Teori Bahasa dan Automata merupakan bagian dari teori komputasi pada ilmu komputer. Beberapa teori komputasi datang dari bahasa dan rekayasa system, terutama yang berbasiskan matematika. Ilmu komputer memiliki 2 komponen utama yaitu : pertama, model dan gagasan mendasar mengenai komputasi, Kedua, teknik rekayasa untuk untuk perancangan sistem komputasi, meliputi perangkat keras dan perangkat lunak, khususnya penerapan rancangan dari teori. Teori bahasa dan Automata merupakan bagian pertama. Secara teoritis ilmu komputer diawali dari sejumlah berbeda disiplin ilmu : ahli biologi mempelajari *Neural network*, insiyur elektro mengembangkan *switching* sebagai tool untuk mengembangakan hardware, matematikawan bekerja mendasarkan logika dan ahli bahasa menyelidiki tata bahasa untuk *natural language*.

Teori bahasa membicarakan bahasa formal (*formal language*), terutama untuk kepentingan perancangan kompilator (*compiler*) dan pemroses naskah (*text processor*). Bahasa formal adalah kumpulan *kalimat*. Semua kalimat dalam sebuah bahasa dibangkitkan oleh sebuah tata bahasa (*grammar*) yang sama. Sebuah bahasa formal bisa dibangkitkan oleh dua atau lebih tata bahasa berbeda. Dikatakan bahasa formal karena grammar diciptakan mendahului pembangkitan setiap kalimatnya. Bahasa manusia bersifat sebaliknya; grammar diciptakan untuk meresmikan kata-kata yang hidup di masyarakat. Otomata merupakan suatu sistem yang terdiri atas sejumlah berhingga *state*, di mana *state* menyatakan informasi mengenai *input*. Otomata juga dianggap sebagai mesin otomatis (bukan mesin fisik) yang merupakan suatu model matematika dari suatu system yang menerima input dan menghasilkan output, serta terdiri dari sejumlah berhingga state. Hubungan di antara bahasa dan otomata adalah bahasa dijadikan sebagai input oleh suatu mesin otomata, selanjutnya mesin otomata akan membuat keputusan yang mengindikasikan apakah input itu diterima atau tidak.

Finite State Automata dan ekspresi regular awalnya dikembangkan berdasar pemikiran *neural network* dan *switching circuit*. Finite Automata merupakan tool yang sangat berguna dalam perancangan *lexical analyzer*, yaitu bagian dari kompilator yang mengelompokkan karakter-karakter kedalam token yang berupa unit terkecil seperti nama, variable, dan *keyword*. Dalam system penulisan kompilator secara otomatis akan mentransformasikan ekspresi regular kedalam finite state automata untuk dipakai sebagai analisa leksikal.

Penerbit Deepublish (CV BUDI UTAMA)
Jl. Rajawali, Gang Elang 6 No.3, Drono, Sardonoharjo, Ngaglik, Sleman
Jl. Kaliurang Km 9,3 Yogyakarta 55581
Telp/Fax : (0274) 4533427
Anggota IKAPI (076/DIY/2012)
cs@deepublish.co.id @penerbitbuku_deepublish
Penerbit Deepublish www.penerbitbukudeepublish.com

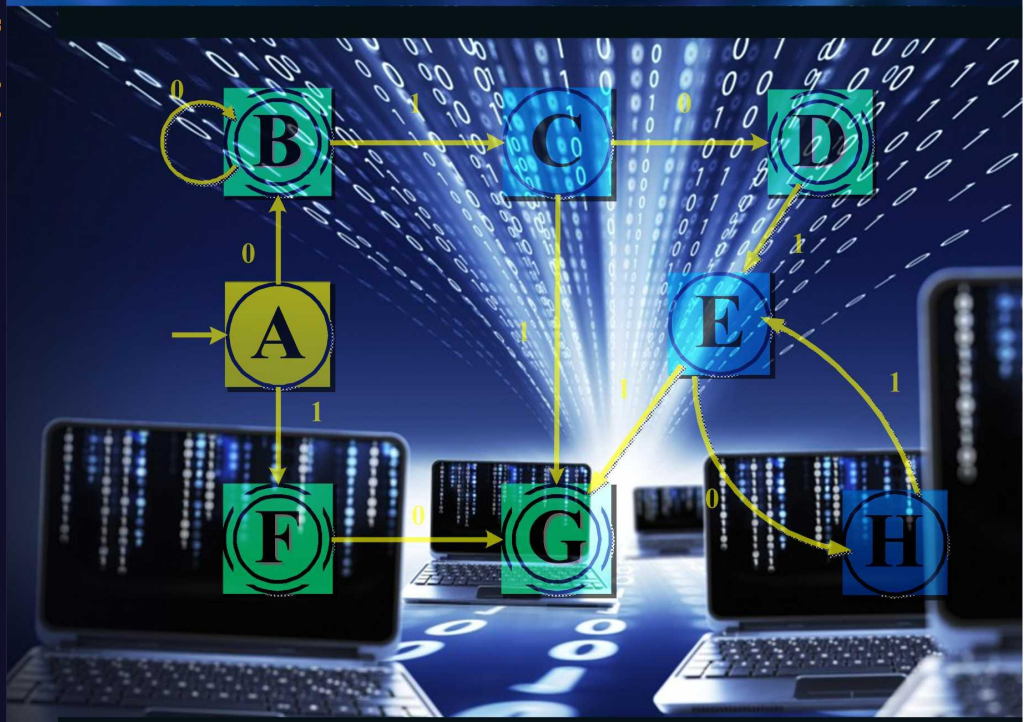
Kategori :

Pengantar Teori Bahasa Formal, Otomata dan Komputasi

Ahmat Adil, S.Kom., M.Sc.

Pengantar Teori

Bahasa Formal, Otomata dan Komputasi



Ahmat Adil, S.Kom., M.Sc.

Pengantar Teori Bahasa Formal, Otomata, dan Komputasi

UU No 28 tahun 2014 tentang Hak Cipta

Fungsi dan sifat hak cipta Pasal 4

Hak Cipta sebagaimana dimaksud dalam Pasal 3 huruf a merupakan hak eksklusif yang terdiri atas hak moral dan hak ekonomi.

Pembatasan Pelindungan Pasal 26

Ketentuan sebagaimana dimaksud dalam Pasal 23, Pasal 24, dan Pasal 25 tidak berlaku terhadap:

i. penggunaan kutipan singkat Ciptaan dan/atau produk Hak Terkait untuk pelaporan peristiwa aktual yang ditujukan hanya untuk keperluan penyediaan informasi aktual;
ii. Penggandaan Ciptaan dan/atau produk Hak Terkait hanya untuk kepentingan penelitian ilmu pengetahuan;
iii. Penggandaan Ciptaan dan/atau produk Hak Terkait hanya untuk keperluan pengajaran, kecuali pertunjukan dan Fonogram yang telah dilakukan Pengumuman sebagai bahan ajar; dan
iv. penggunaan untuk kepentingan pendidikan dan pengembangan ilmu pengetahuan yang memungkinkan suatu Ciptaan dan/atau produk Hak Terkait dapat digunakan tanpa izin Pelaku Pertunjukan, Produser Fonogram, atau Lembaga Penyiaran.

Sanksi Pelanggaran Pasal 113

1. Setiap Orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf i untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan/atau pidana denda paling banyak Rp100.000.000 (seratus juta rupiah).
2. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf c, huruf d, huruf f, dan/atau huruf h untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan/atau pidana denda paling banyak Rp500.000.000,00 (lima ratus juta rupiah).

Ahmat Adil, S.Kom., M.Sc

Pengantar Teori Bahasa Formal, Otomata, dan Komputasi

deepublish
glorify and develop the intellectual of human's life

PENGANTAR TEORI BAHASA FORMAL, OTOMATA, DAN KOMPUTASI

Ahmat Adil

Editor
Prof.Dr.Muhammad Tajuddin,M.Si.

Desain cover
Nama

Sumber
link

Tata letak:
Amira Dzatin Nabila

Ukuran:
xx, 245 hlm, Uk: 15.5x23 cm

ISBN:
No ISBN

Cetakan Pertama:
Oktober 2018

Hak Cipta 2018, Pada Penulis

Isi diluar tanggung jawab percetakan

Copyright © 2018 by Deepublish Publisher
All Right Reserved

Hak cipta dilindungi undang-undang
Dilarang keras menerjemahkan, memfotokopi, atau
memperbanyak sebagian atau seluruh isi buku ini
tanpa izin tertulis dari Penerbit.

PENERBIT DEEPUBLISH
(Grup Penerbitan CV BUDI UTAMA)
Anggota IKAPI (076/DIY/2012)

Jl.Rajawali, G. Elang 6, No 3, Drono, Sardonoharjo, Ngaglik, Sleman
Jl.Kaliurang Km.9,3 – Yogyakarta 55581
Telp/Faks: (0274) 4533427
Website: www.deepublish.co.id
www.penerbitdeepublish.com
E-mail: cs@deepublish.co.id

KATA PENGANTAR

Bismillahirrahmanirrahim
Assalamu'alaikum Wr. Wb.

Puji syukur penulis ucapkan kehadirat Allah SWT atas limpahan rahmat-Nya yang telah tercurah, sehingga penulis bisa menyelesaikan Buku yang berjudul **"Pengantar Teori Bahasa Formal, Otomata, dan Komputasi"**. Buku ini adalah buku ajar mata kuliah Teori Bahasa dan Otomata pada jurusan Teknik Informatika, yang disusun dengan tujuan supaya para mahasiswa mempunyai panduan mendapatkan gambaran yang jelas mengenai sasaran dan tujuan, materi dan contoh soal latihan tentang teori bahasa formal dan mesin penganal bahasa.

Teori Bahasa dan Otomata merupakan bagian dari teori komputasi pada ilmu komputer. Ilmu komputer memiliki 2 komponen utama yaitu: pertama, model dan gagasan mendasar mengenai komputasi, Kedua, teknik rekayasa untuk perancangan sistem komputasi, meliputi perangkat keras dan perangkat lunak, khususnya penerapan rancangan dari teori. Teori bahasa dan Automata merupakan bagian pertama. Secara teoritis ilmu komputer diawali dari sejumlah berbeda disiplin ilmu: ahli biologi mempelajari *Neural network*, insinyur elektro mengembangkan *switching* sebagai *tool* untuk mengembangkan *hardware*, matematikawan bekerja mendasarkan logika dan ahli bahasa menyelidiki tata bahasa untuk *natural language.*

Penulis telah mengampuh mata kuliah ini sejak tahun 1998, dan telah menyusun modul-modul pertemuan sehingga tersusunnya buku ini. Dukungan moral dan material dari berbagai pihak sangatlah membantu tersusunnya buku ini. Untuk itu, penulis ucapkan terima kasih kepada keluarga, sahabat, rekan-rekan, dan pihak-pihak lainnya yang membantu secara moral dan material bagi tersusunnya buku ini.

Penulisan buku ini terlaksana atas bantuan semua pihak yang telah berkenan memberikan saran, masukan yang bermanfaat. Untuk itu penulis

menyampaikan penghargaan dan ucapan terima kasih yang sebesar-besarnya kepada:

1. Direktur Jenderal Pembelajaran dan kemahasiswaan (Belmawa) Ristek Dikti, atas hibah penulisan Buku Ajar dan Pedoman pembelajaran tahun 2018.
2. Semua pihak yang telah banyak membantu dalam memberikan masukan dan saran dalam penyelesaian buku ini.

Buku yang tersusun sekian lama ini tentu masih jauh dari kata sempurna. Untuk itu, kritik dan saran yang membangun sangat diperlukan agar buku ini bisa lebih baik nantinya.

Penulis

DAFTAR ISI

KATA PENGANTAR v

DAFTAR ISI vii

RINGKASAN xiii

RENCANA KEGIATAN PEMBELAJARAN SEMESTER (RKPS) xv

BAB I
BAHASA FORMAL 1

1.1. Pendahuluan 1
1.2. Konsep Bahasa Formal 2
1.2.1. Abjad atau Alfabet (*Alpabet*) 2
1.2.2. Untai atau kata (*String*) 3
1.2.3. Bahasa 6
1.3. Bahasa Alami dan Bahasa Formal 7
1.4. Latihan soal 7

BAB II
OPERASI BAHASA FORMAL 9

2.1. Pendahuluan 9
2.2. Pengertian Dasar Operasi 9
2.3. Teori Dasar Operasi 10
2.3.1. Operasi dasar *String* 10
2.3.2. Beberapa Sifat Operasi 13
2.3.3. Perangkaian (*Concatenation*) 13
2.3.4. Eksponensiasil (*Exponentiation*) 14
2.3.5. Gabungan (*Union*) 14
2.3.6. Irisan *(Intersection*) 14
2.3.7. Sub Bahasa 15
2.3.8. Equal (Sama) 15
2.4. Latihan soal 15

BAB III
TATA BAHASA.....19
3.1. Pendahuluan 19
3.2. Elemen Bahasa Formal 19
3.3. Tata Bahasa Regular 22
3.4. Aturan Produksi.....24
3.5. Derivasi Kalimat dan Penentuan Bahasa 25
3.6. Latihan Soal.....27

BAB IV
OTOMATA BERHINGGA *(FINITE AUTOMATA)*.....30
4.1. Pendahuluan 30
4.2. Konsep Otomata (*Automata*).....31
4.3. Sifat-Sifat Otomata 34
4.4. *Finite State Automata* (FSA).....35
4.5. *Deterministic Finite Automata* (DFA) 42
4.6. *Nondeterministic Finite Automata* (NDFA).....45
4.7. Ekuivalensi NFA-DFA 49
4.8. Ekivalensi NFA ε - MOVE ke NFA TANPA ε - MOVE
NFA ε – MOVE.....54
4.9. Ekivalensi N FA ε - move ke NFA tanpa ε - MOVE.....55
4.10. Latihan Soal.....57

BAB V
FSA DENGAN OUTPUT.....64
5.1. Pendahuluan 64
5.2. Mesin Moore 64
5.3. Mesin Mealy.....67
5.4. Ekuivalensi mesin Moore dengan mesin Mealy 71
5.5. Latihan Soal.....73

BAB VI
TATA BAHASA BEBAS KONTEKS 77
6.1. Pendahuluan 77
6.2. Perbedaan CFG dan Bahasa Regular 77

6.3. Leftmost dan Rightmost Derivation 79
6.4. Pohon Urai 80
6.5. Pumping Lemma untuk Bahasa Bebas Konteks 82
6.6. Sifat-Sifat Tertutup Bahasa Bebas Konteks 83
6.7. Latihan Soal 85

BAB VII
BENTUK NORMAL CHOMSKY 92
7.1. Pendahuluan 92
7.2. Pembentukan Bentuk Normal Chomsky 92
7.3. *Hierarki Chomsky* 103
7.4. Klasifikasi Chomsky 109
7.5. Latihan Soal 113

BAB VIII
PENGHILANGAN REKURSIF KIRI 122
8.1. Pendahuluan 122
8.2. Aturan Produksi Rekursif 122
8.3. Tahapan Penghilangan Rekursif Kiri 125
8.4. Latihan Soal 128

BAB IX
PENYEDERHANAAN TATA BAHASA BEBAS KONTEKS 136
9.1. Pendahuluan 136
9.2. Penghilangan Produksi *Useless* 139
9.3. Penghilangan Produksi Unit 143
9.4. Penghilangan Produksi ε 145
9.5. Latihan Soal 149

BAB X
BENTUK NORMAL GREIBACH 159
10.1. Pendahuluan 159
10.2. Pengertian Bentuk Normal Greibach 159
10.3. Pembentukan *Bentuk Normal Greibach* dengan Substitusi 160

10.4. Pembentukan Normal Greibach Melalui Perkalian Matriks 164
10.5. Latihan Soal 167

BAB XI
DFA DAN TATA BAHASA REGULER 170
11.1. Pendahuluan 170
11.2. Mengkonstruksi aturan produksi Otomata Hingga untuk tata bahasa regular 171
11.3. Tata Bahasa Linier Kiri Dan Linier Kanan 173
11.4. Konversi DFA ke Tata bahasa linier 174
11.5. Konversi Tata bahasa linier ke DFA 175
11.6. Pumping Lemma 176
11.7. Konstruksi NDFA dari Tata bahasa linier Kanan 178
11.8. Konstruksi NDFA dari Tata bahasa linier kiri 179
11.9. Latihan Soal 179

BAB XII
REGULAR EKSPRESI 182
12.1. Pendahuluan 182
12.2. Notasi Ekspresi Regular 184
12.3. Bahasa Reguler 185
12.4. Sifat Bahasa Reguler 187
12.5. Hubungan Ekspresi Regular dan Finite State Automata. 188
12.6. Latihan soal 194

BAB XIII
PUSHDOWN AUTOMATA (PDA) 197
13.1. Pendahuluan 197
13.2. Komponen Kerja PDA 197
13.3. PDA untuk Suatu Tata Bahasa Bebas Konteks 211
13.4. Deskripsi Seketika Pada PDA 215
13.5. Latihan soal 216

BAB XIV
MESIN TURING 218
14.1. Pendahuluan 218
14.2. Komponen Kerja Mesin Turing 219
14.3. Mekanisme Kerja Mesin Turing 221
14.4. Deskripsi Seketika Mesin Turing 233
14.5. Latihan Soal 234

DAFTAR PUSTAKA 238

INDEKS 239

TENTANG PENULIS 244

RINGKASAN

Bidang ilmu komputer meliputi topik yang luas, dari perancangan mesin sampai pemrograman. Untuk mempelajari prinsip-prinsip dasar tentang komputasi, kita mengkonstruksi suatu mesin automata sebagai model abstrak dari komputer dan komputasi. Model ini memiliki fungsi-fungsi yang penting dan umum pada perangkat keras dan perangkat lunak komputer. Meskipun model tersebut sederhana untuk diterapkan langsung pada dunia nyata, keuntungan yang diperoleh dari mempelajarinya adalah memberikan landasan untuk basis dari pengembangan algoritma. Pendekatan ini juga diterapkan pada ilmu sains lainnya.

Teori Bahasa dan Automata merupakan bagian dari teori komputasi pada ilmu komputer. Beberapa teori komputasi datang dari bahasa dan rekayasa sistem, terutama yang berbasiskan matematika. Ilmu komputer memiliki 2 komponen utama yaitu: pertama, model dan gagasan mendasar mengenai komputasi, Kedua, teknik rekayasa untuk perancangan sistem komputasi, meliputi perangkat keras dan perangkat lunak, khususnya penerapan rancangan dari teori. Teori bahasa dan Automata merupakan bagian pertama. Secara teoritis ilmu komputer diawali dari sejumlah berbeda disiplin ilmu: ahli biologi mempelajari *Neural network*, insinyur elektro mengembangkan *switching* sebagai *tool* untuk mengembangkan *hardware,* matematikawan bekerja mendasarkan logika dan ahli bahasa menyelidiki tata bahasa untuk *natural language.*

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

Otomata merupakan suatu sistem yang terdiri atas sejumlah berhingga *state*, di mana *state* menyatakan informasi mengenai *input*. Otomata juga dianggap sebagai mesin otomatis (bukan mesin fisik) yang merupakan suatu model matematika dari suatu sistem yang menerima input dan menghasilkan output, serta terdiri dari sejumlah berhingga *state*. Hubungan di antara bahasa dan otomata adalah bahasa dijadikan sebagai input oleh suatu mesin otomata, selanjutnya mesin otomata akan membuat keputusan yang mengindikasikan apakah input itu diterima atau tidak.

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

RENCANA KEGIATAN PEMBELAJARAN SEMESTER (RKPS)

PROFIL MATAKULIAH

1. Identitas Mata Kuliah

Nama Mata Kuliah	:	Teori Bahasa & Otomata
Kode / Mata Kuliah	:	TSKK370253
SKS	:	3
Semester	:	Ganjil
Jenis	:	MK Wajib
Jam pelaksanaan	:	Tatap muka di kelas = 3 jam per minggu
		Tutorial / responsi = 1 jam per minggu
Semester / Tingkat	:	7 (Tujuh) / 4 (Empat)
Pre-requisite	:	Matematika Diskrit (Teori Himpunan),
Co-requisite	:	-
Bidang Kajian	:	Algorithm

2. Tujuan Pembelajaran (Kompetensi *Hardskill*):

a. Kompetensi Utama

Setelah mengikuti perkuliahan, mahasiswa dapat menyebutkan dan menjelaskan jenis-jenis tata bahasa formal. Mahasiswa juga diharapkan dapat menerapkan beberapa tata bahasa formal kedalam pengaplikasian suatu mesin abstrak seperti otomata, *pushdown* otomata dan mesin turing.

b. Kompetensi Khusus (*Hardskill / Softskill*)

Setelah mengikuti matakuliah ini mahasiswa diharapkan memiliki kompetensi secara *hardskill* dan *softskill* berikut:

Hardskill

1. Mahasiswa memahami konsep dan istilah yang umum digunakan dalam ototama dan bahasa formal.
2. Mahasiswa memahami tipe-tipe bahasa yang termasuk kedalam hierarki Chomsky dan menganalisa tipe-tipe bahasa tersebut

3. Mahasiswa memahami hubungan bahasa formal dan proses kompilasi
4. Mahasiswa memahami Automata Hingga dari suatu bahasa dan jenis-jenis Otomata Hingga dan konsep ekivalensi antar jenis tersebut
5. Mahasiswa dapat memahami tentang Bentuk Normal Chomsky (BNC)
6. Mahasiswa dapat memahami ekspresi regular, ekspresi regular dan Finite Otomata
7. Mahasiswa dapat memahami penggunaan mesin abstrak lain seperti *pushdow*n Automata, mesin Turing, dan meley mesin.

Softskill

a) Inovatif
b) Mandiri
c) Bertanggungjawab untuk setiap tugas yang dibuat, artinya sembarangan dalam mengerjakan suatu tugas

3. Jumlah Jam:

Kegiatan Di dalam kelas		: 3 SKS x 14 Pertemuan
a.	Kuliah & diskusi kelompok	: 33 SKS (14 Pertemuan)
b.	*Mid Term*	: 3 SKS
c.	*Final Term*	: 3 SKS
	Total	: 42 SKS (14 Pertemuan)

4. Buku Referensi

1. Brookshear, Glen J., “Theory of Computation: Formal Language, Automata and Complexity”, The Benjamin/Cummings Publishing Company, 1989
2. Heriyanto, Bambang, Teori Bahasa, Otomata dan Komputasi serta terapannya, Informatika Bandung,2003
3. Hopcroft, Motwani, Ullman: *Introduction to Automata Theory , Languages, and Computation* . Addison-Wesley, 2001

4. Hopcroft, Motwani, Ullman. Introduction to Automata Theory, Languages, and Computation. Addison Wesley, 2007
5. John E. Hopcroft dkk. Teori Bahasa dan Otomata, (terjemahan, Edisi 2, 2007
6. James A. Anderson, Automata Theory with Modern Applications, Cambridge University Press, 2006
7. Linz Peter, "An Introduction to Formal Languages and Automata 5th Edition, Jones & Bartlett Publishers", ISBN: 144961552X, 9781449615529, 2011
8. Martin, J.C.: Introduction to Languages and the Theory of Computation.McGraw– Hill (2002)
9. Munir, Rinaldi, Diktat Kuliah Matematika Informatika(Teori bahasa Formal dan Otomata), ITB
10. Revesz, Gyorgy E., "Introduction to Formal Languages", McGraw Hill Book Company, 1985
11. S. Greibach. Formal languages: origins and directions. Ann. Hist. Comput., 3, 1981
12. Salomaa, A.: Jewels of Formal Language Theory. Computer Science Press (1981)
13. Sipser, M, "Introduction to the Theory of Computation", Cengage Learning, 2012
14. Utdirartatmo, Firrar, "Teknik Kompilasi", J&J Learning Yogyakarta, ISBN: 979-9398-11-8, 2001
15. Utdirartatmo, Firrar, Teori Bahasa dan Otomata, Penerbit Graha Ilmu, Jogyakarta, 2005

5. Jadwal Kegiatan Mingguan

Pertemuan ke	Kemampuan Akhir yang diharapkan	Bagan Kajian (Materi Ajar)	Bentuk /Metode /Strategi Pembelajaran	Kriteria Penilaian (Indikator)
1	Mahasiswa dapat: 1. Mengetahui pengertian Teori Bahasa dan Otomata 2. Mengetahui tujuan mata kuliah Teori bahasa dan Otomata 3. Mengetahui manfaat mempelajari teori	- Pendahuluan - Pengertian bahasa formal, *alphabet*, *String*, dan bahasa	- Ceramah - Diskusi	Mampu memahami Teori Bahasa dan Otomata, Tujuan mata kuliah Teori bahasa dan Otomata, sertai manfaat mempelajari teori bahasa dan otomata

Pertemuan ke	Kemampuan Akhir yang diharapkan	Bagan Kajian (Materi Ajar)	Bentuk /Metode /Strategi Pembelajaran	Kriteria Penilaian (Indikator)
	bahasa dan otomata			
2	Mampu menjelaskan pengertian dasar symbol, operasi dasar *string* dan sifat-sifat operasi bahasa formal	- Pengertian dasar symbol - Operasi dasar *string* - Sifat-sifat operasi bahasa formal	- Ceramah - Diskusi - Tugas/latihan soal	Mampu memahami konsep teori bahasa, operasi bahasa dan sifat operasi bahasa formal
3	Mampu menjelaskan Pengertian tata bahasa, aturan produksi tata bahasa, hierarki dan klasifikasi tata bahasa	- Pengertian tata bahasa - Aturan produksi - Hierarki dan klasifikasi	- Ceramah - Diskusi - Tugas/latihan soal	Dapat memahami aturan produksi tata bahasa, hierarki dan klasifikasi tata bahasa
4	- Mampu menjelaskan konsep dan penerapan otomata berhingga - Mampu mengubah dari NDFA ke DFA	- Konsep FA - Jenis-jenis FA - Sifat-sifat FA - DFA dan NDFA	- Ceramah - Diskusi - Tugas/latihan soal	Dapat memahami, Menjelaskan dan Menggambarkan otomata berhingga dan jenis-jenisnya
5	Mampu menjelaskan otomata berhingga yang mempunyai output	- Mealey mesin - Moore mesin	- Ceramah - Diskusi - Tugas/latihan soal	Dapat memahami otomata berhingga yang mempunyai output, serta ekuivalensi antara mealey mesin dan *moore* mesin
6	Mampu menjelaskan cara menguraikan tata bahasa bebas konteks (CFG) (pohon Urai), Pumping Lemma untuk bahasa bebas konteks, dan Sifat-sifat tertutup bahasa bebas konteks	- Penguraian CFG - Pumping lemma untuk CFG - Sifat-sifat tertutup CFG	- Ceramah - Diskusi - Tugas/latihan soal	Dapat memahami cara menguraikan Tata bahasa bebas konteks (CFG) (pohon Urai), Pumping Lemma untuk bahasa bebas konteks, dan Sifat-sifat tertutup bahasa bebas konteks
7	Mampu menjelaskan Bentuk Normal Chomscky	Pengertian, Pembentukan dan hierarki Bentuk Normal Chomscky	- Ceramah - Diskusi - Tugas/latihan soal	Memahami Konsep Bentuk Normal Chomscky
8	Mampu menjelaskan aturan produksi dan tahapan penghilangan rekursif	Aturan produksi rekursif dan tahapan penghilangan rekursif	- Ceramah - Diskusi - Tugas/latihan soal	Mahasiswa mampu memahami aturan produksi rekursif dan tahapan penghilangan rekursif
9	Mampu menjelaskan aturan produksi Useless, produksi unit dan produksi nul	aturan produksi Usless, produksi unit dan produksi null	- Ceramah - Diskusi - Tugas/latihan soal	Mahasiswa mampu memahami aturan produksi Usless, produksi unit dan produksi nul
10	Mampu menjelaskan Bentuk Normal Greibach, Pembentukan *Bentuk Normal Greibach* dengan Substitusi, dan Pembentukan *Bentuk Normal Greibach* dengan	- Pengertian Bentuk Normal Greibach, - Pembentukan *Bentuk Normal Greibach* dengan Substitusi, dan dengan matrik	- Ceramah - Diskusi - Tugas/latihan soal	Mahasiswa mampu memahami Bentuk Normal Greibach, Pembentukan *Bentuk Normal Greibach* dengan Substitusi, dan

Pertemuan ke	Kemampuan Akhir yang diharapkan	Bagan Kajian (Materi Ajar)	Bentuk /Metode /Strategi Pembelajaran	Kriteria Penilaian (Indikator)
	matrik			Pembentukan *Bentuk Normal Greibach* dengan matrik
11	Mampu menjelaskan konstruksi aturan produksi otomata berhingga untuk tata bahasa regular, konversi DFA ke Tata bahasa regular atau sebaliknya	- Mengkonstruksi aturan produksi Otomata Hingga untuk tata bahasa regular - Tata bahasa Linier kiri dan kanan - Konversi DFA ke tata bahasa linier atau sebaliknya	- Ceramah - Diskusi - Tugas/latihan soal	Mahasiswa mampu memahami konstruksi aturan produksi otomata berhingga untuk tata bahasa regular, konversi DFA ke Tata bahasa regular atau sebaliknya
12	- Mampu menjelaskan arti definisi dan konsep - Kemampuan untuk melakukan operasi terhadap beberapa FA, membentuk tata bahasa yang lebih kompleks, membangun dan mengidentifikasi ekspresi regular, serta melakukan konversi antara ekspresi regular dengan FA.	- Operasi FA dan Regular *Expression*	- Ceramah - Diskusi - Tugas/latihan soal - Kuis	- Mahasiswa mampu melakukan operasi terhadap beberapa FA, membentuk tata bahasa yang lebih kompleks, membangun dan mengidentifikasi ekspresi regular, serta melakukan konversi antara ekspresi regular dengan FA.
13	- Pemahaman terhadap *Pushdown* Automata (PDA), kelebihan-kelebihan dan karakteristik PDA, representasi PDA, dan penggunaan PDA sebagai penerima bahasa. - Pemahaman terhadap *Context Free Grammar* (CFG), definisi dan karakteristiknya, serta kemampuan untuk membuat dan mengurai *string* bahasa yang dibangun oleh CFG. - Pemahaman terhadap hubungan antara CFG dengan PDA dan mengonversikan CFG menjadi PDA ataupun sebaliknya.	- *Pushdown* Automata (PDA) - *Context Free Grammar* (CFG) - PDA vs. CFG	- Ceramah - Diskusi - Tugas/latihan soal	- Mahasiswa memahami *Pushdown* Automata (PDA), kelebihan-kelebihan dan karakteristik PDA, representasi PDA, dan penggunaan PDA sebagai penerima bahasa. - Mahasiswa memahami *Context Free Grammar* (CFG), definisi dan karakteristiknya, serta mampu membuat dan mengurai *string* bahasa yang dibangun oleh CFG. - Mahasiswa memahami hubungan antara CFG dengan

Pertemuan ke	Kemampuan Akhir yang diharapkan	Bagan Kajian (Materi Ajar)	Bentuk /Metode /Strategi Pembelajaran	Kriteria Penilaian (Indikator)
				PDA dan mengonversikan CFG menjadi PDA ataupun sebaliknya.
14	- Pemahaman terhadap konsep **deterministik** mesin sebagai mesin *unrestricted grammar*, konsep dan definisi formal dari *Turing Machine*, diagram transisinya, dan pembangunan *Turing Machine*. - Kemampuan untuk mengembangkan *Turing Machine* sebagai mesin *unrestricted grammar* dan mengkombinasikan beberapa *Turing Machines*. - Kemampuan untuk memodelkan *Turing Machine* ke dalam notasi block untuk membangun kombinasi *Turing Machines*.	- *Turing Machine* - *Combining Turing Machines* - *Basic Building Block*	- Ceramah - Diskusi - Tugas/latihan soal - Kuis	- Mahasiswa memahami konsep deterministik mesin sebagai mesin unrestricted grammar, konsep dan definisi formal dari Turing Machine, diagram transisinya, dan pembangunan Turing Machine. - Mahasiswa mampu mengembangkan Turing Machine sebagai mesin unrestricted grammar dan mengkombinasikan beberapa Turing Machines. - Mahasiswa mampu memodelkan Turing Machine ke dalam notasi block untuk membangun kombinasi Turing Machines.

BAB I
BAHASA FORMAL

1.1. Pendahuluan

Gambar 1.1. Komponen Bahasa Formal

Bahasa adalah suatu sistem yang meliputi pengekspresian gagasan, fakta, konsep, termasuk sekumpulan simbol-simbol dan aturan untuk melakukan manipulasinya. Bahasa bisa juga disebut sebagai rangkaian simbol-simbol yang mempunyai makna. Bahasa formal adalah kumpulan kalimat dimana kalimat dalam sebuah bahasa dibangkitkan oleh sebuah tata bahasa (*grammar*) yang sama. Teori bahasa formal menjadi bagian dari matematika diskrit yang memiliki koneksi dengan ilmu yang lain seperti aljabar, logika, kombinatorik dan komputability. Teori bahasa formal (*formal language*) merupakan landasan ilmu komputer teoritis. Namun, asal dan motivasinya berasal dari sumber yang berbeda:

1. *Switching* sirkuit sebagai model untuk insinyur listrik.
2. *Grammar* sebagai model untuk struktur bahasa alami.
3. Model untuk fenomena biologis:
   Jaringan saraf yang mengarah pada otomata. *Lindenmayer* sistem sebagai model untuk pertumbuhan organisme.
4. Model di bagian yang berbeda dari teori bahasa pemrograman: *parsing*, kompilasi, pengeditan teks.
5. Model untuk pertanyaan matematis (dan filosofis) dari komputasi.

Daftar di atas juga menyediakan contoh penerapan bahasa formal. Area aplikasi baru lainnya adalah *kriptografi* dan grafik komputer.

1.2. Konsep Bahasa Formal

Teori bahasa membicarakan bahasa formal (*formal language*), terutama untuk kepentingan perancangan kompilator (*compiler*) dan pemroses naskah (*text processor*). Sebuah bahasa formal bisa dibangkitkan oleh dua atau lebih tata bahasa berbeda. Dikatakan bahasa formal karena *grammar* diciptakan mendahului pembangkitan setiap kalimatnya. Bahasa manusia bersifat sebaliknya; *grammar* diciptakan untuk meresmikan kata-kata yang hidup di masyarakat.

Bidang teori bahasa formal mempelajari aspek-aspek sintaksis murni bahasa-bahasa yaitu pola struktural internal. Teori bahasa formal muncul dari linguistik, sebagai cara memahami keteraturan sintaksis bahasa alami. Dalam ilmu komputer, bahasa formal sering digunakan sebagai dasar untuk mendefinisikan bahasa pemrograman dan sistem lain di mana kata-kata bahasa dikaitkan dengan makna atau semantik tertentu.

Sementara teori bahasa formal biasanya memperhatikan dirinya sendiri dengan bahasa formal yang dijelaskan oleh beberapa aturan sintaksis, seperti rangkaian *string* panjang hingga (mungkin tak terbatas). Dalam praktiknya, ada banyak bahasa yang dapat dijelaskan oleh aturan, seperti bahasa reguler atau bahasa bebas konteks. Gagasan tata bahasa formal mungkin lebih dekat dengan konsep intuitif "bahasa", yang dijelaskan oleh aturan-aturan sintaksis.

1.2.1. Abjad atau Alfabet (*Alpabet*)

Sebuah alfabet, dalam konteks bahasa formal, dapat berupa apa saja, meskipun sering menggunakan alfabet sebagai set karakter seperti ASCII. Abjad juga bisa tidak terbatas; misalnya logika orde pertama sering diekspresikan menggunakan alfabet yang selain simbol seperti ^, ¬, dan tanda kurung, mengandung banyak elemen x0, x1, x2, ... yang memainkan peran variabel. Unsur-unsur alfabet disebut huruf-hurufnya.

Abjad yang dilambangkan dengan simbol, adalah himpunan berhingga tak kosong dari simbol-simbol.

Contoh

Alfabet biner adalah	= {0, 1}
Alfabet huruf kecil adalah	= {a, b, c, … , z}
Alfabet bilangan asli < 9 adalah	= {1, 2, 3, .., 8}

1.2.2. Untai atau kata (*String*)

Sebuah kata yang terbentuk dari kumpulan alfabet dapat berupa urutan, atau *string*, huruf apa pun yang terbatas. Himpunan semua kata di atas alfabet Σ biasanya dilambangkan dengan Σ* (menggunakan bintang Kleene). Untuk alfabet apa pun hanya ada satu kata dengan panjang 0, kata kosong, yang sering dilambangkan dengan Λ, ε, Δ atau λ. Dengan penggabungan satu atau dua kata untuk membentuk kata baru, yang panjangnya adalah jumlah dari panjang kata-kata asli. Hasil penggabungan kata dengan kata kosong adalah kata aslinya. Untai, kadang-kadang disebut kata atau *word*, adalah barisan berhingga simbol-simbol yang berasal dari suatu alfabet.

Sebagai contoh, 1011 adalah untai yang berasal dari alfabet ∑ = {0, 1}. stmik, bumigora, kelas, mataram, adalah untai yang berasal dari alfabet ∑ = {a, b, c, … , z} 200929001 adalah untai yang berasal dari alfabet = **{**1, 2, 3, 4, … , 9**}**

Sebuah kata (atau *string*) adalah urutan item yang terbatas, yang disebut simbol atau huruf yang dipilih dari himpunan terbatas tertentu yang disebut alfabet. Contoh alfabet umum adalah huruf dalam alfabet (+ ruang interword, tanda baca, dll.), dan bit 0 dan 1. Satu kata dengan panjang satu diidentifikasi dengan simbol satu-satunya. Kata khusus adalah kata kosong (atau kata nol) yang tidak memiliki simbol, dilambangkan dengan Λ (atau λ atau ε atau Δ).

Panjang kata w adalah jumlah simbol di dalamnya, dilambangkan dengan | w |. Panjang kata kosong adalah 0. Jika ada simbol k dalam alfabet, maka ada kata-kata kn panjang n. Jadi ada

$$\sum_{i=0}^{n} k^i = \frac{k^{n+1} - 1}{K - 1}$$

Kata-kata panjang paling banyak n, jika k > 1, dan n + 1 kata, jika k = 1. Himpunan semua kata adalah tak terbatas, yaitu, dapat diberikan sebagai daftar tak terbatas.

Operasi dasar kata-kata adalah gabungan, yaitu menulis kata sebagai suatu gabungan. Penggabungan kata w1 dan w2 dilambangkan hanya dengan w1w2. Contoh penggabungan dalam alfabet {a, b, c}:

w_1	=	aacbba,	w_2 =	caac,	w_1w_2	=	aacbbacaac
w_1	=	aacbba,	w_2 =	Λ,	w_1w_2	=	w_1 = aacbba
w_1	=	λ,	w_2 =	caac,	w_1w_2	=	w_2 = caac

Penggabungan adalah asosiatif, yaitu, $w_1(w_2w_3) = (w_1w_2)w_3$. Sebagai konsekuensi dari ini, penggabungan berulang dapat ditulis tanpa tanda kurung. Di sisi lain, penggabungan biasanya tidak komutatif, $w_1w_2 \neq w_2w_1$, dalam kasus penggabungan aljabar uniter jelas komutatif. The nth (*concatenation*) dari kata w adalah w^n = ww…w ➔ n copy.
Terutama $w^1 = w$ dan $w^0 = \Lambda$, dan selalu $\Lambda^n = \Lambda$.

The mirror image (atau pembalikan) dari kata $w = a_1a_2 \bullet \bullet \bullet a_n$ adalah kata $\hat{w} = a_n \bullet \bullet \bullet a_2a_1$, Terutama $\Lambda = \Lambda$. Jelas memiliki $\widehat{w_1w_2} = \hat{w}_2\hat{w}_1$. Kata u adalah prefix (*resp. suffix*) dari kata w, jika w = uv (resp. w = vu) untuk beberapa kata v. Sebuah kata u adalah sub-kata (atau segmen) dari kata w, jika $w = v_1uv_2$ untuk beberapa kata v_1 dan v_2. Kata u adalah sub kata yang tersebar dari kata w, jika $w = w_1u_1w_2u_2 \bullet \bullet \bullet w_nu_nw_{n+1}$

Dimana $u = u_1u_2 \bullet \bullet \bullet u_n$, untuk beberapa n dan beberapa kata w_1, w_2,. . . , w_{n+1} dan u_1, u_2,. . . , u_n.

1.2.2.1. Untai Kosong

Untai kosong adalah untai yang tidak mempunyai simbol yang berasal dari alfabet. Untai kosong (*null string*) dilambangkan dengan λ, ε, Δ. adalah untai yang berasal dari sembarang alphabet

1.2.2.2. Panjang Untai

Panjang untai adalah jumlah simbol yang membentuknya.

Contoh

Panjang untai 1011, ditulis |1011| = 4
Panjang untai sttrqw, , ditulis | sttrqw | = 6
Panjang untai λ , ditulis | λ | = 0

1.2.2.3. Pangkat Alfabet

Himpunan seluruh untai yang berasal dari alfabet tertentu dapat dinyatakan dalam notasi eksponensial. $\sum^k$ didefinisikan sebagai himpunan untai dengan panjang k, yg masing-masing simbolnya berasal dari $\sum$.

1. $\sum^*$ (*Kleen Star*) didefinisikan sebagai himpunan seluruh untai mulai dari untai kosong sampai untai dengan panjang tertentu.
   $\sum^* = \sum^0 + \sum^1 + \sum^2 + \sum^3 + \ldots = \sum^0 \cup \sum^1 \cup \sum^2 \cup \sum^3 \cup \ldots$
2. $\sum^+$ didefinisikan sebagai himpunan seluruh untai tanpa untai kosong (*null string*)
   $\sum^+ = \sum^0 + \sum^1 + \sum^2 + \sum^3 + \ldots = \sum^0 \cup \sum^1 \cup \sum^2 \cup \sum^3 \cup \ldots$

Contoh

Jika $\sum = \{0, 1\}$, maka:

$\sum 0 = \{ \lambda \}$

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

$\sum 2 = \{00, 01, 10, 11\}$

$\sum 3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

$\sum * = \{\lambda , 0, 1, 00, 01, 10, 11, 000, \ldots\}$

$\sum + = \{0, 1, 00, 01, 10, 11, 000, \ldots\}$

Perbedaan antara $\sum$ dan $\sum^k$ adalah: $\sum$ adalah abjad yang merupakan himpunan tak kosong yang terdiri dari simbol-simbol. $\sum^k$ adalah himpunan untai dengan panjang masing-masing untai adalah k. Simbol dari setiap untai pada $\sum^k$ berasal dari $\sum$

1.2.2.4. Rentengan Untai (*String Concatenation*).

Misal w1 dan w2 adalah untai. Rentengan untai w1 dan w2 menghasilkan untai w1 w2

Contoh

w1 = xx w2 = xyx, maka w1 w2 = xxxyx

w1 = aab w2 = abb, maka w1 w2 = aababb

w1 = λ w2 = xy, maka w1 w2 = xy

w1 = abba w2 =λ, maka w1 w2 = abba

w1 = λ w2 =λ, maka w1 w2 = λ

1.2.3. Bahasa

Bahasa adalah kumpulan kata-kata dari beberapa alfabet. Seringkali bahasa tunggal {w} diidentifikasi dengan satu-satunya kata w, dan bahasa dilambangkan hanya oleh w. Seperangkat notasi digunakan untuk bahasa: $\subseteq$ (inklusi), $\subset$ (penyertaan yang benar), $\cup$ (union), $\cap$ (*intersectin*), - (*difference*) dan $^-$ (pelengkap terhadap himpunan semua kata di atas alfabet). Milik kata w dalam bahasa L dilambangkan dengan $w \in L$, seperti biasa. Perhatikan juga "*negated*" relasi $\not\subseteq$, $\not\subset$ and $\notin$.

Bahasa semua kata di atas alfabet Σ, khususnya Λ, dilambangkan dengan Σ^*. Bahasa dari semua kata tidak kosong di atas alfabet Σ dilambangkan dengan Σ^+. Jadi $L = \Sigma^* - L$ dan $\Sigma^+ = \Sigma^* - \{\Lambda\}$. Ada jumlah bahasa yang tak terbatas pada alfabet mana pun, sehingga bahasa tidak dapat diberikan dalam daftar tak terbatas.

Semua bahasa (lebih dari beberapa abjad Σ) muncul di daftar L_1, L_2,. . . kemudian mendefinisikan bahasa L sebagai berikut: Biarkan w_1, w_2,. . . menjadi daftar semua kata di atas alfabet Σ. Kata wi berada dalam bahasa L jika dan hanya jika bukan dalam bahasa L_i. Jelas bahwa bahasa L bukan salah satu bahasa dalam daftar L_1, L_2,. . . Hipotesis *counter* dengan demikian salah, dan teorema itu benar.

Metode pembuktian di atas adalah turunan dari apa yang disebut metode diagonal. Hanya boleh ada sejumlah cara yang tidak terbatas dalam mendefinisikan bahasa, karena semua definisi semacam itu harus dapat dikenali dalam beberapa bahasa alami, dan dengan demikian mudah dibaca dalam urutan leksikografi. Dalam teori bahasa formal mendefinisikan bahasa dan menyelidiki bahasa melalui definisi mereka adalah yang terpenting. Jadi hanya sebagian kecil dari semua kemungkinan bahasa yang masuk ke penyelidikan.

Bahasa formal adalah himpunan yang mempunyai anggota-anggota berasal dari $\sum^*$. Jika $\sum$ adalah alfabet, dan $L \subseteq \sum^*$ maka L adalah bahasa. Untai-untai yang membentuk suatu bahasa berasal dari suatu alphabet $\sum$.

1.3. Bahasa Alami dan Bahasa Formal

Bahasa menurut kamus Websters adalah *"the body of words and methods of combining words used and understood by a considerable community"*, sedangkan menurut Kamus Besar Bahasa Indonesia (KBBI), bahasa adalah:

1) Sistem lambang bunyi yang arbitrer, yang digunakan oleh anggota suatu masyarakat untuk bekerja sama, berinteraksi, dan mengidentifikasikan diri.
2) Percakapan (perkataan) yang baik, tingkah laku yang baik, sopan santun, perkataan-perkataan yang dipakai oleh suatu bangsa.

Definisi di atas mengungkapkan bahwa suatu bahasa adalah kalimat atau perkataan. Kalimat dalam sebuah bahasa dibentuk dengan menggabungkan satu atau lebih kata-kata. Perhatian dalam pembentukan kalimat hanya tertuju pada sintaks atau bentuk kalimat, dan bukan pada semantik atau makna kalimatnya. Bahasa komunikasi yang digunakan oleh manusia, yaitu bahasa ucap seperti bahasa Indonesia, Inggris, Jerman, Spanyol dan sebagainya disebut sebagai bahasa alami atau bahasa natural (natural language). Sintaks bahasa alami sangat rumit dan tidak mungkin untuk menspesifikasikan semua aturan sintaksnya. Bahasa yang kaidah sintaksnya dapat dispesifikasikan secara matematis dengan baik disebut bahasa formal. Bahasa formal dapat didefinisikan secara abstrak sebagai "sistem matematik". Kaidah sintaks dalam teori bahasa formal tidak hanya bermanfaat untuk studi linguistik bahasa alami seperti penerjemahan secara otomatis dari suatu bahasa ke bahasa lain, tetapi juga berguna untuk studi bahasa pemrograman.

1.4. Latihan soal

Latihan 1

Bahasa pemrograman C++, atau Java, atau lainnya termasuk bahasa formal yang berasal dari alfabet

∑= {a, b, c, … , z, A, B, C, …, Z, 0, 1, 2, 9,< , >, =, +, – , *, / , (,) , . , & , ! , % , ^ , { , } , | , ‘ ,: , ;}

Latihan 2

Berikut adalah beberapa contoh lain bahasa formal.

a. Himpunan seluruh untai yang terdiri dari n buah 0 dan diikuti oleh n buah 1, untuk $n \geq 0$ adalah L = { λ, 01, 0011, 000111, …}
b. Himpunan seluruh untai terdiri dari jumlah simbol 0 dan 1 yang sama adalah: L = { λ, 01, 0011, 0101, 1010, 1100, 0110, …}
c. Himpunan bilangan biner yang nilainya sama dengan bilangan prima adalah: L = {10, 11, 101, 111, 1011, …}
d. * adalah bahasa atas seluruh alphabet ∑
e. Θ adalah bahasa kosong merupakan bahasa atas sembarang alfabet
f. { λ } adalah bahasa yang hanya terdiri dari untai kosong, juga merupakan bahasa atas sembarang alfabet.

Perhatikan bahwa:

$$\Theta \neq \{ \lambda \}$$

Θ tidak memiliki untai
{ λ } memiliki satu untai

Bahasa dapat didefinisikan dengan menggunakan notasi pembentuk himpunan (*set-builder notation*).
L = {w| sifat-sifat w}, Dibaca: L adalah himpunan untai w sedemikian rupa, sehingga memenuhi sifat-sifat w.

Latihan 3

a) L = {w | w terdiri dari simbol-simbol 0 dan 1 yang jumlahnya sama}
b) L = { w| w adalah bilangan bulat biner yang nilainya prima}
c) L = {w| w adalah program C++ yang benar sintaksnya}

BAB II
OPERASI BAHASA FORMAL

2.1. Pendahuluan

Ada banyak operasi bahasa lain selain set-theoretic. Rangkaian bahasa L_1 dan L_2 adalah $L_1L_2 = \{w_1w_2 \mid w_1 \in L_1 \text{ dan } w_2 \in L_2\}$. n^{th} (*concatenation*) dari bahasa L adalah $L^n = \{w_1w_2 \bullet \bullet \bullet w_n \mid w_1, w_2, \ldots, w_n \in L\}$, dan terutama $L^1 = L$ dan $L^0 = \{\Lambda\}$. Khususnya $\emptyset^0 = \{\Lambda\}$! Penutupan rentetan atau bintang Kleenean dari bahasa L adalah $L^+ = \bigcup_{n=1}^{\infty} L^n$ yaitu, himpunan yang diperoleh dengan menggabungkan kata-kata L dalam semua cara yang mungkin, termasuk pemberian gabungan "kosong" Λ. Demikian pula $L^+ = \bigcup_{n=1}^{\infty} L^n$, berisi kata kosong Λ hanya jika sudah di L. (Cf. Σ^* dan Σ^+ di atas.) Jadi $\emptyset^* = \{\Lambda\}$, tetapi $\emptyset^+ = \emptyset$. Perhatikan bahwa sebenarnya $L^+ = L * L = LL^*$. Quotients kiri dan kanan dari bahasa L_1 dan L_2 didefinisikan sebagai $L_1 \setminus L_2 = \{w_2 \mid w_1w_2 \in L_2 \text{ untuk beberapa kata } w_1 \in L_1\}$ (hapus dari kata-kata prefiks L2 yang termasuk dalam L1 dalam semua cara yang mungkin) dan $L_1 / L_2 = \{w_1 \mid w_1w_2 \in L_1 \text{ untuk beberapa kata } w_2 \in L_2\}$ (hapus dari kata-kata L_1 *suffixes* yang termasuk dalam L_2 dalam semua kemungkinan cara). Perhatikan bahwa awalan atau *suffix* bisa kosong. Citra cermin (atau pembalikan) dari bahasa $\acute{L} = \{\hat{w} \mid w \in L\}$. Ada dua mesin fundamental dalam mendefinisikan bahasa: tata bahasa, kata-kata bahasa yang *generate*, dan automata, yang mengenali kata-kata bahasa. Ada banyak cara lain dalam mendefinisikan bahasa, misalnya mendefinisikan bahasa reguler menggunakan ekspresi reguler.

2.2. Pengertian Dasar Operasi

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

b. *String* adalah deretan terbatas (*finite*) simbol-simbol. Sebagai contoh, jika *a*, *b*, dan *c* adalah tiga buah simbol maka *abcb* adalah sebuah *string* yang dibangun dari ketiga simbol tersebut.

c. Jika *w* adalah sebuah *string* maka panjang string dinyatakan sebagai $|w|$ dan didefinisikan sebagai cacahan (banyaknya) simbol yang menyusun *string* tersebut. Sebagai contoh, jika *w* = *abcb* maka $|w| = 4$.

d. *String* hampa adalah sebuah *string* dengan nol buah simbol. *String* hampa dinyatakan dengan simbol ε (atau ^) sehingga $|\varepsilon| = 0$. *String* hampa dapat dipandang sebagai simbol hampa karena keduanya tersusun dari nol buah simbol.

e. Alfabet adalah himpunan hingga (*finite set*) simbol-simbol

Bahasa formal adalah himpunan yang mempunyai anggota-anggota berasal dari $\sum^*$. Jika $\sum$ adalah alfabet, dan $L \subseteq \sum^*$ maka L adalah bahasa. Untai-untai yang membentuk suatu bahasa berasal dari suatu alfabet $\sum$.

Contoh 1.

Bahasa pemrograman C++, atau Java, atau lainnya termasuk bahasa formal yang berasal dari alfabet

$\sum$= {a, b, c, … , z, A, B, C, …, Z, 0, 1, 2, 9,< , >, =, +, – , *, / , (,) , . , & , ! , % , ^ , { , } , | , ‘ ,: , ;}

2.3. Teori Dasar Operasi

2.3.1. Operasi dasar *String*

Def 1. Sebuah *string* dengan panjang n yang dibentuk dari himpunan A adalah barisan dari n symbol, $a_1a_2...a_n$ $a_i \in A$, Panjang *string* x dituliskan dengan |x|

Def 2. *String* kosong (*null string*), dilambangkan dengan ε adalah untaian dengan panjang 0 dan tidak berisi apapun. Panjang *string* x dituliskan dengan |x|

Def 3. Dua buah *string* $a = a_1a_2...a_m$ dan $b=b_1b_2...b_n$ dapat disambungkan menjadi string c dengan panjang m+n sebagai berikut $c = a_1a_2...a_mb_1b_2...b_n$

Operasi penyambungan tersebut dapat pula diterapkan pada himpunan $Z=XY = \{st \mid s \in X \wedge t \in Y\}$

Def 4. (*Closure*) . A^n adalah himpunan *string* dengan panjang n yang dibentuk dari simbol-simbol di himpunan simbol/alfabet A:

Transitif *Closure/Kleen Closure* adalah himpunan seluruh *string* yang dapat dibentuk dari A dengan berbagai panjang, $A^* = A^0 \cup A^1 \cup A^2 \cup A^3 \cup ...$

Jika *string* kosong dikeluarkan, akan diperoleh *positive closure*, $A^+ = A^1 \cup A^2 \cup A^3 \cup ...$

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

a) Prefik *string w* adalah *string* yang dihasilkan dari *string w* dengan menghilangkan *nol* atau lebih simbol-simbol paling belakang dari *string w* tersebut.

   Contoh: *abc*, *ab*, *a*, dan ε adalah semua Prefix(*x*)

b) *ProperPrefix string w* adalah *string* yang dihasilkan dari *string w* dengan menghilangkan *satu* atau lebih simbol-simbol paling belakang dari *string w* tersebut.

   Contoh: *ab*, *a*, dan ε adalah semua *ProperPrefix*(*x*)

c) *Postfix* (atau *Sufix*) *string w* adalah *string* yang dihasilkan dari *string w* dengan menghilangkan *nol* atau lebih simbol-simbol paling depan dari *string w* tersebut.

   Contoh: *abc*, *bc*, *c*, dan ε adalah semua *Postfix*(*x*)

d) *ProperPostfix* (atau *PoperSufix*) *string w* adalah *string* yang dihasilkan dari *string w* dengan menghilangkan *satu* atau lebih simbol-simbol paling depan dari *string w* tersebut.

   Contoh: *bc*, *c*, dan ε adalah semua *ProperPostfix*(*x*)

e) *Head string w* adalah simbol paling depan dari *string w*. Contoh: *a* adalah *Head*(*x*)

f) Tail *string w* adalah *string* yang dihasilkan dari *string w* dengan menghilangkan simbol paling depan dari *string w* tersebut. Contoh: *bc* adalah Tail(*x*)

g) *Substring string w* adalah *string* yang dihasilkan dari *string w* dengan menghilangkan *nol* atau lebih simbol-simbol paling depan

dan/atau simbol-simbol paling belakang dari *string w* tersebut. Contoh: *abc*, *ab*, *bc*, *a*, *b*, *c*, dan ε adalah semua *Substring*(*x*)

h) *ProperSubstring string w* adalah *string* yang dihasilkan dari *string w* dengan menghilangkan *satu* atau lebih simbol-simbol paling depan dan/atau simbol-simbol paling belakang dari *string w* tersebut. Contoh: *ab*, *bc*, *a*, *b*, *c*, dan ε adalah semua *Substring*(*x*)

i) *Subsequence string w* adalah *string* yang dihasilkan dari *string w* dengan menghilangkan *nol* atau lebih simbol-simbol dari *string w* tersebut.

Contoh: *abc*, *ab*, *bc*, *ac*, *a*, *b*, *c*, dan ε adalah semua *Subsequence*(*x*)

j) *ProperSubsequence string w* adalah *string* yang dihasilkan dari *string w* dengan menghilangkan *satu* atau lebih simbol-simbol dari *string w* tersebut.

Contoh: *ab*, *bc*, *ac*, *a*, *b*, *c*, dan ε adalah semua *Subsequence*(*x*)

k) *Concatenation* adalah penyambungan dua buah string. Operator *concatenation* adalah *concate* atau tanpa lambang apapun. Contoh: *concate*(*xy*) = *xy* = *abc123*

l) *Alternation* adalah pilihan satu di antara dua buah *string*. Operator *alternation* adalah *alternate* atau $|$. Contoh: *alternate*(*xy*) = $x|y$ = *abc* atau *123*

m) *Kleene Closure*: $x^* = \varepsilon | x | xx | xxx | \ldots = \varepsilon | x | x^2 | x^3 | \ldots$

n) *Positive Closure*: $x^+ = x | xx | xxx | \ldots = x | x^2 | x^3 | \ldots$

2.3.2. Beberapa Sifat Operasi

a) Tidak selalu berlaku: x = Prefix(x)Postfix(x)

b) Selalu berlaku: x = Head(x)Tail(x)

c) Tidak selalu berlaku: Prefix(x) = Postfix(x) atau Prefix(x) ≠ Postfix(x)

d) Selalu berlaku: ProperPrefix(x) ≠ ProperPostfix(x)

e) Selalu berlaku: Head(x) ≠ Tail(x)

f) Setiap Prefix(x), ProperPrefix(x), Postfix(x), ProperPostfix(x), Head(x), dan Tail(x) adalah Substring(x), tetapi tidak sebaliknya

g) Setiap Substring(x) adalah Subsequence(x), tetapi tidak sebaliknya

h) Dua sifat aljabar *concatenation*:
   - Operasi *concatenation* bersifat asosiatif: x(yz) = (xy)z
   - Elemen identitas operasi *concatenation* adalah ε: $\varepsilon x = x\varepsilon = x$

i) Tiga sifat aljabar *alternation*:
   - Operasi *alternation* bersifat komutatif: x | y = y | x
   - Operasi *alternation* bersifat asosiatif: x | (y | z) = (x | y) | z
   - Elemen identitas operasi *alternation* adalah dirinya sendiri: x | x = x

j) Sifat distributif *concatenation* terhadap *alternation*: x (y | z) = xy | xz

k) Beberapa kesamaan:
   - Kesamaan ke-1: (x*)* = (x*)
   - Kesamaan ke-2: $\varepsilon \mid x^+ = x^+ \mid \varepsilon = x^*$
   - Kesamaan ke-3: (x | y)* = ε | x | y | xx | yy | xy | yx | … = semua *string* yang merupakan *concatenation* dari nol atau lebih x, y, atau keduanya.

2.3.3. Perangkaian (*Concatenation*)

Misal L1 dan L2 merupakan bahasa-bahasa berdasarkan alfabet ∑. Perangkaian L1 dan L2 ditulis: L1 . L_2 = {$w_1.w_2$ | $w_1 \in L_1$ dan $w_2 \in L_2$ }

Contoh

Diketahui, L_1 = {ayam, kucing} dan L_2 = {gajah}, Maka L_1 . L_2 = {ayamgajah, kucinggajah}.

2.3.4. Eksponensiasil (*Exponentiation*)

Misalkan L merupakan suatu bahasa berdasarkan alphabet $\sum$

Contoh

Jika L = {ab} berdasarkan alphabet $\sum$, maka:

$L^o = \{ \lambda \}$

$L^1 = L \quad = \{ab\}$

$L^2 = L.L^1 \quad = \{abab\}$

$L^3 = L.L^2 \quad = \{ababab\}$

2.3.5. Gabungan (*Union*)

Misalkan L_1 dan L_2 adalah bahasa-bahasa berdasarkan suatu abjad. Gabungan dari L_1 dan L_2, ditulis $L_1 \cup L_2$, terdiri dari semua untai yang muncul sekurang-kurangnya sekali dalam L_1 dan L_2 .

$$L_1 \cup L_2 = \{x|x\ L_1 \text{ atau } x\ L_2\}$$

Contoh

Misal $\sum = \{0, 1\}$

$L_1 \quad = \{\varepsilon, 0, 1, 10, 11\}$

$L_2 \quad = \{\varepsilon , 1, 0110, 11010\}$

$L_1 \cup L_2 = \{ , 0, 1, 10, 11, 0110, 11010\}$

2.3.6. Irisan *(Intersection*)

Misalkan L_1 dan L_2 adalah bahasa-bahasa berdasarkan suatu abjad. Irisan dari L_1 dan L_2, ditulis $L_1 \cap L_2$, terdiri dari semua untai yang muncul baik di L_1 maupun di L_2.

$$L_1 \cap L_2 = \{x|x\ L_1 \text{ dan } x\ L_2\}$$

Contoh:

Misal $\sum = \{0, 1\}$

$L_1 \quad = \{\varepsilon , 0, 1, 10, 11\}$

$L_2 \quad = \{\varepsilon , 1, 0110, 11010\}$

$L_1 \cap L_2 = \{\varepsilon , 1\}$

2.3.7. Sub Bahasa

Misalkan L_1 dan L_2 adalah bahasa-bahasa berdasarkan suatu abjad dan jika semua untai di L_1 juga merupakan untai di L_2, maka L_1 disebut sebuah sub bahasa dari L_2 . Notasi $L_1 \subseteq L_2$

Contoh

Jika L_1 = {a,aa,aaa} dan L_2 = {a,aa,aaa,aaaa,aaaaa} maka $L_1 \subseteq L_2$

2.3.8. Equal (Sama)

Dua buah bahasa L_1 dan L_2 dikatakan sama jika kedua bahasa tersebut secara persis mempunyai untai-untai yang sama, artinya jika sebagai himpunan-himpunan keduanya persis sama.

Notasi: $L_1 = L_2$

2.3.9. Star Closure dan Plus Closure

Jika L adalah sebuah bahasa berdasarkan suatu abjad ∑, didefinisikan:

1. Star Closure dari L*

$$L^* = U_{\grave{n}=0} L^n = L^0 + L^1 + L^2 + L^3 + ,\ldots\ldots\ldots\ldots\ldots$$

2. Plus Closure dari L^+

$$L^+ = U_{\grave{n}=0} L^n = L^0 + L^1 + L^2 + L^3 + ,\ldots\ldots\ldots\ldots\ldots$$

2.4. Latihan soal

Latihan 1:

Tentukan nilai dari operasi berikut: a + cb*

Jawab

b* = ε, b, bb, bbb, bbbb, bbbbb, ….

cb* = c, cb, cbb, cbbb, cbbbb, cbbbbb, …

Maka, a+ cb* = a, c, cb, cbb, cbbb, cbbbb, cbbbbb, …

Latihan 2:

Tentukan nilai dari operasi berikut: a (b+c)

Jawab:

= a.b + a.c

= ab, ac

Latihan 3:

Tentukan nilai dari operasi berikut: (ab)* a

Jawab

(ab)* = ε, ab, abab, ababab, …

(ab)* a = a, aba, ababa, abababa

Latihan 4:

Tentukan nilai dari operasi berikut: (a+b)* a

Jawab

(a + b)* =

$(a+b)^0$ = ε

$(a+b)^1$ = a,b

$(a+b)^2$ = (a+b) (a+b) =aa, ab, ba, bb

$(a+b)^3$ = (a+b) (a+b) (a+b) = aaa, aab, aba, abb, baa, bab, bba, bbb,…

$(a+b)^n$

Maka

(a+b)* a = a, aa, ba, aaa, aba, baa, bba, aaaa, aaba, abaa, abba, baaa, baba, bbaa, bbba, …..

Latihan 5:

Tentukan nilai dari operasi berikut: (a+c) b*

Jawab:

b^* = ε, b, bb, bbb, bbbb, bbbbb, ……

(a+c) b* = a, c, ab, cb, abb, cbb, abbb, cbbb, …..

Latihan 6:

Tentukan nilai dari operasi berikut: 1 (0+1)* + 10*

Jawab

$(0+1)^*$ ➔

$(0+1)^0$ = ε

$(0+1)^1$ = 0, 1

$(0+1)^2$ = (0+1)(0+1) = 00, 01, 10, 11

$(0+1)^3$ = (0+1)(0+1) (0+1) = 000,001, 010,011, 100, 101, 110, 111

…….

$1(0+1)^*$ = 1, 10,11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111,…

0* = ε, 0, 00, 000, 0000, 00000, ….

10* = 1, 10, 100, 1000, 10000, 100000, …..

Maka

1 (0+1)* + 10* = 1, 10, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, …..

Latihan 7:

(a*| b)* = (a | b)*

Jawab:

(a|b)* = {ε, "a", "b", "aa", "ab", "ba", "bb", "aaa", ...}

Dengan diketahui a*= ε| a| aa| aaa| aaaa| …..,

Sedangkan b* = ε|b|bb|bbb|bbbb|…

Jadi (a|b)* yang merupakan gabungan concate dari a* dan b* maka

(a|b)* =(ε | a| b| aa| ab| ba| bb | aaa| ...)

Dengan diketahui: a*= ε| a| aa| aaa| aaaa| …..,

Maka(a*|b)* = (ε| a| aa| aaa| aaaa| …..|b)*

= (ε| a|b | aa|bb | aaa|bbb | aaaa| bbbb …..)

Maka terbukti, (a*|b)* = (a|b)*

Latihan 8:

(a.b)*+c

Jawab

(a.b)* = (ab)* = Ɛ, ab, abab, ababab, …

Maka (a.b)*+c = Ɛ, ,c, ab, abab, ababab, …

Latihan 9:

0.1*(0+1)

Jawab:

1* = Ɛ, 1, 11, 111, …

0.1* = 01* = 0, 01, 011, 0111, ….

Maka 0.1*(0+1) = 00, 01, 010, 011, 0110, 0111, 01110, ….

Latihan soal Mandiri:

1. a. Jelaskan bagaimana hubungan antara grammar dengan bahasa?
   b. Jelaskan bagaimana hubungan antara otomata dengan bahasa?
2. Jika L = {a, b}

Maka:

$L^0 = \{\lambda\}$

$L^1 = L = \{a, b\}$

$L^2 = L.L^1 = \{aa, ab, ba, bb\}$

$L^3 = L^2.L = \{aaa, aab, aba, abb, baa, bab, aab, bbb\}$

$L^* = L^0 + L^1 + L^2 + L^3 + \ldots$

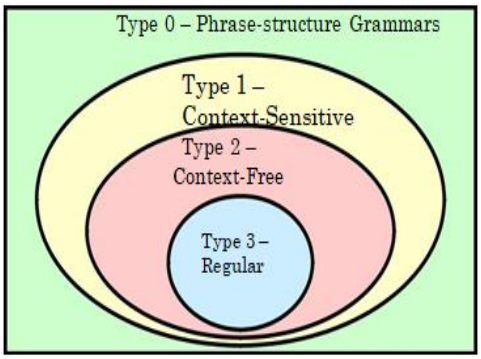
$L^* = \{ , a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa,bab,aab,bbb. \ldots\}$

$L^+ = L^1 + L^2 + L^3 + \ldots$

$L^+ = \{a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa,bab,aab,bbb. \ldots\}$

BAB III
TATA BAHASA

3.1. Pendahuluan



Gambar 3.1. Hierarki Tata Bahasa (*Grammar*)

Tata bahasa (*Grammar*), yang didefinisikan secara formal sebagai kumpulan dari himpunan-himpunan variabel, simbol-simbol terminal, simbol awal yang dibatasi oleh aturan tata bahasa. Suatu tata bahasa dapat menghasilkan sejumlah string dengan menerapkan aturan tata bahasa.

Contoh tata bahasa: $\alpha \rightarrow \beta$ dapat dibaca: α menghasilkan β

Tata bahasa berfungsi untuk menentukan kebenaran dalam penulisan suatu *statement*, sesuai dengan aturan yang terdapat pada suatu program.

Tata bahasa merupakan salah satu bagian penting dalam pembuatan implementasi bahasa formal. Masukan yang tidak sesuai dengan tata bahasa yang telah ditetapkan menyebabkan proses tidak dapat dilakukan. Tata bahasa bebas konteks (*Context Free Grammar*) adalah salah satu yang digunakan dalam implementasi bahasa formal selain tata bahasa regular. Tata bahasa bebas konteks yang digunakan menghasilkan aturan produksi. Aturan produksi ini merupakan pusat dari tata bahasa yang menspesifikasikan bagaimana suatu tata bahasa melakukan transformasi suatu string ke bentuk lain.

3.2. Elemen Bahasa Formal

Elemen-elemen bahasa adalah alphabet, tata bahasa dan semantik.

- **Alphabet** adalah himpunan terhingga dari token-token dimana kalimat dibentuk dalam suatu bahasa. Alphabet merupakan himpunan simbol (karakter) tak kosong yang berhingga, yang digunakan untuk membentuk kata-kata (*string-string*) pada bahasa. Pada beberapa buku, alphabet dilambangkan dengan Σ Istilah huruf, karakter dan simbol adalah *sinonim* menunjukkan elemen alphabet. Jika simbol berbaris bersebelahan, maka diperoleh "*string* simbol". Istilah kalimat, kata dan string adalah *sinonim*
  Contoh: {a,b} → Himpunan yang terdiri dari simbol "a" dan "b".
- **Tata Bahasa** adalah himpunan dari aturan-aturan struktural yang didefinisikan yang berlaku dalam suatu kalimat pada token-token. Grammar G didefinisikan sebagai pasangan 4 tuple: V_t, V_n, S, dan P, dan dituliskan sebagai G(V_t , V_n , S, P), dimana:
  - V_t : himpunan simbol-simbol terminal (alfabet) = kamus
  - V_n : himpunan simbol-simbol non terminal
  - $S \in V$: simbol awal (atau simbol start)
  - P : himpunan produksi

  Contoh:
  1. G1: VT = {I, want, need, You}, V = {S,A,B,C}, P = {S → ABC, A→ I, B→ want | need, C→ You}
     S → ABC
     - IwantYou
     - L(G1)={IwantYou,IneedYou}
  2. G2: VT = {a}, V = {S}, P = {S ® aS | a}
     S → aS → aaS → aaa
     TL(G2) ={an → n ≥ 1-
     L(G2)=,a, aa, aaa, aaaa, …-
- **Semantic** adalah himpunan aturan-aturan yang didefinisikan yang mempunyai efek operasional pada setiap program yang ditulis dalam bahasa apabila ditranslasi dan dieksekusi pada suatu mesin.

Kalimat dalam bahasa inggris dikonstruksi dari himpunan karakter yang terdiri dari huruf, angka, spasi dan tanda-tanda baca. Karakter-karakter ini dibentuk menjadi kata melalui aturan-aturan ejaan dan kamus, dan kemudian kata-kata dibentuk menjadi kalimat berdasarkan aturan-aturan tata bahasa.

Suatu program komputer dapat dikonstruksi dengan cara yang sama dari urutan karakter yang terdapat pada himpunan karakter dari komputer tersebut. Perbedaan yang prinsip antara Bahasa inggris dengan Bahasa pemrograman komputer adalah bahwa aturan-aturan ejaan dan tata bahasa dalam bahasa inggris sangat kompleks dan banyak pengecualian dan keragu-raguan, sementara dalam bahasa pemrograman harus mempunyai struktur yang tepat dan pasti.

Konsep Dasar

a) Anggota alfabet dinamakan simbol terminal.

b) Kalimat adalah deretan hingga simbol-simbol terminal.

c) Bahasa adalah himpunan kalimat-kalimat. Anggota bahasa bisa tak hingga kalimat.

d) Simbol-simbol berikut adalah simbol terminal:
   - Huruf kecil, misalnya: a, b, c
   - , dan * simbol operator, misalnya: +,
   - Simbol tanda baca, misalnya: (,), dan ;
   - *String* yang tercetak tebal, misalnya: **if**, **then**, dan **else**.

e) Simbol-simbol berikut adalah simbol non terminal /Variabel:
   - Huruf besar, misalnya: A, B, C
   - Huruf S sebagai simbol awal
   - *String* yang tercetak miring, misalnya: *expr*
   - Huruf yunani melambangkan string yang tersusun atas simbol-simbol terminal atau simbol-simbol non terminal atau campuran keduanya, misalnya: α,β, dan ε
   - Sebuah produksi dilambangkan sebagai $\alpha \rightarrow \beta$, artinya: dalam sebuah derivasi dapat dilakukan penggantian simbol α dengan simbol β.

- Derivasi adalah proses pembentukan sebuah kalimat atau sentensial. Sebuah derivasi dilambangkan sebagai: $\alpha \rightarrow \beta$.
- Sentensial adalah string yang tersusun atas simbol-simbol terminal atau simbol-simbol non terminal atau campuran keduanya.
- Kalimat adalah string yang tersusun atas simbol-simbol terminal. Kalimat adalah merupakan sentensial, sebaliknya belum tentu.

3.3. Tata Bahasa Regular

Tata bahasa merupakan salah satu bagian penting dalam pembuatan implementasi bahasa formal. Masukan yang tidak sesuai dengan tata bahasa yang telah ditetapkan menyebabkan proses tidak dapat dilakukan. Tata bahasa regular (*regular Grammar*) digunakan dalam implementasi bahasa formal. Tata bahasa regular yang digunakan menghasilkan aturan produksi. Aturan produksi ini merupakan pusat dari tata bahasa yang menspesifikasikan bagaimana suatu tata bahasa melakukan transformasi suatu string ke bentuk lain.

a. Dalam pembicaraan grammar, anggota alfabet dinamakan simbol terminal atau token.
b. Kalimat adalah deretan hingga simbol-simbol terminal.
c. Bahasa adalah himpunan kalimat-kalimat. Anggota bahasa bisa tak hingga kalimat.
d. Simbol-simbol berikut adalah simbol terminal:
   - Huruf kecil awal alfabet, misalnya: a, b, c
   - Simbol operator, misalnya: +, –, dan ×
   - Simbol tanda baca, misalnya: (,), dan ;
   - *String* yang tercetak tebal, misalnya: **if**, **then**, dan **else**.
e. Simbol-simbol berikut adalah simbol non terminal:
   - Huruf besar awal alfabet, misalnya: A, B, C
   - Huruf S sebagai simbol awal
   - *String* yang tercetak miring, misalnya: *expr* dan *stmt*.

f. Huruf besar akhir alfabet melambangkan simbol terminal atau non terminal, misalnya: X, Y, Z.

g. Huruf kecil akhir alfabet melambangkan string yang tersusun atas simbol-simbol terminal, misalnya: x, y, z.

h. Huruf yunani melambangkan string yang tersusun atas simbol-simbol terminal atau simbol-simbol non terminal atau campuran keduanya, misalnya: α, β, dan γ.

i. Sebuah produksi dilambangkan sebagai $\alpha \rightarrow \beta$, artinya: dalam sebuah derivasi dapat dilakukan penggantian simbol α dengan simbol β.

j. Simbol α dalam produksi berbentuk $\alpha \rightarrow \beta$ disebut ruas kiri produksi sedangkan simbol β disebut ruas kanan produksi.

k. Derivasi adalah proses pembentukan sebuah kalimat atau sentensial. Sebuah derivasi dilambangkan sebagai: $\alpha \rightarrow \beta$.

l. Sentensial adalah string yang tersusun atas simbol-simbol terminal atau simbol-simbol non terminal atau campuran keduanya.

m. Kalimat adalah string yang tersusun atas simbol-simbol terminal. Jelaslah bahwa kalimat adalah kasus khusus dari sentensial.

n. Pengertian terminal berasal dari kata *terminate* (berakhir), maksudnya derivasi berakhir jika sentensial yang dihasilkan adalah sebuah kalimat (yang tersusun atas simbol-simbol terminal itu).

o. Pengertian non terminal berasal dari kata *not terminate* (belum/tidak berakhir), maksudnya derivasi belum/tidak berakhir jika sentensial yang dihasilkan mengandung simbol non terminal.

Tata bahasa regular didefinisikan sebagai pasangan 4 tuple: V^T, V^N, S, dan P, dan dituliskan sebagai $G(V^T, V^N, S, P)$, dimana:

V^T : himpunan simbol-simbol terminal (alfabet)

V^N : himpunan simbol-simbol non terminal

$S \in V^N$: simbol awal (atau simbol start)

P : himpunan produksi

Contoh:

1. G_1: V_T = {**I, Love, You**}, V^N = {S,A,B,C},
   P = {S $\rightarrow$ ABC, A$\rightarrow$ **I**, B$\rightarrow$ **Love**, C$\rightarrow$ **You**}
   S $\Rightarrow$ ABC
   $\Rightarrow$ **ILoveYou**

2. G_2: V_T = {a}, V^N = {S}, P = {S $\rightarrow$ aS | a}
   S $\Rightarrow$ aS
   $\Rightarrow$ aaS
   $\Rightarrow$ aaa **L**(G_2) ={a^n | n $\geq$ 1}

3.4. Aturan Produksi

Aturan produksi $\alpha\rightarrow\beta$ yang diterapkan pada suatu string w=aαc mengganti kemunculan. α menjadi β, sehingga string tersebut berubah menjadi w=aβc, sehingga dapat dituliskan aαc ➔ aβc (aαc memproduksi aβc).

Produksi tersebut dapat diterapkan berkali-kali, w_1 ➔ w_2 ➔ w_3 ➔ ... ➔ w_n atau dapat dituliskan w_1 ➔* w_n jika minimal harus ada 1 aturan produksi yang diterapkan: w_1 ➔$^+$ w_n

Contoh 1

Tata bahasa G = {{S} , {a,b}, S , P } dengan aturan produksi P adalah
S $\rightarrow$ aSb
S $\rightarrow$ ε

Maka dapat dihasilkan suatu string: S ➔ aSb ➔ aaSbb ➔aabb sehingga dapat dituliskan: S ➔* aabb

Bahasa yang dihasilkan dari Tata bahasa tersebut adalah: L(G) ={ ε, ab, aabb, aaabbb, aaaabbbb, ... }

Atau dapat pula dituliskan:

$$L(G) = \{a^n b^n \mid n \geq 0 \}$$

Contoh 2

Tata bahasa G = {{S,A} , {a,b}, S , P } dengan aturan produksi P adalah

$S \rightarrow Ab$

$A \rightarrow aAb$

$A \rightarrow \varepsilon$

Maka dapat dihasilkan suatu *string*

S ➔ Ab ➔b

S ➔ Ab ➔ aAbb ➔ abb

S ➔ Ab ➔ aAbb ➔ aaAbbb ➔ aaAbbb

Bahasa yang dihasilkan dari tata bahasa tersebut adalah: L(G) ={b, abb, aabbb, aaabbbb, aaaabbbbb, ... } atau dapat pula dituliskan: L(G) = $\{a^n b^{n+1} \mid n \geq 0\}$

Hierarki Bahasa

Kelas	Mesin Pengenal
Regular language	*Finite State Automata*
Context free language	*Push Down Automata*
Context sensitive language	*Linear Bounded Automata*
Unrestricted language	*Turing Machine*

Kelas	Ruas kiri	Ruas Kanan	Contoh
Regular	$\alpha \in N$	$\leq$ 1 non terminal (paling kiri/kanan)	$P \rightarrow abR$ $Q \rightarrow abc$ $R \rightarrow Scac$
Context free	$\alpha \in N$	-	$P \rightarrow aQb$ $Q \rightarrow abPRS$
Context sensitive	$\alpha \in (T \cup N)^+$	$\lvert\alpha\rvert \leq \lvert\beta\rvert$	$aD \rightarrow Da$ $AD \rightarrow aCD$
Unrestricted	$\alpha \in (T \cup N)^+$	-	$CB \rightarrow DB$ $ADc \rightarrow \varepsilon$

NB: Ruas kiri harus memuat simbol non-terminal

3.5. Derivasi Kalimat dan Penentuan Bahasa

Tentukan bahasa dari masing-masing *grammar* berikut:

1. G^1 dengan Q^1 = {1. S → aAa, 2. A → aAa, 3. A → b}.

<u>Jawab:</u>

Derivasi kalimat terpendek: Derivasi kalimat umum:

S	➔ aAa	(1)	S	➔ aAa	(1)
	➔ aba	(3)		➔ aaAaa	(2)
				…	
				➔ $a^n Aa^n$	(2)
				➔ $a^n ba^n$	(3)

Dari pola kedua kalimat disimpulkan: $L^1(G^1) = \{ a^n ba^n \mid n \geq 1\}$

2. G^2 dengan Q^2 = {1. S → aS, 2. S → aB, 3. B → bC, 4. C → aC, 5. C → a}.

<u>Jawab:</u>

Derivasi kalimat terpendek: Derivasi kalimat umum:

S	➔ aB	(2)	S	➔ aS	(1)
	➔ abC	(3)		…	
	➔ aba	(5)		➔ $a^{n-1}S$	(1)
				➔ $a^n B$	(2)
				➔ $a^n bC$	(3)
				➔ $a^n baC$	(4)
				…	
				➔$a^n ba^{m-1} C$	(4)
				➔ $a^n ba^m$	(5)

Dari pola kedua kalimat disimpulkan: $L^2(G^2) = \{ a^n ba^m \mid n \geq 1, m \geq 1\}$

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

<u>Jawab:</u>

Derivasi kalimat terpendek 1: Derivasi kalimat terpendek 3:

S ➔ abC (2)
➔ abc (4)

Derivasi kalimat terpendek 2:

S ➔ aSBC (1)
➔ aabCBC (2)
➔ aabBCC (5)
➔ aabbCC (3)
➔ aabbcC (4)
➔ aabbcc (6)

S ➔ aSBC (1)
➔ aaSBCBC (1)
➔ aaabCBCBC (2)
➔ aaabBCCBC (5)
➔ aaabBCBCC (5)
➔ aaabBBCCC (5)
➔ aaabbBCCC (3)
➔ aaabbbCCC (3)
➔ aaabbbcCC (4)
➔ aaabbbccC (6)
➔ aaabbbccc (6)

Dari pola ketiga kalimat disimpulkan: $L^3(G^3) = \{a^n b^n c^n \mid n \geq 1\}$

Aturan yang disebutkan pada proses pengenalan dan pembangkitan kalimat. Secara formal, tata bahasa terdiri dari 4 komponen yaitu:

1. Himpunan berhingga, tidak kosong dari simbol-simbol non terminal T^1
2. Himpunan berhingga, dari simbol-simbol non-terminal N
3. Simbol awal $S \in N$, yang merupakan salah satu anggota dari himpunan simbol non-terminal.
4. Himpunan berhingga aturan produksi P yang setiap elemennya dituliskan dalam bentuk:

   $\alpha \rightarrow \beta$, dimana α dan β adalah string yang dibentuk dari himpunan $T \cup N$ dan α harus berisi paling sedikit satu simbol non-terminal.

   Keempat komponen tersebut sering dituliskan sbb: *G = (T,N,S,P)*

   Bahasa yang dihasilkan oleh G ditulis sebagai L(G), yaitu himpunan string yang dapat diturunkan dari simbol awal S dengan menerapkan aturan-aturan produksi yang terdapat pada P.

3.6. Latihan Soal

1. Tentukan sebuah gramar regular untuk bahasa $L^1 = \{ a^n \mid n \geq 1\}$

Jawab:

$Q^1(L^1) = \{S \rightarrow aS \mid a\}$

2. Tentukan sebuah gramar bebas konteks untuk bahasa:

   L^2: himpunan bilangan bulat non negatif ganjil

   Jawab:

   Langkah kunci: digit terakhir bilangan harus ganjil.

   Buat dua buah himpunan bilangan terpisah: genap (G) dan ganjil (J)

   $Q^2(L^2) = \{S \rightarrow J \mid GS \mid JS,\ G \rightarrow 0 \mid 2 \mid 4 \mid 6 \mid 8,\ J \rightarrow 1 \mid 3 \mid 5 \mid 7 \mid 9\}$

3. Tentukan sebuah gramar bebas konteks untuk bahasa:

   L^3 = himpunan semua identifier yang sah menurut bahasa pemrograman Pascal dengan batasan: terdiri dari simbol huruf kecil dan angka, panjang identifier boleh lebih dari 8 karakter

   Jawab:

   Langkah kunci: karakter pertama identifier harus huruf.

   Buat dua buah himpunan bilangan terpisah: huruf (H) dan angka (A)

   $Q^3(L^3) = \{S \rightarrow H \mid HT,\ T \rightarrow AT \mid HT \mid H \mid A,\ H \rightarrow a \mid b \mid c \mid \ldots,\ A \rightarrow 0 \mid 1 \mid 2 \mid \ldots\}$

4. Tentukan gramar bebas konteks untuk bahasa $L^4(G^4) = \{a^n b^m \mid n,m \geq 1,\ n \neq m\}$

   Jawab:

   Langkah kunci: sulit untuk mendefinisikan $L^4(G^4)$ secara langsung. Jalan keluarnya adalah dengan mengingat bahwa $x \neq y$ berarti $x > y$ atau $x < y$.

   $L^4 = L^A \cup L^B$, $L^A = \{a^n b^m \mid n > m \geq 1\}$, $L^B = \{a^n b^m \mid 1 \leq n < m\}$.

   $Q^A(L^A) = \{A \rightarrow aA \mid aC,\ C \rightarrow aCb \mid ab\}$, $Q(L^B) = \{B \rightarrow Bb \mid Db,\ D \rightarrow aDb \mid ab\}$

$Q^4(L^4)$ = {S→ A | B, A → aA | aC, C → aCb | ab, B → Bb | Db, D→ aDb | ab}

5. Tentukan sebuah gramar bebas konteks untuk bahasa:

   L^5 = bilangan bulat non negatif genap. Jika bilangan tersebut terdiri dari dua digit atau lebih maka nol tidak boleh muncul sebagai digit pertama.

   <u>Jawab:</u>

   Langkah kunci: Digit terakhir bilangan harus genap. Digit pertama tidak boleh nol. Buat tiga himpunan terpisah: bilangan genap tanpa nol (G), bilangan genap dengan nol (N), serta bilangan ganjil (J).

   $Q^5(L^5)$ = {S → N | GA | JA, A → N | NA | JA, G→ 2 | 4 | 6 | 8, N→ 0 | 2 | 4 | 6 | 8, J → 1 | 3 | 5 | 7 | 9}

6. G = ({S,A,B,C}, {a,b}, S,Q)

   Aturan produksi:

   S ➔ aS |aB

   B → bC

   C → aC |a

   Kalimat a^3ba^2 atau aaabaa mempunyai derivasi sebagai berikut:

   S ➔aS

   →aaS

   →aaaB

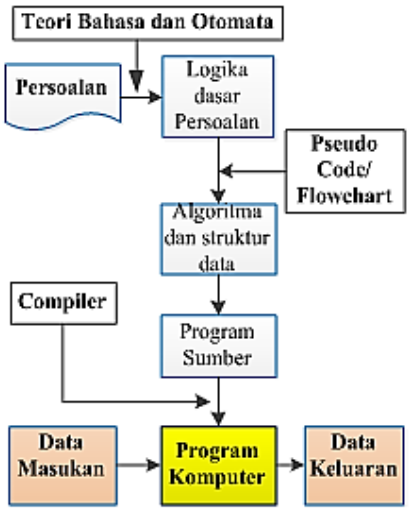
   → aaabC

   →aaabaC

   →aaabaa

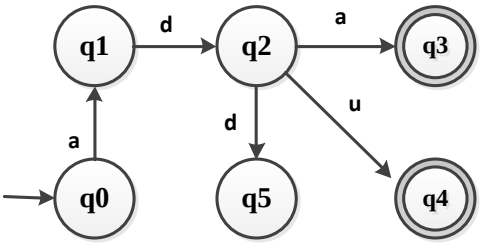
BAB IV
OTOMATA BERHINGGA
(FINITE AUTOMATA)

4.1. Pendahuluan



Gambar 4.1. Model Pedekatan Otomata

Otomata merupakan suatu sistem yang terdiri atas sejumlah berhingga *state*, di mana *state* menyatakan informasi mengenai *input*. Otomata juga dianggap sebagai mesin otomatis (bukan mesin fisik) yang merupakan suatu model matematika dari suatu sistem yang menerima input dan menghasilkan output, serta terdiri dari sejumlah berhingga state. Hubungan di antara bahasa dan otomata adalah bahasa dijadikan sebagai input oleh suatu mesin otomata, selanjutnya mesin otomata akan membuat keputusan yang mengindikasikan apakah input itu diterima atau tidak. Misalnya, jika memiliki sebuah mesin sederhana yang menerima input kata dalam bahasa Indonesia, hal ini bisa dilihat pada gambar 4.2. berikut ini.



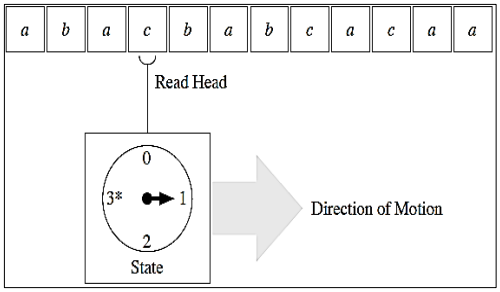
Gambar 4.2. Contoh Diagram FA

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

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

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

4.2. Konsep Otomata (*Automata*)



Gambar 4.3: Ilustrasi Finite Automata.

Automata berasal dari bahasa Yunani automatos, yang berarti sesuatu yang bekerja secara otomatis (mesin). Istilah automaton sebagai bentuk tunggal dan automata sebagai bentuk jamak. Untuk memodelkan hardware dari komputer diperkenalkan model mesin abstrak yaitu otomata. Otomata adalah ilmu yang mempelajari mengenai mesin abstrak, bisa disebut pula adalah suatu model abstrak dari komputer digital yang dapat menerima input secara sekuensial dan dapat

mengeluarkan output. Setiap otomata memiliki beberapa fungsi dasar, dapat membaca input berupa string dari alphabet yang diberikan dari input file. Otomata merupakan suatu sistem yang terdiri dari sejumlah berhingga status, dimana setiap status tersebut menyatakan informasi mengenai input yang lalu, dan dapat pula dianggap sebagai mesin memori. Input pada mesin otomata dianggap sebagai bahasa yang harus dikenali oleh mesin. Disajikan dengan suatu input string, suatu acceptor apakah akan menerima (mengenali) string tersebut atau menolaknya.

Otomata adalah fungsi-fungsi dari komputer digital, yang menerima input, menghasilkan output dan mampu membuat keputusan dalam mentransformasikan input ke output. Otomata merupakan suatu sistem yang terdiri atas sejumlah berhingga state, dimana state menyatakan informasi mengenai input yang lalu dan dapat dianggap sebagai memori mesin. Input pada mesin otomata dianggap sebagai bahasa yang harus dikenali oleh mesin. Selanjutnya, mesin otomata membuat keputusan yang mengindikasikan apakah input itu diterima atau tidak.

Automata adalah mesin abstrak yang dapat mengenali (*recognize*), menerima (*accept*), atau membangkitkan (*generate*) sebuah kalimat dalam bahasa tertentu. Teori Automata adalah teori tentang mesin abstrak yang:

1. Bekerja sekuensial
2. Menerima input
3. Mengeluarkan output

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

a. Fungsi automata sebagai pengenal (*Recognizer*) string-string dari suatu bahasa, dalam hal ini bahasa sebagai masukan dari automata
b. Fungsi automata sebagai pembangkit (*Generator*) string-string dari suatu bahasa, dalam hal ini bahasa sebagai keluaran dari automata Untuk mengenali string-string dari suatu bahasa, akan dimodelkan sebuah automaton yang memiliki komponen sebagai berikut:

1) Pita masukan, yang menyimpan string masukan yang akan dikenali;
2) Kepala pita (*tape head*), untuk membaca/menulis ke pita masukan;
3) *Finite State Controller* (FSC), yang berisi status-status dan aturan-aturan yang mengatur langkah yang dilakukan oleh automaton berdasarkan status setiap saat dan simbol masukan yang sedang dibaca oleh kepala pita;
4) Pengingat (*memory*), untuk tempat penyimpanan dan pemrosesan sementara Automaton pengenal, setelah membaca string masukan dan melakukan langkah-langkah pemrosesan yang diperlukan, akan mengeluarkan keputusan apakah string tersebut dikenali atau tidak.
5) Konfigurasi adalah suatu mekanisme untuk menggambarkan keadaan suatu mesin pengenal , yang terdiri atas:
   - Status FSC
   - Isi pita masukan dan posisi kepala pita
   - Isi pengingat

Mesin pengenal bersifat deterministik bila dalam setiap konfigurasi, hanya ada satu kemungkinan yang dapat dilakukan mesin, jika tidak mesin pengenal bersifat non deterministik.

Otomata yang lebih umum yaitu yang mampu menghasilkan *string output*, dikenal dengan *Tranducer*. Sebagai contoh penggunaan otomata adalah:

1. Mesin Turing.
2. Mesin Karakter
3. Kompiler
4. Mesin Jaja (*Vending Machine*)

Otomata adalah mesin abstrak yang berkaitan dengan teori mesin abstrak, yaitu mesin sekuensial yang menerima input, dan mengeluarkan output, dalam bentuk diskrit.

4.3. Sifat-Sifat Otomata

Otomata adalah suatu mesin sekuensial (otomatis), mempunyai sifat-sifat

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

Jenis- jenis automata:

Jenis	Pita masukan	Arah *Head*	Memori
Finite State	*Read Only*	1 arah	-
Push Down	*Read Only*	1 arah	*stack*
Linear-Bounded	R/W	2 arah	*(bounded)*
Turing Machine	R/W	2 arah	*(unbounded)*

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

1. Pita masukan (input tape) berisi rangkaian simbol (string) yang berasal dari himpunan simbol / alfabet.
2. Setiap kali setelah membaca satu karakter, posisi read head akan berada pada symbol berikutnya.
3. Setiap saat, FSA berada pada status tertentu
4. Banyaknya status yang berlaku bagi FSA adalah berhingga.

Sebuah otomata hingga (jamak: automata keadaan terbatas) adalah mesin abstrak berturut-turut membaca setiap simbol dari string input, dan mengubah statusnya sesuai ke mekanisme kontrol tertentu. Jika mesin, setelah membaca simbol terakhir dari string input, berada di salah satu dari serangkaian status tertentu, maka mesin tersebut dikatakan menerima string input.

Untuk setiap kelas bahasa Chomsky, terdapat sebuah mesin pengenal bahasa. Masing-masing mesin tersebut adalah:

Kelas Bahasa	Mesin Pengenal Bahasa
Unrestricted Grammar (UG)	Mesin Turing (*Turing Machine*), TM
Context Sensitive Grammar (CSG)	*Linear Bounded Automaton*, LBA
Context Free Gammar (CFG)	Automata *Pushdown* (*Pushdown Automata*), PDA
Regular Grammar, RG	Automata Hingga (*Finite Automata*), FA

Catatan:

1. Pengenal bahasa adalah salah satu kemampuan mesin turing.
2. LBA adalah variasi dari Mesin Turing Nondeterministik.
3. Yang akan dibahas dalam kuliah Teori Bahasa dan Automata adalah: TM (sekilas), FA, dan PDA.

4.4. Finite State Automata (FSA)

Finite State Automata/ Otomata berhingga, selanjutnya disebut sebagai FSA, bukanlah mesin fisik tetapi suatu model matematika dari suatu sistem yang menerima *input* dan *output* diskrit. FSA merupakan mesin otomata dari bahasa regular. Suatu FSA memiliki *state* yang banyaknya berhingga, dan dapat berpindah-pindah dari suatu *state* ke *state* lain.

Secara formal FSA dinyatakan oleh 5 tupel atau M=(Q, Σ, δ, S, F), dimana:

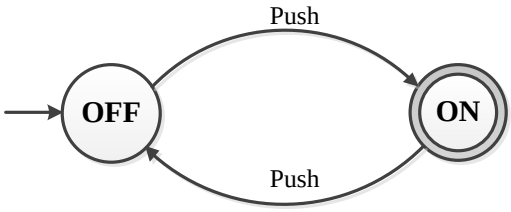
Q = himpunan *state* / kedudukan
Σ = himpunan simbol *input* / masukan / abjad
δ = fungsi transisi
S = *state* awal / kedudukan awal (*initial state*)
F = himpunan *state* akhir

FSA yang memiliki tepat satu *state* berikutnya untuk setiap symbol masukan yang diterima disebut *Deterministic Finite Automata*. Contoh Penerapan Teori Bahasa Otomata, Model switch on/off digambarkan seperti gambar 4.4. berikut:

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

- *Switch* berada dalam *state* "*off*" maka setelah tombol ditekan *state* berubah menjadi "*on*".
- Jika *switch* berada dalam *state* "*on*" maka setelah tombol ditekan *state* berubah menjadi "*off*".

Model pada Gambar 4.4 dapat dipandang sebagai model *finite automato* sederhana.



Gambar 4.4. Contoh Gambar FSA

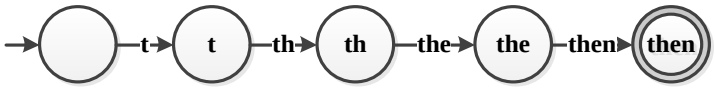
Dalam *finite automata*, ***state*** dinyatakan oleh lingkaran, dan dalam gambar 4.4. *state* diberi nama "*on*" dan *"off"*. ***Arc*** diantara *state* diberi label "input" yang menyatakan pengaruh eksternal pada sistem. Dalam gambar 4.4. kedua *arc* diberilabel '*push*" yang menyatakan *user* menekan tombol tertentu.

Salah satu *state* dinyatakan sebagai ***start state*** atau ***initial state*** yang merupakan *state* dimana sistem berada dalam keadaan awal. Dalam Contoh *start state* adalah *off*. Dalam pembahasan selanjutnya, *start state* ditunjukan oleh kata *start* dan panah menuju *start state* tersebut. Dalam Gambar 4.4. *state* on dinyatakan sebagai ***final*** atau ***accepting*** state.

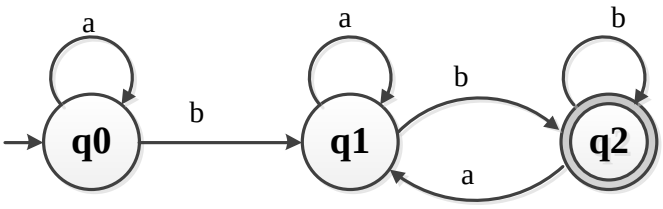
Dalam *state* tersebut, peralatan yang sedang dikontrol oleh *switch* akan beroperasi. Dalam pembahasan selanjutnya, *final State* dinyatakan dalam lingkaran ganda.

Contoh 1:

FA berikut dapat dinyatakan sebagai bagian dari *lexical analyzer*.



Tugas dari FA tersebut adalah mengenali *keyword* “then” sehingga diperlukan lima *state* masing-masing menyatakan posisi yang berbeda dalam kata “then” yang telah dicapai sejauh ini. Posisi ini berhubungan dengan prefix dari kata yang berkisar dari kata string kosong (tidak ada kata yang dikenali sejauh ini) sampai dengan kata lengkap. Dalam Gambar contoh 1, input dinyatakan oleh huruf. *Start state* merupakan string kosong, dan setiap *state* memiliki transisi pada huruf selanjutnya dari kata then ke *state* yang menyatakan prefix selanjutnya yang lebih besar. *State* yang diberi nama “then” dimasuki ketika input mengeja kata “then”. Karena fungsi dari model dalam Gambar contoh 1 adalah mengenali kata then, maka *state* “then” dinyatakan sebagai *accepting state*.
Contoh lain, sebuah otomata seperti pada gambar di bawah ini.



Konfigurasi *Deterministic Finite Automata* di atas secara formal dinyatakan sebagai berikut.

Q = {q0,q1, q2 }
Σ = {a,b}
S = q0
F = { q 2 }

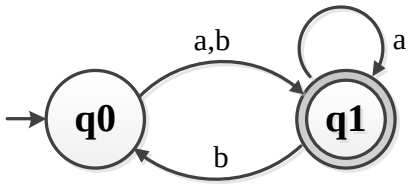
Fungsi transisi yang ada sebagai berikut.

δ(q 0 , a) = q 0
δ (q 0 , b) = q1
δ (q1 , a) = q1
δ (q1 , b) = q 2
δ (q 2 , a) = q1
δ (q 2 , b) = q 2

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

δ	**a**	**b**
q0	q0	q1
q1	q1	q2
q2	q1	q2

Contoh lain bisa dilihat pada gambar di bawah ini.



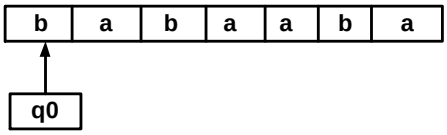
Tabel transisi dari gambar di atas adalah sebagai berikut

δ	**a**	**b**
q0	q1	q1
q1	q1	q0

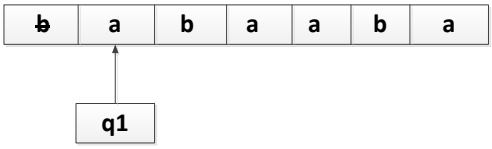
- Model FA diatas menerima input seperti *string*:
  - aaba
  - aaabb
  - baba
  - babaaba
- Berikut *string* yang ditolak oleh FA diatas:
  - aaabbb
  - bbaab
  - babbab
  - ababab

Mekanisme kerja dapat diaplikasikan pada: *elevator, text editor,* analisa leksikal, pencek *parity*.

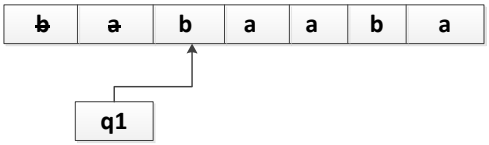
Contoh string
Input: **babaaba**
Proses:
Langkah 1



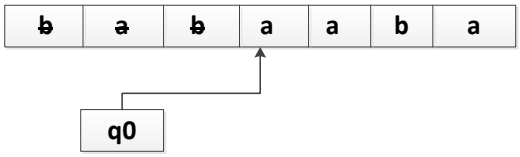
Langkah 2.



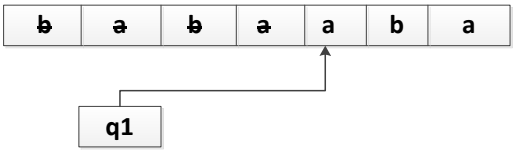
Langkah 3



Langkah 4



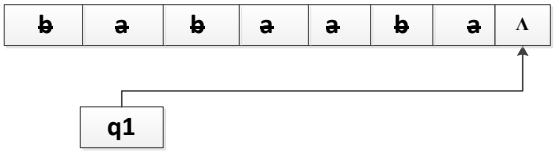
Langkah 5



Langkah 6

Langkah 7

Langkah 8



Ouput : diterima (karena semua symbol habis terbaca dan proses sampai ke state akhir)

Contoh lain,

Q = {Genap, Ganjil}

Σ = {0,1}

S = Genap

F = {Ganjil}

δ	0	1
Genap	Genap	Ganjil
Ganjil	Ganjil	Genap

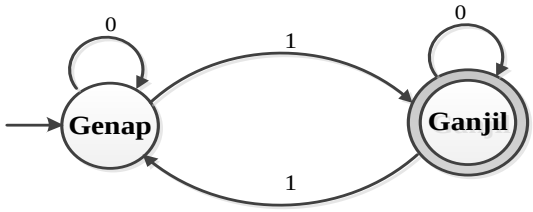
atau

δ(Genap,0) = Genap

δ(Genap,1) = Ganjil

δ(Ganjil,0) = Ganjil

δ(Ganjil,1) = Genap



Misal input: 1101

δ(Genap,~~1~~101)= δ(ganjil,~~11~~01)= δ(genap,~~110~~1)= δ(Ganjil,~~1101~~)

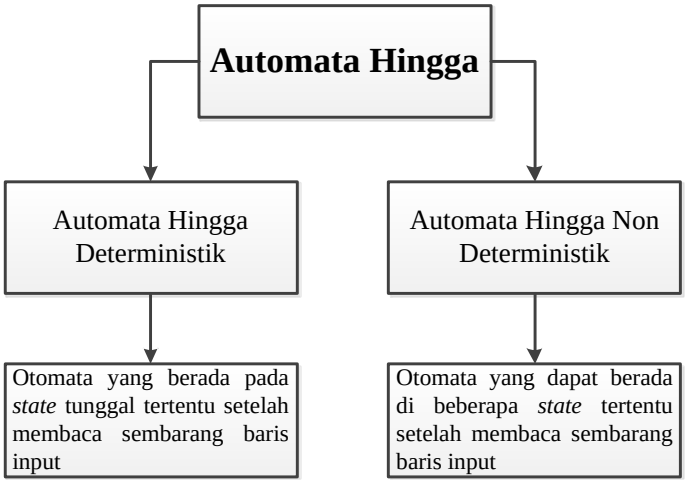
Genap 1 Ganjil 1 Genap 0 Genap 1 Ganjil: **diterima mesin**
(Diterima, karena semua simbol habis terbaca dan proses sampai ke *state* akhir)
Misal input: 1100
Genap 1 Ganjil 1 Genap 0 Genap 0 Genap: **ditolak mesin**

Ada dua jenis FSA:

1. *Deterministic finite automata (*DFA): transisi state FSA akibat pembacaan sebuah simbol bersifat tertentu: $\boldsymbol{\delta: Q \times \sum \rightarrow Q}$
2. *Non deterministik finite automata.*(NFA): transisi state FSA akibat pembacaan sebuah simbol bersifat tak tentu. $\boldsymbol{\delta: Q \times \sum \rightarrow 2^Q}$

Perbedaan DFA dan NDFA

DFA	NDFA
Transisi dari suatu state adalah ke state berikutnya untuk setiap simbol masukan. Oleh karena itu disebut deterministik.	Transisi dari suatu state dapat ke beberapa state berikutnya untuk setiap simbol masukan. Oleh karena itu disebut non-deterministik.
Transisi *string* kosong tidak terlihat di DFA.	NDFA memungkinkan transisi string kosong.
Backtracking diizinkan di DFA	Di NDFA, *backtracking* tidak selalu memungkinkan.
Membutuhkan lebih banyak ruang.	Membutuhkan lebih sedikit ruang.
String diterima oleh DFA, jika transit ke keadaan final.	String diterima oleh NDFA, jika setidaknya salah satu dari semua transisi yang mungkin berakhir dalam a keadaan terakhir.



4.5. Deterministic Finite Automata (DFA)

Deterministic Finite Automata (DFA): dari suatu *state* ada tepat **satu** *state* berikutnya untuk setiap simbol masukan yang diterima.

$$Q = \{q0, q1, q2\}$$

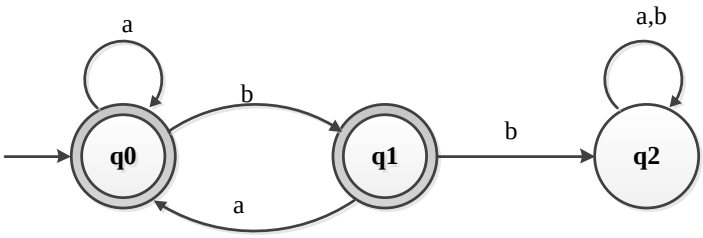
δ diberikan dalam tabel berikut:

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

$S = q0$

$F = \{q0, q1\}$

δ	a	b
q0	q0	q1
q1	q0	q2
q2	q2	q2



Kalimat yang diterima oleh DFA: a, b, aa, ab, ba, aba, bab, abab, baba

Kalimat yang dittolak oleh DFA: bb, abb, abba
DFA ini menerima semua kalimat yang tersusun dari simbol a dan b yang tidak mengandung *substring* bb.

Contoh:
Telusurilah, apakah kalimat-kalimat berikut diterima DFA di atas,
abababaa ➔ diterima
aaaabab ➔ diterima
aaabbaba ➔ ditolak
Jawab:

i) δ (q0, abababaa) ➔ δ (q0, bababaa) ➔ δ (q1, ababaa) ➔ δ (q0, babaa) ➔ δ (q1, abaa) ➔ δ (q0, baa) ➔ δ (q1, aa) ➔ δ (q0, a) ➔ q0
Tracing berakhir di q0 (state AKHIR) ➔ kalimat abababaa diterima

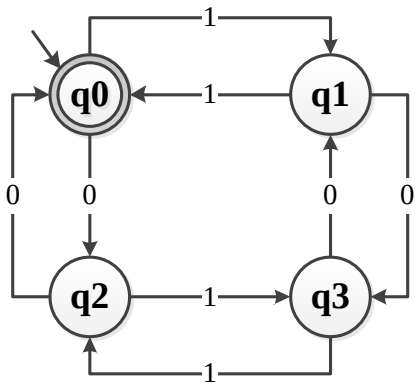
ii) δ (q0, aaaabab) ➔ δ(q0, aaabab) ➔ δ(q0, aabab) ➔ δ (q0, abab) ➔ δ (q0, bab) ➔ δ (q1, ab) ➔ δ (q0 b) ➔ q1 *Tracing* berakhir di q1 (*state* AKHIR) ➔ kalimat aaaababa diterima

iii) δ (q0, aaabbaba) ➔ δ (q0, aabbaba) ➔ δ (q0, abbaba) ➔ δ (q0, bbaba) ➔ δ (q1, bbaba) ➔ δ (q2, baba) ➔ δ (q2, aba) ➔ δ (q2, ba) ➔ δ(q2, a) ➔q2

Tracing berakhir di q2 (bukan *state* AKHIR) ➔ kalimat aaabbaba ditolak. Kesimpulan: sebuah kalimat diterima oleh DFA di atas jika *tracing*nya berakhir di salah satu *state* AKHIR.

Contoh lain:
Pengujian untuk menerima *bit string* dengan banyaknya 0 genap, serta banyaknya 1 genap.

- 0011: diterima.
- 10010: ditolak, karena banyaknya 0 ganjil
- Diagram transisi-nya:



DFA nya

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

Σ = {0,1}

S = q_0

F = $\{ q_0\}$

Fungsi transisi

δ	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Input : 011

Proses : $\delta(q_0, 011)= \delta(q_2,11) =\delta(q_3,1)= q_2$

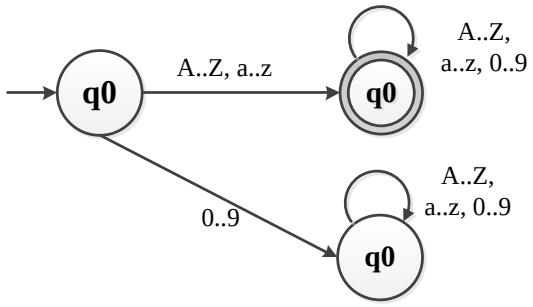
Output : **Ditolak**

Input : 1010

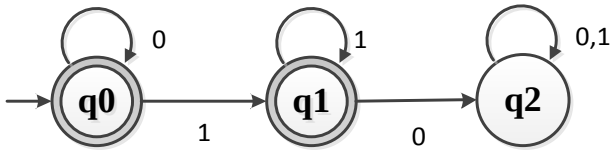
Proses : $\delta(q_0,1010)= \delta(q_1,010) =\delta(q_3,10)=\delta(q_2,0)= q_0$

Output : **Diterima**

Contoh lain DFA: Variabel dalam bahasa pascal diawali oleh huruf (besar/kecil), dan diikuti dengan huruf atau angka.



Contoh DFA lainnya**:**



4.6. Nondeterministic Finite Automata (NDFA)

Non-deterministic Finite Automata (NFA): dari suatu state ada **0, 1 atau lebih** state berikutnya untuk setiap simbol masukan yang diterima. Berikut ini sebuah contoh NFA (Q, ∑, δ, S, F). Dimana:

$$Q = \{q^0, q^1, q^2, q^3, q^4\}$$

δ diberikan dalam tabel berikut:

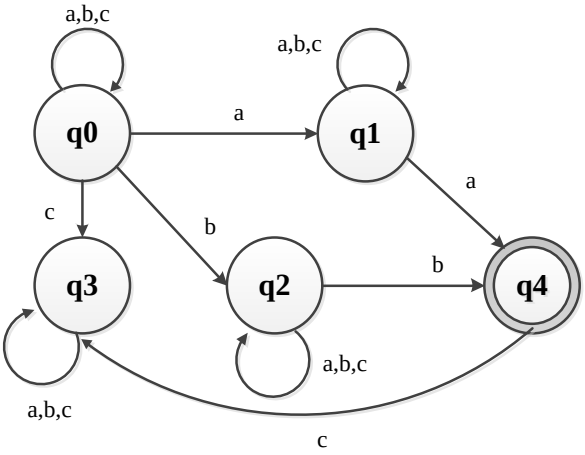
∑= {a, b,c}

S = q^0

F = $\{q^4\}$

δ	a	b	c
Q^0	$\{q^0, q^1\}$	$\{q^0, q^2\}$	$\{q^0, q^3\}$
q^1	$\{q^1, q^4\}$	$\{q^1\}$	$\{q^1\}$
q^2	$\{q^2\}$	$\{q^2, q^4\}$	$\{q^2\}$
q^3	$\{q^3\}$	$\{q^3\}$	$\{q^3, q^4\}$
q^4	∅	∅	∅

Ilustrasi graf untuk NFA adalah sebagai berikut:

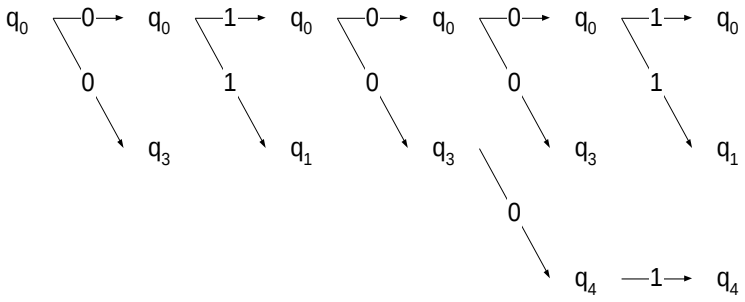


Kalimat yang diterima NFA di atas: aa, bb, cc, aaa, abb, bcc, cbb
Kalimat yang tidak diterima NFA di atas: a, b, c, ab, ba, ac, bc
Sebuah kalimat di terima NFA jika:

- Salah satu *tracing*-nya berakhir di *state* AKHIR, atau
- Himpunan state setelah membaca *string* tersebut mengandung *state* AKHIR
- Perbedaan dengan NFA: fungsi transisi dapat memiliki 0 atau lebih fungsi transisi
- G = ({q_0 , q_1 , q_2 , q_3, q_4 }, {0,1}, δ , q_0 , { q_2 , q_4}}

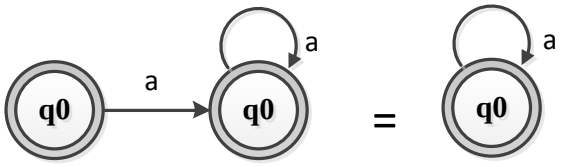
δ	0	1
q_0	{ q_0,q_3}	{q_0,q_1}
q_1	ε	{q_2}
q_2	{q_2}	{q_2}
q_3	{q_4}	ε
q_4	{q_4}	{q_4}

- *String* diterima NFA bila terdapat suatu urutan transisi berdasar input, dari *state* awal ke state akhir.
- Harus mencoba semua kemungkinan.
- Contoh: *string* 01001



Def 2. Dua buah FSA disebut ekuivalen apabila kedua FSA tersebut menerima bahasa yang sama

Contoh: FSA yang menerima bahasa $\{a^n \mid n \geq 0\}$



Def 3. Dua buah state dari FSA disebut *indistinguishable* (tidak dapat dibedakan) apabila:

$\delta(q,w) \in F$ sedangkan $\delta(p,w) \notin F$ dan

$\delta(q,w) \notin F$ sedangkan $\delta(p,w) \in F$ untuk semua $w \in \Sigma^*$

Def 4. Dua buah state dari FSA disebut *distinguishable* (dapat dibedakan) bila terdapat $w \in \Sigma^*$ sedemikian hingga:

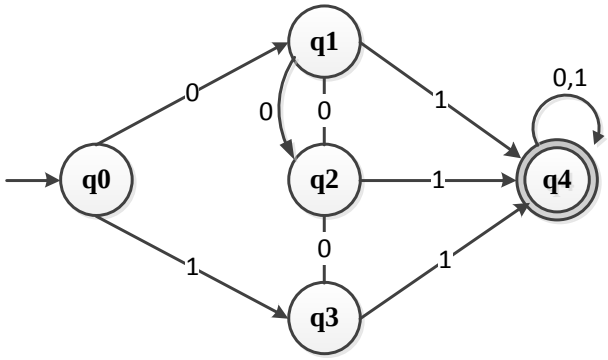
$\delta(q,w) \in F$ sedangkan $\delta(p,w) \notin F$ dan $\delta(q,w) \notin F$ sedangkan $\delta(p,w) \in F$ untuk semua $w \in \Sigma^*$

Prosedur menentukan pasangan status *indistinguishable*

1. Hapus semua *state* yang tak dapat dicapai dari *state* awal.
2. Catat semua pasangan *state* (p,q) yang *distinguishable*, yaitu $\{(p,q) \mid p \in F \wedge q \notin F\}$

3. Untuk setiap pasangan (p,q) sisanya, untuk setiap $a \in \Sigma$, tentukan $\delta(p,a)$ dan $\delta(q,a)$

Contoh



1. Hapus state yang tidak tercapai -> tidak ada
2. Pasangan distinguishable (q_0,q_4), (q_1,q_4), (q_2,q_4), (q_3,q_4).
3. Pasangan sisanya (q_0,q_1), (q_0,q_2), (q_0,q_3), (q_1,q_2) (q_1,q_3) (q_2,q_3)

Pasangan	**State 1**		**State 2**		**Hasil**
	0	**1**	**0**	**1**	
(q_0,q_1)	Q1	Q3	Q2	Q4	*Distinguishable*
(q_0,q_2)	Q1	Q3	Q1	Q4	*Distinguishable*
(q_1,q_2)	Q2	Q4	Q1	Q4	*Indistinguishable*
(q_0,q_3)	Q1	Q3	Q2	Q4	*Distinguishable*
(q_1,q_3)	Q2	Q4	Q2	Q4	*Indistinguishable*
(q_2,q_3)	Q1	Q4	Q2	Q4	*Indistinguishable*

Catatan: jumlah pasangan seluruhnya:

$$C\binom{5}{2} = \frac{5!}{2!3!} = 10$$

Prosedur Reduksi DFA

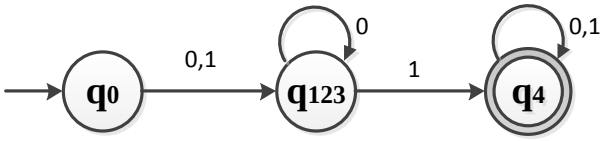
1. Tentukan pasangan status *indistinguishable*.
2. Gabungkan setiap grup *indistinguishable state* ke dalam satu *state* dengan relasi pembentukan grup secara berantai: Jika p dan q

indistingishable dan jika q dan r indistinguishable maka p dan r *indistinguishable*, dan p,q serta r *indistinguishable* semua berada dalam satu grup.

3. Sesuaikan transisi dari dan ke *state-state* gabungan.

   Contoh

   a. Pasangan status indistinguishable (q_1,q_2), (q_1,q_3) dan (q_2,q_3).
   b. Q_1, q_2, q_3 ketiganya dapat digabung dalam satu state q_{123}
   c. Menyesuaikan transisi, sehingga dfa menjadi



4.7. Ekuivalensi NFA-DFA

Tahapan Pengubahan *Non-Deterministic Finite* Automata ke *Deterministic Finite* Automata. Dari sebuah mesin *Non-Deterministic Finite* Automata dapat dibuat mesin DFA yang ekivalen. Ekivalen artinya mampu menerima bahasa yang sama. Suatu DFA dapat dipandang sebagai kasus khusus (subset) dari NFA.

- Jelas bahwa kelas bahasa yang diterima oleh NDFA juga akan diterima oleh DFA
- Namun ternyata DFA juga dapat menyimulasikan NFA; yaitu untuk setiap NFA kita dapat membuat DFA yang ekivalen
- Dapat dibuktikan bahwa DFA dan NFA adalah ekivalen, sehingga dapat disebut FA saja

Simulasi NFA oleh DFA

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

Algoritma:

1. Buat semua *state* yang merupakan subset dari *state* semula. jumlah state menjadi $\mathbf{2^Q}$
2. Telusuri transisi *state–state* yang baru terbentuk, dari diagram transisi.
3. Tentukan state awal: $\{q_0\}$
4. Tentukan *state* akhir adalah *state* yang elemennya mengandung *state* akhir.
5. Reduksi *state* yang tak tercapai oleh *state* awal.

Contoh

Konfigurasi mesin:

Q = ({ q0 }, { q1})

S = { 0, 1 }

S = q0

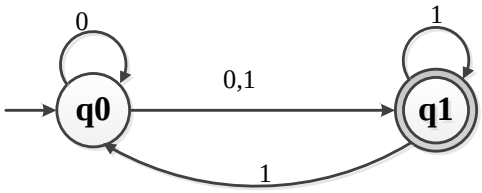
F = q1

Tabel Transisi

δ	0	1
q0	{q0,q1}	Q1
q1	Φ	{q0,q1}

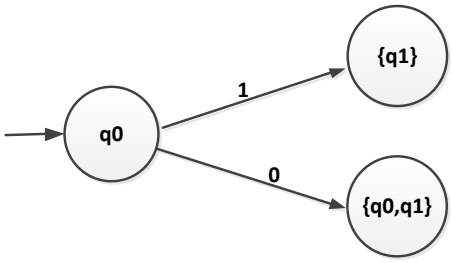
Diagram transisi:

Mesin NFA



Telusuri setiap state yang ada dimulai dari { q0 }:

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

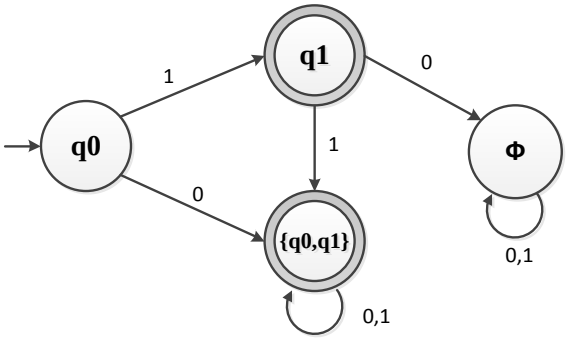


- *State* { q1 }
  - *State* { q1 } bila memperoleh input 0 menjadi *state* f
  - *State* { q1 } bila memperoleh input 1 menjadi *state* { q0, q1 }
- *State* { q0, q1 } bila memperoleh input 0 menjadi *state* { q0, q1 } diperoleh dari δ (q0, 0) = { q0, q1 } δ({ q0, q1}, 0) = { q0, q1 } δ(q1, 0) = Φ
- *State* { q0, q1 } bila memperoleh input 1 menjadi { q0, q1 } diporoleh dari δ(q0, 1) = { q1 } digabung

δ (q1, 1) = { q0, q1 }

δ ({ q0, q1 }, q1) = { q0, q1}

Hasil setelah penelusuran { q1 }, { q0 }, { q0, q1 }
δ(Φ, 0) = Φ
δ(Φ, 1) = δ
F = ({ q1 }, { q0, q1 })



Hasilnya adalah mesin DFA

Konfigurasi Mesin

Q = (q0, { q0, q1 }, q1, f)

∑ = { 1,0 }

S = q0

F = (q1, { q0, q1 })

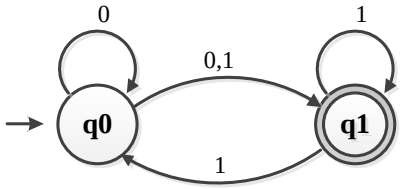
Tabel transisi:

q0	0	1
q0	{ q0, q1 }	q1
q1	Φ	{ q0, q1 }
{ q0, q1 }	{ q0, q1 }	{ q0, q1 }
Φ	Φ	Φ

Contoh Ubahlah NFA berikut menjadi DFA

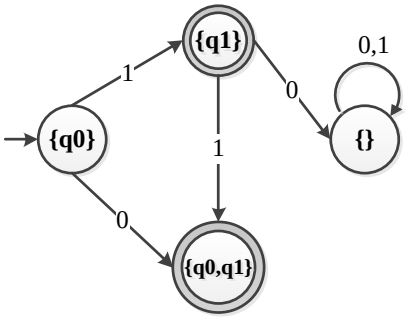
M={{q_0,q_1}, {0,1}, δ, q_0,{q_1}} dengan tabel transisi

δ	0	1
q_0	{q_0,q_1}	{q_1}
q_1	{}	{q_0,q_1}



1. *State* yang akan dibentuk: {}, {q_0} {q_1},{q_0,q_1}
2. Telusuri *state*
3. *State* awal: {q0}
4. *State* akhir yang mengandung q1, yaitu {q1},{q0,q1}

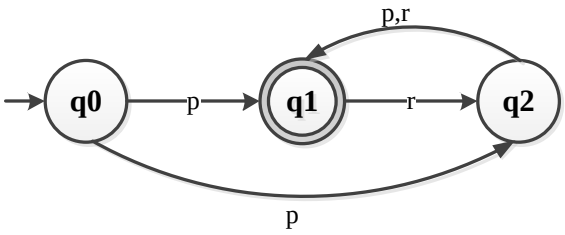
δ	0	1
{}	{}	{}
{q_0}	{q_0,q_1}	{q_1}
{q_1}	{}	{q_0,q_1}
{q_0,q_1}	{q_0,q_1}	{q_0,q_1}



Contoh: Ubahlah NFA berikut menjadi DFA

M={{q_0,q_1 ,q_2}, {p,r}, δ, q_0,{q_1}} dengan tabel transisi

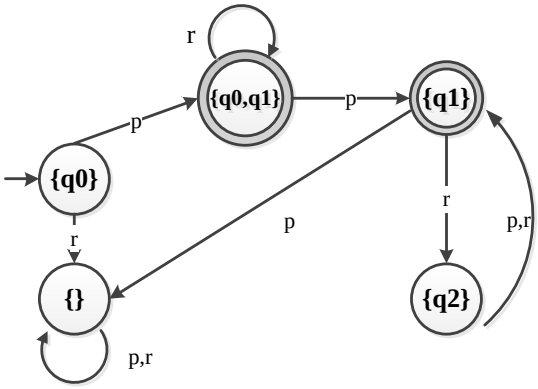
δ	p	r
q_0	{q_1,q_2}	{}
q_1	{}	{q_2}
q_2	{q_1}	{q_1}



1. *State* yang akan dibentuk: {}, {q_0} {q_1},{q_2}, {q_0,q_1}, {q_0,q_2}, {q_1,q_2}, {q_0,q_1,q_2}
2. Telusuri *state:*

δ	p	r
{}	{}	{}
{q_0}	{q_1,q_2}	{}
{q_1}	{}	{q_2}
{q_2}	{q_1}	{q_1}
{q_0,q_1}	{q_1,q_2}	{q_2}
{q_0,q_2}	{q_1,q_2}	{q_1}
{q_1,q_2}	{q_1}	{q_1,q_2}
{q_0,q_1,q_2 }	{q_1,q_2}	{q_1,q_2}

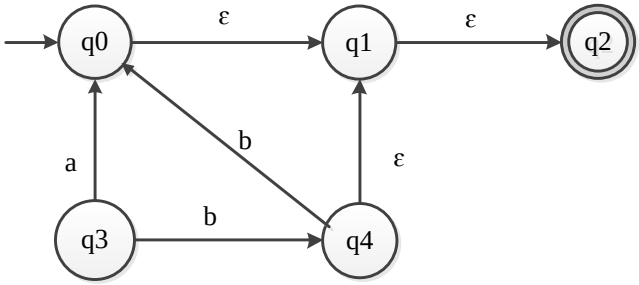
3. *State* awal: $\{q_0\}$
4. *State* akhir yang mengandung q_1, yaitu $\{q_1\},\{q_1,q_2\}$
5. Reduksi $\{q_0,q_1\}\{q_0,q_2\}\{q_0,q_1,q_2\}$ sehingga FSA menjadi



4.8. Ekivalensi NFA ε - MOVE ke NFA TANPA ε - MOVE NFA ε – MOVE

NFA ε – MOVE adalah mesin NFA yang diperbolehkan mengubah state tanpa membaca input. Disebut dengan transisi ε karena tidak bergantung pada suatu input ketika melakukan transisi.

Contoh NFA ε – MOVE



- q0 tanpa membaca input dapat berpindah ke q1
- q1 tanpa membaca input dapat berpindah ke q2

- q4 tanpa membaca input dapat berpindah ke q1

ε - closure adalah himpunan *state- state* yang dapat dicapai dari suatu *state* tanpa membaca input

ε - closure (q0) = { q0, q1, q2 }
ε - closure (q1) = { q1, q2 }
ε - closure (q2) = { q2 }
ε - closure (q3) = { q3 }
ε - closure { q4) = { q1, q2, q4 }

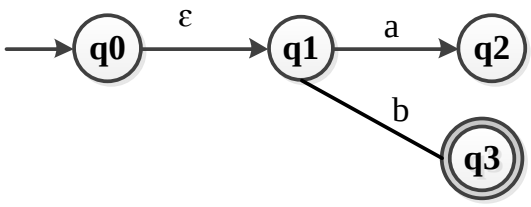
Pada suatu *state* yang tidak memiliki transisi ε maka ε - closurenya adalah *state* itu sendiri.

4.9. Ekivalensi N FA ε - move ke NFA tanpa ε - MOVE

Tahapan – tahapan / langkah ekivalensi NFA dengan ε - move ke NFA tanpa ε - Move:

1. Buat tabel Transisi NFA ε – Move
2. Tentukan - Closure e untuk setiap state
3. Carilah setiap fungsi hasil perubahan dari NFA ε - Move ke NFA tanpa ε - Move dengan rumus: δ' (state, input) = ε - closure (δ (ε - closure (state), input))
4. Berdasarkan hasil no. 3 buat tabel transisi & diagram transisi NFA tanpa ε - Move yang ekivalensi dengan NFA ε - Move ε
5. Tentukan state akhir untuk NFA tanpa ε- Move tersebut, yaitu state – state akhir semua & tambah dengan state – state yang ∑ - closurenya menuju ke salah satu dari state akhir semula.

Contoh mesin NFA ε - Move



Buat Ekivalensi NFA

ε- Move ke NFA tanpa ε – Move

1. Tabel Transisi NFA e – Move

δ	a	b
q0	ϕ	ϕ
q1	q2	q3
q2	ϕ	ϕ
q3	ϕ	ϕ

2. ε - closure (q0) = { q0, q1 }
   ε - closure (q1) = { q1 }
   ε - closure (q2) = { q2 }
   ε - closure (q3) = { q3 }
3. δ' (state, input) = ε - closure (δ(ε - closure (state), input))
   δ' (q0, a) = ε - closure (δ (ε - closure (q0), a))
   = ε - closure (δ ({ q0, q1 }, a))
   = ε - closure (q2)
   = { q2 }
   δ' (q0, b) = ε - closure (δ (ε - closure (q0), b))
   = ε - closure (δ ({ q0, q1 }, b))
   = ε - closure (q3)
   = { q3 }
   δ' (q1, a) = ε - closure (δ (ε - closure (q1), a))
   = ε - closure (δ ({ q1 }, a))
   = ε - closure (q2)
   = { q2 }
   δ' (q1, b) = ε - closure (δ (ε - closure (q1), b))
   = ε - closure (δ ({ q1 }, b))
   = ε - closure (q3)
   = { q3 }
   δ' (q2, a) = ε - closurε (δ (ε - closurε (q2), a))
   = ε - closurε (δ ({ q2 }, a))
   = ε - closurε (ϕ)
   = ϕ
   δ' (q2, b) = ε - closurε (δ (ε - closurε (q2), b))

= ε - closurε (δ ({ q2 }, b))
= ε - closurε (ϕ)
= ϕ

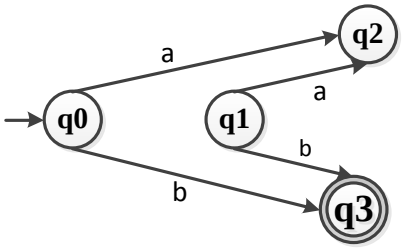
δ' (q3, a) = ε - closurε (δ (ε - closurε (q3), a))
= ε - closurε (δ ({ q3, a })
= ε - closurε (ϕ)
= ϕ

δ' (q3, b) = ε - closurε (δ (ε - closurε (q3), b))
= ε - closurε (δ ({ q3 }, b))
= ε - closurε (ϕ)
= ϕ

4. Tabel Transisi

δ'	**a**	**b**
q0	q2	q3
q1	q2	q3
q2	ϕ	ϕ
q3	ϕ	ϕ

5. Diagram



4.10. Latihan Soal

Latihan 1

Berikut adalah Finite State Automata:

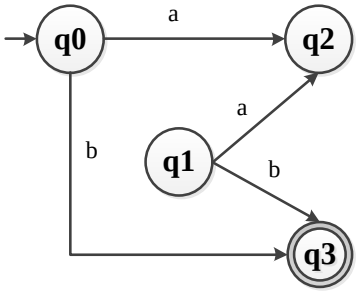
Q = {q0,q1,q2,q3}
∑ = {a,b}
S = q0
F = q3

δ = $\delta(q0,a) = q2$
$\delta(q0,b) = q3$
$\delta(q1,a) = q2$
$\delta(q1,b) = q3$

Tabel transisi:

δ	a	b
q0	q2	q3
q1	q2	q3
q2	-	-
q3	-	-

Diagram FA



Telusurilah, apakah kalimat-kalimat berikut diterima NFA di atas,
ab, abc, aabc, aabb
Jawab:

1. $\delta(q^0,ab) \Rightarrow \delta(q^0,b) \cup \delta(q^1,b) \Rightarrow \{q^0, q^2\} \cup \{q^1\} = \{q^0, q^1, q^2\}$
   Himpunan *state* TIDAK mengandung *state* AKHIR $\Rightarrow$ kalimat ab *tidak diterima*

2. $\delta(q^0,abc) \Rightarrow \delta(q^0,bc) \cup \delta(q^1,bc) \Rightarrow \{\delta(q^0,c) \cup \delta(q^2,c)\} \cup \delta(q^1, c)$

   $\{\{q^0, q^3\} \cup \{q^2\}\} \cup \{q^1\} = \{q^0, q^1, q^2, q^3\}$
   Himpunan *state* TIDAK mengandung *state* AKHIR $\Rightarrow$ kalimat abc *tidak diterima*

3. $\delta(q^0, aabc) \Rightarrow \delta(q^0, abc) \cup \delta(q^1, abc) \Rightarrow \{\delta(q^0, bc) \cup \delta(q^1, bc)\} \cup \delta(q^1, bc) \Rightarrow \{\{\delta(q^0, c) \cup \delta(q^2, c)\} \cup \delta(q^1, c)\} \cup \delta(q^1, c) \Rightarrow \{\{\{q^0, q^3\} \cup \{q^2\}\} \cup \{q^1\}\} \cup \{q^1\} = \{q^0, q^1, q^2, q^3\}$

   Himpunan *state* TIDAK mengandung *state* AKHIR $\Rightarrow$ kalimat aabc *tidak diterima*

4. $\delta(q^0, aabb) \Rightarrow \delta(q^0, abb) \cup \delta(q^1, abb)$

   $\Rightarrow \{\delta(q^0, bb) \cup \delta(q^1, bb)\} \cup \delta(q^1, bb)$

   $\Rightarrow \{\{\delta(q^0, b) \cup \delta(q^2, b)\} \cup \delta(q^1, b)\} \cup \delta(q^1, b)$

   $\Rightarrow \{\{\{q^0, q^2\} \cup \{q^2, q^4\}\} \cup \{q^1\}\} \cup \{q^1\} = \{q^0, q^1, q^2, q^4\}$

   Himpunan *state* mengandung *state* AKHIR $\Rightarrow$ kalimat aabb *diterima*

Kasus:

Misalkan X = (Qx, Σ, δx, q0, Fx) menjadi NDFA yang menerima bahasa L (X). Kita harus desain DFA Y yang setara = (Qy, Σ, δy, q0, Fy) sedemikian hingga L (Y) = L (X). Pengikut prosedur mengubah NDFA menjadi DFA yang setara:

Algoritma

Input : Sebuah NDFA

Output : DFA yang setara

Langkah 1	Buat tabel negara dari NDFA yang diberikan.
Langkah 2	Buat tabel keadaan kosong di bawah kemungkinan memasukkan abjad untuk yang setara DFA.
Langkah 3	Tandai kondisi awal DFA oleh q0 (Sama seperti NDFA).
Langkah 4	Cari tahu kombinasi dari Negara Bagian {Q0, Q1, ..., Qn} untuk setiap input yang memungkinkan alfabet.

Langkah 5 Setiap kali kami menghasilkan status DFA baru di bawah kolom alfabet masukan, kita harus menerapkan langkah 4 lagi, jika tidak lanjut ke langkah 6.

Langkah 6 Negara-negara yang mengandung salah satu negara final dari NDFA adalah final keadaan dari DFA yang setara.

Latihan 2:

Diketahui:

Q = {A, B, C, F}

∑ = {0,1}

S = A

F = F

Fungsi transisi:

δ(A,0) =A

δ(A,1) =B

δ(B,0) =C

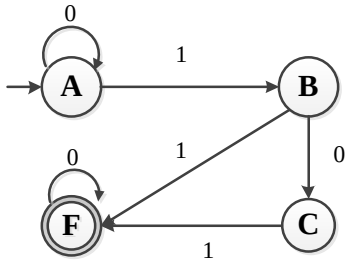
δ(B,1) =F

δ(C,1) =F

δ(F,0) =F

Gambarkan diagram FA untuk fungsi transisi diatas

Jawab:



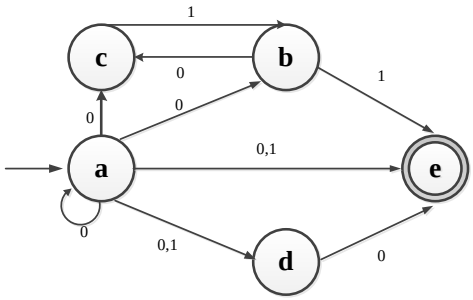
Jika pada FA diatas diinputkan string: 0010100

Maka prosesnya adalah sebagai berikut:

δ(A,~~0~~0101000) = δ(A,~~00~~10100) = δ(B,~~001~~0100) = δ(C,~~0010~~100) = δ(F,~~00101~~00) = δ(F,~~001010~~0) = δ(F,~~0010100~~)

Jadi input: 0010100
Output: diterima
A➔0A ➔00A ➔ 001B ➔ 0010C ➔ 00101F ➔ 001010F ➔ 0010100F

Latihan 3:
Perhatikan NDFA yang ditunjukkan pada gambar dibawah ini.

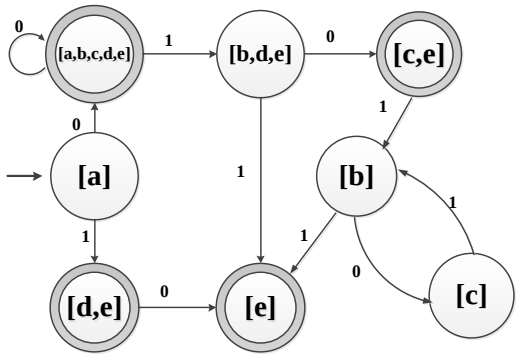


q	δ(q,0)	δ(q,1)
a	{a,b,c,d,e}	{d,e}
b	{c}	{e}
c	∅	{b}
d	{e}	∅
e	∅	∅

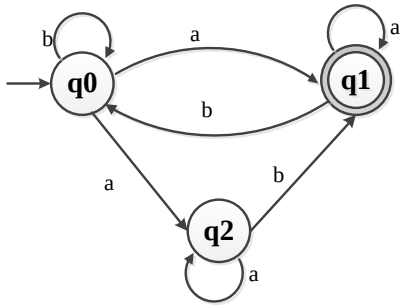
Menggunakan algoritma di atas, didapat DFA yang setara. Tabel *state* DFA ditampilkan di bawah.
Tabel *state* DFA setara dengan NDFA

q	δ(q,0)	δ(q,1)
[a]	[a,b,c,d,e]	[d,e]
[a,b,c,d,e]	[a,b,c,d,e]	[b,d,e]
[d,e]	[e]	∅
[b, d,e]	[c,e]	[e]
[e]	∅	∅
[c,e]	∅	[b]
[b]	[c]	[e]
[c]	∅	[b]

Diagram state DFA adalah sebagai berikut:



Konfigurasi NFA adalah sebagai berikut.
Q = {q0, q1, q2} ; ∑ = {a, b} ; S = q0 ; F = {q1}



Fungsi Transisi

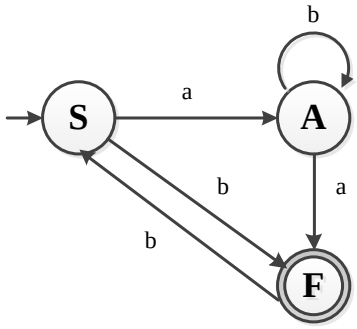
δ(q0 , a) = {q1, q2} ; δ (q0 , b) = {q0}
δ (q1 , a) = {q1} ; δ (q1 , b) = {q0}
δ (q2 , a) = {q2} ; δ (q2 , b) = {q1}

Tabel Transisi

δ	**a**	**b**
q0	{q1, q2}	{q0}
q1	{q1}	{q0}
q2	{q2}	{q1}

Latihan 4:

Diagram FA



Pertanyaan:

Tentukan output jika diinputkan string:

1) abababa
2) aabb
3) ababb
4) ababab

Jawab

1) Input: abababa
   - Proses: δ(s,abababa) = δ(a,bababa) = δ(a,ababa) = δ(f,baba) = δ(s,aba) = δ(a,ba) = δ(a,a) = f
   - Output: diterima
2) Input: aabb
   Proses: δ(s,aabb) = δ(a,abb) = δ(f,bb) = δ(s,b) =f
   Output: diterima
3) Input: ababb
   Proses: δ(s,ababb) = δ(a,babb) = δ(a,abb) = δ(f,bb) = δ(s,b) = f
   Output: diterima
4) Input: ababab
   Proses: δ(s,ababab) = δ(a,babab) = δ(a,abab) = δ(f,bab) = δ(s,ab) = δ(a,b) = a
   Output: ditolak (karena proses tidak sampai ke state akhir F)

BAB V
FSA DENGAN OUTPUT

5.1. Pendahuluan

Keterbatasan Otomata Hingga yang telah dipelajari adalah hanya pada keputusan menerima atau menolak suatu bahasa. Otomata hingga disebut juga sebagai *accepter*. Sedangkan otomata hingga yang mempunyai keluaran (*output*) disebut juga *transducer*. Otomata hingga yang mempunyai *output* terdiri dari dua jenis, yaitu mesin Moore dan Mesin Mealy. Mesin Moore mempunyai keluaran pada *state*. Sedangkan mesin Mealy mempunyai keluaran pada transisi.

FSA: *accepter*, dapat menerima atau tidak.

FSA dengan *output: transducer*

1. Mesin Moore:*output* berasosiasi dengan *state*
2. Mesin Mealy:*output* berasosiasi dengan transisi

5.2. Mesin Moore

Mesin Moore didefinisikan dalam 6-tupel, yaitu $M = (Q, \sum, \delta, S, \Delta, \lambda)$.

Q = himpunan *state*

$\sum$ = himpunan simbol input

δ = fungsi transisi

S = *state* awal, $S \in Q$

Δ = himpunan keluaran (*output*)

Λ = fungsi keluaran (*output*) setiap *state*.

Contoh mesin moore untuk memperoleh modulus 3 pada suatu bilangan biner: $M = (Q,\sum,\delta,S,\Delta,\lambda)$

Q : q0,q1,q2

$\sum$: [0,1]

S : q0

Δ : [0,1,2]

$\lambda(q0) = 0$

$\lambda(q1) = 1$
$\lambda(q2) = 2$

Prinsip:
Jika i diikuti dengan 0, maka hasilnya 2i

$101_2 = 5$ 10102 = 2*5 =10

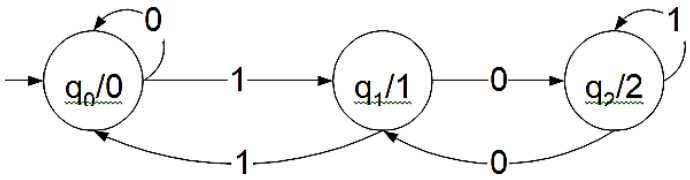
Jika i diikuti dengan 1, maka hasilnya 2i+1

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

Jika i/3 mempunyai sisa p, maka untuk input berikutnya bernilai 0 maka 2i/3 mempunyai sisa 2p mod 3
Untuk p=0 maka 2p mod 3 = 0
Untuk p=1 maka 2p mod 3 = 2
Untuk p=2 maka 2p mod 3 = 1

Jika i/3 mempunyai sisa p, maka untuk input berikutnya bernilai 1 maka (2i+1)/3 mempunyai sisa (2p+1) mod 3
Untuk p=0 maka (2p+1) mod 3 = 1
Untuk p=1 maka (2p+1) mod 3 = 0
Untuk p=2 maka (2p+1) mod 3 = 2

Sehingga didapat mesin FSA sbb:



Komponen *state* akhir dihilangkan, karena keputusan dimunculkan sebagai *output*.

Komponen state Final dari Deterministic Finite Automata dihilangkan, karena disini keputusan dimunculkan sebagai output. Kita lihat contoh penerapan dari Mesin Moore. Misal kita ingin memperoleh sisa pembagian (modulus) suatu bilangan dengan 3. Dimana input

dinyatakan dalam biner. Mesin Moore yang bersesuaian bisa dilihat pada gambar 1. Mesin Moore yang bersesuaian bisa dilihat pada gambar 10.1.Konfigurasi mesin sebagai berikut:

Q = {q0,q1,q2}

Σ = {0,1} (input dalam biner)

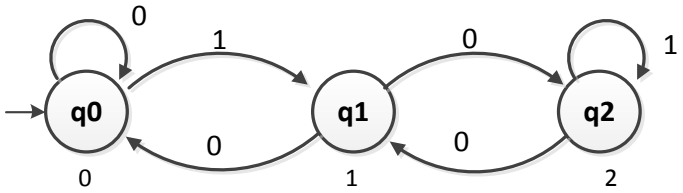
Δ = {0,1,2} (untuk output-nya pada kasus mod dengan 3 maka sisanya kemungkinan adalah (0,1,2)

S = q0

λ (q0) = 0

λ (q1) = 1

λ (q2) = 2



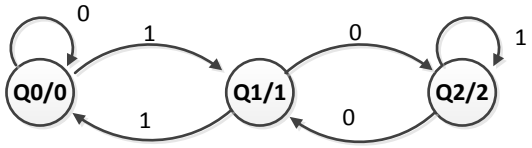
Gambar 10.1. Mesin Moore untuk Modulus 3

Misalkan saja

- 5 mod 3 = ? input 5 dalam biner 101 bila kita masukkan 101 ke dalam mesin, urutan state yang dicapai: q0,q1,q2,q3 Perhatikan state terakhir yang dicapai adalah q2, λ (q2) =2, maka 5 mod 3 = 2
- 10 mod 3 =? input 10 dalam biner 1010 bila kita masukkan 1010 ke dalam mesin, urutan state yang dicapai: q0,q1,q2,q2, q1 λ (q1) =1, maka 10 mod 3 = 1

Contoh 10.1

Berikut adalah mesin Moore yang menghitung sisa pembagian bilangan bulat positif dengan 3.



Gambar 10.2

Q = {q0, q1, q2}
∑ = {0, 1}
S = q0
Δ = { 0, 1, 2}
λ (q0) = 0
λ (q1) = 1
λ (q2) = 2
δ (q0,0) = q0
δ (q0,1) = q1
δ (q1,0) = q2
δ (q1,1) = q0
δ q2,0) = q1
δ (q2,1) = q2

5.3. Mesin Mealy

Pada mesin Mealy output yang dihasilkan berasosiasi dengan transisi. Mesin Mealy didefinisikan sebagai mesin 6-tupel

$$M = (Q, \sum, \delta, S, \Delta, \lambda)$$

Dimana:
Q = himpunan *state*
∑ = himpunan simbol input
δ = fungsi transisi
S = *state* awal
Δ = himpunan *output*
Λ = fungsi *output* untuk setiap transisi

Mesin Mealy tidak mempunyai *state* final, karena keputusan sudah dimunculkan sebagai *output*.

Contoh penerapan Mesin Mealy kita lihat pada gambar 10.3. Mesin itu akan mengeluarkan output apakah menerima (Y) atau menolak (T), suatu masukan. Dimana mesin akan mengeluarkan output 'Y' bila menerima untai yang memiliki akhiran 2 simbol berturutan yang sama, atau secara formal dalam ekspresi regular:

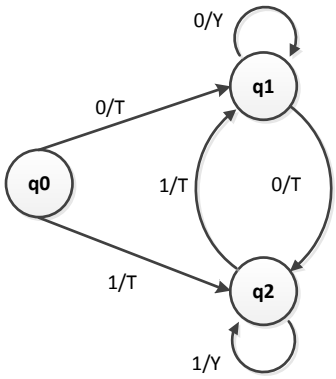
$$(0+1)^*(00+11)$$

Contoh input yang diterima:

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

Konfigurasi dari Mesin Mealy tersebut:

Q = {q0,q1,q2}
Σ = {0,1}
Δ = {Y,T}
S = q0 λ (q0,0) = T
λ (q0,1) = T
λ (q1,0) = Y
λ (q1,1) = T
λ (q2,0) = T
λ (q2,1) = Y



Gambar 10.3. Mesin Mealy

Contoh 10.2

Berikut adalah mesin Mealy yang mengeluarkan *output* menerima atau menolak suatu masukan. Mesin akan mengeluarkan output menerima (Y) bila menerima untai yang berakhiran 00 atau 11.

$Q = \{q0, q1, q2\}$
$\sum = \{0, 1\}$
$S = q0$
$\Delta = \{ Y, T\}$
$\delta(q0,0) = q1$
$\delta(q0,1) = q2$
$\delta(q1,0) = q1$
$\delta(q1,1) = q2$
$\delta(q2,0) = q1$
$\delta(q2,1) = q2$
$\lambda (q0, 0) = T$
$\lambda (q0, 1) = T$
$\lambda (q1, 0) = Y$
$\lambda (q1, 1) = T$
$\lambda (q2\ 0) = T$
$\lambda (q2, 1) = Y$

FSA: *accepter*, dapat menerima atau tidak.
FSA dengan *output: transducer*

1. Mesin Moore:*output* berasosiasi dengan *state*
2. Mesin Mealy:*output* berasosiasi dengan transisi

Mesin Moore

$M = (Q,\Sigma,\delta,S,\Delta,\lambda)$

Q : himpunan *state*
Σ : himpunan simbol input
δ : fungsi transisi
S : *state* awal $S \in Q$
Δ : himpunan *output*
λ : fungsi *output* untuk setiap ***state***

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

$M = (Q,\Sigma,\delta,S,\Delta,\lambda)$

$Q : q_0,q_1,q_2$

$\Sigma : [0,1]$

$S : q_0$

$\Delta : [0,1,2]$

$\lambda(q_0) = 0$

$\lambda(q_1) = 1$

$\lambda(q_2) = 2$

Prinsip:

Jika i diikuti dengan 0, maka hasilnya 2i

$$101_2 = 5 \qquad 1010_2 = 2*5 = 10$$

Jika i diikuti dengan 1, maka hasilnya 2i+1

$$101_2 = 5 \qquad 1011_2 = 2*5+1 = 11$$

Jika i/3 mempunyai sisa p, maka untuk input berikutnya bernilai 0 maka 2i/3 mempunyai sisa 2p mod 3

Untuk p=0 maka 2p mod 3 = 0

Untuk p=1 maka 2p mod 3 = 2

Untuk p=2 maka 2p mod 3 = 1

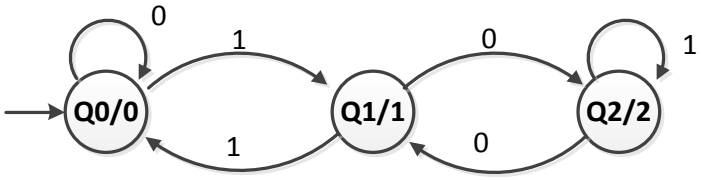
Jika i/3 mempunyai sisa p, maka untuk input berikutnya bernilai 1 maka (2i+1)/3 mempunyai sisa (2p+1) mod 3

Untuk p=0 maka (2p+1) mod 3 = 1

Untuk p=1 maka (2p+1) mod 3 = 0

Untuk p=2 maka (2p+1) mod 3 = 2

Sehingga didapat mesin FSA sbb:



5.4. Ekuivalensi mesin Moore dengan mesin Mealy

Dari suatu Mesin Moore dapat dibuat Mesin Mealy yang ekivalen, begitu juga sebaliknya. Untuk mesin Mealy pada gambar 2 dapat kita buat Mesin Moore yang ekivalen yaitu gambar 3. Bisa kita lihat state pada mesin Moore dibentuk dari q2 q0 q1 0/T 0/T 0/Y 1/T 1/T 1/Y kombinasi state pada Mealy dan banyaknya output. Karena jumlah state Mealy = 3, dan jumlah output = 2, maka jumlah state pada Moore yang ekivalen = 6. Bisa dilihat konfigurasi Mesin Moore yang dibentuk:

$Q = \{q_0Y, q_0T, q_1Y, q_1T, q_2Y, q_2T\}$

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

$\Delta = \{Y,T\}$

$S = q_0$

$\lambda (q_0Y) = Y$

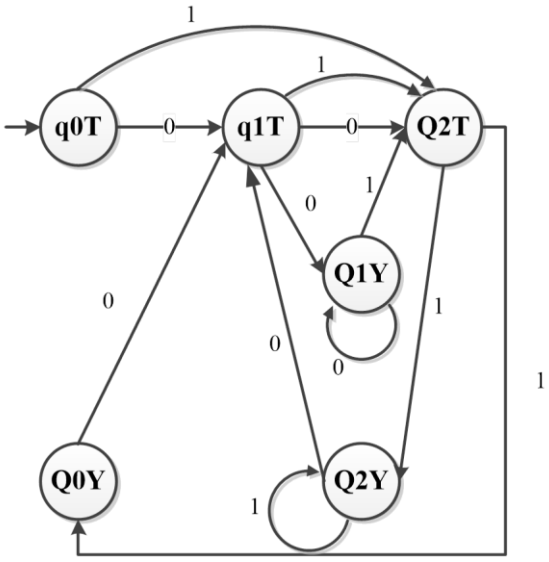
$\lambda (q_0T) = T$

$\lambda (q_1Y) = Y$

$\lambda (q_1T) = T$

$\lambda (q_2Y) = Y$

$\lambda (q_2T) = T$



Bila kita perhatikan dari gambar diatas, state q0Y dapat dihapus karena tidak ada busur yang mengarah ke state tersebut. Untuk memperoleh ekivalensi mesin Mealy dari suatu mesin Moore caranya lebih mudah, cukup dengan menambahkan label output ke setiap transisi dan menghapus label output pada setiap *state*. Gambar diatas. Merupakan mesin Mealy yang ekivalen dengan mesin Moore pada gambar berikut.

Konfigurasi Mesin Mealy tersebut:

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

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

$\Delta = \{0,1,2\}$

$S = q_0$

$\lambda(q_0,0) = 0$

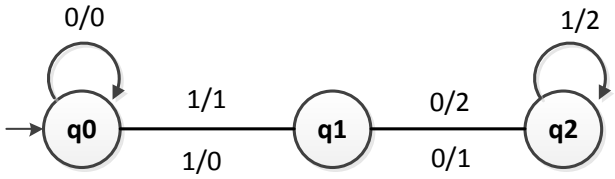
$\lambda(q_0,1) = 1$

$\lambda(q_1,0) = 2$

$\lambda(q_1,1) = 0$

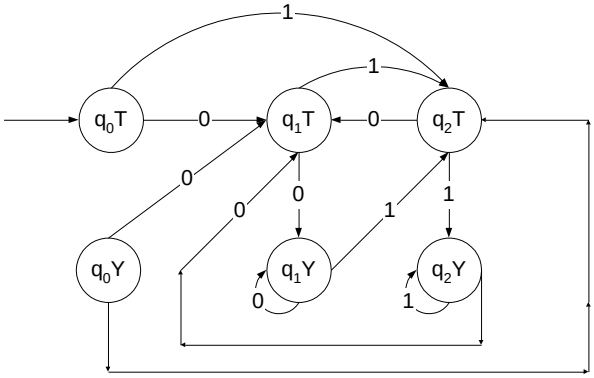
$\lambda(q_2,0) = 1$

$\lambda(q_2,1) = 2$

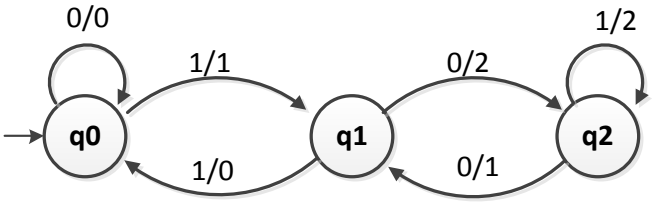


- Mesin Moore ke mesin Mealy

Jml *state* = jml state sebelum * jml output



- Mesin Mealy ke mesin Moore
  Menambah label *output* pada transisi
  Menghapus label *output* pada *state*



5.5. Latihan Soal

Buatlah mesin Mealy dan Moore untuk proses membaca input (0+1)*:

- Jika input berakhir dengan 101, outputnya A
- Jika input berakhir dengan 110, outputnya A
- Jika yang lainnya , 8outputnya C

Soal Latihan 1:

Input 5 (1012), state terakhir q2/2, 5 mod 3 = 2

Input 10 (10102), state terakhir q1/1, 10 mod 3 = 1 mesin mealy

M = (q,s,d,s,d,l)

Q : himpunan state

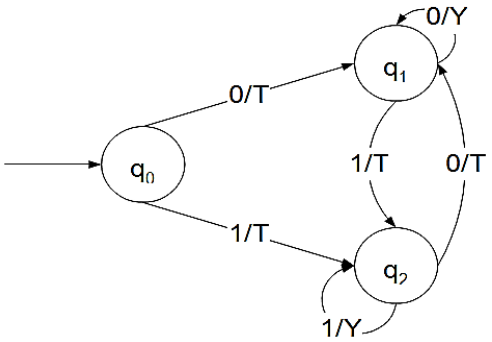
∑ : himpunan simbol input

δ : fungsi transisi
S : state awal S ÎQ
Δ : himpunan output
λ : fungsi output untuk setiap **transisi**

Contoh mesin Mealy untuk mendeteksi ekspresi regular

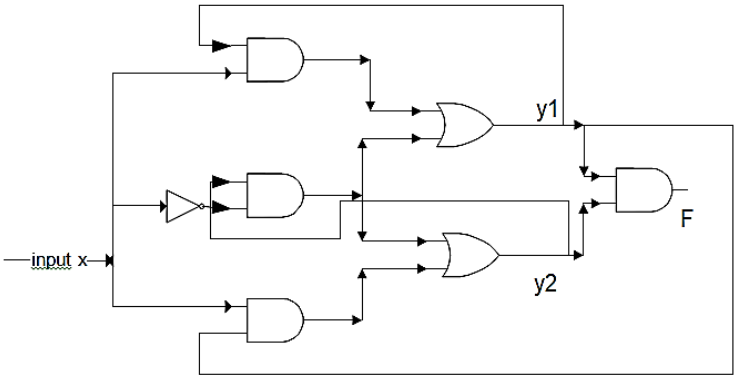
(0+1)*(00+11)

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

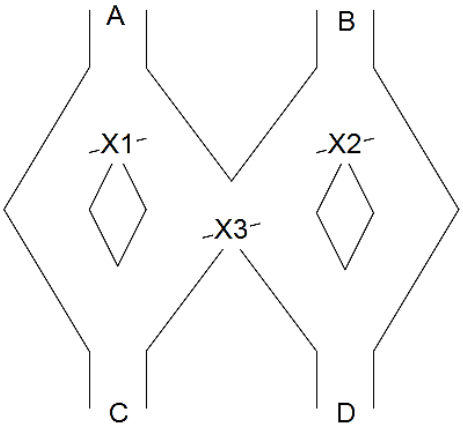


Latihan 2

- Tentukan FSA dari rangkaian sirkuit berikut ini. Asumsi bahwa terdapat waktu yang cukup untuk perambatan sinyal menuju kondisi yang stabil.



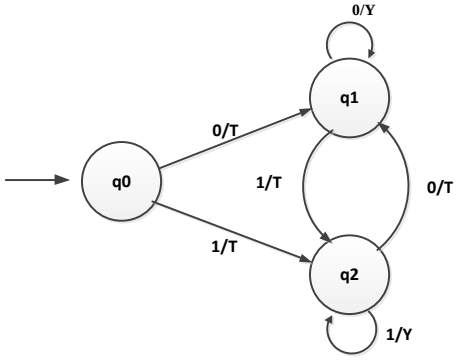
- Kelereng dijatuhkan dari A atau B. Percabangan x1, x2 dan x3 menentukan saluran mana yang akan dilewati kelereng (kiri / kanan). Ketika percabangan dilewati, kelereng berikutnya akan melewati dengan saluran berbeda. Buatlah FSA nya



Latihan 3:
Contoh Mesin Mealy:
Konfigurasi mesin
Q = {q0, q1, q2}
∑ = { 0, 1}

Δ = { Y, T}
S = {q0}

λ = (q0, 0) = T
λ = (q0, 1) = T
λ = (q1, 0) = Y
λ = (q1, 1) = T
λ = (q2, 0) = T
λ = (q2, 1) = Y



Jika Mesin Mealy menerima input "110101", apakah diterima oleh mesin tersebut?

δ (q0, 110101) = δ (q2, 10101)
T
= δ (q2, 0101) = δ (q1, 101)
Y T
= δ (q2, 01) = δ (q1, 1) = q2
Y T T

Tidak diterima input 110101, ditolak oleh Mesin Mealy

BAB VI
TATA BAHASA BEBAS KONTEKS

6.1. Pendahuluan

Tata bahasa bebas Konteks menggambarkan struktur keseluruhan dari sebuah program, tetapi diikuti statement tertentu yang tidak diperbolehkan dalam bahasa pemrograman. Sisa dari perincian tersebut terdiri dari context sensitive. Pembatasan konteks sensitif biasanya ditempatkan secara tidak resmi dalam banyak bahasa asli. Tata bahasa bebas konteks sangat penting dalam penggambaran dan penerjemahan bahasa pemrograman.

Tata bahasa bebas konteks (*Context Free Grammar*), biasa disingkat dengan CFG memiliki batasan sebagai berikut:

Tata-bahasa Tipe 2 (*Context Free Grammar*)**:** G ($\sum$, N, S, P), adalah tata bahasa tipe-1 yang memiliki aturan produksi: $\alpha \rightarrow \beta$ dengan tambahan batasan:

α : hanya terdiri dari 1 simbol non terminal saja, atau $\alpha \in N$

β : tidak dibatasi, atau: $\beta \in \{(\sum \cup N)^*$

6.2. Perbedaan CFG dan Bahasa Regular

Perbedaan antara bahasa bebas konteks dengan bahasa *regular* adalah pada string sisi kanan tanda panah untuk bahasa *regular* harus satu terminal tunggal atau terminal tunggal diikuti non terminal sedangkan untuk bahasa bebas konteks tidak dibatasi.

Contoh:

a) *Grammar* dengan aturan:

$S \rightarrow a|aA$

$A \rightarrow bB$

$B \rightarrow \varepsilon$

Adalah tata bahasa *regular*

b) *Grammar* dengan aturan:

S→a|aA
A→bB | AA
B→ε

Adalah tata bahasa bebas konteks karena ada aturan produksi: A→AA (Hamzah, 2009).

Penyederhanaan Tujuan dari penyederhanaan adalah melakukan pembatasan sehingga tidak menghasilkan pohon penurunan yang memiliki kerumitan yang tidak perlu atau aturan produksi yang tidak berarti.

Contoh 1:

S → AB | a

A→ a

c) Aturan produksi S → AB tidak berarti karena B tidak memiliki penurunan

Contoh 2:

S→A

A→B

B→C

C→D

D→ a | A

d) Memiliki kelemahan terlalu panjang jalannya padahal berujung pada S→a,

Produksi D → A juga menyebabkan kerumitan.

Motivasi awal:

Deskripsi bahasa alami

<kalimat> → <subjek> <predikat>

<subjek> → <kata benda>

<predikat> → <kata kerja>

<kata benda> → kucing

<kata kerja> → berlari

<kata kerja> → menyapu

Contoh kalimat yang dapat dihasilkan

Kucing berlari

Kucing menyapu (sintaks yes, semantik no)
Dalam Tata bahasa bebas konteks

e) Ruas kiri dari aturan produksi terdiri dari SATU simbol non terminal
f) Ruas kanan dapat berupa string yang dibentuk dari simbol terminal dan non terminal

Contoh: $S \rightarrow aSb \mid \varepsilon$
Kalimat-kalimat yang dibangkitkan dari aturan produksi itu adalah ε,ab,aabb,aaabbb,... , $a^n b^n$
Contoh
$A \rightarrow 0A0$
$A \rightarrow 1A1$
$A \rightarrow a$

Kalimat-kalimat yang dibangkitkan dari aturan produksi itu adalah a,01a10, 1001a1001 , 110a011 $\beta a \beta^R$
Contoh: $S \rightarrow aSb \mid SS \mid \varepsilon$
Bahasa yang dihasilkan oleh Tata bahasa dengan aturan produksi di atas adalah: $L = \{w \in (a + b)^* \mid n_a(w) = n_b(w)\}$

6.3. Leftmost dan Rightmost Derivation

Suatu penguraian /penurunan dikatakan *leftmost derivation* bila setiap tahapan penurunan variabel / non terminal terkiri yang diuraikan. Apabila setiap tahapan penurunan variabel / non terminal paling kanan yang diuraikan disebut *rightmost derivation.*

Contoh 1:
G=({A,B,S}, {a,b},S,P} dengan aturan produksi P:
$S \rightarrow AB$
$A \rightarrow aaA \mid \lambda$
$B \rightarrow Bb \mid \lambda$

Menspesifikasikan bahasa

$$L(G) = \{a^{2n} b^m \mid n \geq 0 , m \geq 0\}$$

Leftmost derivation untuk menghasilkan string aab: S ⇒ AB ⇒ aaAB ⇒ aaB ⇒ aaBb ⇒ aab

Righmost derivation untuk menghasilkan string aab: S ⇒ AB ⇒ ABb ⇒ aaABb ⇒aaAb ⇒aab

Contoh 2 :

G=({A,B,S}, {a,b},S,P} dengan aturan produksi P:

S → aAB

A→ bBb

B→ A | λ

Leftmost derivation untuk menghasilkan string abbbb

S ⇒ aAB ⇒ abBbB ⇒ abAbB ⇒ abbBbbB

⇒ abbbbB ⇒ abbbb

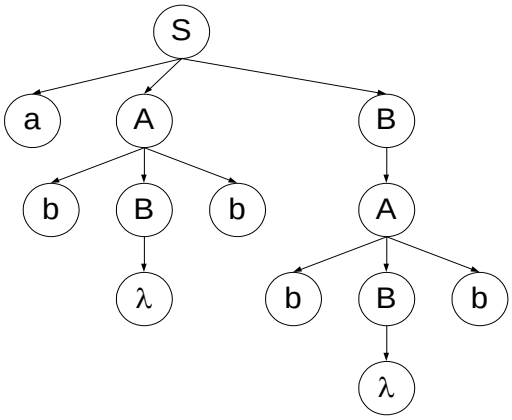
Righmost derivation untuk menghasilkan string aab

S ⇒ aAB ⇒ aA ⇒ abBb ⇒ abAb ⇒ abbBbb ⇒ abbbb

6.4. Pohon Urai

Untuk menampilkan penguraian, dapat dilakukan dengan membentuk pohon urai (sayangnya, urutan penguraian tidak terlihat) .

Contoh pohon urai pada contoh sebelumnya:



Parsing dan Keanggotaan

Untuk menentukan apakah string w berada di L(G), dengan cara secara sistematis membangun semua kemungkinan penurunan, dan mencocokkan hasilnya apakah ada yang sama dengan string w. (disebut exhaustive search parsing)

Contoh: menentukan apakah string ab berada pada bahasa yang dibentuk oleh grammar dengan aturan produksi

$$S \rightarrow SS \mid aSb \mid bSa \mid \lambda$$

Untuk penguraian pertama

1. $S \Rightarrow SS$
2. $S \Rightarrow aSb$
3. $S \Rightarrow bSa$
4. $S \Rightarrow \lambda$

Penguraian nomor 3 dan 4 tidak perlu dilanjutkan.

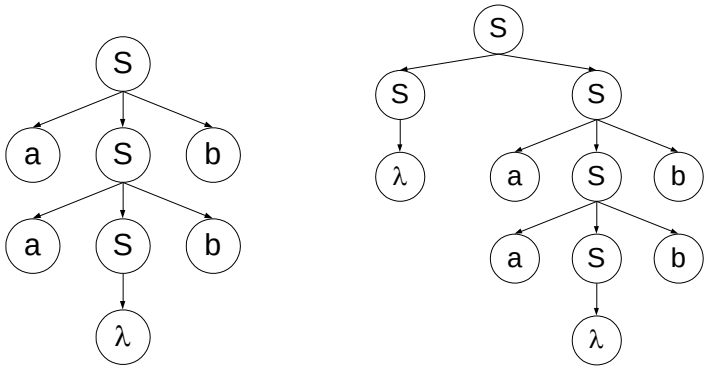
Penguraian 1 membentuk	Penguraian 2 membentuk
1a. $S \Rightarrow SS \Rightarrow SSS$	2a. $S \Rightarrow aSb \Rightarrow aSSb$
1b. $S \Rightarrow SS \Rightarrow aSbS$	2b. $S \Rightarrow aSb \Rightarrow aaSbb$
1c. $S \Rightarrow SS \Rightarrow bSaS$	2c. $S \Rightarrow aSb \Rightarrow abSab$
1d. $S \Rightarrow SS \Rightarrow S$	2d. $S \Rightarrow aSb \Rightarrow ab$

Ambiguitas pada Tata Bahasa dan Bahasa

Tata bahasa bebas konteks G disebut ambigu jika terdapat beberapa $w \in L(G)$ yang mempunyai paling sedikit dua buah pohon penurunan

Contoh: $S \rightarrow SS \mid aSb \mid \lambda$

String aabb mempunyai 2 pohon penurunan:



6.5. Pumping Lemma untuk Bahasa Bebas Konteks

- Jika suatu rangkaian simbol /string yang cukup panjang yang merupakan sebuah bahasa bebas konteks, maka kita dapat menemukan dua substring yang jaraknya berdekatan yang jika dipompa, string baru yang diperoleh merupakan bahasa bebas konteks juga.
- Secara formal, lemma diatas dinyatakan dengan

$$(\forall L)(\exists n)(\forall z)\left[\begin{array}{l} z \in L \wedge |z| \geq n \Rightarrow \\ (\exists u,v,w,x,y)\left(\begin{array}{l} z = uvwxy \wedge |vwx| \leq n \wedge |vx| \geq 1 \Rightarrow \\ (\forall i)\left(uv^i wx^i y \in L\right) \end{array}\right) \end{array}\right]$$

- Syarat " kedua lokasi berdekatan" dinyatakan dengan kondisi $|vwx| \leq n$
- Jika salah satu v atau x diambil sebagai string kosong, maka lemma diatas berubah menjadi lemma untuk bahasa reguler

Contoh Tata bahasa dengan aturan produksi

$S \rightarrow uay$

$A \rightarrow vax$

$A \rightarrow w$

Maka aturan derivasinya

$S \Rightarrow uay \Rightarrow uwy$

$S \Rightarrow uay \Rightarrow uvaxy \Rightarrow uvwxy$

$S \Rightarrow uay \Rightarrow uvaxy \Rightarrow uvvaxxy \Rightarrow uvvwxxy$
Sehingga untuk setiap $i \geq 0$, $uv^iwx^iy \in L$

6.6. Sifat-Sifat Tertutup Bahasa Bebas Konteks

- Gabungan dua CFL merupakan CFL juga

Jika diketahui dua buah CFG $G_1 = (N_1,T_1,S_1,P_1)$ dan $G_2=(N_2,T_2,S_2,P_2)$ yang menghasilkan bahasa L_1 dan L_2 , maka CFG $L_1 \cup L_2$ dapat dibentuk dengan cara:

1. Menggabungkan kedua himpunan dan menambahkan satu simbol variabel baru S
2. Menggabungkan kedua himpunan simbol terminal
3. Menggabungkan kedua himpunan aturan produksi dan menambahkan satu aturan produksi baru

   $S \rightarrow S_1|S_2$ yang digunakan untuk memilih salah satu simbol awal S_1 atau S_2 dari simbol awal baru S: $G_3 = (N_1 \cup N_2 - \cup\{S\}, T_1 \cup T_2, S, P_1 \cup P_2 \cup \{S \rightarrow S_1|S_2\}\}$

- Penyambungan dua CFL merupakan CFL juga

Jika diketahui dua buah CFG $G_1 = (N_1,T_1,S_1,P_1)$ dan $G_2=(N_2,T_2,S_2,P_2)$ yang menghasilkan bahasa L_1 dan L_2 , maka bahasa L_1L_2 dapat dibentuk oleh:

$$G_4 = (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, S, P_1 \cup P_2 \cup \{S \rightarrow S_1S_2\}\}$$

- Klosure Kleene dari CFL adalah CFL juga.

Klosure Kleene dari Tata bahasa $G=(N,T,S_1,P)$ adalah

$$G_5 = (N \cup \{S\} , T , S , P \cup \{S \rightarrow S_1S \mid \varepsilon \})$$

Latihan

$G(L_1) = (\{S , A , B\}, \{a,b\} , S , P)$ dengan P:
$S \rightarrow AB \mid \varepsilon$
$A \rightarrow aB$
$B \rightarrow Sb$

$G(L_2) = (\{S , A , B\}, \{a,b\} , S , P)$ dengan P:
$S \rightarrow aaB$

$A \rightarrow bBb \mid \varepsilon$

$B \rightarrow aA$

Bagaimanakah:

a. CFG $G(L_1 \cup L_2)$
b. CFG $G(L_1L_2)$
c. CFG $G(L_1^*)$

- Bahasa bebas konteks tertutup terhadap substitusi

Contoh

$$L_a = \{0^n1^n \mid n \geq 1\} \text{ dan } L_b = \{ ww^R \mid w \in (0+2)^* \}$$

Dihasilkan oleh Tata bahasa G_a dengan aturan produksi

$$S_a \rightarrow 0S_a1 \mid 01$$

Serta Tata bahasa G_2 dengan aturan produksi

$$S_b \rightarrow 0S_b0 \mid 2S_b2 \mid \varepsilon$$

Didefinisikan Tata bahasa G dengan aturan produksi

$$S \rightarrow aSbS \mid bSaS \mid \varepsilon$$

Jika f adalah substitusi $f(a)= L_a$ dan $f(b) = L_b$ maka
f(L) adalah bahasa yang dihasilkan oleh Tata bahasa dengan aturan produksi

$S \rightarrow S_aSS_bS \mid S_bSS_aS \mid \varepsilon$

$S_a \rightarrow 0S_a1 \mid 01$

$S_b \rightarrow 0S_b0 \mid 2S_b2 \mid \varepsilon$

Tata bahasa Bebas Konteks dan Bahasa Pemrograman

- Tata bahasa bebas konteks digunakan untuk mendefinisikan sintaks bahasa pemrograman
- Menggunakan notasi BNF (Backus-Naur Form)
  - Variabel / non terminal: <...>
  - Terminal: tanpa tanda
  - ← diganti dengan::=

- Contoh statemen if then else
  < if_statement>::= if <expression>
  <then_clause>
  <else_clause>

6.7. Latihan Soal

1. Buatlah masing-masing 5 buah contoh tata bahasa yang memenuhi aturan tata bahasa:
   a. Regular
   b. Bebas Konteks
   c. *Context Sensitive*
   d. *Unrestricted*
2. Tentukan sebuah gramar regular untuk bahasa:

   $$L^1 = \{ a^n \mid n \geq 1\}$$

   <u>Jawab:</u>

   $Q^1(L^1) = \{S \rightarrow aS \mid a\}$
3. Tentukan sebuah gramar bebas konteks untuk bahasa:

   L^2: himpunan bilangan bulat non negatif ganjil
   <u>Jawab:</u>
   Langkah kunci: digit terakhir bilangan harus ganjil.
   Buat dua buah himpunan bilangan terpisah: genap (G) dan ganjil (J)

   $$Q^2(L^2) = \{S \rightarrow J \mid GS \mid JS,\ G \rightarrow 0 \mid 2 \mid 4 \mid 6 \mid 8,\ J \rightarrow 1 \mid 3 \mid 5 \mid 7 \mid 9\}$$
4. Tentukan sebuah gramar bebas konteks untuk bahasa:

   L^3 = himpunan semua identifier yang sah menurut bahasa pemrograman Pascal dengan batasan: terdiri dari simbol huruf kecil dan angka, panjang identifier boleh lebih dari 8 karakter
   <u>Jawab:</u>
   Langkah kunci: karakter pertama identifier harus huruf.

Buat dua buah himpunan bilangan terpisah: huruf (H) dan angka (A)

$$Q^3(L^3) = \{S \rightarrow H \mid HT,\ T \rightarrow AT \mid HT \mid H \mid A,\ H \rightarrow a \mid b \mid c \mid \ldots,\ A \rightarrow 0 \mid 1 \mid 2 \mid \ldots\}$$

5. Tentukan gramar bebas konteks untuk bahasa $L^4(G^4) = \{a^n b^m \mid n,m \geq 1,\ n \neq m\}$

<u>Jawab:</u>

Langkah kunci: sulit untuk mendefinisikan $L^4(G^4)$ secara langsung. Jalan keluarnya adalah dengan mengingat bahwa $x \neq y$ berarti $x > y$ atau $x < y$.

$$L^4 = L^A \cup L^B,\ L^A = \{a^n b^m \mid n > m \geq 1\},\ L^B = \{a^n b^m \mid 1 \leq n < m\}.$$

$$Q^A(L^A) = \{A \rightarrow aA \mid aC,\ C \rightarrow aCb \mid ab\},\ Q(L^B) = \{B \rightarrow Bb \mid Db,\ D \rightarrow aDb \mid ab\}$$

$$Q^4(L^4) = \{S \rightarrow A \mid B,\ A \rightarrow aA \mid aC,\ C \rightarrow aCb \mid ab,\ B \rightarrow Bb \mid Db,\ D \rightarrow aDb \mid ab\}$$

6. Tentukan sebuah gramar bebas konteks untuk bahasa:

L^5 = bilangan bulat non negatif genap. Jika bilangan tersebut terdiri dari dua digit atau lebih maka nol tidak boleh muncul sebagai digit pertama.

<u>Jawab:</u>

Langkah kunci: Digit terakhir bilangan harus genap. Digit pertama tidak boleh nol. Buat tiga himpunan terpisah: bilangan genap tanpa nol (G), bilangan genap dengan nol (N), serta bilangan ganjil (J).

$$Q^5(L^5) = \{S \rightarrow N \mid GA \mid JA,\ A \rightarrow N \mid NA \mid JA,\ G \rightarrow 2 \mid 4 \mid 6 \mid 8,\ N \rightarrow 0 \mid 2 \mid 4 \mid 6 \mid 8,\ J \rightarrow 1 \mid 3 \mid 5 \mid 7 \mid 9\}$$

7. Diketahui grammar $G(V^T, V^N, S, P)$ dimana:

$V^T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$V^N = \{S, G, J, T\}$

$S \in V^N$

$P = \{S \rightarrow J \mid GT \mid JT;\ T \rightarrow GT \mid JT \mid J;\ G \rightarrow 0 \mid 2 \mid 4 \mid 6 \mid 8;\ J \rightarrow 1 \mid 3 \mid 5 \mid 7 \mid 9\}$

a. G termasuk grammar tipe berapa? Berikan alasannya.
b. Buatlah 5 kalimat dengan panjang berbeda yang dapat diturunkan dari grammar G.
c. Tentukan bahasa dari grammar G (L(G)).

8. Diketahui tata bahasa bebas konteks G dengan produksi sebagai berikut:

$S \rightarrow AaA \mid CA \mid BaB$
$A \rightarrow aaBa \mid CDA \mid aa \mid DC$
$B \rightarrow bB \mid bAB \mid bb \mid aS$
$C \rightarrow Ca \mid bC \mid D$
$D \rightarrow bD \mid \varepsilon$

Sederhanakan dan kemudian ubahlah aturan produksi di atas dalam bentuk normal Chomsky (CNF)

Tentukan bahasa dari masing-masing gramar berikut:

9. G^1 dengan $P^1 = \{1.\ S \rightarrow aAa,\ 2.\ A \rightarrow aAa,\ 3.\ A \rightarrow b\}$.

<u>Jawab:</u>

Derivasi kalimat terpendek:

$S \Rightarrow aAa$ (1)
$\Rightarrow aba$ (3)

Derivasi kalimat umum:

$S \Rightarrow aAa$ (1)
$\Rightarrow aaAaa$ (2)
…
$\Rightarrow a^n Aa^n$ (2)
$\Rightarrow a^n ba^n$ (3)

Dari pola kedua kalimat disimpulkan: $L^1(G^1) = \{ a^n ba^n \mid n \geq 1\}$

10. G^2 dengan

P^2 = {1. S $\rightarrow$ aS, 2. S $\rightarrow$ aB, 3. B $\rightarrow$ bC, 4. C $\rightarrow$ aC, 5. C $\rightarrow$ a}.

<u>Jawab:</u>

Derivasi kalimat terpendek:

S $\Rightarrow$ aB (2)
$\Rightarrow$ abC (3)
$\Rightarrow$ aba (5)

Derivasi kalimat umum:

S $\Rightarrow$ aS (1)
…
$\Rightarrow a^{n-1}S$ (1)
$\Rightarrow a^{n}B$ (2)
$\Rightarrow a^{n}bC$ (3)
$\Rightarrow a^{n}baC$ (4)
…
$\Rightarrow a^{n}ba^{m-1}C$ (4)
$\Rightarrow a^{n}ba^{m}$ (5)

Dari pola kedua kalimat disimpulkan: $L^2(G^2)=\{a^n ba^m \mid n \geq 1, m \geq 1\}$

11. G^3 dengan

P^3 = {1. S $\rightarrow$ aSBC, 2. S $\rightarrow$ abC, 3. bB $\rightarrow$ bb,
4. bC $\rightarrow$ bc, 5. CB $\rightarrow$ BC, 6. cC $\rightarrow$ cc}.

<u>Jawab:</u>

Derivasi kalimat terpendek 1:

S $\Rightarrow$ abC (2)
$\Rightarrow$ abc (4)

Derivasi kalimat terpendek 2:

S $\Rightarrow$ aSBC (1)
$\Rightarrow$ aabCBC (2)
$\Rightarrow$ aabBCC (5)

Derivasi kalimat terpendek 3:

S $\Rightarrow$ aSBC (1)
$\Rightarrow$ aaSBCBC (1)
$\Rightarrow$ aaabCBCBC (2)
$\Rightarrow$ aaabBCCBC (5)
$\Rightarrow$ aaabBCBCC (5)
$\Rightarrow$ aaabBBCCC (5)

$\Rightarrow$ aabbCC	(3)	$\Rightarrow$ aaabbBCCC	(3)
$\Rightarrow$ aabbcC	(4)	$\Rightarrow$ aaabbbCCC	(3)
$\Rightarrow$ aabbcc	(6)	$\Rightarrow$ aaabbbcCC	(4)
		$\Rightarrow$ aaabbbccC	(6)
		$\Rightarrow$ aaabbbccc	(6)

Dari pola ketiga kalimat disimpulkan: L^3 $(G^3) = \{ a^n b^n c^n \mid n \geq 1\}$

Menentukan Grammar Sebuah Bahasa

12. Tentukan sebuah gramar regular untuk bahasa $L^1 = \{ a^n \mid n \geq 1\}$

<u>Jawab:</u>

$$P^1(L^1) = \{S \rightarrow aS \mid a\}$$

13. Tentukan sebuah gramar bebas konteks untuk bahasa:

L^2: himpunan bilangan bulat non negatif ganjil

<u>Jawab:</u>

Langkah kunci: digit terakhir bilangan harus ganjil.

Buat dua buah himpunan bilangan terpisah: genap (G) dan ganjil (J)

$$P^2(L^2) = \{S \rightarrow J \mid GS \mid JS,\ G \rightarrow 0 \mid 2 \mid 4 \mid 6 \mid 8,\ J \rightarrow 1 \mid 3 \mid 5 \mid 7 \mid 9\}$$

14. Tentukan sebuah gramar bebas konteks untuk bahasa:

L^3 = himpunan semua identifier yang sah menurut bahasa pemrograman Pascal dengan batasan: terdiri dari simbol huruf kecil dan angka, panjang identifier boleh lebih dari 8 karakter

<u>Jawab:</u>

Langkah kunci: karakter pertama identifier harus huruf.

Buat dua himpunan bilangan terpisah: huruf (H) dan angka (A)

$$P^3(L^3) = \{S \rightarrow H \mid HT,\ T \rightarrow AT \mid HT \mid H \mid A,\ H \rightarrow a \mid b \mid c \mid \ldots,\ A \rightarrow 0 \mid 1 \mid 2 \mid \ldots\}$$

15. Tentukan gramar bebas konteks untuk bahasa

$$L^4(G^4) = \{a^n b^m \mid n,m \geq 1,\ n \neq m\}$$

<u>Jawab:</u>

Langkah kunci: sulit untuk mendefinisikan $L^4(G^4)$ secara langsung. Jalan keluarnya adalah dengan mengingat bahwa $x \neq y$ berarti $x > y$ atau $x < y$.

$L^4 = L^A \cup L^B$, $L^A = \{a^n b^m \mid n > m \geq 1\}$, $L^B = \{a^n b^m \mid 1 \leq n < m\}$.

$P^A(L^A) = \{A \rightarrow aA \mid aC,\ C \rightarrow aCb \mid ab\}$, $Q(L^B) = \{B \rightarrow Bb \mid Db,\ D \rightarrow aDb \mid ab\}$

$P^4(L^4) = \{S \rightarrow A \mid B,\ A \rightarrow aA \mid aC,\ C \rightarrow aCb \mid ab,\ B \rightarrow Bb \mid Db,\ D \rightarrow aDb \mid ab\}$

16. Tentukan sebuah gramar bebas konteks untuk bahasa:

L^5 = bilangan bulat non negatif genap. Jika bilangan tersebut terdiri dari dua digit atau lebih maka nol tidak boleh muncul sebagai digit pertama.

<u>Jawab:</u>

Langkah kunci: Digit terakhir bilangan harus genap. Digit pertama tidak boleh nol. Buat tiga himpunan terpisah: bilangan genap tanpa nol (G), bilangan genap dengan nol (N), serta bilangan ganjil (J).

$$P^5(L^5) = \{S \rightarrow N \mid GA \mid JA,\ A \rightarrow N \mid NA \mid JA,\ G \rightarrow 2 \mid 4 \mid 6 \mid 8,\ N \rightarrow 0 \mid 2 \mid 4 \mid 6 \mid 8,\ J \rightarrow 1 \mid 3 \mid 5 \mid 7 \mid 9\}$$

Contoh lain:

1. Tata Bahasa Bebas Konteks

a) $S \rightarrow x$
b) $S \rightarrow y$
c) $S \rightarrow z$
d) $S \rightarrow S + S$

e) $S \rightarrow S - S$
f) $S \rightarrow S * S$
g) $S \rightarrow S / S$
h) $S \rightarrow (S)$

Tata bahasa diatas dapat membangkitkan *string*:

$$(x + y) * x - z * y / (x + x)$$

Penyelesaian:
S (symbol awal)
$\rightarrow$ S - S (rule 5)
$\rightarrow$ S * S - S (by rule 6, applied to the leftmost S)
$\rightarrow$ S * S - S / S (by rule 7, applied to the rightmost S)
$\rightarrow$ (S) * S - S / S (by rule 8, applied to the leftmost S)
$\rightarrow$ (S) * S - S / (S) (by rule 8, applied to the rightmost S)
$\rightarrow$ (S + S) * S - S / (S) (etc.)
$\rightarrow$ (S + S) * S - S * S / (S)
$\rightarrow$ (S + S) * S - S * S / (S + S)
$\rightarrow$ (x + S) * S - S * S / (S + S)
$\rightarrow$ (x + y) * S - S * S / (S + S)
$\rightarrow$ (x + y) * x - S * y / (S + S)
$\rightarrow$ (x + y) * x - S * y / (x + S)
$\rightarrow$ (x + y) * x - z * y / (x + S)
$\rightarrow$ (x + y) * x - z * y / (x + x)

2. Diberikan tata bahasa untuk bahasa berikut:

$$L = \{a^n b^m : n \neq 2m\}.$$

Penyelesaian
Tata bahasa untuk bahasa diatas adalah
$S \rightarrow aaSb \mid A \mid B \mid ab$
$A \rightarrow aA \mid a$
$B \rightarrow bB \mid b$

BAB VII
BENTUK NORMAL CHOMSKY

7.1. Pendahuluan

Gambar 7.1 Proses Pembentukan Bentuk Normal Chomsky

Bentuk normal Chomsky / Chomsky Normal Form (CNF) merupakan salah satu bentuk normal yang sangat berguna untuk tata bahasa bebas konteks (CFG). *Bentuk normal Chomsky* dapat dibuat dari sebuah tata bahasa bebas konteks yang telah mengalami penyederhanaan yaitu penghilangan produksi *useless*, unit, dan ε. Dengan kata lain, suatu tata bahasa bebas konteks dapat dibuat menjadi *bentuk normal Chomsky* dengan syarat tata bahasa bebas kontesk tersebut:

- Tidak memiliki produksi *useless*
- Tidak memiliki produksi unit
- Tidak memiliki produksi ε

Aturan produksi dalam *bentuk normal Chomsky* ruas kanannya tepat berupa sebuah terminal atau dua variabel. Misalkan: A → BC
A → b
B → a
C → BA | d

7.2. Pembentukan Bentuk Normal Chomsky

Langkah-langkah pembentukan *bentuk normal Chomsky* secara umum sebagai berikut:

- Biarkan aturan produksi yang sudah dalam *bentuk normal Chomsky*
- Lakukan penggantian aturan produksi yang ruas kanannya memuat simbol terminal dan panjang ruas kanan > 1
- Lakukan penggantian aturan produksi yang ruas kanannya memuat > 2 simbol variabel
- Penggantian-penggantian tersebut bisa dilakukan berkali-kali sampai akhirnya semua aturan produksi dalam *bentuk normal Chomsky*
- Selama dilakukan penggantian, kemungkinan kita akan memperoleh aturan-aturan produksi baru, dan juga memunculkan simbol-simbol variabel baru.

Bisa dilihat tahapan-tahapan tersebut pada gambar berikut:
Contoh, tata bahasa bebas konteks (kita anggap tata bahasa bebas konteks pada bab ini sudah mengalami penyederhanaan):
S → bA | aB
A → bAA | aS | a
B → aBB | bS | b

Aturan produksi yang sudah dalam *bentuk normal Chomsky:*
A → a
B → b

Dilakukan penggantian aturan produksi yang belum *bentuk normal Chomsky* ('=>' bisa dibaca berubah menjadi):
S → bA => S → P_1A
S → aB => S → P_1B
A → bAA => S → P_1AA => A → P_1P_3
A → aS => A → P_2S
B → aBB => B → P_2BB => B → P_2P_4
B → bS => B → P_1S

Terbentuk aturan produksi dan simbol variabel baru:
P_1 → b
P_2 → a

$P_3 \rightarrow AA$
$P_4 \rightarrow BB$

Hasil akhir aturan produksi dalam *brntuk normal Chomsky:*

$A \rightarrow a$	$B \rightarrow P_2P_4$
$B \rightarrow b$	$B \rightarrow P_1S$
$S \rightarrow P_1A$	$P_1 \rightarrow b$
$S \rightarrow P_2B$	$P_2 \rightarrow a$
$A \rightarrow P_1P_3$	$P_3 \rightarrow AA$
$A \rightarrow P_2S$	$P_4 \rightarrow BB$

Contoh, tata bahasa bebas konteks:
$S \rightarrow aB \mid CA$
$A \rightarrow a \mid bc$
$B \rightarrow BC \mid Ab$
$C \rightarrow aB \mid b$

Aturan produksi yang sudah dalam *bentuk normal Chomsky:*
$S \rightarrow CA$
$A \rightarrow a$
$B \rightarrow BC$
$C \rightarrow b$

Penggantian aturan produksi yang belum dalam *bentuk normal Chomsky:*
$S \rightarrow aB \Rightarrow S \rightarrow P_1B$
$A \rightarrow bc \Rightarrow S \rightarrow P_2P_3$
$B \rightarrow Ab \Rightarrow B \rightarrow A\ P_2$
$C \rightarrow aB \Rightarrow C \rightarrow P_1B$

Terbentuk aturan produksi dan simbol variabel baru:
$P_1 \rightarrow a$
$P_2 \rightarrow b$
$P_3 \rightarrow c$

Hasil akhir aturan produksi dalam *bentuk normal Chomsky:*
$S \rightarrow CA$
$A \rightarrow a$

$B \rightarrow BC$
$C \rightarrow b$
$S \rightarrow P_1B$
$S \rightarrow P_2P_3$
$B \rightarrow A\ P_2$
$C \rightarrow P_1B$
$P_1 \rightarrow a$
$P_2 \rightarrow b$
$P_3 \rightarrow c$

Contoh, tata bahasa bebas konteks:
$S \rightarrow aAB \mid ch \mid CD$
$A \rightarrow dbE \mid eEC$
$B \rightarrow ff \mid DD$
$C \rightarrow ADB \mid aS$
$D \rightarrow i$
$E \rightarrow jD$

Aturan produksi yang sudah dalam *bentuk normal Chomsky:*
$S \rightarrow CD$
$B \rightarrow DD$
$D \rightarrow i$

Penggantian aturan produksi:
$S \rightarrow aAB \Rightarrow S \rightarrow P_1P_2$
$S \rightarrow ch \Rightarrow S \rightarrow P_3P_4$
$A \rightarrow dbE \Rightarrow A \rightarrow P_5P_6$
$A \rightarrow eEC \Rightarrow A \rightarrow P_8P_9$
$B \rightarrow ff \Rightarrow B \rightarrow P_{10}P_{10}$
$C \rightarrow ADB \Rightarrow C \rightarrow AP_{11}$
$C \rightarrow aS \Rightarrow C \rightarrow P_1S$
$E \rightarrow jD \Rightarrow E \rightarrow P_{12}D$

Terbentuk aturan produksi baru:

$P_1 \rightarrow a$	$P_7 \rightarrow b$
$P_2 \rightarrow AB$	$P_8 \rightarrow e$
$P_3 \rightarrow c$	$P_9 \rightarrow EC$
$P_4 \rightarrow h$	$P_{10} \rightarrow f$
$P_5 \rightarrow d$	$P_{11} \rightarrow DB$
$P_6 \rightarrow P_7E$	$P_{12} \rightarrow j$

Hasil akhir dalam *bentuk normal Chomsky:*

$S \rightarrow CD$	$C \rightarrow AP_{11}$	$P_6 \rightarrow P_7E$
$B \rightarrow DD$	$C \rightarrow P_1S$	$P_7 \rightarrow b$
$D \rightarrow i$	$E \rightarrow P_{12}D$	$P_8 \rightarrow e$
$S \rightarrow P_1P_2$	$P_1 \rightarrow a$	$P_9 \rightarrow EC$
$S \rightarrow P_3P_4$	$P_2 \rightarrow AB$	$P_{10} \rightarrow f$
$A \rightarrow P_5P_6$	$P_3 \rightarrow c$	$P_{11} \rightarrow DB$
$A \rightarrow P_8P_9$	$P_4 \rightarrow h$	$P_{12} \rightarrow j$
$B \rightarrow P_{10}P_{10}$	$P_5 \rightarrow d$	

Algoritma CYK untuk Tata Bahasa Bebas Konteks

Algoritma CYK merupakan algoritma *parsing* dan keanggotaan (*membership)* untuk tata bahasa bebas konteks. Algortima ini diciptakan oleh J. Cocke, DH. Younger, dan T. Kasami. Syarat untuk penggunaan algortima ini adalah tata bahasa harus berada dalam *bentuk normal Chomsky* . Obyektif dari algortima ini adalah untuk menunjukkan apakah suatu *string* dapat diperoleh dari suatu tata bahasa.

Algoritma CYK sebagai berikut:

begin

1) for i:= 1 to n do
2) V_{i1}:= {A| A $\rightarrow$ a aturan produksi dimana simbol ke- i adalah a };
3) for j:= 2 to n do
4) for i:= 1 to (n-j+1) do
   begin
5) V_{ij}:=Ø;

6) for k:=1 to (j – 1) do

7) V_{ij}:= V_{ij} υ (A | A → BC adalah suatu produksi, dimana B di V_{ik} dan C di $V_{i+k,j-k}$ }

end

end

Penjelasan:

- *n* = panjang untai yang akan diperiksa, missal: untuk untai 'ada', *n* = | ada | =3
- *i* akan menyatakan kolom ke-
- *j* akan menyatakan baris ke-
- tahapan no (1) dan (2) untuk mengisi Tabel baris pertama kolom 1 – *n*
- no (3), interasi dari baris ke- 2 sampai *n*
- no (4), interasi untuk mengisi kolom 1 sampai (*n* – baris + *1*) pada suatu baris.
- no (5) inisialisasi V_{ij} dengan *Ø*
- no (6) dan no (7), interasi untuk memeriksa mana saja yang menjadi anggota V_{ij}

Lihat contoh kasus, dimana terdapat tata bahasa bebas konteks (simbol awal S):

S → AB | BC

A → BA | a

B → CC | b

C → AB | a

Periksalah apakah untai 'baaba' termasuk kedalam bahasa tersebut

Pertama – tama akan dibuat tabel untuk V_{ij} ($V_{kolom, baris}$) sebagai berikut:

	b	a	a	b	a
			i →		
	1	2	3	4	5
↓ 1					
2					
3					
4					
5					

Tabel diatas kita gunakan unruk mempermudah dalam menyelesaikan persoalan, *i* akan menyatakan kolom, *j* akan menyatakan baris. Diketahui *n* = 5. Dari Algoritma langkah (1) dan (2) bisa mengisi baris pertama pada tabel, sebagai berikut:

- Untuk V_{11}, kita periksa variabel yang bisa menurunkan 'b', dari B → b kita isi V_{11}= {B}
- Untuk V_{21}, kita periksa variabel yang bisa menurunkan 'a', dari A → a dan C → a kita isi V_{21}{A,C}
- Untuk V_{31}, kita periksa varibel yang bisa menurunkan 'a', dari A → a dan C → a kita isi V_{31}={A,C}
- Untuk V_{41}, kita periksa variabel yang bisa menurunkan 'b', dari B → b kita isi V_{41}={B}
- Untuk V_{51}, kita periksa variabel yang bisa menurunkan'a', dari A → a dan C → A kita isi V_{51}={A,C}

Dari hasil tersebut kita bisa tabel:

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

↓

Selanjutnya akan mengisi baris ke-2 sampai *n* sebagai berikut
Pada baris ke -2 (*k =1)*

- Untuk V_{12}, periksa V_{ik}- $V_{i+k,\ j-k}$, berarti V_{11}-V_{21}, yaitu B-A,C, variabel yang bisa menurunkan BA atau BC adalah S dan A, maka V_{12} kita isi {S, A}
- Untuk V_{22}, periksa V_{ik} – $V_{i+k,\ j-k}$, berarti V_{21}-V_{31}, yaitu A,C-A,C, variabel yang bisa menurunkan AA, AC, CA, atau CC adalah B maka V_{22} kita isi {B}

- Untuk $V_{32,}$ periksa V_{ik}-$V_{i+k,\ j-k}$, berarti V_{31}-V_{41} yaitu A, C-B, variabel yang bisa menurunkan AB atau CB adalah S dan C, maka V_{12} kita isi {S, C}
- Untuk V_{42}, periksa $V_{ik-Vi+k,\ j-k}$ berarti $V_{41-}V_{51}$, yaitu A,C-B, variabel yang bisa menurunkan AB atau CB adalah S dan C, maka V_{12} kita isi {S,A}

Dari hasil tersebut kita bisa mengisi tabel:

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

Pada baris ke –3 (*k* = 1 sampai 2):

- Untuk V_{13}, periksa $V_{ik-}V_{i+k,\ j-k}$, berarti $V_{11-}V_{22}$ & $V_{12-}V_{31}$, yaitu B-B & S,A-A,C, variabel yang bisa menurunkan BB, SA,SC,AA, atau AC adalah tidak ada, maka V_{13} kita isi ∅
- Untuk V_{23}, periksa $V_{ik-}V_{i+k,\ j-k}$, berarti $V_{21-}V_{32}$ & $V_{22-}V_{41}$, yaitu A,C-S,C & B-B, variabel yang bisa menurunkan AS, AC, CS, CC, atau BB adalah B , maka V_{23} kita isi {B}
- Untuk V_{33}, periksa $V_{ik-}V_{i+k,\ j-k}$, berarti $V_{31-}V_{42}$ & $V_{32-}V_{51,}$ yaitu A,C-S,A & S,C-A,C variabel yang bisa menurunkan AS, AA, CS, CA, SA, SC, CA, atau CC adalah B, maka V_{33} kita isi {B}

Dari hasil tersebut kita bsa mengisi tabel:

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

Pada baris ke –4 (k = 1 sampai 3):

- Untuk V_{14}, periksa V_{ik}-$V_{i+k,\ j-k}$, berarti V_{11}-V_{23} & V_{12}-V_{32} & V_{13}-$V_{41,}$ yaitu B-B & S,A-S,C & ∅-B, variabel yang bisa menurunkan BB, SS, SC, AS AC adalah tidak ada, maka V_{14} kita isi ∅
- Untuk V_{24}, periksa V_{ik}-$V_{i+k,\ j-k}$, berarti V_{21}-V_{33} & V_{22}-V_{42} & V_{23}-$V_{51,}$ yaitu A,C-B & B-S,A & B-S,A & B-A,C, variabel yang bisa menurunkan AC, AB, BS, BA, BC adalah S, C, A, maka V_{24} kita isi {S,A,C}

Dari hasil tersbut kita bisa mengisi tabel:

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

Pada baris ke –5 (k = 1 sampai 4)

- Untuk V_{15}, periksa V_{ik}-$V_{i+k,\ j-k}$, berarti V_{11}-V_{24} & V_{12}-V_{33} & V_{13}-V_{42} & V_{14}-V_{51} yaitu B-S,A,C & S,A-B & ∅-S,A & ∅-A,C, variabel yang bisa menurunkan BA, BC, SA, SC, SB, atau AB adalah A,S,C maka V_{15} kita isi {S,A,C}

Dari hasil tersbut bisa mengisi tabel:

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

Perhatikan, syarat suatu untai dapat diturunkan dari simbol awal, V1n memuat simbol awal. Terlihat pada tabel, simbol awal S termuat di V_{15}, maka untai ‘baaba’ dapat diturunkan oleh tata bahasa tersebut.

Kita bisa mencoba-coba untuk membuat pohon penurunan dari untai ‘baaba’, Lihat untuk contoh lain, terdapat tata bahasa bebas konteks:

S → AB | b
A → BA | a
B → AS | b

Periksalah apakah untai ‘aaab’ termasuk ke dalam bahasa tersebut, Pertama-tama kita akan membuat tabel untuk V_{ij} (Vkolom, baris) sebagai berikut:

	a	a	a	b
			i →	
	1	2	3	4
1				
2				
3				
4				

Diketahui *n* = 4. Dari algoritma langkah (1) dan(2) bisa mengisi baris pertama pada tabel, sebagai berikut:

- Untuk V_{11}, kita periksa variabel yang bisa menurunkan ‘a’, dari A → a kita isi V_{11} = {A}
- Untuk V_{21}, kita periksa variabel yang bisa menurunkan ‘a’, dari A → a kita isi V_{21} = {A}
- Untuk V_{31}, kita periksa variabel yang bisa menurunkan ‘a’, dari A → a kita isi V_{31} = {A}
- Untuk V_{41}, kita periksa variabel yang bisa menurunkan ‘b’, dari B → b dan S → b kita isi V_{41} = {S,B}

Dari haisl tersebut bisa mengisi tabel:

	a	a	a	b
			i →	
	1	2	3	4
j ↓ 1	A	A	A	S,B
2				
3				
4				

Selanjutnya akan mengisi baris ke –2 sampai *n* sebagai berikut: Pada baris ke –2 (k = 1):

- Untuk V_{12}, periksa $V_{ik-}V_{i+k,\ j-k}$, berarti $V_{11-}V_{21,}$ yaitu A-A, variabel yang bisa menurunkan AA adalah tidak ada, maka V_{12} diisi ∅
- Untuk V_{22}, periksa $V_{ik-}V_{i+k,\ j-k}$, berarti $V_{21-}V_{31}$,yaitu A-A, variabel yang bisa menurunkan AA adalah tidak ada, maka V_{22} diisi ∅
- Untuk V_{32}, periksa $V_{ik-}V_{i+k,\ j-k}$, berarti $V_{31-}V_{41}$,yaitu A,S-B, variabel yang bisa menurunkan AS atau AB adalah S dan B, maka V_{32} kita isi {S,B}

Dari hasil tersebut bisa mengisi tabel:

	a	a	a	b
			i →	
	1	2	3	4
j ↓ 1	A	A	A	S,B
2	∅	∅	S,B	
3				
4				

Pada baris ke –3 (k = 1 sampai 2)

- Untuk V_{13}, periksa $V_{ik-}V_{i+k,\ j-k}$, berarti $V_{11-}V_{22}$ & $V_{12-}V_{31}$, yaitu A-∅ & ∅-A, variabel yang bisa menurunkannya adalah tidak ada, maka V_{13} diisi ∅
- Untuk V_{23}, periksa $V_{ik-}V_{i+k,\ j-k}$, berarti $V_{21-}V_{32}$ & $V_{22-}V_{41}$, yaitu A-SB & ∅-SB, variabel yang bisa menurunkan AS atau AB adalah S dan B, maka V_{23} diisi {S,B}

Dari hasil tersebut bisa mengisi tabel:

	a	a	a	b
			i →	
	1	2	3	4
j ↓ 1	A	A	A	S,B
2	∅	∅	S,B	
3	∅	S,B		
4				

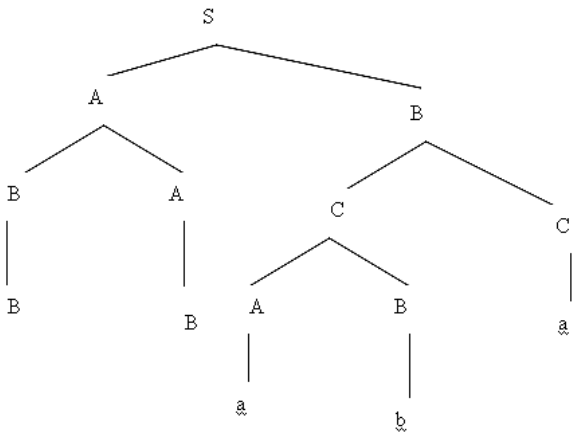
Pada baris ke –4 (k = 1 sampai 3):

- Untuk V_{14}, periksa V_{ik}-$V_{i+k,\ j-k}$, berarti V_{11}-V_{23} & V_{12}-V_{32} & V_{13}-V_{41}, yaitu A-SB & ∅-SB, variabel yang bisa menurunkan AS atau AB adalah S dan B, maka V_{14} di isi {S,B}

Dari hasil tersebut bisa mengisi tabel:

	a	a	a	b
			i →	
	1	2	3	4
j ↓ 1	A	A	A	S,B
2	∅	∅	S,B	
3	∅	S,B		
4	S,B			

Terlihat pada tabel, simbol awal S termuat di V_{14}, maka untai ‘aaab’ dapat diturunkan oleh tata bahasa tersebut.



Pohon Penurunan Untuk Untai „baaba“

7.3. Hierarki Chomsky

Tata bahasa (*grammar*) bisa didefinisikan secara formal sebagai kumpulan dari himpunan-himpunan variabel, simbol-simbol terminal, simbol awal, yang dibatasi oleh aturan-aturan produksi. Pada tahun 1959, seorang ahli bernama Noam Chomsky melakukan penggolongan tingkatan

bahasa menjadi empat, yang disebut dengan hierarki Chomsky. Penggolongan tersebut bisa dilihat pada tabel berikut.

Bahasa	**Mesin Otomata**	**Batasan Aturan Produksi**
Regular	*Finite State Automata* (FSA) meliputi *Deterministic Finite Automata* (FSA) & *Non Deterministic Finite Automata* (NDFA)	α adalah sebuah symbol variabel. β maksimal memiliki sebuah symbol variabel yang bila ada terletak diposisi paling kanan
Bebas konteks / *Context Free*	*Push Down Automata* (PDA)	α berupa sebua symbol variabel
Context sensitive	*Linier bounded Automata*	$\lvert \alpha \rvert \leq \lvert \beta \rvert$
Unrestricted / phase Structure / Normal Language	Mesin Turing	Tidak ada batasan

Secara umum tata bahasa dirumuskan sebagai:

$\alpha \rightarrow \beta$, yang berarti α menghasilkan β atau α menurunkan β.

Di mana α menyatakan simbol-simbol pada ruas kiri aturan produksi (sebelah kiri tanda '→') dan β menyatakan simbol-simbol pada ruas kanan aturan produksi (sebelah kanan tanda '→'). Simbol variabel / non terminal adalah simbol yang masih bisa diturunkan dan ditandai dengan huruf besar seperti A, B, C, dst. Simbol terminal adalah simbol yang sudah tidak bisa diturunkan dan ditandai dengan huruf kecil seperti a, b, c, dst.

Aturan produksi:

T→a (dibaca "T menghasilkan a")

E→T | T+E (dibaca "E menghasilkan T atau E menghasilkan T + E")

Simbol " | " dibaca 'atau'; digunakan untuk mempersingkat aturan produksi yang mempunyai ruas kiri yang sama.

Jadi penulisan aturan produksi E → T | T + E adalah singkatan dari 2 buah aturan produksi:

E → T E → T + E

Tabel: Contoh-Contoh Aturan Produksi

Bahasa	Contoh Aturan Produksi
Regular / tipe 1	A → def A → bc A → bcdE C → D
Context Free / tipe 2	B → CDeFg D → BcDe
Context Sensitive / tipe 1	D → ef (|D| < |ef|) E →λ (pengecualian)
Natural Language / tipe 0	Abc → deF

Dalam tata bahasa hierarki Chomsky terbagi menjadi empat jenis:

- Ketik 0: Tanpa batasan.
- Tipe 1: Tata bahasa CS.
- Tipe 2: tata bahasa CF.
- Tipe 3: Grammar linier yang memiliki produksi dari bentuk Xi → wXj atau Xj → w di mana Xi dan Xj adalah nonterminals dan w $\in \sum^*_T$, apa yang disebut grammar linear kanan.

Grammars of Types 1 dan 2 menghasilkan apa yang disebut CS-bahasa dan CF-bahasa, masing-masing, keluarga yang sesuai bahasa dilambangkan oleh CS dan CF. Bahasa yang dihasilkan oleh tata bahasa Tipe 0 disebut bahasa yang bisa dihitung komputabel (Bahasa-CE), keluarga yang terkait dilambangkan oleh CE. Nama berasal dari Fakta bahwa kata-kata dalam bahasa CE dapat didaftar secara algoritme, yaitu, ada algoritme yang menjalankan output tanpa batas persis semua kata-kata dari bahasa satu per satu. Seperti Algoritma sebenarnya diperoleh melalui mekanisme derivasi dari tata bahasa. Di sisi lain tangan, bahasa selain bahasa-CE tidak dapat dicantumkan secara algoritme dengan cara ini. Ini adalah karena definisi algoritma yang formal dan diterima secara umum.
Bahasa yang dihasilkan oleh tata bahasa Tipe 3 sudah tidak asing lagi:
Teorema 9. Keluarga bahasa yang dihasilkan oleh tata bahasa
Tipe 3 adalah keluarga R dari bahasa reguler.

Ada tentu saja tata bahasa kiri-linier terkait di mana produksi adalah dari bentuk Xi → Xjw dan Xj → w. Tipe 3 juga bisa didefinisikan dengan baik menggunakan ini.
Hierarki Chomsky dapat dianggap sebagai hierarki keluarga bahasa: R ⊂ CF ⊂ CS ⊂ CE.

Seperti disebutkan di atas, bahasa Lpal dari semua palindrom di atas alfabet setidaknya bersanding dua simbol adalah CF tetapi tidak teratur, menunjukkan bahwa inklusi pertama adalah tepat. Yang lain inklusi juga benar, seperti yang akan terlihat selanjtnya.

Bahasa reguler ditutup di bawah banyak operasi pada bahasa, yaitu, operasi pada bahasa reguler selalu menghasilkan bahasa reguler. Operasi semacam itu termasuk misalnya operasi set-theoretic, concatenation, concatenation closure, dan mirror image. Lain kelompok bahasa dalam hierarki Chomsky ditutup di bawah beberapa operasi bahasa. Ini sebenarnya membuat mereka unit klasifikasi alami: kelompok yang lebih besar selalu mengandung bahasa yang entah bagaimana berbeda secara radikal, bukan hanya bahasa yang diperoleh dari bahasa di keluarga yang lebih kecil dengan beberapa operasi umum. Keluarga selain R tidak demikian ditutup bahkan di bawah semua operasi di atas, di persimpangan tertentu dan komplementasi merepotkan.

Tata bahasa dapat selalu diganti oleh tata bahasa dari jenis yang sama yang dihasilkan bahasa yang sama dan tidak memiliki terminal di sisi kiri produksi. Jika grammar awal adalah G = ($\sum_N$, $\sum_T$, X0, P), maka grammar yang baru adalah G′ =($\sum'_N$, $\sum_T$, X0, P′) di mana $\sum'_N = \sum_N \cup \sum'_T$, $\sum'_T = \{a' \mid a \in \sum_T\}$

($\sum'_T$ adalah "alfabet bayangan" terputus dari $\sum_T$), dan P′ diperoleh dari P dengan mengubah masing-masing simbol terminal a di setiap produksi ke simbol “prima” yang sesuai a′, dan menambahkan penghentian produksi a′ → a.

Setiap keluarga dalam hierarki Chomsky ditutup di bawah operasi ∪, rentetan, ∗ dan +. Kasus kelopmok R sudah ditangani. Jika bahasa L dan L ′ adalah dihasilkan oleh tata bahasa G = ($\sum_N$,$\sum_T$, X0, P) dan G ′ = ($\sum'_N$,$\sum'_T$, X'_0, P ′) dari jenis yang sama, maka dapat diasumsikan pertama

itu $\sum_N \cap \sum'_N = \emptyset$, dan kedua yang tersisa sisi tangan produksi tidak mengandung terminal (oleh Lemma di atas). $L \cup L'$ kemudian dihasilkan oleh tata bahasa $H = (\Delta_N, \Delta_T, Y_0, Q)$ dari jenis yang sama di mana $\Delta_N = \sum_N \cup \sum'_N \cup \{Y_0\}$, $\Delta_T = \sum_T \cup \sum'_T$, Y_0 adalah nonterminal baru, dan Q diperoleh dengan cara berikut:

1. Ambil semua produksi dalam P dan P ′.
2. Jika jenisnya adalah Tipe 1, hapus produksi $X_0 \rightarrow \Lambda$ dan $X'_0 \rightarrow \Lambda$ (jika ada).
3. Tambahkan produksi $Y_0 \rightarrow X_0$ dan $Y_0 \rightarrow X'_0$.
4. Jika jenisnya adalah Tipe 1 dan Λ ada di L atau L′, tambahkan produksi Y0 →Λ.

LL′ pada gilirannya dihasilkan oleh Tata bahasa H ketika butir 3. dan 4. diganti oleh 3'. Tambahkan produksi $Y_0 \rightarrow X_0X'_0$. Jika jenisnya adalah Tipe 1 dan ada di L (resp. L′), tambahkan produksi $Y_0 \rightarrow X'_0$ (resp. $Y_0 \rightarrow X_0$).

4'. Jika jenisnya adalah Tipe 1 dan muncul di L dan L′, tambahkan produksi $Y_0 \rightarrow \Lambda$.

Perhatikan bagaimana sangat penting untuk membuat dua asumsi di atas, sehingga derivasi yang berdekatan tidak saling mengganggu grammars of Types 0 dan 1.

Jika G adalah Tipe 2, maka L* dihasilkan oleh tata bahasa

$$K = (\sum_N \cup \{Y_0\}, \sum_T, Y_0, Q)$$

di mana Q diperoleh dari P dengan menambahkan produksi

$$Y_0 \rightarrow \Lambda \text{ dan } Y_0 \rightarrow Y_0X_0.$$

L^+ dihasilkan jika produksi $Y_0 \rightarrow \Lambda$ diganti dengan $Y_0 \rightarrow X_0$. Untuk Tipe 1 konstruksinya sedikit lebih terlibat. Jika G adalah Tipe 1, baru lagi Y1 nonterminal ditambahkan, dan Q diperoleh sebagai berikut: Hapus dari P (mungkin) produksi $X_0 \rightarrow \Lambda$, dan tambahkan produksi Y0 $\rightarrow \Lambda$, $Y_0 \rightarrow X_0$ dan $Y_0 \rightarrow Y_1X_0$.

Kemudian, untuk setiap terminal a, tambahkan produksi $Y_1a \rightarrow Y_1X_0a$ dan $Y_1a \rightarrow X_0a$.

L^+ pada gilirannya dihasilkan jika produksi $Y_0 \rightarrow \Lambda$ dihilangkan (kapanpun diperlukan). Catatan betapa pentingnya lagi untuk terminal untuk tidak muncul di sisi kiri dari produksi, untuk mencegah derivasi yang berdekatan dari campur satu sama lain. Memang, derivasi baru hanya dapat dimulai ketika yang berikutnya sudah dimulai dengan terminal.
Untuk Tipe 0 konstruksinya cukup mirip dengan Tipe 1.

Hasil penutupan tambahan yang cukup mudah dilihat adalah setiap keluarga dalam hierarki Chomsky ditutup di bawah gambar cermin bahasa. Ada keluarga bahasa selain yang ada dalam hierarki Chomsky itu, misalnya.

- Bahasa yang dihasilkan oleh tata bahasa linier, yang disebut bahasa linear (keluarga LIN),
- Melengkapi bahasa-bahasa CE, yang disebut co-CE-bahasa (keluarga co − Cϵ), dan
- Persimpangan Cϵ dan co − Cϵ, yang disebut bahasa computabel (keluarga C).

Bahasa yang bisa dikomputasi adalah bahasa-bahasa yang masalah keanggotaannya adalah rithmically decidable, cukup dengan mencantumkan kata-kata dalam bahasa dan komplemennya secara bergiliran, dan memeriksa daftar mana yang akan berisi kata masukan yang diberikan.

Tidak perlu memasukkan dalam keluarga bahasa di atas keluarga bahasa dihasilkan oleh grammar yang bertambah panjang, karena sama dengan CS:

Untuk setiap grammar yang bertambah panjang, ada CS-grammar yang menghasilkan bahasa yang sama.

Mari mempertimbangkan kasus di mana dalam tata bahasa $G = (\sum_N, \sum_T, X0, P)$ hanya ada satu peningkatan panjang produksi $p \rightarrow q$ bukan dari bentuk yang diizinkan, yaitu, tata bahasa $G' = (\sum_N, \sum_T, X0, P - \{p \rightarrow q\})$ adalah CS.

Oleh Lemma di atas, dapat diasumsikan bahwa tidak ada terminal di tangan kiri sisi produksi G. Mari kita tunjukkan bagaimana G ditransformasikan menjadi setara CS-grammar $G_1 = (\Delta_N, \sum_T, X_0, Q)$. Untuk

itu kami melambangkan $p = U_1 \cdots Um$ dan $q = V_1 \cdots V_n$ di mana masing-masing Ui dan Vj adalah nonterminal, dan $n \geq m \geq 2$. Kami mengambil nonterminal baru Z1,. . . , Zm dan biarkan $\Delta_N = \sum_N \cup \{Z_1, \ldots, Zm\}$. Q kemudian terdiri dari produksi P, tentu saja tidak termasuk $p \rightarrow q$, ditambah produksi baru yang mengurus tindakan yang terakhir ini produksi:

$$U_1U_2 \cdot \cdot Um \rightarrow Z_1U_2 \cdots Um,\ Z_1U_2U_3 \cdot \cdot Um \rightarrow Z_1Z_2U_3 \cdot \cdot Um,$$

$$\ldots Z_1 \cdot \cdot Zm - 1Um \rightarrow Z_1 \cdot \cdot Zm - 1ZmVm + 1 \cdots Vn,$$

$$Z_1Z_2 \cdot \cdot ZmVm + 1 \cdots Vn \rightarrow V_1Z_2 \cdots ZmVm + 1 \cdots Vn,$$

$$\ldots V1 \cdots Vm - 1ZmVm + 1 \cdots Vn \rightarrow V1 \cdots Vm - 1VmVm + 1 \cdots Vn.$$

(Di sini garis bawah hanya menunjukkan penulisan ulang.) G1 tata bahasa yang dihasilkan adalah CS dan menghasilkan bahasa yang sama dengan G. Perhatikan bagaimana seluruh urutan produksi baru seharusnya selalu diterapkan dalam derivasi. Memang, jika selama ini urutan beberapa produksi lainnya bisa diterapkan, maka mereka bisa diterapkan sudah sebelum urutan, atau setelah itu.

Sebuah *grammar* G yang bertambah secara umum kemudian ditransformasikan ke CS- yang setara. tata bahasa sebagai berikut. Kami mungkin sekali lagi membatasi diri pada kasus di mana tidak ada terminal di sisi kiri produksi. Mari kita tunjukkan dengan G ′ diperoleh dengan menghapus dari G semua produksi bukan dari bentuk yang diizinkan (jika ada). Yang dihapus produksi kemudian ditambahkan kembali satu per satu ke G′ mentransformasikannya setiap kali ke alent CS-grammar seperti yang dijelaskan di atas. Hasil akhir adalah CS-grammar yang menghasilkan bahasa yang sama dengan G.

7.4. Klasifikasi Chomsky

Menurut Noam Chomsky secara umum tata bahasa dikelompokan menjadi empat kelas. Kelas yang paling umum adalah tata bahasa tak beraturan (*unrestricted grammars*), bukan kelas prasa struktur (non phrase-structured). Tiga kelas yang lain adalah prasa struktur (*phase-structured*) yaitu: *contex-sensitive, contex-free*, dan regular.

Masing-masing tata bahasa mempunyai pengenal yaitu:

1. *Regular Grammar* ➔ *Finite State Automata(FSA)*
2. *Contex-free Grammar* ➔ *Push-Down Automata(PDA)*
3. *Contex-Sensitive Grammar* ➔ *Linear Bounded Automata (LBA)*
4. *Unrestricted Grammar* ➔ *Turing Machine* (TM)
5. Berdasarkan komposisi bentuk ruas kiri dan ruas kanan produksinya ($\alpha \rightarrow \beta$), Noam Chomsky mengklasifikasikan 4 tipe *grammar:*

   a. *Grammar* tipe ke-0: *Unrestricted Grammar* (UG)

      Ciri: $\alpha, \beta \in (V^T \mid V^N)^*, |\alpha| > 0$
      Batasan pada tipe ini ialah ruas kiri (α) minimal mempunyai satu simbol non terminal, sedangkan ruas kanan (β) tidak mempunyai batasan dalam aturan produksinya,
      Sebagai contoh:
      Abc ➔ aa
      Be ➔ aBaB

   b. *Grammar* tipe ke-1: *Context Sensitive Grammar* (CSG)

      Ciri: $\alpha, \beta \in (V^T \mid V^N)^*, 0 < |\alpha| \leq |\beta|$
      Batasan pada tipe ini yaitu, panjang *string* pada ruas kiri (α) lebih kecil atau sama dengan ruas kanan (β). Seperti terlihat pada
      Contoh berikut:
      AC ➔ Ed
      AB ➔ aDF

   c. *Grammar* tipe ke-2: *Context Free Grammar* (CFG)

      Ciri: $\alpha \in V^N, \beta \in (V^T \mid V^N)^*$
      Batasan pada tata bahasa bebas konteks yaitu ruas kiri (α) tepat mempunyai satu simbol non terminal, sedangkan ruas kanan (β) tidak dibatasi,
      Sebagai contoh:
      A ➔ Bcde
      B ➔ cfcgh

d. Grammar tipe ke-3: Regular Grammar (RG)

Ciri: $\alpha \in V^N$, $\beta \in \{V^T, V^T V^N\}$ atau $\alpha \in V^N$, $\beta \in \{V^T, V^N V^T\}$

Batasan untuk tipe regular adalah ruas kiri (α) tepat mempunyai satu simbol non terminal dan ruas kanan maksimal mempunyai satu simbol non terminal yang terletak paling kanan, jadi ruas kanan bisa memiliki simbol terminal yang tidak terbatas,

Sebagai contoh:

A ➔ aaa

B ➔ aaB

Ciri-ciri RG sering dituliskan sebagai**: $\alpha \in V^N$, $\beta \in \{a, bC\}$ atau $\alpha \in V^N$, $\beta \in \{a, Bc\}$**

Tipe sebuah *grammar* (atau bahasa) ditentukan dengan aturan sebagai berikut:

A language is said to be type-i (i = 0, 1, 2, 3) language if it can be specified by a type-i grammar but can't be specified any type-(i+1) grammar.

Contoh Analisa Penentuan *Type Grammar*

1. *Grammar* G^1 dengan $Q^1 = \{S \rightarrow aB, B \rightarrow bB, B \rightarrow b\}$. Ruas kiri semua produksinya terdiri dari sebuah V^N maka G^1 kemungkinan tipe CFG atau RG. Selanjutnya karena semua ruas kanannya terdiri dari sebuah V^T atau string $V^T V^N$ maka G^1 adalah RG.

2. *Grammar* G^2 dengan $Q^2 = \{S \rightarrow Ba, B \rightarrow Bb, B \rightarrow b\}$. Ruas kiri semua produksinya terdiri dari sebuah V^N maka G^2 kemungkinan tipe CFG atau RG. Selanjutnya karena semua ruas

kanannya terdiri dari sebuah V^T atau *string* $V^N V^T$ maka G^2 adalah RG.

3. *Grammar* G^3 dengan Q^3 = {S → Ba, B → bB, B → b}. Ruas kiri semua produksinya terdiri dari sebuah V^N maka G^3 kemungkinan tipe CFG atau RG. Selanjutnya karena ruas kanannya mengandung string $V^T V^N$ (yaitu bB) dan juga string $V^N V^T$ (Ba) maka G^3 bukan RG, dengan kata lain G^3 adalah CFG.

4. *Grammar* G^4 dengan Q^4 = {S → aAb, B → aB}. Ruas kiri semua produksinya terdiri dari sebuah V^N maka G^4 kemungkinan tipe CFG atau RG. Selanjutnya karena ruas kanannya mengandung string yang panjangnya lebih dari 2 (yaitu aAb) maka G^4 bukan RG, dengan kata lain G^4 adalah CFG.

5. *Grammar* G^5 dengan Q^5 = {S → aA, S → aB, aAb → aBCb}. Ruas kirinya mengandung *string* yang panjangnya lebih dari 1 (yaitu aAb) maka G^5 kemungkinan tipe CSG atau UG. Selanjutnya karena semua ruas kirinya lebih pendek atau sama dengan ruas kananya maka G^5 adalah CSG.

6. *Grammar* G^6 dengan Q^6 = {aS → ab, SAc → bc}. Ruas kirinya mengandung *string* yang panjangnya lebih dari 1 maka G^6 kemungkinan tipe CSG atau UG. Selanjutnya karena terdapat ruas kirinya yang lebih panjang daripada ruas kananya (yaitu SAc) maka G^6 adalah UG.

7.5. Latihan Soal

Latihan 1.

Berikut adalah tata bahasa bebas konteks yang sudah disederhanakan.

S ➔bA | aB
A ➔ bAA | aS | a
B ➔ aBB | bS | b

Dari aturan produksi diatas, aturan produksi yang sudah dalam bentuk normal Chomsky (CNF) adalah:

A ➔ a
B ➔ b

Aturan produksi yang belum dalam CNF adalah:

S ➔ bA ➔ S ➔ P1A
S ➔ aB ➔ S ➔ P2B
A ➔ bAA ➔ A ➔ P1AA ➔ A ➔ P1P3
A ➔ aS ➔ A ➔ P2S
B ➔ aBB ➔ B ➔ P2BB ➔ B ➔ P2P4
B ➔ bS ➔ B ➔ P1S

Terbentuk aturan produksi baru:

P1 ➔ b
P2 ➔ a
P3 ➔ AA
P4 ➔ BB

Hasil ahir aturan produksi dalam CNF adalah:

A ➔ a
B ➔ b
S ➔ P1A
S ➔ P2B
A ➔ P1P3
A ➔ P2S
B ➔ P2P4
B ➔ P1S
P1 ➔ b

P2 ➔ a
P3 ➔ AA
P4 ➔ BB

Latihan 2:
Berikut adalah tata bahasa bebas konteks yang sudah disederhanakan.
S ➔ aB | CA
A➔ a | bc
B ➔ BC | Ab
C ➔ aB | b

Dari aturan produksi diatas, aturan produksi yang sudah dalam bentuk normal Chomsky (CNF) adalah:
S ➔ CA
A ➔ a
B ➔ BC
C ➔ b

Aturan produksi yang belum dalam CNF adalah:
S ➔ aB ➔ S ➔ P1B
A ➔ bc ➔ A ➔ P2P3
B ➔ Ab ➔ B ➔ AP2
C ➔ aB ➔ C ➔ P1B

Terbentu aturan produksi baru:
P1 ➔ a
P2 ➔ b
P3 ➔ c

Hasil ahir aturan produksi dalam CNF adalah:
S ➔ CA
A ➔ a
B ➔ BC
C ➔ b
S ➔ P1B
A ➔ P2P3

B ➔ AP2
C ➔ P1B
P1 ➔ a
P2 ➔ b
P3 ➔ c

Latihan 3:
Tata bahasa bebas konteks
S ➔ aAB | ch | CD
A ➔ dbE | eEC
B ➔ ff | DD
C ➔ ADB |aS
D ➔ i
E ➔ jD

Aturan produksi yang sudah dalam bentuk CNF adalah:
S ➔ CD
B ➔ DD
D ➔ i

Penggantian aturan produksi:
S ➔aAB ➔ S ➔ P1 P2
S ➔ ch ➔ S ➔ P3 P4
A ➔ dbE ➔ A ➔ P5 P6
A ➔ eEC ➔ A ➔ P8 P9
B ➔ ff ➔ B ➔ P10 P10
C ➔ ADB ➔ C ➔ AP11
C ➔ aS ➔ C ➔ P1 S
E ➔ jD ➔ E ➔ P12 D

Aturan produksi baru:
P1 ➔ a
P2 ➔ AB
P3 ➔ c
P4 ➔ h

P5 ➔ d
P6 ➔ P7 E
P7 ➔ b
P8 ➔ e
P9➔ EC
P10 ➔ f
P11 ➔ DB
P12➔ j

Bentuk Normal Chomsky:
S ➔ CD
B ➔ DD
D ➔ i
S ➔ P1 P2
S ➔ P3 P4
A ➔ P5 P6
A ➔ P8 P9
B ➔ P10 P10
C ➔ AP11
C ➔ P1 S
E ➔ P12 D

Latihan 4:

1. Transformasikan tata bahasa bebas konteks berikut kedalam CNF.
   S ➔aSaA | A
   A➔abA | b
2. Gunakan algoritma CYK untuk memeriksa apakah untai ‘aabab’ termasuk dalam tata bahasa bebas konteks berikut.
   S ➔AB | BC
   A ➔BA | a
   B ➔CC | b
   C ➔AB | a

Latihan 5:

Sebuah CFG disebut BNF jika mengikuti produksi berikut:

A → a

B → BC

S → є

Dengan A, B, C adalah non terminal dan a adalah terminal

Langkah 1 Jika simbol awal S terjadi di beberapa sisi kanan, buat simbol awal baru S ' dan produksi baru S '→ S.

Langkah 2 Hapus produksi Null. (Menggunakan algoritma penghilangan produksi Null dibahas sebelumnya)

Langkah 3 Hapus unit produksi. (Menggunakan algoritma penghilangan produksi Unit dibahas sebelumnya)

Langkah 4 Ganti setiap produksi A → B1... Bn di mana n> 2 dengan A → B1C di mana C → B2... Bn. Ulangi langkah ini untuk semua produksi yang memiliki dua atau lebih simbol di dalam sisi kanan.

Langkah 5 Jika sisi kanan dari setiap produksi dalam bentuk A → aB di mana a adalah terminal dan A, B non-terminal, maka produksi digantikan oleh A → XB dan X → Sebuah. Ulangi langkah ini untuk setiap produksi yang dalam bentuk A → aB.

Latihan 6:

Konversi CFG berikut menjadi CNF

S → ASA | aB,

A → B | S,

B → b | є

Penyelesaian

1) Karena S muncul dalam R.H.S, kita menambahkan keadaan baru S0 dan S0 → S ditambahkan ke produksi set dan itu menjadi:

   S0 → S

   S → ASA | aB

   A → B | S

   B → b | є

2) Sekarang dapat menghapus produksi nol:
$B \rightarrow \epsilon$ dan $A \rightarrow \epsilon$
Setelah menghapus $B \rightarrow \epsilon$, set produksi menjadi:
$S0 \rightarrow S$, $S \rightarrow ASA \mid aB \mid a$, $A \rightarrow B \mid S \mid \epsilon$, $B \rightarrow b$
Setelah menghapus $A \rightarrow \epsilon$, set produksi menjadi:
$S0 \rightarrow S$,
$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA \mid S$,
$A \rightarrow B \mid S$,
$B \rightarrow b$

3) Sekarang kita akan menghapus unit produksi.
Setelah menghapus $S \rightarrow S$, set produksi menjadi:
$S0 \rightarrow S$,
$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$,
$A \rightarrow B \mid S$,
$B \rightarrow b$

Setelah menghapus $S0 \rightarrow S$, set produksi menjadi:
$S0 \rightarrow ASA \mid aB \mid a \mid AS \mid SA$,
$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$
$A \rightarrow B \mid S$,
$B \rightarrow b$

Setelah menghapus $A \rightarrow B$, set produksi menjadi:
$S0 \rightarrow ASA \mid aB \mid a \mid AS \mid SA$,
$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$
$A \rightarrow S \mid b$
$B \rightarrow b$

Setelah menghapus $A \rightarrow S$, set produksi menjadi:
$S0 \rightarrow ASA \mid aB \mid a \mid AS \mid SA$,
$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$
$A \rightarrow b \mid ASA \mid aB \mid a \mid AS \mid SA$,
$B \rightarrow b$

4) Sekarang kita akan menemukan lebih dari dua variabel dalam R.H.S

Di sini,

S0 → ASA

S → ASA

A → ASA

Melanggar dua Non-terminal di R.H.S.

Maka kita akan menerapkan langkah 4 dan langkah 5 untuk mendapatkan set produksi akhir berikutnya yang ada di dalamnya CNF:

S0 → AX | aB | a | AS | SA

S → AX | aB | a | AS | SA

A → b | AX | aB | a | AS | SA

B → b

X → SA

5) Kita harus mengubah produksi

S0 → aB

S → aB

A → aB

Dan set produksi akhir menjadi:

S0 → AX | YB | a | AS | SA

S → AX | YB | a | AS | SA

A → b | AX | YB | a | AS | SA

B → b

X → SA

Y → a

Latihan 6:

Tata bahasa bebas konteks (kita anggap tata bahasa bebas konteks pada bab ini sudah mengalami penyederhanaan):

S → bA | aB

A → bAA | aS | a

B → aBB | bS | b

Aturan produksi yang sudah dalam bentuk normal Chomsky:
A → a
B → b

Dilakukan penggantian aturan produksi yang belum bentuk normal Chomsky ('=>' bisa dibaca berubah menjadi):
S →_ bA => S →P1A
S → aB => S → P2B
A → bAA => A_ P1AA => A→ P1P3
A → aS => A → P2S
B → aBB => B → P2BB => B → P2P4
B → bS => B → P1S

Terbentuk aturan produksi dan simbol variabel baru:
P1 → b
P2 → a
P3 →AA
P4 → BB

Hasil akhir aturan produksi dalam brntuk normal Chomsky:
A →a
B → b
S → P1A
S → P2B
A→ P1P3
A →_ P2S
B → P2P4
B → P1S
P1 → b
P2 → a
P3 → AA
P4 → BB

Terbentuk aturan produksi dan simbol variabel baru:
P1 → a
P2 → b

P3 → c

Hasil akhir aturan produksi dalam bentuk normal Chomsky:

S → CA
A → a
B → BC
C → b
S → P1B
S →P2P3
B →A P2
C → P1B
P1 → a
P2 → b
P3 → c

BAB VIII
PENGHILANGAN REKURSIF KIRI

8.1. Pendahuluan

Gambar 8.1 Tahapan Penghilangan Rekursif Kiri

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

8.2. Aturan Produksi Rekursif

Aturan Produksi yang rekursif memilki ruas kanan (hasil produksi) yang memuat simbol variabel pada ruas kiri. Sebuah aturan produksi dalam bentuk: $A \rightarrow \beta A$, merupakan aturan produksi yang rekursif kanan, $\beta=(V \cup T)^*$ atau kumpulan simbol variabel dan terminal.

Aturan produksi berikut adalah aturan produksi yang rekursif,

$S \rightarrow dS$

$A \rightarrow Ab$

$B \rightarrow adB$

Setiap hasil produksi (ruas kanan) mengandung variabel yang sama dengan ruas kiri. Aturan produksi yang rekursif dibagi menjadi dua, yaitu rekursif kiri dan rekursif kanan. Aturan produksi dikatakan rekursif kiri jika pada awal hasil produksi mengandung variabel yang sama dengan ruas kiri.

Bentuk umum aturan produksi rekursif kiri:

$$A \rightarrow A\beta \qquad ; \lambda = (V \cup T)^*$$

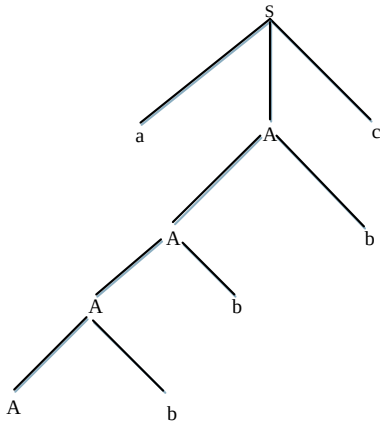
Sebagai contoh:
S → Sa
A → Ab
B → Bad

Aturan produksi dikatakan rekursif kanan jika pada akhir hasil produksi mengandung variabel yang sama dengan ruas kiri.
Bentuk umum aturan produksi rekursif kanan:

$$A \rightarrow \beta A \; ; \beta = (V \cup T)^*$$

Sebagai contoh:
S → aS
A → bdA
B → aB

Produksi yang rekursif kanan menyebabkan pohon penurunan tumbuh ke kanan, sedangkan produksi yang rekursif kiri menyebabkan pohon penurunan tumbuh ke kiri.
Sebagai contoh:
S → aAc
A → Ab | ε



Gambar 8.2. Pohon Penurunan Tumbuh Ke Kiri

Dalam banyak penerapan tata bahasa, rekursif kiri tak diinginkan karena mengakibatkan penurunan yang menghasilkan *loop*, sehingga perlu menghilangkan rekursif kiri dari aturan produksi.

Contoh lain aturan produksi yang rekursif kanan:

S → dS

B → adB

Produksi dalam bentuk: A → Aβ

Merupakan aturan produksi yang rekursif kiri, contohnya:

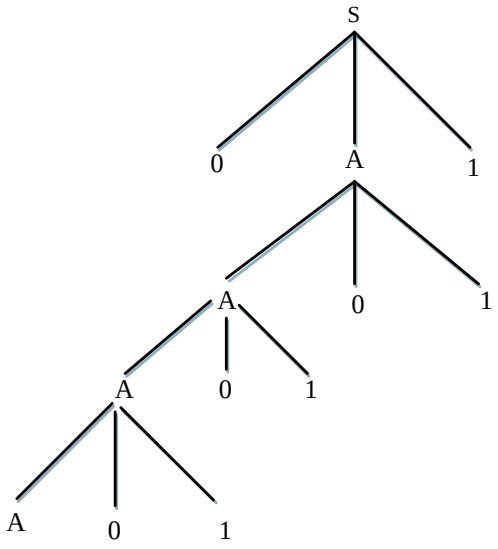
S ➔ Sd

B → Bad

Produksi yang rekursif kanan menyebabkan pohon penurunan tumbuh ke kanan, sebaliknya produksi yang rekursif kiri menyebabkan pohon penurunan tumbuh ke kiri. Bisa dilihat pohon penurunan pada gambar 8.3 dari tata bahasa bebas konteks dengan aturan produksi:

S → 0A1

A → A01 | ε



Gambar 8.3. Pohon Penurunan Sebuah CFG yang Rekursif Kiri

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

8.3. Tahapan Penghilangan Rekursif Kiri

Langkah-langkah penghilangan rekursif kiri:

1. Pisahkan aturan produksi yang rekursif kiri dan yang tidak,
   Contoh: Aturan produksi yang rekursif kiri:
   $A \rightarrow A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \mid ..... \mid A\alpha_n$
   Aturan produksi yang tidak rekursif kiri (termasuk produksi ε):

   $$A \rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \mid ..... \beta_m$$

2. Tentukan symbol-simbol $\alpha_1, \alpha_2, ... \alpha_n$, dan $\beta_1, \beta_2, ... \beta_m$ dari setiap aturan produksi yang memiliki simbol ruas kiri yang sama
3. Lakukan penggantian aturan produksi yang rekursif kiri, menjadi sebagai berikut:
   1) $A \rightarrow \beta_1 Z \mid \beta_2 Z \mid .... \beta_m Z$
   2) $Z \rightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid .... \alpha_n$
   3) $Z \rightarrow \alpha_1 Z \mid \alpha_2 Z \mid \alpha_3 Z \mid .... \alpha_n Z$

   Penggantian diatas dilakukan untuk setiap aturan produksi dengan simbol ruas kiri yang sama. Bisa muncul simbol variabel baru Z_1, Z_2 dan seterusnya, sesuai banyaknya variabel yang menghasilkan produksi yang rekursif kiri.
4. Hasil akhir berupa aturan produksi pengganti ditambah dengan aturan produksi semula yang tidak rekursif kiri.

Contoh: tata bahasa bebas konteks: $S \rightarrow Sab \mid aSc \mid dd \mid ff \mid Sbd$
Pertama-tama kita lakukan pemisahan aturan produksi
Aturan produksi yang rekursif kiri: $S \rightarrow Sab \mid Sbd$
Dari situ kita tentukan: Untruk simbol ruas kiri S: α_1=ab, α_2=bd
Aturan produksi yang tidak rekursif kiri: $S \rightarrow aSc \mid dd \mid ff$
Dari situ kita dapatkan: Untuk simbol ruas kiri S: β_1=aSc, β_2=dd, β_3=ff

Kita lakukan penggantian aturan produksi yang rekursif kiri:
Untuk yang memiliki simbol ruas kiri S:
S → Sab | Sbd, digantikan oleh:

i. S → $aScZ_1$ | dd Z_1 | ffZ_1
ii. Z_1 → ab | bd
iii. Z_1 → abZ_1 | bd Z_1

Hasil akhir setelah penghilangan rekursif kiri adalah:
S → aSc | dd | ff
S → $aScZ_1$ | dd Z_1 | ffZ_1
Z_1 → ab | bd
Z_1 → abZ_1 | bd Z_1

*Pada kasus diatas S adalah satu-satunya simbol variabel yang menghasilkan produksi rekursif kiri.
Contoh lain, terdapat tata bahasa bebas konteks:
S → Sab | Sb | cA
A → Aa | a | bd

Pertama-tama kita lakukan pemisahan aturan produksi. Aturan produksi yang rekursif kiri:
S → Sab | Sb
A → Aa

Dari situ kita tentukan: Untuk simbol ruas kiri S: α_1= ab, α_2 =b
Untuk simbol ruas kiri A: α_1 = a
Aturan produksi yang tidak rekursif kiri:
S → cA
A → a | bd

Dari situ kita dapatkan
Untuk simbol ruas kiri S: β_1 = cA
Untuk simbol ruas kiri A: β_1 = a, β_2 = bd
Kita lakukan penggantian aturan produksi yang rekursif kiri:
Untuk yang memiliki simbol ruas kiri S:
S → Sab | Sb, digantikan oleh:

i. S → cAZ_1

ii. $Z_1 \rightarrow ab \mid b$
iii. $Z_1 \rightarrow abZ_1 \mid bZ_1$

Untuk yang memiliki simbol ruas kiri A:
$A \rightarrow Aa$, digantikan oleh:
i. $A \rightarrow a\ Z_2 \mid bdZ_2$
ii. $Z_2 \rightarrow a$
iii. $Z_2 \rightarrow a\ Z_2$

Hasil akhir setelah penghilangan rekursif kiri adalah:
$S \rightarrow cA$
$A \rightarrow a \mid bd$
$S \rightarrow cAZ_1$
$Z_1 \rightarrow ab \mid b$
$Z_1 \rightarrow abZ_1 \mid bZ_1$
$A \rightarrow a\ Z_2 \mid bdZ_2$
$Z_2 \rightarrow a$
$Z_2 \rightarrow a\ Z_2$

- Perhatikan bahwa penghilangan rekursif kiri memunculkan simbol variabel baru, dan aturan produksi baru yang rekursif kanan.
  Contoh lain, terdapat tata bahasa bebas konteks:
  $S \rightarrow Sa \mid aAc \mid c \mid \varepsilon$
  $A \rightarrow Ab \mid ba$
  Pertama-tama kita lakukan pemisahan aturan produksi. Aturan produksi yang rekursif kiri:
  $S \rightarrow Sa$
  $A \rightarrow Ab$
  Dari situ kita tentukan:
  Untuk simbol ruas kiri S: $\alpha_1 = a$
  Untuk simbol ruas kiri A: $\alpha_1 = b$
  Aturan produksi yang tidak rekursif kiri:
  $S \rightarrow aAc \mid c \mid \varepsilon$
  $A \rightarrow ba$
  Dari situ kita dapatkan

Untuk simbol ruas kiri S:β_1 = aac, β_2= c, β_3 = ε
Untuk simbol ruas kiri A: β_1 = ba

- Perhatikan produksi ε termasuk produksi yang tidak rekursif kiri
  Lakukan penggantian aturan produksi yang rekursif kiri: Untuk yang memilki simbol ruas kiri S:
  S → Sa, digantikan oleh:
  i. S → aAcZ_1 | cZ_1 | Z_1
  ii. Z_1 → a
  iii. Z_1 → a Z_1

  Untuk yang memiliki simbol ruas kiri A:
  A → Ab, digantikan oleh:
  i. A → ba Z_2
  ii. Z_2 → b
  iii. Z_2 → bZ_2

  Hasil akhir setelah penghilangan rekursif kiri adalah:
  S → aAc | c | ε
  S → aAcZ_1 | cZ_1 | Z_1
  A → ba
  A → ba Z_2
  Z_1 → a
  Z_1 → a Z_1
  Z_2 → b
  Z_2 → b Z_2

8.4. Latihan Soal

Latihan 1:

Hilangkan rekursif kiri dari aturan produksi berikut!
S →Sa | aAc | c | ε
A →Ab | ba
Penyelesaian:
Aturan produksi rekursif kiri, S →Sa

$$\alpha 1 = a$$

Aturan produksi rekursif kiri, A →Ab

$$\alpha 1 = b$$

Aturan produksi tidak rekursif kiri, S → aAC | c | ε

$$\beta 1 = aAc \; ; \qquad \beta 2 = c \; ; \qquad \beta 3 = \varepsilon$$

Aturan produksi tidak rekursif kiri, A → ba

$$\beta 1 = ba$$

Pergantian aturan produksi rekursif kiri: S Sa digantikan oleh:
S → aAcZ1 | cZ1 | Z1
Z1 → a
Z1 → aZ1

Pergantian aturan produksi rekursif kiri: A →Ab digantikan oleh:
A → baZ2
Z2 → b
Z2 → bZ2

Hasil akhir setelah penghilangan rekursif kiri adalah:
S → aAC | c | ε
S →aAcZ1 | cZ1 | Z1
A → ba
A → baZ2
Z1→ a
Z1 → aZ1
Z2 → b
Z2 → bZ2

Latihan 2
Hilangkan rekursif kiri dari aturan produksi berikut!

S →Sab | aSc | dd | ff | Sbd

Penyelesaian: Aturan produksi tidak rekursif kiri:

S → aSc | dd | ff

Penghilangan rekursif kiri dari aturan produksi:

S →Sab | aSc | dd | ff | Sbd

Rekursif kiri	Tidak rekursif kiri
S →Sab | Sbd	S → aSc | dd | ff
$\alpha 1$= ab ; $\alpha 2$ = bd	$\beta 1$ = aSc ; $\beta 2$ = dd ; $\beta 2$ = ff
S → aScZ1 | dd Z1 | ff Z1 | aSc | dd | ff Z1 → ab | bd Z1→ abZ1 | bdZ1	

Latihan 3:

Hilangkan rekursif kiri dari aturan produksi berikut!

S →Sab | Sb | cA

A →Aa | a | bd

Penyelesaian:

Aturan produksi tidak rekursif kiri,

S → cA

A → a | bd

Penghilangan rekursif kiri dari aturan produksi:

S →Sab | Sb | cA

Rekursif kiri	Tidak rekursif kiri
S →Sab | Sb	S → cA
$\alpha 1$= ab ; $\alpha 2$ = b	$\beta 1$ = cA
S → cAZ1 | cA Z1 → ab | b Z_1→ abZ1 | bZ1	

Penghilangan rekursif kiri dari aturan produksi:

A →Aa | a | bd

Rekursif kiri	Tidak rekursif kiri
A →Aa	A → a | bd
$\alpha 1$= a	$\beta 1$ = a; β_2 = bd
A → aZ1 | bdZ_2 | a | bd Z_2 → a Z_2→ aZa_2	

Hasil akhir setelah
penghilangan rekursif kiri
adalah:

$S \rightarrow cAZ1 \mid cA$
$A \rightarrow aZ2 \mid bdZ2 \mid a \mid bd$
$Z1 \rightarrow abZ1 \mid bZ1 \mid ab \mid b$
$Z2 \rightarrow aZ2 \mid a$

Latihan 4:
$S \rightarrow AB$
$A \rightarrow aA \mid a$
$B \rightarrow bB \mid b$

Untai yang ingin diperoleh: 'aabbb' **Penyelesaian:**

1. Penurunan terkiri (leftmost derivation)

   $S \rightarrow AB \rightarrow aAB \rightarrow aaB \rightarrow aabB \rightarrow aabbB \rightarrow aabbb$

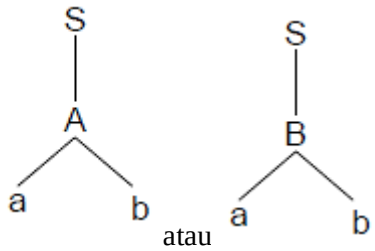
2. Penurunan terkanan (rightmost derivation)

   $S \rightarrow AB \rightarrow AbB \rightarrow AbbB \rightarrow Abbb \rightarrow aAbbb \rightarrow aabbb$

   Dalam pohon penurunan ada yang disebut Ambiguitas. Ambiguitas terjadi apabila terdapat lebih dari satu pohon penurunan yang berbeda tetapi untuk memperoleh untai yang sama.

Latihan 5:
$S \rightarrow A \mid B$
$A \rightarrow ab$
$B \rightarrow ab$

Untuk memperoleh untai yang sama yaitu ‘ab’, dapat digambarkan dua pohon penurunan yang berbeda tetapi untuk mencapai untai yang sama yaitu ‘ab’.



Latihan 6:

Terdapat tata bahasa bebas konteks:

S → Sab | Sb | cA

A →Aa | a | bd

Pertama-tama kita lakukan pemisahan aturan produksi Aturan produksi yang rekursif kiri:

S → Sab | Sb

A →Aa

Dari situ kita tentukan:

Untuk simbol ruas kiri S: $\alpha 1$= ab, $\alpha 2$ =b

Untuk simbol ruas kiri A: $\alpha 1$ = a

Aturan produksi yang tidak rekursif kiri:

S → cA

A → a | bd

Dari situ kita dapatkan

Untuk simbol ruas kiri S: $\beta 1$ = cA

Untuk simbol ruas kiri A: $\beta 1$ = a, $\beta 2$ = bd

Kita lakukan penggantian aturan produksi yang rekursif kiri: Untuk yang memiliki simbol ruas kiri S:

S → Sab | Sb, digantikan oleh:

i. S → cAZ1

ii. Z1 → ab | b
iii. Z1 → abZ1 | bZ1

Untuk yang memiliki simbol ruas kiri A:
A →Aa, digantikan oleh:

i. A → a Z2 | bdZ2
ii. Z2 → a
iii. Z2 → a Z2

Hasil akhir setelah penghilangan rekursif kiri adalah:
S → cA
A →a | bd
S → cAZ1
Z1 → ab | b
Z1 → abZ1 | bZ1
A → a Z2 | bdZ2
Z2 → a
Z2 → a Z2

*Perhatikan bahwa penghilangan rekursif kiri memunculkan simbol variabel baru, dan aturan produksi baru yang rekursif kanan.

Latihan 7:
Terdapat tata bahasa bebas konteks:
S → Sa |aAc | c | ε
A →Ab | ba

Pertama-tama kita lakukan pemisahan aturan produksi Aturan produksi yang rekursif kiri:
S → Sa
A →Ab

Dari situ ditentukan:
Untuk simbol ruas kiri S: α1 = a
Untuk simbol ruas kiri A: α1 = b
Aturan produksi yang tidak rekursif kiri:
S → aAc | c | ε

A → ba

Dari situ didapatkan untuk simbol ruas kiri S:$\beta 1$ = aAc, $\beta 2$= c, $\beta 3$ = ε untuk simbol ruas kiri A: $\beta 1$ = ba
Perhatikan produksi ε termasuk produksi yang tidak rekursif kiri
Kita lakukan penggantian aturan produksi yang rekursif kiri:
Untuk yang memilki simbol ruas kiri S:
S → Sa, digantikan oleh:

i. S → aAcZ1 | cZ1 | Z1
ii. Z1 → a
iii. Z1 → a Z1

Untuk yang memiliki simbol ruas kiri A:
A →Ab, digantikan oleh:

i. A → ba Z2
ii. Z2 → b
iii. Z2 → bZ2

Hasil akhir setelah penghilangan rekursif kiri adalah:
S → aAc | c | ε
S → aAcZ1 | cZ1 | Z1
A → ba
A → ba Z2
Z1→ a
Z1→ a Z1
Z2→ b
Z2 → b Z2

Tugas:
Lakukan penghilangan rekursif kiri pada tata bahasa bebas konteks berikut!

1. A → Aa | aBc
2. A → Aa | Ab | cd
   B → BAa | A | c

3. Gambarkan pohon penurunan untuk memperoleh string ‘bbabbaab’ dari tata bahasa bebas konteks berikut:
   $S \rightarrow AA$
   $A \rightarrow AAA \mid a \mid bA \mid Ab$
4. Buktikan bahwa tata bahasa bebas konteks berikut ambigu:
   $S \rightarrow SaS \mid SbS \mid c$
   Untuk untai: ‘cacbc’

BAB IX
PENYEDERHANAAN TATA BAHASA BEBAS KONTEKS

9.1. Pendahuluan

Gambar 9.1. Tahapan penyederhanaan CFG

Dalam CFG, dapat terjadi semua aturan produksi dan simbol tidak diperlukan untuk derivasi string. Selain itu, dapat trjadi beberapa produksi nol dan produksi unit. Produksi useless, unit, dan H harus dihilangkan secara bersamaan dari tata bahasa bebas konteks. Urutan penghilangan Produksi useless, unit, dan H adalah seperti gambar 9.1.

Penyederhanaan Tata bahasa bertujuan untuk melakukan pembatasan sehingga tidak menghasilkan pohon penurunan yang memiliki kerumitan yang tidak perlu atau aturan produksi yang tidak berarti. Eliminasi produksi dan simbol ini disebut penyederhanaan CFG.

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

Contoh:

$S \rightarrow A$
$A \rightarrow B$
$B \rightarrow C$
$C \rightarrow D$
$D \rightarrow a \mid A$

Berdasarkan aturan produksi diatas terlihat beberapa aturan yang sebetulnya tidak perlu sebab akhirnya tetap berujung pada $S \rightarrow a$, aturan

produksi D → A juga menyebabkan kerumitan karena akan kembali lagi ke atas A → B.

Penyederhanaan pada dasarnya terdiri dari langkah-langkah berikut:

1. Pengurangan CFG
2. Penghapusan Unit Produksi
3. Penghapusan Produksi Null

CFG berkurang dalam dua tahap:

Tahap 1 : Penurunan tata bahasa yang setara, G ', dari CFG, G, sehingga masing-masing variabel memperoleh beberapa string terminal.

Prosedur Derivasi:

Langkah 1 : Sertakan semua simbol, W1, yang memperoleh beberapa terminal dan menginisialisasi i = 1.

Langkah 2 : Sertakan semua simbol, Wi + 1, yang memperoleh Wi.

Langkah 3 : Increment i dan ulangi Langkah 2, sampai Wi + 1 = Wi.

Langkah 4 : Sertakan semua aturan produksi yang memiliki Wi di dalamnya.

Tahap 2 : Turunan dari tata bahasa yang setara, G ", dari CFG, G', sedemikian rupa sehingga masing-masing simbol muncul dalam bentuk sentensial.

Prosedur Derivasi:

Langkah 1 : Sertakan simbol awal di Y1 dan inisialisasi i = 1.

Langkah 2 : Sertakan semua simbol, Yi + 1, yang dapat diturunkan dari Yi dan mencakup semua aturan produksi yang telah diterapkan.

Langkah 3 : Increment i dan ulangi Langkah 2, sampai Yi + 1 = Yi.

Cari penyederhanaan tata bahasa G, yang memiliki aturan produksi, P: S →AC | B

A → a

C → c | BC

E → aA | e

Penyelesaian

Tahap 1

T = { a, c, e }

W1 = { A, C, E } dari aturan A → a, C → c dan E →aA

W2 = { A, C, E } U { S } dari aturan S →AC

W3 = { A, C, E, S } U Φ

Sejak W2 = W3, dapat menurunkan G' sebagai:

G' = { { A, C, E, S }, { a, c, e }, P, {S}}dimana

P: S →AC,

A →a,

C →c ,

E →aA | e

Tahap 2:

Y1 = { S }

Y2 = { S, A, C } dari aturan S → AC

Y3 = { S, A, C, a, c } dari aturan A → a and C → c

Y4 = { S, A, C, a, c }

Since Y3 = Y4, dapat menurunkan G" sebagai:

G" = { { A, C, S }, { a, c }, P, {S}}

dimana P: S → AC

A → a

C → c

Contoh 1:

S → AB | a

A→a

- Aturan produksi S → AB tidak berarti karena B tidak memiliki penurunan

  Contoh 2:

  S→A

  A→B

  B→C

  C→D

D → a | A

- Memiliki kelemahan terlalu panjang jalannya padahal berujung pada S → a,
- Produksi D → A juga menyebabkan kerumitan.

Cara Penyederhanaan:

1. Penghilangan produksi useless (tidak berguna)
2. Penghilangan produksi unit
3. Penghilangan produksi ε

9.2. Penghilangan Produksi *Useless*

Produksi useless didefinisikan sebagai produksi yang memuat simbol variabel yang tidak memiliki penurunan yang akan menghasilkan terminalterminal. Produksi ini tidak berguna karena bila diturunkan tidak akan pernah selesai (masih ada variabel yang tersisa). Selain itu, variabel yang memiliki penurunan berupa terminal atau terminal- terminal, tapi tidak dapat dicapai dari S, maka variabel tersebut juga termasuk useless.

Di sini produksi *useless* didefinisikan sebagai:

- Produksi yang memuat symbol variabel yang tidak memiliki penurunan yang akan menghasilkan terminal-terminal seluruhnya.
- Produksi yang tidak akan pernah dicapai dengan penurunan apapun dari simbol awal, sehingga produksi itu redundan (berlebih)

Contoh:

S → aSa | Abd | Bde

A → Ada

B→ BBB | a

Maka

1) Simbol variabel A tidak memiliki penurunan yang menuju terminal, sehingga bisa dihilangkan
2) Konsekuensi no (1), aturan produksi S → Abd tidak memiliki penurunan

Penyederhanaan menjadi:

S→aSa | Bde

B→ BBB | a
Contoh:
S→ Aa | B
A→ab | D
B→ b | E
C→ bb
E→ aEa
Maka:

1) Aturan produksi A → D, simbol variabel D tidak memiliki penurunan.
2) Aturan produksi C → bb, Penurunan dari simbol S, dengan jalan manapun tidak akan pernah mencapai C
3) Simbol variabel E tidak memiliki aturan produksi yang menuju terminal
4) Konsekuensi no (3) Aturan produksi B → E, simbol variabel E tidak memiliki penurunan.

Maka produksi yang useless:
A → D
C → bb
E → aEa
B → E
Penyederhanaannya menjadi:
S → Aa | B
A → ab
B → b
Contoh:
S → aAb | cEB
A → dBE | eeC
B → ff
C → ae
D → h

Analisa:

1) Aturan produksi S → cEB, A → dBE dapat dihilangkan (E tidak memiliki penurunan)
2) Aturan produksi D → h, redundan

Sisa aturan produksi

S → aAb

A → eeC

B → ff

C → ae

Analisis lagi

B → ff juga redundan,

Hasil penyederhanaan menjadi:

S → aAb

A → eeC

C → ae

Contoh lain lagi:

S → aB

A → bcD | dAC

B → e | Ab

C → bCb | adF | ab

F → cFB

Analisis

1) Aturan produksi A → bcD, variabel D tidak memiliki penurunan
2) Konsekuensi no (1), simbol variabel A tidak memiliki penurunan yang menuju terminal (tinggal A → dAC)
3) Konsekuensi no (2), B → Ab tidak memiliki penurunan
4) Simbol variabel F tidak memiliki penurunan yang menuju terminal

Konsekuensi no (4), C → adF tidak memiliki penurunan

Setelah disederhanakan menjadi:

S → aB

B → e

C → bCb | ab

Contoh lain lagi:
S → aBD
B → cD | Ab
D → ef
A → Ed
F → dc
Analisa

1) Aturan produksi A → Ed, E tidak memiliki penurunan
2) Aturan produksi F → dc, redundan

Sisa aturan produksi:
S → aBD
B → cD | Ab
D → ef
Analisa lagi
B → Ab, A tidak memiliki penurunan.
Hasil penyederhanaan:
S → aBD
B → cD
D → ef
Contoh lain:
S → Abc | ab
A → AAA | ε
Aturan produksi setelah disederhanakan:
S → Abc | ab
A → AAA | ε

Ingat A → ε juga harus diperhitungkan

PRINSIP
Setiap kali melakukan penyederhanaan diperiksa lagi aturan produksi yang tersisa, apakah semua produksi yang *useless* sudah hilang.

9.3. Penghilangan Produksi Unit

Produksi unit adalah aturan produksi yang menghasilkan satu variabel saja. Misal A → B. Keberadaan aturan produksi ini memperpanjang aturan produksi secara keseluruhan. Untuk mempersingkat aturan produksi, kita dapat melakukan penyederhanaan.

- Produksi dimana ruas kiri dan kanan aturan produksi hanya berupa satu simbol variabel, misalkan: A → B, C → D.
- Keberadaannya membuat tata bahasa memiliki kerumitan yang tak perlu.
- Penyederhanaan dilakukan dengan melakukan penggantian aturan produksi unit.

Contoh:
S → Sb
S → C
C → D
C → ef
D → dd

Dilakukan penggantian berturutan mulai dari aturan produksi yang paling dekat menuju ke penurunan terminal-terminal ('=>' dibaca 'menjadi'):

- C → D => C → dd
- S → C => S → dd | ef

Sehingga aturan produksi setelah penyederhanaan:
S → Sb
S → dd | ef
C → dd
C → ef
C → dd

Contoh lain:
S → A
S → Aa
A → B
B → C

B → b
C → D
C → ab
D → b

Penggantian yang dilakukan:

- C → D => C → b
- B → C => B → b | ab, karena B → b sudah ada, maka cukup dituliskan B → ab
- A → B => A → ab | b
- S → A => ab | b

Sehingga aturan produksi setelah penyederhanaan:
S → ab | b
S → Aa
A → ab | b
B → ab
B → b
C → b
C → ab
D → b

Contoh lain:
S → Cba | D
A → bbC
B → Sc | ddd
C → eAn | f | C
D → E | SABC
E → gh

Penggantian yang dilakukan:

- D → E menjadi D → gh
- C → C , kita hapus
- S → D menjadi S → gh | SABC

Sehingga aturan produksi setelah penyederhanaan:
S → Cba | gh | SABC
A → bbC
B → Sc | ddd
C → eA | f
D → gh | SABC
E → gh

9.4. Penghilangan Produksi ε

Produksi ε adalah produksi dalam bentuk α → ε atau bisa dianggap sebagai produksi kosong (*empty*). Penghilangan produksi ε dilakukan dengan melakukan penggantian produksi yang memuat variabel yang bisa menuju produksi ε, atau biasa disebut *nullable.*
Prinsip penggantiannya bisa dilihat kasus berikut:
S → bcAd
A → ε

A *nullable* serta A → ε satu-satunya produksi dari A, maka variabel A bisa ditiadakan, hasil penyederhanaan tata bahasa bebas konteks menjadi: S → bcd
Tetapi bila kasusnya: S → bcAd
A → bd | ε

A *nullable,* tapi A → ε bukan satu-satunya produksi dari A, maka hasil penyederhanaan:
S → bcAd | bcd
A → bd

Contoh lagi, terdapat tata bahasa bebas konteks:
S → Ab | Cd
A → d
C → ε

Variabel yang *nullable* adalah variabel C. Karena penurunan C → ε merupakan penurunan satu-satunya dari C, maka kita ganti S → Cd menjadi S → d. Kemudian produksi C → ε kita hapus.

Setelah penyederhanaan menjadi: S → Ab | d
A → d

Contoh:
S → dA | Bd
A → bc
A → ε
B → c

Variabel yang *nullable* adalah variabel A. A → ε bukan penurunan satu-satunya dari A (terdapat A → bc), maka kita ganti S → dA menjadi S → dA | d.A → ε kita hapus.
Setelah penyederhanaan:
S → dA | d | Bd
A → bc
B → c

Contoh:
S → aBcd
B → ab|ε

Pada kasus diatas B bernilai *nullable*, maka penyederhanaannya menjadi:
S → aBcd | acd
B → ab

Contoh tata bahasa bebas konteks:
S → AaCD
A → CD | AB
B → b | ε
C → d | ε
D → ε

Variabel yang *nullable* adalah variabel B, C, D. Kemudian dari A → CD, maka variabel A juga *nullable* (A → ε). Karena D hanya memilki penurunan D → ε, maka kita sederhanakan dulu:

- S → AaCD => S → AaC
- A → CD => A → C

- D → ε kita hapus

Selanjutnya kita lihat variabel B dan C memiliki penurunan ε, meskipun bukan satu-satunya penurunan, maka dilakukan penggantian:

- A → AB => A → AB | A | B
- S → AaC => S → AaC | aC | Aa | a
- B → ε dan C → ε kita hapus

Setelah penyederhanaan:

S → AaC | aC | Aa | a
A → C | AB | A | B
B → b
C → ε

Variabel yang *nullable* adalah A, B, C. Dari S → AB, maka S juga *nullable*. Kita lakukan penggantian:

- A → aCa => A → aa
- B → bA => B → bA | b
- B → BB => B → BB | B
- A → abB => A → abB | ab
- S → AB => S → AB | A | B | ε
- C → ε, B → ε, A → ε dihapus

Perhatikan untuk penggantian S → AB kita tetap mempertahankan S → ε, karena S merupakan simbol awal. Ini merupakan satu-satunya perkecualian produksi ε yang tidak dihapus, yaitu produksi ε yang dihasilkan oleh simbol awal.

Hasil akhir dari penyederhanaan:

S → AB | A | B | ε
A → abB | ab | aa
B → bA | b | BB | B

Contoh tata bahasa bebas konteks:

S → aAb
A → aAb | ε

Hasil penyederhanaan:
S → aAb | ab
A → aAb | ab

Contoh tata bahasa bebas konteks:
S → ABaC
A → BC
B → b | ε
C → D | ε
D → d

Hasil penyederhanaan:
S → ABaC | BaC | AaC | ABa | aC | Aa | Ba | a
A → B | C | BC
B → b
C → D
D → d

Contoh:
S → AA | C | bd
A → Bb | ε
B → AB | d
C → de

Hilangkan produksi ε, sehingga menjadi:
S → A | AA | C | bd
A → Bb
B → B | AB | d
C → de

Selanjutnya penghilangan produksi unit menjadi:
S → Bb | AA | de | bd
A → Bb
B → AB | d
C → de

Penghilangan produksi unit bisa menghasilkan produksi *useless*.

Terakhir dilakukan penghilangan produksi *useless:*
S → Bb | AA | de | bd
A → Bb
B → AB | d

Hasil akhir aturan produksi tidak lagi memiliki produksi ε, produksi unit, maupun produksi *useless*.

9.5. Latihan Soal

Latihan 1:
S → aSa | Abd | Bde
A →Ada
B → BBB | a

Maka
1) Aturan Produksi A→Ada, Simbol variabel A tidak memiliki penurunan yang menuju terminal, sehingga bias dihilangkan
2) Konsekuensi no (1), aturan produksi S →Abd tidak memiliki penurunan

Penyederhanaan menjadi:
S→aSa | Bde
B →<u>BBB | a</u>

Latihan 2:
S →Aa | B
A →ab | D
B →b | E
C → bb
E → aEa

Maka:
1) Aturan produksi A → D, simbol variabel D tidak memiliki penurunan.
2) Aturan produksi C →bb, Penurunan dari simbol S, dengan jalan manapun tidak akan pernah mencapai C

3) Aturan Produksi E→aEa, Simbol variabel E tidak memiliki aturan produksi yang menuju terminal
4) Konsekuensi no (3) Aturan produksi B →E, simbol variabel E tidak memiliki penurunan.

Maka produksi yang *useless*:
A → D
C →bb
E → aEa
B →E

Penyederhanaannya menjadi:
S→_Aa | B
A→ ab
B → b

Latihan 3:
S → aAb | cEB
A → dBE | eeC
B → ff
C→ ae
D → h

Analisa:
1) Aturan produksi S → cEB, A → dBE dapat dihilangkan (E tidak memiliki penurunan)
2) Aturan produksi D → h, redundan
   Sisa aturan produksi
   S → aAb
   A → eeC
   B → ff
   C → ae

Analisis lagi
B → ff juga redundan,

Hasil penyederhanaan menjadi:
S → aAb
A → eeC
C → ae

Contoh lain lagi:
S → aB
A → bcD | dAC
B → e | Ab
C →bCb | adF | ab
F → cFB

Analisis

1) Aturan produksi A → bcD, variabel D tidak memiliki penurunan
2) Konsekuensi no (1), simbol variabel A tidak memiliki penurunan yang menuju terminal (tinggal A → dAC)
3) Konsekuensi no (2), B →Ab tidak memiliki penurunan
4) Simbol variabel F tidak memiliki penurunan yang menuju terminal
5) Konsekuensi no (4), C → adF tidak memiliki penurunan Setelah disederhanakan menjadi:
   S → aB
   B → e
   C →bCb | ab

Latihan 4:
S → aBD
B → cD | Ab
D → ef
A → Ed
F →_ dc

Analisa

1) Aturan produksi A → Ed, E tidak memiliki penurunan
2) Aturan produksi F→_ dc, redundan

Sisa aturan produksi:
S → aBD
B → cD | Ab
D → ef

Latihan 5:
S →Abc | ab
A →AAA | ε

Aturan produksi setelah disederhanakan:
S →Abc | ab
A →AAA | ε

Ingat A _ ε juga harus diperhitungkan

PRINSIP

Setiap kali melakukan penyederhanaan diperiksa lagi aturan produksi yang tersisa, apakah semua produksi yang useless sudah hilang. Penghilangan Produksi Unit Produksi dimana ruas kiri dan kanan aturan produksi hanya berupa satu symbol variabel, misalkan: A → B, C → D.

Keberadaannya membuat tata bahasa memiliki kerumitan yang tak perlu. Penyederhanaan dilakukan dengan melakukan penggantian aturan produksi unit.

Contoh:
S → Sb
S → C
C → D
C → ef
D → dd

Dilakukan penggantian berturutan mulai dari aturan produksi yang paling dekat menuju ke penurunan terminal-terminal (‘=>’ dibaca ‘menjadi’):
C → D => C → dd
S → C => S → dd | ef

Sehingga aturan produksi setelah penyederhanaan:
S → Sb
S → dd | ef
C → dd
C → ef
C → dd

Latihan 6
S →A
S →Aa
A → B
B → C
B → b
C → D
C → ab
D → b

Penggantian yang dilakukan:
C → D => C → b
B → C => B → b | ab, karena B → b sudah ada, maka cukup dituliskan
B → ab
A → B => A → ab | b
S →A => ab | b

Sehingga aturan produksi setelah penyederhanaan:
S → ab | b
S →Aa
A → ab | b
B → ab
B → b
C → b
C → ab
D → b

Latihan 7:

Terdapat tata bahasa bebas konteks:

S → Cba | D
A → bbC
B → Sc | ddd
C → eAn | f | C
D → E | SABC
E → gh

Penggantian yang dilakukan:

D → E menjadi D → gh
C → C, kita hapus
S → D menjadi S → gh | SABC

Sehingga aturan produksi setelah penyederhanaan:

S → Cba | gh | SABC
A → bbC
B → Sc | ddd
C → eA | f
D → gh | SABC
E → gh

Latihan 8:

Terdapat tata bahasa bebas konteks:

S →Ab | Cd
A → d
C → ε

Variabel yang nullable adalah variabel C. Karena penurunan C → ε merupakan penurunan satusatunya dari C, maka kita ganti S → Cd menjadi S → d. Kemudian produksi C → ε kita hapus.

Setelah penyederhanaan menjadi:

S →Ab | d
A → d

Latihan 9

Terdapat tata bahasa bebas konteks:

S → dA | Bd

A → bc

A →ε

B →c

Variabel yang nullable adalah variabel A. A →ε bukan penurunan satu-satunya dari A (terdapat A →bc), maka kita ganti S → dA menjadi S → dA | d. A → ε kita hapus.

Setelah penyederhanaan:

S → dA | d | Bd

A → bc

B → c

Latihan 10:

Terdapat tata bahasa bebas konteks:

S →AaCD

A → CD | AB

B → b | ε

C → d | ε

D → ε

Variabel yang nullable adalah variabel B, C, D. Kemudian dari A → CD, maka variabel A juga nullable (A → ε). Karena D hanya memilki penurunan D → ε, maka kita sederhanakan dulu:

S →AaCD => S →AaC

A → CD => A → C

D → ε kita hapus

Selanjutnya kita lihat variabel B dan C memiliki penurunan ε, meskipun bukan satu-satunya penurunan, maka dilakukan penggantian:

A →AB => A →AB | A | B

S →AaC => S →AaC | aC | Aa | a

B → ε dan C → ε kita hapus Setelah penyederhanaan:

S →AaC | aC | Aa | a
A→C | AB | A | B
B → b
C → ε

Variabel yang nullable adalah A, B, C. Dari S →AB, maka S juga nullable. Kita lakukan penggantian:
A → aCa => A → aa
B → bA => B → bA | b
B → BB => B → BB | B
A → abB => A → abB | ab
S →AB => S →AB | A | B | ε
C → ε, B → ε, A → ε dihapus

*Perhatikan untuk penggantian S _AB kita tetap mempertahankan S → ε, karena S merupakan symbol awal. Ini merupakan satu-satunya perkecualian produksi ε yang tidak dihapus, yaitu produksi ε yang dihasilkan oleh simbol awal.
Hasil akhir dari penyederhanaan:
S →AB | A | B | ε
A → abB | ab | aa
B →bA | b | BB | B

Contoh tata bahasa bebas konteks:
S → aAb
A → aAb | ε

Hasil penyederhanaan:
S → aAb | ab
A → aAb | ab

Latihan 11
Tata bahasa bebas konteks
S → AA | C | bd
A → Bb | H
B → AB | d

C → de

Pertama-tama lakukan penghilangan produksi Ɛ
S → AA | A | H | C | bd
A → Bb
B → B | AB | d
C → de

Latihan 12
Hilangkan aturan produksi useless dari aturan produksi:
S → AB | CA
B → BC | AB
A → a
C → aB | b

Penyelesaian:

S→ ~~AB~~ | CA
~~B → BC | AB~~
A → a
C → aB | b

S → CA
A → a
C → b

Latihan 13
Hilangkan aturan produksi unit dari aturan produksi:
S → Aa | B
B → A | bb
A → a | bc | B

Penyelesaian:
A → B => A → A | bb => A → bb
B → A => B → a | bc | bb
S → B => S → Aa | a | bc | bb

Sehingga
S → Aa | a | bc | bb
B → a | bc | bb
A → a | bc | bb

Tugas:

Hilangkan variabel nullable dari tata bahasa bebas konteks berikut:

1. S → AaB
   A → bA | Ɛ
   B → aB | bB | Ɛ
2. S → ABaC
   A → BC
   B → b | Ɛ
   C → D | Ɛ
   D → d
3. S → ABC
   A → aA | Ɛ
   B → bB | Ɛ
   C → Ɛ

BAB X
BENTUK NORMAL GREIBACH

10.1. Pendahuluan

Bentuk Normal Greibach/Greibach Normal Form (GNF) adalah suatu tata bahasa bebas konteks (CFG) yang aturan produksinya berada dalam bentuk: A → a

a : simbol terminal(tunggal), a ∈ T

: rangkaian simbol-simbol variabel (V*)

Dengan kata lain, suatu CFG dikatakan bentuk normal Greibach bila hasil produksinya (ruas kanan) diawali dengan satu simbol terminal, selanjutnya diikuti oleh rangkaian simbol variabel.

Tujuan:

Mahasiswa memahami tentang bentuk normal greibach (GNF) dan dapat menurunkannya dari suatu tata bahasa bebas konteks

10.2. Pengertian Bentuk Normal Greibach

Bentuk normal Greibach merupakan bentuk normal yang memiliki banyak konsekuensi teoritis dan prkatis. Dalam *bentuk normal Greibach* membatasi posisi munculnya terminal-terminal dan variabel-variabel. Suatu tata bahasa bebas konteks (CFG) dikatakan dalam *bentuk normal Greibach / Greibach Normal Form.* Tata bahasa bebas konteks dikatakan dalam bentuk normal Greibach (GNF) jika setiap aturan produksinya mempunyai bentuk,: A → aα, dengan:

a : simbol terminal (tunggal), a ∈ T

α : rangkaian simbol-simbol variabel (V*)

Dengan kata lain, suatu tata bahasa bebas konteks dalam *bentuk normal Greibach* bila hasil produksinya (ruas kanan) diawali dengan satu simbol terminal, selanjutnya bisa diikuti oleh rangkaian simbol variabel. Contoh tata bahasa bebas konteks dalam bentuk *bentuk normal Greibach:*

S → a | aAB

$A \rightarrow aB$
$B \rightarrow cS$

Suatu tata bahasa bebas konteks dapat dibentuk menjadi bentuk normal Greibach jika:

- Sudah dalam *bentuk normal Chomsky*
- Tidak bersifat rekursif kiri
- Tidak menghasilkan ε

Terdapat dua cara pembentukan *bentuk normal Greibach* , yaitu melalui substitusi dan perkalian matriks. Pada bagian berikutnya kita membahasa kedua cara tersebut.

10.3. Pembentukan *Bentuk Normal Greibach* dengan Substitusi

Secara umum langkah-langkah untuk mendapatkan *bentuk normal Greibach:*

1. Tentukan urutan simbol-simbol variabel yang ada dalam tata bahasa. Misalkan terdapat *m* variabel dengan urutan A_1, A_2, ..., A_m
2. Berdasarkan urutan simbol yang ditetapkan pada langkah (1) seluruh aturan produksi yang ruas kanannya diawali dengan simbol variabel dapat dituliskan dalam bentuk

   $$A_h \rightarrow A_i \gamma$$

   Dimana h $\neq$ *i* (rekrusif kiri sudah dihilangkan), γ bisa berupa simbol-simbol variabel
   a. Jika $h < i$, aturan produksu ini sudah benar (tidakperlu diubah)
   b. Jika $h > i$, aturan produksi belum benar. Lakukan substitusi berulang-ulang terhadap A_i (ganti A_i pada produksi ini dengan ruas kanan produksi dari variabel A_i) sehingga suatu saat diperoleh produksi dalam bentuk $A_h \rightarrow A_p \gamma$ (dimana $h \leq p$)
      i) Jika $h = p$, lakukan penghilangan rekursif kiri
      ii) Jika $h < p$, aturan produksi sudah benar
3. Jika terjadi penghilangan rekursif kiri pada tahap (2b), sejumlah simbol variabel baru yang muncul dari operasi ini dapat disisipkan

pada urutan variabelsemula dimana saja asalkan ditempatkan tidak sebelum A_h (di kiri)

4. Setelah langkah (2) & (3) dikerjakan maka aturan-aturan produksi yang ruas kanannya dimulai simbol variabel sudah berada dalam urutan yang benar

   $A_x \rightarrow A_y \gamma$ (di mana x < y)

   Produksi-produksi yang lain ada dalam bentuk:

   $A_x \rightarrow a \gamma$ (a = simbol terminal)

   $B_x \rightarrow \gamma$

   (B_2 = simbol variabel baru yang akan muncul sebagai akibat dari operasi penghilangan rekursif kiri)

5. *Bentuk normal Greibach* diperoleh dengan cara melakukan substitusi mundur mulai dari variabel A_m, lalu A_{m-1}, A_{m-2}, ... Dengan cara ini aturan produksi dalam bentuk $A_x \rightarrow A_y \gamma$ dapat diubah sehinga ruas kanannya dimulai dengan simbol terminal.

6. Produksi dalam bentuk Bx $\rightarrow \gamma$ juga dapat diubah dengan cara substitusi seperti pada langkah (5)

   Contoh (tata bahasa bebas konteks sudah dalam *bentuk normal Chomsky* dan memenuhi syarat untuk diubah ke *bentuk normal Greibach*), simbol awal adalah S:

   S → CA
   A → a | d
   B → b
   C → DD
   D → AB

Aturan produksi yang simbol pertama pada ruas kanan adalah symbol variabel:

- S → CA (sudah memenuhi aturan karena S<C)
- C → DD (sudah memenuhi karena C<D)
- D → AB (tidak memenuhi, karena D>A)

Aturan produksi yang belum memenuhi syarat adalah: D → AB, karena ruas kiri > simbol pertama pada ruas kanan. Maka kita lakukan sibstitusi pada simbol variabel A, aturan produksi menjadi: D → aB | dB

Setelah semua aturan produksi sudah memenuhi ketentuan urutan variabel, kita lakukan substitusi mundur pada aturan produksi yang belum dalam *bentuk normal Greibach* ('=>' dibaca 'menjadi'):

- C → DD => C → aBD | dBD
- S → CA => S → aBDA | dBDA

Perhatikan substitusi mundur dimulai dari aturan produksi yang memiliki ruas kiri dengan urutan variabel paling akhir (kasus di atas: S<A<B<C<D, maka C lebih dulu disubstitusikan daripada S)

Hasil akhir aturan produksi yang sudah dalam *bentuk normal Greibach:*

S → aBDA | dBDA
A → a | d
B → b
C → aBD | dBD
D → aB | dB

Perhatikan: setiap substitusi kita lakukan pada simbol variabel pertamapada ruas kanan (pada aturan produksi yang belum *bentuk normal Greibach* tentunya).

Prinsipnya:

- Biarkan aturan produksi yang sudah dalam *bentuk normal Greibach*
- Tentukan pengurutan simbol variabel, berdasarkan kondisi aturan produksi yang ada buatlah urutan sedemikian sehingga memudahkan untuk proses selanjutnya. Mulailah terlebih dahulu dari seimbol awal.
- Lakukan perubahan pada aturan produksi yang belum memenuhi ketentuan urutan tersebut dan bila perlu selama proses itu bisa dilakukan substitusi dan penghilangan rekursif kiri
- Lakukan substitusi mundur sedemikian rupa sehingga semua aturan produksi akan diawali dengan tepat sebuah simbol terminal. Proses substitusi mundur dimulai dari aturan produksi dengan urutan paling akhir.
- Lakukan substitusi mundur juga pada aturan produksi baru yang muncul sebagai hasil penghilangan rekursif kiri.

Contoh lain (simbol awal A):
A → BC
B → CA | b
C → AB | a

Kita tentukan urutan simbol: A,B,C (A<B<C)
A → BC (sudah memenuhi karena A<B)
B → CA (sudah memenuhi karena B<C)
C → AB (padahal C > A sehingga harus diubah)
Pengubahan C → AB:
C → AB => C → BCB => C → CACB | bCB
Untuk C → CACB lakukan penghilangan rekursif kiri menjadi

- C → bCBZ1 | aZ_1
- Z_1 → ACB
- Z_1 → $ACBZ_1$

Kita lihat seluruh hasil produksi dari variabel C, sudah dalam *bentuk normal Greibach:*

$$C \rightarrow bCBZ_1 \mid aZ_1 \mid bCB \mid a$$

Setelah semua aturan produksi sudah memenuhi ketentuan urutan variabel, kita laukan substitusi mundur:

$$B \rightarrow CA \Rightarrow B \rightarrow bCBZ_1A \mid aZ_1A \mid bCBA \mid aA$$

$$A \rightarrow BC \Rightarrow A \rightarrow bCBZ_1AC \mid aZ_1AC \mid bCBAC \mid aAC \mid bC$$

Selanjutnya lakukan pula substitusi pada aturan produksi dengan variabel baru yang terbentuk (pada contoh ini Z_1):

- $Z_1 \rightarrow ACB \Rightarrow Z_1 \rightarrow bCBZ_1ACCB \mid aZ_1ACCB \mid bCBACCB \mid aACCB \mid bCCB$
- $Z_1 \rightarrow ACBZ_1 \Rightarrow Z_1 \rightarrow bCBZ_1ACCBZ_1 \mid aZ_1ACCBZ_1 \mid bCBACCBZ_1 \mid aACCBZ_1 \mid bCCBZ_1$

Hasil akhir aturan produksi dalam bentuk *bentuk normal Greibach:*
A → $bCBZ_1AC \mid aZ_1AC \mid bCBAC \mid aAC \mid bC \mid$
B → $bCBZ_1A \mid aZ_1A \mid bCBA \mid aA \mid B$

$C \rightarrow bCBZ_1 \mid aZ_1 \mid bCB \mid a$
$Z_1 \rightarrow bCBZ_1ACCB \mid aZ_1ACCB \mid bCBACCB \mid aACCB \mid bCCB$
$Z_1 \rightarrow bCBZ_1ACCBZ_1 \mid aZ_1ACCBZ_1 \mid bCBACCBZ_1 \mid aACCBZ_1 \mid bCCBZ_1$

10.4. Pembentukan Normal Greibach Melalui Perkalian Matriks

Pembentukan normal Greibach melalui perkalian matriks didasari pemikiran bahwa kumpulan aturan produksi dpt. dianggap sebagai sistem persamaan linier.

Misal terdapat aturan produksi,

$A \rightarrow BC$
$B \rightarrow CA \mid b$
$C \rightarrow AB \mid a$

Jika diganti:
" → " dengan "=" dan " | " dengan " + " , maka aturan produksi diatas menjadi bentuk sistem persamaan linier,

$A = BC$
$B = CA + b$
$C = AB + a$

Sistem persamaan linier diatas dapat ditulis dalam bentuk persamaan matriks sebagai berikut,

$V = VR + S$
V = vektor baris 1 x n yang mengandung simbol-simbol variabel.
R = matriks n x n yang mengandung simbol variabel
S = vektor baris 1 x n yang mengandung simbol terminal dan variabel

Contoh

Buat bentuk normal Greibach dari contoh sebelumnya melalui perkalian matriks.

Penyelesaian:

Ubah tata bahasa bebas konteks ke dalam bentuk persamaan linier.

$A = BC$
$B = CA + b$
$C = AB + a$

Nyatakan persamaan linier sebagai persamaan matriks,

$$V = VR + S$$

$$\begin{bmatrix} A & B & C \end{bmatrix} = \begin{bmatrix} A & B & C \end{bmatrix} \begin{bmatrix} 0 & 0 & B \\ C & 0 & 0 \\ 0 & A & 0 \end{bmatrix} + \begin{bmatrix} A & B & C \end{bmatrix}$$

Diperoleh

$$\mathbf{V} = \begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{C} \end{bmatrix}$$

$$\mathbf{R} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{B} \\ \mathbf{C} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A} & \mathbf{0} \end{bmatrix}$$

$$\mathbf{S} = \begin{bmatrix} \mathbf{0} & \mathbf{B} & \mathbf{A} \end{bmatrix}$$

Selanjutnya bentuk persaman matriks:
V = SQ + S
Q = RQ + R
Q adalah matriks yang mengandung simbol-simbol baru.
V = SQ + S

$$\begin{bmatrix} A & B & C \end{bmatrix} = \begin{bmatrix} 0 & b & a \end{bmatrix} \begin{bmatrix} D & E & F \\ G & H & I \\ J & K & L \end{bmatrix} + \begin{bmatrix} 0 & b & a \end{bmatrix}$$

Didapat:

A = bG + aJ B = bH + aK + b C = bI + aL + a

$\Rightarrow$ Persamaan 1

$$Q = RQ + R$$

$$\begin{bmatrix} D & E & F \\ G & H & I \\ J & K & L \end{bmatrix} = \begin{bmatrix} 0 & 0 & B \\ C & 0 & 0 \\ 0 & A & 0 \end{bmatrix} \begin{bmatrix} D & E & F \\ G & H & I \\ J & K & L \end{bmatrix} + \begin{bmatrix} 0 & 0 & B \\ C & 0 & 0 \\ 0 & A & 0 \end{bmatrix}$$

Didapat:

D = BJ G = CD + C J = AG

E = BK H = CE K = AH + A

F = BL+B I = CF L = AI

Persamaan 2

Substitusi persamaan 1 ke 2, didapat:

D = bHJ + aKJ + bJ

E = bHK + aKK + bK

F = bHL + aKL + bL + bH + aK + b

G = bID + aLD + aD+ bI+aL+a

H = bIE + aLE+aE

I = bIF + aLF+aF

J = bGG +aJG

K = bGH + aJH +bG+aJ

L = bGI + Aji

Didapat bentuk normal Greibach:

A → bG | aJ

B → bH | aK | b

C → bI | aL | a

D → bHJ | aKJ | bJ

E → bHK | aKK | bK

F → bHL | aKL | bL | bH | aK | b

G → bID | aLD | aD | bI | aL | a

H → bIE | aLE | aE

I → bIF | aLF | aF

J → bGG | aJG

K → bGH | aJH | bG | aJ

L → bGI | aJI

10.5. Latihan Soal

1. S → AA | d
   A → SS | b
2. S → AB | BC
   A → BA | a
   B → CC | b
   C → AB

Penyelesaian:

1. S → AA | d
   A → SS | b
   Aturan produksi yang sudah dalam bentuk Normal Greibach adalah:
   S → d
   A → b

   Aturan produksi yang belum dalam bentuk Normal Greibach adalah:
   S → AA
   A → SS

   Urutan simbol: S < A
   Karena A < S (lihat soal), maka harus dilakukan pengubahan.
   Proses pengubahan:
   A → SS | b: A → AAS | dS | b

   Aturan produksi menjadi rekursif kiri:
   $\alpha 1$ = AS
   $\beta 1$ = dS ; $\beta 2$ = b
   A → dSZ1 | bZ1
   Z1 → AS
   Z1 → ASZ1

Dalam bentuk normal Greibach:
A → dSZ1 | bZ1 | dS | b

Lakukan pengubahan terhadap variabel baru.
Z1 → dSZ1S | bZ1S |dSS | bS
Z1 → dSZ1SZ1 | bZ1SZ1 | dSSZ1 | bSZ1

Lakukan pengubahan terhadap variabel S.
S → AA|d => S → dSZ1 A | bZ1A | dSA | bA | d

Hasil akhir aturan produksi dalam bentuk Normal Greibach adalah:
S → dSZ1 A | bZ1A | dSA | bA | d
A → A → dSZ1 | bZ1 | dS | b
Z1 → dSZ1S | bZ1S |dSS | bS
Z1 → dSZ1SZ1 | bZ1SZ1 | dSSZ1 | bSZ1

Substitusi untuk variabel S
S → dZ1 A | bZ1 A | d

Hasil akhir aturan produksi dalam bentuk Normal Greibach adalah:
S → dZ1 A | bZ1 A | d
A → dZ1 | bZ1 | b
Z1 → dZ1 | bZ1
Z1 → dZ1Z1 | bZ1Z1

2. S → AB | BC
A → BA | a
B → CC | b
C → AB

Latihan
Konversi CFG berikut menjadi CNF
S → XY | Xn | p
X → mX | m
Y → Xn | o

Penyelesaian:
Di sini, S tidak muncul di sisi kanan produksi apa pun dan tidak ada unit atau nullproduksi dalam set aturan produksi. Jadi, kita bisa melompati Langkah 1 ke Langkah 3.
Langkah 4:
Sekarang setelah diganti
X dalam S → XY | Xo | p dengan mX | m

didapatkan
S → mXY | mY | mXo | mo | p.

Dan setelah diganti
X di Y → Xn | o

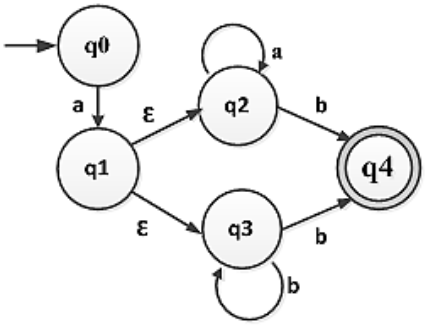
dengan sisi kanan
X → mX | m

kita dapatkan
Y → mXn | mn | o.

Dua produksi baru O → o dan P → p ditambahkan ke set produksi dan kemudian kami dating ke final GNF sebagai berikut:
S → mXY | mY | mXC | mC | p
X → mX | m
Y → mXD | mD | o
O → o
P → p

BAB XI
DFA DAN TATA BAHASA REGULER

11.1. Pendahuluan



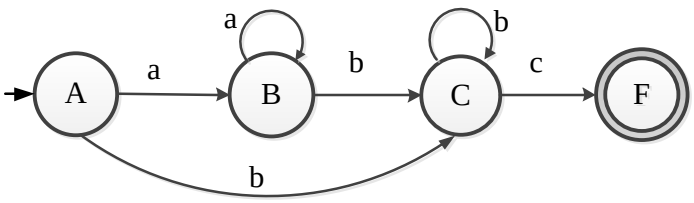
Gambar 11.1. Gambar FA Untuk Ekspresi a(a*+b*)b

Otomata hingga menspesifikasikan sebuah bahasa sebagai himpunan semua untai yang menggerakkannya dari state awal ke state akhir atau ke himpunan state akhir. Otomata berikut menerima ekspresi reguler a(a*+b*)b. Selain dengan ekspresi reguler, kita dapat mengkonstruksi aturanaturan produksi untuk suatu tata bahasa reguler. Aturan produksi untuk bahasa reguler α →β, dengan ketentuan:

α adalah sebuah simbol variabel/non-terminal

β maksimal memiliki sebuah variabel.

Jika ada selalu terletak pada posisi paling kanan. Simbol terminal dilambangkan dengan huruf kecil. Simbol variabel dilambangkan dengan huruf besar Secara umum, dapat dibuktikan bahwa untuk setiap ekspresi reguler, ada aotomata terbatas yang mendefinisikan bahasa reguler yang sama. Dapat dilihat bahwa NFA berikut mendefinisikan hal yang sama bahasa sebagai ekspresi reguler a* b^+ c:

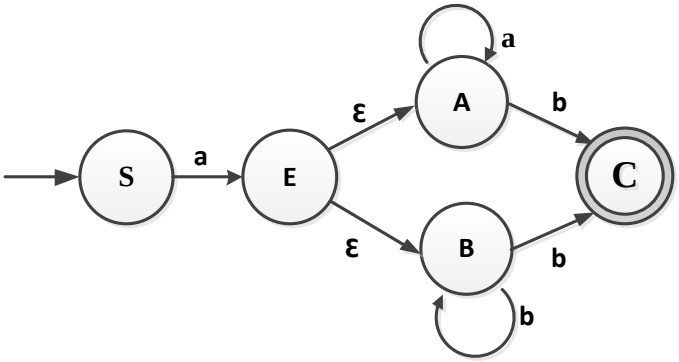


11.2. Mengkonstruksi aturan produksi Otomata Hingga untuk tata bahasa regular

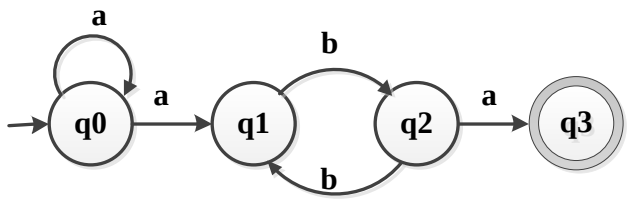
S → aE		S → aE
E → A		E → A | B
E → B		A → aA | b
A → aA	atau	B → bB | b
A → b		
B → bB		
B → b		

Tata Bahasa Regular
G = {S,E,A,B}
T = {a,b}
P = {S→aE, E → A | B, A → aA | b, B → bB | b}
S = S

Diagram FA



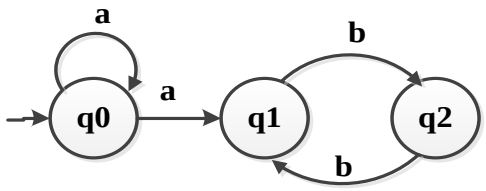
Contoh lain:
Tentukan aturan produksi untuk bahasa yang diterima oleh otomata berikut.



Dari q3 tidak ada transisi keluar dan bukan state akhir, maka q3 diabaikan. Jika diidentikkan: S untuk q0, A untuk q1, dan B untuk q2, maka:

G = { S, A, B, C}

T = {a,b}



Aturan produksi

S →Ɛ | aS | aA

A → bB

B →Ɛ | bA

Selain membuat suatu aturan produksi dari sebuah Otomata, juga bisa membuat otomata untuk suatu tata bahasa regular, seperti contoh berikut:

Gambarkan sebuah otomata hingga dari aturan produksi berikut!

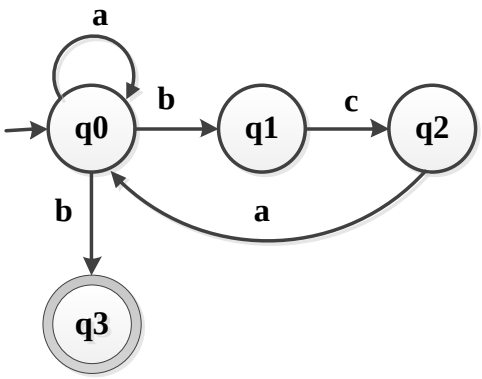
S → aS | bA | b

A → cB

B → aS

Penyelesaian:

Identikkan: S dengan q0
A dengan q1
B dengan q2



11.3. Tata Bahasa Linier Kiri Dan Linier Kanan

Suatu Tata bahasa G (T, N, S, P) disebut linier kiri jika seluruh aturan produksinya berbentuk

$A \rightarrow xB$

$A \rightarrow x$

Dengan A, B $\in$ N dan x $\in$ T*

Suatu Tata bahasa G (T, N, S, P) disebut linier kiri jika seluruh aturan produksinya berbentuk

$A \rightarrow Bx$

$A \rightarrow$ **x dengan A, B $\in$ N dan x $\in$ T***

Tata bahasa reguler bila bersifat linier kiri atau linier kanan.

Contoh 1

Tata bahasa G = {{S} , {a,b}, S , P } dengan aturan produksi P adalah S $\rightarrow$ abS |a

adalah Tata bahasa linier kanan /reguler

Tata bahasa G = {{S, S_1,S_2 } , {a,b}, S , P } dengan aturan produksi P adalah

$S \rightarrow S_1ab$

$S_1 \rightarrow S_1ab \mid S_2$

$S_2 \rightarrow a$

adalah Tata bahasa linier kiri /reguler

Tata bahasa G = {{S, A, B} , {a,b}, S , P } dengan aturan produksi P adalah

$S \rightarrow A$

$A \rightarrow aB \mid \lambda$

$B \rightarrow Ab$

adalah bukan Tata bahasa regular

11.4. Konversi DFA ke Tata bahasa linier

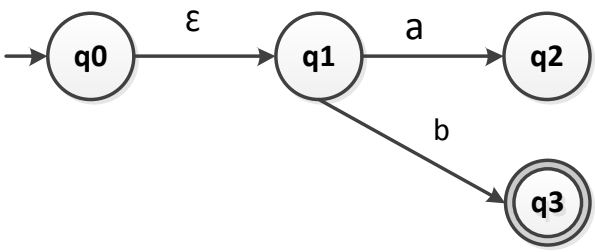
Setiap DFA dapat diubah menjadi Tata bahasa yang memiliki aturan produksi yang linier. Aturan pengubahan ini adalah sebagai berikut:

Setiap transisi status $\delta(A,a)=B$ diubah menjadi aturan produksi

$A \rightarrow aB$

Setiap status akhir P diubah menjadi aturan produksi $P \rightarrow \varepsilon$

Contoh FSA berikut



Tata bahasa linier untuk FSA tersebut yaitu G = ({a,b}, {S,S_1,S_2,S_3},S, P) dengan aturan produksi P adalah:

$S \rightarrow S_1$

$S_1 \rightarrow aS_2$

$S_2 \rightarrow bS_3$

$S_3 \rightarrow \varepsilon$

11.5. Konversi Tata bahasa linier ke DFA

- Setiap aturan produksi $A \rightarrow ab$ diubah menjadi transisi status $\delta(A,a)=B$
- Setiap aturan produksi $A \rightarrow a$ diubah menjadi $\delta(A,a)=S_F$
- Untuk $a \in T^*$ dengan $|a|>1$ dapat dibuat state tambahan
- Setiap aturan produksi $A \rightarrow B$ diubah menjadi $\delta(A,\varepsilon)=B$

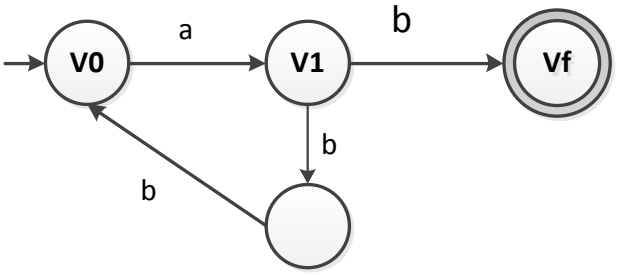
Contoh

Tata bahasa $G = (\{a,b\},\{V_0,V_1\}, V_0, P)$ dengan P:

$V_0 \rightarrow aV_1$

$V_1 \rightarrow abV_0 \mid b$

Mesin FSA nya menjadi



Hal mendasar tentang bahasa reguler

1. Terdapat suatu algoritma untuk menentukan diterima atau tidaknya suatu suatu string pada bahasa L, dengan menggunakan FSA.
2. Terdapat suatu algoritma untuk menentukan suatu bahasa reguler kosong, finite atau infinite
   - Dengan DFA, jika terdapat lintasan dari simpul start ke simpul Final, maka bahasa tersebut tidak kosong.
   - Cari simpul simpul yang membentuk siklus. Jika terdapat lintasan dari simpul start ke simpul Final yang melalui simpul yang membentuk siklus, maka bahasa tersebut infinite. Jika tidak, maka bahasa tersebut finite.

Penerapan ekspresi reguler

- Digunakan untuk memerinci unit-unit leksikal sebuah bahasa pemrograman (token).
  contoh ekspresi reguler ‘bilangan real positif’
  (0+1+...+9)(0+1+...+9)*.(0+1+...+9) (0+1+...+9)*
  contoh ekspresi reguler ‘bilangan bulat’: (‘+’ + ’-‘ + λ) (0+1+...+9)(0+1+...+9)*
- Editor text

11.6. Pumping Lemma

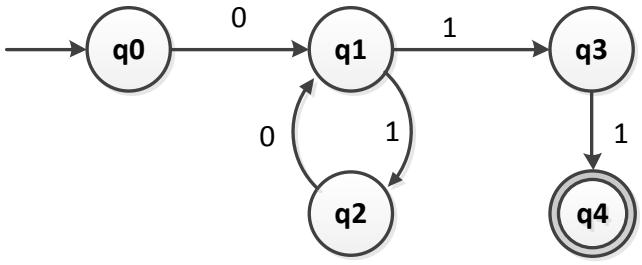
Apabila suatu bahasa merupakan bahasa reguler maka akan dapat diterima oleh mesin DFA $M=(Q,\Sigma,\delta,q_0,F)$, dengan sejumlah state n.

Apabila string w dengan $|w| \geq n$ diinputkan dalam DFA, maka pasti ada simpul k dalam DFA yang dikunjungi lebih dari satu kali.

Apabila string diantara simpul k yang sama tersebut ‘dipompa’, maka sisanya pasti masih diterima oleh DFA tersebut.

Contoh

Bahasa yang menerima ekspresi reguler 0(10)*11



- Ambil string $w \in L$, dengan $|w| \geq n$: w= 01011

$$Q_0\ \underline{0}\ q_1\ \underline{1}\ q_2\ \underline{0}\ q_1\ \underline{1}\ q_3\ \underline{1}\ q_4$$

- Simpul q_1 dikunjungi 2 kali.
- String diantara simbol q_1 tersebut ‘dipompa’ keluar

$$Q_0\ \underline{0}\ q_1\ \underline{1}\ q_3\ \underline{1}\ q_4$$

- String 011 tersebut masih dapat diterima oleh FSA.

Secara formal, Misal L adalah sebuah bahasa reguler infinite, maka terdapat sebuah konstanta n dengan sifat bahwa jika w adalah sebuah string dalam L yang panjangnya lebih besar atau sama dengan n maka kita bisa menulis w=uvx sedemikian sehingga $uv^ix \in L$ untuk semua $i \geq 0$. dengan $|v| \geq 1$ dan $|uv| \leq n$.
Notasi matematisnya

$$(\forall L)(\exists n)(\forall z)\begin{bmatrix} z \in L \wedge |z| \geq n \Rightarrow \\ (\exists u,v,w)\big(z = uvw \wedge |uv| \leq n \wedge |v| \geq 1 \wedge (\forall i)\big(uv^i w \in L\big)\big) \end{bmatrix}$$

Penjelasan

- Mengidentifikasi sifat yang harus dimiliki oleh suatu bahasa reguler.
- Cara untuk menentukan apakah sebuah bahasa tidak reguler
- Untuk memperlihatkan bahwa suatu bahasa infinite tidak reguler, maka kita tunjukkan bahwa untuk nilai n yang cukup besar, sekurang-kurangnya satu untai yang panjangnya n atau lebih besar gagal untuk dapat 'dipompa'.

Contoh:
L = $\{a^{i\wedge 2} \mid i \geq 1\}$
$\{a^1, a^4, a^9, a^{16}, ...\}$
{a , aaaa , aaaaaaaaa , aaaaaaaaaaaaaaaa, ... }

Suatu string dalam L harus mempunyai panjang yang berupa nilai kuadrat (1,4,9,16, ..., n^2, ...)
Misal bahwa L adalah bahasa reguler.
Perhatikan bahwa terdapat sebuah nilai n sedemikian sehingga $a^{n\wedge 2} \in L$,
Menurut pumping lemma dapat kita tuliskan $a^{n\wedge 2}$ =uvx, sedemikian hingga

- $1 \leq |v| \leq n$
- $(\forall i)\ (uv^iw \in L)$

Karena $|v| \geq 1$ maka jelas bahwa
$|uvw| < |uv^2w| < |uv^2w| < ...$

Ambil i=2 maka kita dapatkan

$$N^2 = |uvw| < |uv^2w| \leq n^2 + n < (n+1)^2$$

Jelas $n^2 < |uv^2w| < (n+1)^2$

Panjang $|uv^2w|$ bukan merupakan kuadrat sempurna, karena berada diantara 2 nilai kuadrat sempurna yang berurutan.

Berarti $uv^2w \notin L$

Jadi disimpulkan bahwa $L = \{a^{i\wedge 2} \mid i \geq 1\}$ bukan merupakan bahasa reguler.

11.7. Konstruksi NDFA dari Tata bahasa linier Kanan

Let G = (N, S, P, S).

Konstruksi NDFA

M = (Q, S, d, S, F), where

Q = N U {X}, X not in N or S

F = {X} d dibangun oleh

If A → a B is in P, then B is in d(A,a)

If A → a is in P, then X is in d (A,a)

S → a T	T → 1 T	▪ δ(S,a) = {T, X}
S → b T	T → 2 T	▪ δ (S,b) = {T,X}
S → a	T → a	▪ δ (S,1) = F
S → b	T → b	▪ δ (S,2) = F
T → a T	T → 1	▪ δ (T,a) = {T,X}
T → b T	T → 2	▪ δ (T,b) = {T,X}
		▪ δ (T,1) = {T,X}
		▪ δ (T,2) = {T,X}

Contoh:

S → Sa

S → Sb

S → S1

S → S2

S → a
S → b
S => a
S => S1 => a1
S => S2 => Sb2
=> S1 b2 => a1 b2

11.8. Konstruksi NDFA dari Tata bahasa linier kiri

G = (N, S, P, S).

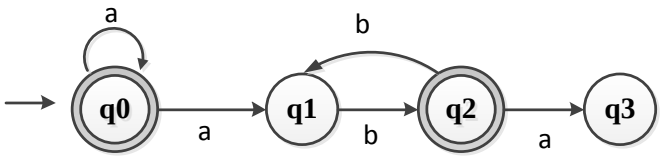
Konstruksi NDFA
M = (Q, S, d, X, F), dimana
Q = N U {X}, X not in N or S
F = {S}
d dikonstruksi oleh
If A -> B a is in P, then A is in d(B,a)
If A -> a is in P, then A is in d (X,a)

S -> S a	d(X,a) = {S}
S -> S b	d(X,b) = {S}
S -> S 1	d(X,1) = F
S -> S 2	d(X,2) = F
S -> a	d(S,a) = {S}
S -> b	d(S,b) = {S}
	d(S,1) = {S}
	d(S,2) = {S}

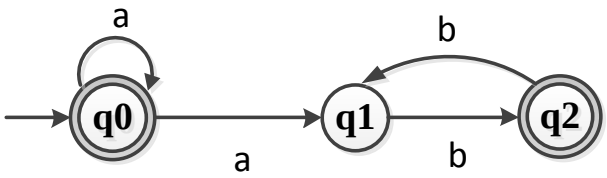
11.9. Latihan Soal

Latihan 1:

Tentukan aturan produksi untuk bahasa yang diterima oleh otomata berikut.



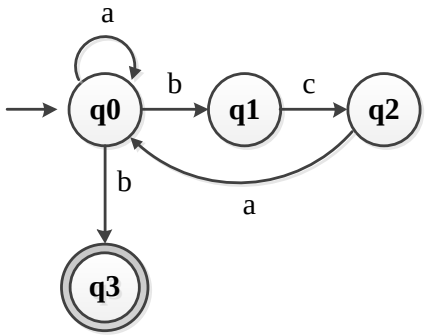
Dari q3 tidak ada transisi keluar dan bukan state akhir, maka q3 diabaikan.
Jika diidentikkan: S untuk q0, A untuk q1, dan B untuk q2, maka:
V = { S, A, B, C}
T = { a, b}



Aturan produksi
S →ε | aS | aA
A → bB
B →ε | bA

Latihan 2:
Gambarkan sebuah otomata hingga dari aturan produksi berikut!
S → aS | bA | b
A → cB
B → aS

Penyelesaian:
Identikkan:
S dengan q0
A dengan q1
B dengan q2



Latihan 2:

Gambarkan sebuah otomata hingga dari aturan produksi berikut!

S → aB | bA | ε

A → abaS

B → babS

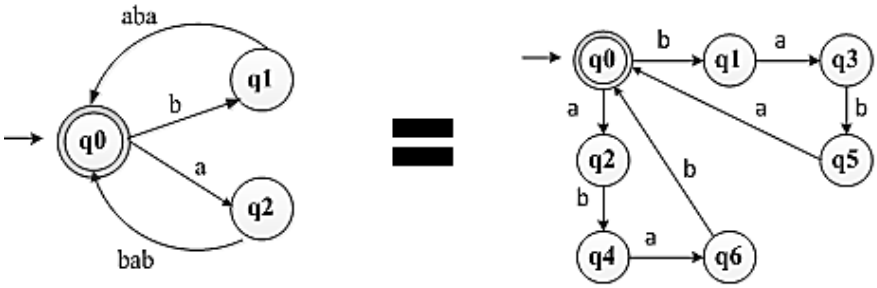
Penyelesaian:

Identikkan:

S dengan q0

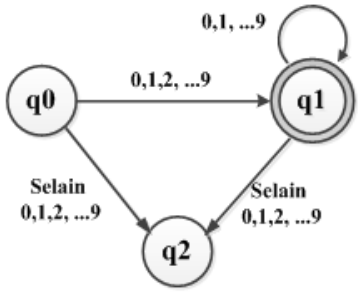
A dengan q1

B dengan q2



BAB XII
REGULAR EKSPRESI

12.1. Pendahuluan



Gambar 12.1. FSA Menerima Bilangan Integer Tak Bertanda

Sebuah bahasa dinyatakan regular jika terdapat finite state automata yang dapat menerimanya. Bahasa-bahasa yang diterima oleh suatu finite state automata bisa dinyatakan secara sederhana dengan ekspresi regular (regular expression). Ekspresi regular selanjutnya kita sebut sebagai ER, memungkinkan menspesifikasikan atau mendefinisikan bahasa-bahasa. Ekspresi regular memberikan suatu pola (pattern) atau template untuk untai/string dari suatu bahasa. Untai yang menyusun suatu bahasa regular akan cocok (match) dengan pola bahasa itu. Banyak masalah pada perancangan perangkat lunak yang bisa disederhanakan dalam dengan melakukan pengubahan notasi ekspresi regular ke dalam implementasi computer dari finite state automata yang bersangkutan. Penerapan ekspresi regular yang tampak misalnya pencarian (searching) untai karakter (string) pada suatu file, biasanya fasilitas ini ada pada text editor. Dalam kasus itu dilakukan penerapan finite state automata pada untai-untai yang terdapat dalam file tersebut. Contoh penerapan yang lain adalah pembatasan data masukan yang diperkenankan, misalnya suatu field masukan hanya menerima input bilangan (0..9). Bisa dilihat otomatanya pada gambar 12.1.

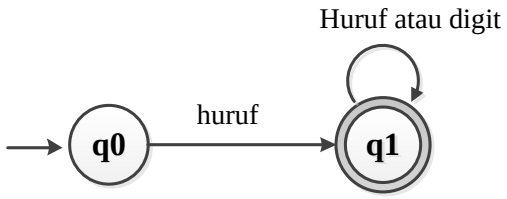
Bila dalam bahasa Indonesia bisa dikatakan bahwa otomata pada gambar menerima masukan symbol input antara 0 sampai 9 sedang ekspresi regularnya dinyatakan sebagai berikut: (digit)digit)* dengan digit adalah 0..9

Dalam suatu kompilator, ekspresi regular bisa diaplikasikan untk melakukan analisis leksikal, yaitu mengidentifikasikan unit-unit leksikal yang dikenal dalam program. Unit leksikal ini biasanya disebut dengan token. Token-token pada suatu bahsa pemrograman kebanyakan tanpa kecuali dinyatakan sebagai ekspresi regular . Misalkan suatu identifier baik huruf besar atau huruf kecil yang kemudian diikuti huruf atau digit, dengan tanpa pembatasan jumlah panjang bisa dinyatakan sebagai: (huruf)(huruf+digit)*

Contoh otomata pada gambar 2 berguna mengenali identifier, bila huruf A..Z, a..z, dan digit berupa 0..9.

Bila dalam bahasa FORTRAN dibatasi panjang identifier maksimal 6 (enam), maka ekspresi regular untuk identifier pada FORTRAN bisa dinyatakan sebagai: (huruf)(huruf+digit)^5

Dalam implementasinya suatu finite state automata akan diterjemahkan menjadi kode dalam sebuah bahasa pemrograman.



Gambar 12.2. FSA Mengenali Identifier

- Bahasa disebut reguler jika terdapat FSA yang dapat menerimanya.
- Bahasa reguler dinyatakan secara sederhana dengan ekspresi reguler/regular expression (RE).
- Bahasa regular dapat dinyatakan sebagai ekspresi regular dengan menggunakan 3 operator: *concate, alternate,* dan *closure.*
- Dua buah ekspresi regular adalah ekuivalen jika keduanya menyatakan bahasa yang sama
- Contoh penerapan: *searching string* pada file

- RE -> NFA dengan ε Move -> DFA

12.2. Notasi Ekspresi Regular

Sekarang kita akan memperkenalkan notasi baru disini yaitu (* + + ∪ .) yaitu:

***** (karakter asterisk), berarti bisa tidak muncul, bisa juga muncul berhingga kali (0-n)

+ (pada posisi superscript/diatas) berarti minimal muncul satu kali (1-n) + atau ∪ berarti union

. (titik) berarti konkatensi, biasanya titik bisa dihilangkan, misal ab bermakna seperti a.b

***) Closure**

Operasi closure adalah operasi yang mendefinisikan bahasa dari string null sampai dengan jumlah tidak berhingga. Notasi untuk menyatakan operasi closure adalah dengan symbol bintang kleene ∑*

contoh:

1. Jika ∑ = {x}, maka: ∑* = L4 = { Λ, x, xx, xxx, xxxx, …}
2. Jika ∑ = {0,1}, maka: ∑* = { Λ 0 1 00 01 10 11 000 001 … }
3. Jika ∑ = {a,b,c}, maka: ∑* ={ Λ a b c aa ab ac ba bb bc ca cb cc aaa …}

Definisi ekspresi reguler

Jika Σ merupakan himpunan simbol, maka

1. ∅ , λ , Dan a ∈Σ adalah ekspresi reguler dasar
2. Jika r dan t masing masing merupakan ekspresi reguler maka komposisi berikut merupakan ekspresi reguler:

Ekspresi	Makna
r+t	himpunan string gabungan R∪T
rt	operasi penyambungan string thd himpunan
r^*	Kleene closure dari R
(r)	r

Contoh ekspresi reguler

- (0+1)*: himpunan seluruh string yang dapat dibentuk dari simbol ‘0’ dan ‘1’
- (0+1)*00(0+1)*: himpunan string biner yang mengandung paling sedikit satu substring ‘00’
- (0+1)*00: himpunan string biner yang diakhiri dengan ‘00’
- ER: ab*cc
  contoh string yang dibangkitkan: abcc, abbcc, abbbcc, abbbbcc, acc, (b bisa tidak muncul atau muncul sejumlah berhingga kali)
- ER: 010*
  contoh string yang dibangkitkan: 01, 010, 0100, 01000 (jumlah 0 diujung bisa tidak muncul, bisa muncul berhingga kali)
- ER: a*d
  contoh string yang dibangkitkan: d, ad, aad, aaad
- ER: a+ d
  contoh string yang dibangkitkan: ad, aad, aaad (a minimal muncul sekali)
- ER: a*∪b*(ingat ‘∪’ berarti atau)
  contoh string yang dibangkitkan: a, b, aa, bb, aaa, bbb, aaaa, bbbb
- ER: (a∪b)
  contoh string yang dibangkitkan: a, b
- ER: (a∪b)*
  contoh string yang dibangkitkan: a, b, ab, ba, abb, bba, aaaa, bbbb (untai yang memuat a atau b) * perhatikan: notasi ‘∪’ kadang dituliskan juga sebagai ‘+’
- ER: 01*+0
  contoh string yang dibangkitkan: 0, 01, 011, 0111, 01111, (string yang berawalan dengan 0, dan selanjutnya boleh diikuti deretan 1)

12.3. Bahasa Reguler

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

Contoh:

Tentukan bahasa reguler yang dibentuk oleh r=(aa)*
Jawab:
L(r) = L((aa)*)
= { λ, aa, aaaa, aaaaaa, ... }
= { a^{2n} | n ≥ 0 }
Menyatakan himpunan string a dengan jumlah genap

Tentukan bahasa reguler yang dibentuk oleh r=(aa*)(bb)*b
Jawab:
L(r) = L((aa)* (bb)*b)
= { a^{2n} b^{2m+1} | n,m ≥ 0 }

Tentukan ekspresi reguler pembentuk bahasa pada Σ = {0,1}, yaitu

L(r) = { w ∈ Σ* | w memiliki substring '00' }

Jawab: r = (0+1)*00(0+1)*

Tentukan ekspresi reguler pembentuk bahasa pada Σ = {a,b}, yaitu L(r) = { ab^nw | n≥ 3 , w ∈ $\{a , b\}^+$ }
Jawab: r = abbb(a+b)(a+b)*

Latihan:
Carilah seluruh string pada L((a+b)*b(a+ab)*) dengan panjang string kurang dari 4.
Tentukan ekspresi reguler pembentuk bahasa pada Σ = {a,b,c}, yaitu

a. L(r) = { w ∈ Σ* | w memiliki tepat sebuah simbol 'a' }
b. L(r) = { w ∈ Σ* | w mengandung tepat 3 buah simbol 'a'}
c. L(r) = { w ∈ Σ* | w mengandung kemunculan masing masing simbol minimal satu kali}

Tentukan ekspresi reguler pembentuk bahasa pada Σ = {0,1}, yaitu

a. L(r) = { w ∈ Σ* | w diakhiri dengan string 01 }
b. L(r) ={ w ∈ Σ* | w tidak diakhiri dengan string 01 }
d. L(r) ={ w ∈ Σ* | w mengandung simbol '0' sebanyak genap }

e. L(r) ={ w $\in$ Σ* | kemunculan string '00' pada w sebanyak kelipatan 3 }

Tentukan ekspresi reguler pembentuk bahasa pada Σ = {a,b}, yaitu L(r) = { w $\in$ Σ* | |w| mod 3 = 0 }

12.4. Sifat Bahasa Reguler

Tertutup terhadap operasi himpunan sederhana

Jika L_1 dan L_2 adalah bahasa reguler, maka $L_1 \cup L_2$,

$L_1 \cap L_2$, L_1L_2, ~(L_1) dan L_1^* adalah bahasa reguler juga

Tertutup terhadap homomorphic image.

Jika L_1 adalah bahasa reguler, maka homomorphic image h(L_1) adalah bahasa reguler juga.

Dimisalkan Σ dan Γ adalah alfabet, maka fungsi homomorphic dinyatakan dengan, h: $\Sigma \rightarrow \Gamma$

Jika w = a_1 a_2 ... $A_{n,}$ maka h(w) = h(a_1) h(a_2) ... H(a_n)

Jika L adalah bahasa pada Σ maka homomorphic, *image* bahasa L adalah:

$$h(L)=\{h(w) \mid w \in L\}$$

Contoh:

Dimisalkan Σ = {a,b} dan Γ = {a,b,c} dan didefinisikan

$$H(a) = ab \text{ dan } h(b) = bbc$$

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

$$H(L)= \{ abab, abbbcab\}$$

Dimisalkan Σ = {a,b} dan Γ = {b,c,d} dan didefinisikan

$$H(a) = dbcc \text{ dan } h(b) = bdc$$

Homomorphic image bahasa L(r) yang dibentuk dari ekspresi reguler:

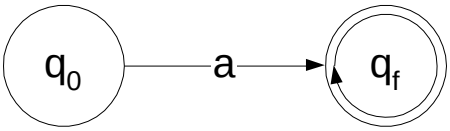
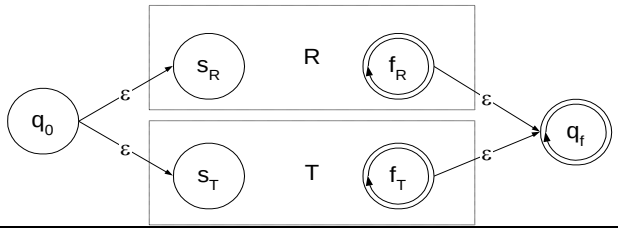
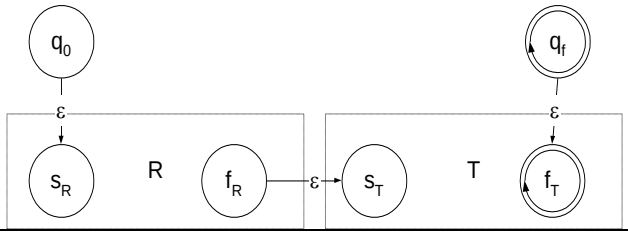
$$r = (a+b^*)(aa)^*$$

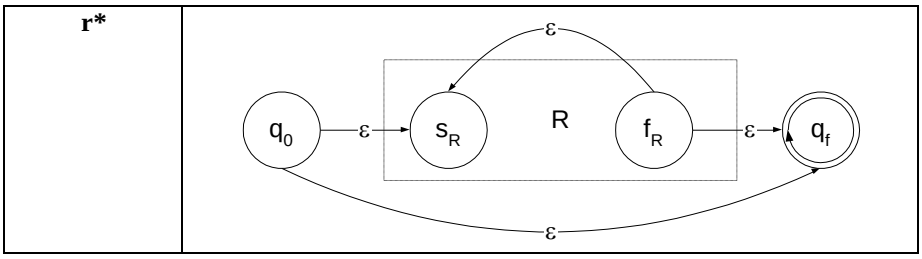
Adalah h(L(r)) yang dibentuk dengan ekspresi regular:

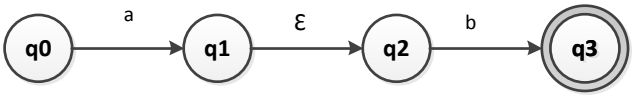
$$R = (dbcc + (bdc)^*) (dbccdbcc)^*$$

12.5. Hubungan Ekspresi Regular dan Finite State Automata.

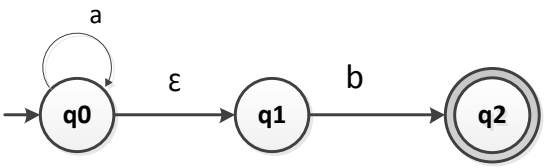
Untuk setiap ekspresi regular ada satu Non-deterministic Finite Automata dengan transisi ε (NFA ε-move) yang ekivalen. Sementara untuk Deterministic Finite Automata ada satu ekspresi regular dari bahasa yang diterima oleh Deterministic Finite Automata. Sederhananya kita bisa membuat suatu Non-deterministic Finite Automata ε-move dari suatu ekspresi regular. Bisa dilihat contohnya pada gambar 3,4,5. Yang perlu diperhatikan disitu, state akhir akan menandakan apakah suatu input diterima atau tidak

Ekspresi	FSA
ε	
∅	
a	
r+t	
rt	

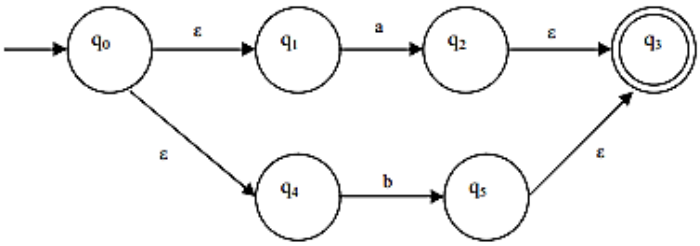




Gambar 12.3. NFA ε-Move Untuk ER: Ab



Gambar 12.4. NFA ε-Move Untuk ER: A*B

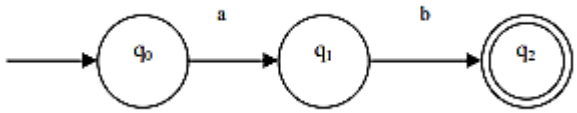


Gambar 12.5. NFA ε-Move Untuk ER: A ∪ B

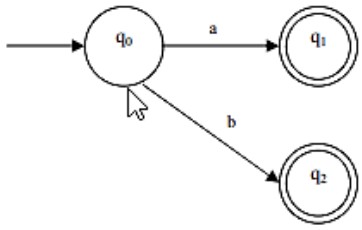
Kemudian dari Non-deterministic Finite Automata ε-move tersebut dapat kita ubah ke Non-deterministic Finite Automata dan

selanjutnya ke Deterministic Finite Automata, atau prosesnya sebagai berikut: NFA ε-move ⇒ NFA ⇒ DFA

Bila ekspresi regularnya cukup sederhana kita bisa saja langsung mengkonstruksi NFA-nya, tanpa melalui NFA ε-move. Misalkan saja NFA tanpa ε-move untuk ER: ab bisa dilihat pada gambar 6

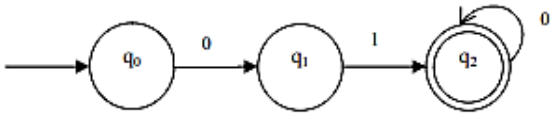


Gambar 12.6. NFA untuk ER: ab



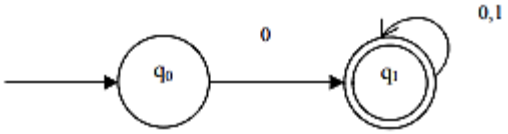
Gambar 12.7. NFA untuk ER: a ∪ b

Contoh-contoh lain bisa dilihat pada gambar 8 sampai 15

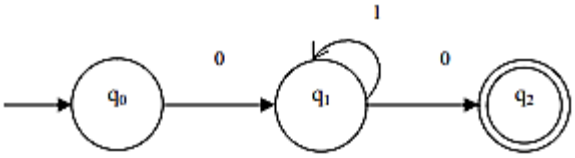


Gambar12.8. NFA untuk ER: 010*

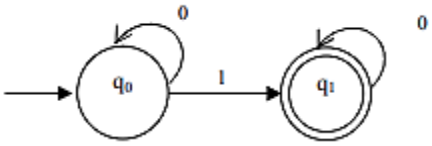
Gambar12.9. NFA untuk ER: 0 (1∪0)



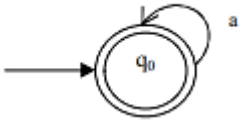
Gambar 12.10. NFA untuk ER: 0 (1∪0)*



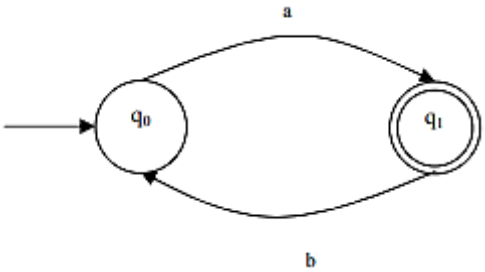
Gambar 12.11. NFA untuk ER: 01*0



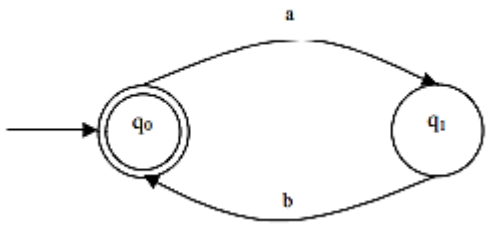
Gambar 12.12. NFA untuk ER: 0*10



Gambar 12.13. NFA untuk ER: a*



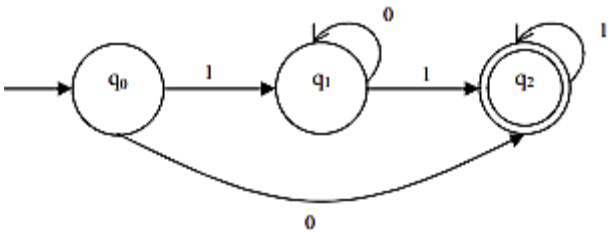
Gambar 12.14. NFA untuk ER: a (ba)*



Gambar 12.15. NFA untuk ER: (ab)*

Dari sebuah finite state automata (NFA atau DFA) kita bisa menentukan ekspresi regular yang diterima oleh otomata bersangkutan. Terdapat langkah-langkah secara formal untuk menentukan ekspresi regular dari suatu finite state automata, tetapi kita bisa juga secara langsung menentukan ekspresi regular-nya dengan mengamati perilaku dari otomata tersebut. Kita lihat dari gambar 16, input yang menuju pada final state (q2) adalah 0 atau 10*1, pada state q2 menerima input 1 dalam jumlah berapapun (1*) akan tetap di q2. Bisa dikatakan mesin itu menerima 01* atau 10*11*, yang dinyatakan dalam ekspresi regular

$$01^* \cup 10^*11^*$$

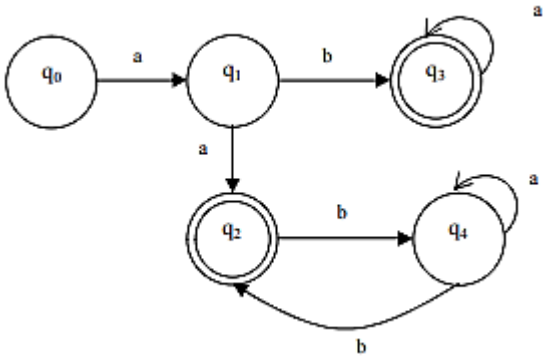


Gambar 12.16. Mesin FSA

Pada gambar 17 kita lihat final state adalah q2 dan q3. Input menuju q3 adalah ab dengan di q3 bisa menerima a*, sehingga bila digabung menjadi aba*. Input yang menuju q2 adalah aa, selanjutnya dari q2 menerima ba* b kembali ke q2, sehingga bila digabung menjadi

aa(ba*b)*. Akhirnya kita peroleh ekspresi regular untuk gambar17: aba*∪aa(ba*b)*, bisa pula dipersingkat menjadi:

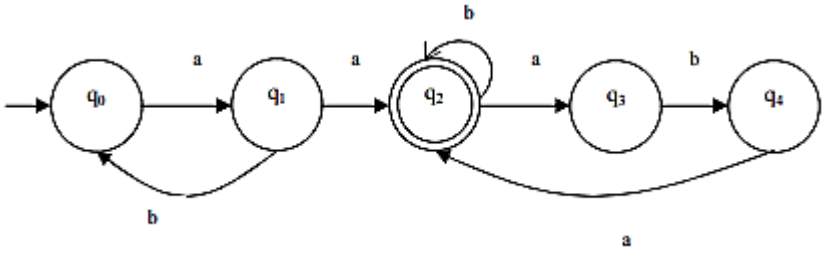
a(ba*∪a(ba*b)*)



Gambar 12.17. Mesin FSA

Pada gambar 18 input yang menuju final state (q2) adalah a(ba*)a, karena dari q1 bila menerima (ba)* kembali ke q1 lagi. Di q2 bila menerima b* tetap di q2. Selanjutnya menerima (aba) kembali ke q2, dan di q2 bila menerima b* tetap di q2, atau digabung (abab*). Maka ekspresi regularnya:

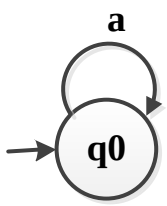
a(ba)*ab*(abab*)*



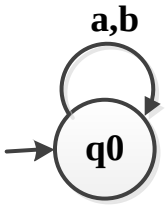
Gambar 12.18. Mesin FSA

12.6. Latihan soal

1. Tentukan Bahasa regular yang dibentuk oleh:
   a. ab*a
   b. 0*1*
   c. (ab)*a
   d. (a + c) b*
2. Contoh: Tentukan FSA untuk ekspresi reguler:
   a. 01
   b. 0+11
   c. 01*+1
   d. (0+1*)*
3. Carilah seluruh string pada L((a|b)*b(a|ab)*) dengan panjang string kurang dari 4.
4. Gambar (diagram FA)

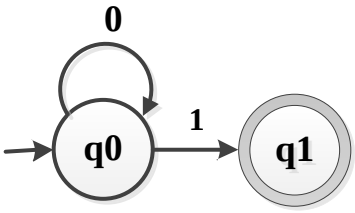


   Berdasarkan transisi pada gambar, untuk berada pada state awal diperlukan input: Ɛ, a, aa, aaa, … . Sehingga ER = a*
5. Gambar (diagram FA)



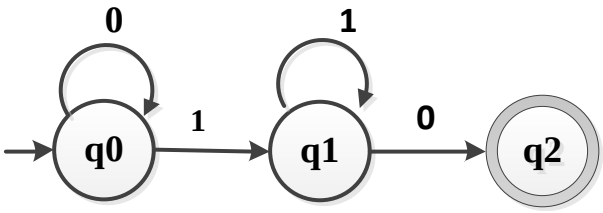
   Berdasarkan transisi pada gambar, untuk berada pada state awal diperlukan input: Ɛ, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, … . Sehingga ER = (a+b)*

6. Gambar Diagram FA



ER untuk bahasa yang diterima oleh otomaton diatas adalah 0^*1

7. Gambar Diagram FA



ER untuk bahasa yang diterima oleh otomaton pada gambar adalah 0^*11^*0

8. Tentukan ekspresi reguler pembentuk bahasa pada $\Sigma = \{a,b,c\}$, yaitu
   a. L(r) = { w $\in$ Σ^* | w memiliki tepat sebuah simbol ‘a’ }
   b. L(r) = { w $\in$ Σ^* | w mengandung tepat 3 buah simbol ‘a’}
   c. L(r) = { w $\in$ Σ^* | w mengandung kemunculan masing masing simbol minimal satu kali}
9. Tentukan ekspresi reguler pembentuk bahasa pada $\Sigma = \{0,1\}$, yaitu
   a. L(r) = { w $\in$ Σ^* | w diakhiri dengan string 01 }
   b. L(r) ={ w $\in$ Σ^* | w tidak diakhiri dengan string 01 }
   c. L(r) ={ w $\in$ Σ^* | w mengandung simbol ‘0’ sebanyak genap }
   d. L(r) ={ w $\in$ Σ^* | kemunculan string ’00’ pada w sebanyak kelipatan 3 }

Tentukan ekspresi reguler pembentuk bahasa pada $\Sigma = \{a,b\}$, yaitu $L(r) = \{ w \in \Sigma^* \mid |w| \bmod 3 = 0 \}$

Kesamaan 2 ekspresi regular:

$(a\ b)^*\ a = a\ (b\ a)^*$

Bukti:

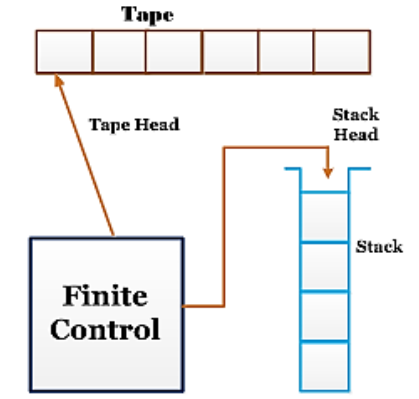
$(a\ b)^*\ a = (\varepsilon \mid (ab) \mid (abab) \mid \ldots)\ a = (\varepsilon\ a \mid (aba) \mid (ababa) \mid \ldots) = (a \mid (aba) \mid (ababa) \mid \ldots) = a\ (\varepsilon \mid (ba) \mid (baba) \mid \ldots) = a\ (b\ a)^*$

Latihan 2. Buktikan kesamaan ekspresi regular berikut:

1. $(a^* \mid b)^* = (a \mid b)^*$
2. $(a \mid b^*)^* = (a \mid b)^*$
3. $(a^*\ b)^*\ a^* = a^*\ (b\ a^*)^*$
4. $(a\ a^*)(\varepsilon \mid a) = a^*$

BAB XIII
PUSHDOWN AUTOMATA (PDA)

13.1. Pendahuluan



Gambar 13.1. Ilustrasi komponen mesih PushDown Automata

Dalam pembahasan tentang bahasa regular telah diperkenalkan suatu mesin dengan jumlah status yang terbatas atau dikenal dengan nama mesin FA.

Karena keterbtasan tempat penyimpanan informasi yang telah terjadi pada string masukan yang sedang dibaca maka bahasa yang dapat dikenalinya pun memiliki sifat yang amat terbatas yaitu bahasa regular.

Apabila mesin FA itu dilengkapi oleh suatu storage tak terhingga kapasitasnya serta dengan struktur stack (yaitu struktur dengan aturan FIFO), maka mesin ini mampu menyimpan sejumlah informasi yang disimpan pada saat pembacaan symbol-simbol di bagian depan string, dan dapat dipergunakan pada saatpembacaan symbol dibagian depan string, dan dapat dipergunakan pada saat pembacaan symbol-simbol berikutnya dari string. Mekanisme stack yang dimilki menyebabkan mesin dinamakan PushDown Automaton (PDA)

13.2. Komponen Kerja PDA

Berbeda dengan bahasa regular, pada bahasa context free model automata menggunakan ***Push Down Automata*** **(PDA)**, dimana automata ini menggunakan ***stack***.

Agar dapat menggunakan model automata pada bahasa *context free*, maka diperlukan:

a. Sebuah *stack* atau memori *push down* yang dapat menyimpan sederetan simbol dengan panjang yang sebarang dan tak berhingga.

b. Selama proses pembacaan simbol pada *stack*, simbol teratas pada sebuah *stack* PDA memiliki kemungkinan sebagai berikut:
   - *Stack* tidak dapat diubah, atau
   - Simbol pada stack teratas akan dihapus *(pop*) dan digantikan dengan simbol yang lain *(push*).

Contoh sebuah *stack*

Top *Stack* →

A
B
C

Bila dilakukan operasi *pop*, maka kondisi stack menjadi:

Top *Stack* →

D
E

Bila dilakukan operasi *push* B, maka kondisi *stack* menjadi:

Top *Stack* →

B
D
E

Sebuah PDA dinyatakan dalam 7 tupel:

$M = (Q, \sum, \Gamma, \Delta, S, F, Z)$

Q = himpunan *state*

$\sum$ = himpunan simbol input

r = simbol-simbol tumpukan / *stack*

Δ = fungsi transisi

S = *state* awal, $S \in Q$

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

Z = simbol awal tumpukan / top *stack*, $Z \in \Gamma$

Dari komponen diatas dapat disimpulkan bahwa:

- Definisi untuk Q, ∑, S, F sama dengan yang ada pada otomata hingga.
- Tupel baru adalah Γ, Z yang berhubungan dengan *stack*.
- Δ memiliki kemiripan dengan δ pada otomata hingga dengan beberapa perbedaan.

PDA dapat dianggap sebagai otomata hingga yang dilengkapi dengan *stack*. Sebuah PDA yang menerima *input*, selain bisa berpindah *state* juga bisa melakukan operasi pada *stack*.

Kondisi atau konfigurasi PDA pada suatu saat dinyatakan dengan *state* dan *stack*. Jenis transisi pada PDA;

1. Membaca simbol *input*
2. Tanpa membaca simbol *input*.

PDA dapat dianggap sebagai otomata hingga yang dilengkapi dengan *stack*. Sebuah PDA yang menerima *input*, selain bisa berpindah *state* juga bisa melakukan operasi pada *stack*. Kondisi atau konfigurasi PDA pada suatu saat dinyatakan dengan *state* dan *stack*. Jenis transisi pada PDA;

1. Membaca simbol input

Pada PDA yang membaca simbol input, terdapat sejumlah pilihan yang mungkin, bergantung pada simbol *input*, simbol pada *top-stack*, dan *state*. Setiap pilihan terdiri dari *state* berikutnya dan simbol-simbol (bias satu, beberapa, atau kosong) untuk mengganti simbol pada *top-stack*.

Penggantian simbol pada *top-stack* bisa berupa *push*, untuk satu atau beberapa simbol, atau berupa *pop*untuk simbol kosong. Setelah membuat pilihan, kemudian PDA membaca simbol *input* berikutnya.

2. Tanpa membaca simbol input

Jenis transisi tanpa membaca input adalah transisi yang dilakukan tanpa membaca *input* atau ε. Transisi ini memungkinkan PDA memanipulasi isi *stack* atau berpindah *state* tanpa membaca *input*.

Jenis-jenis PDA:

1. PDA *null stack,* yaitu PDA yang melakukan penerimaan *input* dengan *stack* kosong.
2. PDA *final state,* yaitu PDA yang melakukan penerimaan *input* yang pilihan transisinya menyebabkan PDA mencapai *final state.*

Contoh

Sebuah PDA $Q = \{q1, q2\}$ $\sum = \{a, b\}$ $\Gamma = \{A, , Z\}$
$S = q1$ $Z = Z$ $F = \{q2\}$

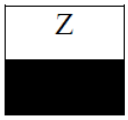
PDA tersebut memiliki fungsi transisi:

$\Delta(q1, \varepsilon, Z) = \{(q2, Z)\}$ $\Delta(q1, a, Z) = \{(q1, AZ)\}$
$\Delta (q1, b, Z) = \{(q1, BZ)\}$ $\Delta(q1, a, A) = \{(q1, AA)\}$
$\Delta (q1, b, A) = \{(q1, \varepsilon)\}$ $\Delta(q1, a, B) = \{(q1, \varepsilon)\}$
$\Delta (q1, b, B) = \{(q1, BB)\}$

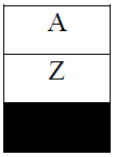
Kita bisa membaca fungsi transisi tsb. sebagai berikut.

$$\Delta (q1, a, Z) = \{(q1, AZ)\}$$

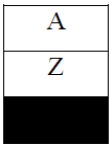
Mesin dengan konfigurasi: *State* q1 dan *top-stack* Z membaca *input* 'a'



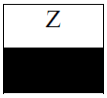
Konfigurasi menjadi: *State* q1 , *push* A ke *stack,* A menjadi *top-stack*



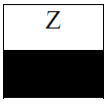
$\Delta(q1, b, A) = \{(q1, \varepsilon)\}$ Mesin dengan konfigurasi: *State* q1 dan *top-stack* A membaca *input* 'b'



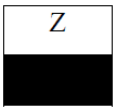
Konfigurasi menjadi: *State* q1, *pop* A dari *stack*, elemen di bawah A menjadi *top-stack*



$\Delta(q1, \varepsilon, Z) = \{(q2, Z)\}$ Mesin dengan konfigurasi: *State* q1 dan *top-stack* Z tanpa membaca *input*.



Konfigurasi menjadi: *State* q2, *stack* tidak berubah



Contoh 13.2

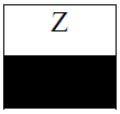
Sebuah PDA

$Q = \{q1, q2\}$	$\sum = \{a, b\}$	$\Gamma = \{A, B, Z\}$
$S = q1$	$Z = Z$	$F = \{q2\}$

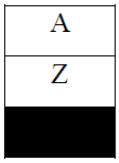
PDA tersebut memiliki fungsi transisi:

$\Delta(q1, \varepsilon, Z) = \{(q2, Z)\}$	$\Delta(q1, a, Z) = \{(q1, AZ)\}$
$\Delta(q1, b, Z) = \{(q1, BZ)\}$	$\Delta(q1, a, A) = \{(q1, AA)\}$
$\Delta(q1, b, A) = \{(q1,\varepsilon)\}$	$\Delta(q1, a, B) = \{(q1,\varepsilon)\}$
$\Delta(q1, b, B) = \{(q1, BB)\}$	

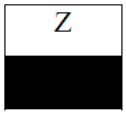
Tentukan apakah PDA diatas dapat menerima *string* ‘abba’ Penyelesaian:



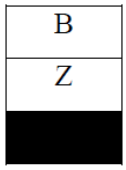
1. Konfigurasi awal mesin: state q1 , *top-stack* Z, membaca input ‘a’. Fungsi transisinya: $\Delta(q1, a, Z) = \{(q1, AZ)\}$ Konfigurasi mesin menjadi: *state* q1 dan *push* A



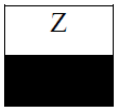
2. Membaca *input* ‘b’. Fungsi transisinya: Δ(q1, b, A) = {(q1, ε)} Konfigurasi mesin menjadi: *state* q1 dan *top-stack* di *pop*



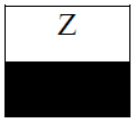
3. Membaca *input* ‘b’. Fungsi transisinya: Δ(q1, b, Z) = {(q1, BZ)} Konfigurasi mesin menjadi: *state* q1 dan B di *push*



4. Membaca *input* ‘a’. Fungsi transisinya: Δ(q1, a, B) = {(q1, ε)} Konfigurasi mesin menjadi: *state* q1 dan *top-stack* di *pop*



5. Semua input sudah selesai dibaca. Fungsi transisinya: Δ(q1, ε, Z) = {(q2, Z)} Konfigurasi mesin menjadi: *state* q2 *State* q2 berada dalam F *(final state),* maka ‘abba’ diterima oleh PDA



Jika diketahui bahasa sebagai berikut:

$$L=\{w\$w^R \mid w\in\{a,b,c,d\}^*\}$$

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

Push down automata akan mengenal sebuah bahasa dengan cara sebagai berikut:

- Sebuah word w dibaca dari kiri ke kanan.
- Sebuah automata memiliki dua buah state:

State 1: Menyimpan bagian pertama dari word.

State 2: Mengecek bagian kedua dari word.

State 1 : Selama simbol $ belum dibaca, masukkan (push) simbol kapital (huruf besar) untuk setiap simbol yang dibaca ke dalam stack ($a \rightsquigarrow A, b \rightsquigarrow B, \ldots$). Jika simbol $dibaca, maka stack tidak terjadi perubahan dan akan berpindah ke state 2.

State 2 : Lakukan pengecekan untuk setiap simbol yang dibaca, apakah ada kesesuaian dengan simbol kapital yang ada pada stack teratas. Jika ada, simbol kapital ini harus dihapus (pop) dari stack teratas.

Jika antara simbol yang dibaca dengan simbol kapital pada stack tidak ada kesesuaian, maka tidak terjadi transisi. Push down automata akan melakukan blok dan word tidak bisa diterima.

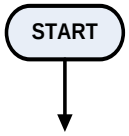
Jika pada state 2 selalu terjadi kesesuaian antara simbol yang dibaca dengan simbol kapital pada stack teratas, maka simbol terbawah dari stack # akan dihapus dan push down automata akan menerima word tersebut ketika stack telah kosong.

Sebuah Pushdown Automata (PDA) adalah kumpulan beberapa hal sbb:

$$M = (Q, \Sigma, \Gamma, q^0, \Delta, F),$$

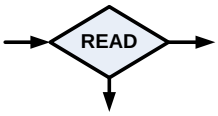
dimana:

- Q: himpunan hingga state,
- Σ: alfabet input,
- Γ: alfabet *stack*,
- $q^0 \in Q$: state awal,

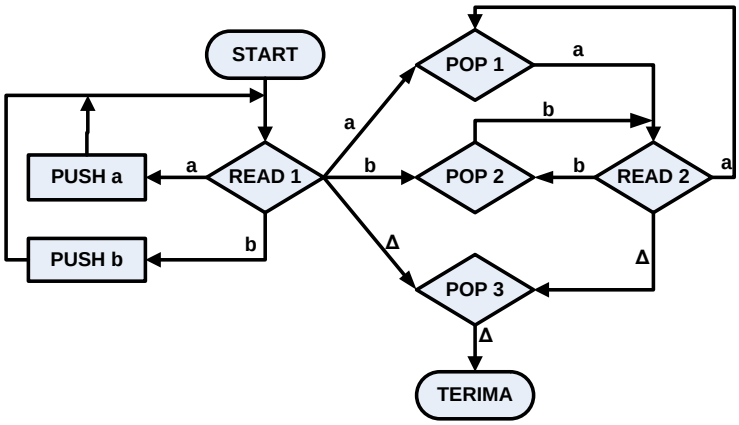


- $\Delta \in \Gamma$: simbol awal *stack*,
- $F \subseteq Q$: himpunan state penerima,

- READ



Contoh



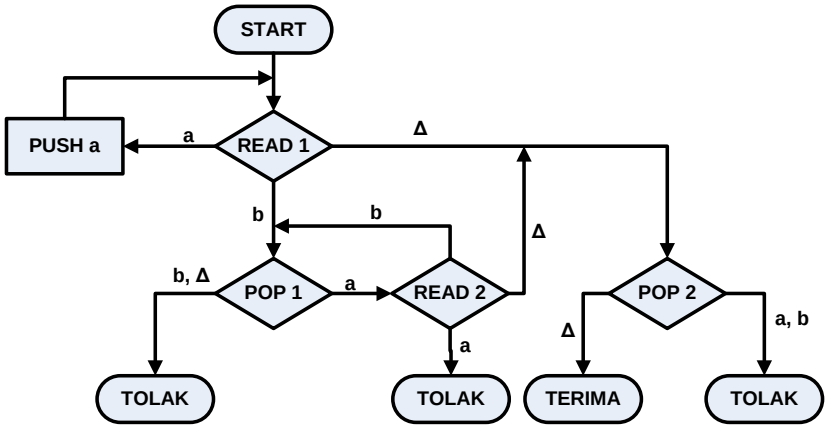
string input: babbab

STATE	TAPE								STACK							
START	b	a	b	b	a	b	Δ	..	Δ	..						
READ 1	**b**	a	b	b	a	b	Δ	..	Δ	..						
PUSH b	**b**	a	b	b	a	b	Δ	..	b	Δ	..					
READ 1	**b**	**a**	b	b	a	b	Δ	..	b	Δ	..					
PUSH a	**b**	**a**	b	b	a	b	Δ	..	a	b	Δ	..				
READ 1	**b**	**a**	**b**	b	a	b	Δ	..	a	b	Δ	..				
PUSH b	**b**	**a**	**b**	b	a	b	Δ	..	b	a	b	Δ	..			
READ 1	**b**	**a**	**b**	**b**	a	b	Δ	..	b	a	b	Δ	..			
POP 2	**b**	**a**	**b**	**b**	a	b	Δ	..	a	b	Δ	..				
READ 2	**b**	**a**	**b**	**b**	**a**	b	Δ	..	a	b	Δ	..				
POP 1	**b**	**a**	**b**	**b**	**a**	b	Δ	..	b	Δ	..					
READ 2	**b**	**a**	**b**	**b**	**a**	**b**	Δ	..	b	Δ	..					
POP 2	**b**	**a**	**b**	**b**	**a**	**b**	Δ	..	Δ	..						
READ 2	**b**	**a**	**b**	**b**	**a**	**b**	**Δ**	..	Δ	..						
POP 3	**b**	**a**	**b**	**b**	**a**	**b**	**Δ**	..	Δ	..						
TERIMA	**b**	**a**	**b**	**b**	**a**	**b**	**Δ**	..	Δ	..						

- abba

STATE	TAPE						STACK					
START	a	b	b	a	Δ	..	Δ	..				
READ 1	**a**	b	b	a	Δ	..	Δ	..				
PUSH a	**a**	b	b	a	Δ	..	a	Δ	..			
READ 1	**a**	**b**	b	a	Δ	..	a	Δ	..			
PUSH b	**a**	**b**	b	a	Δ	..	b	a	Δ	..		
READ 1	**a**	**b**	**b**	a	Δ	..	b	a	Δ	..		
POP 2	**a**	**b**	**b**	a	Δ	..	a	Δ	..			
READ 2	**a**	**b**	**b**	**a**	Δ	..	a	Δ	..			
POP 1	**a**	**b**	**b**	**a**	Δ	..	Δ	..				
READ 2	**a**	**b**	**b**	**a**	**Δ**	..	Δ	..				
POP 3	**a**	**b**	**b**	**a**	**Δ**	..	Δ	..				
TERIMA	**a**	**b**	**b**	**a**	**Δ**	..	Δ	..				

CONTOH 2



Input string - aaabbb

STATE	TAPE								STACK							
START	a	a	a	b	b	b	Δ	..	Δ	..						
READ 1	**a**	a	a	b	b	b	Δ	..	Δ	..						
PUSH a	**a**	a	a	b	b	b	Δ	..	a	Δ	..					
READ 1	**a**	**a**	a	b	b	b	Δ	..	a	Δ	..					
PUSH a	**a**	**a**	a	b	b	b	Δ	..	a	a	Δ	..				
READ 1	**a**	**a**	**a**	b	b	b	Δ	..	a	a	Δ	..				
PUSH a	**a**	**a**	**a**	b	b	b	Δ	..	a	a	a	Δ	..			
READ 1	**a**	**a**	**a**	**b**	b	b	Δ	..	a	a	a	Δ	..			
POP 1	**a**	**a**	**a**	**b**	b	b	Δ	..	a	a	Δ	..				
READ 2	**a**	**a**	**a**	**b**	**b**	b	Δ	..	a	a	Δ	..				
POP 1	**a**	**a**	**a**	**b**	**b**	b	Δ	..	a	Δ	..					
READ 2	**a**	**a**	**a**	**b**	**b**	**b**	Δ	..	a	Δ	..					
POP 1	**a**	**a**	**a**	**b**	**b**	**b**	Δ	..	Δ	..						
READ 2	**a**	**a**	**a**	**b**	**b**	**b**	**Δ**	..	Δ	..						
POP 2	**a**	**a**	**a**	**b**	**b**	**b**	**Δ**	..	Δ	..						
TERIMA	**a**	**a**	**a**	**b**	**b**	**b**	**Δ**	..	Δ	..						

- Definisi: PDA adalah pasangan 7 tuple

  $M = (Q, \Sigma, \Gamma, q^0, Z^0, \delta, F)$, dimana:
  Q: himpunan hingga state,
  Σ: alfabet input,

Γ: alfabet *stack*,

$q^0 \in Q$: state awal,

$Z^0 \in \Gamma$: simbol awal *stack*,
$F \subseteq Q$: himpunan state penerima,
fungsi transisi δ: $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$ (himpunan bagian dari $Q \times \Gamma^*$)

- Untuk state $q \in Q$, simbol input $a \in \Sigma$, dan simbol *stack* $X \in \Gamma$, $\delta(q, a, X) = (p, \alpha)$ berarti: PDA bertransisi ke state p dan mengganti X pada stack dengan string α.
- Konfigurasi PDA pada suatu saat dinyatakan sebagai triple (q, x, α), dimana:
  $q \in Q$: state pada saat tersebut, $x \in \Sigma^*$: bagian string input yang belum dibaca, dan $\alpha \in \Gamma^*$: string yang menyatakan isi *stack* dengan karakter terkiri menyatakan *top of stack*.
- Misalkan $(p, ay, X\beta)$ adalah sebuah konfigurasi, dimana: $a \in \Sigma$, $y \in \Sigma^*$, $X \in \Gamma$, dan $\beta \in \Gamma^*$. Misalkan pula $\delta(p, a, X) = (q, \gamma)$ untuk $q \in Q$ dan $\gamma \in \Gamma^*$. Dapat kita tuliskan bahwa: $(p, ay, X\beta) \Rightarrow (q, y, \gamma\beta)$.

<u>Contoh (PDA Deterministik):</u>

$$\text{PDA: } M = (Q, \Sigma, \Gamma, q^0, Z^0, \delta, F)$$

Pengenal palindrome $L = \{xcx^T \mid x \in (a \mid b)^*\}$, dimana x^T adalah cermin(x), mempunyai tuple: $Q = \{q^0, q^1, q^2\}$, $F = \{q^2\}$, $\Sigma = \{a, b, c\}$, $\Gamma = \{A, B, Z^0\}$, dan fungsi transisi δ terdefinisi melalui tabel berikut:

No.	State	Input	*TopStack*	Hasil
1	q_0	a	Z_0	(q_0, AZ_0)
2	q_0	b	Z_0	(q_0, BZ_0)
3	q_0	a	A	(q_0, AA)
4	q_0	b	A	(q_0, BA)
5	q_0	a	B	(q_0, AB)
6	q_0	b	B	(q_0, BB)

No.	State	Input	*TopStack*	Hasil
7	q_0	c	Z_0	(q_1, Z_0)
8	q_0	c	A	(q_1, A)
9	q_0	c	B	(q_1, B)
10	q_1	a	A	(q_1, ε)
11	q_1	b	B	(q_1, ε)
12	q_1	ε	Z_0	(q_2, Z_0)

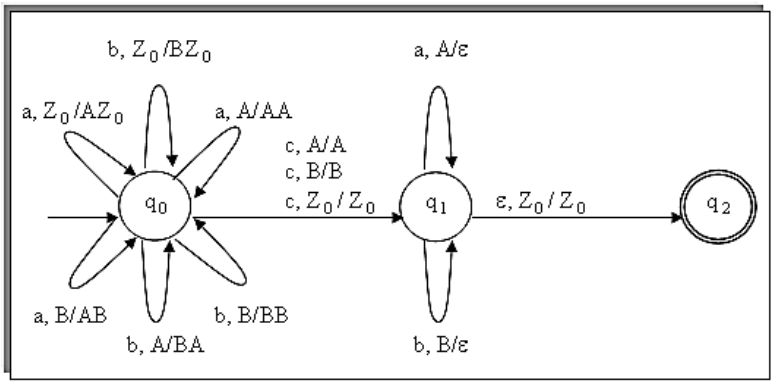
Sebagai contoh, perhatikan bahwa fungsi transisi No. 1 dapat dinyatakan sebagai: $\delta(q^0, a, Z^0) = (q^0, aZ^0)$. Pada tabel transisi tersebut terlihat bahwa pada state q^0 PDA akan melakukan PUSH jika mendapat input a atau b dan melakukan transisi state ke state q^1 jika mendapat input c. Pada state q^1 PDA akan melakukan POP.
Berikut ini pengenalan dua string oleh PDA di atas:

1. abcba: $(q^0, abcba, Z^0) \Rightarrow (q^0, bcba, AZ^0)$ (1)
   $\Rightarrow (q^0, cba, BAZ^0)$ (4)
   $\Rightarrow (q^1, ba, BAZ^0)$ (9)
   $\Rightarrow (q^1, a, AZ^0)$ (11)
   $\Rightarrow (q^1, \varepsilon, Z^0)$ (10)
   $\Rightarrow (q^2, \varepsilon, Z^0)$ (12) **(*diterima*)**
2. acb: $(q^0, acb, Z^0) \Rightarrow (q^0, cb, AZ^0)$ (1)
   $\Rightarrow (q^1, b, AZ^0)$ (8), **(*crash* → *ditolak*)**
3. ab: $(q^0, ab, Z^0) \Rightarrow (q^0, b, AZ^0)$ (1)
   $\Rightarrow (q^0, \varepsilon, BAZ^0)$ (4) **(*crash* → *ditolak*)**

Penerimaan dan penolakan tiga string di atas dapat dijelaskan sebagai berikut:

1. String abcba diterima karena *tracing* sampai di state penerima (q^2) dan string "abcba" selesai dibaca (string yang belum dibaca = ε)
2. String acb ditolak karena konfigurasi akhir (q^1, b, a z^0) sedangkan fungsi transisi δ(q^1, b, a) tidak terdefinsi
3. String ab ditolak karena konfigurasi akhir (q^0, ε, baz^0) sedangkan fungsi transisi δ(q^0, ε, b) tidak terdefinsi

Ilustrasi graf fungsi transisi PDA di atas ditunjukkan melalui gambar berikut:



- Notasi (p, ay, Xβ) ⇒ (q, y, γβ) dapat diperluas menjadi: (p, x, α) ⇒* (q, y, β), yang berarti konfigurasi (q, y, β) di capai melalui sejumlah (0 atau lebih) transisi.
- Ada dua cara penerimaan sebuah kalimat oleh PDA, yang masing-masing terlihat dari konfigurasi akhir, sebagaimana penjelasan berikut:

  Jika M = (Q, Σ, Γ, q^0, Z^0, δ, F) adalah PDA dan x ∈Σ*, maka x *diterima dengan state akhir* (*accepted by final state*) oleh PDA M

jika: $(q^0, x, Z^0) \Rightarrow^* (q, \varepsilon, \alpha)$ untuk $\alpha \in \Gamma^*$ dan $q \in A$. x *diterima dengan stack hampa* (*accepted by empty stack*) oleh PDA

M jika: $(q^0, x, Z^0) \Rightarrow^* (q, \varepsilon, \varepsilon)$ untuk $q \in Q$.

<u>Contoh (PDA Non-Deterministik):</u>

PDA $M = (Q, \Sigma, \Gamma, q^0, Z^0, \delta, F)$ pengenal palindrome

$L = \{xx^T \mid x \in (a \mid b)^*\}$ mempunyai komponen tuple berikut:

$Q = \{q^0, q^1, q^2\}$, $F = \{q^2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, Z^0\}$, dan fungsi transisi δ terdefinisi melalui tabel berikut:

No.	St.	In.	*TS*	Hasil
1	q_0	a	Z_0	$(q_0, aZ_0),(q_1, Z_0)$
2	q_0	b	Z_0	$(q_0, bZ_0),(q_1, Z_0)$
3	q_0	a	a	$(q_0, aa),(q_1, a)$
4	q_0	b	a	$(q_0, ba),(q_1, a)$
5	q_0	a	b	$(q_0, ab),(q_1, b)$
6	q_0	b	b	$(q_0, bb),(q_1, b)$

No.	St.	In.	*TS*	Hasil
7	q_0	ε	Z_0	(q_1, Z_0)
8	q_0	ε	a	(q_1, a)
9	q_0	ε	b	(q_1, b)
10	q_1	a	a	(q_1, ε)
11	q_1	b	b	(q_1, ε)
12	q_1	ε	Z_0	(q_2, ε)

Pada tabel transisi tersebut terlihat bahwa pada state q^0 PDA akan melakukan PUSH jika mendapat input a atau b dan melakukan transisi *state* ke state q^1 jika mendapat input ε. Pada state q^1 PDA akan melakukan POP. Kedua Contoh di atas menunjukkan bahwa PDA dapat dinyatakan sebagai mesin PUSH-POP.

Berikut ini pengenalan string "baab" oleh PDA di atas:

1. $(q^0, baab, Z^0) \Rightarrow (q^0, aab, bZ^0)$ (2 kiri)

   $\Rightarrow (q^0, ab, abZ^0)$ (5 kiri)

   $\Rightarrow (q^1, ab, abZ^0)$ (3 kanan)

   $\Rightarrow (q^1, b, bZ^0)$ (11)

$\Rightarrow (q^1, \varepsilon, Z^0)$ (10)

$\Rightarrow (q^2, \varepsilon, Z^0)$ (12) **(*diterima*)**

2. $(q^0, baab, Z^0) \Rightarrow (q^1, baab, Z^0)$ (2 kanan) **(*crash* → *ditolak*)**

3. $(q^0, baab, Z^0) \Rightarrow (q^0, aab, bZ^0)$ (2 kiri)

$\Rightarrow (q^0, ab, abZ^0)$ (5 kiri)

$\Rightarrow (q^0, b, aabZ^0)$ (3 kiri)

$\Rightarrow (q^1, b, aabZ^0)$ (4 kanan) **(*crash* → *ditolak*)**

4. $(q^0, baab, Z^0) \Rightarrow (q^0, aab, bZ^0)$ (2 kiri)

$\Rightarrow (q^0, ab, abZ^0)$ (5 kiri)

$\Rightarrow (q^0, b, aabZ^0)$ (3 kiri)

$\Rightarrow (q^0, \varepsilon, baabZ^0)$ (4 kiri)

$\Rightarrow (q^1, \varepsilon, baabZ^0)$ (9) **(*crash* → *ditolak*)**

q0,aba,z = q0,ba,az = q1, a, az = q1, ε, z =q2, ε,ε

13.3. PDA untuk Suatu Tata Bahasa Bebas Konteks

PDA adalah merupakan penerima bahasa-bahasa bebas konteks, sehingga dari suatu tata bahasa bebas konteks kita dapat memperoleh sebuah PDA, begitu juga sebaliknya. Sebuah PDA bisa dibuat untuk kumpulan aturan produksi dari suatu tata bahasa bebas konteks.

Langkah-langkahnya adalah sebagai berikut:

1. Definisikan:

   Q = {q1, q2, q3 }

   S = q1

   F = { q3 }

   ∑ = simbol terminal

Untuk yang berhubungan dengan *stack*, tentukan:

Γ = semua simbol variabel, simbol terminal, dan Z (simbol awal *stack*)

2. Mesin ini dimulai dengan mem-*push* Z pada top *stack*.

   Pada setiap langkah berikutnya dilakukan salah satu dari dua hal berikut:

   - Jika *top-stack* adalah variabel , misal A, kita gantikan dengan ruas kanan dari A, misal A□ w, maka kita ganti dengan A dengan w.
   - Jika *top-stack* adalah terminal, dan sama dengan simbol masukan berikutnya, maka *top-stack* kita *pop* dari *stack*.

3. Berdasarkan aturan diatas, kita dapat mengkonstruksi empat tipe transisi berikut.

   - Δ (q1 , ε, Z) = {(q2 , SZ)} untuk mem-*push* simbol awal (S) ke *stack*.
   - Δ (q2 , ε, A) = {(q2 , w) | A → w adalah sebuah simbol produksi dalam tata bahasa bebas konteks itu} untuk semua variabel A.
   - Δ(q2 , a, a) = {(q2 ,ε)} untuk setiap simbol terminal (untuk mem-*pop* pembandingan terminal yang sama)

   Δ (q2 , ε, Z) = {(q3 , Z)}, bila selesai membaca semua *input* dan *top-stack* adalah Z, berarti *string input* sukses diterima oleh PDA (q3 *state* akhir)

Contoh 13.4

Sebuah tata bahasa bebas konteks, D □ aDa | bDb | c PDA nya dapat dikonstruksi menjadi:

Q = {q1, q2, q3 }

S = q1

F = { q3 }

∑ = {a, b, c}

Γ = {D, a, b, c, Z}

Fungsi transisinya:

$$\Delta\ (q1, \varepsilon, Z) = \{(q2, DZ)\}$$

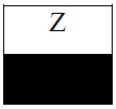
$$\Delta\ (q2, \varepsilon, D) = \{(q2, aDa), (q2, bDb), (q2, c)\}$$

$$\Delta\ (q2, a, a) = \Delta\ (q2, b, b) = \Delta\ (q2, c, c) = \{(q2, \varepsilon)\}$$

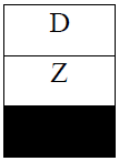
$$\Delta\ (q2, \varepsilon, Z) = \{(q3, Z)\}$$

Dari aturan produksi yang ada, tata bahasa bebas konteks tersebut bias menurunkan untai 'aca' dari D ➔ aDa ➔ aca Karena tata bahasa bebas konteks bisa menurunkan *string* 'aca' , maka PDA juga harus dapat menerima untai tersebut.

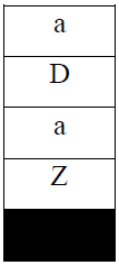
Langkah pemeriksaan:



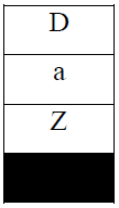
1. Konfigurasi awal mesin: state q1 , *top-stack* Z, tanpa menerima *input* (ε). Fungsi transisinya: Δ(q1, ε, Z) = {(q2, DZ)} Konfigurasi mesin menjadi: *state* q2 dan *push D*



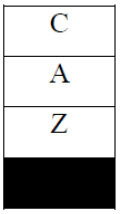
2. Tanpa menerima *input* (ε). Fungsi transisinya: Δ (q2, ε, D) = {(q2, aDa)} Konfigurasi mesin menjadi: *state* q2, *pop top-stack push*'aDa'



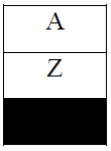
3. Menerima *input* ‘a’ Fungsi transisinya: Δ (q2, a, a) = {(q2, ε)} Konfigurasi mesin menjadi: *state* q2 , *pop top-stack*



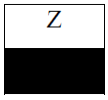
4. Tanpa menerima *input* (ε) Fungsi transisinya: Δ(q2, ε, D) = {(q2, c)} Konfigurasi mesin menjadi: *state* q2 , *pop top-stack, push* c



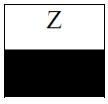
5. Menerima *input* ’c’ Fungsi transisinya: Δ(q2, c, c) = {(q2, ε)} Konfigurasi mesin menjadi: *state* q2 , *pop top-stack*



6. Menerima *input* ’a’ Fungsi transisinya: Δ(q2, a, a) = {(q2, ε)} Konfigurasi mesin menjadi: *state* q2 , *pop top-stack*



7. Tanpa menerima *input* (ε) Fungsi transisinya: Δ(q2, ε, Z) = {(q3, Z)} Konfigurasi mesin menjadi: *state* q3



Tidak ada transisi lagi dari q3. Karena q3 *state* akhir dan semua *input* sudah selesai dibaca, sehingga menandakan untai ‘aca’ diterima oleh PDA tersebut.

13.4. Deskripsi Seketika Pada PDA

Langkah 1 s.d. 7 pada contoh soal sebelumnya, dapat juga dinyatakan dalam suatu notasi yang disebut deskripsi seketika (*instantaneous description*). Deskripsi seketika tersebut digunakan untuk menyatakan secara formal konfigurasi PDA pada suatu saat. Perubahan dari suatu kondisi ke kondisi berikutnya dipisahkan dengan tanda ‘$\vdash$’. Konfigurasi suatu saat dapat dinyatakan dengan *triplet*: (q, w, u), Dimana q menyatakan *state*, w adalah *string* yang belum dibaca, sedangkan u adalah isi *stack* dengan simbol terkiri adalah *top-stack*.

Contoh

Kerjakan contoh 14.4 dengan deskripsi seketika.

Q = {q1, q2, q3 }

S = q1

F = { q3 }

$\sum$ = {a, b, c}

Γ = {D, a, b, c, Z}

Fungsi transisinya:

$$\Delta(q1, \varepsilon, Z) = \{(q2, DZ)\}$$

$$\Delta (q2, \varepsilon, D) = \{(q2, aDa), (q2, bDb), (q2, c)\}$$

$$\Delta (q2, a, a) = \Delta (q2, b, b) = \Delta (q2, c, c) = \{(q2, \varepsilon)\}$$

$$\Delta (q2, \varepsilon, Z) = \{(q3, Z)\}$$

Tentukan apakah PDA diatas dapat menerima *string* ‘aca’

Tahapan nomor 1 s.d. 7 dapat dinyatakan sebagai berikut:

(q1, aca, Z) $\vdash$ (q2, aca, DZ) $\vdash$ (q2, aca, aDaZ) $\vdash$ (q2, ca, DaZ) $\vdash$ (q2, ca, caZ) $\vdash$ (q2 , a, aZ) $\vdash$ (q2 , $\vdash$, Z) $\vdash$ (q3, ε, Z)

13.5. Latihan soal

1. Sebuah PDA Q = {q1, q2} ; ∑ = {0, 1, 2} ; Γ= {Z, B, G} ; S = {q1 , q2} ; Z = Z ; F = ∅

   PDA tersebut memiliki fungsi transisi:

   Δ(q1, 0, Z) = {(q1, BZ)} Δ (q1, 0, B) = {(q1, BB)}
   Δ (q1, 0, G) = {(q1, BG)} Δ (q1, 2, Z) = {(q2, Z)}
   Δ (q1, 2, B) = {(q2, B)} Δ (q1, 2, G) = {(q2, G)}
   Δ (q2, 0, B) = {(q2, ε)} Δ (q2, ε, Z) = {(q2, ε)}
   Δ (q1, 1, Z) = {(q1, GZ)} Δ (q1, 1, B) = {(q1, GB)}
   Δ (q1, 1, G) = {(q1, GG)} Δ (q2, 1, G) = {(q2, ε)}

   Tentukan apakah PDA diatas dapat menerima *string* '121'

Penyelesaian:

Z

a. Konfigurasi awal mesin: *state* q1 , *top-stack* Z, menerima input '1'. Fungsi transisinya: Δ (q1, 1, Z) = {(q1, GZ)}
   Konfigurasi mesin menjadi: *state* q1 dan *push* G

G
Z

b. Membaca input '2' Fungsi transisinya: Δ(q1, 2, G) = {(q2, G)}
   Konfigurasi mesin menjadi: *state* q2 dan *stack* tetap

G
Z

c. Membaca input '1' Fungsi transisinya: Δ(q2, 1, G) = {(q2, ε)}
   Konfigurasi mesin menjadi: *state* q2 dan G di *pop*

Z

d. Tanpa membaca input (ε) Fungsi transisinya: Δ(q2, ε, Z) = {(q2, ε)} Konfigurasi mesin menjadi: *state* q2 dan Z di *pop Stack* kosong

Latihan

PDA pada latihan sebelumnya:

Q = {q1, q2, q3 }

S = q1

F = { q3 }

∑ = {a, b, c}

Γ= {D, a, b, c, Z}

Fungsi transisi latihan 2:

Δ(q1, ε, Z) = {(q2, DZ)}

Δ (q2, ε, D) = {(q2, aDa), (q2, bDb), (q2, c)}

Δ (q2, a, a) = Δ (q2, b, b) = Δ (q2, c, c) = {(q2, ε)}

Δ (q2, ε, Z) = {(q3, Z)}

Kerjakan latihan 2, dengan deskripsi seketika.

Fungsi transisi latihan 2:

Δ (q1, ε, Z) = {(q2, DZ)}

Δ (q2, ε, D) = {(q2, aDa), (q2, bDb), (q2, c)}

Δ (q2, a, a) = Δ (q2, b, b) = Δ (q2, c, c) = {(q2, ε)}

Δ (q2, ε, Z) = {(q3, Z)}

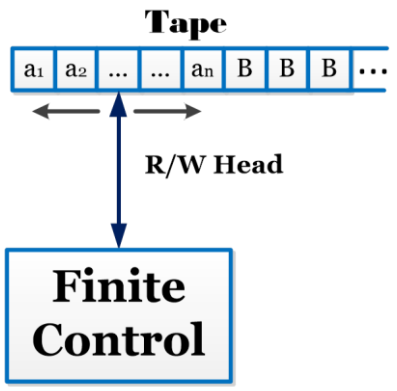
Penyelesaian

(q1 , bcb, Z) ├ (q2, bcb, DZ) (q2, bcb, bDbZ) ├ (q2, cb, DaZ) ├ (q2, cb, cbZ) ├ (q2 , b, bZ) ├ (q2 , ε, Z) ├ (q3, ε, Z)

BAB XIV
MESIN TURING

14.1. Pendahuluan

Mesin Turing adalah model yang sangat sederhana dari komputer. Secara esensial, mesin Turing merupakan sebuah finite automaton yang miliki sebuah tape tunggal dengan panjang tak terhingga yang dapat membaca dan menulis data. Mesin Turing menggunakan notasi seperti ID-ID pada PDA untuk menyatakan konfigurasi dari komputasinya.



Gambar 14.1. Ilustrasi Mesin Turing

Stack pada PDA memiliki keterbatasan akses. Elemen yang dapat diakses hanya elemen yang ada pada top stack. Pada Mesin Turing, memori akan berupa suatu tape yang pada dasarnya merupakan array dari sel-sel penyimpanan. Visualisasi dari sebuah mesin Turing diberikan oleh gambar 14.1

Mesin terdiri dari sebuah finite control, yang dapat berada dalam sebuah himpunan berhinggadari state. Terdapat sebuah tape yang dibagi ke dalam kotak-kotak atau sel-sel. Setiap sel dapat menampung sebuah dari sejumlah berhingga dari simbol. Pada awalnya, input yang merupakan string dari simbol dengan panjang berhingga dipilih dari input alphabet, ditempatkanpada tape. Sel-sel tape yang lain, perluasan secara infinite ke kiri dan ke kanan, pada awalnya menampung simbol khusus yang dinamakan blank.

Blank bukan sebuah input symbol, dan mungkin terdapat symbol tape yang lain disamping input symbol dan blank. Ada sebuah tape head

yang selalu ditempatkan pada salah satu dari sel-sel tape. Mesin turing dikatakan men-scan sel tersebut. Pada awalnya, tape head berada pada sel paling kiri yang menampung input. Sebuah pergerakan mesin Turing adalah sebuah fungsi dari state dari finite control tape symbol yang di-scan.

Perbedaan mesin Turing dengan FSA dan PDA

FSA dan PDA		Mesin Turing	
1	Pita masukan hanya dapat dibaca.	1	Pita masukan dapat dibaca dan ditulis.
2	Head hanya dapat digerakkan ke kanan	2	Head dapat digerakkan ke kiri maupun ke kanan.
3	Pita masukan hanya berisi string masukan.	3	Pita masukan juga berfungsi sebagai tempat penyimpanan yang pada pengaksesannya tidak dibatasi. *)

14.2. Komponen Kerja Mesin Turing

Mesin Turing bisa dianalogikan seperti komputer sederhana, dengan sejumlah state sebagai memori, pita sebagai secondary storage, fungsi transisi sebagai program. Sebuah mesin Turing secara formal dinyatakan dalam 7 tupel, yaitu:

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

dimana:

Q: himpunan hingga state,

Σ: alfabet input,

Γ: alfabet Pita , $\Sigma \subseteq \Gamma$

$S \in Q$: state awal,

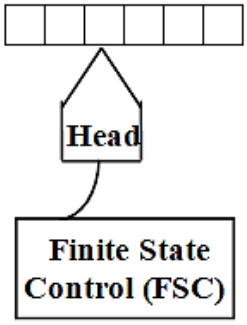
$Ь \in \Gamma$: simbol pita kosong (blank)

$F \subseteq Q$: himpunan state akhir/penerima,

fungsi transisi δ: $Q \times \Gamma \rightarrow (Q \times \Gamma \times (R,L))$

$$\delta(q_0, 0) = (q_1, 1, R)$$

Ilustrasi TM sebagai sebuah ‘mesin’:



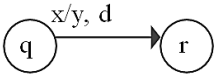
: Pita TM ➔Setiap sel berisi sebuah karakter dari kalimat yang akan dikenali. Di kanan kiri kalimat terdapat tak hingga simbol hampa

: membaca dan menulisi sel pita TM, bisa bergerak ke kiri atau ke kanan

FSC : otak dari TM, diimplementasikan dari algoritma pengenalan kalimat

Ciri-ciri TM:

- Selain sebagai penerima bahasa, TM dapat juga berfungsi sebagai 'komputer'; fungsi TM yang kedua inilah inti dari kuliah Teori Komputasi. Pembahasan TM sebagai mesin penerima hanyalah sebagai perkelanalan akan TM.
- TM terdiri dari pita, head, dan sebuah control.
- Pita TM bisa tunggal atau banyak, finite di kiri infinite di kanan atau double infinite.
- Pita TM terdiri dari sel-sel.
- Petiap sel pita berisi sebuah karakter $\in (\Gamma \cup \{\varepsilon\})$. Sesaat sebelum beroperasi sel-sel pita terkiri berisi karakter-karakter string input x $\in \Sigma^*$.
- Head TM dapat membaca/menulisi pita, dapat bergerak ke kiri (L), kanan (R), atau diam (S).
- Sebuah transisi δ (q, x) = (r, y, d), dimana $q \in Q$, $r \in (Q \cup \{h\})$, x dan $y \in (\Gamma \cup \{\varepsilon\})$, dan $d \in \{R, L, S\}$, berarti:
  TM berada dalam stata q, headnya sedang membaca karakter x pada sebuah sel pita. Selanjutnya TM akan mengganti x dengan y sedangkan headnya akan bergeser ke arah d.

  Notasi graf : 

  - Jika r = h maka TM halt dan tidak beraksi lagi. Pada kasus ini string input diterima.

- Jika TM tidak berkasi lagi pada stata $r \neq h$ maka string ditolak.

- Konfigurasi sesaat dari TM: $(q, x \underline{a} y)$, dimana $q \in (Q \cup \{h\})$, x dan $y \in (\Gamma \cup \{\varepsilon\})^*$, dan $a \in (\Gamma \cup \{\varepsilon\})$ adalah isi sel pita yang sedang dibaca head.
- Notasi tracing oleh TM:

$$(q, x \underline{a} y) \vdash (r, z \underline{b} w):$$

TM beralih konfigurasi melalui satu gerakan

$$(q, x \underline{a} y) \vdash^* (r, z \underline{b} w):$$

TM beralih konfigurasi melalui nol atau lebih gerakan

contoh: konfigurasi TM sekarang adalah $(q, aba\underline{a}\varepsilon)$ dan terdapat $\delta(q, a) = (q, \varepsilon, L)$, maka:

$(q, aba\underline{a}\varepsilon)$ $(q, ab\underline{a}\varepsilon)$ (*infinite* ε *dinotasikan sebagai single* ε) $(q, aba\underline{a}\varepsilon) \vdash^* (q, a\underline{b}\varepsilon)$ (*menyatakan dua move*)

- TM menerima string x: $(q^o, \underline{\varepsilon}x) \vdash (h, y\underline{a}z)$, dimana:

  q^o = stata awal, $x \in \Sigma^*$, h = stata halt, y dan
  $z \in (\Gamma \cup \{\varepsilon\})^*$, $a \in (\Gamma \cup \{\varepsilon\})$

Karena grammar RG $\subseteq$ CFG $\subseteq$ CSG $\subseteq$ UG maka TM sebagai penerima bahasa UG juga harus menerima bahasa-bahasa CSG, CFG, dan RG. Berikut ini adalah dua ilustrasi yang menunjukkan bahwa TM bisa dibangun dari FA (penerima RG) serta TM sebagai penerima palindrome (salah satu bahasa CFG).

14.3. Mekanisme Kerja Mesin Turing

Dalam satu pergerakan, mesin Turing akan:

- Merubah state. Next state dapat sama dengan current state.
- Menulis sebuah tape symbol dalam sel yang di-scan. Tape symbol ini mengganti symbol apapun yang ada dalam sel tersebut. Secara opsional, simbol yang dituliskan dapat sama dengan simbol yang sekarang ada dalam tape.

- Memindahkan tape head ke kiri atau ke kanan.

Pada mesin Turing, memori berupa pita yang berupa array (deretan) sel-sel penyimpanan. Setiap sel mampu menyimpan sebuah simbol tunggal. Pita tersebut tidak mempunyai sel pertama dan sel terakhir. Pita dapat memuat informasi dalam jumlah tak terbatas, dan dapat diakses bagian manapun dari pita dengan urutan bagaimanapun. Terdapat sebuah head yang menunjukkan posisi yang diakses pada pita. Head dapat bergerak ke kanan atau ke kiri untuk membaca input dari pita dan sekaligus juga bias melakukan penulisan pada pita/mengubah isi pita.

Pembacaan fungsi transisi: δ(q1, a) = (q1, a, R) pada *state* q1 head menunjuk karakter 'a' pada pita, menjadi *state* q1, *head* bergerak ke kanan δ(q1, b) = (q1, a, R). Pada *state* q1 head menunjuk karakter 'b' pada pita, menjadi *state* q1, *head* menulis karakter 'a' lalu bergerak ke kanan δ(q1, b) = (q2, b, L). Pada *state* q1 head menunjuk karakter 'b' pada pita, menjadi *state* q1, *head* bergerak ke kanan.

Contoh 14.1

Misal terdapat mesin Turing:

Q = {S, F}

∑ = {a, b}

Γ = {a, b, b}

F = {F}

S = {S}

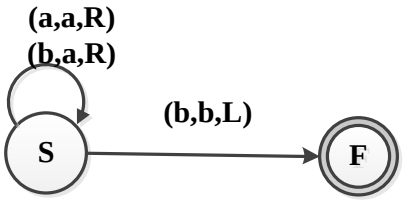
Fungsi transisinya:

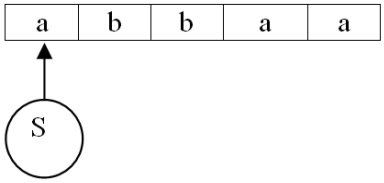
δ(S, a) = (S, a, R)

δ (S, b) = (S, a, R)

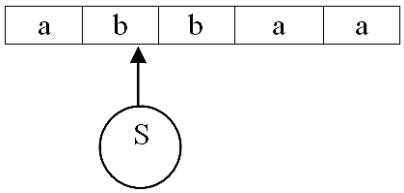
δ (S, b) = (F, b, L)

Misal pita yang akan dibaca adalah 'abbaa'

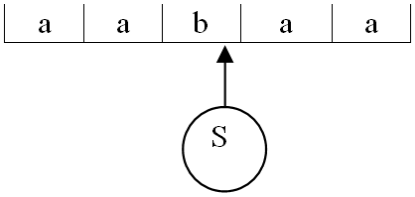




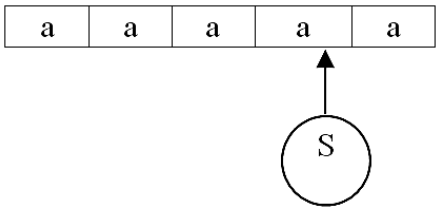
1. Fungsi transisi δ(q1, a) = (q1, a, R). Pada *state* q1 head menunjuk karakter ‘a’, menjadi *state* q1, *head* bergerak ke kanan.



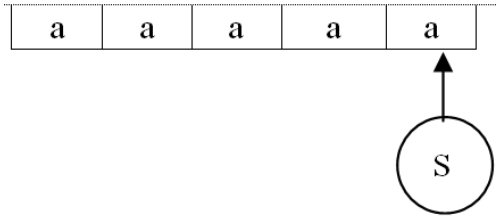
2. Fungsi transisi δ (q1, b) = (q1, a, R). Pada *state* q1 head menunjuk karakter ‘b’, menjadi *state* q1, *head* menulis karakter ‘a’, lalu bergerak ke kanan



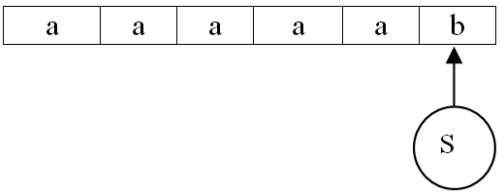
3. Fungsi transisi δ (q1, b) = (q1, a, R). Pada *state* q1 head menunjuk karakter ‘b’, menjadi *state* q1, *head* menulis karakter ‘a’, lalu bergerak ke kanan.



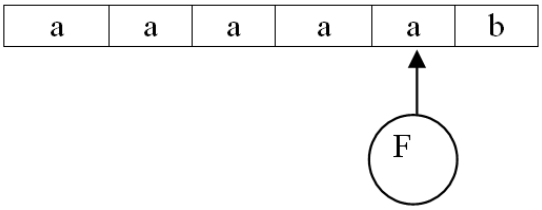
4. Fungsi transisi δ (q1, a) = (q1, a, R). Pada *state* q1 head menunjuk karakter ‘a’, menjadi *state* q1, *head* bergerak ke kanan



5. Fungsi transisi δ(q1, a) = (q1, a, R). Pada *state* q1 head menunjuk karakter ‘a’, menjadi *state* q1, *head* bergerak ke kanan



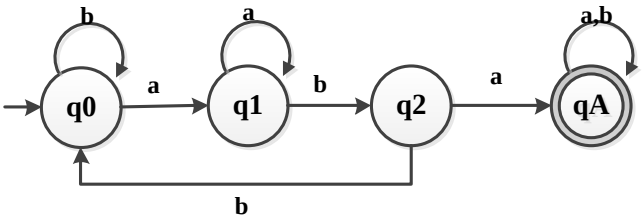
6. Fungsi transisi δ (q1, b) = (q2, b, L). Pada *state* q1 head menunjuk karakter ‘b’, menjadi *state* q1, *head* bergerak ke kiri



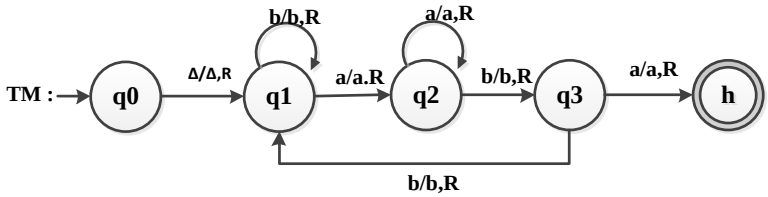
7. Tidak ada transisi dari q2, mesin Turing berhenti (*halt*). Karena q2 adalah state akhir, maka input diterima

<u>TM untuk bahasa regular (RG).</u>

1. L = { x ∈ {a│b}* │ x mengandung substring aba}(*coba tentukan grammar L ini* !)
   L bisa dituliskan sebagai ekspresi regular r = {a│b}* aba {a│b}*.
   FA-nya adalah:



Head FA tidak pernah melakukan overwrite dan juga selalu bergerak ke kanan. Karena itu TM yang ekuivalen dengan FA tersebut adalah (Δ = simbol hampa = ε):



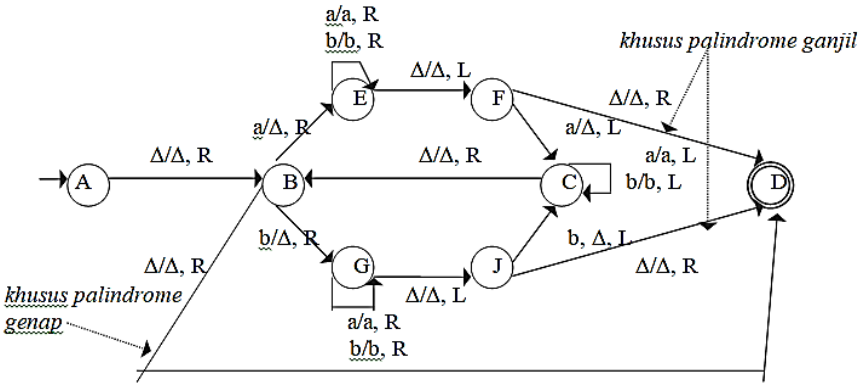
Perhatikan bahwa TM ini akan berhenti beraksi begitu sebuah substring aba ditemukan. Perhatikan juga head selalu bergerak ke kanan dan sebuah karakter dioverwrite dengan karakter yang sama (identik dengan tidak terjadi overwrite).

<u>TM Penerima Palindrome</u>

Algoritma proses pengenalan palindrome:

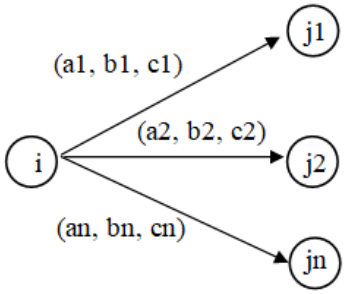
1. String masukan wcw^R ditempatkan pada pita sebagai $\Delta\, wcw^R\, \Delta$.
2. Head TM akan membaca simbol pertama dari w, menghapus simbol tersebut (yaitu menggantinya dengan Δ), lalu menuju simbol terakhir dari w^R. Jika simbol terakhir w^R ini sama dengan simbol pertama w maka TM akan menghapus simbol terakhir w^R tersebut (juga dengan menggantinya dengan Δ).
3. Head TM kembali ke simbol kedua w, ketiga w, dan seterusnya sampai ke pusat string (palindrome ganjil) atau hingga tersisa dua

simbol (simbol terakhir w dan simbol terakhir w^R) dengan perlakuan yang sama seperti pada langkah 2.



Ilustrasi TM sebagai sebuah graf berarah:

1) Sebagaimana graf, TM terdiri dari beberapa *node* dan beberapa *edge*. Dari satu node mungkin terdapat satu atau lebih edge yang menuju node lainnya atau dirinya sendiri.
2) Sebuah node menyatakan sebuah *stata* (*state*). Dua stata penting adalah stata awal S (*start*) dan stata penerima H (*halt*). Sesaat sebelum proses pengenalan sebuah kalimat, TM berada pada stata S. Jika kalimat tersebut dikenali maka, setelah selesai membaca kalimat tersebut, TM akan akan berhenti pada stata H.
3) Sebuah edge mempunyai 'bobot' yang dinotasikan sebagai triple: (a, b, d). a adalah karakter acuan bagi karakter dalam sel pita TM yang sedang dibaca head. Jika yang dibaca head adalah karakter a maka a akan di-*overwrite* dengan karakter b dan head akan berpindah satu sel ke arah d (kanan atau kiri).
4) Kondisi *crash* akan terjadi jika ditemui keadaan sebagai berikut:



TM sedang berada pada stata i. Jika TM sedang membaca simbol ax ≠ a1 ≠ a2 ≠ … ≠ an maka TM tidak mungkin beranjak dari stata i. Jadi pada kasus ini penelusuran (*tracing*) TM ber-akhir pada stata i.

<u>Contoh</u>:

Rancanglah sebuah mesin turing pengenal bahasa L = $\{a^n b^n \mid n \geq 0)$.

<u>Jawab</u>:

L tersebut terdiri dari 2 kelompok kalimat yaitu ε dan non-ε. Kelompok non-ε adalah: ab, aabb, aaabbb, dan seterusnya. Untuk dapat menerima kalimat ε TM harus mempunyai edge dari S ke H dengan bobot (ε ,ε , R). TM menerima kalimat-kalimat: ab, aabb, aaabbb, dan seterusnya, dengan algoritma sebagai berikut:

1. Mulai dari S, head membaca simbol a.
2. Head membaca simbol a. Tandai simbol a yang sudah dibaca tersebut, head bergerak ke kanan mencari simbol b pasangannya.
3. Head membaca simbol b. Tandai simbol b yang sudah dibaca tersebut, head bergerak ke kiri mencari simbol a baru yang belum dibaca/ditandai.
4. Ulangi langkah 2 dan 3.
5. Head sampai ke H hanya jika semua simbol a dan simbol b dalam kalimat $a^n b^n$ selesai dibaca.

Algoritma di atas lebih diperinci lagi sebagai berikut:

1. Mulai dari S, head membaca simbol a.
2. *Overwrite* a tersebut dengan suatu simbol (misalkan A) untuk menandakan bahwa a tersebut sudah dibaca.

   Selanjutnya head harus bergerak ke *kanan* untuk mencari sebuah b sebagai pasangan a yang sudah dibaca tersebut.

   i) Jika yang ditemukan adalah simbol a maka a tersebut harus dilewati (*tidak boleh dioverwrite*), dengan kata lain a *dioverwrite dengan* a *juga* dan head bergerak ke *kanan*.

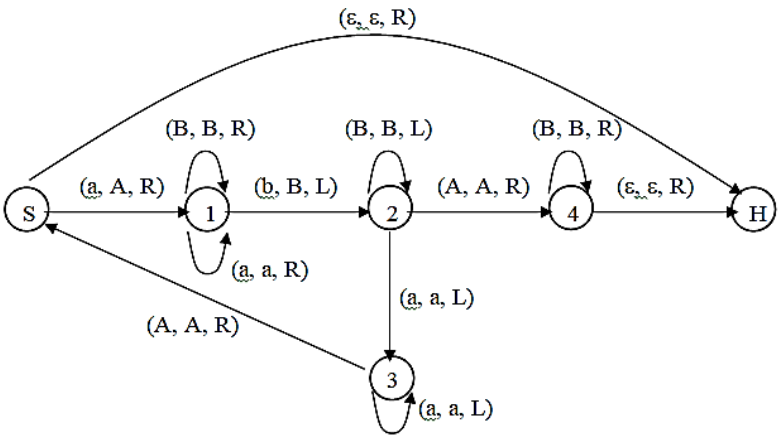
   ii) Jika TM pernah membaca simbol b ada kemungkinan ditemukan simbol B. Simbol B tersebut harus dilewati (*tidak boleh dioverwrite*), artinya B *diover-write dengan* B *juga* dan head bergerak ke *kanan*.
3. Head membaca simbol b, maka b tersebut harus dioverwrite dengan simbol lain (misalnya B) untuk menandakan bahwa b tersebut (sebagai pasangan dari a) telah dibaca, dan head bergerak ke *kiri* untuk mencari simbol A.

   i) Jika ditemukan B maka B tersebut harus dilewati (*tidak boleh dioverwrite*), dengan kata lain B *dioverwrite dengan* B *juga* dan head bergerak ke *kiri*.

   ii) Jika ditemukan a maka a tersebut harus dilewati (*tidak boleh dioverwrite*), dengan kata lain a *dioverwrite dengan* a *juga* dan head bergerak ke *kiri*.
4. Head membaca simbol A, maka A tersebut harus dilewati (*tidak boleh dioverwrite*), dengan kata lain A *dioverwrite dengan* A *juga* dan head bergerak ke *kanan*.
5. Head membaca simbol a, ulangi langkah 2 dan 3.
6. (Setelah langkah 3) head membaca simbol A, maka A tersebut harus dilewati (*tidak boleh dioverwrite*), dengan kata lain A *dioverwrite dengan* A *juga* dan head bergerak ke *kanan*.
7. Head membaca simbol B, maka B tersebut harus dilewati (*tidak boleh dioverwrite*), dengan kata lain B *dioverwrite dengan* A *juga* dan head bergerak ke *kanan*.

8. Head membaca simbol ε, maka ε di*overwrite* dengan ε dan head bergerak ke *kanan* menuju stata H.

Skema graf Mesin Turing di atas adalah:



(ε, ε, R)

Contoh:

Lakukan *tracing* dengan mesin *turing* di atas untuk kalimat-kalimat: aabb, aab.

Jawab:

i) (S,aabb)⇒ (1,Aabb) ⇒ (1,Aabb) ⇒ (2,AaBb) ⇒ (3,AaBb) ⇒ (S,AaBb)⇒ (1,AABb) ⇒ (1,AABb) ⇒ (2,AABB) ⇒ (2,AABB) ⇒ (4,AABB)⇒ (4,AABB) ⇒ (4,AABBε) ⇒ (H,AABBεε)

ii) (S,aab) ⇒ (1,Aab) ⇒ (1,Aab) ⇒ (2,AaB) ⇒ (3,AaB) ⇒ (S,AaB) ⇒ (1,AAB) ⇒ (1,AAbε) ⇒ crash, karena dari node 1 tidak ada edge dengan bobot komponen pertamanya hampa (ε)

Contoh

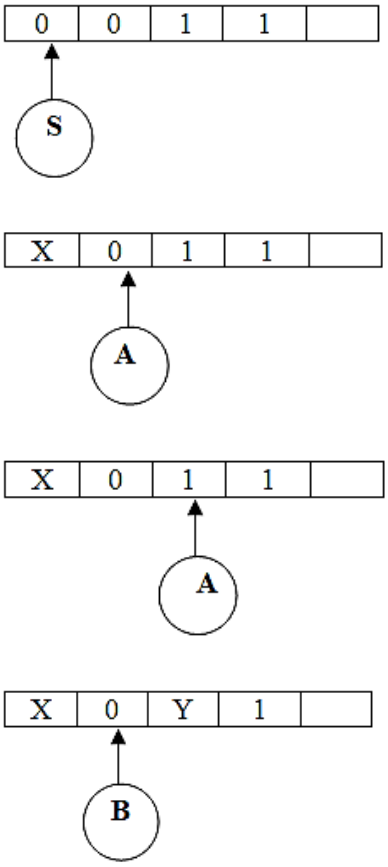
Misal terdapat mesin Turing:

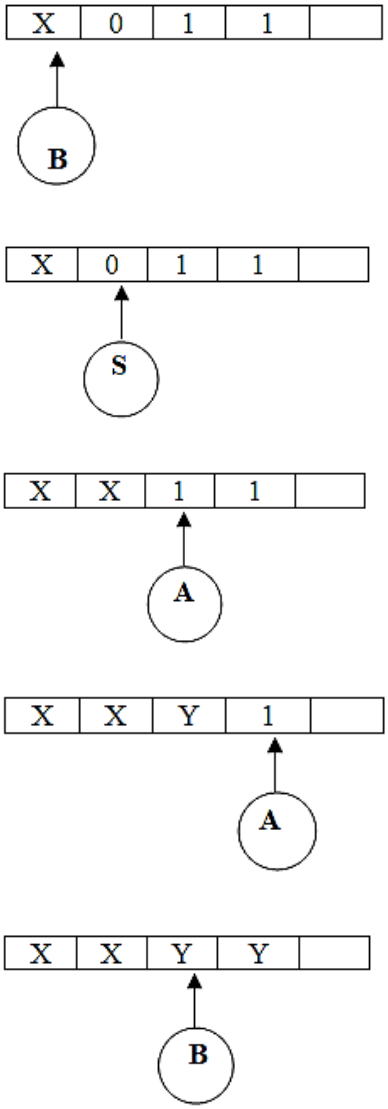
Q = {S, A, B, C, D} ∑ = {0, 1}

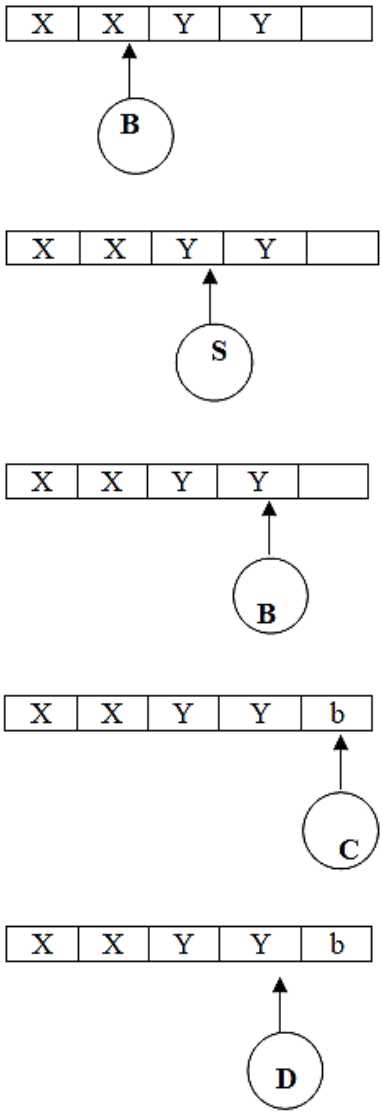
Γ= {0, 1, X, Y, b} F = {D} S = {S}

δ	0	1	X	Y	b
S	(A, X, R)	-	-	(C, Y, R)	-
A	(A, 0, R)	(B, Y, L)	-	(A, Y, R)	-
B	(B, 0, L)	-	S, X, R)	(B, Y, L)	-
C	-	-	-	(C, Y, R)	(D, b, L)
D	-	-	-		

Pita yang akan dibaca ‘0011’
Operasi Mesin Turing



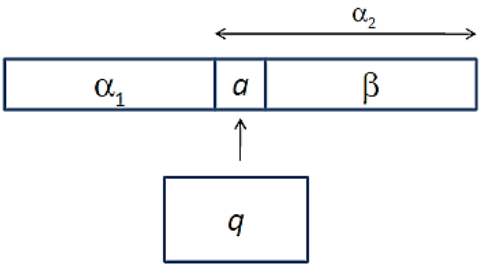

X 0 1 1
B
X 0 1 1
S
X X 1 1
A
X X Y 1
A
X X Y Y
B



Karena tidak ada transisi lagi dari q4 , maka Mesin Turing berhenti (*halt*) dan karena q4 adalah state akhir, maka input diterima.

14.4. Deskripsi Seketika Mesin Turing

- Keadaan sebuah Mesin Turing setiap saat dicirikan oleh tiga hal:
  1. Status sekarang (q)
  2. Simbol yang sedang diterima/dibaca
  3. Posisi head ("nomor sel" yang sedang dibaca) pada pita.



- Jika α2 = aβ, maka konfigurasi sesaat mesin Turing pada gambar di atas dapat dinyatakan secara tekstual oleh deskripsi sesaat (*instantaneous description*):

  α1 q α2

  yang artinya:
  - Mesin sedang berada pada status q
  - a1α2 adalah string yang tertera pada pita
  - Mesin sedang membaca simbol paling kiri dari α2

  Contoh gerakan ke kiri oleh d(p, Xi) = (q, Y, L):

  $X_1X_2 \ldots X_{i-1}p\ X_i\ X_i+1 \ldots X_n$ $X_1X_2 \ldots qX_{i-1}\ Y\ X_{i+1} \ldots X_n$

  Contoh gerakan ke kanan oleh d(p, Xi) = (q, Y, R):

  X1X2 … Xi – 1p Xi Xi +1 … Xn ├ X1X2 …Xi – 1 Y q Xi + 1 … Xn

- Sebuah *string* (kalimat) diterima oleh mesin Turing M = (Q, ∑, Γ, δ, q0 , B, F) jika mesin tersebut mencapai status akhir.
  Dengan kata lain suatu kalimat w diterima oleh M jika terdapat rangkaian deskripsi sesaat:

  q0 w ├* α_1 p α_2

yang dalam hal ini p ϵ F dan α_1 α_2 ϵ Γ*

Contoh:

Q = {a, b, c, d, e},

Γ= {0, 1, X, Y, B},

∑ = {0, 1, B}

q0 = a,

F = {e},

dan fungsi transisi d dinyatakan oleh tabel berikut:

	0	**1**	**X**	**Y**	**B**
a	(b,X, R)			(d,Y, R)	
b	(b, 0, R)	(c, Y, L)		(b,Y, R)	
c	(c, 0, L)		(a, X, L)	(c, Y, R)	
d				(d,Y, R)	(e, B, R)

Maka komputasi string '0011' oleh mesin Turing M dinyatakan dalam rangkaian deskripsi sesaat berikut:

a0011 ├ Xb011 ├ X0b11_ Xc0Y1 ├ cX0Y1_
Xa0Y1 ├ XXbY1 ├ XXYa1 ├ XXcYY ├ XcXYY ├
XXaYY ├ XXYdY ├ XXYYd ├ XXYYBe (accepted!)

14.5. Latihan Soal

Latihan 1:

Tulis deksripsi seketika mesin Turing pada contoh 15.2 Jika diberi input 011.

Penyelesaian:

(q0, 011) ├ (q1, X11) ├ (q2, XY1) ├ (q0, XY1) ├ (q3, XY1)

Latihan 2:

Misal terdapat mesin Turing:

Q = {q0, q1, q2, q3, q4} ∑ = {a, b}

Γ= {a, b, X, Y, b} F = {q4} S = {q0}

Fungsi transisi

δ	a	b	X	Y	b
Q0	(q1, X, R)	-	-	(q3, Y,R)	-
Q1	(q1, a, R)	(q2, Y, L)	-	(q1, Y,R)	-
Q2	(q2, a, L)	-	(q0, X,R)	(q2,Y, L)	-
Q3	-	-	-	(q3, Y,R)	(q4, b, L)
Q4	-	-	-	-	-

Lakukan operasi Mesin Turing diatas, jika pita yang dibaca 'aaabbb'

Latihan 3:

Terdapat mesin turing dengan konfigurasi

Q = { q1, q2 }

Σ = { a, b }

Γ = { a, b, *b* }

F = { q2 } R = Right

S = { q1 } L = Left

Fungsi transisi

δ (q1, a) = (q1, a, R)

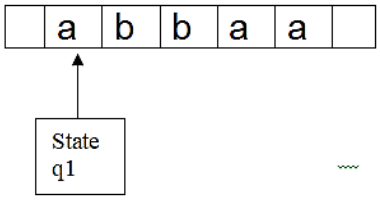
δ (q1, b) = (q1, a, R)

δ (q1, b) = (q2, b ,L)

- δ (q1, a) = (q1, a, R)
  State q , head menunjuk karakter 'a' pada pita menjadi q1, state q1, head bergerak ke kanan.
- δ (q1, b) = (q1, a, R)
  State q1, head menunjuk karakter 'b' pada pita menjadi state q1, head menulis karakter 'a' lalu bergerak ke kanan
- δ (q1, *b*) = (q2, *b* , L)
  Pada State q1, head menunjuk karakter ' *b* ' pada pita menjadi state q2, head bergerak ke kiri.

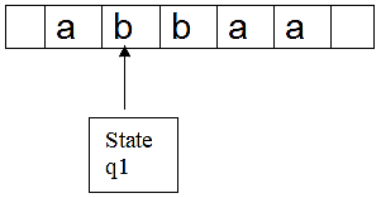
Jika mesin turing tersebut menerima input 'abbaa', apakah diterima oleh mesin turing atau ditolak

1. Pita yang dibaca 'abbaa'



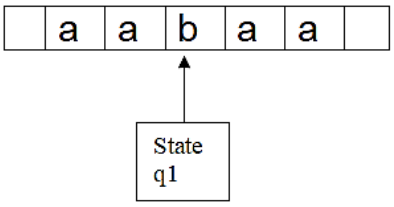
Fungsi transisi δ(q1) = (q1, a, R)

2.



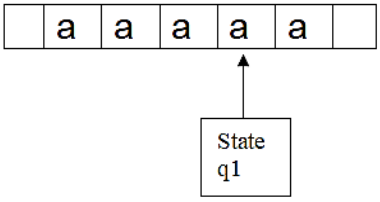
Fungsi transisi δ (q1, b) = (q1, a, R)
Head menulis ‘a’ lalu bergerak ke kanan

3.



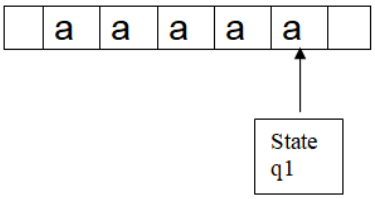
Fungsi transisi δ (q1, b) = (q1, a, R)
Head menulis ‘a’ lalu bergerak ke kanan

4.



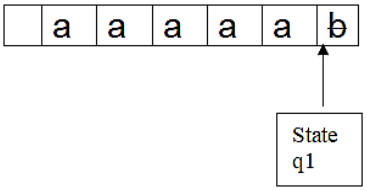
Fungsi transisi δ (q1, a) = (q1, a, R)
Head bergerak ke kanan

5.



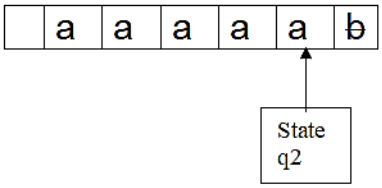
Fungsi transisi δ (q1, a) = (q1, a, R)
Head bergerak ke kanan

6.



Fungsi transisi δ (q1, b) = (q2, ƀ, L)
Head menunjuk ‘ b ’ lalu head bergerak ke kiri

7.



State berhenti pada q2, q2 adalah final state, maka state input ‘ abbaa’ diterima.

DAFTAR PUSTAKA

Brookshear, Glen J., “Theory of Computation: Formal Language, Automata and Complexity”, The Benjamin/Cummings Publishing Company, 1989

Heriyanto, Bambang, Teori Bahasa, Otomata dan Komputasi serta terapannya, Informatika Bandung,2003

Hopcroft, Motwani, Ullman. Introduction to Automata Theory, Languages, and Computation. Addison Wesley, 2007

Hopcroft, Motwani, Ullman: *Introduction to Automata Theory , Languages, and Computation* . Addison-Wesley, 2001

James A. Anderson, Automata Theory with Modern Applications, Cambridge University Press, 2006

John E. Hopcroft dkk. Teori Bahasa dan Otomata, (terjemahan, Edisi 2, 2007

Linz Peter, “An Introduction to Formal Languages and Automata 5th Edition, Jones & Bartlett Publishers”, ISBN: 144961552X, 9781449615529, 2011

Martin, J.C.: Introduction to Languages and the Theory of Computation.McGraw– Hill (2002)

Munir, Rinaldi, Diktat Kuliah Matematika Informatika(Teori bahasa Formal dan Otomata), ITB

Revesz, Gyorgy E., “Introduction to Formal Languages”, McGraw Hill Book Company, 1985

S. Greibach. Formal languages: origins and directions. Ann. Hist. Comput., 3, 1981

Salomaa, A.: Jewels of Formal Language Theory. Computer Science Press (1981)

Sipser, M, “Introduction to the Theory of Computation”, Cengage Learning, 2012

Utdirartatmo, Firrar, “Teknik Kompilasi”, J&J Learning Yogyakarta, ISBN: 979-9398-11-8, 2001

Utdirartatmo, Firrar, Teori Bahasa dan Otomata, Penerbit Graha Ilmu, Jogyakarta, 2005

INDEKS

A

Abjad, 5, 6, 14, 15, 35, 59
Abjad, 2
Accepting state, 36, 37
Alfabet, 2, 3, 4, 5, 6, 7, 8, 10, 11, 13, 20, 21, 22, 23, 34, 59, 60, 106, 187, 203, 206, 207, 219
Algoritma, xiii, 50, 59, 61, 96, 98, 101, 105, 116, 117, 175, 225, 227, 228
Alternation, 12, 13
Ambiguitas, 81, 131
Analisa, xiv, 38, 111, 141, 142, 150, 151
Aturan produksi, xviii, xix, 19, 22, 24, 25, 27, 77, 78, 79, 80, 81, 82, 83, 84, 87, 93, 94, 95, 96, 103, 104, 110, 113, 114, 115, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 132, 133, 134, 136, 137, 139, 140, 141, 142, 143, 144, 145, 149, 150, 152, 153, 154, 157, 159, 160, 161, 162, 163, 164, 168, 169, 171, 172, 173, 174, 175, 179, 180, 181, 211, 213
Aturan produksi, 24, 104, 105, 122, 149, 150
Automata, xiii, xiv, xvi, xvii, 9, 25, 31, 32, 34, 35, 36, 37, 41, 42, 45, 182, 183, 192, 197, 203, 238
Automata, 49, 65, 104, 110, 188, 189, 197, 203

B

Bahasa, v, xiii, xiv, xv, xvi, xvii, xviii, xix, 1, 2, 6, 7, 8, 9, 10, 13, 14, 15, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 30, 31, 32, 34, 35, 44, 47, 49, 59, 64, 77, 78, 79, 81, 82, 83, 84, 85, 86, 87, 89, 90, 91, 92, 93, 96, 97, 101, 103, 104, 105, 106, 107, 108, 109, 111, 115, 119, 122, 125, 136, 156, 159, 170, 171, 173, 174, 175, 176, 177, 178, 179, 182, 183, 184, 185, 186, 187, 188, 194, 195, 196, 197, 203, 211, 213, 220, 221, 224, 227, 238
Bahasa formal, v, xiii, xv, xvi, xvii, xviii, 1, 2, 6, 7, 8, 10, 19, 22

C

Chomsky, 34, 87, 92, 93, 94, 95, 96, 103, 104, 105, 106, 108,

109, 110, 113, 114, 116, 120, 121, 160, 161
Closure, 11, 12, 15, 55, 56, 106, 183, 184
Concatenation, 4, 5, 9, 12, 13, 106
Context free gammar (cfg), 35
Context free language, 25
Context sensitive grammar, 110
Context sensitive language, 25

D

Deterministic finite automata, 49, 104, 188, 190
DFA, xix, 41, 42, 43, 44, 45, 48, 49, 52, 53, 59, 60, 61, 170, 174, 175, 176, 179, 184, 190, 192
Diagram, xx, 31, 43, 50, 55, 57, 60, 61, 194
Diskrit, 1, 33, 34, 35

E

Eksponensiasil, 14
Ekspresi reguler, 9, 170, 176, 183, 184, 185, 186, 187, 194, 195, 196
Ekuivalensi, xviii, 49, 71
Equal, 15

F

Finite state, xiv, 25, 33, 34, 35, 104, 110, 182, 183, 188, 192
Formal language, xiii, 1, 2
FSA, 34, 35, 41, 47, 54, 64, 65, 69, 71, 74, 75, 104, 110, 174, 175, 177, 183, 188, 192, 193, 194
Fungsi transisi, 35, 38, 44, 46, 60, 64, 67, 69, 74, 198, 200, 201, 207, 208, 209, 210, 216, 219, 222, 234

G

Gabungan, 4, 9, 14, 17, 49, 82, 184
Grammar, xiii, 1, 2, 18, 19, 20, 22, 23, 25, 35, 77, 81, 86, 87, 103, 105, 106, 108, 109, 110, 111, 112, 221, 224

H

Head, 33, 34, 218, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 233, 235, 237

I

Input, xiv, 30, 31, 32, 33, 34, 35, 36, 37, 38, 41, 46, 49, 50, 51, 54, 55, 56, 59, 61, 64, 65, 66, 67, 68, 69, 70, 73, 76, 182, 188, 192, 193, 194, 198, 199, 200, 201, 202, 203, 204, 206, 207, 208, 210, 212, 213, 214, 215, 216, 218, 219, 220, 222, 224, 232, 234, 235, 237
Irisan, 14

K

Kleen star, 5

Kompilator, xiii, xiv, 2, 183
Komputasi, v, xiii, 1, 234
Kriptografi, 1

L

Leftmost, 79, 80, 91, 131

M

Mesin mealy, 64, 67, 68, 69, 71, 72, 73, 75, 76
Mesin moore, 64, 65, 66, 69, 70, 71, 72
Mesin turing, xv, 33, 35, 104, 218, 219, 221, 229, 233, 235
Move nfa ε, 54

N

Ndfa, 45, 59, 60, 61, 104, 178, 179
Normal greibach, 125, 159, 160, 161, 162, 163, 164, 166, 168

O

Operasi, xviii, xix, 4, 9, 10, 11, 13, 15, 16, 106, 160, 161, 184, 187, 198, 199, 230, 235
Otomata, xiv, xv, xvii, xviii, xix, 1, 18, 30, 31, 32, 33, 34, 35, 37, 64, 171, 172, 179, 180, 181, 182, 183, 192, 199
Output, xiv, xviii, 30, 32, 33, 35, 63, 64, 65, 66, 67, 68, 69, 71, 72, 73, 74, 105

P

Perangkaian, 13
Pohon urai, 80
Postfix, 11, 13
Prefik, 11
Produksi unit, xviii, 92, 136, 139, 143, 148, 149, 152
Produksi *useless*, 92, 139, 148, 149
Produksi ε, 92, 125, 128, 134, 139, 145, 147, 148, 149, 156
Properprefix, 11, 13
Propersubsequence, 12
Propersubstring, 12
Pumping lemma, xviii, 82, 176, 177
Push down, 25, 34, 104, 197, 198, 203
Pushdown automata, 35

R

Regular grammar, 111
Regular language, 25
Rekursif, xviii, 122, 123, 124, 125, 126, 127, 128, 129, 130, 132, 133, 134, 160, 161, 162, 163, 167
Rekursif, 122
Rightmost derivation, 79, 131

S

Simbol, 1, 2, 3, 4, 5, 8, 10, 11, 12, 19, 20, 21, 22, 23, 25, 27, 28, 31, 33, 34, 35, 41, 42, 43, 45, 64, 67, 68, 69, 73, 77, 79, 82, 83, 85, 89, 93, 94, 96, 97, 100, 103, 104, 106, 110, 111, 117, 120, 122, 125, 126, 127, 128,

132, 133, 134, 136, 137, 139, 140, 141, 143, 147, 149, 150, 151, 156, 159, 160, 161, 162, 163, 164, 165, 167, 170, 176, 184, 185, 186, 195, 197, 198, 199, 203, 204, 207, 211, 212, 215, 218, 219, 221, 222, 225, 227, 228, 229, 233

State, xiv, 30, 31, 32, 35, 36, 37, 38, 40, 41, 42, 43, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 58, 59, 61, 63, 64, 65, 66, 67, 68, 69, 71, 72, 73, 74, 170, 172, 175, 176, 180, 182, 183, 188, 192, 193, 194, 198, 199, 200, 201, 202, 203, 204, 206, 207, 208, 209, 210, 212, 213, 214, 215, 216, 218, 219, 221, 222, 223, 224, 226, 232, 235, 237

String, xviii, xix, 2, 3, 4, 5, 10, 11, 12, 13, 19, 20, 21, 22, 23, 24, 25, 27, 31, 32, 33, 34, 37, 38, 39, 41, 43, 46, 60, 63, 77, 79, 80, 81, 82, 90, 96, 110, 111, 112, 135, 136, 137, 175, 176, 177, 182, 183, 184, 185, 186, 187, 194, 195, 197, 201, 204, 206, 207, 208, 209, 210, 212, 213, 215, 216, 218, 219, 220, 221, 225, 233, 234

Subsequence, 12, 13

Sufix, 11

Switching, v, xiii, xiv, 1

T

Tabel transisi, 38, 52, 53, 55, 208, 210

Tail, 12, 13

Tape, 33, 34, 218, 221, 222

Tata bahasa, v, xiii, xv, xviii, xix, 1, 2, 9, 19, 21, 22, 25, 27, 77, 78, 84, 87, 90, 91, 92, 93, 94, 95, 96, 97, 100, 101, 103, 104, 105, 106, 107, 108, 109, 110, 113, 114, 115, 116, 119, 122, 124, 125, 126, 127, 132, 133, 134, 135, 136, 137, 143, 145, 146, 147, 148, 152, 154, 155, 156, 158, 159, 160, 161, 164, 170, 171, 172, 178, 179, 211, 212, 213

Tatabahasa linier, 173, 174, 175

Tatabahasa reguler, 170

Transisi, 35, 37, 38, 41, 43, 44, 46, 49, 50, 52, 53, 54, 55, 57, 58, 60, 64, 67, 69, 72, 73, 74, 172, 174, 175, 180, 188, 194, 198, 199, 200, 201, 203, 207, 208, 209, 210, 212, 215, 216, 217, 219, 220, 222, 223, 224, 232, 235, 236, 237

U

Unrestricted grammar, xx, 109, 110

Unrestricted language, 25

Untai, 3, 4, 5, 6, 8, 10, 14, 15, 68, 69, 97, 100, 101, 103, 116,

131, 132, 135, 170, 177, 182, 185, 213, 215

V

Variabel, 2, 19, 79, 82, 84, 92, 93, 94, 98, 99, 100, 101, 102, 103, 104, 119, 120, 122, 123, 125, 126, 127, 133, 137, 139, 140, 141, 143, 145, 146, 147, 149, 150, 151, 152, 154, 155, 158, 159, 160, 161, 162, 163, 164, 168, 170, 212

TENTANG PENULIS

Ahmat Adil, S.Kom., M.Sc. dilahirkan di Lunti, Manggarai (Nusa Tenggara Timur), 1 Desember 1972. Penulis menyelesaikan pendidikan tinggi di Jurusan Teknik Informatika, Institut Sains dan Teknologi Palapa (ISTP) Malang tahun 1995, dan menyelesaikan pendidikan master di Program Studi Teknologi Informasi Fakultas Matematika dan Ilmu Pengetahuan Alam (MIPA), Institut Pertanian Bogor (IPB) tahun 2006. Mulai bekerja sebagai dosen tetap di STMIK Bumigora Mataram sejak tahun 1996, mengajar mata kuliah Statistika, Matematika Diskrit, Pemrograman Web, dan saat ini lebih fokus mengajar mata kuliah Teori Bahasa & Otomata dan Sistem Informasi Geografis, di semester ganjil serta Analisis dan Perancangan Perangkat Lunak di semester genap.

Sampai saat ini penulis telah menyelesaikan beberapa riset di bidang Sistem Informasi Geografis yang dibiayai oleh DP2M Dikti seperti *Model Pemetaan Industri Kecil dan Menengah Berbasis Sistem Informasi Geografis* (2009), *Sistem Informasi Geografis Pemetaan Populasi Penduduk dan Komodo di Taman Nasional Komodo* (2012-2013), *Pemodelan Perkembangan Pemukiman di Kawasan Taman Nasional Komodo Berbasis Sistem Informasi Geografis* (2014-2015), Analisa Spasial Pemetaan lokasi wisata Agro (Studi Kasus di Lombok Barat) tahun 2016. Selain itu juga pengabdian kepada masyarakat dengan judul *Biometric Security: Model Control Pelayanan Kesehatan Gratis untuk Masyarakat Miskin di Kota Mataram* (2013) dan *IBM: Produk Kerajinan di Taman Nasional Komodo* (2016). Selain melakukan pengajaran, peneliti an, dan pengabdian kepada masyarakat, penulis juga mempublikasikan arti kel di jurnal ilmiah, seperti *Analisa Spasial Menentukan Lokasi Pemukiman di Pulau Komodo, Analisis Spasial Pemetaan Habitat Biawak Komodo di Pulau Rinca, Analisa Spasial Pemetaan lokasi wisata Agro*

(Studi Kasus di Lombok Barat), Studi Kasus Model Strategis Pemasaran Terintergrasi Berbasis Teknologi Informasi di Nusa Tenggara Barat, Implementasi Google Maps Api Berbasis Android Untuk Lokasi Fasilitas Umum Di Kabupaten Sumbawa.

Penulis juga pernah menjadi pemakalah di seminar nasional, Aplikasi SIG dalam Pemetaan Pemukiman di Taman Nasional Komodo, seminar nasional "Teknologi Informasi dan Aplikasinya (SENTIA), Analisa Spasial Sebaran Pemukiman di Pulau Renca (Taman Nasional Komodo), Seminar nasional "Teknologi Informasi dan Aplikasinya (SENTIA), Politeknik Negeri Malang, Perancangan Sistem Informasi Pemasaran Produk Kerajinan di Pulau Komodo, Seminar Nasional Aptikom 2016 "Pengembangan Konten Digital Warisan Budaya Alam untuk Mendukung *e-tourism*, Implementasi *Rest* api *web service* Dalam Membangun Aplikasi Multiplatform, SocioTECH.

Buku ini adalah buku kedua yang penulis buat, Pada tahun 2017 telah terbit buku dengan judul **Sistem Informasi Geografis** yang diterbitkan oleh penerbit Andi Ofset Yogyakarta, (ISSBN 978-979-29-6331-1).