BUKU AJAR

TEORI BAHASA FORMAL DAN *AUTOMATA*

PENULIS

NURUL KHAIRINA

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MEDAN AREA
MEDAN
2020

TEORI BAHASA FORMAL DAN *AUTOMATA*

Penulis : Nurul Khairina
ISBN : 978-602-1577-34-9

Editor : Muhammad Khoiruddin Harahap
Penata Letak : Fachrurozi
Desain Sampul : Muhammad Khoiruddin Harahap

Jalan Kolam No 1. Medan Estate, Medan
Telephone : 061-7366878
e-mail: lp2m@uma.ac.id

Diterbitkan oleh :
University Medan Area Press
Jalan Kolam No 1. Medan Estate, Medan
Website**:** https://uma.ac.id/

Dicetak dan di distribusikan oleh:
University Medan Area Press

SINOPSIS

Teori Bahasa Formal dan *Automata* merupakan bidang ilmu yang penting di dalam perkuliahan Ilmu Komputer/ Teknik Informatika. Dalam perkuliahan, *automata* diperlukan untuk memahami mata kuliah Teknik Kompilasi.

Topik buku ajar Teori Bahasa Formal dan *Automata* ini berisikan penjelasan teori dasar, contoh sederhana, contoh soal, penyelesaian contoh soal, rangkuman dan tugas mandiri. Buku ajar ini ditulis menggunakan bahasa dan kalimat sederhana agar memberikan kemudahan bagi pembaca.

Buku ajar Teori Bahasa Formal & *Automata* ini terdiri dari 14 Bab, yaitu Konsep Dasar Teori Bahasa Formal & Mesin *Automata*, Hirarki Chomsky, FSA, Ekuivalensi NFA menjadi DFA, NFA dengan ε – *move*, Ekspresi Reguler, Aturan Produksi FSA, FSA dengan *output*, Pohon Penurunan, Penyerderhanaan Tata Bahasa Bebas Konteks, Bentuk Normal Chomsky, Penghilangan Rekursif Kiri, *Push Down Automata*, dan Mesin Turing.

Bab 1 sampai dengan Bab 5 difokuskan pada konsep dasar *automata* yang ditandai dengan teori yang terkait pada mesin FSA, NFA, dan DFA. Bab 6 memberikan penjelasan tentang Ekspresi Reguler yang kemudian berhubungan dengan Aturan Produksi FSA serta FSA dengan *output* pada Bab 7 dan Bab 8.

Pada Bab 9, Pohon Penurunan merupakan topik yang akan bermanfaat untuk membahami Penyerderhanaan Tata Bahasa Bebas Konteks, Bentuk Normal Chomsky, Penghilangan Rekursif Kiri yang akan dibahas kemudian pada Bab 10 sampai Bab 12.

Bab 13 dan Bab 14 merupakan pembahasan paling tinggi dalam Teori Bahasa Formal dan *Automata*, implementasi dari Teori Bahasa Formal dan *Automata* juga dapat lebih jelas terlihat pada dua bab terakhir. Bab 13 menjelaskan tentang konsep dasar mesin *Push Down Automata* (PDA) yang mampu mengelola bahasa Bebas Konteks, kemudian Bab 14 menjelaskan tentang konsep dasar mesin Turing yang mampu mengelola *natural language/ unrestricted.*

KATA PENGANTAR

Alhamdulillah, segala puji bagi Allah SWT atas kasih sayang dan rahmat Nya, penulis dapat menyelesaikan buku ajar Teori Bahasa Formal dan *Automata* ini dengan baik.

Buku ajar Teori Bahasa Formal dan *Automata* ini diperuntukkan bagi mahasiswa, dosen, dan khalayak umum yang ingin mempelajari *automata* secara mendalam.

Teori Bahasa Formal dan *Automata* ini merupakan ilmu dasar dalam Ilmu Komputer/ Teknik Informatika, yang nantinya akan bermanfaat dan membantu para pembaca dalam memahami mata kuliah Teknik Komputasi.

Untuk memudahkan pembaca dalam memahami buku ajar ini, penulis memaparkan teori dengan contoh-contoh yang sederhana. Disamping itu, juga terdapat contoh soal dan penyelesaiannya, serta tugas mandiri yang dapat dikerjakan oleh mahasiswa di setiap akhir bab pembahasan.

Penulis berterimakasih kepada seluruh pihak yang telah terlibat, yaitu orangtua yang selalu memberikan motivasi dan dukungan kepada penulis, Bapak Muhammad Khoiruddin Harahap, ST, M. Kom yang telah bersedia membimbing penulis sampai penulisan buku ini selesai, staff LP2M dan PGHC Universitas Medan Area yang telah membantu dalam penerbitan buku ini.

Penulis menyadari sepenuhnya dengan segala keterbatasan bahwa masih banyak kekurangan pada buku ajar ini. Oleh karena itu, penulis terbuka terhadap kritik dan saran yang positif dan membangun untuk perbaikan buku ajar ini ke depannya.

Medan, 9 November 2019

Penulis

DAFTAR ISI

BAB I
DASAR TEORI BAHASA FORMAL DAN MESIN *AUTOMATA*

Capaian Pembelajaran Mata Kuliah

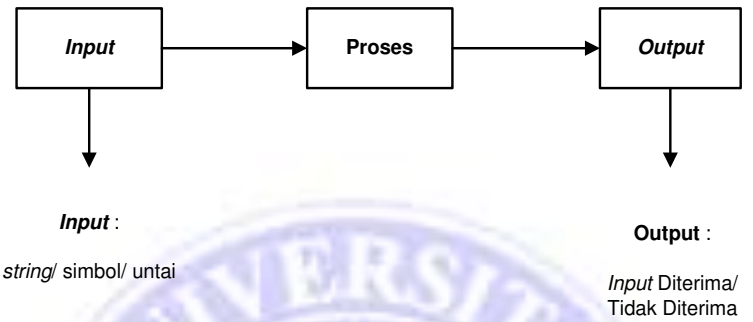
- ❖ Mahasiswa mampu memahami pendahuluan mata kuliah Teori Bahasa Formal & *Automata*

1.1 Konsep Teori Bahasa Formal dan *Automata*

Terdapat dua istilah dasar yang paling penting di dalam mempelajari teori bahasa dan mesin *Automata*. Teori bahasa formal dan mesin *Automata* berasal dari dua kata yang berbeda, yaitu: "bahasa" dan "mesin *automata*". Bahasa merupakan himpunan atau deretan *string* (untai/simbol) yang memiliki makna. Mesin *automata* merupakan istilah untuk model dari sebuah mesin yang dapat memproses *input* dan menghasilkan *output*. Mesin *automata* juga merupakan mesin abstrak yang dibangun dari model matematika yang mampu mengenali dan menerima sebuah kata maupun kalimat dalam bahasa tertentu.

Pada mesin *automata*, *string* yang masuk akan diproses dan mesin *automata* akan mengeluarkan pernyataan yang menyatakan bahasa tersebut **"diterima"** atau **"tidak diterima"**. Secara umum, istilah *input* pada mesin *automata* dapat berupa abjad (a, b, c, dst)

dan bilangan biner (0, 1). Secara rinci dapat dilihat pada gambar di bawah ini :

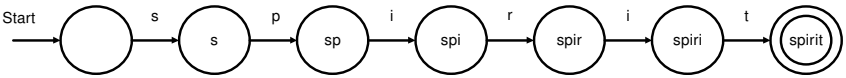


Gambar 1.1. Konsep Dasar Mesin *Automata*

Pada Gambar 1.1, *input* dapat berupa *string/*simbol seperti huruf alfabetik antara a-z, dan bilangan biner 0 atau 1. *Input* ini akan di proses pada mesin *automata*. Proses yang akan terjadi pada mesin *automata* merupakan proses membaca dan mengenali setiap *input*, dan output dari mesin *automata* adalah keputusan apakah deretan *input* tersebut dapat dikenali dengan baik atau tidak. Kondisi output inilah yang dinyatakan dengan *input* dapat diterima atau tidak dapat diterima.

Lexical analyzer merupakan bagian dari *compiler, lexical analyzer* adalah bentuk dari penerapan teori bahasa formal dan *automata*. *Lexical analyzer* memiliki tugas untuk memecah teks-teks *input* menjadi *logical unit,* seperti : *identifier*, *keyword* dan *punctuation*. Peran mesin *automata* dalam *lexical analyzer* dapat

dilihat pada proses mesin *automata* yang mengenali *input* kata *spirit* berikut ini :



Gambar 1.2 Pemodelan Mesin *automata* dalam Mengenali *Input*

Catatan :

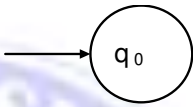
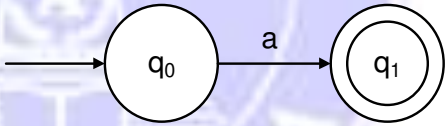
Lexical Analyzer : membaca dan mengenali kata-kata, kemudian menghasilkan rangkaian simbol atau *string* yang lebih sederhana

Compiler : menterjemahkan program komputer yang ditulis dalam bahasa pemrograman tertentu menjadi program komputer dalam bahasa pemrograman lain

1.2 Komponen Dasar Mesin *Automata*

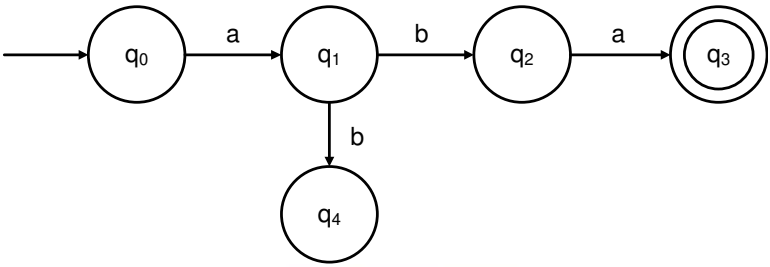
Mesin *automata* memiliki beberapa komponen dasar yang perlu diketahui, yaitu:

Tabel 1.1 Komponen Dasar Mesin *Automata*

No	Komponen	Simbol
1	*String*/ *input*/simbol/ untai	Alfabetik (a, b, c, d sampai z) serta bilangan biner 0 dan 1
2	*State* Awal	
3	*State* Akhir/ *Final State* **misalkan state akhir q_4*	
4	*State* dengan *Input* **state awal q_0 dengan input a menuju state final q_1*	

1.3 Cara Mesin *Automata* Membaca *Input*

Sebuah *input* yang masuk dan diproses pada mesin *automata*, dapat memiliki 2 kondisi, yaitu: dapat dibaca dengan baik **("diterima")** dan tidak dapat dibaca dengan baik **("tidak diterima")**. Untuk mempermudah memahami hal ini, dapat dilihat pada gambar 1.3 di bawah ini :



Gambar 1.3 Mesin *Automata* Sederhana

Gambar di atas, terlihat adanya *input* (berupa huruf **a**) dari *state* awal **q_0** menuju *state* **q_1**. Sebuah *input* dapat diterima, apabila dapat dibaca dari *state* awal menuju *state* akhir.

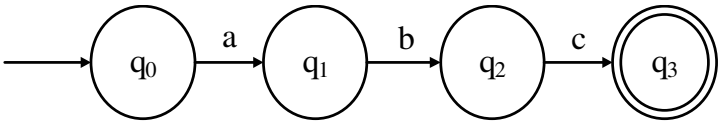
Jika kita perhatikan gambar di atas, terdapat *input* **'aba'** yang dapat dibaca dari *state* awal q_0 menuju *state* akhir q_3. Sehingga dapat dikatakan *input* **'aba'** merupakan *input* yang dapat **diterima** pada mesin *automata*.

Sedangkan *input* yang dibaca dari *state* q_0 menuju *state* q_4 (berupa huruf a dan b) adalah *input* yang tidak dapat dibaca, dengan kata lain **'ab'** merupakan bahasa yang **tidak diterima** oleh mesin *automata*.

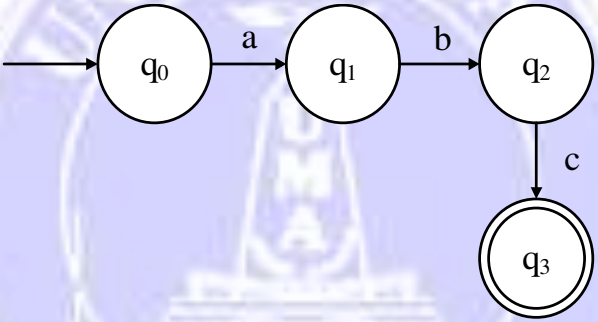
1.4 Cara Membuat Gambar Mesin *Automata*

Dalam membuat gambar mesin *automata*, harus terlebih dahulu memahami *state* awal dan *state* akhir dengan baik. Misalkan kita ingin membuat mesin *automata* dengan *input* 'abc'. Model mesin *automata* yang digambarkan dapat bervariasi, namun harus tetap

mengikuti konsep mesin *automata*. Berikut ini dua variasi model mesin *automata* untuk satu *input* 'abc' :



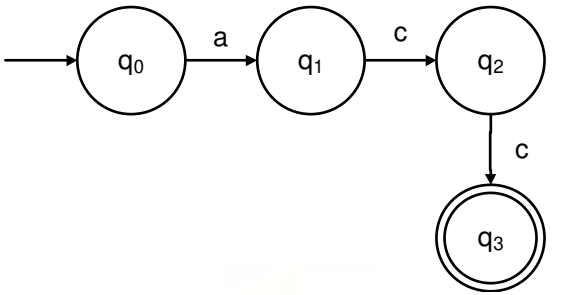
Gambar 1.4 Mesin *automata* dengan *input* 'abc' (Variasi 1)



Gambar 1.5 Mesin *automata* dengan *input* 'abc' (Variasi 2)

Contoh Soal 1.1

Terdapat sebuah mesin *automata*, tentukan bahasa yang dapat diterima dari mesin *automata* berikut :

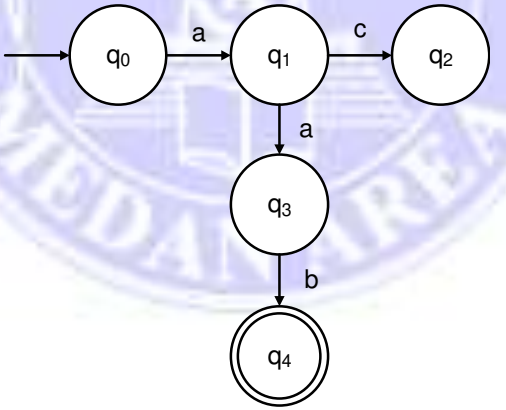


Penyelesaian Contoh Soal 1.1 :

Bahasa yang dapat diterima oleh mesin *automata*: acc

Contoh Soal 1.2

Terdapat sebuah mesin *automata*, tentukan bahasa yang dapat diterima dari mesin *automata* berikut :



Penyelesaian Contoh Soal 1.2

Bahasa yang dapat diterima oleh mesin *automata*: aab

Contoh Soal 1.3

Gambarkanlah mesin *automata* yang dapat menerima *input* berikut ini :

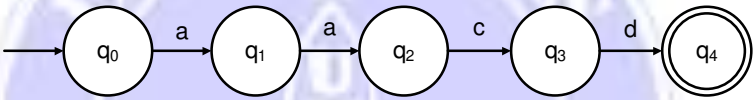
a. aacd

b. abcd

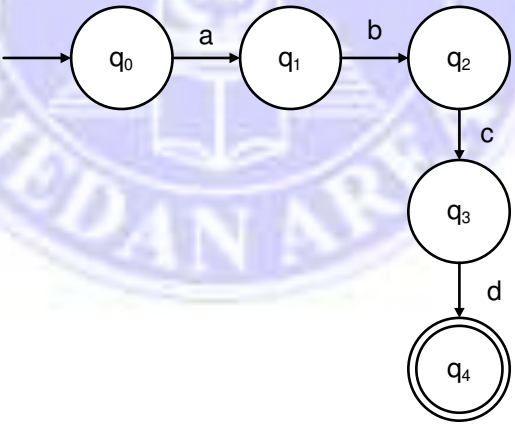
Penyelesaian Contoh Soal 1.3

Mesin *automata* yang dapat menerima *input* :

a. aacd :



b. abcd :

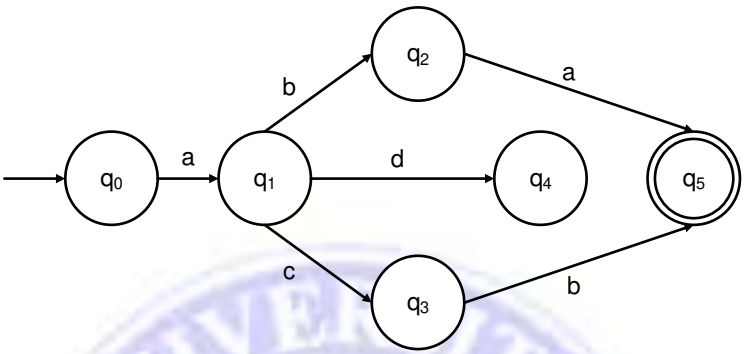


Rangkuman

1. Pada mesin *automata*, terdapat beberapa istilah penting seperti *input/* bahasa, *state*, *state* awal, dan *state* akhir.
2. Dalam membuat sebuah mesin *automata*, harus terlebih dahulu memahami bagaimana sebuah mesin *automata* dapat membaca *input*.
3. Pada mesin *automata*, terdapat *input/* bahasa yang dapat di terima oleh mesin dan tidak dapat diterima oleh mesin.
4. Sebuah *input/* bahasa yang dapat diterima oleh mesin *automata* adalah *input* yang dapat dibaca dari *state* awal sampai *state* akhir
5. Sebuah *input/* bahasa yang tidak dapat diterima oleh mesin *automata* adalah *input* yang tidak dapat dibaca dari *state* awal sampai *state* akhir, atau dengan kata lain, proses pembacaan *input* akan berhenti pada *state* tertentu sebelum sampai ke *state* akhir.

Tugas

1. Buatlah mesin *automata* yang dapat menerima *input* berupa bilangan biner ‘1010’
2. Buatlah mesin *automata* yang dapat menerima *input* berupa bilangan biner dari bilangan desimal 9
3. Tentukanlah bahasa yang dapat diterima oleh mesin *automata* berikut :


q0
q1
q2
q3
q4
q5
a
b
a
d
c
b

❖ Daftar Istilah Simbol Latin

Latin	Uppercase	Lowercase	Latin	Uppercase	Lowercase
Alpha	A	α	Nu	N	ν
Beta	B	β	Xi	Ξ	ξ
Gamma	Γ	γ	Omicron	O	o
Delta	Δ	δ	Pi	Π	π
Epsilon	E	ε	Rho	P	ρ
Zeta	Z	ζ	Sigma	Σ	σ
Eta	H	η	Tau	T	τ
Theta	Θ	θ	Upsilon	Υ	υ
Lota	I	ι	Phi	Φ	ϕ
Kappa	K	κ	Chi	X	χ
Lambda	Λ	λ	Psi	Ψ	ψ
Mu	M	μ	Omega	Ω	ω

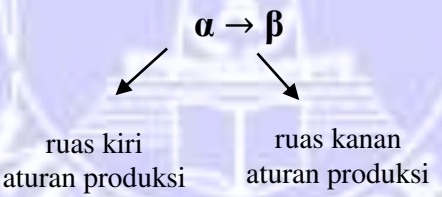
BAB II
HIRARKI CHOMSKY

Capaian Pembelajaran Mata Kuliah

- Mahasiswa mampu memahami dasar Hirarki Chomsky

2.1. Hirarki Chomsky

Hirarki Chomsky merupakan pembagian tingkatan bahasa yang dibatasi oleh aturan produksi masing-masing. Noam Chomsky membagi tingkatan bahasa menjadi 4, yaitu bahasa reguler, bahasa bebas konteks, bahasa *context sensitive*, dan bahasa *unrestricted*. Sebuah aturan produksi dapat ditulis dalam bentuk :



(dibaca : α menurunkan β atau α menghasilkan β)

Pada ruas kiri dan ruas kanan aturan produksi, terdapat istilah simbol variabel dan simbol terminal. Berikut penjelasan lebih rinci terhadap variabel dan terminal :

a. Variabel/non terminal merupakan simbol yang masih bisa diturunkan. Terdiri dari huruf abjad antara A - Z yang ditulis dengan huruf besar.

   Contoh : S, A, B, C, D, dan sebagainya

b. Terminal merupakan simbol yang tidak dapat diturunkan lagi. Terdiri dari huruf abjad antara a – z yang ditulis dengan huruf kecil.

   Contoh : a, b, c, d, e, dan sebagainya.

Dalam aturan produksi, juga terdapat tanda ” | ” yang artinya ” atau ”. Apabila terdapat dua aturan produksi yang memiliki variabel ruas kanan yang sama, maka aturan produksi tersebut dapat dituliskan dengan lebih singkat menggunakan tanda ” | ” tanpa mengurangi maknanya.

Contoh

$$A \rightarrow cd$$

$$A \rightarrow def$$

Dua aturan produksi di atas dapat dituliskan dengan lebih singkat menggunakan tanda | , sehingga :

$$A \rightarrow cd \mid def$$

1.1.1 Bahasa Reguler

Bahasa reguler *(Type 3)* merupakan tata bahasa yang berkaitan dengan mesin *Finite State Automata* (FSA). Tata bahasa yang dapat diterima pada tata Bahasa Reguler, merupakan tata bahasa yang memenuhi aturan produksi sebagai berikut :

- Aturan produksi ruas kiri (α) harus memiliki sebuah simbol variabel
- Aturan produksi ruas kanan (β) maksimal memiliki sebuah simbol variabel (apabila ada, harus terletak di posisi paling kanan).

Contoh

a. Tata bahasa reguler yang dapat diterima :

   S → a

   S → A

   S → cD

   A → efG

b. Tata bahasa reguler yang tidak dapat diterima :

   s → a

   As → c

   C → Bd

   C → aBD

   ε → bdC

Catatan :

Pada tata bahasa reguler, ruas kiri haruslah berisi variabel.

$\varepsilon \rightarrow$ bdC (tidak dapat diterima)

Aturan produksi di atas dianggap tidak dapat diterima oleh mesin *automata*, karena ε tidak dapat menurunkan aturan produksi apapun dan dianggap tidak memenuhi persyaratan aturan produksi tata bahasa reguler.

1.1.2 Bahasa Bebas Konteks

Bahasa bebas konteks *(context free/ Type 2)* merupakan tata bahasa yang berkaitan dengan mesin *Push Down Automata* (PDA). Tata bahasa yang dapat diterima pada tata Bahasa Bebas Konteks, merupakan tata bahasa yang memenuhi aturan produksi sebagai berikut :

- Aturan produksi ruas kiri (α) harus memiliki sebuah simbol variabel.

Contoh

a. Tata bahasa bebas konteks yang dapat diterima :

   A → c

   B → cD

   S → Ad

   C → AA

   S → ε

b. Tata bahasa bebas konteks yang tidak dapat diterima :

   $a \rightarrow Cd$

   $Aa \rightarrow c$

   $CD \rightarrow f$

   $\varepsilon \rightarrow aB$

1.1.3 Bahasa *Context Sensitive*

Bahasa *context sensitive/ Type 1)* merupakan tata bahasa yang berkaitan dengan mesin *Linear Bounded Automata.* Tata bahasa yang dapat diterima pada tata Bahasa *Context Sensitive*, merupakan tata bahasa yang memenuhi aturan produksi sebagai berikut :

- Aturan produksi ruas kiri (α) minimal harus memiliki sebuah variabel
- Jumlah *string* pada ruas kiri harus lebih kecil sama dengan jumlah string pada ruas kanan. Secara matematika dapat dituliskan : $| \alpha | \leq | \beta |$

Contoh

a. Tata bahasa *context sensitive* yang dapat diterima :

   $A \rightarrow c$

   $Ab \rightarrow ef$

   $AC \rightarrow FG$

sA → cDf

Ef → AbGh

B → ε

b. Tata bahasa *context sensitive* yang tidak dapat diterima :

s → a

Sab → Cd

AACD → ef

CD → ε

scG → ε

Catatan :

Pada tata bahasa *context sensitive* (syarat :| α | ≤ | β |) :

B → ε (dapat diterima)

Aturan produksi di atas dianggap dapat diterima oleh mesin *automata*, karena *context sensitive* hanya menghitung jumlah *string* ruas kiri dan kanan, dimana :

| B | = 1, | ε | = 1

Sehingga **B → ε** dianggap memenuhi aturan produksi *context sensitive*.

1.1.4 Bahasa *Unrestricted*

Bahasa *unrestricted (Phase Structure/ Natural Language/ Type 0)* berkaitan dengan Mesin Turing. Bahasa ini dikatakan bahasa

natural karena sudah termasuk bahasa yang dapat dimengerti oleh manusia. Tata bahasa yang dapat diterima pada tata Bahasa *Unrestricted*, merupakan tata bahasa yang memenuhi aturan produksi sebagai berikut :

- Aturan produksi pada ruas kiri (α) harus memiliki minimal sebuah simbol variabel

Contoh

a. Tata bahasa *unrestricted* yang dapat diterima :

   Abc → e

   bCDef → DC

   abEF → Cd

   gHj → ε

b. Tata bahasa *unrestricted* yang tidak dapat diterima :

   efg → h

   ab → ε

Contoh Soal 2.1

Berikanlah pernyataan ”diterima” atau ”tidak diterima” sesuai dengan tata bahasanya untuk aturan produksi tata **Bahasa Reguler** di bawah ini :

a. A → Df
b. S → bcG
c. ε → mnC

Penyelesaian Contoh Soal 2.1

Tata Bahasa Reguler :

a. A → Df : Diterima
b. S → bcG : Tidak Diterima
c. ε → mnC : Tidak Diterima

Contoh Soal 2.2

Berikanlah pernyataan ”diterima” atau ”tidak diterima” sesuai dengan tata bahasanya untuk aturan produksi tata **Bahasa Bebas Konteks** di bawah ini :

a. X → Ca
b. AB → m
c. ε → sN

Penyelesaian Contoh Soal 2.2

Tata Bahasa Bebas Konteks

a. X → Ca : Diterima
b. AB → m : Tidak Diterima
c. ε → sN : Tidak Diterima

Contoh Soal 2.3

Berikanlah pernyataan ”diterima” atau ”tidak diterima” sesuai dengan tata bahasanya untuk aturan produksi tata **Bahasa *Context Sensitive*** di bawah ini :

a. $Y \rightarrow \varepsilon$
b. $ABCD \rightarrow xy$
c. $X \rightarrow y$

Penyelesaian Contoh Soal 2.3

Tata Bahasa *Context Sensitive* :

a. $Y \rightarrow \varepsilon$: Diterima
b. $ABCD \rightarrow xy$: Tidak Diterima
c. $X \rightarrow y$: Diterima

Contoh Soal 2.4

Berikanlah pernyataan ”diterima” atau ”tidak diterima” sesuai dengan tata bahasanya untuk aturan produksi tata **Bahasa *Unrestricted*** di bawah ini :

a. $eFg \rightarrow \varepsilon$
b. $rs \rightarrow \varepsilon$
c. $xyz \rightarrow ab$

Penyelesaian Contoh Soal 2.4

Tata Bahasa *Unrestricted* :

a. $eFg \rightarrow \varepsilon$: Diterima
b. $rs \rightarrow \varepsilon$: Tidak Diterima
c. $xyz \rightarrow ab$: Tidak Diterima

Rangkuman :

1. Terdapat empat tingkatan bahasa yang dibagi oleh Noam Chomsky, yaitu : bahasa reguler, bahasa bebas konteks, bahasa *context sensitive*, dan bahasa *unrestricted*.
2. Sebuah aturan produksi memiliki ruas kanan dan ruas kiri yang dapat dimodelkan secara matematika dengan $\alpha \rightarrow \beta$.
3. Setiap tingkatan bahasa memiliki batasan aturan produksi tersendiri beserta jenis mesin *automata* yang berbeda.

Tugas :

Berilah centang √ (Diterima) atau X (Tidak Diterima) untuk setiap aturan produksi dibawah ini dan berikan alasannya kenapa aturan produksi tersebut tidak dapat diterima.

No	Aturan Produksi	Bahasa Reguler	Bahasa Bebas Konteks	Bahasa Context Sensitive	Bahasa Unrestricted
1	FFF → GGG				
2	cd → ε				
3	B → cDD				
4	A → ε				
5	B → Fg				
6	eF → ABCD				
7	ε → aG				
8	DE → ε				
9	efgHi → abcd				

10	W → xyZ				
11	Rs → vW				
12	T → uVw				
13	x → Y				
14	mnO → PQ				
15	S → abcD				
16	BCDEFG → i				
17	A → cdefghi				
18	S → A				
19	ε → e				
20	ε → ABCDEF				

BAB III

MESIN *FINITE STATE AUTOMATA* (FSA)

Capaian Pembelajaran Mata Kuliah

- Mahasiswa mampu memahami dasar mesin FSA *(Finite State Automata)*
- Mahasiswa dapat memahami dasar mesin DFA *(Deterministic Finite Automata*)
- Mahasiswa dapat memahami dasar mesin NFA *(Non Deterministic Finite Automata*)

3.1 *Finite State Automata* (FSA)

Sesuai dengan *hirarki chomsky* yang telah dijelaskan pada bab sebelumnya, mesin *Finite State Automata* (FSA) berkaitan dengan tata bahasa reguler. Konsep dasar mesin ini sama dengan mesin *automata*, dimana mesin mampu menerima *input* yang akan di proses, dan akan menghasilkan *output* yang sesuai.

Mesin FSA terdiri dari 5 tupel atau dapat ditulis dengan :

$M = (Q, \Sigma, \delta, S, F)$

Keterangan :

Q = Jumlah *state*

Σ = *Input*

δ = Fungsi transisi

S = *State* awal

F = *State* akhir

Mesin FSA dibagi ke dalam dua jenis, yaitu *Deterministic Finite Automata* (DFA) dan *Non Deterministic Finite Automata* (NFA), secara lebih lengkap dapat dilihat pada sub bab di bawah ini :

3.1.1 *Deterministic Finite Automata*

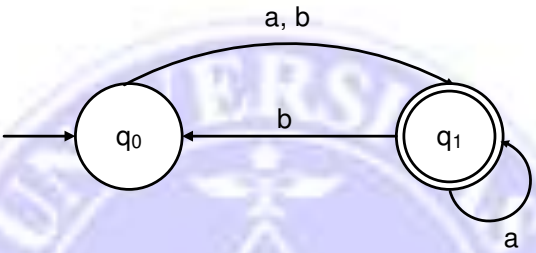
Ciri khas mesin DFA ditandai dengan adanya kondisi antara satu *state* dengan *state* lainnya, dimana setiap *state* tersebut selalu tepat ada satu *state* berikutnya. Dalam *finite state automata*, ada beberapa istilah yang akan sering digunakan, yaitu :

- Konfigurasi mesin : deskripsi mesin secara formal
- Fungsi transisi : hubungan antara *state* dan *input* yang disajikan dalam fungsi
- Tabel transisi : hubungan antara *state* dan *input* yang disajikan dalam tabel

Ada dua cara dalam memahami mesin DFA, yaitu membuat fungsi transisi dan tabel transisi dari gambar mesin, dan membuat gambar mesin dari tabel transisi.

3.1.1.1 Membuat Fungsi Transisi dan Tabel Transisi dari Gambar Mesin

Terdapat mesin DFA sederhana yang terdiri dari dua *state* (q_0 dan q_1 dengan *input* a dan b. State q_0 dianggap sebagai *state* awal, *state* q_1 dianggap sebagai *state* final.



Gambar 3.1 Mesin DFA Sederhana

Dari gambar 3.1 di atas, dapat dituliskan konfigurasi mesin, fungsi transisi dan tabel transisi sebagai berikut :

a. Konfigurasi mesin :

   $Q = \{q_0, q_1\}$

   $\Sigma = \{a, b\}$

   $S = q_0$

   $F = \{q_1\}$

b. Fungsi transisi :

   $\delta\ (q_0\ ,\ a) = q_1$

   $\delta\ (q_0\ ,\ b) = q_1$

δ (q_1 , a) = q_1

δ (q_1 , b) = q_0

c. Tabel transisi :

δ	a	b
q_0	q_1	q_1
q_1	q_1	q_0

3.1.1.2 Membuat Gambar Mesin dari Tabel Transisi :

Terdapat tabel transisi yang terdiri dari q_0 dan q_1 dengan *input* dan b. Dimana q_0 merupakan *state* awal, dan q_1 adalah *state* final, yang dapat dilihat pada tabel berikut ini :

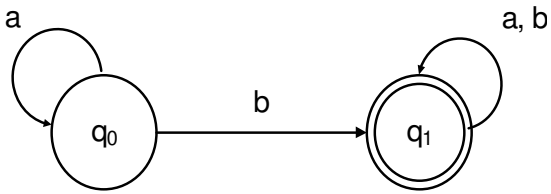
Tabel 3.1 Tabel Transisi

δ	a	b
q_0	q_0	q_1
q_1	q_1	q_1

Dari tabel transisi tersebut, ada dua langkah yang harus dilakukan untuk membuat gambar mesin DFA, yaitu :

a. Gambarkan dua buah state yaitu q_0 sebagai *state* awal, dan q_1 sebagai *state* final.
b. Buatlah arah panah dari masing-masing *state* menuju *state* tujuan, sesuaikan dengan perintah dalam tabel transisi.

Berikut ini mesin DFA yang sesuai dengan tabel 3.1 :



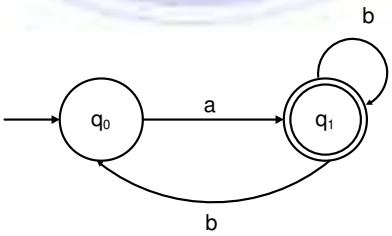
Gambar 3.2 Mesin DFA Sesuai dengan Tabel Transisi

3.1.2 *Non Deterministic Finite Automata*

Secara umum, mesin NFA hampir sama dengan mesin DFA, hanya saja pada mesin NFA, sebuah *state* boleh tidak memiliki/ memiliki 1 arah busur/ lebih dari 1 arah busur pada sebuah *input* yang sama untuk menuju pada *state* berikutnya. Pada mesin NFA, fungsi transisi dan tabel transisi ditulis menggunakan kurung kurawal (‘ { } ‘), dan boleh terdapat himpunan kosong (∅) apabila tidak ada hubungan sebuah *state* pada *state* berikutnya. Berikut ini contoh mesin NFA beserta fungsi transisi dan tabel transisinya:

Contoh :

a. Mesin NFA



Gambar 3.3 Mesin NFA

b. Fungsi Transisi :

δ (q_0 , a) = {q_1}

δ (q_0 , b) = ∅

δ (q_1 , a) = ∅

δ (q_1 , b) = {q_0, q_1}

c. Tabel Transisi :

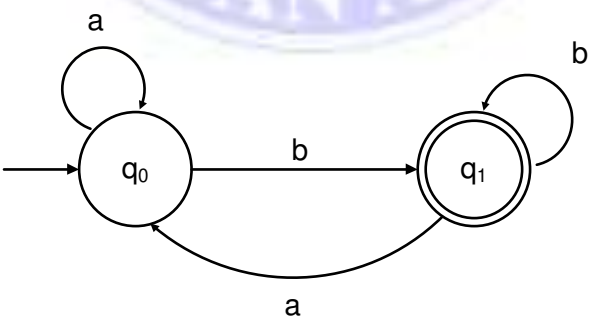
Tabel 3.2 Tabel Transisi

δ	a	b
q_0	{q_1}	∅
q_1	∅	{q_0, q_1}

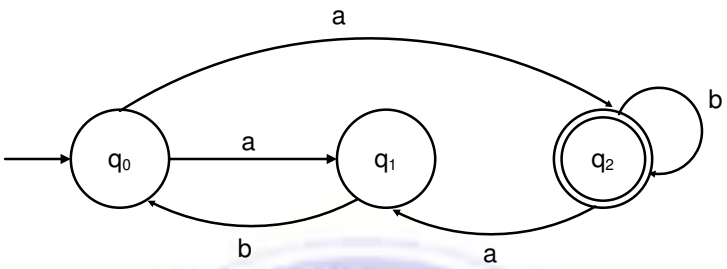
Contoh Soal 3.1

Buatlah fungsi transisi dan tabel transisi dari mesin *automata* di bawah ini :

a. Mesin DFA :



b. Mesin NFA



Penyelesaian Contoh Soal 3.1

a. Mesin DFA :

- Fungsi Transisi :

  δ (q_0 , a) = q_1

  δ (q_0 , b) = q_1

  δ (q_1 , a) = q_0

  δ (q_1 , b) = q_1

- Tabel Transisi :

δ	a	b
q_0	q_0	q_1
q_1	q_0	q_1

b. Mesin NFA :

- Fungsi Transisi :

  δ (q_0 , a) = {q_1, q_2}

$\delta\ (q_0\ ,\ b) = \emptyset$

$\delta\ (q_1\ ,\ a) = \emptyset$

$\delta\ (q_1\ ,\ b) = \{q_0\}$

$\delta\ (q_2\ ,\ a) = \{q_1\}$

$\delta\ (q_2\ ,\ b) = \{q_2\}$

- Tabel Transisi :

δ	a	b
q_0	$\{q_1, q_2\}$	$\emptyset$
q_1	$\emptyset$	$\{q_0\}$
q_2	$\{q_1\}$	$\{q_2\}$

Contoh Soal 3.2

Buatlah mesin automata dari tabel transisi berikut ini :

a. Tabel Transisi Mesin DFA :

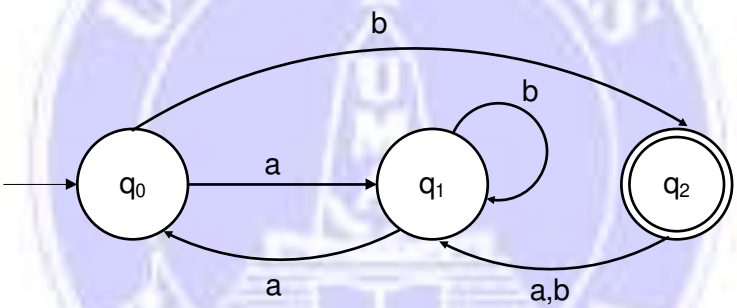
δ	a	b
q_0	q_1	q_2
q_1	q_0	q_1
q_2	q_1	q_1

b. Tabel Transisi Mesin NFA :

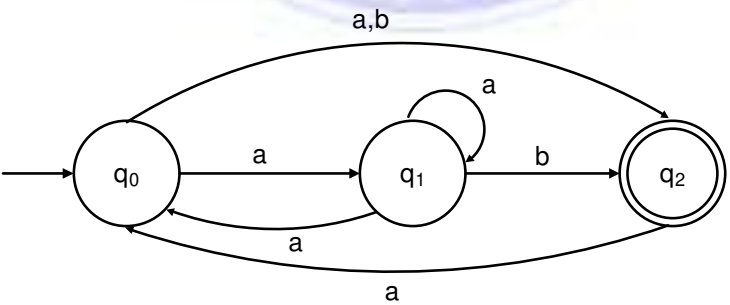
δ	a	b
q_0	$\{q_1,q_2\}$	$\{q_2\}$
q_1	$\{q_0,q_1\}$	$\{q_2\}$
q_2	$\{q_0\}$	$\emptyset$

Penyelesaian Contoh Soal 3.2

a. Mesin DFA :



b. Mesin NFA :



Rangkuman

1. Mesin DFA ditandai dengan adanya hubungan antara satu *state* dengan *state* lainnya, dimana setiap *state* tersebut selalu tepat ada satu *state* berikutnya.
2. Mesin NFA ditandai dengan sebuah *state* boleh tidak memiliki arah busur/ 1 arah busur/ lebih dari 1 arah busur pada sebuah *input* yang sama untuk menuju pada *state* berikutnya.
3. Perbedaan mesin NFA dengan mesin DFA juga dapat dilihat dari adanya penggunaan kurung kurawal (‘{ }’) pada fungsi transisi dan tabel transisi mesin NFA, dan adanya simbol himpunan kosong ($\emptyset$) yang digunakan apabila tidak ada *input* yang menuju ke sebuah *state*.

Tugas

1. Gambarkan mesin *Deterministic Finite Automata* sesuai dengan konfigurasi berikut :

   $Q = \{q_0, q_1, q_2\}$

   $\Sigma = \{a, b\}$

   $S = q_0$

   $F = \{q_2\}$

   Fungi transisi :

δ	A	b
q_0	q_0	q_1
q_1	q_1	q_2

q_2	q_1	q_0

2. Gambarkan mesin *Non Deterministic Finite Automata* sesuai dengan konfigurasi berikut :

   $Q = \{q_0, q_1, q_2\}$

   $\Sigma = \{0, 1\}$

   $S = q_0$

   $F = \{q_1\}$

   Fungi transisi :

δ	0	1
q_0	$\{q_0, q_2\}$	q_2
q_1	$\{q_0, q_2\}$	q_1
q_2	$\emptyset$	$\emptyset$

3. Buatlah *Non-Deterministic Finite Automata* yang dapat menerima *string* :
   a. 001
   b. 10010
   c. 111000

BAB IV
EKUIVALENSI NFA *(NON DETERMINISTIC FINITE AUTOMATA)* MENJADI DFA *(DETERMINISTIC FINITE AUTOMATA)*

Capaian Pembelajaran Mata Kuliah

- Mahasiswa dapat memahami ekuivalensi NFA *(Non Deterministic Finite Automata*) menjadi DFA *(Deterministic Finite Automata)*

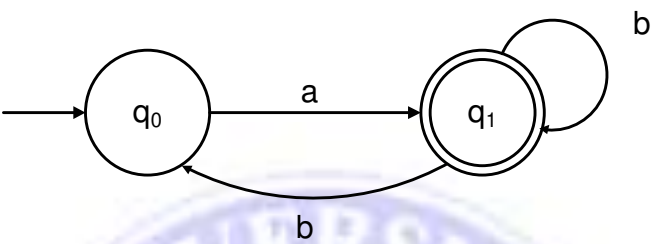
4.1 Konsep Dasar Ekuivalensi NFA Menjadi DFA

Ekuivalensi NFA menjadi DFA merupakan langkah untuk merubah mesin NFA menjadi DFA. Secara umum, langkah-langkah untuk membuat ekuivalensi antara kedua mesin tersebut adalah:

1. Membuat tabel transisi dari mesin NFA
2. Membentuk *state-state* baru sesuai dengan transisinya, yang dimulai dari *state* awal, dimana setiap *state* dituliskan dengan menggunakan kurung kurawal (‘{ }’).
3. Lakukan penyesuaian semua *input* pada setiap *state* tujuan.
4. Ulangi tahapan ke-2 dan ke-3 sampai semua *state* sudah sesuai dengan tabel transisi.

4.2 Ekuivalensi NFA ke DFA

Berikut mesin NFA yang akan diubah menjadi mesin DFA :



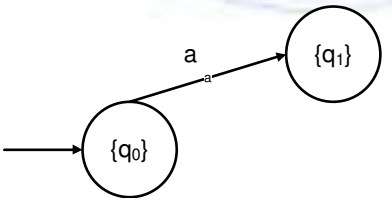
Gambar 4.1 Mesin NFA

Tahapan merubah NFA menjadi DFA:

1. Tabel transisi :

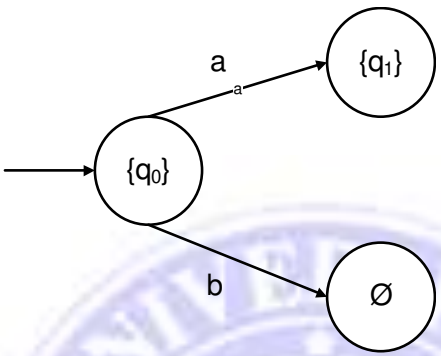
δ	A	b
q_0	$\{q_1\}$	$\emptyset$
q_1	$\emptyset$	$\{q_0, q_1\}$

2. Gambarkan mesin *automata* baru sesuai dengan tabel transisi pada tahapan sebelumnya.
   a. Tahap 1 : *(State q_0 terhadap input a)*



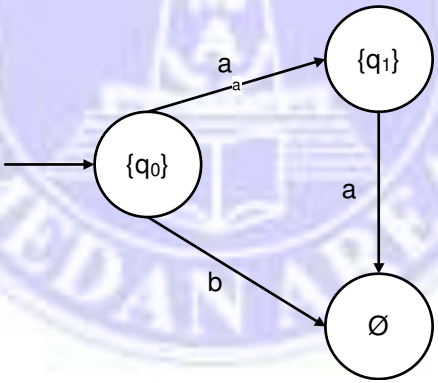
Gambar 4.2 Tahap 1 Ekuivalensi NFA-DFA

b. Tahap 2 : *(State q_0 terhadap input a)*



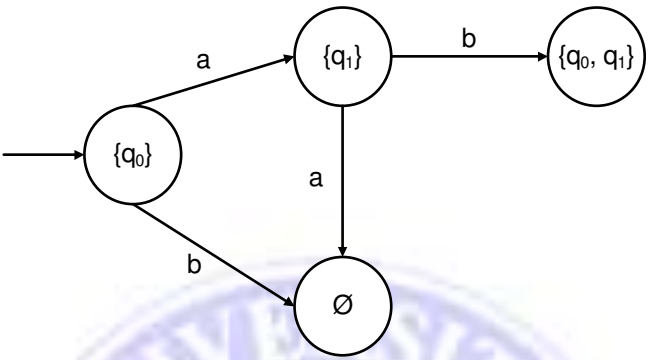
Gambar 4.3 Tahap 2 Ekuivalensi NFA-DFA

c. Tahap 3 : *(State q_1 terhadap input a)*



Gambar 4.4 Tahap 3 Ekuivalensi NFA-DFA

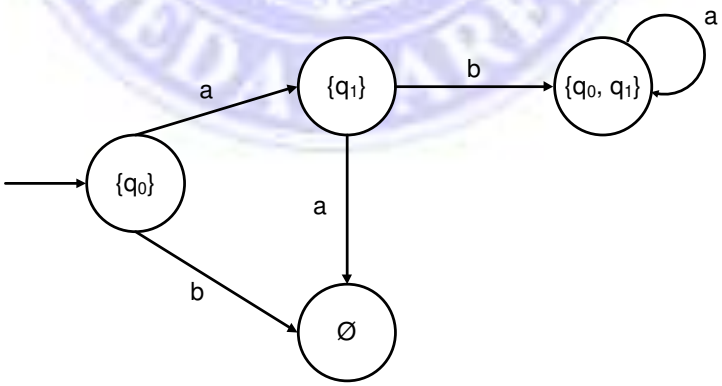
d. Tahap 4 : *(State q_1 terhadap input b)*



Gambar 4.5 Tahap 4 Ekuivalensi NFA-DFA

e. Tahap 5 : Perhatikan tabel transisi, dimana :

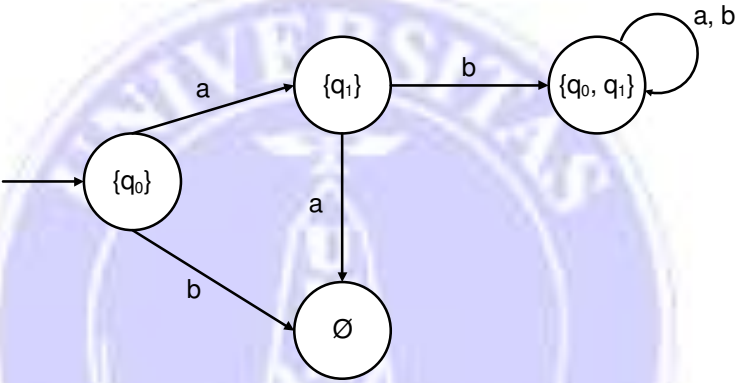
- *State* $\{q_0, q_1\}$ diberi *input* a = *State* $\{q_0, q_1\}$.
  Karena δ $(q_0, a) = \{q_1\}$ dan δ $(q_1, a) = \emptyset$, maka :
  δ $([q_0, q_1], a) = \{q_0, q_1\}$.
- Sehingga kita tambahkan *input* a pada state $\{q_0, q_1\}$.



Gambar 4.6 Tahap 5 Ekuivalensi NFA-DFA

f. Tahap 6 : Perhatikan tabel transisi, dimana :

- *State* $\{q_0, q_1\}$ diberi *input* b = *State* $\{q_0, q_1\}$.
  Karena $\delta\ (q_0, b) = \emptyset$ dan $\delta\ (q_1, b) = \{q_0, q_1\}$,
  maka : $\delta\ ([q_0, q_1], b) = \{q_0, q_1\}$.
- Sehingga kita tambahkan *input* b pada state $\{q_0, q_1\}$.



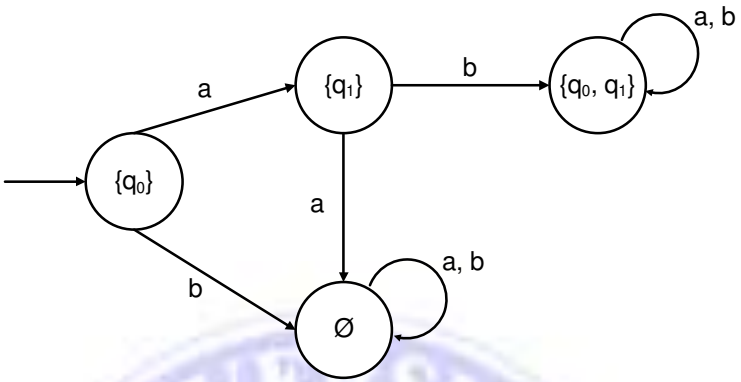
Gambar 4.7 Tahap 6 Ekuivalensi NFA-DFA

g. Tahap 7 : Perhatikan tabel transisi, dimana :

$\delta\ (q_0, b) = \emptyset$

$\delta\ (q_1, a) = \emptyset$

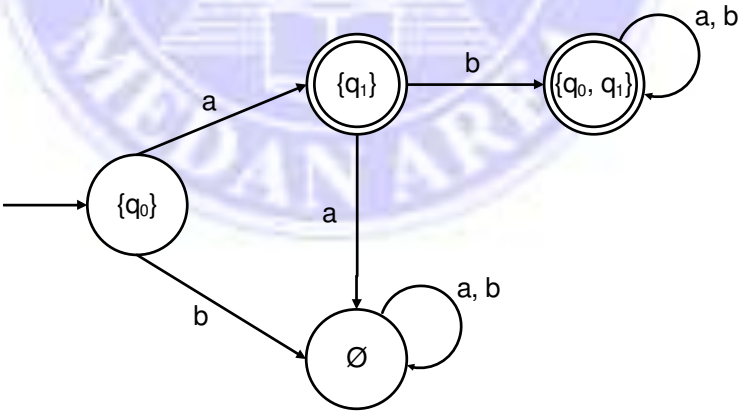
- Dari dua fungsi transisi di atas, *state* $\emptyset$ menerima input a dan b. Sehingga kita bisa menambahkan input a dan b pada *state* $\emptyset$



Gambar 4.8 Tahap 7 Ekuivalensi NFA-DFA

h. Tahap 8 : *Final State*

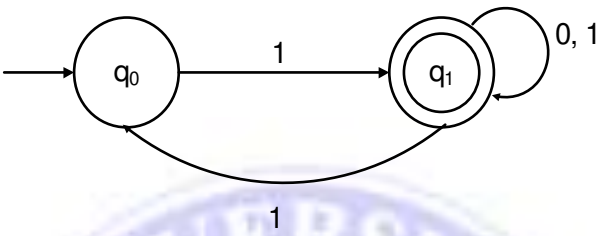
Apabila kita lihat kembali mesin NFA pada Gambar 4.1, *final state* terletak pada *state* $\{q_1\}$. Maka *final state* mesin DFA yang baru adalah semua *state* yang mengandung $\{q_1\}$. Maka *final state* adalah $\{q_1\}$ dan $\{q_0, q_1\}$.



Gambar 4.9 DFA yang Ekuivalen dengan NFA

Contoh Soal 4.1

Buatlah mesin DFA dari mesin NFA berikut ini :



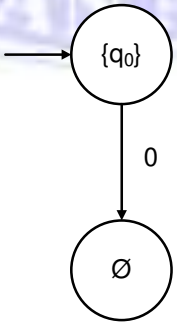
Penyelesaian Contoh Soal 4.1

Langkah-langkah membuat mesin DFA dari mesin NFA :

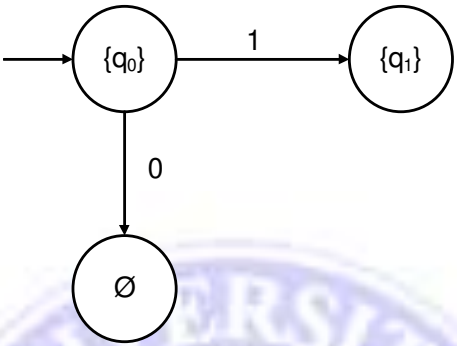
1. Tabel Transisi :

δ	0	1
q_0	Ø	$\{q_1\}$
q_1	$\{q_1\}$	$\{q_0, q_1\}$

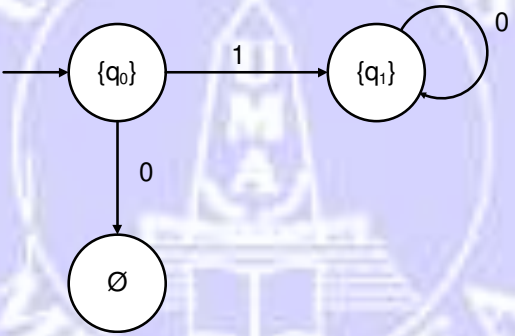
2. Mesin DFA :
   a. Tahap 1 : *(State q_0 terhadap input 0)*



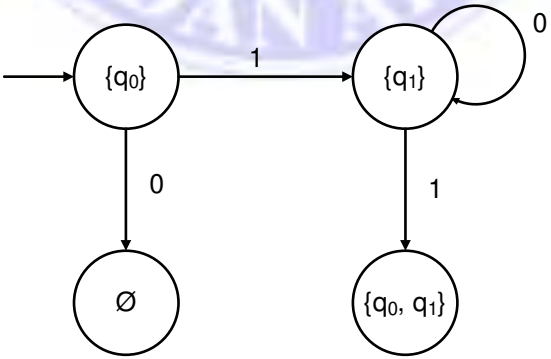
b. Tahap 2 : *(State q_0 terhadap input 1)*



c. Tahap 3 : *(State q_1 terhadap input 0)*

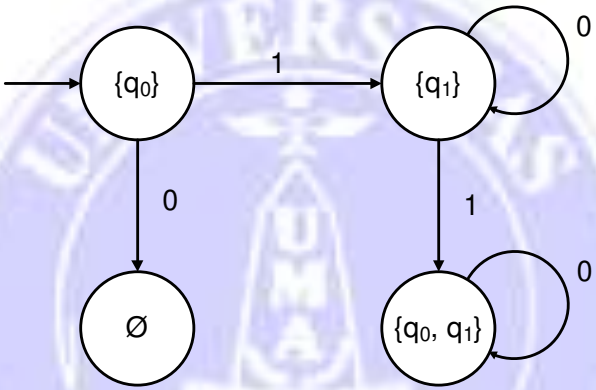


d. Tahap 4 : *(State q_1 terhadap input 1)*



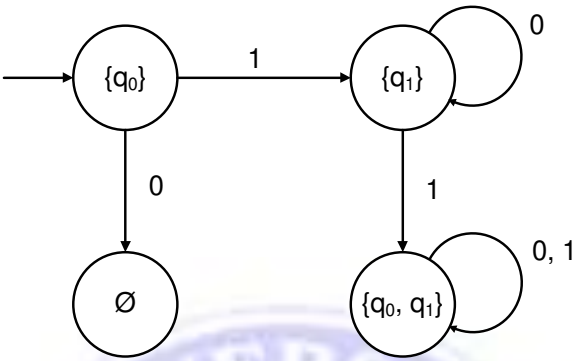
e. Tahap 5 :

- *State* $\{q_0, q_1\}$ diberi *input* 0 = *State* $\{q_0, q_1\}$.
  Karena $\delta\ (q_0, 0) = \emptyset$ dan $\delta\ (q_1, 0) = \{q_1\}$, maka :
  $\delta\ ([q_0, q_1], 0) = \{q_0, q_1\}$.
- Sehingga kita tambahkan *input* 0 pada *state* $\{q_0, q_1\}$.



f. Tahap 6 :

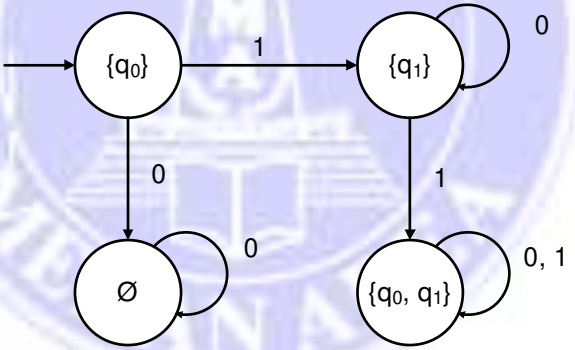
- *State* $\{q_0, q_1\}$ diberi *input* 1 = *State* $\{q_0, q_1\}$.
  Karena $\delta\ (q_0, 1) = \{q_1\}$ dan $\delta\ (q_1, 1) = \{q_0, q_1\}$,
  maka : $\delta\ ([q_0, q_1], 1) = \{q_0, q_1\}$.
- Sehingga kita tambahkan *input* b pada *state* $\{q_0, q_1\}$.



g. Tahap 7 : Perhatikan tabel transisi, dimana :

δ (q_0, 0) = Ø

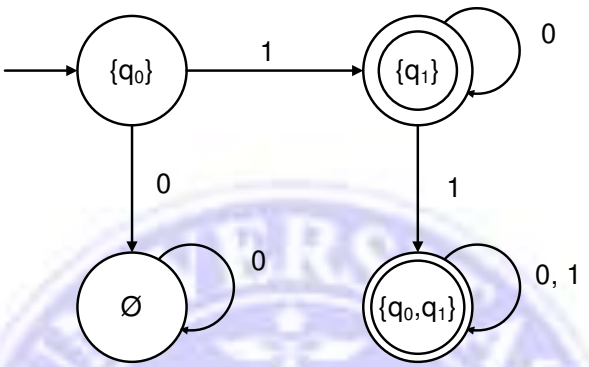
- Dari fungsi transisi diatas, *state* Ø menerima *input* 0. Sehingga kita bisa menambahkan *input* 0 pada *state* Ø.



h. Tahap 8 : *Final State*

Apabila kita lihat kembali mesin NFA, *final state* terletak pada *state* {q_1}. Maka *final state* mesin DFA yang baru adalah semua *state* yang mengandung {q_1}. Maka *final state* adalah {q_1} dan {q_0, q_1}.

Setelah melalui 8 tahapan ekuivalensi, berikut ini adalah mesin DFA yang ekuivalen dengan mesin NFA.



Rangkuman

1. Ekuivalensi mesin NFA ke mesin DFA merupakan proses merubah mesin NFA menjadi DFA, ataupun membuat mesin DFA dari mesin NFA.
2. Terdapat 8 langkah untuk merubah mesin NFA menjadi DFA yang dimulai dari menggambar mesin DFA sesuai dengan tabel transisi mesin NFA, sampai menentukan *final state* dari mesin DFA yang baru terbentuk.
3. Pada mesin DFA yang ekuivalen dengan mesin NFA, terdapat *state* himpunan kosong (∅) yang dapat terbentuk karena tidak adanya *state* tujuan dari sebuah *input*.
4. Penentuan *final state* pada mesin DFA yang baru dapat dipengaruhi oleh *state* lain yang mengandung *final state* pada mesin NFA sebelumnya.

5. Sebuah *state* himpunan kosong (∅) dapat digambarkan memiliki input apabila pada tabel transisi terdapat *input* dari state awal menuju *state* himpunan kosong.

Tugas

1. Buatlah *Deterministic Finite Automata* yang ekuivalen dengan *Non Deterministic Finite Automata* berikut :

   $Q = \{A, B, C, D\}$

   $\Sigma = \{0, 1\}$

   $S = A$

   $F = \{D\}$

   Fungsi transisinya dinyatakan dalam tabel transisi :

δ	0	1
A	A, B	B
B	B, C	C
C	D	-
D	-	C

2. Buatlah *Deterministic Finite Automata* yang ekuivalen dengan *Non Deterministic Finite Automata* berikut :

   $Q = \{q_1, q_2, q_3\}$

   $\Sigma = \{0, 1\}$

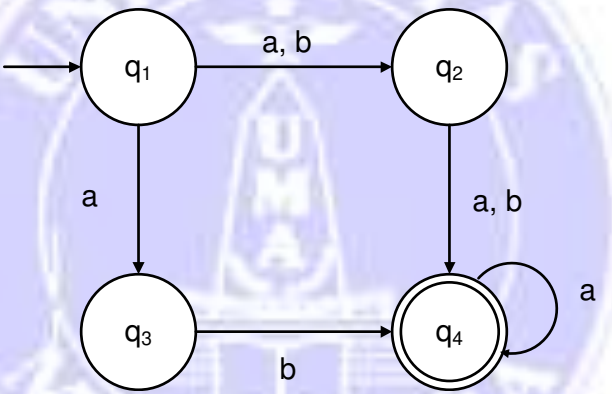
   $S = q_1$

   $F = \{q_3\}$

Fungsi transisinya dinyatakan dalam tabel transisi :

δ	0	1
q_1	q_2	-
q_2	q_1, q_3	q_1
q_3	-	q_1

3. Buatlah mesin *Deterministic Finite Automata* yang ekuivalen dengan mesin *Non Deterministic Finite Automata* berikut :



BAB V
NON DETERMINISTIC FINITE AUTOMATA (NFA) DENGAN ε – MOVE

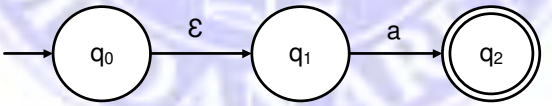
Capaian Pembelajaran Mata Kuliah

- Mahasiswa dapat memahami NFA *(Non Deterministic Finite Automata*) dengan ε - *move.*

5.1 Konsep Dasar NFA dengan ε - *move*

Pada NFA dengan ε - *move, state* tujuan dapat ditelusuri tanpa harus membaca *input* dari *state* sebelumnya. Artinya dengan adanya ε - *move,* sebuah *state* tidak membaca *input* untuk menuju *state* berikutnya.

Contoh :



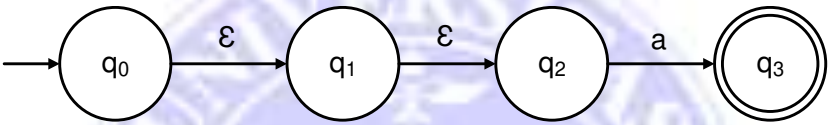
Gambar 5.1 Mesin NFA dengan ε - *move*

Pada gambar 5.1 di atas, dengan adanya ε (*epsilon*) di antara q_0 dan q_1, maka pembacaan *state* dari q_0 dapat langsung menuju q_1 walaupun tidak ada *input*.

5.2 Membuat *ε- Closure* dari mesin NFA dengan *ε – move*

NFA dengan *ε – move* memiliki sebuah komponen lain yaitu *ε- Closure*. Secara sederhana, *ε- closure* dapat diartikan sebagai kumpulan *state-state* yang dapat ditelusuri tanpa harus membaca *input*.

Contoh :



Gambar 5.2 Mesin NFA dengan *ε - move*

Dari contoh gambar 5.2 diatas, ada dua hal yang penting dalam menentukan *ε – closure*, yaitu :

- Pertama kali *ε – closure* dibentuk oleh q_0, yang diikuti dengan *state* lain yang masih berhubungan dengan q_0, namun masih tetap memiliki transisi *ε*. Secara rinci dapat dilihat pada *ε – closure* q_0 dan q_1.

  ε- closure $(q_0) = \{q_0, q_1, q_2\}$

  ε- closure $(q_1) = \{q_1, q_2\}$

- Bagi *state* yang tidak memiliki *ε* dalam arah menuju *state* berikutnya, maka *ε – closure* dari *state* tersebut adalah *state*

itu sendiri. Secara rinci dapat dilihat pada ε – *closure* q_2 dan q_3

ε- *closure* $(q_2) = \{q_2\}$

ε- *closure* $(q_3) = \{q_3\}$

Dari langkah penentuan ε- *closure* sebelumnya, maka secara lengkap ε- *closure* dari gambar 5.2 adalah :

ε- *closure* $(q_0) = \{q_0, q_1, q_2\}$

ε- *closure* $(q_1) = \{q_1, q_2\}$

ε- *closure* $(q_2) = \{q_2\}$

ε- *closure* $(q_3) = \{q_3\}$

5.3 Ekuivalensi NFA dengan ε – *move* ke NFA tanpa ε – *move*

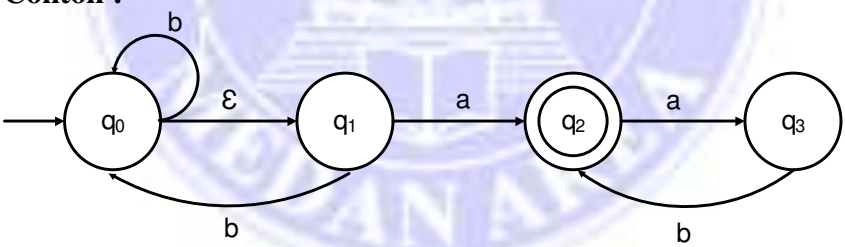
Adapun langkah-langkah ekuivalensi NFA dengan ε – *move* ke NFA tanpa ε – *move* adalah sebagai berikut:

a. Membuat kembali tabel transisi dari mesin NFA ε – *move* yang akan diekuivalensikan.
b. Bentuklah ε – *closure* yang berkaitan antara satu *state* dengan *state* lainnya
c. Telusurilah fungsi transisi yang dapat terbentuk dari perubahan NFA ε – *move* ke NFA tanpa ε – *move* dengan rumus :

δ (*state*, input) = ε*_closure* (δ (ε*_closure* (*state),* input))

d. Bentuklah tabel transisi dari hasil penelusuran fungsi transisi atau yang disebut dengan tabel transisi tanpa ε – *move*
e. Buatlah diagram transisi mesin NFA tanpa ε – *move*. Pada langkah ini, gambarlah semua *state* yang ada pada tabel transisi selain *state* kosong ($\emptyset$).
f. Tentukan *final state* NFA tanpa ε – *move* yang merupakan gabungan dari :
   1) *Final state* mesin NFA dengan ε – *move*
   2) S*tate* lain yang ε – *closure* nya memiliki hubungan dengan *final state* mesin NFA dengan ε – *move*

Contoh :



Gambar 5.3 Mesin NFA dengan *ε – move*

Kita akan melakukan ekuivalensi terhadap mesin NFA dengan ε – *move* pada gambar 5.3 di atas, adapun langkah-langkahnya adalah sebagai berikut:

1. Tabel transisi NFA dengan ε – *move* :

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

2. ε*_closure* :

   ε*_closure* $(q_0) = \{q_0, q_1\}$

   ε*_closure* $(q_1) = \{q_1\}$

   ε*_closure* $(q_2) = \{q_2\}$

   ε*_closure* $(q_3) = \{q_3\}$

3. Fungsi transisi :

   $\delta'(q_0, a) = \varepsilon$*_closure* $(\delta(\varepsilon$*_closure* $(q_0), a))$

   $= \varepsilon$ *_closure* $(\delta(\{q_0, q_1\}, a))$

   $= \varepsilon$ *_closure* (q_2)

   $= \{q_2\}$

   $\delta'(q_0, b) = \varepsilon$*_closure* $(\delta(\varepsilon$*_closure* $(q_0), b))$

   $= \varepsilon$ *_closure* $(\delta(\{q_0, q_1\}, b))$

   $= \varepsilon$ *_closure* (q_0)

   $= \{q_0, q_1\}$

   $\delta'(q_1, a) = \varepsilon$*_closure* $(\delta(\varepsilon$*_closure* $(q_1), a))$

$= \varepsilon\_closure\ (\delta\ (\{q_1\}, a))$

$= \varepsilon\_closure\ (q_2)$

$= \{q_2\}$

$\delta'\ (q_1, b) = \varepsilon\_closure\ (\delta\ (\varepsilon\_closure\ (q_1), b))$

$= \varepsilon\_closure\ (\delta\ (\{q_1\}, b))$

$= \varepsilon\_closure\ (q_0)$

$= \{q_0, q_1\}$

$\delta'\ (q_2, a) = \varepsilon\_closure\ (\delta\ (\varepsilon\_closure\ (q_2), a))$

$= \varepsilon\_closure\ (\delta\ (\{q_2\}, a))$

$= \varepsilon\_closure\ (q_3)$

$= \{q_3\}$

$\delta'\ (q_2, b) = \varepsilon\_closure\ (\delta\ (\varepsilon\_closure\ (q_2), b))$

$= \varepsilon\_closure\ (\delta\ (\{q_2\}, b))$

$= \varepsilon\_closure\ (\emptyset)$

$= \emptyset$

$\delta'\ (q_3, a) = \varepsilon\_closure\ (\delta\ (\varepsilon\_closure\ (q_3), a))$

$= \varepsilon\_closure\ (\delta\ (\{q_3\}, a))$

$= \varepsilon\_closure\ (\emptyset)$

$= \emptyset$

$\delta'\ (q_3, b) = \varepsilon\_closure\ (\delta\ (\varepsilon\_closure\ (q_3), b))$

$= \varepsilon\_closure\ (\delta\ (\{q_3\}, b))$

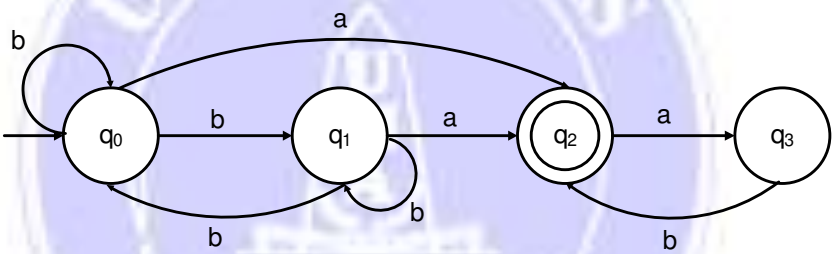
$= \varepsilon\_closure\ (q_2)$

$= \{q_2\}$

4. Tabel transisi NFA tanpa ε – *move* :

δ	a	b
q_0	$\{q_2\}$	$\{q_0, q_1\}$
q_1	$\{q_2\}$	$\{q_0, q_1\}$
q_2	$\{q_3\}$	$\emptyset$
q_3	$\emptyset$	$\{q_2\}$

5. NFA tanpa ε – *move* :

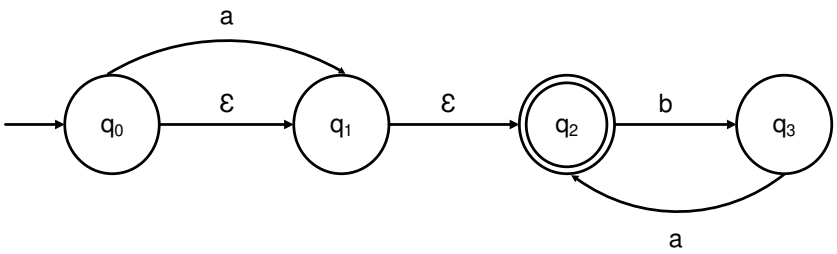


6. *Final state* :

   State q_2 merupakan *state final* mesin NFA dengan ε – *move*. Apabila kita lihat kembali pada tahap 2, pada ε*_closure* tidak terdapat *state* lain yang memuat *state* q_2. Sehingga *state final* untuk NFA tanpa ε – *move* tetap *state* q_2.

Contoh Soal 5.1

Buatlah mesin NFA tanpa ε – *move* yang ekuivalen dengan mesin NFA dengan ε – *move* berikut ini :



Penyelesaian Contoh Soal 5.1

Langkah-langkah ekuivalensi :

1. Tabel transisi :

δ	A	b
q_0	$\{q_1\}$	$\emptyset$
q_1	$\emptyset$	$\emptyset$
q_2	$\emptyset$	$\{q_3\}$
q_3	$\{q_2\}$	$\emptyset$

2. *ε_closure* :

   ε_closure $(q_0) = \{q_0, q_1, q_2\}$

   ε_closure $(q_1) = \{q_1, q_2\}$

   ε_closure $(q_2) = \{q_2\}$

   ε_closure $(q_3) = \{q_3\}$

3. Fungsi transisi :

   $\delta'(q_0, a)$ = *ε_closure* $(\delta$ (*ε_closure* $(q_0), a))$

   = *ε _closure* $(\delta (\{q_0, q_1, q_2\}, a))$

$= \varepsilon\_closure\ (q_1)$

$= \{q_1, q_2\}$

$\delta'\ (q_0, b) = \varepsilon\_closure\ (\delta\ (\varepsilon\_closure\ (q_0), b))$

$= \varepsilon\_closure\ (\delta\ (\{q_0, q_1, q_2\}, b))$

$= \varepsilon\_closure\ (q_3)$

$= \{q_3\}$

$\delta'\ (q_1, a) = \varepsilon\_closure\ (\delta\ (\varepsilon\_closure\ (q_1), a))$

$= \varepsilon\_closure\ (\delta\ (\{q_1, q_2\}, a))$

$= \varepsilon\_closure\ (\emptyset)$

$= \emptyset$

$\delta'\ (q_1, b) = \varepsilon\_closure\ (\delta\ (\varepsilon\_closure\ (q_1), b))$

$= \varepsilon\_closure\ (\delta\ (\{q_1, q_2\}, b))$

$= \varepsilon\_closure\ (q_3)$

$= \{q_3\}$

$\delta'\ (q_2, a) = \varepsilon\_closure\ (\delta\ (\varepsilon\_closure\ (q_2), a))$

$= \varepsilon\_closure\ (\delta\ (\{q_2\}, a))$

$= \varepsilon\_closure\ (\emptyset)$

$= \emptyset$

$\delta'\ (q_2, b) = \varepsilon\_closure\ (\delta\ (\varepsilon\_closure\ (q_2), b))$

$= \varepsilon\_closure\ (\delta\ (\{q_2\}, b))$

$= \varepsilon\_closure\ (q_3)$

$= \{q_3\}$

δ' (q_3, a) = ε_*closure* (δ (ε_*closure* (q_3), a))

= ε _*closure* (δ ({q_3}, a))

= ε _*closure* (q_2)

= {q_2}

δ' (q_3, b) = ε_*closure* (δ (ε_*closure* (q_3), b))

= ε _*closure* (δ ({q_3}, b))

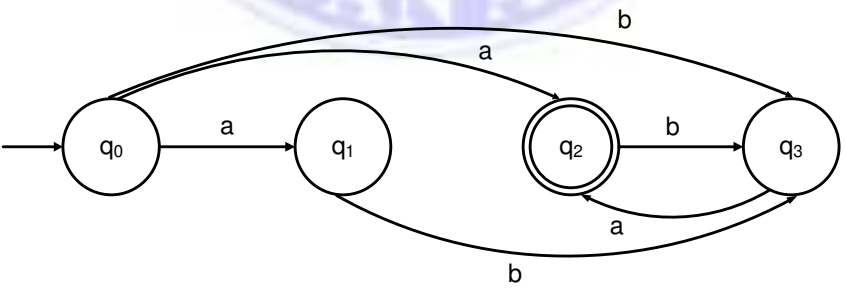
= ε _*closure* ($\emptyset$)

= $\emptyset$

4. Tabel transisi NFA tanpa ε – *move* :

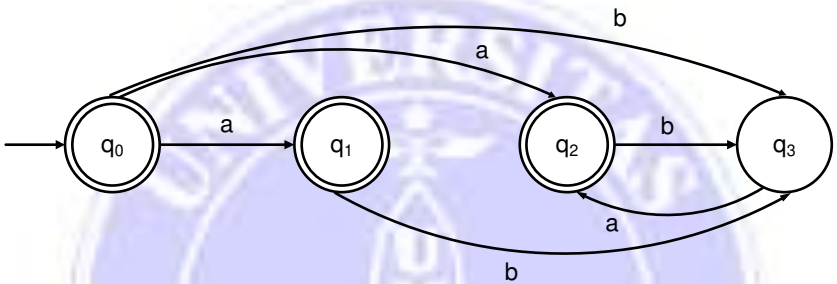
δ	a	B
q_0	{q_1, q_2}	{q_3}
q_1	$\emptyset$	{q_3}
q_2	$\emptyset$	{q_3}
q_3	{q_2}	$\emptyset$

5. NFA tanpa ε – *move* :



6. *Final state* :

   State q_2 merupakan *state final* mesin NFA dengan ε – *move*. Apabila kita lihat kembali pada tahap 2, pada ε*_closure* terdapat *state* lain yang memuat *state* q_2, yaitu : q_0 dan q_1. Sehingga *state final* untuk NFA tanpa ε – *move* adalah *state* q_0, q_1, dan q_2.



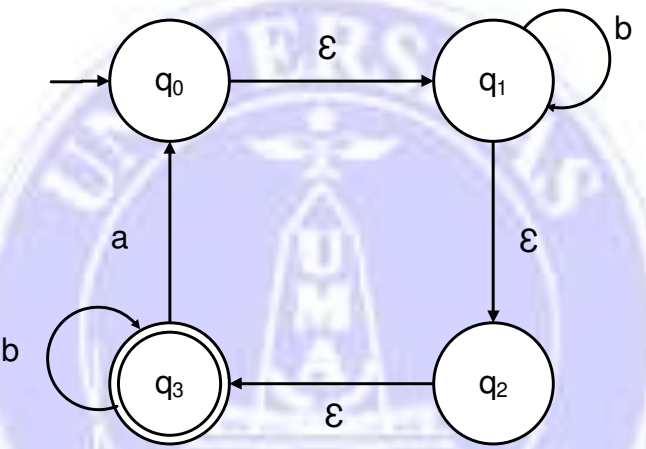
Rangkuman

1. Mesin NFA dengan ε - *move* ditandai dengan adanya sebuah *state* tujuan yang dapat ditelusuri tanpa harus membaca *input* dari *state* sebelumnya.
2. ε – *closure* dari sebuah *state* adalah himpunan dari dirinya sendiri dan diikuti dengan *state* lain yang masih berhubungan dengan *state* tersebut melalui transisi ε.
3. Sebuah *state* yang tidak memiliki transisi ε dalam arah menuju *state* berikutnya, ε – *closure*-nya adalah *state* itu sendiri.
4. Proses ekuivalensi sebuah mesin NFA dengan ε – *move* menjadi mesin NFA tanpa ε – *move* memiliki beberapa tahapan, yang dimulai dari tahap membuat tabel transisi,

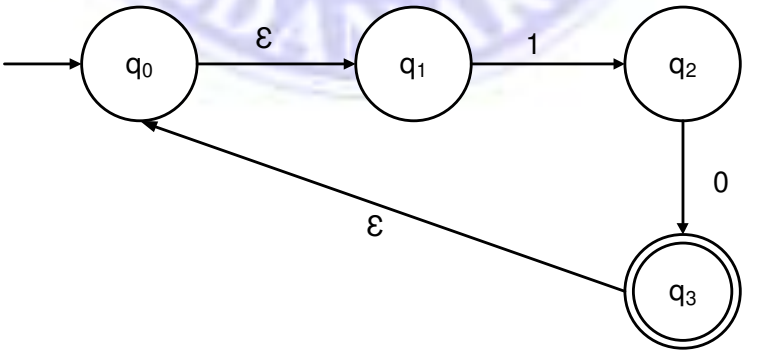
menentukan ε – *closure*, membuat tabel transisi untuk NFA tanpa ε – *move*, dan diakhiri dengan membuat diagram transisi mesin NFA tanpa ε – *move*.

Tugas

1. Buatlah ε – *closure* dari NFA dengan ε – *move* berikut :



2. Buatlah mesin NFA tanpa ε – *move* dari mesin NFA dengan ε – *move* berikut ini :



3. Buatlah mesin NFA tanpa ε – *move* dari konfigurasi mesin NFA dengan ε – *move* berikut ini :

   $Q = \{q_0, q_1, q_2, q_3\}$

   $\Sigma = \{0, 1\}$

   $S = q_0$

   $F = q_3$

   ε*_closure* :

   a. *State* q_0 adalah q_0 dan q_1

   b. *State* q_1 adalah q_1

   c. *State* q_2 adalah q_2

   d. *State* q_3 adalah q_2 dan q_3

   Tabel transisi :

δ	0	1
q_0	$\{q_2\}$	$\emptyset$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\emptyset$	$\emptyset$
q_3	$\emptyset$	$\{q_1\}$

BAB VI
EKSPRESI REGULER (ER)

Capaian Pembelajaran Mata Kuliah :

- ❖ Mahasiswa dapat memahami ekspesi regular dari sebuah mesin automata

6.1 Notasi Ekspresi Reguler

Ada beberapa notasi penting dalam ekspresi reguler, yaitu:

No	Notasi	Keterangan
1	*Asterisk* (*)	*String* bisa tidak muncul, namun bisa juga muncul hingga n kali
2	*Superscript* ($^+$)	*String* minimal muncul satu kali
3	*Union* (+ atau ∪)	*String* yang muncul tetap melalui proses pemilihan
4	Konkatenasi/titik (.)	Penggabungan *string*

Contoh :

a. ER : cd*e
   - ➢ *String* : ce, cdde, cddde

b. ER : xy^+z
   - *String* : xyz, xyyz, xyyyz

c. ER : a + b*
   - *String* : a, b, bb

d. ER : m* ∪ n
   - *String* : n, m, mm

e. ER : ab* ∪ m*n
   - *String* : a, ab, n, mn, abb, mmn

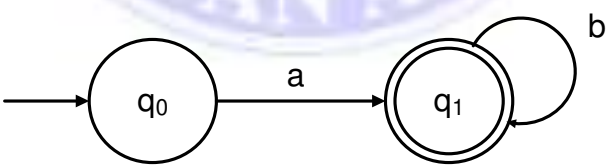
f. ER : x.y + a.b
   - *String* : xy, ab

6.2 Implementasi Ekspresi Reguler pada Mesin FSA

6.2.1 Membaca ER dari Mesin FSA

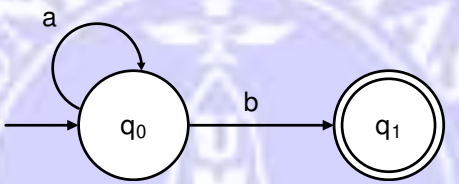
Membaca ekspresi reguler dari sebuah mesin FSA dapat dilakukan apabila sudah memahami notasi ER beserta *string* yang dapat dibangkitkan.

Contoh :

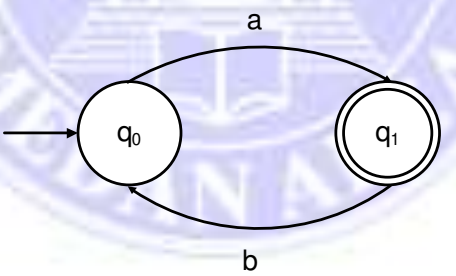


Gambar 6.1 Mesin NFA untuk ER : ab*

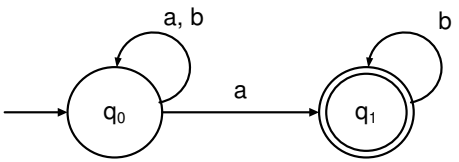
Pada gambar 6.1, terdapat satu *state* awal dan *state final*. Ekspresi reguler dari mesin NFA diatas dibaca dari arah panah yang keluar dari *state* q_0 menuju q_1, kemudian diikuti dengan *state* q_1 terhadap *input* b yang *looping* ke dirinya sendiri. Nilai b dapat muncul berulang lebih dari satu kali. Sehingga ER dari mesin NFA diatas adalah ab*. Cara membaca ER dari mesin NFA dan mesin NFA dengan ε – *move* juga dapat dilihat pada beberapa contoh di bawah ini:



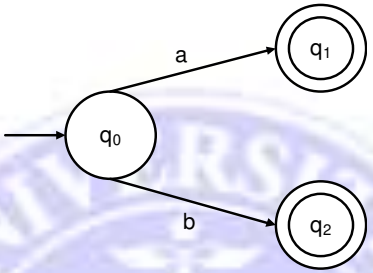
Gambar 6.2 Mesin NFA untuk ER : a*b



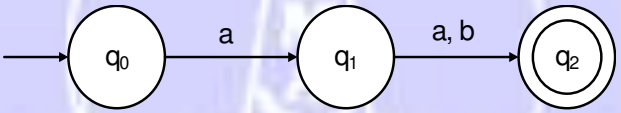
Gambar 6.3 Mesin NFA untuk ER : a (ba)*



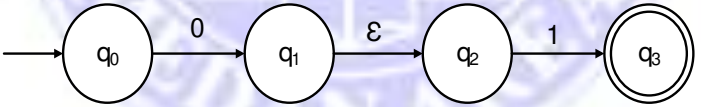
Gambar 6.4 Mesin NFA untuk ER : (ab)* ab*



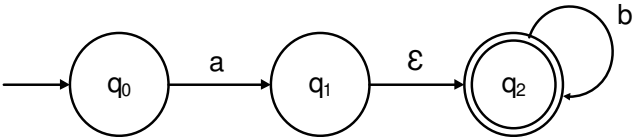
Gambar 6.5 Mesin NFA untuk ER : a ∪ b



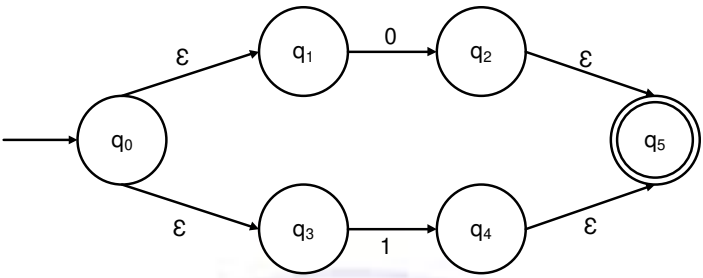
Gambar 6.6 Mesin NFA untuk ER : a (a ∪ b)



Gambar 6.7 Mesin NFA $\varepsilon - move$ untuk ER : 01



Gambar 6.8 Mesin NFA $\varepsilon - move$ untuk ER : ab*



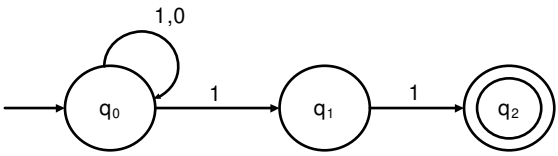
Gambar 6.9 Mesin NFA $\varepsilon - move$ untuk ER : 0 ∪ 1

6.2.2 Membuat Mesin NFA dari ER

Pada sub bab sebelumnya sudah dipaparkan cara membaca ekspresi reguler dari mesin NFA dan NFA dengan $\varepsilon - move$. Pada sub bab berikut ini, kita akan membuat diagram mesin NFA dari ekspresi reguler yang diberikan. Dalam membuat mesin NFA dari ER, diperlukan kemampuan menganalisis *state* yang sesuai dengan notasi ekspresi reguler.

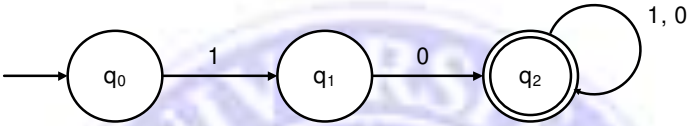
Contoh :

- ER : (1+0)*11
  Tanda + merupakan notasi pemilihan *string* yang akan dibangkitkan, sedangkan ER : (1+0)*11
  Tanda * merupakan input *string* yang akan berulang *(looping)*.
  Sehingga diagram NFA menjadi :



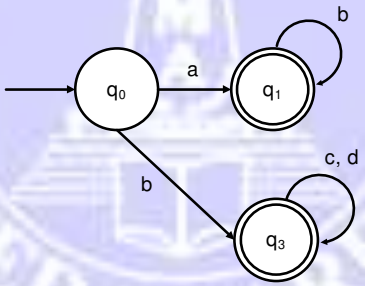
Gambar 6.10 Mesin NFA untuk ER : (1+0)*11

- ER : 10 (1+0)*



Gambar 6.11 Mesin NFA untuk ER : 10(1+0)*

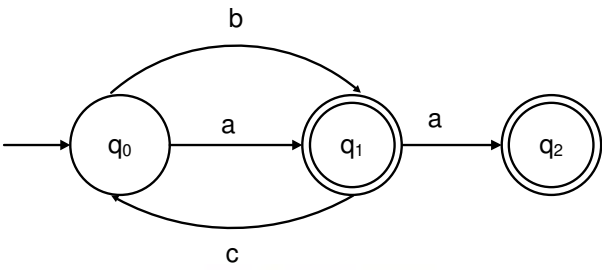
- ER : ab* ∪ b (cd)*



Gambar 6.12 Mesin NFA untuk ER : ab* ∪ b (cd)*

Contoh Soal 6.1

Buatlah ekspresi reguler dari mesin NFA berikut ini:

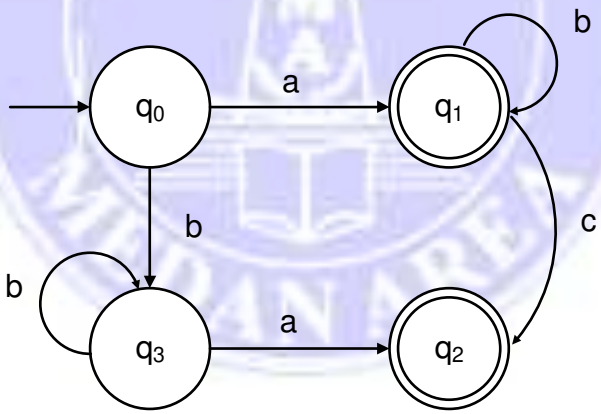


Penyelesaian Contoh Soal 6.1

Ekspresi reguler dari mesin NFA: (bcb)* ∪ a ∪ aa*

Contoh Soal 6.2

Buatlah ekspresi reguler dari mesin NFA berikut ini:



Penyelesaian Contoh Soal 6.2

Ekspresi reguler dari mesin NFA : ab* ∪ ab*c ∪ bb*a

Contoh Soal 6.3

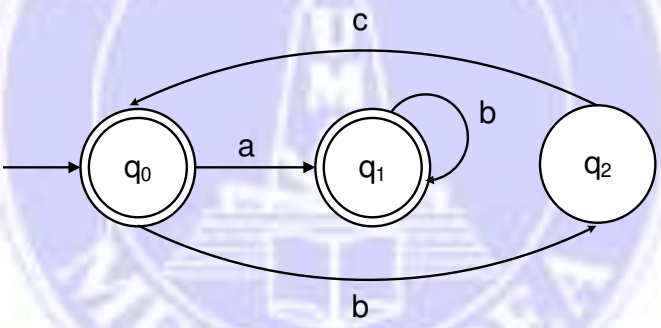
Buatlah mesin NFA dari ekspresi reguler beikut ini:

a. ER : a*b ∪ (bc)*

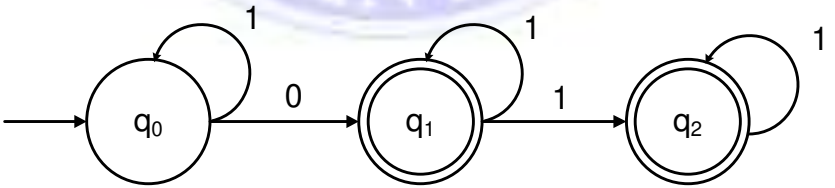
b. ER : 011* + 1*01*

Penyelesaian Contoh Soal 6.3

a.



b.



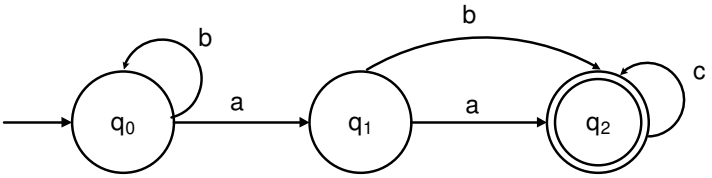
Rangkuman

1. Dalam implementasi ekspresi reguler, terdapat dua notasi ekspresi reguler yang sering digunakan, yaitu :
   - *Asterisk* (*) => digunakan untuk membuat *string* muncul lebih dari satu kali atau tidak muncul sama sekali
   - *Union* (+ atau ∪) => digunakan untuk membuat pilihan terhadap *string* mana yang akan muncul
2. Sebuah mesin NFA dapat memiliki beberapa pola ekspresi reguler, tergantung dari daya nalar seseorang yang membacanya. Artinya dalam membaca ataupun membuat ekspresi reguler (ER), setiap orang dapat berkreasi sesuai kreatifitasnya sendiri, namun tetap mengikuti aturan ekspresi reguler yang telah ditentukan.

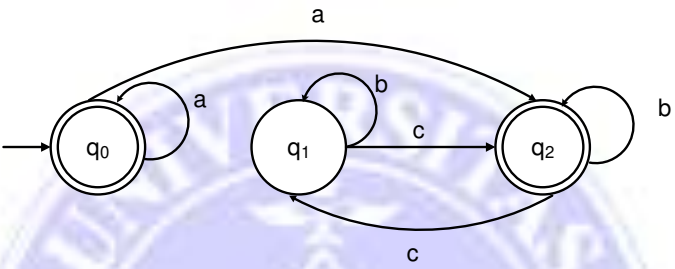
Tugas

1. Tuliskan minimal 3 untai *string* yang dapat dibangkitkan dari ER berikut :
   a. a*bc + (cd*)a*
   b. (001)* + 11^{+}00*
   c. xy*z^{+} ∪ a*b^{+} c*

2. Tentukan ekspresi reguler dari mesin NFA berikut :
   a.



b.



3. Buatlah ekspresi reguler dari mesin NFA berikut :

$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{a, b, c, d\}$

$S = q_0$

$F = \{q_2, q_3\}$

δ	a	b	c	d
q_0	$\{q_0, q_1\}$	$\emptyset$	$\emptyset$	$\emptyset$
q_1	$\{q_2\}$	$\{q_1, q_2\}$	$\emptyset$	$\emptyset$
q_2	$\emptyset$	$\emptyset$	$\{q_3\}$	$\emptyset$
q_3	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_2\}$

BAB VII
ATURAN PRODUKSI *FINITE STATE AUTOMATA* (FSA)

Capaian Pembelajaran Mata Kuliah

- Mahasiswa dapat memahami aturan produksi suatu FSA *(Finite State Automata)*

7.1. Aturan Produksi Tata Bahasa Reguler dari Mesin FSA

Aturan produksi tata bahasa reguler berbentuk:

$$\alpha \rightarrow \beta$$

α = Ruas sisi kiri harus berisi sebuah simbol variabel

β = Ruas sisi kanan maksimal ada sebuah simbol variabel yang terletak di sisi paling kanan

Secara formal, konfigurasi aturan produksi tata bahasa reguler dapat dituliskan sebagai berikut:

V = Kumpulan simbol variabel atau non terminal

T = Kumpulan simbol terminal

P = Kumpulan aturan produksi

S = Simbol awal

Berikut ini langkah-langkah membentuk aturan produksi tata bahasa reguler dari mesin FSA:

1. Berikan simbol berupa variabel pada setiap *state* FSA, seperti : A, B, C, D, S, E, dan sebagainya.
2. Bentuklah aturan produksi untuk setiap *state* yang akan menuju kepada *state* berikutnya.
3. Gunakan tanda panah (→) untuk menghubungkan satu *state* ke *state* lainnya, serta gunakan tanda " | " pada *state* tujuan yang sama.

Catatan :

a. Apabila masih terdapat *input* dari *final state*, baik menuju *state* nya sendiri maupun *state* lain, *state* ini juga akan dibuatkan perumpamaan berupa variabel.

b. Apabila pada *final state* tidak lagi terdapat *input* yang menuju *state* itu sendiri maupun *state* lain, maka aturan produksi dapat dibangun dari hubungan antara sebuah variabel *state* dan *input* saja, tanpa menambahkan variabel *final state*. (Lihat penerapannya pada penyelesaian contoh soal No. 1 langkah 4 atau pada contoh singkat di bawah ini) :

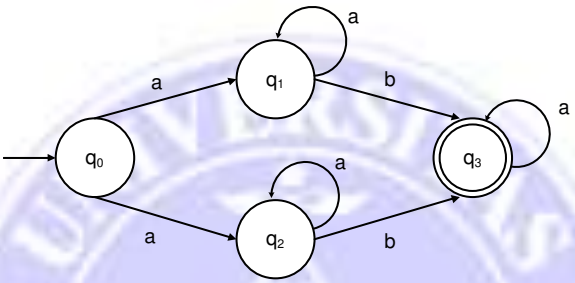
Contoh : *(final state q_3 tidak memiliki input ke state manapun)*

Aturan produksi antara *state* q_2 dan *state* q_3 :

- *State* q_2 = B
- *Input* = c

❖ Aturan produksi : B → c

Contoh :



Gambar 7.1 Mesin FSA

Langkah-langkah membuat aturan produksi dari mesin FSA, adalah sebagai berikut :

a. Pada gambar mesin FSA diatas, dapat kita umpamakan:

- *State* q_0 = S
- *State* q_1 = C
- *State* q_2 = D
- *State* q_3 = E

b. Aturan produksi antara *state* q_0 dan *state* q_1:

- *State* q_0 = S
- *State* q_1 = C
- *Input* = a

❖ Aturan produksi : S → aC

c. Aturan produksi antara *state* q_0 dan *state* q_2 :

- *State* q_0 = S
- *State* q_2 = D
- *Input* = a
- Aturan produksi : S → aD

d. Aturan produksi untuk *state* q_1 dan *input* a :

- *State* q_1 = C
- *Input* = a
- Aturan produksi : C → aC

e. Aturan produksi untuk *state* q_2 dan *input* a :

- *State* q_2 = D
- *Input* = a
- Aturan produksi : D → aD

f. Aturan produksi antara *state* q_1 dan *state* q_3 :

- *State* q_1 = C
- *State* q_3 = E
- *Input* = b
- Aturan produksi : C → bE

g. Aturan produksi antara *state* q_2 dan *state* q_3 :

- *State* q_2 = D
- *State* q_3 = E
- *Input* = b
- Aturan produksi : D → bE

h. Aturan produksi untuk *state* q_3 dan *input* a :

- *State* q_3 = E
- *Input* = a
- Aturan produksi : E → aE

Aturan produksi secara lengkap dapat dituliskan sebagai berikut :

$$S \rightarrow aC \mid aD$$

$$C \rightarrow aC \mid bE$$

$$D \rightarrow aD \mid bE$$

$$E \rightarrow aE$$

Secara formal tata bahasa reguler yang diperoleh dapat dituliskan:

V = {S, C, D, E}

T = {a, b}

P = { S → aC | aD , C → aC | bE , D → aD | bE , E → aE }

S = S

7.2. Konversi Aturan Produksi ke Mesin FSA

Untuk dapat membuat mesin FSA dari aturan produksi tata bahasa reguler, dibutuhkan pemahaman tentang hubungan antara sebuah mesin FSA dengan aturan produksi tata bahasa reguler.

Berikut langkah-langkah membuat mesin FSA dari aturan produksi tata bahasa reguler:

1. Mencermati aturan produksi dengan baik, perhatikan dan perkirakan beberapa variabel yang akan ada hubungannya dengan *state* yang akan dibangun.
2. Mencermati aturan produksi dengan memperhatikan hubungan antara *input* dan *state* yang mungkin akan terbentuk dari langkah 1.
3. Hubungkan *state* dengan *input* sesuai dengan aturan produksi
4. Tentukan *final state* sesuai dengan urutan aturan produksi. *Final state* dapat dilihat dari akhir alur aturan produksi yang hanya menuju ke terminal.

Contoh :

Terdapat aturan produksi :

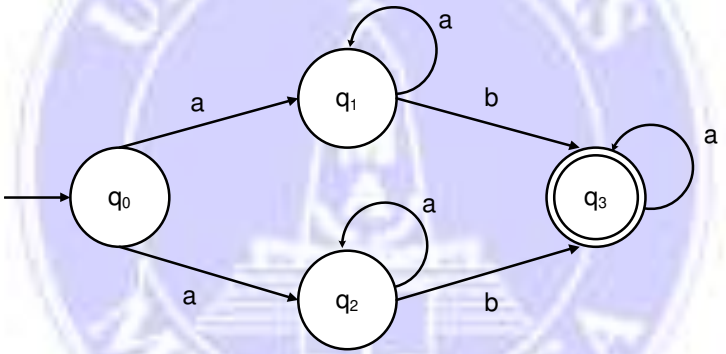
$$S \rightarrow aS \mid cC \mid c$$

$$C \rightarrow dD$$

$$D \rightarrow aS$$

Langkah-langkah membuat mesin FSA dari aturan produksi diatas adalah sebagai berikut :

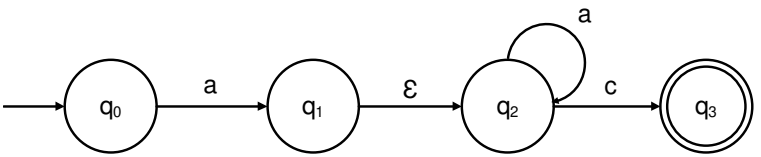
1. Umpamakan :
   a. Variabel S adalah *state* q_0
   b. Variabel C adalah *state* q_1
   c. Variabel D adalah *state* q_2
   d. Variabel S → c, maka kita tambahkan *state* q_3
2. Simbol teminal yang posisinya berada di dekat variabel merupakan *input* dari satu *state* menuju *state* lain.
3. Gambarkan mesin FSA sesuai dengan aturan produksi dan *state* yang telah diumpamakan pada langkah 1.



Gambar 7.2 Mesin FSA Hasil Konversi dari Aturan Produksi

Contoh Soal 7.1

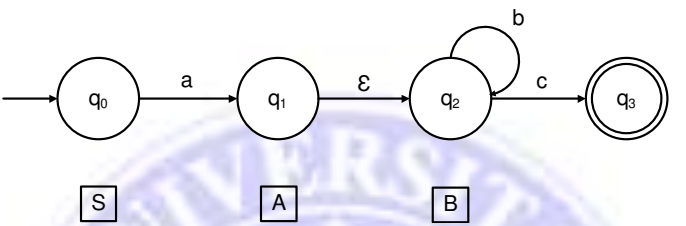
Buatlah aturan produksi dari mesin FSA di bawah ini:



Penyelesaian Contoh Soal 7.1

Langkah-langkah menentukan aturan produksi :

a. Umpamakan setiap *state* sebagai sebuah variabel :



- *State* q_0 = S
- *State* q_1 = A
- *State* q_2 = B

b. Penentuan aturan produksi berdasarkan hubungan *state*, perumpaan variabel dan *input* antar *state*.

1) Aturan produksi antara *state* q_0 dan *state* q_1 :
   - *State* q_0 = S
   - *State* q_1 = A
   - *Input* = a
   - ❖ Aturan produksi : S → aA

2) Aturan produksi antara *state* q_1 dan *state* q_2 :
   - *State* q_1 = A
   - *State* q_2 = B

- *Input* = -
- ❖ Aturan produksi : A → B

3) Aturan produksi antara *state* q_2 dan *input* b :
   - *State* q_2 = B
   - *Input* = b
   - ❖ Aturan produksi : B → bB

4) Aturan produksi antara *state* q_2 dan *state* q_3 :
   - *State* q_2 = B
   - *Input* = c
   - ❖ Aturan produksi : B → c

Aturan produksi secara lengkap dapat dituliskan sebagai berikut :

$$S \rightarrow aA$$

$$A \rightarrow B$$

$$B \rightarrow bB \mid c$$

Secara formal tata bahasa reguler yang diperoleh dapat dituliskan:

V = {S, A, B}

T = {a, b, c}

P = { S → aA , A → B , B → bB | c }

S = S

Contoh Soal 7.2

Buatlah mesin FSA dari aturan produksi dibawah ini:

$$S \rightarrow abC \mid D \mid cC \mid \varepsilon$$

$$C \rightarrow aS \mid a$$

$$D \rightarrow bS$$

Penyelesaian Contoh Soal 7.2

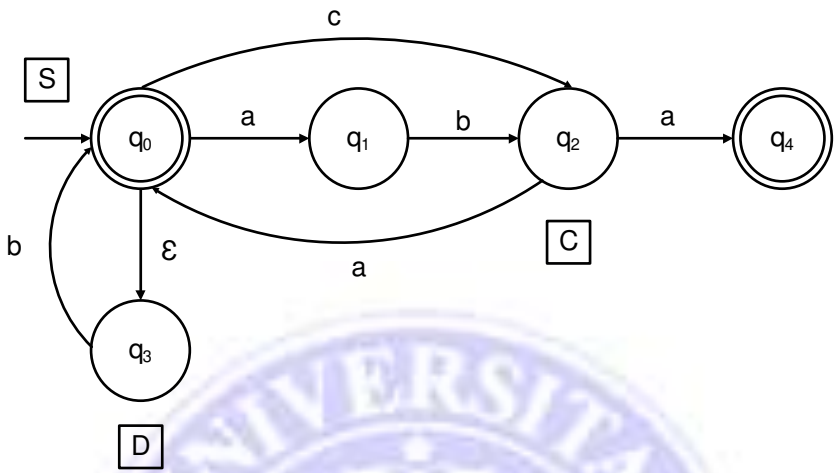
Langkah-langkah membuat mesin FSA dari aturan produksi diatas adalah sebagai berikut :

a. Umpamakan :
   - Variabel S adalah *state* q_0
   - Variabel C adalah *state* q_2
   - Variabel D adalah *state* q_3

b. Simbol teminal yang posisinya berada di dekat variabel merupakan *input*, yaitu : a, b, d

c. Analisis aturan produksi khusus, seperti :
   - $C \rightarrow D$, menunjukkan adanya ε di antara *state* q_0 dengan q_3
   - $S \rightarrow \varepsilon$, menunjukkan bahwa q_0 merupakan *state final*
   - $C \rightarrow a$, menunjukkan bahwa q_4 merupakan *state final*

d. Gambarkan mesin FSA sesuai dengan langkah-langkah yang telah dipaparkan.



Rangkuman

1. Dalam membuat aturan produksi dari mesin FSA ataupun sebaliknya, harus memahami :
   a. Cara mengumpamakan *state* menjadi variabel tertentu.
   b. Cara membentuk aturan produksi dari hubungan *state* asal, *input* dan *state* tujuan
   c. Cara menentukan *final state*
2. Apabila masih terdapat *input* dari *final state*, baik menuju *state* nya sendiri maupun *state* lain, *state* ini juga akan dibuatkan perumpamaan berupa variabel.
3. Apabila pada *final state* tidak lagi terdapat input yang menuju *state* itu sendiri maupun *state* lain, maka aturan produksi dapat dibangun dari hubungan antara sebuah variabel *state* dan input saja, tanpa menambahkan variabel *final state.*

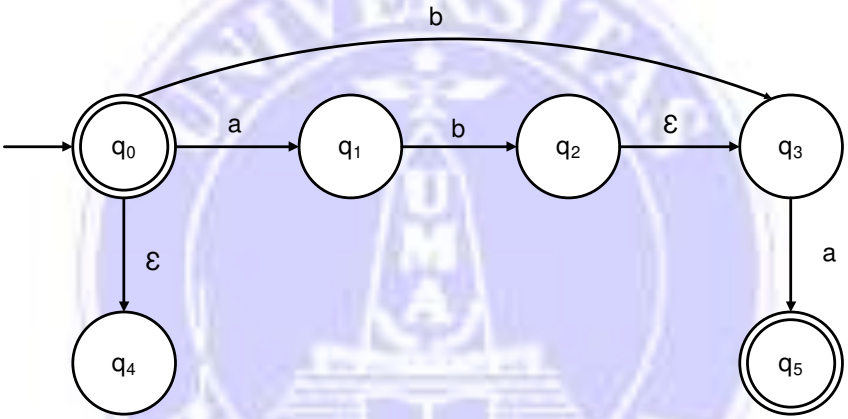
Tugas

1. Buatlah FSA dari aturan produksi berikut :

$$S \rightarrow 1B$$

$$B \rightarrow 01B \mid \varepsilon$$

2. Buatlah aturan produksi tata bahasa reguler dari mesin FSA berikut ini :



BAB VIII
FSA DENGAN *OUTPUT*

Capaian Pembelajaran Mata Kuliah

- Mahasiswa dapat memahami mesin FSA *(Finite State Automata)* dengan *Output*

8.1 Mesin Moore

Mesin Moore merupakan salah satu jenis mesin FSA yang dapat memiliki luaran tertentu dan tidak terbatas pada hanya sebuah *input* diterima atau tidak dapat diterima. Konfigurasi mesin Moore terdiri dari 6 tupel, yaitu M = (Q, Σ, δ, S, Δ, λ), dimana :

Q = *state*

Σ = *input*

δ = fungsi transisi

S = *output* awal

Δ = *output*

λ = fungsi *output* untuk setiap *state*

8.2 Mesin Mealy

Mesin Mealy merupakan salah satu jenis mesin FSA yang memiliki output berdasarkan transisi. Adapun konfigurasi mesin Mealy terdiri dari 6 tupel, yaitu M = (Q, Σ, δ, S, Δ, λ), dimana :

Q = *state*

Σ = *input*

δ = fungsi transisi

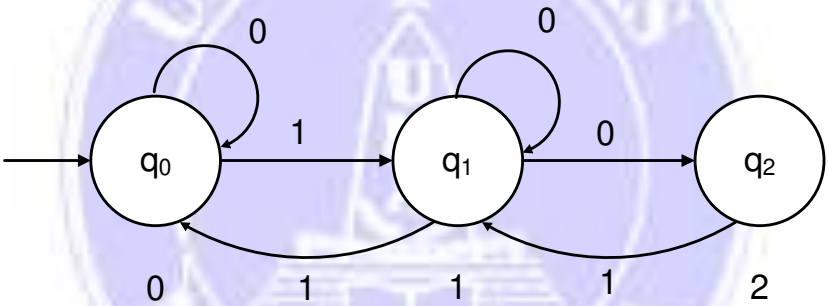
S = *output* awal

Δ = *output*

λ = fungsi *output* untuk setiap transisi

Contoh Soal 8.1

Dengan mesin Moore dibawah ini :



Hitunglah :

a. 7 mod 2

b. 5 mod 2

c. 8 Mod 3

Penyelesaian Contoh Soal 8.1

a. 7 mod 2

- Angka 7 diubah kedalam biner, yaitu : 0111
- Uji coba 0111 ke dalam mesin moore.

- State yang akan dilalui adalah: q_0. q_1, q_0, q_1, dan proses uji coba ini berhenti di state q_1, dan mengeluarkan output 1.
- Hal ini terbukti, karena 7 mod 2 = 1.

b. 5 mod 2

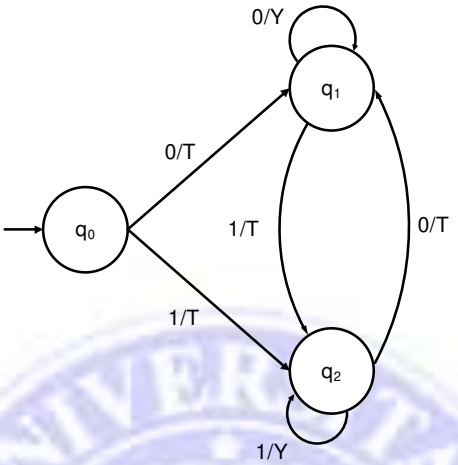
- Angkat 5 diubah ke dalam biner, yaitu : 0101
- Uji coba 0101 ke dalam mesin moore
- State yang akan dilalui adalah : q_0. q_1, q_2, q_1, dan proses uji coba ini berhenti di state q_1, dan mengeluarkan output 1.
- Hal ini terbukti, karena 5 mod 2 = 1.

c. 8 mod 3

- Angkat 8 diubah ke dalam biner, yaitu : 1000
- Uji coba 1000 ke dalam mesin moore
- State yang akan dilalui adalah : q_0. q_1, q_1, q_2, dan proses uji coba ini berhenti di state q_2, dan mengeluarkan output 2.
- Hal ini terbukti, karena 8 mod 3 = 2.

Contoh Soal 8.2

Dengan mesin Mealy berikut, lakukan uji coba apakah mesin dapat menerima *input* berupa 01010, 00101, dan 00011.



Penyelesaian Contoh Soal 8.2

Hasil pengujian mesin Mealy :

a. 01010 : T => Tidak dapat diterima
b. 00101 : T => Tidak dapat diterima
c. 00011 : Y => Dapat diterima

Rangkuman

1. Mesin Moore dapat dimodifikasi sedapat mungkin agar sesuai dengan *output* yang ingin diperoleh. Mesin Moore sering digunakan untuk menyelesaikan permasalahan matermatika.
2. *State* pada setiap mesin Moore merupakan *output* dari hasil pemrosesan *input* permasalahan matematika.
3. Mesin Mealy memberikan keputusan dalam setiap prosesnya, apakah sebuah *input* dapat diterima atau tidak. Mesin Mealy

akan mengeluarkan pernyataan Y apabila *input* dapat diterima, dan juga akan mengeluarkan pernyataan T apabila *input* tidak diterima.

Tugas

1. Gambarkan mesin Moore yang dapat menyelesaikan perhitungan modulo matematika berikut :
   a. 10 mod 3
   b. 13 mod 2
   c. 17 mod 4
2. Gambarkan mesin Mealy yang dapat menerima *input* bilangan biner dan memberikan output bilangan desimal, dimana bilangan dibatasi antara 0 – 5.

BAB IX
POHON PENURUNAN

Capaian Pembelajaran Mata Kuliah

- ❖ Mahasiswa dapat memahami tentang dasar pohon penurunan

9.1. Pohon Penurunan dalam Tata Bahasa Bebas Konteks

Istilah pohon dalam tata bahasa bebas konteks sebenarnya memiliki arti penurunan. Dalam turunan, selalu ada posisi orang tua *(parent)* dan anak *(children).* Pada tata bahasa bebas konteks, *parent* merupakan simbol variabel, sedangkan *children* merupakan simbol terminal. Pohon penurunan dibentuk berdasarkan aturan produksi tata bahasa bebas konteks $\alpha \rightarrow \beta$.

9.2. Pembentukan Pohon Penurunan/ *Parsing*

Pohon penurunan atau *parsing* dapat dilakukan dengan beberapa langkah berikut, yaitu :

1. Pahami setiap aturan produksi yang akan dibuat menjadi pohon penurunan.
2. Cabang pohon penurunan hanya dapat diturunkan dari sebuah simbol variabel (huruf besar).
3. Tentukan model pohon penurunan, apakah penurunan kiri atau penurunan kanan.

4. Setelah pohon penurunan selesai terbentuk, maka bacalah *string* yang diperoleh. Pada pohon penurunan kanan, pembacaan *string*/ terminal dimulai dari sisi paling kanan pohon, sedangkan pada pohon penurunan kiri, pembacaan *string* dimulai dari sisi paling kiri pohon.

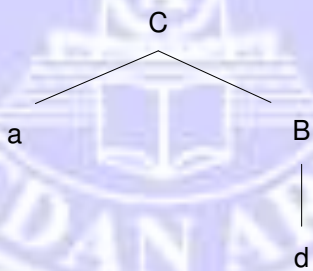
Contoh :

Terdapat aturan produksi tata bahasa bebas konteks seperti berikut ini, buatlah pohon penurunan yang sesuai.

$$C \rightarrow aB$$

$$B \rightarrow d$$

Pohon penurunan :



String yang dihasilkan : ad

9.2.1. Penurunan Kiri *(Leftmost Derivation)*

Penurunan kiri melakukan pembentukan pohon penurunan dari simbol variabel yang paling kiri.

Contoh :

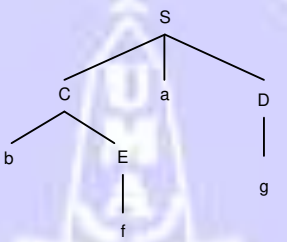
$$S \rightarrow CaD$$

$$C \rightarrow bE$$

$$D \rightarrow g$$

$$E \rightarrow f$$

- Pohon penurunan :



- *String* yang diperoleh = ‘bfag’

9.2.2. Penurunan Kanan *(Rightmost Derivation)*

Penurunan kanan melakukan pembentukan pohon penurunan dari simbol variabel yang paling kanan.

Contoh :

- Aturan Produksi :

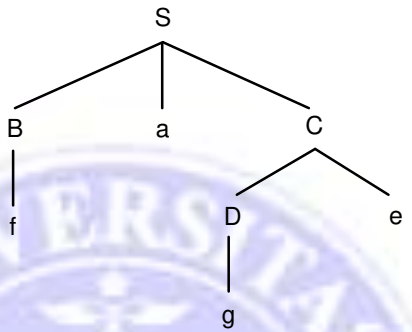
$$S \rightarrow BaC$$

$$C \rightarrow De$$

$$D \rightarrow g$$

$$B \rightarrow f$$

- Pohon penurunan :



- *String* yang diperoleh = ‘fage’

9.2.3. Ambiguitas

Sebuah pohon penurunan dikatakan ambigu apabila terdapat lebih dari satu model pohon penurunan untuk memperoleh sebuah *string*.

Contoh :

- Aturan Produksi :

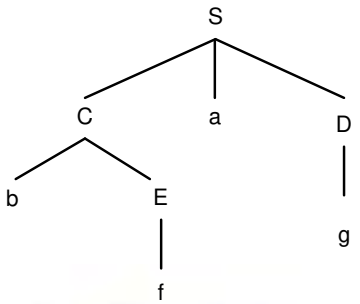
$$S \rightarrow CaD$$

$$C \rightarrow bE$$

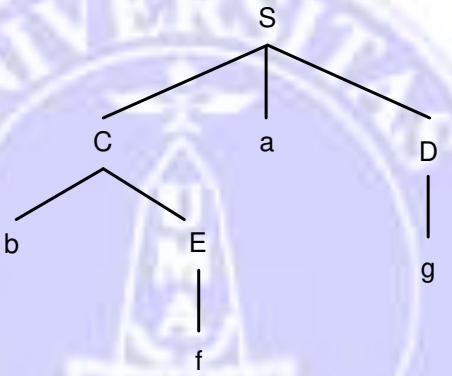
$$D \rightarrow g$$

$$E \rightarrow f$$

- Pohon Penurunan *(Leftmost Derivation → string* : bfag) :



- Pohon Penurunan *(Righmost Derivation → string* : bfag) :



9.3. Membentuk Pohon Penurunan dari *String*

Untuk dapat membentuk sebuah pohon penurunan dari sederetan *string,* dibutuhkan kemahiran dalam membaca aturan produksi secara tepat. Pohon penurunan yang dibentuk berdasarkan *string* yang ingin dicapai, tetap dibentuk dari beberapa aturan produksi tata bahasa bebas konteks, namun pembentukan pohon penurunannya disesuaikan agar memperoleh *string* yang ingin dicapai.

Contoh :

- Aturan Produksi :

$$S \rightarrow cAB$$

$$B \rightarrow CD$$

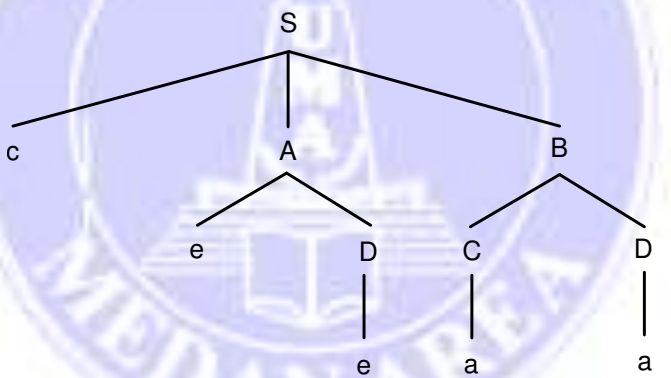
$$A \rightarrow c \mid eD$$

$$D \rightarrow e \mid a$$

$$C \rightarrow a$$

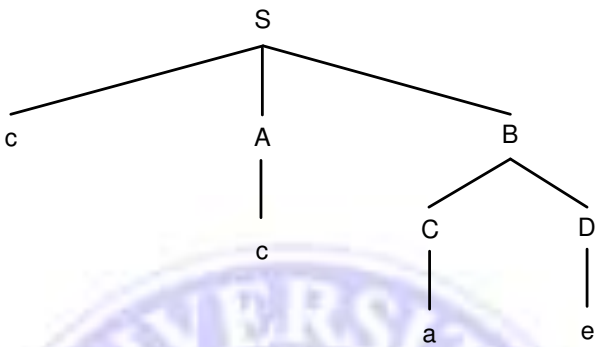
- *String* yang ingin diperoleh : ceeaa

- Pohon Penurunan :



Dari contoh di atas, pohon penurunan dapat dibentuk sedemikian rupa agar sesuai dengan *string* yang menjadi target pencapaian. Namun apabila diperhatikan, dengan aturan produksi yang sama, dapat dihasilkan sebuah pohon penurunan lain yang menghasilkan *string* yang berbeda.

- Pohon Penurunan Versi 2 :



- *String* yang diperoleh : ccae

Contoh Soal 9.1

Buatlah pohon penurunan dari aturan produksi tata bahasa bebas konteks berikut ini :

$$S \rightarrow Cd$$

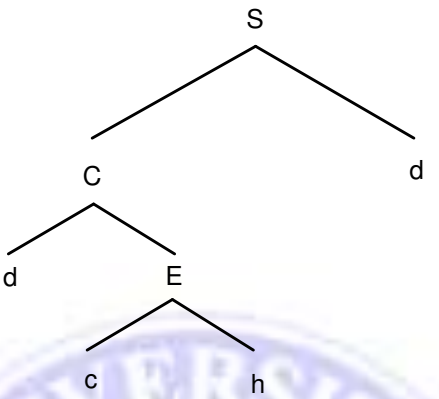
$$C \rightarrow dE \mid F$$

$$F \rightarrow eg$$

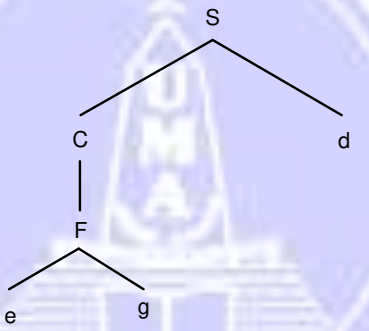
$$E \rightarrow ch$$

Penyelesaian Contoh Soal 9.1

a. Pohon penurunan versi 1 (*String* : dchd)



b. Pohon penurunan versi 2 (*String* : egd)



Contoh Soal 9.2

Buatlah pohon penurunan dari aturan produksi berikut ini agar memperoleh *string* aaabbbc.

$$S \rightarrow AB$$

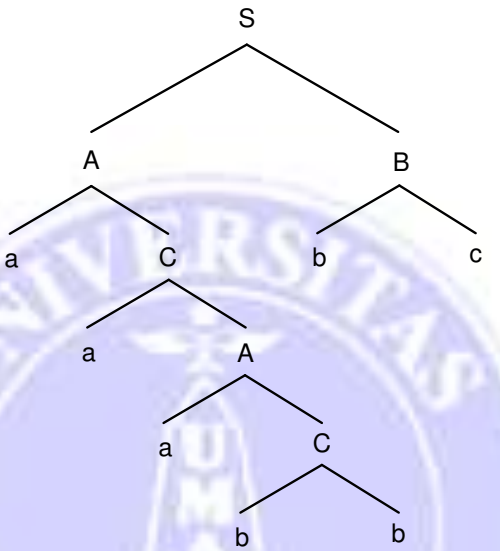
$$A \rightarrow aC \mid a$$

$$C \rightarrow bb \mid aA$$

$$B \rightarrow c \mid bc$$

Penyelesaian Contoh Soal 9.2

Pohon penurunan dengan *string* yang ingin dicapai aaabbbc :



Rangkuman

1. Pohon penurunan atau *parsing* dapat dibuat dalam dua model, yaitu : penurunan kanan *(Righmost Derivation)* dan penurunan kiri *(Leftmost Derivation)*.
2. Aturan produksi yang memiliki turunan adalah aturan produksi yang terdiri dari simbol variabel (huruf besar), sedangkan simbol terminal (huruf kecil) tidak dapat diturunkan lagi.
3. *String* yang diperoleh dari pohon penurunan merupakan urutan terminal yang dibaca dari sisi paling kiri pohon penurunan ke sisi paling kanan pohon penurunan.

Tugas

1. Buatlah beberapa versi pohon penurunan dari aturan produksi tata bahasa bebas konteks dibawah ini :

$$S \rightarrow AbCd \mid ACdE$$

$$A \rightarrow BcD \mid a \mid b$$

$$B \rightarrow bc \mid bC$$

$$C \rightarrow aa \mid cd$$

$$D \rightarrow e$$

$$E \rightarrow cd$$

2. Buatlah pohon penurunan dari aturan produksi tata bahasa bebas konteks berikur ini menggunakan model *Rightmost Derivation* dan *Leftmost Derivation.* Berikanlah kesimpulan apakah pohon penurunan tersebut mengalami ambiguitas atau tidak.

$$S \rightarrow AbD$$

$$A \rightarrow cDB$$

$$B \rightarrow bc$$

$$D \rightarrow e$$

BAB X
PENYEDERHANAAN TATA BAHASA BEBAS KONTEKS

Capaian Pembelajaran Mata Kuliah

❖ Mahasiswa dapat memahami penyederhanaan tata bahasa bebas konteks

10.1. Penghilangan Produksi *Useless*

Penyederhanaan tata bahasa bebas konteks dengan penghilangan produksi *useless* atau produksi yang tidak berguna/ berlebihan, dapat dilakukan dengan beberapa cara, yaitu :

a. Menghilangkan aturan produksi yang mengandung variabel yang tidak menghasilkan terminal
b. Menghilangkan aturan produksi yang tidak dapat dicapai dari simbol awal

Contoh :

S → aSb | Abe | Bdc

A → bda | bDc

C → CCd | b

B → d

D → EE

Dari aturan produksi ini, terdapat aturan produksi yang *useless,* yaitu :

1. Variabel A tidak memiliki terminal, maka aturan produksi A → bda dapat dihilangkan.
2. Dari penghilangan aturan produksi A → bda, maka aturan produksi yang berhubungan, yaitu S → Abe dapat dihilangkan.
3. Aturan produksi C → CCd | b dapat dihapus, karena tidak ada aturan produksi lainnya yang membutuhkan aturan produksi C. Dengan kata lain, aturan produksi ini tidak akan dicapai dari aturan produksi lainnya.
4. Aturan produksi D → EE dapat dihapus, karena tidak ada aturan produksi E yang dapat dicapai dari aturan produksi lainnya.
5. Karena aturan produksi D → EE dihapus, sehingga aturan produksi A → bDc juga dapat dihapus.

Sehingga hasil penyederhanaan tata bahasa bebas konteks, menjadi :

S → aSb | Bdc

B → d

10.2. Penghilangan Produksi Unit

Penyederhanaan tata bahasa bebas konteks dengan penghilangan produksi unit, dapat dilakukan dengan menghapus aturan produksi yang ruas kiri dan ruas kanannya hanya terdiri dari satu simbol variabel dan menggantinya dengan terminal yang sesuai.

Contoh :

S → Sa

S → B

B → gh

B → C

C → d

Penghilangan produksi unit dapat dilakukan dengan :

1. Melihat hubungan antara aturan produksi :

   S → B

   B → gh

   B → C

   C → d

   Dimana aturan produksi S → B dan B → C dapat disederhanakan pada aturan produksi B → gh dan C → d. Sehingga aturan produksi S → B dan B → C dapat dihapus dan diganti dengan S → gh | d.

2. Melihat hubungan antara aturan produksi :

   B → C

   C → d

   Variabel C dapat dihilangkan, sehingga turan produksi dapat lebih sederhana, yaitu : B → d

Sehingga hasil penyederhanaan tata bahasa bebas konteks, menjadi :

S → Sa

S → gh | d

B → d

10.3. Penghilangan Produksi ε

Penyederhanaan tata bahasa bebas konteks dengan penghilangan produksi ε, dapat dilakukan dengan menghapus aturan produksi $\alpha \rightarrow \varepsilon$ (produksi kosong) yang berhubungan dengan variabel lain. Aturan produksi ini juga disebut *nullable*.

Contoh :

S → aBd

B → ε

Aturan produksi B dapat dihilangkan karena termasuk *nullable*, sehingga variabel B pada aturan produksi S → aBd juga dapat dihapuskan.

Sehingga hasil penyederhanaan tata bahasa bebas konteks, menjadi :
S → ad

Contoh :
S → dAc
A → bc | ε

Aturan produksi A mengandung aturan produksi *nullable*, sehingga A → ε dapat dihapus. Akibat dari aturan produksi *nullable*, variabel A pada aturan produksi S dapat dihapus, sehingga aturan produksi S menjadi S → dc.

Pada aturan produksi A → bc | ε, apabila A → ε sudah dihapus, sementara aturan produksi *nullable* bukan merupakan satu-satunya aturan produksi pada A, sehingga pada aturan produksi S ditambahkan S → dc setelah aturan produksi S → dAc.

Sehingga hasil penyederhanaan tata bahasa bebas konteks, menjadi :

S → dAc | dc

A → bc

Contoh :

S → cA | dC

A → bd

A → ε

C → d

Aturan produksi A mengalami *nullable,* sehingga A → ε dapat dihapus. Akibat dari hal tersebut, variabel A pada S → cA dapat dihapus.

Dampak dari adanya aturan produksi A yang mengalami *nullable* dan aturan produksi A bukan satu-satunya aturan produksi pada S, sehingga aturan S dapat berubah menjadi S → cA | c | Dc

Sehingga hasil penyederhanaan tata bahasa bebas konteks, menjadi :

S → cA | c | dC

A → bd

C → d

Contoh :

S → CBAd

B → AC

A → d | ε

C → E | ε

E → f

Aturan produksi A dan C merupakan aturan produksi *nullable,* dan dapat dihapus. Dampak penghapusan tersebut berpengaruh pada aturan produksi S dan B. Sehingga aturan produksi S dan B menjadi :

S → CBAd | CBd | BAd | CAd | Cd | Bd | Ad | d

B → AC | A | C

Sehingga hasil penyederhanaan tata bahasa bebas konteks menjadi :

S → CBAd | CBd | BAd | CAd | Cd | Bd | Ad | d

B → AC | A | C

A → d

C → E

E → f

Contoh Soal 10.1

Lakukanlah penghilangan produksi *useless* dari aturan produksi dibawah ini :

S → Acb | Bde

A → bDc

B → Cd | c

E → F

Contoh Soal 10.2

Lakukan penghilangan produksi unit dari aturan produksi dibawah ini :

S → Ac

S → B

B → C

C → D

C → ef

D → gh

Contoh Soal 10.3

Lakukan penghilangan produksi ε dari aturan produksi dibawah ini :

S → dBA | dE

B → bd | ε

A → ε

E → f

Penyelesaian Contoh Soal 10.1

Penghilangan produksi *useless* :

S → Acb | Bde

A → bDc

B → Cd | c

E → F

- Sesuai dengan aturan penyederhanaan tata bahasa konteks untuk penghilangan produksi *useless,* kita dapat mulai menghapus produksi E → F, karena produksi ini tidak menuju ke terminal.
- Aturan produksi B → Cd juga dapat dihapus karena aturan produksi B tidak menuju ke teriminal.
- Akibat dari penghapusan produksi B, maka aturan produksi S → Bde juga dapat dihapus.
- Aturan produksi A → bDc dapat dihapus, karena tidak ada satupun aturan produksi lainnya yang membutuhkan D.
- Akibat dari penghapusan aturan produksi A, maka S → Acb juga dapat dihapus.
- Sehingga hasil penyederhanaan tata bahasa bebas konteks dengan penghilangan *useless* menjadi :

S → Bde | b

B → c

Penyelesaian Contoh Soal 10.2

Penghilangan produksi unit :

S → Ac

S → B

B → C

C → D

C → ef

D → gh

- Sesuai dengan aturan penyederhanaan tata bahasa konteks untuk penghilangan produksi unit, S → B, B → C, C → D dapat dihapus, sehingga aturan produksi S dapat langsung menjadi S → gh.
- Aturan produksi S → B, B → C dapat dihapus, sehingga aturan produksi S dapat langsung menjadi S → ef.
- Aturan produksi C → D dapat dihapus, sehingga aturan produksi C dapat langsung menjadi C → gh.
- Sehingga hasil penyederhanaan tata bahasa bebas konteks dengan penghilangan unit menjadi :

  S → gh

S → ef

C → gh

Penyelesaian Contoh Soal 10.3

Penghilangan produksi ε :

S → dBA | dE

B → bd | ε

A → ε

E → f

- Sesuai dengan aturan penyederhanaan tata bahasa konteks untuk penghilangan produksi ε, A → ε dapat dihapus
- Dampak dari penghapusan A → ε, variabel A pada S → dBA dapat diganti dan diubah menjadi S → dB.
- Aturan produksi B → ε dapat dihapus, sehingga berdampak pada aturan produksi S → dBA, sehingga aturan produksi S menjadi S → dA.
- Sehingga hasil penyederhanaan tata bahasa bebas konteks dengan penghilangan unit menjadi :

  S → dBA | dB | dA | dE

  B → bd

  E → f

Rangkuman

1. Penyederhanaan tata bahasa bebasa konteks terbagi menjadi 3, yaitu : penghilangan produksi *useless*, penghilangan produksi unit, dan penghilangan produksi ε.
2. Penghilangan produksi *useless* menghapus variabel yang tidak menghasilkan terminal, serta menghapus variabel yang tidak dapat dicapai dari variabel lain (yang tidak dibutuhkan).
3. Penghilangan produksi unit menghapus aturan produksi dimana ruas kiri dan ruas kanannya hanya terdiri dari satu simbol variabel.
4. Penghilangan produksi ε menghapus aturan produksi $\alpha \rightarrow \varepsilon$ (produksi kosong) yang berhubungan dengan variabel lain.

Tugas

1. Lakukan penyederhanaan tata bahasa bebas konteks ini dengan penghilangan *useless*, unit, dan ε :

   $S \rightarrow b \mid bA \mid C \mid D$

   $A \rightarrow bC \mid \varepsilon$

   $C \rightarrow Ab$

   $D \rightarrow bDE$

   $E \rightarrow ee$

2. Lakukan penyederhanaan tata bahasa bebas konteks ini dengan penghilangan *useless*, unit, dan ε :

S → BbC | bC | Ba | c

B → C | BD | B | D

D → e

C → f

3. Lakukan penyederhanaan tata bahasa bebas konteks ini dengan penghilangan *useless*, unit, dan ε :

   S → cD | ccD

   C → ε

   D → bC

   D → ε

BAB XI
BENTUK NORMAL CHOMSKY

Capaian Pembelajaran Mata Kuliah

- Mahasiswa mampu memahami teori tentang Bentuk Normal Chomsky

11.1. Bentuk Normal Chomsky

Bentuk normal Chomsky merupakan bentuk normal yang akan berguna untuk tata bahasa bebas konteks *(Context Free Grammar).* Bentuk normal Chomsky dapat dibentuk dari tata bahasa bebas konteks yang telah mengalami penghilangan *useless*, unit dan ε. Ciri khas bentuk normal Chomsky dapat dilihat dengan adanya terminal atau dua variabel pada aturan produksi ruas kanan.

11.2. Membangun Bentuk Normal Chomsky

Adapun beberapa tahapan dalam membangun bentuk normal Chomsky, adalah sebagai berikut:

1. Identifikasi aturan produksi yang belum dalam bentuk normal Chomsky.
2. Ganti aturan produksi yang ruas kanannya memiliki simbol terminal yang lebih dari 1.

3. Ganti aturan produksi yang ruas kanannya memiliki simbol variabel yang lebih dari 2.
4. Penggantian pada tahap 1 dan 2 akan tetap dilakukan sampai aturan produksi dalam bentuk normal Chomsky.
5. Dalam tahap penggantian aturan produksi seperti yang dilakukan pada tahap 1 dan 2, simbol terminal dan variabel akan diganti dengan beberapa simbol variabel baru.

Contoh :

Terdapat tata bahasa bebas konteks :

$S \rightarrow bC \mid aB$

$C \rightarrow bCC \mid aS \mid b$

$D \rightarrow cBB \mid bS \mid a$

- Aturan produksi yang sudah dalam bentuk normal Chomsky :

  $C \rightarrow b$

  $D \rightarrow a$

- Proses penggantian aturan produksi yang belum dalam bentuk normal Chomsky dengan simbol variabel baru seperti P_1, P_2, P_3 dan seterusnya.

  $S \rightarrow bC$ diganti menjadi $S \rightarrow P_1 C$

  $S \rightarrow aB$ diganti menjadi $S \rightarrow P_2 B$

  $C \rightarrow bCC$ diganti menjadi $C \rightarrow P_1 CC \Rightarrow C \rightarrow P_1 P_3$

$C \rightarrow aS$ diganti menjadi $C \rightarrow P_2\ S$

$D \rightarrow cBB$ diganti menjadi $D \rightarrow P_4\ BB \Rightarrow D \rightarrow P_4\ P_5$

$D \rightarrow bS$ diganti menjadi $D \rightarrow P_1\ S$

- Aturan produksi dan variabel yang baru terbentuk :

  $P_1 \rightarrow b$

  $P_2 \rightarrow a$

  $P_3 \rightarrow CC$

  $P_4 \rightarrow c$

  $P_5 \rightarrow BB$

- Hasil aturan produksi dalam bentuk normal Chomsky :

  $C \rightarrow b$

  $D \rightarrow a$

  $S \rightarrow P_1\ C$

  $S \rightarrow P_2\ B$

  $C \rightarrow P_1\ P_3$

  $C \rightarrow P_2\ S$

  $D \rightarrow P_4\ P_5$

  $D \rightarrow P_1\ S$

  $P_1 \rightarrow b$

  $P_2 \rightarrow a$

  $P_3 \rightarrow CC$

$P_4 \rightarrow c$

$P_5 \rightarrow BB$

Contoh Soal 11.1

$S \rightarrow aBD \mid cd \mid EF$

$B \rightarrow beG \mid aGE$

$D \rightarrow cc \mid FF$

$E \rightarrow BFD \mid cS$

$F \rightarrow g$

$G \rightarrow aF$

Penyelesaian Contoh Soal 11.1

- Aturan produksi yang sudah dalam bentuk normal Chomsky :

  $S \rightarrow EF$

  $D \rightarrow FF$

  $F \rightarrow g$

- Proses penggantian aturan produksi yang belum dalam bentuk normal Chomsky dengan simbol variabel baru seperti P_1, P_2, P_3 dan seterusnya.

  $S \rightarrow aBD$ diganti menjadi $S \rightarrow P_1 P_2$

  $S \rightarrow cd$ diganti menjadi $S \rightarrow P_3 P_4$

  $B \rightarrow beG$ diganti menjadi $B \rightarrow P_5 P_6 \Rightarrow B \rightarrow P_5 P_7 G$

$B \rightarrow aGE$ diganti menjadi $B \rightarrow P_1 P_8$

$D \rightarrow cc$ diganti menjadi $D \rightarrow P_9 P_9$

$E \rightarrow BFD$ diganti menjadi $E \rightarrow B P_{10}$

$E \rightarrow cS$ diganti menjadi $E \rightarrow P_3 S$

$G \rightarrow aF$ diganti menjadi $G \rightarrow P_1 F$

- Aturan produksi dan variabel yang baru terbentuk :

$P_1 \rightarrow a$

$P_2 \rightarrow BD$

$P_3 \rightarrow c$

$P_4 \rightarrow d$

$P_5 \rightarrow b$

$P_6 \rightarrow P_7 G$

$P_7 \rightarrow e$

$P_8 \rightarrow GE$

$P_9 \rightarrow c$

$P_{10} \rightarrow FD$

- Hasil aturan produksi dalam bentuk normal Chomsky :

$S \rightarrow EF$

$D \rightarrow FF$

$F \rightarrow g$

$S \rightarrow P_1 P_2$

$S \rightarrow P_3 P_4$

$B \rightarrow P_5 P_7 G$

$B \rightarrow P_1 P_8$

$D \rightarrow P_9 P_9$

$E \rightarrow B P_{10}$

$E \rightarrow P_3 S$

$G \rightarrow P_1 F$

$P_1 \rightarrow a$

$P_2 \rightarrow BD$

$P_3 \rightarrow c$

$P_4 \rightarrow d$

$P_5 \rightarrow b$

$P_6 \rightarrow P_7 G$

$P_7 \rightarrow e$

P_8 GE

$P_9 \rightarrow c$

$P_{10} \rightarrow FD$

Rangkuman

1. Bentuk normal Chomsky dapat dibentuk dari tata bahasa bebas konteks yang telah mengalami penghilangan *useless*, unit dan ε.

2. Ciri-ciri aturan produksi yang sudah dalam bentuk normal Chomsky adalah :
   a. Aturan produksi yang ruas kanannya memiliki simbol terminal yang lebih dari 1.
   b. Aturan produksi yang ruas kanannya memiliki simbol variabel yang lebih dari 2.
3. Aturan produksi yang belum dalam bentuk normal Chomsky akan diubah menjadi beberapa simbol variabel baru.
4. Aturan produksi akhir yang sudah melalui proses perubahan ke dalam bentuk normal chomsky merupakan gabungan dari :
   a. Aturan produksi yang sebelumnya teridentifikasi sebagai bentuk normal Chomsky
   b. Aturan produksi yang sudah diubah ke dalam bentuk normal Chomsky
   c. Aturan produksi yang terbentuk dari simbol variabel baru.

Tugas

1. Ubahlah tata bahasa bebas konteks berikut ini ke dalam bentuk normal chomsky :

   $S \rightarrow bCd \mid ab$

   $A \rightarrow c$

2. Ubahlah tata bahasa bebas konteks berikut ini ke dalam bentuk normal chomsky :

S → bSbC | C | EE

C → bdC | d

D → e

3. Ubahlah tata bahasa bebas konteks berikut ini ke dalam bentuk normal chomsky :

   S → bS | aEF

   B → bCC | d

   F → ef

   G → h | KL

BAB XII
PENGHILANGAN REKURSIF KIRI

Capaian Pembelajaran Mata Kuliah

- Mahasiswa dapat memahami teori tentang penghilangan rekursif kiri

12.1. Aturan Produksi Rekursif

Aturan produksi rekursif merupakan aturan produksi yang mengalami perulangan baik dari ruas kanan ataupun ruas kiri. Aturan produksi rekursif juga aturan produksi yang pada ruas kanannya terdapat simbol variabel dari ruas kiri aturan produksinya, secara formal dapat dituliskan sebagai berikut :

$$A \rightarrow \beta A$$

dimana β merupakan kumpulan simbol variabel dan terminal.

12.1.1. Aturan Produksi Rekursif Kanan

Sebuah aturan produksi yang apabila diturunkan dengan pohon penurunan, memiliki arah penurunan lebih banyak ke sisi kanan, disebut dengan aturan produksi rekursif kanan.

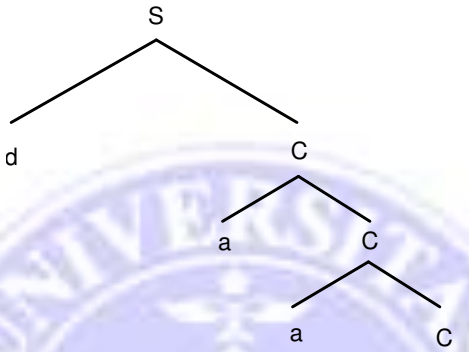
Contoh :

Terdapat aturan produksi :

$$S \rightarrow dC$$

$$C \rightarrow aC \mid \varepsilon$$

Pohon penurunan dari aturan produksi :



Dari pohon penurunan di atas, dapat dilihat bahwa cabang pohon penurunan lebih banyak ke arah kanan dan berulang, sehingga aturan produksi tersebut disebut dengan aturan produksi rekursif kanan.

12.1.2. Aturan Produksi Rekursif Kiri

Sebuah aturan produksi yang apbila diturunkan dengan pohon penurunan, memiliki arah penurunan lebih banyak ke sisi kiri, disebut dengan aturan produksi rekursif kiri.

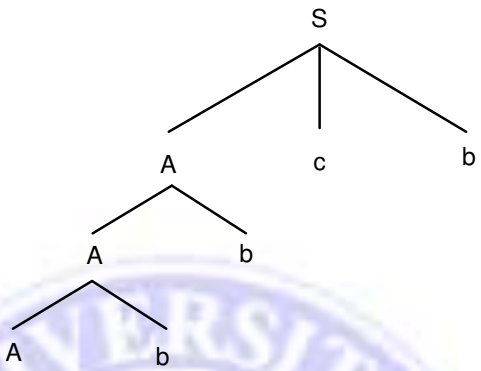
Contoh :

Terdapat aturan produksi :

$$S \rightarrow Acb$$

$$A \rightarrow Ab \mid \varepsilon$$

Pohon penurunan dari aturan produksi :



Dari pohon penurunan di atas, dapat dilihat bahwa cabang pohon penurunan lebih banyak ke arah kiri dan berulang, sehingga aturan produksi tersebut disebut dengan aturan produksi rekursif kiri.

12.2. Penghilangan Rekursif Kiri

Adapun beberapa tahapan dalam penghilangan rekursif kiri adalah sebagai berikut :

1. Telusuri mana aturan produksi yang rekursif kiri dan yang bukan rekursif kiri. Penelusuran boleh dibantu dengan membuat pohon penurunan.
2. Untuk aturan produksi yang memiliki simbol ruas kiri yang sama, lakukan penggantian dengan simbol variabel Z_1, Z_2 seperti pada aturan berikut ini :
   1) $A \rightarrow \beta_1 Z \mid \beta_2 Z \mid \ldots \beta_m Z$
   2) $Z \rightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \ldots. \alpha_n$
   3) $Z \rightarrow \alpha_1 Z \mid \alpha_2 Z \mid \alpha_3 Z \mid \ldots. \alpha_n Z$

3. Hasil akhir pada penghilangan aturan produksi rekursi kiri merupakan gabungan antara aturan produksi pengganti dan aturan produksi yang tidak rekursif kiri.

Contoh Soal 12.1

Terdapat tata bahasa bebas konteks seperti berikut ini, lakukan penghilangan aturan produksi yang mengalami rekursif kiri.

$$S \rightarrow Sac \mid aSb \mid ee \mid gg \mid Sde$$

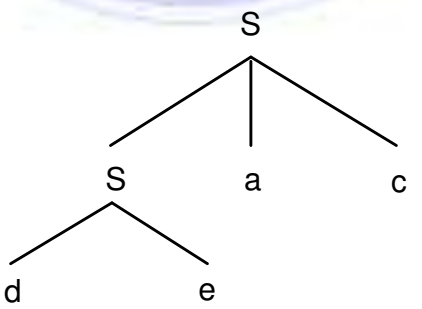
Penyelesaian Contoh Soal 12.1

Adapun tahapan dalam penghilangan aturan produksi rekursif kiri, adalah sebagai berikut :

a. Dari penelusuran aturan produksi, aturan produksi yang merupakan rekursif kiri adalah :

$$S \rightarrow Sac \mid Sde$$

Apabila aturan produksi tersebut dibuktikan dengan pohon penurunan, maka diperoleh pohon penurunan rekursif kiri sebagai berikut :



Dari sini kita tentukan simbol ruas kiri :

S : α_1 = ac, α_1 = de

b. Dari penelusuran aturan produksi, aturan produksi yang tidak rekursif kiri adalah :

$$S \rightarrow aSb \mid ee \mid gg$$

Dari sini kita tentukan simbol ruas kiri :

S : β_1 = aSb, β_2 = ee, β_3 = gg

c. Proses penggantian aturan produksi yang rekursif kiri (tahap a) dilakukan terhadap simbol ruas kiri S :

- $S \rightarrow Sac \mid Sde$, digantikan oleh :
  - i. $S \rightarrow aSbZ_1 \mid eeZ_1 \mid ggZ_1$
    (penambahan Z_1 pada setiap akhir aturan produksi yang tidak rekursif kiri)
  - ii. $Z_1 \rightarrow ac \mid de$
    (Z_1 merupakan kumpulan terminal yang ada pada aturan produksi rekursif kiri $S \rightarrow Sac \mid Sde$)
  - iii. $Z_1 \rightarrow acZ_1 \mid deZ_1$
    (penambahan Z_1 pada setiap akhir aturan produksi $Z_1 \rightarrow ac \mid de$)

d. Hasil akhir aturan produksi setelah mengalami penghilangan aturan produksi rekursif kiri merupakan gabungan dari aturan produksi tahap 2 dan tahap 3 :

$$S \rightarrow aSb \mid ee \mid gg$$

$$S \rightarrow aSbZ_1 \mid eeZ_1 \mid ggZ_1$$

$$Z_1 \rightarrow ac \mid de$$

$$Z_1 \rightarrow acZ_1 \mid deZ_1$$

Contoh Soal 12.2

Terdapat tata bahasa bebas konteks seperti berikut ini, lakukan penghilangan aturan produksi yang mengalami rekursif kiri.

$$S \rightarrow Sc \mid cBb \mid f \mid \varepsilon$$

$$B \rightarrow Ba \mid de$$

Penyelesaian Contoh Soal 12.2

Adapun tahapan dalam penghilangan aturan produksi rekursif kiri, adalah sebagai berikut :

a. Dari penelusuran aturan produksi, aturan produksi yang merupakan rekursif kiri adalah :

$$S \rightarrow Sc$$

$$B \rightarrow Ba$$

Dari sini kita tentukan simbol ruas kiri :

- Untuk simbol ruas kiri S : $\alpha_1 = c$
- Untuk simbol ruas kiri B : $\alpha_1 = a$

b. Dari penelusuran aturan produksi, aturan produksi yang tidak rekursif kiri adalah :

$$S \rightarrow cBb \mid f \mid \varepsilon$$

$$B \rightarrow de$$

Dari sini kita tentukan simbol ruas kiri :

- Untuk simbol ruas kiri S : $\beta_1 = cBb$, $\beta_2 = f$, $\beta_3 = \varepsilon$
- Untuk simbol ruas kiri A : $\beta_1 = de$

c. Proses penggantian aturan produksi yang rekursif kiri dilakukan terhadap simbol ruas kiri S dan B :

- Untuk yang memiliki simbol ruas kiri S :
- $S \rightarrow Sc$, digantikan oleh :

i. $S \rightarrow cBbZ_1 \mid f\,Z_1 \mid Z_1$

(penambahan Z_1 pada setiap akhir aturan produksi yang tidak rekursif kiri)

ii. $Z_1 \rightarrow c$

(Z_1 merupakan kumpulan terminal yang ada pada aturan produksi rekursif kiri $S \rightarrow Sc$)

iii. $Z_1 \rightarrow cZ_1$

(penambahan Z_1 pada setiap akhir aturan produksi $Z_1 \rightarrow c$)

- Untuk yang memiliki simbol ruas kiri B :

- $B \rightarrow Ba$, digantikan oleh :
  i. $B \rightarrow deZ_2$

     (penambahan Z_2 pada setiap akhir aturan produksi yang tidak rekursif kiri)
  ii. $Z_2 \rightarrow a$

     (Z_2 merupakan kumpulan terminal yang ada pada aturan produksi rekursif kiri $B \rightarrow Ba$)
  iii. $Z_2 \rightarrow aZ_2$

     (penambahan Z_2 pada setiap akhir aturan produksi $Z_2 \rightarrow a$)

d. Hasil akhir aturan produksi setelah mengalami penghilangan aturan produksi rekursif kiri merupakan gabungan dari aturan produksi tahap b dan tahap c :

$$S \rightarrow cBb \mid f \mid \varepsilon$$

$$S \rightarrow cBbZ_1 \mid f\,Z_1 \mid Z_1$$

$$B \rightarrow de$$

$$B \rightarrow deZ_2$$

$$Z_1 \rightarrow c$$

$$Z_1 \rightarrow cZ_1$$

$$Z_2 \rightarrow a$$

$$Z_2 \rightarrow aZ_2$$

Rangkuman

1. Aturan produksi rekursif kiri merupakan aturan produksi yang apabila diturunkan dengan pohon penurunan, memiliki arah penurunan lebih banyak ke sisi kiri.
2. Aturan produksi rekursif kiri membuat sebuah aturan produksi selalu berulang *(loop)*, sehingga perlu diganti dengan variabel-variabel baru yang sesuai.
3. Pembuktian aturan produksi rekursif kiri dapat dilakukan dengan membuat pohon penurunan terlebih dahulu.

Tugas

1. Hilangkan rekursif kiri pada tata bahasa bebas konteks berikut :

$$S \rightarrow SCa \mid Ac$$
$$A \rightarrow Sb \mid Aad \mid b$$
$$C \rightarrow Sc \mid Cce \mid d$$

2. Hilangkan rekursif kiri pada tata bahasa bebas konteks berikut :

$$S \rightarrow SSB \mid SSB \mid abd$$
$$B \rightarrow abe \mid BSc \mid Bce$$
$$C \rightarrow de$$

BAB XIII
PUSH DOWN AUTOMATA

Capaian Pembelajaran Mata Kuliah

- Mahasiswa dapat memahami teori tentang *Push Down Automata*

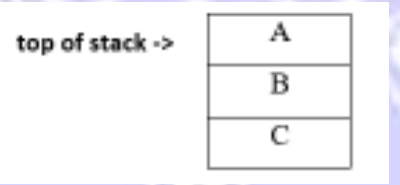
13.1. *Push Down Automata*

Push Down Automata (PDA) merupakan mesin *automata* yang mampu mengelola tata bahasa bebas konteks. *Push Down Automata* juga dapat diasumsikan sebagai tempat penyimpanan yang tidak terbatas dalam bentuk *stack*/tumpukan, sehingga *Push Down Automata* juga sering disebut NFA dengan *stack*.

Push Down Automata digunakan untuk menentukan apakah sebuah deretan *string* dapat diterima atau tidak dapat diterima. Namun yang membuat PDA berbeda dari mesin *automata* lainnya adalah penggunaan *stack* dalam penentuan sebuah *string* dapat diterima atau tidak diterima. Sebuah deretan *string* dapat diterima oleh mesin PDA apabila seluruh *string* sudah terbaca dengan baik, fungsi transisi dapat mencapai *final state* dan semua elemen *stack* sudah kosong (hanya tinggal Z_0 sebagai *top of stack*).

Stack/ tumpukan memiliki banyak ruang untuk penyimpanan data, puncak paling atas dari *stack* sering disebut *top of stack*. Cara mengambil dan memasukkan data pada *stack*, mengikuti aturan LIFO *(Last In First Out).*

Ilustrasinya puncak *stack* dapat dilihat pada Gambar 1 dibawah ini:



Gambar 13.1 Puncak Stack *(Top of stack)*

Sesuai dengan aturan LIFO, apabila ada elemen data yang akan diambil, maka elemen yang terambil adalah yang terletak di *stack* paling atas *(top of stack).* Begitu juga dengan memasukkan elemen data, elemen data akan tetap diletakkan pada posisi paling atas *stack.*

Secara formal, mesin *Push Down Automata* (PDA) dapat dinyatakan dengan 7 tupel, M = (Q, Σ, Γ, Δ, S, F, Z). Secara rinci dapat dilihat pada keterangan dibawah ini :

Q = Kumpulan *State*

Σ = Simbol *Input*

Γ = Simbol *Stack*

δ = Fungsi Transisi

S = *State* Awal

F = *State* Final

Z = Simbol Awal Tumpukan / *Top of Stack*

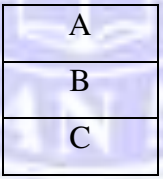
13.2. Akses Elemen pada *Stack (Pop* dan *Push)*

Terdapat dua istilah yang harus diingat dalam mengakses elemen data pada *stack*, yaitu :

1) *Pop* : perintah untuk mengambil elemen dari *stack*
2) *Push* : perintah untuk memasukkan elemen ke dalam *stack*

Contoh :

Tambahkan sebuah elemen D pada *stack* dibawah ini :



Untuk menambahkan sebuah elemen pada *stack* di atas, kita dapat menggunakan perintah *push*. Maka data yang ada pada *stack* sekarang menjadi :

D
A
B
C

Contoh :

Ambil elemen X pada *stack* dibawah ini :

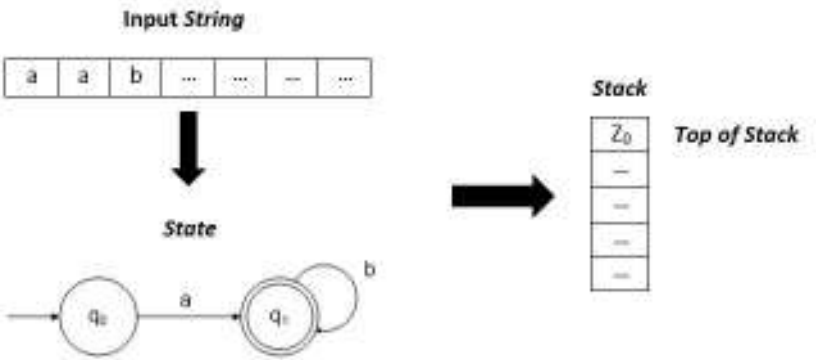
X
Y
A
B
C

Untuk mengambil sebuah elemen pada *stack* diatas, kita dapat menggunakan perintah *pop*. Maka data yang ada pada *stack* sekarang menjadi :

Y
A
B
C

13.3. Hubungan *Push Down Automata* pada *Input*, *State* dan *Stack*

Dalam *Push Down Automata*, hubungan antara *input*, *state* dan *stack* sangatlah erat. *Input* akan diterima dan di proses oleh mesin *automata*, kemudian akan diteruskan dan disimpan pada *stack.* Secara visualisasi dapat dilihat pada gambar di bawah ini :



Gambar 13.2 Hubungan input, *State* dan *Stack*

Pada gambar di atas, *input* string dapat berupa huruf alfabet antara a-z dan A-Z. *State* dalam PDA dapat terdiri dari q_0, q_1, q_2 seterusnya sampai *final state*. Dan terakhir adalah *stack,* stack diawali dengan Z_0 sebagai *top of stack* dan akan bertambah sesuai dengan proses PDA mengenali *input*.

13.4. Fungsi Transisi pada *Push Down Automata*

Fungsi transisi (δ) pada mesin *Push Down Automata* sedikit berbeda dari mesin-mesin *automata* lainnya. *Push Down Automata* dikatakan *Non Deterministic*, karena beberapa transisi pada *pop* dan *input* yang sama tetap diperbolehkan, dan tidak harus semua transisi mengalami *pop* dan *push*. Untuk memahami bagaimana sebuah deretan *string* dapat diterima oleh PDA, kita terlebih dahulu harus memahami bagaimana membentuk fungsi transisi dan bagaimana kaitannya *state* diagram.

Fungsi transisi pada *Push Down Automata* memiliki format :

$$\delta\ (q_0, a, b) = \{(q_1, w)\}$$

Keterangan :

δ = Fungsi Transisi

q_0 = *State* Awal

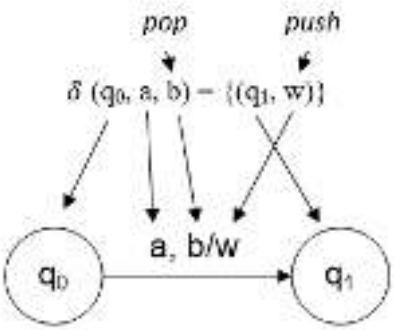
a = *Input* berupa string

b = *Pop* (dapat berupa alfabet maupun λ)

q_1 = *State* Tujuan

w = *Push* (dapat berupa alfabet maupun λ)

Ilustrasi hubungan antara fungsi transisi dan *state*, dapat dilihat pada gambar dibawah ini :

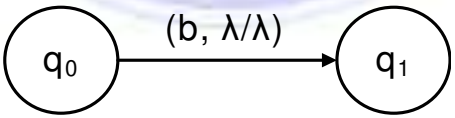


Gambar 13.3 Hubungan Fungsi Transisi dan *State*

Pada posisi *pop* dan *push,* adakalanya digunakan simbol λ yang dapat diartikan seperti *empty* (ε). Sehingga apabila mendapat perintah *pop/ push* λ , maka tidak ada perubahan elemen pada *stack.*

Contoh :

Terdapat fungsi transisi δ (q_0, b, λ) = {(q_1, λ)}. *State* diagram dari fungsi transisi tersebut adalah :



13.5. Cara *Push Down Automata* Memproses *String*

Selain perlu memahami fungsi transisi, selanjutnya kita perlu memahami bagaimana kerja mesin *Push Down Automata* dalam memperoses sebuah deretan *string*. Setiap *string* yang dimasukkan akan diproses dan akhirnya ditentukan apakah *string* tersebut dapat diterima atau tidak dapat diterima.

Mesin *Push Down Automata* memiliki cara yang berbeda dalam memproses *string*. Apabila terdapat sebuah deretan *input string*, PDA akan mulai membaca *input* dari arah kiri ke arah kanan, dan langsung memprosesnya dengan mesin *automata*, pilihan yang terjadi pada mesin *automata* terdiri dari dua hal, yaitu:

1) Apabila terdapat perintah *pop,* maka elemen pada *stack* yang posisinya paling atas akan dikeluarkan,
2) Apabila terdapat perintah *push*, maka input *string* akan dimasukkan ke posisi paling atas pada *stack*.

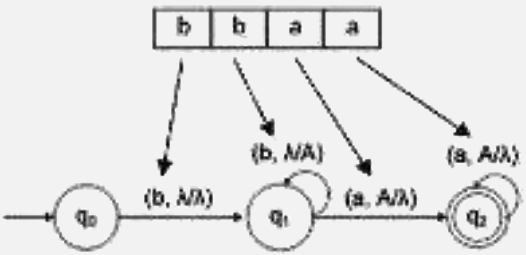
Berikut ini tahapan *Push Down Automata* memproses *input* / *string*:

1. Deretan *string* yang dianggap sebagai input, akan dimasukkan ke dalam deretan *array* yang memiliki ruang-ruang yang tak berhingga,

2. Fungsi transisi yang dimiliki oleh *Push Down Automata* akan membaca perintah untuk memproses sebuah *input*, dan hal ini akan berjalan dari satu *state* menuju *state* lain,
3. Setiap input yang diproses akan dilanjutkan ke *stack* melalui perintah *pop/push* (*stack* pertama kali hanya terdiri dari Z_0),
4. Proses pembacaan setiap input dan fungsi transisi akan tetap berlangsung dari *state* q_0 sampai *state final*, hal ini terus akan belangsung sampai seluruh *input* selesai dibaca dan sampai elemen pada *stack* menjadi kosong (pada *stack* hanya ada Z_0),
5. Sederatan *input* dapat diterima oleh mesin *Push Down Automata* apabila *input* dapat dibaca secara menyeluruh oleh fungsi transisi pada mesin *Push Down Automata* dan posisi *stack* kembali menjadi kosong,
6. Sederatan *input* yang tidak dapat diterima oleh mesin *Push Down Automata* apabila *input* tidak dapat dibaca secara menyeluruh (hanya dapat dibaca sampai deretan *input* urutan ke - n) oleh fungsi transisi pada mesin *Push Down Automata* dan posisi *stack* tidak kosong.

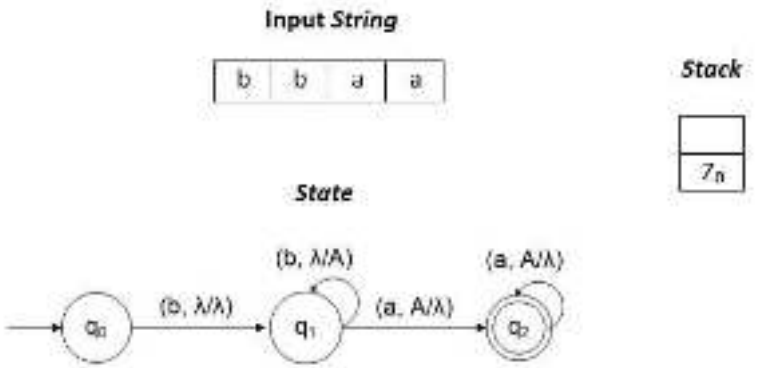
Catatan :

- Untuk awal mula pembacaan fungsi transisi pada *state* q_0 dan pada akhir pembacaan fungsi transisi menuju *final state*, perintah yang dilakukan adalah *push* λ.
- Beberapa jenis fungsi transisi δ yang harus dipahami :
  1. δ = (a, λ/ λ) => baik perintah *pop/push* tidak membawa perubahan yang berarti pada *stack*, artinya tidak ada variabel yang akan di *pop/push*.
  2. δ = (a, λ/ A) => lakukan *push* A pada *stack*
  3. δ = (a, A/ λ) => lakukan *pop* A pada *stack*
- Pada fungsi transisi δ = (b, λ/ λ), b merupakan *input* yang menjadi acuan bagi fungsi transisi saat membaca setiap deretan *string*. Dimana artinya *input* a dalam deretan *string*, hanya akan dapat dibaca apabila *input* sama dengan variabel a pada fungsi transisi. Ilustrasinya adalah sebagai berikut :



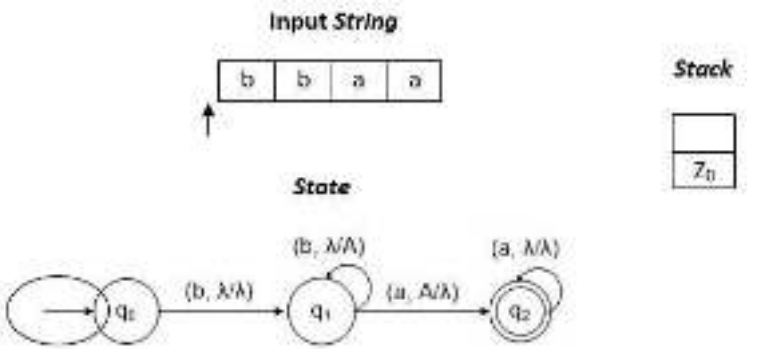
Contoh :

Terdapat *string*, fungsi transisi, mesin PDA dan keadaan *stack* sebagai berikut, apakah string bbaa dapat diterima oleh PDA atau tidak.

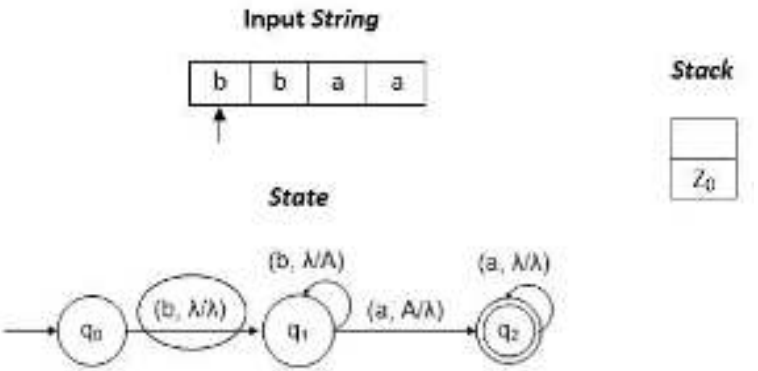


Langkah-langkah PDA dalam memproses *string* adalah sebagai berikut :

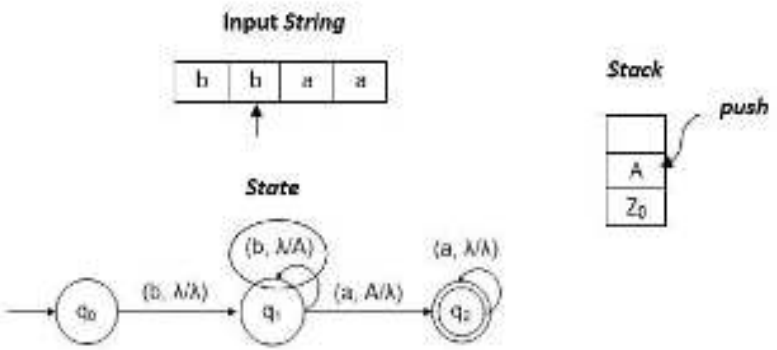
1. Pembacaan awal akan dilakukan untuk *state* awal q_0, dan belum ada data yang di *pop/push* ke *stack*.



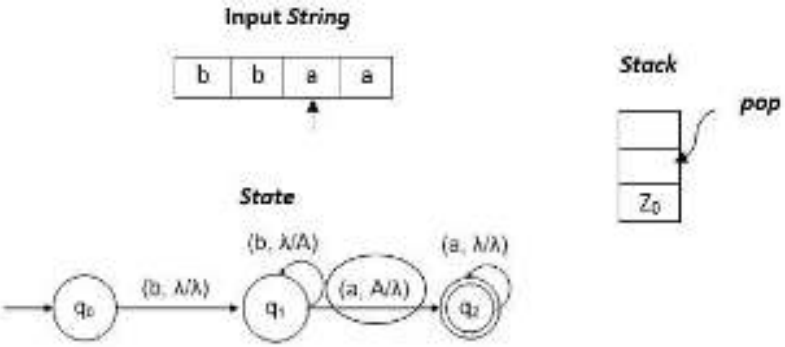
2. Pada tahap ini, dilakukan pembacaan *input string* b dan fungsi transisi $\delta = (b, \lambda/\lambda)$ menuju q_1, dimana dilakukan *push* λ pada *stack*, namun tidak terjadi penambahan data pada *stack.*



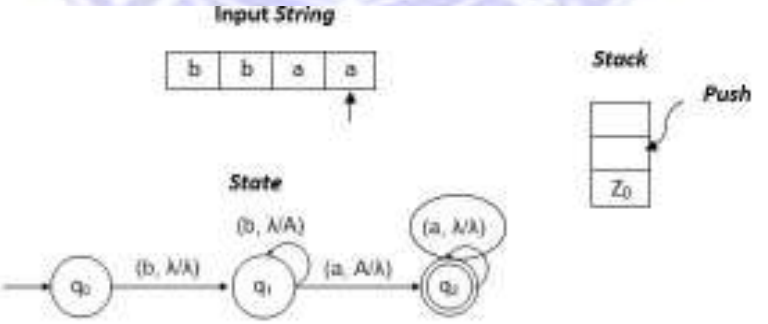
3. Selanjutnya, akan dilakukan pembacaan *input string* b dan fungsi transisi $\delta = (b, \lambda/A)$ pada q_1, dimana dilakukan *push* A pada *stack*, sehingga terjadi penambahan data A pada *stack.*



4. Selanjutnya, akan dilakukan pembacaan *input string* a dan fungsi transisi δ = (a, A/λ) menuju q_2, dimana dilakukan *pop* A pada *stack*, sehingga data A dikeluarkan dari *stack.*



5. Selanjutnya, akan dilakukan pembacaan *input string* a dan fungsi transisi δ = (a, λ/λ) pada q_2, dimana dilakukan *push* λ pada *stack*, sehingga tidak terjadi perubahan apapun pada *stack.*



6. Pada tahap 5, dapat dilihat bahwa seluruh *input string* sudah terbaca selurunya, fungsi transisi juga sudah mencapai *final state*, serta data pada *stack* sudah kosong dan hanya tertinggal Z_0. Dari ketiga kondisi ini dapat disimpulkan bahwa *input string* bbaa dapat diterima dengan baik oleh mesin PDA.

13.6. Membentuk *Push Down Automata* yang Dapat Menerima *String*

Pada sub bab sebelumnya sudah dipaparkan bagaimana cara kerja *Push Down Automata* dalam membaca deretan *string*. Pada sub bab ini akan dijelaskan bagaimana caranya membentuk *Push Down Automata* yang dapat menerima sederetan *string* tertentu. Ada beberapa hal yang perlu dipahami untuk membuat *Push Down Automata*, yaitu kemahiran membentuk fungsi transisi, kemampuan membedakan perintah *pop* dan *push*, serta kejelian memasukkan dan mengeluarkan elemen data dari *stack*.

Adapun langkah-langkah membentuk *Push Down Automata* adalah sebagai berikut :

1. Masukkan sederatan *string* kedalam *array* kusus *string* (Σ),
2. Tentukan berapa jumlah *state* yang dibutuhkan (Q) beserta dengan *state* awal (S) dan *final state* (F),
3. Tentukan simbol yang akan di *pop/push* ke dalam *stack* (Γ)
4. Pastikan *stack* dalam keadaan kosong (hanya ada Z_0).

5. Bentuk fungsi transisi dengan formula δ (q_0, b, λ) = {(q_1, λ)} melalui simulasi diagram mesin PDA.
6. Setelah selesai tahap 5, maka diperolehlah mesin PDA yang dapat menerima sederetan *string*.

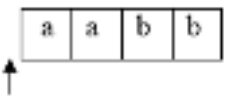
Contoh :

Mesin *Push Down Auomata* yang dapat menerima *input string* aabb. Langkah-langkah membentuk *Push Down Automata* yang sesuai adalah :

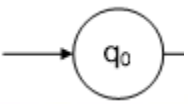
1. *Input String* :

a	a	b	b

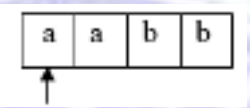
2. Q = {q_0, q_1, q_2}
   S = {q_0}
   F = {q_2}
3. Simbol yang akan di *pop/push* adalah A
4. Z = Z_0
5. Sketsa *diagram* PDA beserta pembuatan fungsi transisi :
   a. Berdasarkan *string* pada langkah (1) yaitu :



Pada awal *state* q_0, kita belum membentuk fungsi transisi, sehingga belum ada elemen yang akan di *pop/push* ke *stack.*



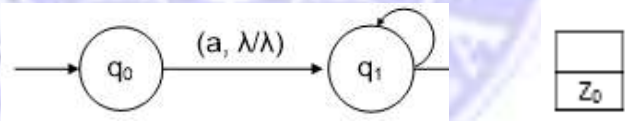
b. Tahap selanjutnya, akan dibaca *input* a :



maka kita dapat asumsikan fungsi transisi berawal dari *state* q_0 menuju *state* q_1 dan melakukan push λ ke *stack.* Sehingga fungsi transisi dapat ditulis :

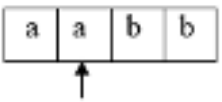
$$\delta\ (q_0, a, \lambda\ ) = \{(q_1, \lambda)\}$$

Sehingga *state* dan *stack* dapat diilustrasikan sebagai berikut :



- Kondisi *stack* masih kosong karena fungsi transisi melakukan *push* λ.

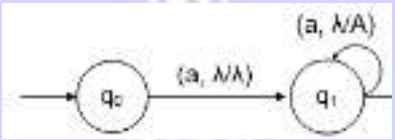
c. Tahap selanjutnya, akan dibaca *input* a :



maka kita dapat asumsikan fungsi transisi berada pada *state* q_1 dan kita akan mencoba melakukan *looping* di *state* q_1, serta melakukan *push* A ke *stack*. Sehingga fungsi transisi dapat ditulis :

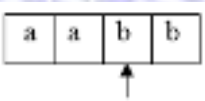
$$\delta\ (q_1, a, \lambda\ ) = \{(q_1, A)\}$$

Sehingga *state* dan *stack* dapat diilustrasikan sebagai berikut :



- Kondisi *stack* berisi simbol A, karena fungsi transisi melakukan *push* A.

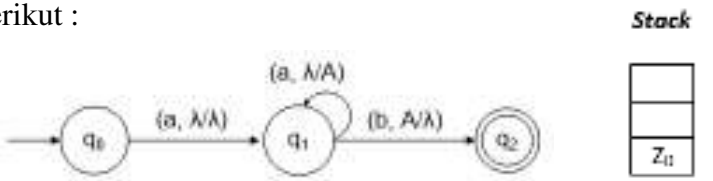
d. Tahap selanjutnya, akan dibaca *input* b :



maka kita dapat asumsikan fungsi transisi yang sekarang berada pada *state* q_1 akan berlanjut ke *state* q_2 yang merupakan *final state*, serta melakukan *pop* A dari *stack*. Sehingga fungsi transisi dapat ditulis :

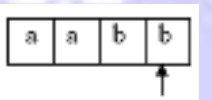
$$\delta\ (q_1, b, A\ ) = \{(q_2, \lambda)\}$$

Sehingga *state* dan *stack* dapat diilustrasikan sebagai berikut :



- Kondisi *stack* kosong karena fungsi transisi melakukan *pop* A.

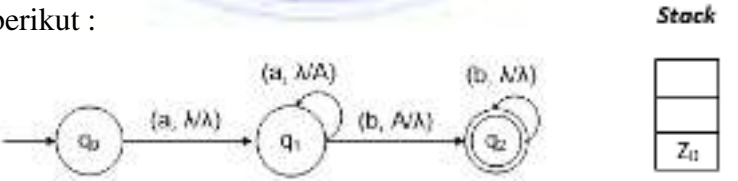
e. Tahap selanjutnya, akan dibaca *input* b :



maka kita dapat asumsikan fungsi transisi berada pada *state* q_2 dan kita akan mencoba melakukan *looping* di *state* q_2 yang merupakan *final state*, serta melakukan *push* λ ke *stack.* Sehingga fungsi transisi dapat ditulis sebagai berikut:

$$\delta\ (q_2, b, \lambda\ ) = \{(q_2, \lambda)\}$$

Sehingga *state* dan *stack* dapat diilustrasikan sebagai berikut :



- Kondisi *stack* kosong karena fungsi transisi melakukan *push.* λ.

f. Karena seluruh input *string* sudah terbaca dengan baik, dan *stack* juga sudah kosong, maka berikut ini fungsi transisi beserta *Push Down Automata* yang dapat menerima *input* aabb :

- Fungsi Transisi :

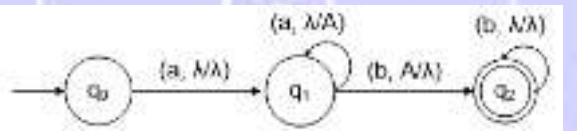
  δ (q_0, a, λ) = {(q_1, λ)}

  δ (q_1, a, λ) = {(q_1, A)}

  δ (q_1, b, A) = {(q_2, λ)}

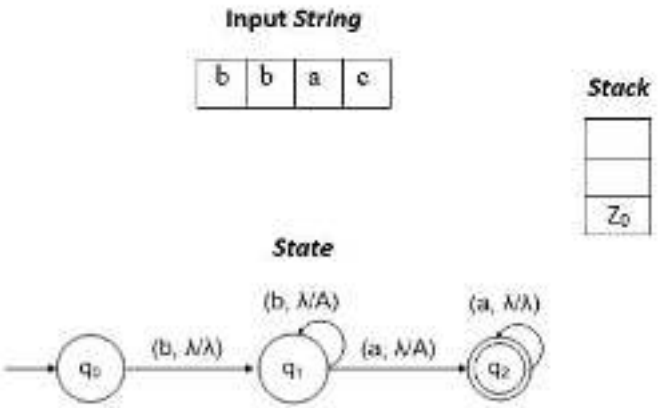
  δ (q_2, b, λ) = {(q_2, λ)}

- *Push Down Automata* :



Contoh Soal 13.1

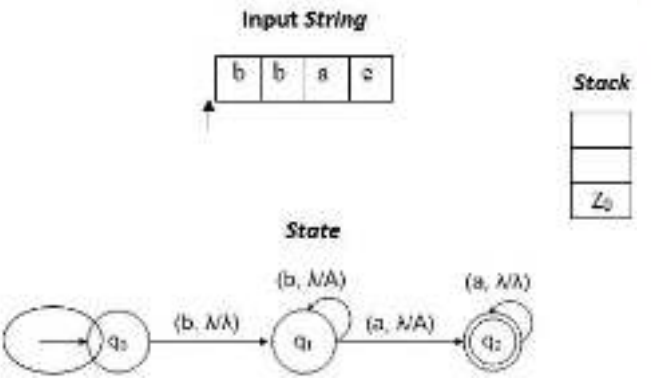
Terdapat *string*, fungsi transisi, mesin *Push Down Automata* dan keadaan *stack* sebagai berikut. Apakah *string* bbac dapat diterima oleh *Push Down Automata* ?



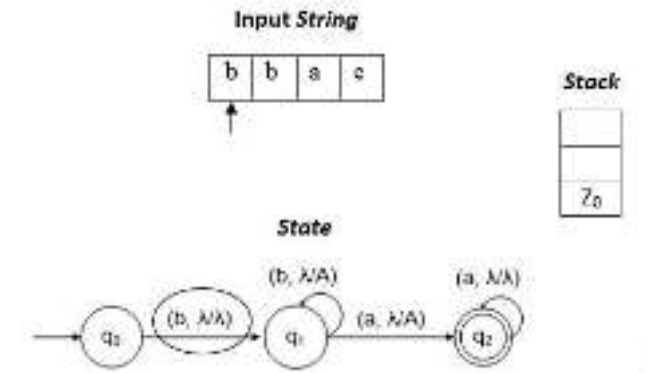
Penyelesaian Contoh Soal 13.1

Langkah-langkah PDA dalam memproses *string* adalah sebagai berikut :

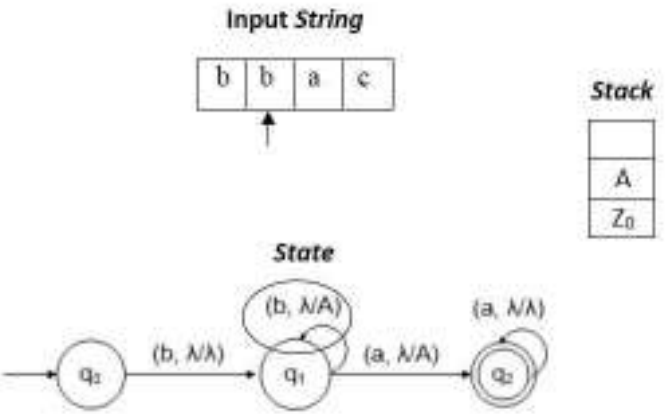
a. Pembacaan awal akan dilakukan untuk *state* awal q_0, dan belum ada data yang di *pop/push* ke *stack*.



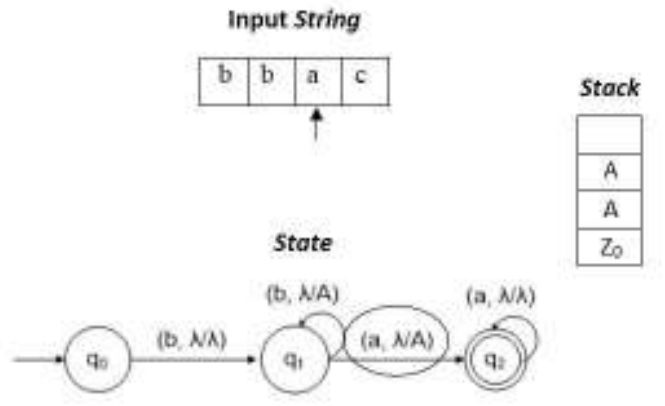
b. Pada tahap ini, dilakukan pembacaan *input string* b dan fungsi transisi $\delta = (b, \lambda/\lambda)$ menuju q_1, dimana dilakukan *push* λ pada *stack*, namun tidak terjadi penambahan data pada *stack.*



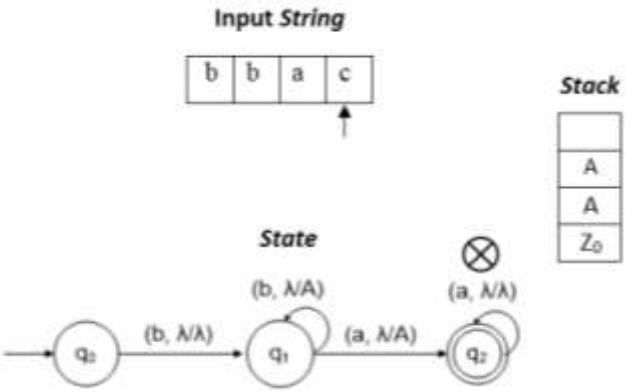
c. Selanjutnya, akan dilakukan pembacaan *input string* b dan fungsi transisi $\delta = (b, \lambda/A)$ pada q_1, dimana dilakukan *push* A pada *stack*, sehingga terjadi penambahan data A pada *stack.*



d. Selanjutnya, akan dilakukan pembacaan *input string* a dan fungsi transisi δ = (a, λ/A) menuju q_2, dimana dilakukan *push* A pada *stack*, sehingga data A ditambahkan pada *stack.*



e. Pada tahap ini *input string* dideteksi tidak dapat dibaca, karena saat pembacaan *input string* c, ditemukan kondisi yang tidak sesuai antara *input string* yang terbaca dengan fungsi transisi δ = (a, λ/λ) pada q_2. Kemudian saat dilakukan *push* λ pada *stack*, tidak terjadi perubahan apapun pada *stack.*



f. Pada tahap e, terdapat dua kondisi :
   1) tidak semua *input string* dapat terbaca dengan baik, terdapat *input string* c yang tidak dapat dibaca oleh mesin PDA.
   2) Terdapat data dua variabel A pada *stack*, sehingga *stack* tidak sepenuhnya kosong.

g. Dari dua kondisi tersebut, dapat diambil kesimpulan bahwa *input string* bbac tidak dapat diterima oleh mesin PDA.

Contoh Soal 13.2

Buatlah *Push Down Automata* dari konfigurasi berikut yang dapat menerima *input* accba :

$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{a, b\}$

$\Gamma = A$

$S = q_0$

$F = q_3$

$Z = Z_0$

Fungsi Transisi (δ) :

$\delta\ (q_0, a, \lambda\ ) = \{(q_1, \lambda)\}$

$\delta\ (q_1, c, \lambda\ ) = \{(q_1, A)\}$

$\delta\ (q_1, c, A\ ) = \{(q_2, \lambda)\}$

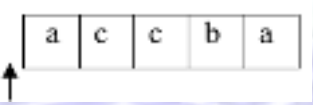
δ (q_2, b, λ) = {(q_2, λ)}

δ (q_2, a, λ) = {(q_3, λ)}

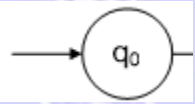
Penyelesaian Contoh Soal 13.2

Langkah-langkah membuat PDA :

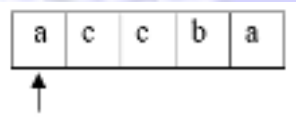
a. Urutkan *string* dan masukkan ke list *input* :



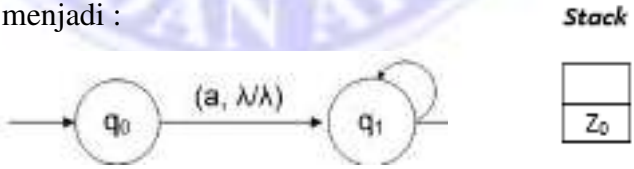
Pada awal *state* q_0, belum ada fungsi transisi, sehingga belum ada elemen yang akan di *pop/push* ke *stack.*



b. Tahap selanjutnya, akan dibaca *input* a :

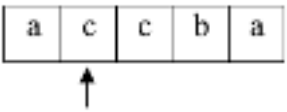


dengan fungsi transisi δ (q_0, a, λ) = {(q_1, λ)}, sehingga PDA dan *stack* menjadi :

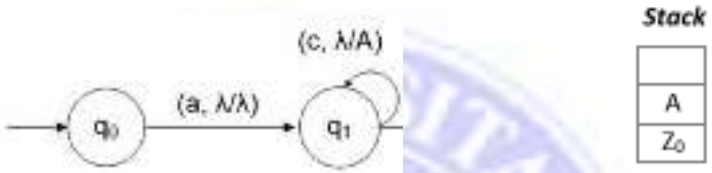


- Kondisi *stack* masih kosong karena fungsi transisi melakukan *push* λ.

c. Tahap selanjutnya, akan dibaca *input* c :

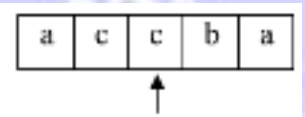


dengan fungsi transisi δ (q_1, c, λ) = {(q_1, A)}, sehingga PDA dan *stack* menjadi :

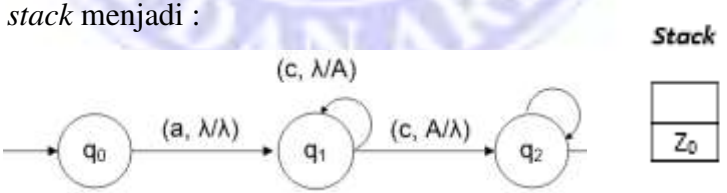


- Kondisi *stack* berisi simbol A karena fungsi transisi melakukan *push* A.

d. Tahap selanjutnya, akan dibaca *input* c :

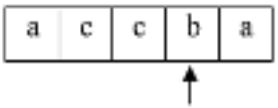


dengan fungsi transisi δ (q_1, c, A) = {(q_2, λ)}, sehingga PDA dan *stack* menjadi :



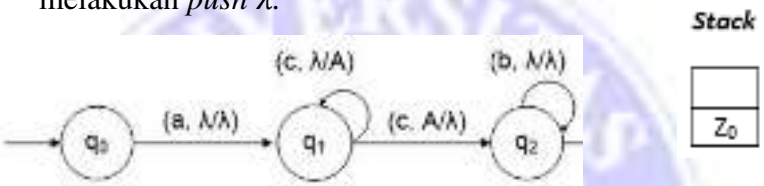
- Kondisi *stack* kembali kosong karena fungsi transisi melakukan *pop* A.

e. Tahap selanjutnya, akan dibaca *input* b :

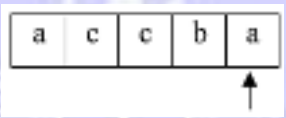


dengan fungsi transisi δ (q_2, b, λ) = {(q_2, λ)}, sehingga PDA dan *stack* menjadi :

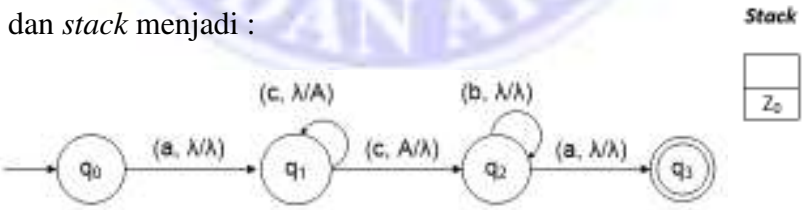
- Kondisi *stack* kembali kosong karena fungsi transisi melakukan *push* λ.



f. Tahap selanjutnya, akan dibaca *input* a :

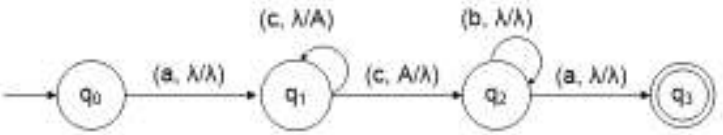


dengan fungsi transisi δ (q_2, a, λ) = {(q_3, λ)}, sehingga PDA dan *stack* menjadi :



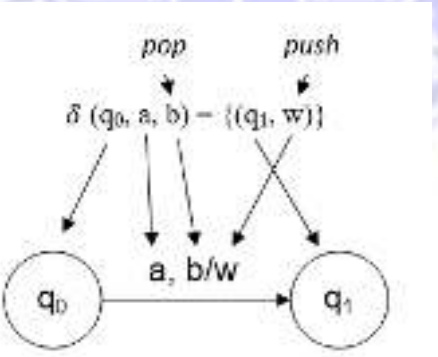
- Kondisi *stack* kembali kosong karena fungsi transisi melakukan *push* λ.

g. Karena seluruh *input string* sudah terbaca dengan baik, dan *stack* juga sudah kosong, maka berikut ini *Push Down Automata* yang dapat menerima *input* accba :



Rangkuman

1. Proses pada mesin PDA terdiri dari dua hal, yaitu :
   a. Apabila terdapat perintah *pop,* maka elemen pada *stack* yang posisinya paling atas akan dikeluarkan,
   b. Apabila terdapat perintah *push*, maka *input string* akan dimasukkan ke posisi paling atas pada *stack*.
2. Ilustrasi hubungan antara fungsi transisi dan *state*, dapat dilihat pada gambar dibawah ini :

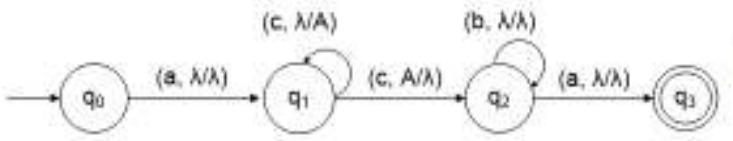


3. Sebuah deretan *string* dapat diterima oleh mesin PDA apabila seluruh *string* sudah terbaca dengan baik, fungsi transisi dapat

mencapai *final state* dan semua elemen *stack* sudah kosong (hanya tinggal Z_0 sebagai *top of stack*).

Tugas

1. Buatlah *Push Down Automata* yang dapat menerima *input* abccada. Tentukan konfigurasi yang sesuai seperti jumlah *state* awal, *state* akhir dan fungsi transisi. Tunjukkan setiap perubahan proses yang terjadi dalam pembuatan mesin PDA beserta dengan keadaan *stack*.
2. Lakukan analisis terhadap mesin *Push Down Automata* berikut, tuliskan proses 3 deretan *string* yang dapat diterima oleh mesin *Push Down Automata* tersebut.



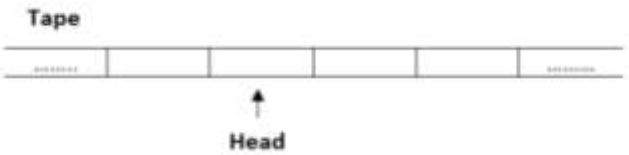
BAB XIV
MESIN TURING

Capaian Pembelajaran Mata Kuliah

❖ Mahasiswa dapat memahami teori tentang Mesin Turing

14.1. Mesin Turing

Mesin Turing merupakan mesin yang mampu mengelola *natural language* dan tidak memiliki aturan produksi. Mesin Turing memiliki kemampuan yang lebih tinggi dari *Push Down Automata.* Berbeda dengan mesin *Push Down Automata* yang menggunakan *stack*, mesin Turing menggunakan memori berbentuk pita (*Tape)* yang panjangnya tak berhingga. Pada mesin Turing, terdapat sebuah penunjuk yang akan mengakses setiap ruang pita, penunjuk ini disebut dengan *Head*. Ilustrasinya pita *(Tape)* beserta *Head*, dapat dilihat pada gambar berikut :



Gambar 14.1 Pita *(Tape)* Mesin Turing

Konfigurafi mesin Turing dapat dinyatakan dalam 7 Tupel, M = {Q, Σ, Γ, δ, S, F, ƀ}, secara rinci dapat dipaparkan sebagai berikut:

Q = Kumpulan *State*

Σ = Simbol *Input*

Γ = Simbol Pita *(Tape)*

δ = Fungsi Transisi

S = *State* Awal

F = *State* Final

ƀ = Simbol kosong *(blank)* pada pita

Catatan :

Ruang kosong pada pita yang tidak berisi apapun, akan dianggap berisi simbol ƀ *(blank).*

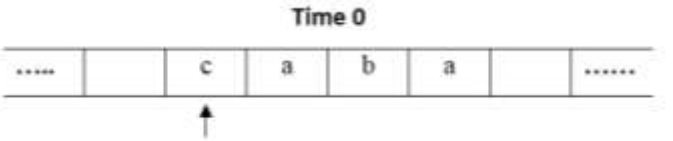
14.2. *Head* pada Mesin Turing

Head merupakan penunjuk yang kusus memperlihatkan posisi ruang pita yang sedang di akses. Satu kali pergerakan *Head* dalam mesin Turing memiliki 3 urutan perintah, yaitu :

a. Membaca *input string (Read)*
b. Menulis *input string (Write)*
c. Bergerak ke kiri *(Left)* atau ke kanan *(Right)*

Contoh :

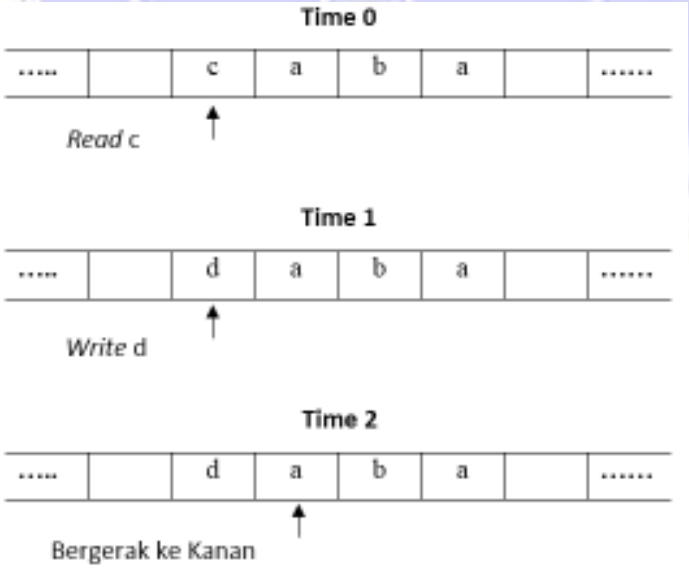
Pada sebuah mesin Turing :



Posisi *Head* berada di *input string* c, dan *Head* mendapat perintah:

1. *Read* c
2. *Write* d
3. Bergerak ke Kanan

Maka proses yang terjadi adalah :



Sehingga jika pada awalnya pita mesin Turing terdiri dari input *string* **caba**, sekarang pita mesin Turing terdiri dari *input string* **daba**.

14.3. Hubungan antara Fungsi Transisi dan State pada Mesin Turing

Seperti pada *Push Down Automata* sebelumnya, mesin Turing juga memiliki hubungan antara fungsi transisi dengan *state* yang cukup unik. Fungsi transisi pada mesin Turing memiliki format :

$$\delta\ (q_0, a) = (q_1, b, R)$$

atau

$$\delta\ (q_0, a) = (q_1, b, L)$$

Keterangan :

δ = Fungsi Transisi

q_0 = *State* Awal

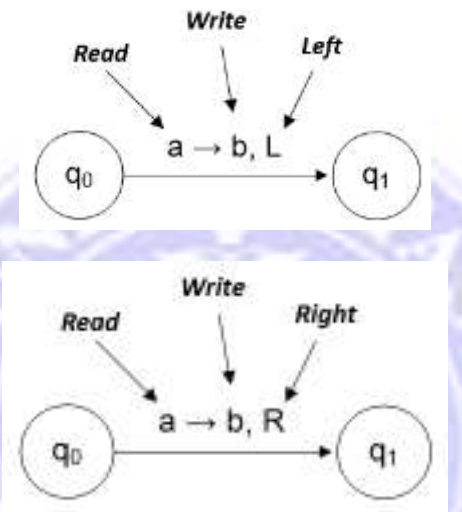
a = *Input* berupa *string*

b = *Input* yang akan dituliskan dalam ruang yang sama pada input a

q_1 = *State* Tujuan

R/L = Pergerakan *Head* : *Right* (R), *Left* (L).

Pada gambar di bawah ini dapat dilihat ilustrasi hubungan antara pergerakan *Head* seperti *Read*, *Write*, dan *Left* dengan *state* q_0 dan q_1.



Gambar 14.2 Hubungan Fungsi Transisi dan *State* pada Mesin Turing

14.4. Cara Kerja Mesin Turing

Mesin Turing merupakan salah satu mesin *automata deterministic* yang dapat memproses *input string* dan memberi keputusan apakah deretan *string* tersebut dapat diterima atau tidak dapat diterima.

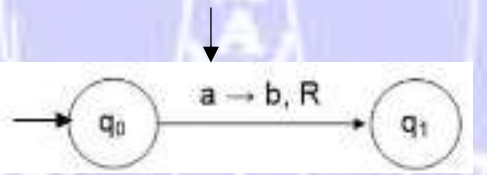
14.4.1. Mesin Turing Memproses *Input/ String*

Untuk dapat memahami cara mesin Turing membaca *input/ string*, diperlukan pemahaman mendalam tentang fungsi transisi. Deretan *string* pada pita mesin Turing dapat berubah sesuai dengan fungsi transisi. *String* dapat diterima apabila sudah mencapai *final state* dan *string* tidak dapat diterima apabila proses pembacaan *string* berhenti pada *state* yang bukan *final state* atau kondisi mesin melakukan *looping* yang tidak terhingga.

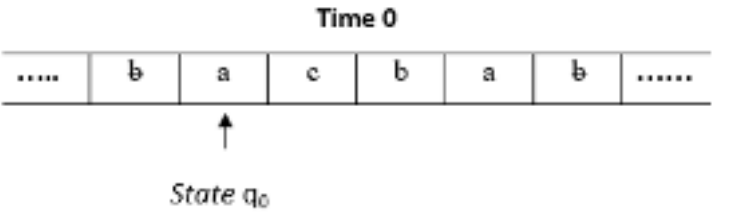
Contoh :

Terdapat fungsi transisi dan diagram transisi sebagai berikut :

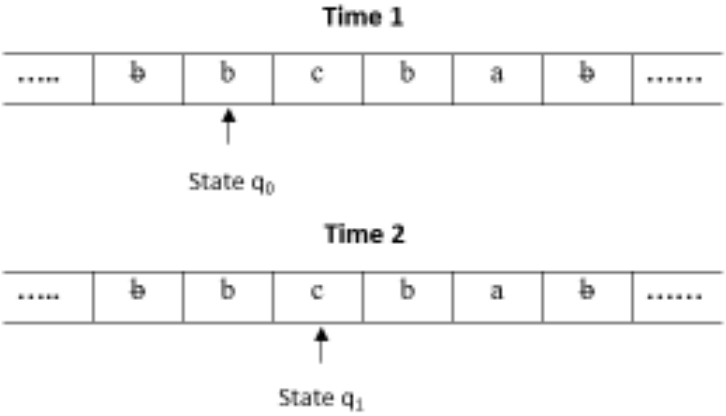
$$\delta\ (q_0, a) = (q_1, b, R)$$



Terdapat pita mesin Turing :



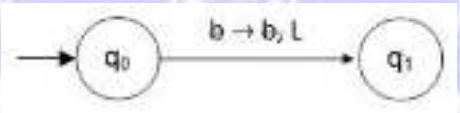
Dengan menerapkan fungsi transisi $\delta\ (q_0, a) = (q_1, b, R)$, pita mesin Turing menjadi :



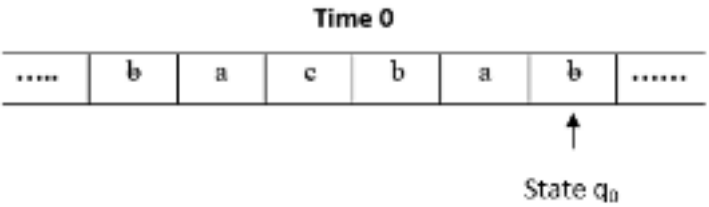
Contoh :

Terdapat fungsi transisi dan diagram transisi sebagai berikut :

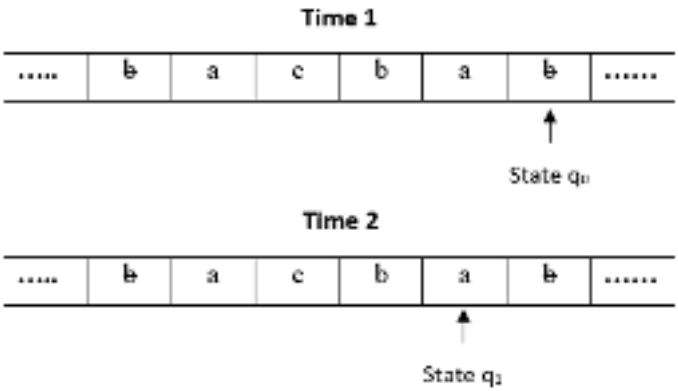
$$\delta\ (q_0, \text{b}) = (q_1, \text{b}, L)$$



Terdapat pita mesin Turing :



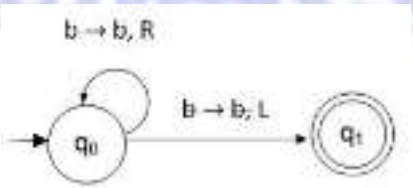
Dengan menerapkan fungsi transisi $\delta\ (q_0, \text{b}) = (q_1, \text{b}, L)$, pita mesin Turing menjadi :



- Pada mesin Turing ini, dapat dilihat pada Time 1 dan Time 2, tidak terjadi perubahan *string*. Hal ini terjadi karena *Head* mendapat perintah membaca ~~b~~ dan menuliskan ~~b~~, sehingga tidak terjadi perubahan apapun. Namun *Head* tetap bergeser ke kiri.

Contoh :

Pada mesin Turing dibawah ini, analisislah apakah deretan *string* **bbb** dan **bba** dapat diterima atau tidak.

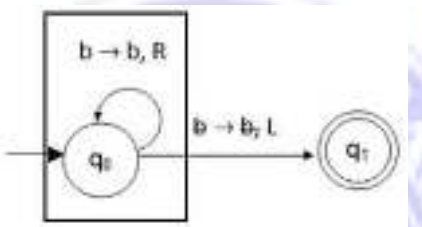


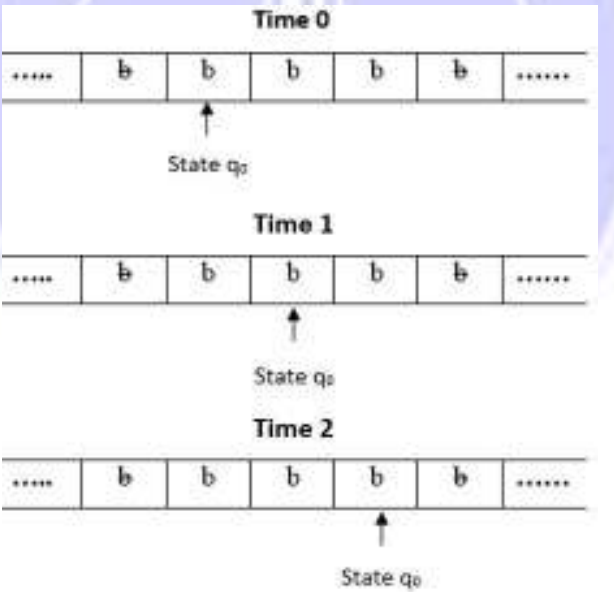
Analisis terhadap dua deret *string* tersebut adalah sebagai berikut:

a. *String* bbb :

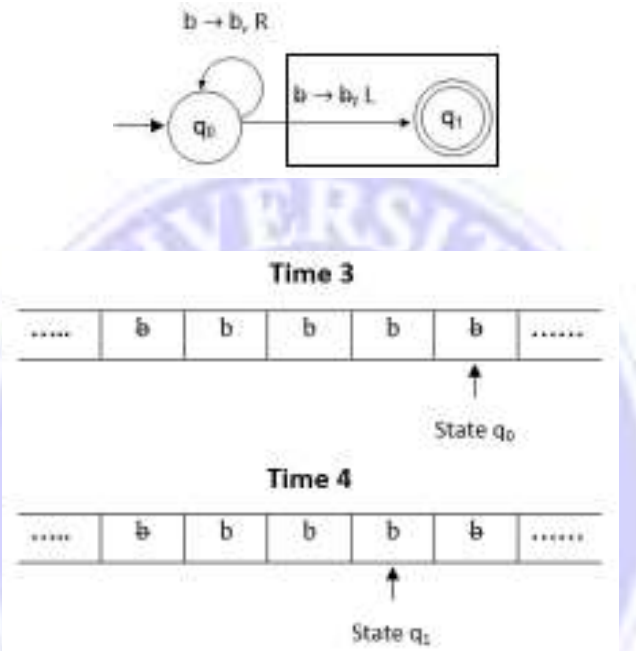
Dari mesin di atas, untuk *string* bbb, kita dapat membuat pita mesin Turing seperti berikut :

- Time 0, Time 1, dan Time 2 merupakan keadaan pita mesin Turing untuk pembacaan fungsi transisi δ (q_0, b) = (q_0, b, R) :





- Time 3 dan Time 4 merupakan keadaan pita mesin Turing untuk pembacaan fungsi transisi δ (q_0, ~~b~~) = (q_1, ~~b~~, L) :



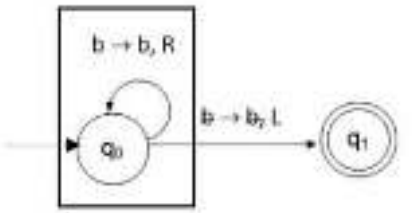
- Kesimpulan :

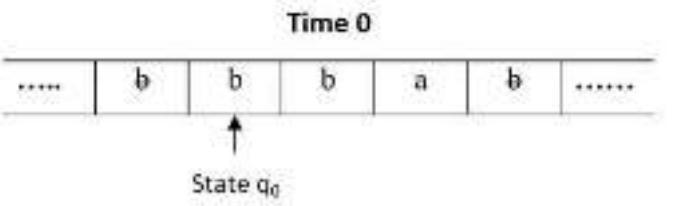
  String bbb dapat diterima oleh mesin Turing, karena *string* ini dapat diproses oleh mesin Turing sampai *final state.*

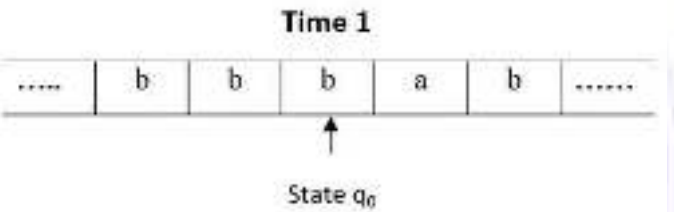
b. *String* bba :

Dari mesin di atas, untuk *string* bba, kita dapat membuat pita mesin Turing seperti berikut :

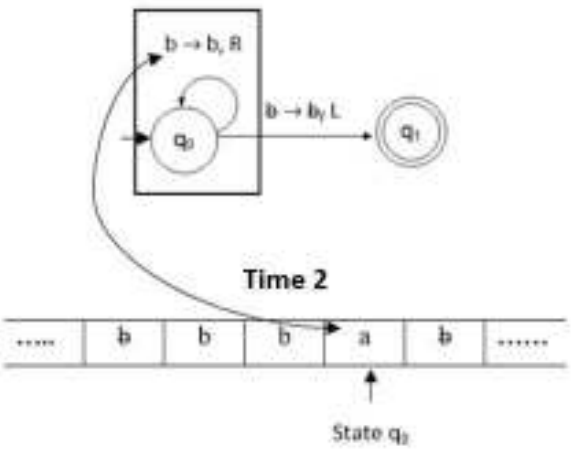
- Time 0 dan Time 1 merupakan keadaan pita mesin Turing untuk pembacaan fungsi transisi δ (q_0, b) = (q_0, b, R) :







- Pada Time 2, *input string* tidak dapat dibaca, karena terdapat perbedaan antara *string* dan fungsi transisi. Pada fungsi transisi, *head* diperintahkan membaca *string* b, namun pada pita, yang terindikasi adalah *string* a. Akibat tidak adanya keselarasan ini, maka proses pembacaan *string* tidak dapat dilanjutkan lagi. Secara rinci dapat dilihat pada gambar berikut :



- Kesimpulan :

  String bba tidak dapat diterima oleh mesin Turing, karena *string* ini tidak dapat diproses oleh mesin Turing sampai *final state.*

14.4.2. Membuat Mesin Turing yang Dapat Menerima Deretan *String*

Adapun langkah-langkah dalam membuat mesin Turing yang dapat menerima sebuah deretan *string* adalah sebagai berikut :

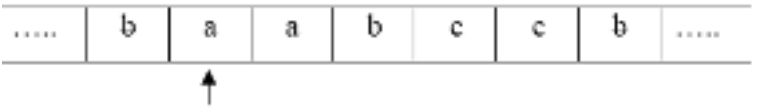
1. Perhatikan deretan *string* yang akan dikenali
2. Gambarkan deretan *string* pada pita mesin Turing secara berurutan
3. Bentuklah fungsi transisi berdasarkan pergerakan *Head* yang memungkinkan untuk menerima deretan *string*

4. Bentuklah mesin Turing dari fungsi transisi yang telah dibentuk pada tahap selanjutnya.

Contoh :

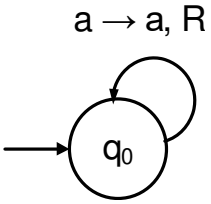
Terdapat sederetan *string* aabbcc, adapun proses pembentukan mesin Turing yang dapat menerima deretan *string* tersebut adalah:

1. Ilustrasi pita mesin Turing yang berisi *string* aabcc :

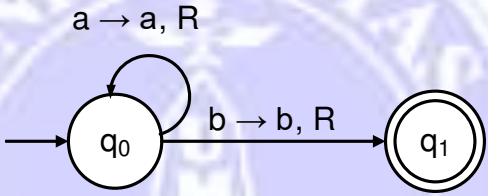


2. Fungsi transisi dan diagram transisi yang dapat menerima *string* aabcc dapat dibangun dari ilustrasi pergerakan *Head* pada setiap ruang pita.
   a. Posisi *head* sekarang berada pada *string* a, maka kita bisa asumsikan perintah *head* :
      - ❖ Membaca *string* a
      - ❖ Menulis *string* a
      - ❖ Bergerak ke kanan *(Right)*

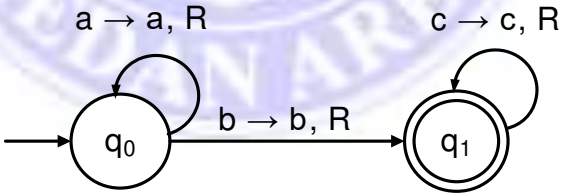
      Kemudian kita dapat berasumsi bahwa posisi *Head* berada di *state* q_0 dan mengalami *looping* (karena terdapat dua *string* a yang akan dibaca). Sehingga fungsi transisi dan diagram transisi yang dapat dibentuk adalah $\delta (q_0, a) = (q_0, a, R)$:



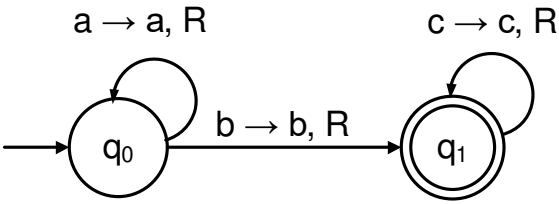
b. Untuk dapat menerima *string* b, proses pembacaan mesin Turing dapat dilakukan dari *state* q_0 menuju *state* q_1, sehingga fungsi transisi dan diagram transisi yang dapat dibentuk adalah $\delta\ (q_0\ ,\ b) = (q_1,\ b,\ R)$:



c. Untuk dapat menerima *string* cc, proses pembacaan mesin Turing dapat dilakukan pada *state* q_1 dan mengalami *looping*, sehingga fungsi transisi dan diagram transisi yang dapat dibentuk adalah $\delta\ (q_1\ ,\ c) = (q_1,\ c,\ R)$:

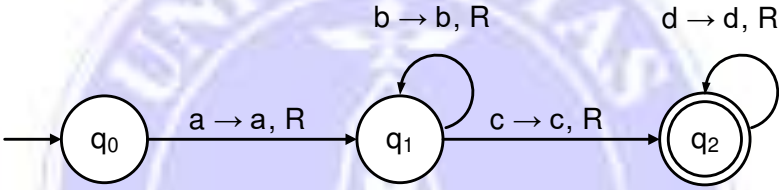


d. Setelah melalui tiga tahapan sebelumnya, maka diperolehlah mesin turing yang dapat menerima *string* aabcc :



Contoh Soal 14.1

Pada mesin Turing dibawah ini, analisislah apakah deretan *string* **abbcdd** dan **abbad** dapat diterima atau tidak.

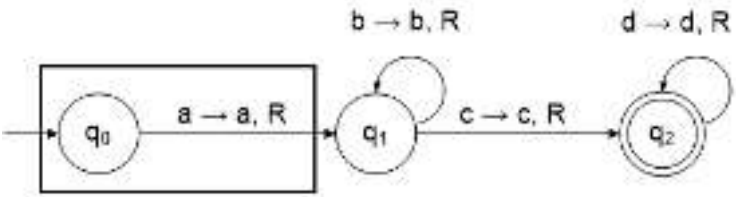


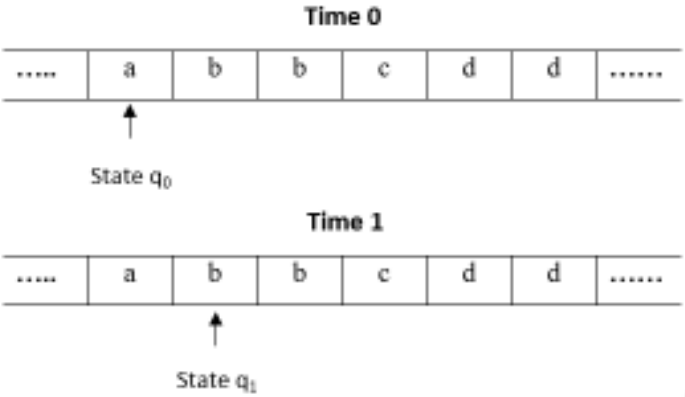
Penyelesaian Contoh Soal 14.1

a. *String* abbcdd :

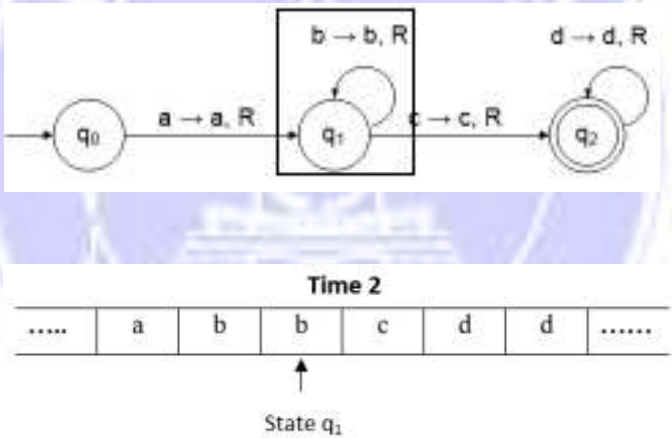
Dari mesin di atas, untuk *string* abbcdd, kita dapat membuat pita mesin Turing seperti berikut :

- Time 0 dan Time 1 merupakan keadaan pita mesin Turing untuk pembacaan fungsi transisi δ (q_0, a) = (q_1, a, R) :

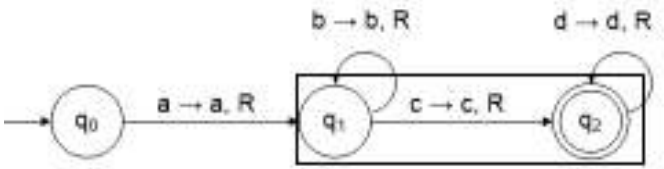


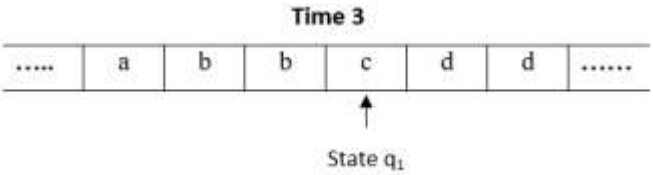


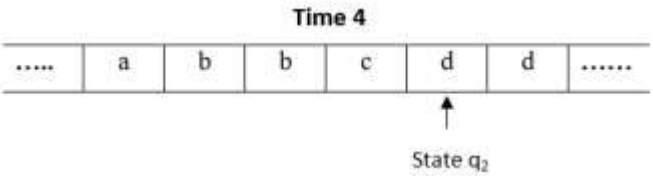
- Time 2 merupakan keadaan pita mesin Turing untuk pembacaan fungsi transisi δ (q_1, b) = (q_1, b , R) :



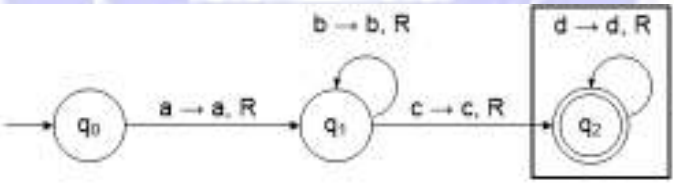
- Time 3 dan Time 4 merupakan keadaan pita mesin Turing untuk pembacaan fungsi transisi δ (q_1, c) = (q_2, c, R) :

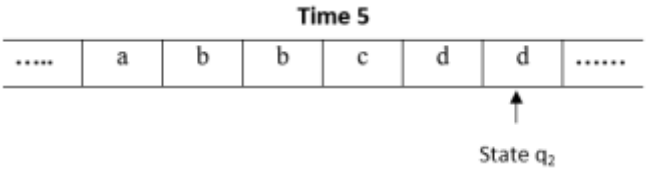






- Time 5 merupakan keadaan pita mesin Turing untuk pembacaan fungsi transisi δ (q_2, d) = (q_2, d , R) :





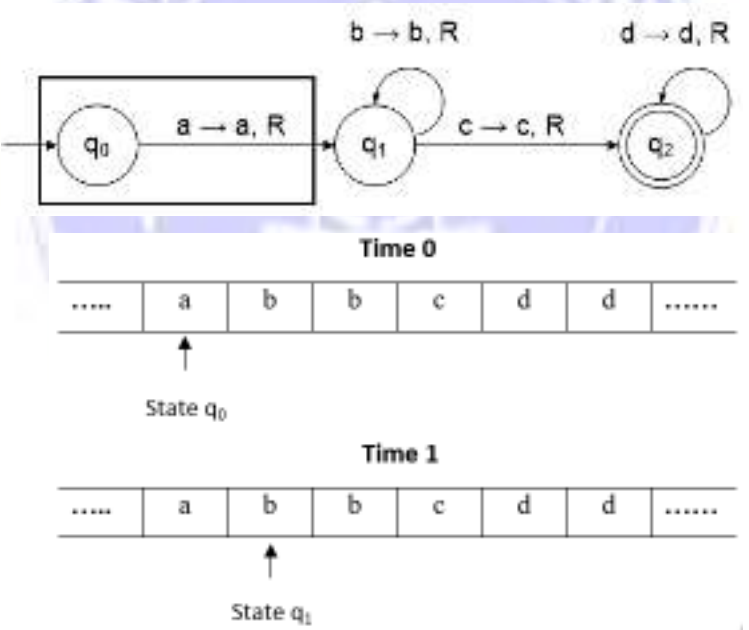
- Kesimpulan :

  String abbcdd dapat diterima oleh mesin Turing, karena *string* ini dapat diproses oleh mesin Turing sampai *final state.*

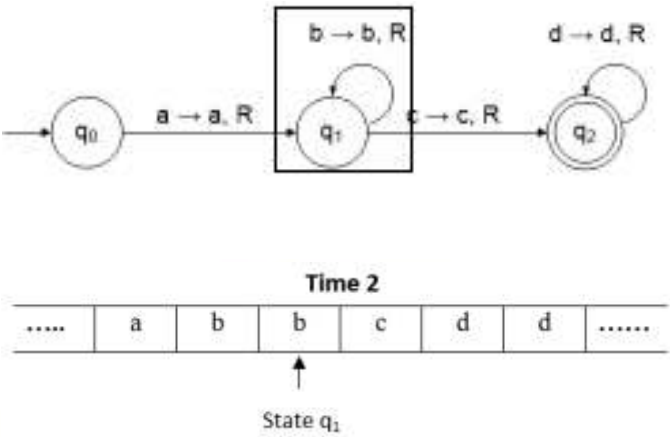
b. *String* abbad :

Dengan mesin Turing di atas, untuk *string* abbad, kita dapat membuat pita mesin Turing seperti berikut :

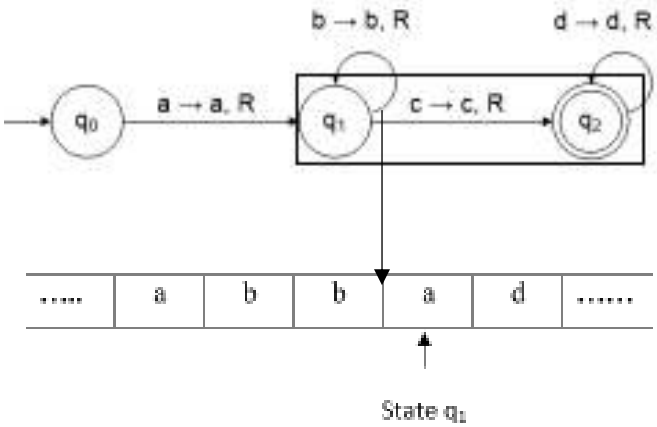
- Time 0 dan Time 1 merupakan keadaan pita mesin Turing untuk pembacaan fungsi transisi δ (q_0, a) = (q_1, a, R) :



- Time 2 merupakan keadaan pita mesin Turing untuk pembacaan fungsi transisi δ (q_1, b) = (q_1, b , R) :



- Pada Time 3, input *string* tidak dapat dibaca, karena terdapat perbedaan antara *string* dan fungsi transisi. Pada fungsi transisi, *head* diperintahkan membaca *string* a, namun pada pita, yang terindikasi adalah *string* c. Akibat tidak adanya keselarasan ini, maka proses pembacaan *string* tidak dapat dilanjutkan lagi. Secara rinci dapat dilihat pada gambar berikut:



- Kesimpulan :

  String abbad tidak dapat diterima oleh mesin Turing, karena *string* ini tidak dapat diproses oleh mesin Turing sampai *final state.*

Rangkuman

1. Mesin Turing menggunakan memori berbentuk pita (*Tape)* yang panjangnya tak berhingga. Pada mesin Turing, terdapat sebuah penunjuk yang akan mengakses setiap ruang pita, penunjuk ini disebut dengan *Head*.
2. *Head* dalam mesin Turing memiliki 3 urutan perintah, yaitu :
   a. Membaca *input string (Read)*
   b. Menulis *input string (Write)*
   c. Bergerak ke kiri *(Left)* atau ke kanan *(Right)*
3. *String* dapat diterima apabila sudah mencapai *final state* dan *string* tidak dapat diterima apabila proses pembacaan *string* berhenti pada *state* yang bukan *final state* atau kondisi mesin melakukan *looping* yang tidak terhingga.

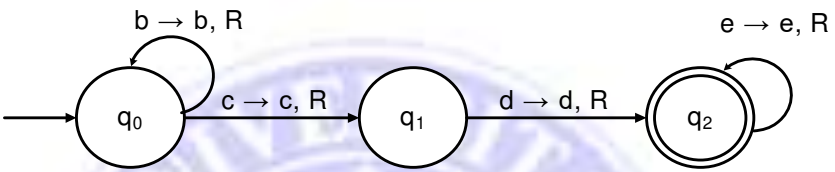
Tugas

1. Buatlah fungsi transisi, pita mesin Turing dan diagram Mesin Turing yang dapat menerima *input* :
   a. aabccdd

b. 0100110

c. cdeaabb

2. Analisis dan tuliskanlah 5 deretan *string* yang dapat diterima oleh mesin Turing berikut ini :



DAFTAR PUSTAKA

Adil, A., 2018. *Pengantar Teori Bahasa Formal, Otomata, dan Komputasi.* Yogyakarta: Deepublish.

Aranski, A. W., 2018. *Teori Bahasa dan Otomata.* Padang: Pustaka Galeri Mandiri.

E.Hopcroft, J., Motwani, R. & D.Ullman, J., 2001. *Introduction to Automata Theory, Languages and Computation 2nd Edt.* Second ed. London: Pearson Education Inc.

Meduna, A., 2014. *Formal Languages and Computation : Models and Their Applications.* 1st Edition ed. Florida: CRC Press.

Misdram, M., 2019. *Teknik Kompilasi.* Yogyakarta: Qiara Media.

Peter, L., 2016. *An Introduction to Formal Languages and Automata.* 6th Edition ed. Burlington: Jones & Barlett Publishers.

Revesz, G. E., 2015. *Introduction to Formal Languages Dover Books on Mathematics.* Edition Reprint ed. North Chelmsford: Courier Corporation.

Utdirartatmo, F., 2009. *Teori Bahasa dan Otomata.* I ed. Yogyakarta: J & J Learning Yogyakarta.

GLOSARIUM

Ambiguitas

Terdapat lebih dari satu model pohon penurunan untuk menyelesaikan kasus yang sama dalam memperoleh sebuah deretan *string*.

Aturan Produksi

Sebuah aturan yang menjadi standart acuan bagi empat tingkatan bahasa Hieraki Chomsky

Automata

Mesin yang dibuat dari model matematika yang dapat membaca dan mengenali setiap *input*, dan dapat mengambil kesimpulan apakah deretan *input* tersebut dapat diterima atau tidak dapat diterima

Bentuk Normal Chomsky

Bentuk normal yang digunakan pada tata bahasa bebas konteks *(Context Free Grammar)*

Compiler

Menterjemahkan program komputer yang ditulis dalam bahasa pemrograman tertentu menjadi program komputer dalam bahasa pemrograman lain

***Deterministic Finite Automata* (DFA)**

Jenis dari mesin *Finite State Automata* (FSA) yang memiliki ciri khas dimana setiap *state* memiliki tepat satu *state* tujuan berikutnya

ε - *move*

Menelusuri *state* tujuan tanpa membaca *input* dari *state* sebelumnya

Ekspresi Reguler

Bahasa yang dapat diterima oleh mesin *Finite State Automata*

Ekuivalensi

Konversi sebuah jenis mesin *automata* menjadi jenis mesin *automata* lainnya

Final State

State akhir yang berfungsi untuk mengeluarkan *output* mesin *automata*.

***Finite State Automata* (FSA)**

Mesin *Automata* yang dapat memproses tata bahasa bebas reguler

Fungsi Transisi

Hubungan antara *state* dan *input* yang disajikan dalam fungsi

Head

Penunjuk pada pita mesin Turing yang berfungsi menunjukkan posisi ruang pita yang sedang diakses

Hierarki Chomsky

Tingkatan bahasa yang dibatasi oleh aturan produksi tertentu.
Identifier Konstanta, variabel

Input

Bahasa, simbol, variabel yang akan di proses pada mesin *automata*

K*eyword*

Kata kunci dari sebuah kalimat

Leftmost Derivation

Pembentukan pohon penurunan dari simbol variabel yang paling kiri.

Lexical Analyzer

Membaca dan mengenali kata-kata, kemudian menghasilkan rangkaian simbol atau *string* yang lebih sederhana

Linear Bounded Automata

Mesin *Automata* yang dapat memproses bahasa *Context Sensitive*

Logical Unit

Unit terkecil dalam *input* seperti *identifier*, *keyword* dan *punctuation*.

Mesin Mealy

Salah satu jenis mesin *Finite State Automata* yang memiliki output berdasarkan transisi.

Mesin Moore

Mesin *Finite Statae Automata* yang dapat memiliki luaran tertentu dan tidak terbatas pada hanya sebuah *input* diterima atau tidak dapat diterima

Mesin Turing

Mesin *Automata* yang mampu memproses bahasa *Unrestricted/ Natural Language*

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

Bagian dari mesin *Finite State Automata* (FSA) yang memiliki ciri khas dimana sebuah *state* boleh tidak memiliki 1 arah busur atau lebih pada *input* yang sama menuju *state* berikutnya.

Nullable

Aturan produksi kosong

Output

Bahasa, simbol, dan variabel yang dapat diterima oleh mesin *automata*

Parsing

Pohon Penurunan

Pop

Perintah untuk mengambil elemen dari *stack*

Punctuation

Tanda baca, simbol

Push

Perintah untuk memasukkan elemen ke dalam *stack*

***Push Down Automata* (PDA)**

Mesin *Automata* yang dapat memproses bahasa Bebas Konteks *(Context Free)*

Rekursif

Aturan produksi yang mengalami perulangan baik dari ruas kanan ataupun ruas kiri.

Rightmost Derivation

Pembentukan pohon penurunan dari simbol variabel yang paling kanan.

Stack

Tumpukan data yang dinamis yang menerapkan prinsip LIFO *(Last In First Out)*

State

Informasi tentang *input*, dan dianggap juga sebagai memori mesin *automata*

String

Input mesin automata yang terdiri dari kumpulan huruf alfabet a-z dan biner 0 atau 1

Tabel Transisi

Hubungan antara *state* dan *input* yang disajikan dalam tabel

Tape

Memori mesin Turing yang berbentuk pita dengan panjang yang tak terhingga

Terminal

Simbol aturan produksi yang tidak dapat diturunkan lagi, dapat dibuktikan melalui pohon penurunan

Tupel

Variabel dan simbol khusus yang mendeskripsikan konfigurasi sebuah mesin *Automata*.

Useless

Aturan produksi yang tidak berguna atau berlebihan

Variabel

Simbol dari aturan produksi yang masih dapat diturunkan lagi, hal ini dapat dibuktikan dengan pohon penurunan

ε- *Closure*

Kumpulan *state-state* yang dapat ditelusuri tanpa harus membaca *input*

INDEKS

T

U

V

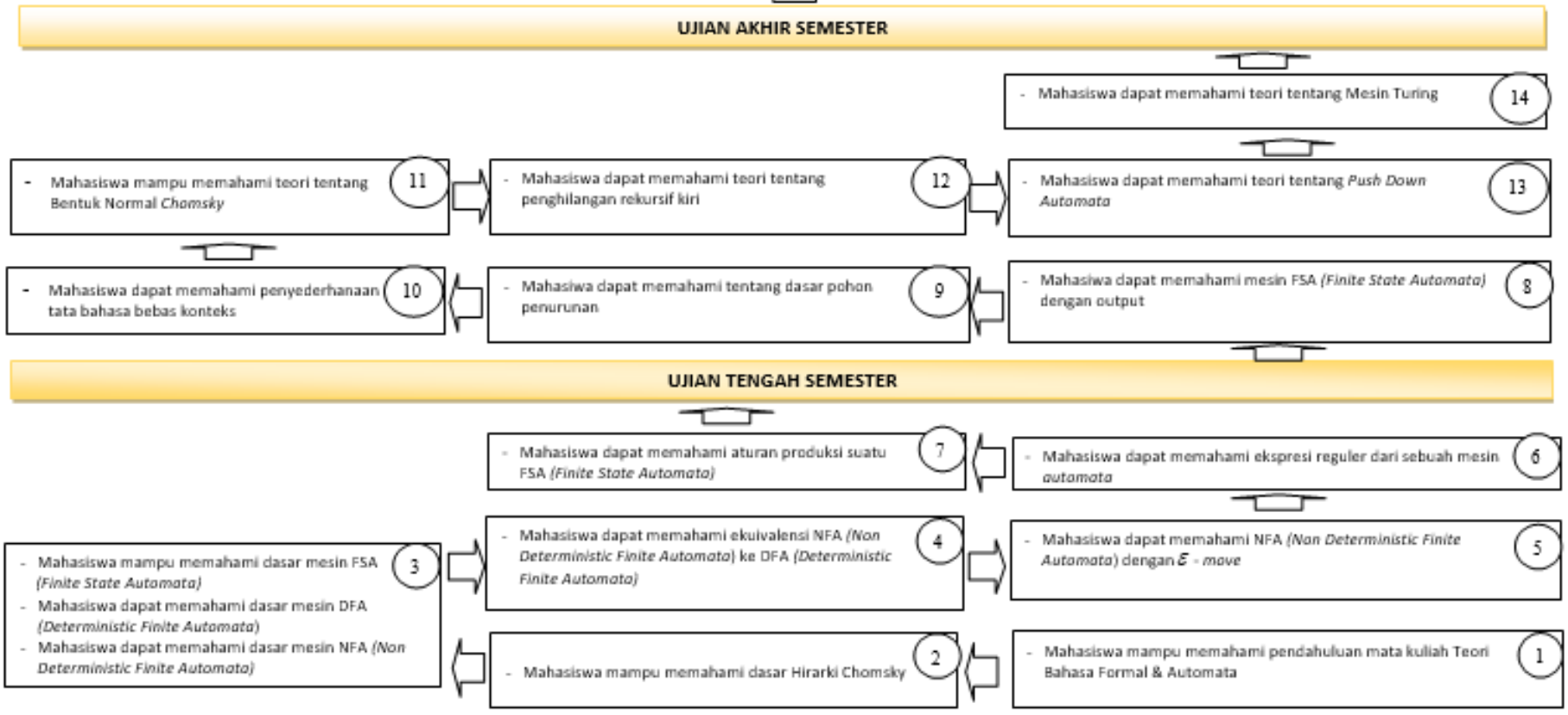
LAMPIRAN

	UNIVERSITAS MEDAN AREA **FAKULTAS TEKNIK** **PROGRAM STUDI TEKNIK INFORMATIKA**				
	RENCANA PEMBELAJARAN SEMESTER				
MATA KULIAH (MK)		**KODE**	**BOBOT (sks)**	**SEMESTER**	**Tgl. Penyusunan**
Teori Bahasa Formal & *Automata*		TIF16022	3 SKS	IV	1 Maret 2019
Pengembang RPS		**Koordinator RMK**		**Ketua PRODI**	
Program Studi Teknik Informatika				Juanda Hakim Lubis, ST, M.Kom	
Capaian Pembelajaran (CP)	**CPL-PRODI**				
	Diisi dengan CPL prodi yang dibebankan pada mata kuliah, dilengkapi dengan kode sesuai dengan komponen dikti (S, PP, KU, KK)	1. Menunjukkan sikap bertanggungjawab atas pekerjaan di bidang keahliannya secara mandiri. (S-9) 2. Menguasai prinsip dan teknik penyelesaian permasalahan dengan menggunakan: kalkulus, matriks, statistika, aproksimasi, optimasi liner, pemodelan dan simulasi. (PP-7) 3. Mampu mengkaji implikasi pengembangan atau implementasi ilmu pengetahuan teknologi yang memperhatikan dan menerapkan nilai humaniora sesuai dengan keahliannya berdasarkan kaidah, tata cara dan etika ilmiah dalam rangka menghasilkan solusi, gagasan, desain atau kritik seni. (KU-3) 4. Mampu menyelesaikan persoalan komputasi dan pemodelan matematis melalui pendekatan eksak, stokastik, probabilistik dan numerik secara efektif dan efisien. (KK-6)			
	CPMK				
	CP-MK merupakan uraian spesifik dari CPL-Prodi yang berkaitan dengan mata kuliah Teori Bahasa Formal & Automata	Setelah mengikuti kuliah ini, mahasiswa mampu menjelaskan menjelaskan ciri-ciri dari tata bahasa yang termasuk ke dalam klasifikasi Chomsky, menjelaskan tata bahasa yang termasuk ke dalam kelas tata bahasa reguler, bebas konteks *(Context Free)*, *Context Sensitive*, dan *Natural Language*, serta dapat memecahkan persoalan-persoalan yang berkaitan dengan deretan simbol *(string)*, serta dapat menerapkan Teori Bahasa Formal dan *Automata* pada bidang informatika lainnya seperti teknik kompilasi dan rekayasa perangkat lunak.			

Diskripsi Singkat MK	Mata kuliah ini berisi penjelasan teori dan pemodelan tata bahasa *(grammar)* yang dijelaskan pada Hirarki Chomsky. Tata bahasa yang dimaksud adalah *regular grammar*, *context sensitive grammar*, *context free grammar*, dan *unrestricted grammar*. Pemodelan bahasa mengikuti model mesin yang bersesuaian dengan tata bahasa.
Dosen pengampu	Nurul Khairina, S.Kom, M.Kom
Matakuliah syarat	Matematika Diskrit

CPMK

Setelah mengikuti kuliah ini, mahasiswa mampu menjelaskan menjelaskan ciri-ciri dari tata bahasa yang termasuk ke dalam klasifikasi Chomsky, menjelaskan tata bahasa yang termasuk ke dalam kelas tata bahasa reguler, bebas konteks *(Context Free)*, *Context Sensitive*, dan *Natural Language*, serta dapat memecahkan persoalan-persoalan yang berkaitan dengan deretan simbol (string), serta dapat menerapkan Teori Bahasa Formal dan Automata pada bidang informatika lainnya seperti teknik kompilasi dan rekayasa perangkat lunak.



Mg Ke-	Kemampuan Akhir yang diharapkan (Sub-CPMK)	Materi/ Bahan Kajian	Metode Pembelajaran	Waktu	Pengalaman Belajar Mahasiswa	Kriteria dan Indikator Penilaian	Bobot Nilai (%)
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
1	Mahasiswa mampu memahami pendahuluan mata kuliah Teori Bahasa Formal & *Automata*	Teori Bahasa dan kaitannya dengan mesin *Automata*	Problem Based Learning & Inquiry (PBL)	3 x 50 menit	**Tugas 1 :** Membuat mesin *automata* dari sederetan *string* yang diberikan Membaca *string* yang dapat diterima oleh mesin *automata*	**Kriteria Penilaian :** Ketepatan dan penguasaan materi **Indikator Penilaian :** Ketepatan memahami konsep dasar Teori Bahasa Formal & *Automata* **Bentuk Penilaian :** **Non-Tes :** Keaktifan dalam diskusi **Tes :** Ketepatan memahami cara membuat mesin *automata* dan cara membaca *string* dari mesin *automata*	7 %
2	Mahasiswa mampu memahami dasar Hirarki Chomsky	Teori tentang Hirarki Chomsky yang terdiri dari : 1. Tata bahasa regular 2. Tata bahasa bebas 3. konteks 4. Tata bahasa *context sensitive*	Problem Based Learning & Inquiry (PBL)	3 x 50 menit	**Tugas 2 :** Menentukan apakah sebuah aturan produksi memenuhi tata Bahasa regular, bebas konteks, *context sensitive* atau *Natural Language*	**Kriteria Penilaian :** Ketepatan dan penguasaan materi **Indikator Penilaian :** Ketepatan memahami Hirarki Chomsky	7 %

		5. Tata bahasa Unrestricted/ *Natural Language*				**Bentuk Penilaian :** **Non-Tes :** Keaktifan dalam diskusi **Test :** Ketepatan memahami aturan produksi dengan tata bahasa	
3	Mahasiswa mampu memahami dasar mesin FSA *(Finite State Automata)* Mahasiswa dapat memahami dasar mesin DFA *(Deterministic Finite Automata*) Mahasiswa dapat memahami dasar mesin NFA *(Non Deterministic Finite Automata*)	Mesin FSA *(Finite State Automata)* Mesin DFA *(Deterministic Finite Automata*) NFA *(Non Deterministic Finite Automata*)	Problem Based Learning & Inquiry (PBL)	3 x 50 menit	**Tugas 3 :** Membuat mesin DFA dan NFA dari fungsi transisi dan tabel transisi Membuat mesin NFA yang dapat menerima *string* tertentu	**Kriteria Penilaian :** Ketepatan dan penguasaan materi **Indikator Penilaian :** Ketepatan memahami fungsi transisi dan tabel transisi pada mesin DFA dan NFA **Bentuk Penilaian :** **Non-Tes :** Keaktifan dalam diskusi **Test :** Ketepatan membuat fungsi transisi dan tabel transisi	7 %
4	Mahasiswa dapat memahami ekuivalensi NFA *(Non Deterministic Finite Automata*) ke DFA *(Deterministic Finite Automata)*	Mengubah mesin NFA *(Non Deterministic Finite Automata)* menjadi mesin DFA *(Deterministic Finite Automata*)	Problem Based Learning & Inquiry (PBL)	3 x 50 menit	**Tugas 4 :** Membuat DFA *(Deterministic Finite Automata*) yang ekuivalen dengan NFA	**Kriteria Penilaian :** Ketepatan dan penguasaan materi **Indikator Penilaian :**	7 %

					(Non Deterministic Finite Automata)	Ketepatan memahami ekuivalensi DFA *(Deterministic Finite Automata)* dengan NFA *(Non Deterministic Finite Automata*) **Bentuk Penilaian :** **Non-Tes :** Keaktifan dalam diskusi **Test :** Ketepatan dalam menentukan ekuivalensi antar mesin *automata*	
5	Mahasiswa dapat memahami NFA *(Non Deterministic Finite Automata*) dengan ε *- move*	Perbedaan antara mesin NFA *(Non Deterministic Finite Automata*) dengan NFA *(Non Deterministic Finite Automata*) dengan ε *- move*	Problem Based Learning & Inquiry (PBL)	3 x 50 menit	**Tugas 5 :** Mengubah NFA *(Non Deterministic Finite Automata*) dengan ε *- move* menjadi NFA *(Non Deterministic Finite Automata*) tanpa ε *- move*	**Kriteria Penilaian :** Ketepatan dan penguasaan materi **Indikator Penilaian :** Ketepatan memahami ekuivalensi NFA *(Non Deterministic Finite Automata*) tanpa ε *- move* **Bentuk Penilaian :** **Non-Tes :** Keaktifan dalam diskusi	7 %

						Test : Ketepatan dalam menentukan ekuivalensi antar mesin NFA *(Non Deterministic Finite Automata*)	
6	Mahasiswa dapat memahami ekspesi regular dari sebuah mesin *automata*	Notasi ekspresi reguler dan hubungannya dengan FSA *(Finite State Automata)*	Problem Based Learning & Inquiry (PBL)	3 x 50 menit	**Tugas 6 :** Menentukan ekspresi regular dari mesin *automata*	**Kriteria Penilaian :** Ketepatan dan penguasaan materi **Indikator Penilaian :** Ketepatan memahami ekpresi reguler **Bentuk Penilaian :** **Non-Tes :** Keaktifan dalam diskusi **Test :** Ketepatan dalam menentukan ekspresi regular dari sebuah bahasa	7%
7	Mahasiswa dapat memahami aturan produksi suatu FSA *(Finite State Automata)*	Simbol-simbol aturan produksi, penulisan formal sebuah aturan produksi dari sebuah mesan FSA *(Finite State Automata)*	Problem Based Learning & Inquiry (PBL)	3 x 50 menit	**Tugas 7 :** Menentukan aturan produksi yang sesuai dengan tata Bahasa regular dari mesin automata	**Kriteria Penilaian :** Ketepatan dan penguasaan materi **Indikator Penilaian :** Ketepatan memahami aturan produksi	7 %

						Bentuk Penilaian : **Non-Tes :** Keaktifan dalam diskusi **Test :** Ketepatan dalam menentukan aturan produksi yang sesuai dengan mesin *automata*	
Evaluasi Tengah Semester							
8	Mahasiswa dapat memahami mesin FSA *(Finite State Automata)* dengan Output	Komponen pada mesin Moore dan mesin Mealy beserta perbedaan kedua mesin tersebut	Problem Based Learning & Inquiry (PBL)	3 x 50 menit	**Tugas 8 :** Membuat mesin Moore dan mesin Mealy untuk menyelesaikan sebuah permasalahan	**Kriteria Penilaian :** Ketepatan dan penguasaan materi **Indikator Penilaian :** Ketepatan memahami mesin Moore dan mesin Mealy dalam menyelesaikan masalah **Bentuk Penilaian :** **Non-Tes :** Keaktifan dalam diskusi **Test :** Ketepatan dalam menentukan perbedaan antara mesin Moore dan mesn Mealy	8 %

9	Mahasiswa dapat memahami tentang dasar pohon penurunan	Tata Bahasa bebas konteks, *parsing* dan *ambiguitas*	Problem Based Learning & Inquiry (PBL)	3 x 50 menit	**Tugas 9 :** Membuat pohon penurunan dari tata Bahasa Bebas Konteks	**Kriteria Penilaian :** Ketepatan dan penguasaan materi **Indikator Penilaian :** Ketepatan memahami pohon penurunan **Bentuk Penilaian :** **Non-Tes :** Keaktifan dalam diskusi **Test :** Ketepatan dalam menentukan pohon penurunan	7 %
10	Mahasiswa dapat memahami penyederhanaan tata bahasa bebas konteks	Penghilangan produksi *useless,* produksi unit dan produksi ε	Problem Based Learning & Inquiry (PBL)	3 x 50 menit	**Tugas 10 :** Menghilangkan aturan produksi *useless*, produksi unit dan produksi ε dari sebuah tata Bahasa bebas konteks	**Kriteria Penilaian :** Ketepatan dan penguasaan materi **Indikator Penilaian :** Ketepatan memahami penyederhanaan tata Bahasa bebas konteks **Bentuk Penilaian :** **Non-Tes :** Keaktifan dalam diskusi **Test :** Ketepatan dalam	7 %

						penghilangan produksi dari sebuah tata bahasa bebas konteks	
11	Mahasiswa mampu memahami teori tentang Bentuk Normal Chomsky	Aturan produksi dalam Bentuk Normal Chomsky	Problem Based Learning & Inquiry (PBL)	3 x 50 menit	**Tugas 11 :** Mentransformasikan tata bahasa Bebas Konteks ke dalam Bentuk Normal Chomsky	**Kriteria Penilaian :** Ketepatan dan penguasaan materi **Indikator Penilaian :** Ketepatan memahami Bentuk Normal Chomsky **Bentuk Penilaian :** **Non-Tes :** Keaktifan dalam diskusi **Test :** Ketepatan dalam mengubah tata bahasa Bebas Konteks ke dalam Bentuk Normal Chomsky	7 %
12	Mahasiswa dapat memahami teori tentang penghilangan rekursif kiri	Aturan produksi rekursif, tahap penghilangan rekursif kiri	Problem Based Learning & Inquiry (PBL)	3 x 50 menit	**Tugas 12 :** Menghilangkan rekursif kiri pada tata bahasa bebas konteks	**Kriteria Penilaian :** Ketepatan dan penguasaan materi **Indikator Penilaian :** Ketepatan memahami bentuk aturan produksi rekursif **Bentuk Penilaian :** **Non-Tes :**	7 %

						Keaktifan dalam diskusi **Test :** Ketepatan mengilangkan rekursif kiri pada tata bahasa bebas konteks	
13	Mahasiswa dapat memahami teori tentang *Push Down Automata*	Mekanisme kerja *Push Down Automata, Push Down Automata* untuk suatu tata bahasa Bebas Konteks	Problem Based Learning & Inquiry (PBL)	3 x 50 menit	**Tugas 13 :** Menentukan aturan produksi tata bahasa Bebas Konteks yang dapat diterima oleh *Push Down Automata*	**Kriteria Penilaian :** Ketepatan dan penguasaan materi **Indikator Penilaian :** Ketepatan memahami *Push Down Automata* **Bentuk Penilaian :** **Non-Tes :** Keaktifan dalam diskusi **Test :** Ketepatan menentukan aturan produksi tata bahasa Bebas Konteks yang dapat diterima oleh *Push Down Automata*	7 %
14	Mahasiswa dapat memahami teori tentang Mesin Turing	Mekanisme kerja Mesin Turing	Problem Based Learning & Inquiry (PBL)	3 x 50 menit	**Tugas 14 :** Menentukan apakah sebuah bahasa dapat diterima oleh mesin Turing	**Kriteria Penilaian :** Ketepatan dan penguasaan materi **Indikator Penilaian :** Ketepatan memahami mesin Turing	7 %

						Bentuk Penilaian : **Non-Tes :** Keaktifan dalam diskusi **Test :** Ketepatan menentukan bahasa yang dapat diterima oleh mesin Turing	
	Evaluasi Akhir Semester						

Referensi :

1. Utdirartatmo, F., 2009. *Teori Bahasa dan Otomata.* I ed. Yogyakarta: J & J Learning Yogyakarta.
2. E.Hopcroft, J., Motwani, R. & D.Ullman, J., 2001. *Introduction to Automata Theory, Languages and Computation 2nd Edt.* Second ed. London: Pearson Education Inc.
3. Aranski, A. W., 2018. *Teori Bahasa dan Otomata.* Padang: Pustaka Galeri Mandiri.