite state automata* yang dapat menerimanya.
* Bahasa-bahasa yang diterima oleh FSA bisa dinyatakan secara sederhana dengan ekspresi regular (regular expression).
* Ekspresi regular memberikan suatu pola (*pattern*) atau *template* untuk untai/*string* dari suatu bahasa.
* Banyak masalah pada perangkat lunak yang bisa disederhanakan dengan melakukan pengubahan notasi ekspresi regular ke dalam implementasi komputer dari FSA yang bersangkutan.
* Contoh : Finite State Automata untuk mengenal bilangan bulat /integer tidak bertanda

0..9

q2

q0

0..9

* Ekspresi Regularnya adalah : misal 0..9 disimbolkan sebagai digit, maka ERnya adalah : (digit)(digit)\*

**10.1. Notasi Ekspresi Regular**

* Notasi yang digunakan untuk ER adalah :
1. \* : berarti bisa tidak muncul, bisa juga muncul berhingga kali (0-n)
2. + : berarti minimal muncul satu kali (1-n)
3. + : berarti union atau bisa diganti dengan notasi U
4. . : berarti konkatenasi, biasanya tanpa ditulis titiknya, misal ab sama dengan a.b

* Contoh ekspresi regular (ER) :
1. ER : ab\*cc

Contoh *string* yang bisa dibangkitkan abcc, acc, abbcc, abbbcc, dst. (b bisa tidak muncul atau muncul sejumlah berhingga kali)

1. ER : 010\*

Contoh *string* yang bisa dibangkitkan 01,010, 0100,01000, dst. (0 bisa tidak muncul atau muncul sejumlah berhingga kali)

1. ER : a+d

Contoh *string* yang bisa dibangkitkan ad,aad, aaad,aaaad dst. (a minimal muncul satu kali)

1. ER : a\* U b\*

Contoh *string* yang bisa dibangkitkan a, b, aa, bb, dst.

1. ER : 01\*+0

Contoh *string* yang bisa dibangkitkan 0, 01,011, dst.

**10.2. Hubungan ER dengan FSA**

* Untuk setiap ER ada satu NFA dengan transisi ε (NFA ε-move) yang ekivalen.
* Sementara untuk setiap DFA ada satu ER dari bahasa yang diterima oleh DFA.
* Hubungannya dapat digambarkan sebagai berikut :

NFA

NFA ε-move

DFA

ER

**5**

* **Aturan Produksi untuk Suatu Finite State Automata**
* **FSA dengan Output**

|  |
| --- |
| JUMLAH PERTEMUAN : 2 PERTEMUANTUJUAN INSTRUKSIONAL KHUSUS :1. Mengetahui penggambaran Aturan produksi dan bisa melakukan penggolongan bahasa regular
2. Membuat aturan produksi dari diagram FSA yang diketahui atau sebaliknya.
3. Memahami konsep dasar FSA dengan Output dan penerapannya dalam beberapa mesin
4. Memahami konfigurasi dan cara kerja Mesin Moore dan Mealy

  |

**Materi :**

**Aturan Produksi untuk suatu Finite State Automata**

Tata Bahasa (grammer) didefinisikan dengan empat (4) tupel G = ({V, T, P, S}) dimana :

V = Himpunan simbol variabel / non terminal

T = Himpunan simbol terminal

P = Kumpulan aturan produksi

S = Simbol awal

**5.1 Aturan Produksi Bahasa Reguler**

Kita masih ingat dengan aturan produksi dari bahasa regular (tipe 3) yaitu :

α 🡪 β

α adalah sebuah simbol variabel.

β maksimal memiliki sebuah simbol variabel yang bila ada terletak diposisi paling kanan

Batasannya bertambah lagi, dimana ruas kanan maksimal memiliki sebuah simbol variabel yang terletak paling kanan. Artinya bisa memiliki simbol terminal dengan jumlah tidak dibatasi, tetapi bila terdapat simbol variabel maka simbol variabel tersebut hanya berjumlah satu (1) dan terletak paling kanan.

**5.2 Mengkonstruksi Aturan Produksi dari Suatu Finite State Automata**

Dalam mengkonstruksi aturan produksi tata bahasa regular dari suatu FSA , perlu kita ingat yang menjadi perhatian adalah state-state yang bisa menuju ke state akhir.

Contoh 1 : Mesin FSA

 q0

 q1

 a

 q2

 q3

 ε

 ε

 q4

b

 a

 b

 b

Pada mesin FSA contoh 1, memiliki simbol input ‘a’ dan ‘b’.

* + - * Misal kita identikan state awal qo dengan simbol awal S.

δ (q0, a) = q1

Dapat ditulis :

S 🡪 aE

Dimana E kita identikan dengan q1.

* + - * Dari q1 terdapat transisi :

δ (q1, ε) = q2 dan δ (q1, ε) = q3

Dapat ditulis :

E 🡪 A E 🡪 B

Dimana A kita identikan dengan q2 dan B kita identikan dengan q3.

* + - * Selanjutnya dapat kita lihat, dari state q2 dengan input ‘a’ kembali ke state q2 dan dari state q3 dengan input ‘b’ kembali ke state q3.

δ (q2, a) = q2 dan δ (q3, b) = q3

Dapat ditulis :

A 🡪 aA B 🡪 bB

* + - * Selanjutnya, dari state q2 dengan input ‘b’ menuju state q4 dan dari state q3 dengan input ‘b’ menuju ke state q4. Sementara q4 adalah himpunan state akhir dan dari state q4 tidak ada lagi busur keluar, maka :

δ (q2, b) = q4 dan δ (q3, b) = q4

Dapat ditulis :

A 🡪 b B 🡪 b

* + - * Kumpulan aturan produksi yang kita peroleh bisa ditulis sebagai berikut :

S 🡪 aE

E 🡪 A | B

A 🡪 aA | b

B 🡪 bB | b

Secara formal dapat ditulis :

V = {S, E, A, B}

T = {a, b}

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

S = S

* 1. **Finite State Automata untuk Suatu Tata Bahasa Reguler**

Jika sebelumnya dari suatu diagram transisi FSA dapat dibuat aturan-aturan produksi tata bahasa regularnya, maka sebaliknya bisa juga mengkonstruksi diagram transisi FSA untuk suatu tata bahasa regular yang diketahui aturan-aturan produksinya.

Contoh 2 : Tata bahasa regular

S 🡪 aB | bA | ε

A 🡪 abaS

B 🡪 babS

Kita dapat langsung gambar atau rancang diagram transisi FSA nya!

S identik dengan q0 ; A identik dengan q4; dan B identik dengan q1.

Lengkapnya adalah sebagai berikut :

 a

 q0

 b

 q2

 a

 q3

 b

 q4

 b

 q5

 a

 q6

 b

 a

 q1

* 1. **FSA dengan Output**
		1. **Mesin Moore**

Suatu keterbatasan dari Finite State Automata yang sudah dipelajari adalah keputusannya terbatas pada diterima atau ditolak saja. Automata tersebut disebut sebagai accepter, dalam hal ini disebut Fiite State Accepter.

Kita dapat mengkonstruksi suatu Finite State Automata yang memiliki keputusan beberapa keluaran atau output, dalam hal ini disebut Finite State Transducer. Pada mesin Moore, output akan berasosiasi dengan state.

Mesin Moore memiliki 6 (Enam) tupel, M = (Q, Σ, δ, S, Δ, λ). Dimana :

Q = Himpunan State

Σ = Himpunan Simbol Input

δ = Fungsi Transisi

S = State Awal

Δ = Himpunan Output

λ = Fungsi Output untuk setiap State

Keterangan : Komponen state akhir dari Deterministic Finite Automata dihilangkan, karena disini keputusan dimunculkan sebagai output.

Contoh 1 :

Penerapan Mesin Moore

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 :

q0

0

1

q2

0

1

q1

0

1

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

* + 1. **Mesin Mealy**

Bila output pada mesin Moore berasosiasi dengan state, maka output pada Mesin Mealy akan beasosiasi dengan transisi. Mesin Mealy didefinisikan dalam 6 (enam) tupel, yaitu :

Q = Himpunan State

Σ = Himpunan Simbol Input

δ = Fungsi Transisi

S = State Awal

Δ = Himpunan Output

λ = Fungsi Output untuk setiap Transisi

Mesin ini akan mengeluarkan output apakah menerima (Y) atau menolak (T), suatu masukan. Dimana mesin akan mengeluarkan output ‘Y’ bila menerima untai yagn memiliki akhiran 2 simbol berurutan yang sama.

Misal string yang diterima oleh Mesin Mealy adalah : 00, 11, 011, 100, 11011, 0100, 010011, 010100, 101011, dll.

Konfigurasi Mesin Mealy tersebut :

Q = {q0, q1, q2}

Σ = {0, 1} (input dalam biner)

S = q0

Δ = {Y, T}

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 Mesin Mealy

q2

0/T

1/Y

q1

0/Y

1/T

q0

0/T

1/Y

**Latihan :**

1. Diketahui FSA sebagai berikut :

 q0

 b

 a

 q1

 a

 b

 q2

 a

 b

Kontruksi tata bahasa regular nya !

1. Buatlah FSA dari aturan-aturan produksi tata bahasa regular berikut ini :

S 🡪 0A

A 🡪 10A | ε

1. Buatlah FSA dari aturan-aturan produksi tata bahasa regular berikut ini :

A 🡪 00B | 11D | 0

B 🡪 00D | 11C | 1

C 🡪 00B | 11D | 0

D 🡪 00D | 1

**6**

 **Penyederhanaan Tata Bahasa Bebas Konteks**

|  |
| --- |
| JUMLAH PERTEMUAN : 1 PERTEMUANTUJUAN INSTRUKSIONAL KHUSUS :1. Memahami tata bahasa bebas konteks dan tujuan penyederhanaannya
2. Memahami berbagai cara penyederhanaan tata bahasa bebas konteks berdasarkan logika penghilangan

  |

**Materi :**

**Penyederhanaan Tata Bahasa Bebas Konteks**

**6.1 Tujuan Penyederhanaan**

Penyederhanaan tata bahasa bebas konteks bertujuan untuk melakukan pembatasan sehingga tidak menghasilkan pohon penurunan yang memiliki kerumitan yang tidak perlu atau aturan produksi yang tidak berarti.

Suatu tatabahasa bebas kontek dapat melakukan penyederhanaan dengan melakukan :

1. Penghilangan Produksi Useless
2. Penghilangan Produksi Unit
3. Penghilangan Produksi ε

**6.2 Penghilangan Produksi Useless**

Disini produksi useless didefinisikan sebagai :

* 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 simbolo 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

Bisa dilihat :

* Simbol variable A tidak memiliki penurunan yang menuju terminal, sehingga bisa dihilangkan.
* Konsekuensi no 1, aturan produksi S 🡪 Abd tidak memiliki penurunan

Maka tata bahasa hasil penyederhanaan adalah :

S 🡪 aSa | Bde

B 🡪 BBB | a

Contoh 2 :

Diketahui tata bahasa bebas konteks sebagai berikut

S 🡪 Aa | B

A 🡪 ab | D

B 🡪 b | E

C 🡪 bb

E 🡪 aEa

Bisa dilihat :

* Aturan produksi A 🡪 D, simbol variable D tidak memiliki penurunan, sehingga bisa dihilangkan.
* Aturan produksi C 🡪 bb, jika dilakukan penurunan dari simbol awal S, dengan jalan manapun tidak akan pernah dicapai. Sehingga bisa dihilangkan.
* Simbol variable E tidak memiliki aturan produksi yang menuju terminal ( E 🡪 aEa ) satu-satunya aturan produksi dari E, sehingga bisa dihilangkan.
* Konsekuensi no 3, aturan produksi B 🡪 E, simbol variable E tidak memiliki penurunan, sehingga bisa dihilangkan.

Dapat dilihat, produksi yang useless adalah :

A 🡪 D ; C 🡪 bb ; E 🡪 aEa ; B 🡪 E

Maka tata bahasa hasil penyederhanaan menjadi :

S 🡪 Aa | B

A 🡪 ab

B 🡪 b

**6.3 Penghilangan Produksi Unit**

Produksi unit adalah produksi yang ruas kiri dan kanan aturan produksinya hanya berupa satu simbol variable. Dengan adanya bentuk prioduksi unit ini membuat tata bahasa memiliki kerumitan yang tidak perlu atau menambah panjang penurunan. Penyederhanaan ini dilakukan dengan melakukan penggantian aturan produksi unit.

Contoh 3 :

Diketahui tata bahasa bebas konteks sebagai berikut :

S 🡪 Sb

S 🡪 C

C 🡪 D

C 🡪 ef

D 🡪 dd

Kita lakukan penggantian berurutan mulai dari aturan produksi yang paling dekat menuju terminal-terminal (=> ‘ dibaca menjadi’)

* C 🡪 D => C 🡪 dd
* S 🡪 C => S 🡪 dd | ef

Sehingga aturan produksi setelah penyederhanaan :

S 🡪 Sb | dd | ef

C 🡪 dd | ef

D 🡪 dd

Contoh 4 :

Diketahui tata bahasa bebas konteks sebagai berikut :

S 🡪 A

S 🡪 Aa

A 🡪 B

B 🡪 C

B 🡪 b

C 🡪 D

C 🡪 ab

D 🡪 b

Penggantian yang dilakukan :

* C 🡪 D => b
* B 🡪 C => B 🡪 b | ab, karena B 🡪 b sudah ada maka kita

cukup tulis B 🡪 ab

* A 🡪 B => A 🡪 ab | b
* S 🡪 A => S 🡪 ab | b

Sehingga aturan produksi hasil penyederhanaan :

S 🡪 A => S 🡪 ab | b

S 🡪 Aa

A 🡪 B => A 🡪 ab | b

B 🡪 ab

B 🡪 b

C 🡪 b

C 🡪 ab

D 🡪 b

**6.4 Penghilangan Produksi Empty**

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 bias disebut nullable. Prinsip penggantian bias dilihat kasus berikut :

Kasus 1.

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

Contoh 5 : Diketahui tata bahasa bebas konteks sebagai berikut :

S 🡪 Ab | Cd

A 🡪 d

C 🡪 ε

Variabel yang nullabel adalah C. karena penurunan C 🡪 ε merupakan penurunan satu-satunya dari C, maka kita ganti S 🡪 Cd manjadi S 🡪 d. kemudian produksi C 🡪 ε dihapus.

Tata bahasa bebas konteks setelah penyederhanaan :

S 🡪 Ab | d

A 🡪 d

Contoh 6 :

S 🡪 dA | Bd

A 🡪 bc

A 🡪 ε

B 🡪 c

Variabel yang nullable adalah A, A 🡪 ε bukan satu-satunya produksi dari A. Maka kita ganti S 🡪 dA manjadi S 🡪 dA | d kemudian A 🡪 ε dihapus.

Tata bahasa bebas konteks hasil penyederhanaan :

S 🡪 dA | d | Bd

A 🡪 bc

B 🡪 c

**Latihan :**

1. Hilangkan semua produksi useless dari tata bahasa bebas konteks berikut :

S 🡪 AB | CA

B 🡪 BC | AB

A 🡪 a

C 🡪 aB | b

1. Hilangkan semua produksi unit dari tata bahasa bebas konteks berikut :

S 🡪 A | aB

A 🡪 cD | B

B 🡪 bB | D

D 🡪 aa

1. Hilangkan semua produksi empty dari tata bahasa bebas konteks berikut :

S 🡪 AaB | aaB

A 🡪 ε

B 🡪 bbA | ε

4. 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

**7**

 **Bentuk Normal Chomsky**

|  |
| --- |
| JUMLAH PERTEMUAN : 2 PERTEMUANTUJUAN INSTRUKSIONAL KHUSUS :1. Mengetahui pengertian bentuk normal chomsky
2. Memahami Langkah-langkah pembentukan bentuk normal chomsky
 |

**Materi :**

**Bentuk Normal Chomsky**

**7.1 Pengertian Bentuk Normal Chomsky**

Suatu tata bahasa bebas konteks dapat dikonstruksi menjadi bentuk normal chomsky (Chomsky Normal Form) jika tidak lagi mengandung produksi-produksi yang useless, unit, dan empty (ε). Dengan kata lain, untuk mengkonstruksi suatu CNF dari suatu tata bahas bebas konteks adalah dengan cara menghilangkan produksi-produksi useless, unit, dan ε.

Aturan produksi dari CNF adalah, ruas kanan adalah sebuah simbol terminal atau dua simbol variable.

Jadi dalam 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.

**7.2 Pembentukan Bentuk Normal Chomsky**

Langkah – langkah untuk mengkonstruksi CNF dari suatu tata bahasa bebas konteks :

1. Biarkan aturan-aturan produksi yang sudah dalam bentuk CNF.
2. Lakukan penggantian-penggantian terhadap aturan-aturan produksi yang ruas kanannya mengandung lebih dari satu simbol terminal.
3. Lakukan penggantian-penggantian terhadap aturan-aturan produksi yang ruas kanannya mengandung lebih dari dua simbol variabel.
4. Dalam melakukan penggantian ini dimungkinkan terciptanya suatu simbol-simbol variable baru dan aturan-aturan produksi baru.
5. Hasil akhir adalah gabungan dari aturan-aturan produksi yang sudah dalam bentuk CNF, aturan-aturan produksi yang telah dilakukan penggantian, dan aturan-aturan produksi yang baru.

Contoh 1 :

Diketahui tata bahasa bebas konteks (sudah tidak mengandung produksi useless, unit dan ε) sebagai berikut :

S 🡪 aB | SS | c

B 🡪 BBB | cd | a

Langkah membuat CNF dari tata bahasa bebas konteks contoh 1 :

* Biarkan aturan-aturan produksi yang sudah dalam bentuk CNF

S 🡪 SS ; S 🡪 c ; B 🡪 a

* Lakukan penggantian aturan-aturan produksi yang belum dalam bentuk CNF

yang belum dalam bentuk CNF :

S 🡪 aB => S 🡪 Z1B

B 🡪 BBB => B 🡪 Z2 B

B 🡪 cd => B 🡪 Z3 d => B 🡪 Z3 Z4

* Simbol variable baru dan aturan produksi baru yang terbentuk :

Z1 🡪 a Z2 🡪 BB Z3 🡪 c Z4 🡪 d

* Hasil akhir :

S 🡪 SS | c | Z1B

B 🡪 Z2B | Z3 Z4 | a

Z1 🡪 a

Z2 🡪 BB

Z3 🡪 c

Z4 🡪 d

Contoh 2 :

Diketahui tata bahasa bebas konteks sebagai berikut :

S 🡪 aa | Ba | cd

B 🡪 AA | AAA | Dc

A 🡪 abcd

D 🡪 De | ee | f

Langkah membuat CNF dari tata bahasa bebas konteks contoh 2 :

* Biarkan aturan-aturan produksi yang sudah dalam bentuk CNF

B 🡪 AA D 🡪 f

* Lakukan penggantian aturan-aturan produksi yang belum dalam bentuk CNF

yang belum dalam bentuk CNF :

S 🡪 aa => S 🡪 Z1 a => S 🡪 Z1 Z1

S 🡪 Ba => S 🡪 B Z1

S 🡪 cd => S 🡪 Z2 d => S 🡪 Z2 Z3

B 🡪 AAA => B 🡪 Z4 A

B 🡪 Dc => B 🡪 D Z2

A 🡪 abcd => A 🡪 Z1bcd => A 🡪 Z1 Z5 cd

=> A 🡪 Z1 Z5 Z2 d => A 🡪 Z1 Z5 Z2 Z3

=> A 🡪 Z6 Z2 Z3 => A 🡪 Z6 Z7

* Simbol variable baru dan aturan produksi baru yang terbentuk :

Z1 🡪 a Z3 🡪 d Z5 🡪 b Z7 🡪 Z2 Z3

Z2 🡪 c Z4 🡪 AA Z6 🡪 Z1 Z5

* Hasil akhir :

S 🡪 Z1 Z1 | B Z1 | S 🡪 Z2 Z3

B 🡪 Z4 A | D Z2

A 🡪 Z6 Z7

Z1 🡪 a Z3 🡪 d Z5 🡪 b Z7 🡪 Z2 Z3

Z2 🡪 c Z4 🡪 AA Z6 🡪 Z1 Z5

**Latihan :**

1. Buatlah bentuk normal chomsky dati tata bahasa bebas konteks berikut ini :

S 🡪 aSaA | A

A 🡪 abA | b

1. Buatlah bentuk normal chomsky dati tata bahasa bebas konteks berikut ini :

S 🡪 abAB

A 🡪 bAB | ε

B 🡪 BAa | A | ε

1. Buatlah bentuk normal chomsky dati tata bahasa bebas konteks berikut ini :

S 🡪 abEab

E 🡪 AB

A 🡪 bA | BBBA

B 🡪 BAa | AbCd

**8**

**Penghilangan Bentuk Left Linear Grammer**

|  |
| --- |
| JUMLAH PERTEMUAN : 1 PERTEMUANTUJUAN INSTRUKSIONAL KHUSUS :1. Memahami Langkah-langkah penghilangan bentuk left linear grammer
 |

**Materi :**

**Penghilangan Bentuk Left Linear Grammer**

Dalam banyak penerapan tata bahasa, left linear grammer tak diinginkan. Untuk menghindari penurunan yang bisa mengakibatkan loop perlu menghilangkan sifat left linear grammer dari aturan produksi. Penghilangan left linear grammer disini memungkinkan suatu tata bahasa bebas konteks nantinya diubah kedalam bentuk normal greibach.

**8.1 Tahap Penghilangan Left Linear Grammer**

Langkah-langkahpenghilangan left linear grammer adalah sebagai berikut :

* Pisahkan aturan-aturan produksi yang left linear grammer dan yang tidak.

Misal :

Aturan produksi yang left linear grammer : A 🡪 Aα1 | Aα2 | Aα3 |… Aαn

Aturan produksi yang tidak left linear grammer : A 🡪 β1 | β2 | β3 |… βm

* Dari situ kita bisa tentukan α1, α2, α3,.. αn, dan β1, β2, β3,… βm dari setiap aturan produksi yang memiliki simbol ruas kiri yanG sama.
* Lakukan penggantian aturan produksi yang left linear grammer, menjadi sebagai berikut :
1. A 🡪 β1Z | β2Z |…. βmZ
2. Z 🡪 α1 | α2 | α3 | …. αn
3. Z 🡪 α1Z | α2Z | α3Z | …. αnZ

Penggantian diatas dilakukan untuk setiap aturan produksi dengan simbol ruas kiri yang sama. Bisa muncul simbol variabel baru Z1, Z2,.. Zn dan seterusnya sesuai banyaknya variabel yang menghasilkan produksi yang left linear grammer.

* Hasil akhir berupa aturan produksi pengganti ditambah dengan aturan produksi semula yang tidak left linear grammer.

Contoh 1:

Diketahui tata bahasa bebas konteks sebagai berikut :

S 🡪 Sab | aSc | dd | ff | Sbd

* Pertama-tama lakukan pemisahan aturan produksi

Aturan produksi yang left linear grammer :

S 🡪 Sab | Sbd

Dapat ditentukan untuk simbol ruas kiri S : α1 = ab ; α2 = bd

Aturan produksi yang tidak left linear grammer :

S 🡪 aSc | dd | ff

Dapat ditentukan untuk simbol ruas kiri S : β1 = aSc ; β2 = dd ; β3 = ff ;

* Lakukan penggantian aturan produksi yang masih left linear grammer :

Untuk simbol ruas kiri S : S 🡪 Sab | Sbd

* 1. S 🡪 aScZ1 | ddZ1 | ffZ1
	2. Z1 🡪 ab | bd
	3. Z1 🡪 abZ1 | bdZ1
* Hasil akhir :

S 🡪 aSc | dd | ff

S 🡪 aScZ1 | ddZ1 | ffZ1

Z1 🡪 ab | bd

Z1 🡪 abZ1 | bdZ1

Contoh 2 :

Tata bahasa bebas konteks :

S 🡪 Sab | Sb | cA

A 🡪 Aa | a | bd

* Pertama-tama lakukan pemisahan aturan produksi

Aturan produksi yang left linear grammer :

S 🡪 Sab | Sb

A 🡪 Aa

Dapat ditentukan untuk simbol ruas kiri S : α1 = ab ; α2 = b

Dapat ditentukan untuk simbol ruas kiri A : α1 = a

Aturan produksi yang tidak left linear grammer :

S 🡪 cA

A 🡪 a | bd

Dapat ditentukan untuk simbol ruas kiri S : β1 = cA

Dapat ditentukan untuk simbol ruas kiri A : β1 = a; β2 = bd

* Lakukan penggantian aturan produksi yang masih left linear grammer :

Untuk simbol ruas kiri S : S 🡪 Sab | Sb

1. S 🡪 cAZ1
2. Z1 🡪 ab | b
3. Z1 🡪 abZ1 | bZ1

Untuk simbol ruas kiri A : A 🡪 Aa

1. A 🡪 aZ2 | bdZ2
2. Z2 🡪 a
3. Z2 🡪 aZ2
* Hasil akhir adalah :

S 🡪 cA

A 🡪 a | bd

S 🡪 cAZ1

Z1 🡪 ab | b

Z1 🡪 abZ1 | bZ1

A 🡪 aZ2 | bdZ2

Z2 🡪 a

Z2 🡪 aZ2

**Latihan :**

* 1. Lakukan penghilangan bentuk left linear grammer dari tata bahasa bebas konteks berikut :

A 🡪 Abd | d

B 🡪 Bac | a | BAd

* 1. Lakukan penghilangan bentuk left linear grammer dari tata bahasa bebas konteks berikut :

S 🡪 Abd | d | Sad

A 🡪 Sa | aA | AA

B 🡪 ac | a | Bcad

**DAFTAR PUSTAKA**

* John E. Hopcorf & Jeffrey D Ullman (1979).*Introduction to automata Theory, languages & Computation.*Addison Wessley Inc.
* Firrar utdirartatmo (2001).*Teori Bahasa dan Otomata.*J&J Learning.Yogyakarta.
* Alfred v. a. & ullman J.D., Compilers Principles Technique and Tools, Addison Wesley, 1988.
* Stephen K.O. & Steven N., Turbo Pascal 7: The Complete Reference, McGraw-Hill, 1993.