RESUME

TEORI BAHASA DAN AUTOMATA

Disusun Oleh :

Hayatun Nupus

(21346011)

Lecturer:

Widya Darwin S.Pd,.M.Pd.T

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS NEGERI PADANG

TAHUN 2023

DAFTAR ISI

A. Pengantar Teori bahasa & Otomata

1. Konsep Teori Bahasa & Otomata

- **Tingkatan Bahasa Pemrograman**

a. Bahasa Mesin

Bahasa mesin adalah bahasa yang dapat dipahami langsung oleh komputer. Instruksi-instruksi dalam bahasa mesin direpresentasikan dalam bentuk bilangan biner (0 dan 1).

b. Bahasa Assembly

Bahasa assembly merupakan bahasa pemrograman rendah tingkat yang menggunakan kode simbolik untuk mewakili instruksi-instruksi dalam bahasa mesin. Setiap instruksi dalam bahasa assembly setara dengan instruksi dalam bahasa mesin.

c. Bahasa Tingkat Rendah (Low-level)

Bahasa tingkat rendah adalah bahasa pemrograman yang memiliki sedikit atau tanpa abstraksi dari perangkat keras komputer. Contoh bahasa tingkat rendah adalah C dan C++, yang memberikan kontrol langsung terhadap memori dan perangkat keras komputer.

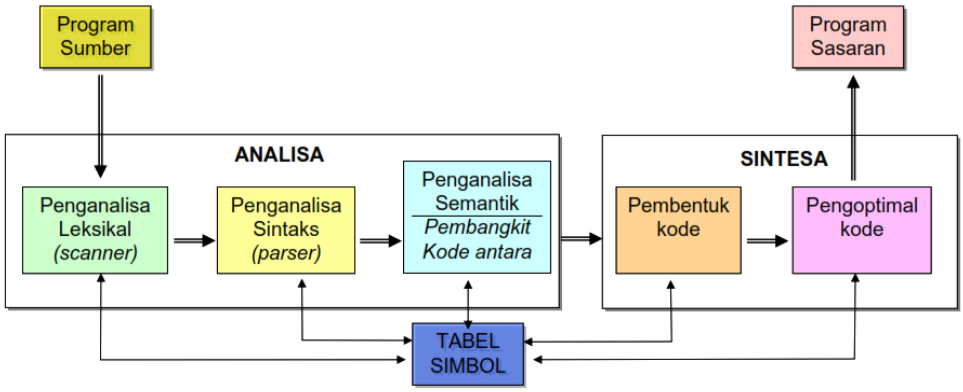
d. Bahasa Tingkat Tinggi (High-level)

Bahasa tingkat tinggi adalah bahasa pemrograman yang dirancang dengan tingkat abstraksi yang lebih tinggi. Bahasa ini menggunakan konstruksi sintaksis dan fitur-fitur yang lebih mudah dipahami oleh manusia. Contoh bahasa tingkat tinggi termasuk Python, Java, C#, dan JavaScript.

2. Cara supaya sebuah mesin dapat memahami kode program yang dibuat oleh programmer:

a Pemilihan bahasa pemrograman
b Penulisan kode program
c Kompilasi atau interpretasi
d Eksekusi program

3. Bagan Pokok Proses Kompilasi



Keterangan:

1. **Sumber Program** yaitu kode program yang ditulis oleh seorang programmer.

   ANALISA:
2. **Penganalisa Leksikal (scanner):**
   - Menerima kode program sebagai input.
   - Menerjemahkan urutan karakter menjadi token-token (unit-unit terkecil yang memiliki makna dalam bahasa pemrograman).
   - Menyimpan token-token tersebut untuk digunakan dalam langkah-langkah berikutnya.
3. **Penganalisa Sintaks (parser):**
   - Menggunakan token-token yang dihasilkan pada langkah sebelumnya.
   - Membentuk struktur sintaksis program dengan menerapkan aturan-aturan sintaksis bahasa pemrograman.
   - Mengecek kebenaran sintaksis program dan mendeteksi adanya kesalahan sintaksis.
4. **Penganalisa Semantik/ Pembangkit Kode antara:**
   - Mengevaluasi makna dari struktur sintaksis program.
   - Memeriksa konsistensi tipe data dan penggunaan variabel.
   - Mendeteksi kesalahan semantik yang mungkin terjadi.
5. **TABEL SIMBOL** yaitu struktur data yang digunakan oleh kompiler untuk menyimpan informasi tentang identifier (nama, variabel, fungsi, dll).
6. **Program sasaran** yaitu hasil akhir dari proses kompilasi.

   SINTESA:
7. **Pembentuk Kode**
   - Menerjemahkan representasi internal program menjadi instruksi-instruksi dalam bahasa mesin target.
   - Menyesuaikan instruksi-instruksi tersebut dengan arsitektur dan perangkat keras komputer yang digunakan.
8. **Pengoptimal Kode**
   - Mengoptimalkan instruksi-instruksi dalam bahasa mesin untuk meningkatkan performa atau efisiensi program.
   - Menggunakan teknik-teknik seperti penggabungan instruksi, pengurangan beban memori, dan optimisasi register.

4. Pengertian Teori Bahasa dan Otomata

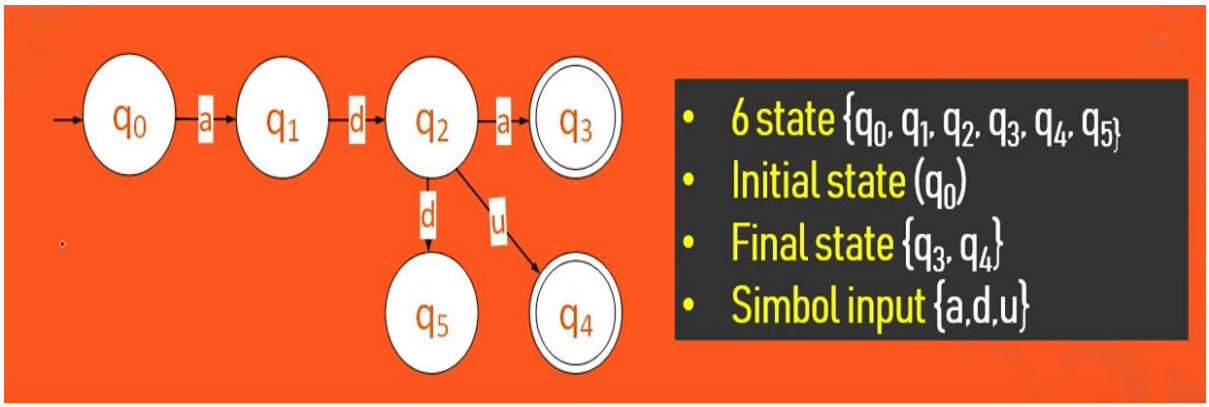
Teori Bahasa dan Otomata adalah bidang studi dalam ilmu komputer yang mempelajari bahasa formal dan mesin yang dapat memproses dan mengenali bahasa tersebut. Teori Bahasa dan Otomata mencakup konsep dasar yang berkaitan dengan bahasa formal, otomata, dan komputabilitas.

Bahasa Formal: Bahasa formal adalah himpunan string-string yang dibentuk dari alfabet tertentu dengan aturan-aturan tertentu. Bahasa formal

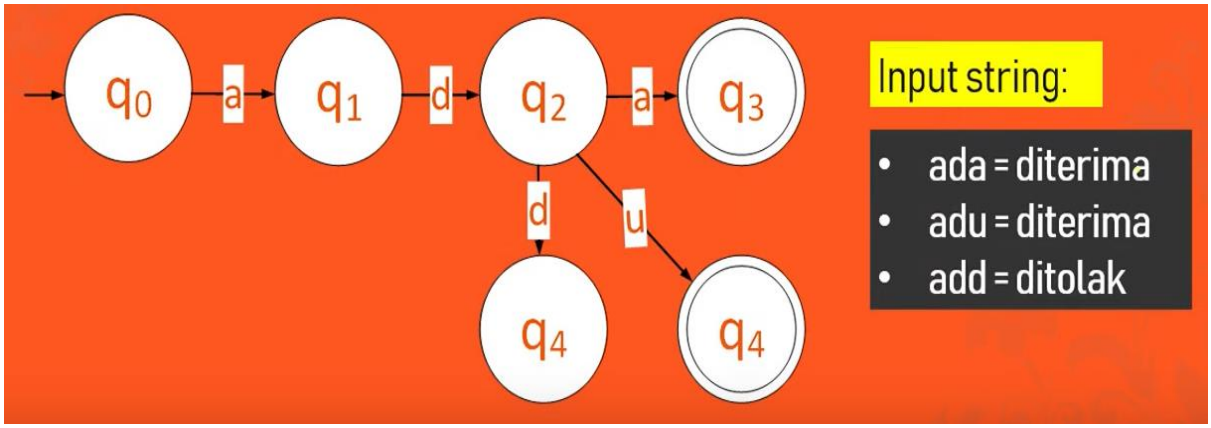
digunakan untuk merepresentasikan dan memodelkan bahasa-bahasa manusia, serta digunakan sebagai dasar dalam pengembangan bahasa pemrograman.

Otomata: Otomata adalah mesin abstrak yang dapat mengenali dan memproses bahasa formal. Otomata dapat berupa model matematika yang terdiri dari keadaan (state) dan transisi (transition) antara keadaan-keadaan tersebut. Otomata digunakan dalam pengenalan bahasa, pemrosesan string, dan analisis sintaksis.

5. Mesin otomata sederhana



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



B. Simbol, String, dan Bahasa

1. Pengertian Simbol

Simbol adalah unit dasar atau elemen terkecil yang digunakan dalam teori otomata. Simbol dapat berupa karakter tunggal, angka, atau simbol-simbol lain yang didefinisikan dalam konteks spesifik.

2. Pengertian String

String adalah urutan terbatas dari simbol-simbol yang diambil dari suatu alfabet. String dapat terdiri dari satu atau lebih simbol. Misalnya, jika alfabetnya adalah {0, 1}, maka contoh string yang mungkin adalah "010101" atau "1100". String juga dapat kosong (ε), yang berarti tidak mengandung simbol apa pun.

3. Pengertian Bahasa

Bahasa adalah himpunan string-string yang terdiri dari simbol-simbol yang diberikan. Bahasa formal adalah himpunan string yang dibentuk dari alfabet tertentu dengan aturan-aturan tertentu.

- Input pada mesin automata dianggap sebagai bahasa yang harus dikenali oleh mesin.
- Mesin akan mengindikasikan apakah suatu bahasa dapat diterima atau tidak.

C. Tata Bahasa Hirarki Chomsky dan Aturan

1. Simbol Terminal dan Non-Terminal

a Simbol Terminal

Simbol terminal (juga dikenal sebagai simbol terminal atau terminal) adalah simbol-simbol yang merupakan unit terkecil yang tidak dapat didefinisikan lebih lanjut dalam konteks produksi atau aturan bahasa.

- Huruf kecil apfabet, misalnya: **a, b, c**
- Simbol operator, misalnya: **+, -, dan ‘**
- Simbol tanda baca, misalnya: **(,), dan :**
- String yang bercetak tebal, misalnya: **if, then, dan else**

b Simbol Non-Terminal

Simbol non-terminal (juga dikenal sebagai simbol non-terminal atau non-terminal) adalah simbol-simbol yang dapat didefinisikan lebih lanjut dalam konteks produksi atau aturan bahasa. Simbol non-terminal digunakan untuk merepresentasikan konstruksi gramatikal yang lebih kompleks atau abstrak dalam bahasa.

- Huruf besar alfabet, misalnya: A, B, C
- Huruf S sebagai simbol awal
- String yang tercetak miring, misalnya: expr dan stmt

2. Aturan Produksi

Aturan produksi dalam teori bahasa dan otomata dinyatakan dalam bentuk tata bahasa atau bahasa formal. Bentuk yang umum digunakan dalam tata bahasa

adalah tata bahasa bebas konteks (context-free grammar) dan tata bahasa berkonteks (context-sensitive grammar)

Contoh aturan produksi

A -> α

Dalam contoh di atas, A adalah simbol non-terminal yang memproduksi string α. Simbol non-terminal A pada sisi kiri aturan produksi dapat digantikan dengan string α pada sisi kanan aturan produksi. Simbol non-terminal dan simbol terminal (simbol yang tidak dapat digantikan lebih lanjut) digunakan dalam aturan produksi untuk memodelkan struktur dan sintaksis dari bahasa.

Contoh aturan produksi tata bahasa bebas konteks untuk bahasa aritmatika sederhana:

1. **E -> E + T**
2. **E -> T**
3. **T -> T * F**
4. **T -> F**
5. **F -> (E)**
6. **F -> id**

Aturan produksi di atas menjelaskan bagaimana ekspresi aritmatika dalam bahasa tersebut dapat dibentuk. Simbol non-terminal E, T, dan F merepresentasikan ekspresi, term, dan faktor secara berurutan. Simbol terminal +, *, (,), dan id (identifier) merupakan simbol-simbol terminal yang tidak dapat digantikan lebih lanjut. Aturan produksi tersebut menggambarkan struktur sintaksis yang sah dalam bahasa aritmatika tersebut.

3. Hirarki Chomsky

a Grammar

Grammar (tata bahasa) adalah sistem aturan yang digunakan untuk menggambarkan struktur sintaksis dari suatu bahasa. Grammar terdiri dari simbol-simbol dan aturan produksi yang menentukan bagaimana simbol-simbol tersebut dapat digabungkan untuk membentuk string-string yang sah dalam bahasa.

b 4 tingkatan tata Bahasa menurut Chemosky

- **Tata Bahasa Tipe-0 (Unrestricted Grammar)**

  Tata bahasa tipe-0, juga dikenal sebagai tata bahasa tidak terbatas, memiliki tingkat ekspresivitas paling tinggi. Tata bahasa ini tidak memiliki batasan pada aturan produksi dan dapat digunakan untuk menggambarkan bahasa-bahasa dengan struktur yang sangat kompleks.

- **Tata Bahasa Tipe-1 (Context-Sensitive Grammar)**

Tata bahasa tipe-1, juga dikenal sebagai tata bahasa konteks-sensitif, memiliki tingkat ekspresivitas yang lebih tinggi. Tata bahasa ini menggunakan aturan produksi yang mempertimbangkan konteks sekitarnya. Aturan produksi dalam tata bahasa ini dapat memperluas atau mempersempit string dalam bahasa tergantung pada konteksnya. Tata bahasa ini cocok untuk mengenali bahasa-bahasa dengan struktur hierarkis yang lebih kompleks.

- **Tata Bahasa Tipe-2 (Context-Free Grammar)**

  Tata bahasa tipe-2, juga dikenal sebagai tata bahasa bebas konteks, memiliki tingkat ekspresivitas yang sedikit lebih tinggi. Tata bahasa ini menggunakan aturan produksi dengan simbol non-terminal di sisi kiri dan simbol-simbol terminal dan/atau non-terminal di sisi kanan. Tata bahasa ini cocok untuk mengenali bahasa-bahasa dengan struktur hierarkis sederhana.

- **Tata Bahasa Tipe-3 (Regular Grammar)**

  Tata bahasa tipe-3, juga dikenal sebagai tata bahasa regulas, memiliki tingkat ekspresivitas paling rendah. Tata bahasa ini menggunakan aturan produksi sederhana yang disebut produksi terbatas (right-linear production) atau produksi terbalik (left-linear production). Tata bahasa ini cocok untuk mengenali bahasa-bahasa dengan struktur linear seperti bahasa regular dan ekspresi reguler.

D. Finite State Automata

1. Pengertian FSA

FSA merupakan singkatan dari Finite State Automaton atau Finite State Automata, yang dalam bahasa Indonesia dikenal juga sebagai Mesin Automata Berhingga. FSA adalah model komputasional yang digunakan dalam teori bahasa formal, teori otomata, dan ilmu komputer untuk menganalisis dan menggambarkan perilaku sistem berbasis keadaan yang terbatas.

2. Arti bentuk (Symbol) pada graph transisi FSA

- **Bulatan (State/Node)**
  Bulatan digunakan untuk mewakili keadaan atau state dalam FSA. Setiap bulatan mewakili satu keadaan atau state tertentu dalam mesin. State-state ini dapat berupa state awal (start state), state akhir (final state), atau state transisi.
- **Panah (Transition)**
  Panah digunakan untuk menghubungkan keadaan-keadaan dalam FSA. Panah menunjukkan transisi dari satu keadaan ke keadaan lainnya. Setiap panah memiliki label yang menunjukkan input yang memicu transisi tersebut. Misalnya, jika ada panah dengan label "a" yang menghubungkan state A

dengan state B, maka itu berarti saat menerima input "a", mesin akan berpindah dari state A ke state B.

- **Garis ganda atau lingkaran (Final State/Accept State)**
  Garis ganda atau lingkaran digunakan untuk menandai state akhir atau final state dalam FSA. Jika mesin berada pada state akhir setelah memproses urutan input, maka urutan input tersebut diterima atau diakui oleh FSA.
- **Lingkaran dengan panah masukan (Start State)**
  Lingkaran dengan panah masukan digunakan untuk menandai state awal atau start state dalam FSA. Ini menunjukkan keadaan awal mesin sebelum memproses urutan input.

3. Pernyataan FSA secara formal

FSA dinyatakan dalam 5 tupel:

$$A = (Q, \Sigma, \delta, q_0, F)$$

1. Himpunan keadaan (**Q**).
2. Himpunan simbol input (Σ)
3. Fungsi transisi (δ), memuat satu keadaan asal dan satu simbol input dan satu keadaan tujuan.
4. Keadaan awal (q_0)merupakan salah satu dari Q.
5. Himpunan **keadaan final** atau yang **diterima**, dinotasikan dengan **F ($F \subseteq Q$)**

4. Dua jenis kelompok FSA

a Deterministic Finite Automaton (DFA)

DFA adalah jenis FSA di mana setiap keadaan memiliki tepat satu transisi untuk setiap simbol input yang diterima. Artinya, dari suatu keadaan dan simbol input yang diberikan, hanya ada satu kemungkinan transisi yang dapat diambil. DFA dapat digambarkan sebagai graf berarah dengan keadaan awal, keadaan akhir, dan transisi yang ditentukan dengan jelas. DFA lebih sederhana dan lebih mudah dimengerti daripada jenis FSA lainnya.

b Non-deterministic Finite Automaton (NFA)

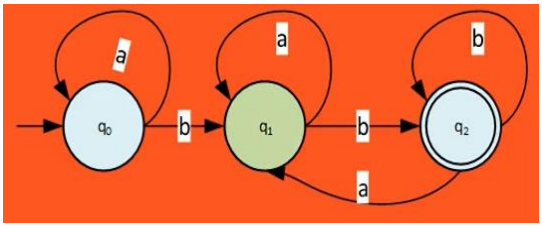
NFA adalah jenis FSA di mana keadaan memiliki lebih dari satu kemungkinan transisi untuk simbol input tertentu. Dalam NFA, transisi yang diambil dapat tidak jelas atau nondeterministik, artinya ada pilihan yang bisa diambil untuk simbol input yang sama dari keadaan yang sama. NFA dapat memiliki beberapa keadaan awal dan/atau beberapa keadaan akhir.

E. Deterministic Finite State Automata

1 Pengertian DFA

DFA (Deterministic Finite Automaton) atau Automata Berhingga Deterministik adalah jenis mesin automata atau model komputasional yang digunakan dalam teori bahasa formal, teori otomata, dan ilmu komputer. DFA terdiri dari sejumlah keadaan (states) yang terbatas dan transisi antara keadaan-keadaan tersebut berdasarkan simbol input yang diterima.

2 Contoh DFA



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

- δ
  - δ(q0,a)=q0
  - δ(q0,b)=q1
  - δ(q1,a)=q1
  - δ(q1,b)=q2
  - δ(q2,a)=q1
  - δ(q2,b)=q2

- Tabel transisi

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

3 String input

DFA (Deterministic Finite Automaton), suatu string input dinyatakan diterima jika DFA berakhir di salah satu keadaan akhir (final state) setelah memproses seluruh string input tersebut. Dengan kata lain, DFA mengenali atau menerima string input jika string input tersebut sesuai dengan aturan transisi yang telah ditentukan dan mengarahkan DFA ke salah satu keadaan akhir.

F. Non-Deterministic FA (NFA)

1. Pengertian NFA

NFA adalah model komputasi yang digunakan dalam teori bahasa formal dan teori otomata. Ini adalah jenis otomata yang terdiri dari sejumlah keadaan yang dapat bertransisi dari satu keadaan ke keadaan lainnya berdasarkan simbol masukan yang diberikan.

2. Pernyataan NFA secara formal

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

Ket:

- Q =Himpunan state/kedudukan
- Σ =Himpunan symbol input/masukan/abjad
- Δ =Fungsi transisi
- Q0 =State awal q0, dimana q0 ε Q
- F =Himpunan state akhir,FQ

3. Perbedaeterministic FA dan Non Deterministic FA

DFA	NFA
• Setiap keadaan hanya memiliki satu transisi yang ditentukan untuk setiap simbol masukan. Dengan kata lain, jika otomata berada dalam keadaan tertentu dan menerima simbol masukan tertentu, hanya ada satu keadaan berikutnya yang dapat dicapai.	• Sebuah keadaan dapat memiliki beberapa transisi yang berbeda untuk simbol masukan yang sama. Dalam NFA, ketika simbol masukan tertentu muncul, otomata dapat memilih untuk melakukan transisi ke salah satu atau lebih keadaan berikutnya, atau bahkan tidak melakukan transisi sama sekali.
• DFA adalah otomata deterministik karena perilaku otomata sepenuhnya ditentukan oleh keadaan saat ini dan simbol masukan saat ini. Ini berarti jika diberikan keadaan dan simbol masukan, akan selalu ada keadaan berikutnya yang ditentukan secara unik.	• NFA adalah otomata nondeterministik karena perilaku otomata tidak sepenuhnya ditentukan oleh keadaan saat ini dan simbol masukan saat ini. Dalam NFA, ada kemungkinan beberapa jalur atau pilihan yang dapat diambil saat melakukan transisi ke keadaan berikutnya.
• DFA menerima suatu bahasa	• NFA menerima suatu bahasa

jika setelah memproses semua simbol masukan, otomata berada dalam keadaan akhir (diterima) atau keadaan non-akhir (ditolak).	jika ada setidaknya satu jalur atau urutan transisi yang menghasilkan keadaan akhir (diterima). Dalam NFA, tidak semua jalur yang mungkin perlu menghasilkan keadaan akhir, yang berarti bahasa diterima secara nondeterministik.
• DFA lebih mudah dianalisis dan diimplementasikan karena perilakunya dapat sepenuhnya ditentukan oleh fungsi transisi yang jelas.	NFA lebih rumit dianalisis dan diimplementasikan karena keadaan dan transisinya yang nondeterministik memerlukan mekanisme tambahan untuk menentukan pilihan transisi yang harus diambil.

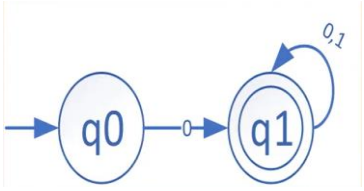
G. Ekuivalensi Non-Deterministic FA (NFA)

1. Pengertian Ekuivalensi NFA

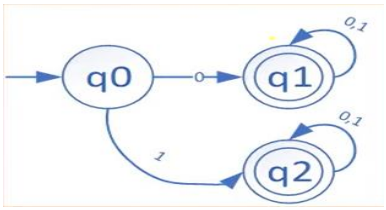
Ekivalensi NFA (Nondeterministic Finite Automaton) adalah konsep yang mengacu pada kemampuan dua NFA yang berbeda untuk mengenali bahasa yang sama. Dua NFA dikatakan setara jika keduanya mengenali bahasa yang sama, yaitu jika mereka menghasilkan keluaran yang identik ketika diberi masukan yang sama.

Ekivalensi DFA (Deterministic Finite Automaton) merujuk pada konsep di mana dua DFA yang berbeda mengenali bahasa yang sama. Dua DFA dikatakan setara jika keduanya memiliki perilaku yang sama, yaitu jika keduanya menerima dan menolak string yang sama.

2. Contoh ekuivalensi NFA dan DFA



(Deterministic FA)



(Non-Deterministic FA)

Meskipun berbeda kedua ekuivalen, ekuivalen berarti mampu menerima bahasa yang sama

H. ε-move dan ε-closure pada Non Deterministic Finite State Otomata

1. Dalam Non-Deterministic Finite State Automaton (NFA), terdapat konsep ε-move (ε-transisi) dan ε-closure (ε-penutupan) yang memungkinkan transisi tambahan yang tidak memerlukan simbol masukan tertentu.

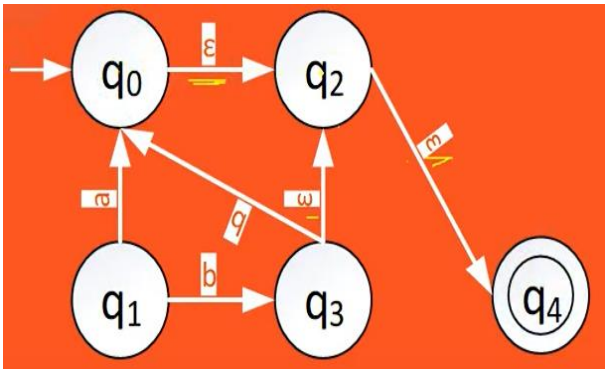
a ε-move (ε-transisi)

ε-move atau ε-transisi adalah kemampuan NFA untuk berpindah dari satu keadaan ke keadaan lainnya tanpa memerlukan simbol masukan apa pun. Dalam NFA, ε-transisi memungkinkan otomata untuk bergerak secara bebas tanpa mengonsumsi simbol masukan. Artinya, saat berada dalam keadaan tertentu, NFA dapat langsung beralih ke keadaan lain tanpa memproses simbol masukan.

b ε-closure (ε-penutupan)

ε-closure atau ε-penutupan adalah kumpulan semua keadaan yang dapat dicapai dari keadaan awal dengan menggunakan satu atau lebih ε-transisi. Dalam NFA, ε-closure dari suatu keadaan adalah himpunan semua keadaan yang dapat dijangkau dari keadaan tersebut dengan melakukan nol atau lebih ε-transisi.

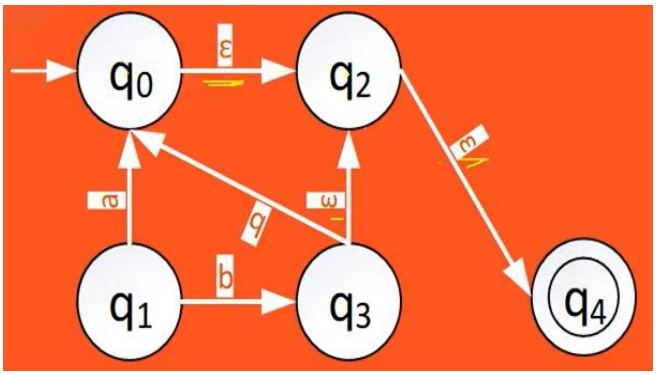
Contoh NFA dengan ε-move



- Dari q0 tanpa membaca input dapa berpindah ke q2
- Dari q2 tanpa membaca input dapat berpindah ke q4

- Dari q3 tanpa membaca input dapat berpindah ke q2

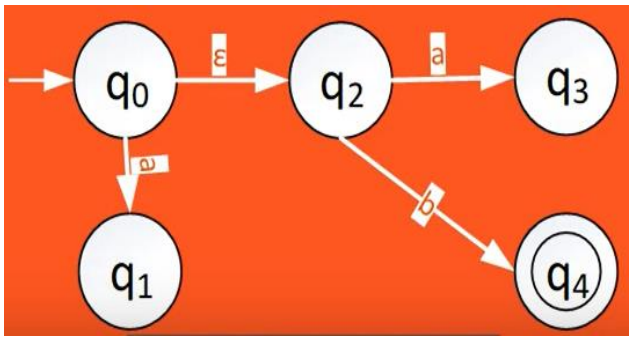
Contoh ε-clouser untuk NFA dengan ε-move



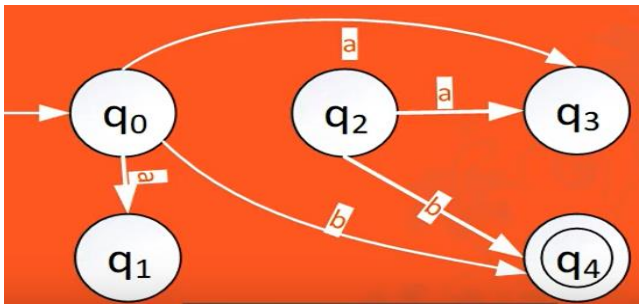
- ε-clouser (q0)={q0,q2,q4}
- ε-clouser (q1)={q1}
- ε
- ε-clouser (q2)={q2,q4}
- ε-clouser (q3)={q3,q2,q4}
- ε-clouser(q4)={q4}

I. Ekuivalen NFA dengan ε-move ke NFA tanpa ε-move

1 Contoh NFA tanpa ε-move dari NFA dengan ε-move yang ekuivalen



(NFA dengan ε-move)



(NFA tanpa ε-move)

J. Finite State Transducer (Moore Machine)

1 Finite State Trnasducer

Finite State Transducer (FST) adalah model matematika yang digunakan untuk menggambarkan dan memodelkan pemetaan antara urutan simbol masukan dan urutan simbol keluaran. FST merupakan perluasan dari Finite State Automaton (FSA) dengan tambahan fungsi keluaran.

2 Jenis Finite State Transducer

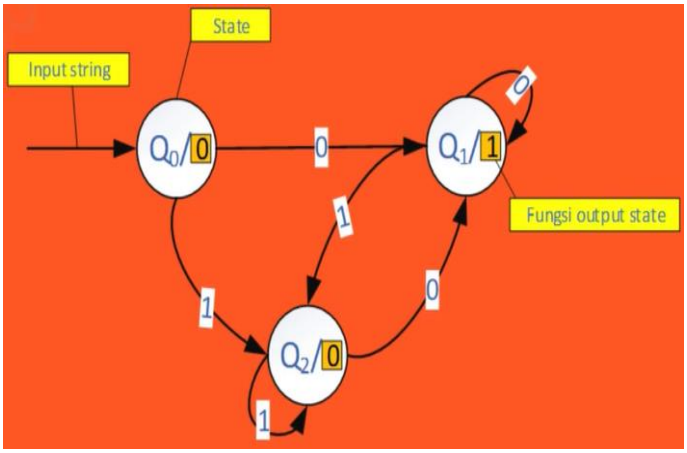
a. Moore Machine

Didefenisikan dengan 6 tupel

- **Keadaan (State):** Moore Machine memiliki sejumlah keadaan diskrit, yang merepresentasikan kondisi sistem pada suatu waktu. Setiap keadaan memiliki keluaran yang terkait.
- **Transisi (Transition):** Transisi dalam Moore Machine ditentukan oleh masukan yang diterima. Setiap transisi mengubah keadaan dari satu keadaan ke keadaan lainnya. Namun, perubahan keadaan tidak mempengaruhi keluaran pada saat transisi terjadi.
- **Keluaran (Output):** Setiap keadaan dalam Moore Machine memiliki keluaran yang terkait. Output ditentukan oleh keadaan saat ini, tetapi tidak bergantung pada transisi antara keadaan.
- **Keadaan Awal (Initial State):** Moore Machine memiliki keadaan awal di mana mesin dimulai saat pertama kali dijalankan.
- **Keadaan Akhir (Final State):** Moore Machine dapat memiliki keadaan akhir tertentu yang menandakan akhir dari urutan masukan atau kondisi yang dicapai.

Contoh Moore Machine

- Q ={Q0,Q1,Q2}
- Σ ={0,1}
- S ={Q0}
- Δ ={0,1}
- ɤ0 =0
- ɤ1 =1
- ɤ2 =0
  - δ
    - δ(Q0,0)=Q1
    - δ(Q0,1)=Q2
    - δ(Q1,0)=Q1
    - δ(Q1,1)=Q2
    - δ(Q2,0)=Q1
    - δ(Q2,1)=Q2



K. Pohon Penurunan, Parsing, Ambiguitas

1. Teori Bahasa bebas konteks atau CFG

CFG adalah singkatan dari Context-Free Grammar. Ini adalah tata bahasa formal yang menggambarkan sintaksis dari bahasa formal. Grammar konteks-bebas terdiri dari kumpulan aturan produksi yang menentukan bagaimana simbol-simbol (non-terminal) dapat digantikan oleh urutan simbol lainnya, termasuk terminal (token) dan non-terminal. Grammar ini dimulai dengan satu simbol non-terminal tunggal yang disebut simbol awal, dan dengan menerapkan aturan produksi, ia dapat menghasilkan semua kalimat yang valid dalam bahasa tersebut.

2. Batasan aturan produksi CFG

a. Setiap aturan produksi harus terdiri dari sebuah simbol non-terminal di sebelah kiri dan urutan simbol (terminals dan/atau non-terminals) di sebelah kanan.
b. Simbol non-terminal hanya dapat digantikan oleh urutan simbol di sebelah kanan aturan produksi.
c. Setiap simbol non-terminal harus memiliki setidaknya satu aturan produksi yang menggantikannya.
d. Tidak ada aturan produksi yang menghasilkan urutan simbol kosong (ε) kecuali jika aturan tersebut secara khusus didefinisikan.

3. Contoh aturan produksi CFG

- S -> NP VP Aturan ini menyatakan bahwa sebuah kalimat (S) terdiri dari frasa nomina (NP) diikuti oleh frasa verbal (VP).
- NP -> Det N Aturan ini menyatakan bahwa frasa nomina (NP) terdiri dari determiner (Det) diikuti oleh nomina (N).
- VP -> V NP Aturan ini menyatakan bahwa frasa verbal (VP) terdiri dari kata kerja (V) diikuti oleh frasa nomina (NP).
- NP -> NP PP Aturan ini menyatakan bahwa frasa nomina (NP) dapat diikuti oleh frasa preposisi (PP).
- PP -> P NP Aturan ini menyatakan bahwa frasa preposisi (PP) terdiri dari preposisi (P) diikuti oleh frasa nomina (NP).

4. CFG vs RE

- **Kemampuan Ekspresif**

  CFG lebih ekspresif daripada regular expression. CFG dapat menggambarkan bahasa konteks-bebas, yang mencakup struktur bahasa yang lebih kompleks seperti tanda kurung bersarang, tanda kurung seimbang, dan pola rekursif. Regular expression, di sisi lain, dapat menggambarkan bahasa reguler, yang kurang ekspresif dan tidak dapat mengatasi struktur bersarang atau pola rekursif.

- **Formalisme**

  CFG adalah tata bahasa formal yang terdiri dari kumpulan aturan produksi yang mendefinisikan bagaimana simbol-simbol dapat digantikan. CFG menggunakan non-terminal, terminal, dan aturan produksi untuk menghasilkan rangkaian string yang valid dalam bahasa tersebut. Regular expression, di sisi lain, adalah pola string yang digunakan untuk pencocokan dan pencarian pola. Regular expression terdiri dari karakter, metakarakter, dan operator untuk mendefinisikan pola.

- **Parsing**

  CFG dapat diparsing menggunakan teknik seperti top-down parsing atau bottom-up parsing untuk menganalisis dan menghasilkan kalimat yang valid dalam bahasa tersebut. Regular expression, di sisi lain, biasanya digunakan untuk pencocokan dan pencarian pola dalam string, bukan untuk parsing atau menghasilkan bahasa terstruktur.

- **Kelas Bahasa**

  CFG menggambarkan bahasa konteks-bebas, yang merupakan subset dari bahasa reguler. Regular expression menggambarkan bahasa reguler, yang merupakan subset dari bahasa konteks-bebas. Bahasa konteks-bebas dapat memiliki struktur dan pola yang lebih kompleks daripada bahasa reguler.

5. Tree

Pohon atau tree adalah sebuah graph terhubung tidak sirkuler yang memiliki satu simpul (node)/vertex yang disebut akar (root) dan dari situ kita memiliki lintasan ke setiap simpul.

- **Derivation Tree (Pohon Penurunan)**

  Derivation Tree atau Pohon Penurunan adalah struktur pohon yang digunakan untuk menggambarkan langkah-langkah penurunan suatu string dalam tata bahasa konteks-bebas (Context-Free Grammar/CFG). Pohon penurunan menggambarkan cara-cara bagaimana aturan produksi CFG

diterapkan untuk menghasilkan string yang valid dalam bahasa yang didefinisikan oleh CFG.

- **Proses penurunan/parsing**
  1 Letmost Derivation
    Simbol variabel yang paling kiri diturunkan terlebih dahulu
  2 Rightmost Derivation
    Simbol variabel yang paling kanan diturunkan terlebih dahulu

6. Ambiguitas

Ambiguitas terjadi jika terdapat lebih dari satu pohon penurunan yang berbeda untuk memperoleh suatu untai.

Contoh ambiguitas

Ambiguitas dapat menimbulkan masalah pada bahasa-bahasa tertentu, baik pada bahasa alami maupun pada bahasa pemrogrman. Bila suatu struktur bahasa memiliki lebih dari satu dekomposisi, dan susunannya akan menentukan arti, maka artinya menjadi ambigu.

L. Pengaturan aturan produksi rekursif kiri

1. Pengertian aturan produksi rekursif kiri

Aturan produksi rekursif kiri adalah jenis aturan produksi dalam sebuah tata bahasa konteks-bebas (Context-Free Grammar/CFG), di mana simbol non-

terminal pada sisi kiri aturan produksi dapat muncul secara rekursif (berulang) pada sisi kanan aturan produksi.

Dalam aturan produksi rekursif kiri, non-terminal pada sisi kiri aturan produksi akan muncul kembali sebagai simbol pertama pada sisi kanan aturan produksi. Dengan kata lain, non-terminal dapat dihasilkan secara berulang dalam satu aturan produksi.

Contoh aturan produksi rekursif kiri:
S -> S + S
S -> a

Dalam contoh di atas, aturan produksi pertama, S -> S + S, memiliki simbol non-terminal S pada sisi kiri dan juga pada sisi kanan. Ini mengindikasikan adanya rekursi kiri, karena aturan produksi tersebut dapat menghasilkan simbol S secara berulang.

M. Teknik penghilangan rekursif kiri

1. **Alasan mengapa aturan produksi rekursif kiri harus dihindari atau ditransformasikan:**
   - **Ambiguitas**

     Aturan produksi rekursif kiri dapat menyebabkan ambiguitas dalam tata bahasa. Ketika ada lebih dari satu cara untuk menurunkan suatu string, parser atau pemroses bahasa dapat menghasilkan lebih dari satu pohon penurunan atau hasil yang valid. Ini dapat menyebabkan ketidakjelasan dalam pemahaman dan pengolahan bahasa.

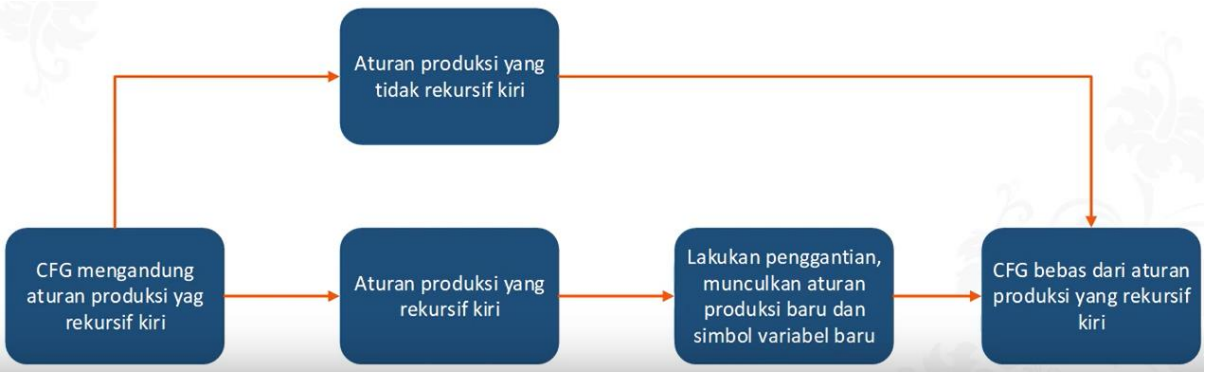
   - **Ketidakefisienan**

     Rekursi kiri dalam parsing dapat menyebabkan loop tak terbatas atau ketidakefisienan dalam pemrosesan bahasa. Jika rekursi kiri tidak ditangani dengan benar, parser dapat terjebak dalam loop tak berujung saat mencoba untuk mengurai string dengan aturan produksi yang rekursif kiri. Hal ini dapat menyebabkan waktu komputasi yang sangat lama atau bahkan kegagalan parsing.

   - **Keterbatasan beberapa parser**

     Beberapa jenis parser atau algoritma parsing, seperti parser LL(1), tidak dapat menangani secara langsung aturan produksi rekursif kiri. Parser ini membutuhkan transformasi aturan produksi menjadi rekursif kanan atau menggunakan teknik lain seperti rekursi kiri bersama atau pemotongan (factorization) untuk dapat melakukan parsing dengan benar.

2. Tahapan penghilangan rekursif kiri



a. Identifikasi aturan produksi rekursif kiri

Pertama, identifikasi aturan produksi yang memiliki rekursi kiri. Aturan produksi rekursif kiri memiliki simbol non-terminal pada sisi kiri yang juga muncul sebagai simbol pertama pada sisi kanan aturan produksi.

b. Membuat non-terminal baru

Buatlah sebuah non-terminal baru yang akan menggantikan simbol rekursif kiri pada sisi kanan aturan produksi. Misalnya, jika aturan produksi rekursif kiri adalah A -> Aα | β, maka buatlah non-terminal baru A' yang akan menggantikan A pada aturan produksi rekursif kiri.

c. Transformasi rekursi kiri menjadi rekursi kanan

Ubah aturan produksi rekursif kiri menjadi aturan produksi rekursif kanan menggunakan non-terminal baru yang telah dibuat. Gantikan setiap kemunculan simbol rekursif kiri dengan non-terminal baru di sisi kanan aturan produksi. Misalnya, A -> Aα | β akan menjadi A -> βA', A' -> αA' | ε.

d. Menghapus rekursi langsung

Jika terdapat rekursi langsung dalam aturan produksi, yaitu ketika non-terminal di sisi kanan aturan produksi diikuti oleh simbol yang sama non-terminal, maka rekursi langsung juga harus dihilangkan. Untuk menghapus rekursi langsung, buatlah non-terminal baru yang akan menggantikan setiap kemunculan rekursi langsung.

e. Penyesuaian aturan produksi lain

Perlu dilakukan penyesuaian pada aturan produksi lain yang menggunakan non-terminal yang telah mengalami transformasi. Pastikan untuk memperbarui referensi ke non-terminal baru dan mengubah urutan aturan produksi jika diperlukan.

f. Mengulangi proses

Jika masih ada aturan produksi rekursif kiri, ulangi langkah-langkah di atas untuk menghilangkan rekursi kiri dari aturan produksi yang tersisa.

N. Contoh kasus penghilangan rekursif kiri

S -> SAB | AB

A -> AA | a

B -> b

Langkah-langkah penghilangan rekursi kiri adalah sebagai berikut:

a. Identifikasi aturan produksi rekursif kiri:

- Aturan produksi S -> SAB adalah rekursif kiri karena S muncul sebagai simbol pertama di sisi kanan.
- Aturan produksi A -> AA juga rekursif kiri karena A muncul sebagai simbol pertama di sisi kanan.

b. Membuat non-terminal baru:

- Buat non-terminal baru S' untuk menggantikan S pada aturan produksi rekursif kiri S -> SAB.
- Buat non-terminal baru A' untuk menggantikan A pada aturan produksi rekursif kiri A -> AA.

Setelah langkah ini, aturan produksi kita menjadi:

S -> S'AB | AB

S' -> S'AB | ε A -> A'A | a

A' -> A'A | ε

B -> b

c. Transformasi rekursi kiri menjadi rekursi kanan:

- Ganti setiap kemunculan simbol rekursif kiri dengan non-terminal baru yang sesuai pada sisi kanan aturan produksi.

Setelah langkah ini, aturan produksi kita menjadi:

S -> S'AB | AB

S' -> S'ABB | AB

A -> A'AA | a

A' -> A'AA | ε

B -> b

O. Pengantar penyederhanaan aturan produksi context free grammar

1. **Tujuan penyederhanaan CFG**
   - **Meningkatkan keterbacaan**

     CFG yang lebih sederhana cenderung lebih mudah dipahami dan dibaca oleh manusia. Dengan mengurangi kompleksitas aturan produksi, struktur bahasa yang dihasilkan menjadi lebih jelas dan lebih mudah dipahami.\

   - **Mempermudah analisis sintaksis**

     CFG yang disederhanakan dapat mempermudah proses analisis sintaksis, baik secara manual maupun otomatis. Dalam parsing, parser akan lebih efisien dan lebih cepat dalam mencari struktur sintaksis yang benar.

   - **Mengurangi ambiguitas**

     Penyederhanaan CFG dapat membantu mengurangi atau menghilangkan ambiguitas dalam bahasa yang dihasilkan oleh CFG. Ambiguitas dapat mengakibatkan lebih dari satu penafsiran yang valid untuk string yang sama, sehingga menyulitkan pemrosesan dan pemahaman bahasa. Dengan menyederhanakan CFG, kita dapat menghindari atau meminimalkan ambiguitas tersebut.

   - **Mengoptimalkan pemrosesan bahasa**

     CFG yang disederhanakan memungkinkan pemrosesan bahasa yang lebih efisien. Dengan aturan produksi yang lebih sedikit dan lebih sederhana, waktu dan sumber daya yang dibutuhkan dalam parsing dan pemrosesan bahasa dapat dikurangi.

   - **Kemudahan modifikasi dan perluasan**

     CFG yang sederhana lebih mudah dimodifikasi atau diperluas dengan menambahkan atau mengubah aturan produksi. Jika CFG terlalu kompleks, modifikasi atau perluasan bahasa mungkin menjadi lebih sulit atau membingungkan.

2. **Cara penyederhanaan CFG**
   - **Menghilangkan simbol non-terminal yang tidak digunakan**

     Identifikasi simbol non-terminal yang tidak digunakan dalam produksi CFG. Simbol-simbol ini dapat dihapus dari CFG, karena mereka tidak berkontribusi pada pembentukan string yang valid dalam bahasa yang dihasilkan.

   - **Menghilangkan simbol non-terminal yang tidak dapat mencapai simbol terminal:**

     Identifikasi simbol non-terminal yang tidak dapat mencapai simbol terminal dalam CFG. Jika suatu simbol non-terminal tidak dapat mencapai

simbol terminal dalam satu atau beberapa langkah produksi, maka simbol non-terminal tersebut dapat dihapus.

- **Menghilangkan simbol non-terminal yang tidak dapat dijangkau dari simbol start:**
  Identifikasi simbol non-terminal yang tidak dapat dijangkau dari simbol start dalam CFG. Jika suatu simbol non-terminal tidak dapat dijangkau dari simbol start melalui langkah produksi yang valid, maka simbol non-terminal tersebut dapat dihapus.

- **Menggabungkan aturan produksi yang serupa**
  Identifikasi aturan produksi yang memiliki sisi kiri dan sisi kanan yang serupa atau mirip. Gabungkan aturan produksi tersebut menjadi satu aturan produksi tunggal dengan variasi yang diatur melalui simbol alternatif (|).

- **Menghapus atau mengganti aturan produksi yang tidak perlu**
  Identifikasi aturan produksi yang tidak diperlukan atau redundan dalam CFG. Aturan produksi yang tidak berkontribusi pada pembentukan string yang valid atau dapat diwakili oleh aturan produksi lain yang lebih sederhana dapat dihapus atau diganti dengan aturan produksi yang lebih ringkas.

- **Mengubah aturan produksi menjadi bentuk normal yang sederhana:**
  CFG dapat diubah menjadi bentuk normal yang lebih sederhana, seperti bentuk Chomsky Normal Form (CNF) atau bentuk Greibach Normal Form (GNF). Proses ini melibatkan transformasi aturan produksi untuk memastikan setiap aturan produksi memiliki bentuk yang terdefinisi dengan jelas.

P. Penghilangan produksi kosong auran produksi context free grammar

1. **Defenisi produksi ε**
   Produksi ε (epsilon) dalam konteks Context-Free Grammar (CFG) mengacu pada aturan produksi yang mengizinkan simbol non-terminal untuk menghasilkan string kosong atau string dengan panjang nol. Dalam produksi ε, ε merupakan representasi dari string kosong atau string dengan tidak ada simbol di dalamnya.

   Aturan produksi ε umumnya ditulis dalam bentuk:

   A -> ε

Q. Penghilangan produksi unit aturan produksi context free grammar

1. **Defnisi produksi unit**
   Produksi unit dalam konteks Context-Free Grammar (CFG) mengacu pada aturan produksi yang memungkinkan satu simbol non-terminal menghasilkan satu simbol non-terminal lainnya. Aturan produksi unit menghubungkan satu simbol non-terminal dengan simbol non-terminal lainnya dalam satu langkah produksi.

Aturan produksi unit umumnya ditulis dalam bentuk:

A -> B

Contoh penghilangan produksi unit

- S->Sb
- S->C
- C->D
- C->ef
- D->dd

Proses penggantian:

- C->D -> C->dd
- S->C ->S->dd|ef

Setalah penyederhanaan

- S->Sb
- S->dd|ef
- C->dd
- C->ef
- D->dd

R. Penghilangan produksi useless pada aturan produksi context free grammar

1. **Defenisi produksi useless**

   Produksi yang tidak berguna (useless production) dalam konteks Context-Free Grammar (CFG) merujuk pada aturan produksi yang tidak berkontribusi pada pembentukan string yang valid dalam bahasa yang didefinisikan oleh CFG tersebut. Aturan produksi yang tidak berguna dapat berupa aturan produksi yang tidak pernah digunakan atau tidak dapat diakses dari simbol start CFG.

2. **Contoh produksi useless**
   - S->aSa | Abd | Bde
   - A->Ada
   - B->BBB |a

   Dapat diketahui bahwa:

   a. Simbol variabel A tidak memiliki penurunan yang menuju terminal sehingga dapat dihilangkan
   b. Konsekuensi no (1), aturan produksi S-> Abd tidak memiliki penurunan

S. Contoh penyederhanaan aturan produksi context free grammar

1. Tiga penyederhanaan dilakukan bersama pada suatu CFG

a. Menghilangkan produksi ε (epsilon):

- Menghapus aturan produksi yang menghasilkan string kosong (ε).
- Menghapus simbol non-terminal yang hanya menghasilkan string kosong.

b. Menghilangkan produksi unit:

- Menggabungkan aturan produksi yang memiliki satu simbol non-terminal di sisi kanan menjadi satu aturan produksi dengan dua atau lebih simbol non-terminal.

c. Menghapus simbol non-terminal yang tidak dapat dijangkau:

- Menghapus simbol non-terminal yang tidak dapat dijangkau dari simbol start CFG.
- Menghapus aturan produksi yang tidak terhubung dengan simbol start atau tidak berkontribusi pada pembentukan string yang valid.

2. Urutan langkah penyederhanaan CFG

T. Pengantar chomsky normal foem (CNF)

1. Bentuk normal chomsky

Bentuk Normal Chomsky (Chomsky Normal Form/CNF) adalah salah satu bentuk normal untuk Context-Free Grammar (CFG). Dalam bentuk ini, setiap aturan produksi CFG memiliki dua bentuk spesifik:

- Aturan produksi dengan sisi kanan yang terdiri dari dua simbol non-terminal:

  A -> BC

- Aturan produksi dengan sisi kanan yang terdiri dari satu simbol terminal:

  A -> a

2. Syarat konversi CFD ke CNF

- Tidak memiliki produksi useless
- Tidak memiliki produksi unit
- Tidak memiliki produksi ε (kosong)

3. Aturan produksi CNF

Ruas kanannya tepat berupa sebuah symbol terminal atau dua variabel. Jika terapat lebih dari satu symbol terminal maka harus dilakukan penggantian dan juga jika terdapat lebih dari dua symbol variabel harus dilakukan perubahan.

U. Pembentukan Chomsky Normal Form dari Context Free Grammar

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

2. **Tahapan pembentukan CNF**
   - **Menyederhanakan Ekspresi Logika**

     Langkah pertama dalam pembentukan CNF adalah menyederhanakan ekspresi logika awal. Ini melibatkan penggunaan aturan dan properti aljabar Boolean untuk mengurangi kompleksitas ekspresi logika. Misalnya, menggunakan aturan De Morgan untuk mengubah penyangkalan negasi dan konjungsi menjadi disjungsi.

   - **Mengubah ke Dalam Bentuk Normal**

     Setelah menyederhanakan ekspresi logika, langkah selanjutnya adalah mengubahnya menjadi bentuk normal. Ada beberapa bentuk normal yang mungkin, tetapi yang paling umum adalah bentuk disjungsi normal (DNF) atau bentuk konjungsi normal (CNF). Dalam tahapan ini, kita akan berfokus pada pembentukan CNF.

   - **Mentransformasikan ke CNF**

     Untuk mentransformasikan ekspresi logika menjadi CNF, kita harus memperhatikan beberapa aturan transformasi. Aturan-aturan ini meliputi:

     a. Distributivitas: Menggunakan hukum distribusi untuk mengubah konjungsi dalam ekspresi logika menjadi disjungsi.
     b. Asosiativitas: Menggunakan asosiativitas konjungsi atau disjungsi untuk mengelompokkan klausa-klausa bersama-sama.
     c. Komutativitas: Mengubah urutan konjungsi atau disjungsi tidak mempengaruhi hasil akhir.
     d. Menghilangkan Implikasi: Menggunakan aturan De Morgan untuk menghilangkan implikasi dalam ekspresi logika.
     e. Hukum Idempoten: Menghilangkan klausa yang berulang dalam konjungsi atau disjungsi.

   - **Normalisasi CNF**

     Setelah mentransformasikan ekspresi logika ke dalam bentuk CNF, langkah terakhir adalah melakukan normalisasi CNF. Ini melibatkan mengatur klausa-klausa dalam CNF secara teratur, seperti menghapus klausa yang

berulang, menyusun variabel dalam urutan tertentu, dan menghapus klausa yang kosong.

V. Algoritma Cocke-Younger-Kasami (CFG-CNF)

1. Pengertian CYK

Algoritma Cocke-Younger-Kasami (CYK) adalah algoritma parsing yang digunakan untuk memeriksa apakah suatu rangkaian kata dapat dihasilkan oleh sebuah CFG yang berada dalam bentuk normal Chomsky (CNF). Algoritma ini beroperasi dengan menggunakan tabel dinamis yang disebut dengan tabel CYK.

2. Syarat-syarat CYK

- CFG dalam CNF
  Algoritma CYK membutuhkan CFG yang berada dalam bentuk normal Chomsky (CNF). CFG harus diubah menjadi CNF sebelum digunakan dengan algoritma CYK. CFG dalam CNF memiliki aturan produksi yang terbatas pada dua simbol non-terminal atau satu simbol terminal.
- Tabel CYK yang memadai
  Algoritma CYK menggunakan tabel CYK untuk mengorganisir informasi tentang kemungkinan produksi dalam CFG. Tabel CYK harus memiliki ukuran yang cukup untuk menampung informasi tentang substring yang mungkin dalam rangkaian kata.

3. Tujuan CYK

- **Parsing**
  Tujuan utama dari algoritma CYK adalah melakukan parsing atau analisis sintaksis pada rangkaian kata untuk memeriksa apakah rangkaian kata tersebut dapat dihasilkan oleh CFG yang diberikan. Jika rangkaian kata dapat dihasilkan oleh CFG, algoritma CYK akan membangun tabel CYK yang mencerminkan struktur sintaksis yang mungkin.

- **Pengenalan Struktur**
  Selain memeriksa apakah rangkaian kata dapat dihasilkan oleh CFG, algoritma CYK juga memberikan informasi tentang struktur sintaksis yang mungkin dalam bentuk tabel CYK. Setiap sel tabel CYK akan berisi informasi tentang simbol-simbol non-terminal yang dapat menghasilkan substring yang sesuai dalam rangkaian kata.

- **Pemulihan Struktur**
  Jika rangkaian kata diterima oleh CFG, algoritma CYK dapat digunakan untuk memulihkan struktur sintaksis dari tabel CYK. Dengan memeriksa tabel CYK, kita dapat menentukan bagaimana simbol-simbol non-terminal dalam CFG dikombinasikan untuk menghasilkan rangkaian kata.

4. Algoritma CYK

```
1)  begin
2)    for i:= 1 to n do
3)       Vi1 ={ A| A→a aturan produksi di mana symbol ke-i adalah a}
4)       for j:= 2 to n do
5)          for i:= 1 to (n-j+1) do
6)             begin
7)                Vij:=Ø;
8)                for k:=1 to (j-1) do
9)                   Vij =Vij ∪ { A | A → BC adalah produksi di mana B di Vik dan C di Vi+k,j-k
10)            end
11)   end
```

Ket:

- n= panjang string yang akan diperiksa, misalnya ‘ada’, n = |ada| =3
- I menyatakan kolom ke-
- J menyatakan baris ke-
- Tahapan no (2) dan (3) untuk mengisi tabel baris pertama kolom 1-n
- No (4), literasi dari baris ke-2 sampai n
- No (5), literasi untuk mengisi kolom 1 sampai (n-baris+1) pada suatu baris
- N0 (7), inisialisasi Vij dengan ø
- No (8) dan (9), literasi untuk memeriksa mana saja yang menjadi anggota Vij