RANGKUMAN MATERI

“TORI BAHASA & AUTOMATA”

“Disusun sebagai tugas akhir & UAS pada mata kuliah Teori Bahasa & Automata , dengan Dosen Widya Darwin S.Pd, M.Pd.T”

Disusun Oleh :

IFDAL LISYUKRI
21346012

PROGRAM STUDI S1 TEKNIK INFORMATIKA

JURUSAN TEKNIK ELEKTRONIKA

FAKULTAS TEKNIK

UNIVERSITAS NEGERI PADANG

TAHUN 2023

KATA PENGANTAR

Puji syukur kita hanturkan kehadiran Tuhan Yang Maha Esa, yang telah melimpahkan rahmat dan karunianya kepada kita, sehingga kita dapat menyusun dan menyajikan makalah Teori bahasa & Automata ini dengan tepat waktu. Tak lupa pula kita mengucapkan terima kasih kepada berbagai pihak yang telah memberikan dorongan dan motivasi. Sehingga makalah ini dapat tersusun dengan baik.

Makalah ini dibuat sebagai salah satu syarat untuk memenuhi tugas mata kuliah Teori bahasa & Automata . Kami menyadari bahwa dalam penyusunan makalah ini masih terdapat banyak kekurangan dan jauh dari kata sempurna. Oleh karena itu, Kami mengharapkan kritik dan saran untuk menyempurnakan makalah ini dan dapat menjadi acuan dalam menyusun makalah-makalah selanjutnya. Kami muga memohon maaf apabila dalam penulisan makalah ini terdapat kesalahan kata - kata, pengetikan dan kekeliruan, sehingga membingungkan pembaca dalam memahami maksud penulis.

Adapun makalah ini penulis rangkum dari beberapa sumber yang dapat dipercaya yang sajian penulisnya, dirangkum dalam lembar Daftar Pustaka dengan harapan makalah ini dapat menambah pengetahuan kita tentang Pendidikan Di Era Teknologi Informasi Dan Komunikasi. Demikian yang dapat kami sampaikan. Akhir kata, semoga makalah ini dapat menambah wawasan bagi kita semua.

Lengayang, Juni 2023

Ifdal Lisyukri

DAFTAR ISI

Teori Bahasa & Automata

BAB I
PENGANTAR TEORI BAHASA DAN AUTOMATA

Pengantar Teori Bahasa dan Automata adalah cabang ilmu komputer yang mempelajari bahasa formal, otomata, dan komputabilitas.

A. Bahasa Formal:

Bahasa adalah himpunan simbol-simbol yang memiliki arti atau makna tertentu.Bahasa formal adalah bahasa yang ditentukan oleh aturan-aturan tertentu yang terdiri dari simbol-simbol dari suatu alfabet.

Contoh bahasa formal adalah bahasa pemrograman, bahasa matematika, dan bahasa alami yang diubah menjadi bentuk formal.

B. Teori Bahasa:

Teori Bahasa adalah studi tentang bahasa formal, struktur bahasa, dan kemampuan bahasa untuk mengungkapkan gagasan dan informasi.Menjelaskan struktur bahasa seperti tata bahasa, semantik, dan sintaksis.Membahas hubungan antara bahasa formal dengan mesin komputasi.

C. Otomata:

Otomata adalah mesin abstrak yang menerima masukan dan menghasilkan keluaran berdasarkan aturan tertentu.Digunakan untuk memodelkan dan menganalisis perilaku sistem yang memiliki sifat-sifat yang dapat diotomatiskan.Jenis otomata yang umum adalah Automata Finite (FA), Pushdown Automata (PDA), dan Turing Machine (TM).

D. Automata Finite (FA):

FA adalah model otomata yang memiliki jumlah keadaan terbatas.Terdiri dari himpunan keadaan, alfabet input, fungsi transisi, keadaan awal, dan keadaan akhir.Digunakan untuk mengenali bahasa-bahasa formal, seperti regular expressions dan regular languages.

E. Context-Free Grammar (CFG):

CFG adalah bentuk notasi untuk menggambarkan struktur sintaksis dari suatu bahasa.Terdiri dari simbol-simbol terminal, simbol-simbol non-terminal, aturan produksi, dan simbol awal.Digunakan untuk menghasilkan struktur sintaksis dalam bahasa pemrograman dan analisis sintaksis.

F. Pushdown Automata (PDA):

PDA adalah model otomata yang memiliki kemampuan penyimpanan sementara dalam bentuk tumpukan (stack). Digunakan untuk mengenali bahasa-bahasa konteks-bebas, seperti context-free grammar.

G. Turing Machine (TM):

TM adalah model otomata yang lebih kuat daripada FA dan PDA.Terdiri dari kepala pembaca, pita tak hingga, tabel transisi, dan keadaan awal/akhir.Digunakan untuk mengenali bahasa-bahasa rekursif dan komputabilitas.

H. Komputabilitas:

Konsep komputabilitas mempelajari batasan dan kemampuan sistem komputasi.Memiliki hubungan dengan teori bahasa dan otomata dalam memahami apa yang dapat dihitung dan apa yang tidak.

BAB II
SIMBOL STRING & BAHASA

Simbol, String, dan Bahasa adalah konsep dasar dalam teori bahasa formal dan automata.

A. Simbol:

Simbol adalah unit dasar yang digunakan dalam sebuah bahasa.Simbol dapat berupa huruf, angka, karakter khusus, atau lambang lainnya.Dalam konteks teori bahasa formal, simbol-simbol ini membentuk alfabet.

B. Alfabet:

Alfabet adalah himpunan simbol-simbol yang dapat digunakan dalam sebuah bahasa.Misalnya, dalam bahasa Inggris, alfabet terdiri dari huruf-huruf A-Z.Dalam teori bahasa formal, alfabet sering kali disimbolkan dengan simbol Sigma (Σ).

C. String:

String adalah urutan terbatas dari simbol-simbol yang berasal dari sebuah alfabet.String dapat berupa kombinasi simbol-simbol apa pun yang terdefinisi dalam alfabet.Misalnya, "Hello" adalah sebuah string dalam bahasa Inggris.

D. Panjang String:

Panjang sebuah string adalah jumlah simbol yang terkandung di dalamnya.Panjang string dapat dihitung dengan menghitung jumlah simbol-simbol dalam string tersebut.Misalnya, panjang string "Hello" adalah 5.

E. Concatenation:

Concatenation adalah operasi penggabungan dua atau lebih string untuk membentuk string baru. Dalam operasi concatenation, string-string ditempatkan secara berurutan. Misalnya, jika kita menggabungkan string "Hello" dan "World", hasilnya adalah "HelloWorld".

F. Bahasa:

Bahasa adalah himpunan string yang terdiri dari simbol-simbol dalam suatu alfabet.Bahasa dapat berupa himpunan string yang valid atau himpunan string yang memenuhi suatu aturan tertentu.Contoh bahasa termasuk bahasa Inggris, bahasa pemrograman, dan bahasa alami.

G. Bahasa Formal:

Bahasa formal adalah bahasa yang didefinisikan secara formal melalui aturan-aturan yang ketat.Bahasa formal dapat dijelaskan dengan menggunakan alat-alat matematis, seperti teori bahasa formal.Contoh bahasa formal termasuk regular languages, context-free languages, dan formal languages.

H. Ekspresi Reguler:

Ekspresi reguler adalah notasi untuk menggambarkan bahasa-bahasa regular menggunakan simbol-simbol dan operasi-reguler.Ekspresi reguler digunakan untuk mengenali, menghasilkan, atau memanipulasi string yang sesuai dengan aturan tertentu.

BAB III
TATA BAHASA HIRARKI CHOMSKY & ATURAN PRODUKSI

Tata Bahasa Hirarki Chomsky dan Aturan Produksi adalah konsep penting dalam teori bahasa formal dan linguistik.

A. Tata Bahasa Hirarki Chomsky:

Tata Bahasa Hirarki Chomsky adalah hierarki yang mengklasifikasikan bahasa-bahasa formal berdasarkan kompleksitas dan struktur sintaksisnya.Terdapat empat tingkat tata bahasa dalam hierarki Chomsky: tata bahasa tak terbatas, tata bahasa konteks-sensitif, tata bahasa konteks-bebas, dan tata bahasa reguler.Setiap tingkat dalam hierarki Chomsky lebih terbatas daripada tingkat sebelumnya.

B. Tata Bahasa Tak Terbatas (Unrestricted Grammar):

Tata Bahasa Tak Terbatas adalah tingkat paling tinggi dalam hierarki Chomsky.Tata bahasa ini memiliki aturan produksi yang sangat fleksibel dan dapat menghasilkan bahasa-bahasa yang sangat kompleks.Gramatika tak terbatas tidak terbatasi oleh jumlah produksi atau panjang string.

C. Tata Bahasa Konteks-Sensitif (Context-Sensitive Grammar):

Tata Bahasa Konteks-Sensitif adalah tingkat kedua dalam hierarki Chomsky.Tata bahasa ini menggunakan aturan produksi yang mempertimbangkan konteks dan memiliki batasan struktural yang lebih ketat daripada tata bahasa tak terbatas.Setiap aturan produksi dalam tata bahasa konteks-sensitif harus memiliki jumlah simbol yang sama atau lebih banyak pada sisi kiri daripada sisi kanan.

D. Tata Bahasa Konteks-Bebas (Context-Free Grammar):

Tata Bahasa Konteks-Bebas adalah tingkat ketiga dalam hierarki Chomsky.Tata bahasa ini menggunakan aturan produksi yang tidak bergantung pada konteks, artinya setiap simbol non-terminal pada sisi kiri dapat menggantikan simbol non-terminal apa pun pada sisi kanan.Aturan produksi dalam tata bahasa konteks-bebas biasanya terdiri dari simbol non-terminal diikuti oleh simbol terminal atau simbol non-terminal.

E. Tata Bahasa Reguler (Regular Grammar):

Tata Bahasa Reguler adalah tingkat terendah dalam hierarki Chomsky.Tata bahasa ini menggunakan aturan produksi yang sederhana dengan batasan struktural yang ketat.Aturan produksi dalam tata bahasa reguler memiliki format yang sederhana, seperti simbol non-terminal diikuti oleh simbol terminal.

F. Aturan Produksi:

Aturan produksi, juga dikenal sebagai produksi atau aturan derivasi, adalah aturan yang digunakan dalam gramatika untuk menghasilkan struktur sintaksis dalam bahasa formal.Aturan produksi terdiri dari simbol non-terminal yang diubah menjadi urutan simbol terminal dan/atau simbol non-terminal.

Misalnya, aturan produksi dalam tata bahasa konteks-bebas dapat dituliskan sebagai A -> XYZ, di mana A adalah simbol non-terminal dan XYZ adalah urutan simbol terminal dan/atau simbol non-terminal.

.

BAB IV
FINITE STATE AUTOMATA

Finite State Automata (FSA), juga dikenal sebagai Finite State Machine (FSM), adalah model matematis yang digunakan untuk mewakili dan mengenali bahasa-bahasa yang diperlakukan secara terbatas.

A. Definisi:

Finite State Automata adalah mesin abstrak yang terdiri dari sejumlah keadaan (states) yang terbatas.FSA beroperasi dengan membaca urutan simbol input dari sebuah alfabet yang ditentukan.FSA melakukan transisi antar keadaan berdasarkan simbol input yang sedang dibaca.

B. Komponen FSA:

States (Keadaan): FSA memiliki himpunan keadaan, di mana setiap keadaan mewakili kondisi yang berbeda dalam mesin.

Transisi: FSA memiliki fungsi transisi yang menentukan bagaimana mesin berpindah dari satu keadaan ke keadaan lain berdasarkan simbol input.

Alfabet: FSA beroperasi pada alfabet tertentu, yang merupakan himpunan simbol-simbol yang valid dalam bahasa yang dikenali oleh mesin.

Keadaan Awal: FSA memiliki satu keadaan awal di mana mesin dimulai.

Keadaan Akhir (Final States): FSA memiliki satu atau beberapa keadaan akhir yang menunjukkan bahwa mesin telah mengenali sebuah string.

C. Jenis-jenis FSA:

- Deterministic Finite Automata (DFA): DFA adalah FSA di mana setiap kombinasi keadaan dan simbol input memiliki tepat satu transisi yang didefinisikan.
- Nondeterministic Finite Automata (NFA): NFA adalah FSA di mana terdapat beberapa transisi yang mungkin dari suatu keadaan dengan simbol input yang sama.
- NFA dengan ε-Transisi (ε-NFA): ε-NFA adalah NFA yang juga memiliki transisi ε, yang memungkinkan perpindahan ke keadaan berikutnya tanpa membaca simbol input.

D. Pengenalan Bahasa oleh FSA:

FSA dapat digunakan untuk mengenali bahasa-bahasa formal, seperti bahasa regular.Sebuah string dikatakan diterima oleh FSA jika mesin berakhir di salah satu keadaan akhir setelah membaca seluruh string input.FSA dapat digunakan untuk mengenali dan membedakan bahasa-bahasa yang dapat dihasilkan oleh regular expressions dan regular grammars.

E. Keterbatasan FSA:

FSA hanya dapat mengenali bahasa-bahasa yang dapat dikenali oleh bahasa regular.FSA tidak dapat mengenali bahasa-bahasa dengan struktur yang lebih kompleks, seperti bahasa konteks-bebas atau bahasa tak terbatas.

BAB V
DETERMINISTIC FINITE AUTOMATA

Deterministic Finite Automata (DFA) adalah jenis Finite State Automata (FSA) di mana setiap kombinasi keadaan dan simbol input memiliki tepat satu transisi yang didefinisikan.

A. Definisi:

Deterministic Finite Automata (DFA) adalah model matematis yang terdiri dari sejumlah keadaan (states) yang terbatas.DFA beroperasi dengan membaca urutan simbol input dari sebuah alfabet yang ditentukan.DFA melakukan transisi antar keadaan berdasarkan simbol input yang sedang dibaca.

B. Komponen DFA:

- States (Keadaan): DFA memiliki himpunan keadaan, di mana setiap keadaan mewakili kondisi yang berbeda dalam mesin.
- Transisi: DFA memiliki fungsi transisi yang menentukan bagaimana mesin berpindah dari satu keadaan ke keadaan lain berdasarkan simbol input.
- Alfabet: DFA beroperasi pada alfabet tertentu, yang merupakan himpunan simbol-simbol yang valid dalam bahasa yang dikenali oleh mesin.
- Keadaan Awal: DFA memiliki satu keadaan awal di mana mesin dimulai.
- Keadaan Akhir (Final States): DFA memiliki satu atau beberapa keadaan akhir yang menunjukkan bahwa mesin telah mengenali sebuah string.

C. Fungsi Transisi:

- Fungsi transisi DFA adalah pemetaan yang menghubungkan keadaan saat ini dengan keadaan berikutnya berdasarkan simbol input yang sedang dibaca.
- Setiap transisi didefinisikan secara eksplisit untuk setiap kombinasi keadaan dan simbol input.
- Transisi dapat dinyatakan dalam bentuk tabel transisi atau diagram grafik.

D. Penerimaan Bahasa oleh DFA:

DFA dapat digunakan untuk mengenali bahasa-bahasa formal, seperti bahasa regular.Sebuah string dikatakan diterima oleh DFA jika mesin berakhir di salah satu keadaan akhir setelah membaca seluruh string input.Jika DFA berakhir di keadaan selain keadaan akhir, string dianggap ditolak.

E. Kegunaan DFA:

- DFA digunakan dalam pemrosesan bahasa alami, kompilasi, verifikasi perangkat lunak, dan banyak aplikasi lainnya.
- DFA digunakan untuk memodelkan dan menganalisis sistem yang mengikuti pola-pola tertentu atau menjalankan alur kerja yang terbatas.

F. Keterbatasan DFA:

- DFA hanya dapat mengenali bahasa-bahasa yang dapat dikenali oleh bahasa regular.

- DFA tidak dapat mengenali bahasa-bahasa dengan struktur yang lebih kompleks, seperti bahasa konteks-bebas atau bahasa tak terbatas.

BAB VI
NON DETERMINISTIC FINITE STATE AUTOMATA

Non-Deterministic Finite Automata (NFA) adalah jenis Finite State Automata (FSA) di mana terdapat beberapa transisi yang mungkin dari suatu keadaan dengan simbol input yang sama.

A. Definisi:

Non-Deterministic Finite Automata (NFA) adalah model matematis yang terdiri dari sejumlah keadaan (states) yang terbatas.NFA beroperasi dengan membaca urutan simbol input dari sebuah alfabet yang ditentukan.NFA memiliki kemampuan untuk melakukan transisi non-deterministik, artinya terdapat beberapa transisi yang mungkin dari satu keadaan dengan simbol input yang sama.

B. Komponen NFA:

- States (Keadaan): NFA memiliki himpunan keadaan, di mana setiap keadaan mewakili kondisi yang berbeda dalam mesin.
- Transisi: NFA memiliki fungsi transisi yang menentukan bagaimana mesin berpindah dari satu keadaan ke keadaan lain berdasarkan simbol input.
- Alfabet: NFA beroperasi pada alfabet tertentu, yang merupakan himpunan simbol-simbol yang valid dalam bahasa yang dikenali oleh mesin.
- Keadaan Awal: NFA memiliki satu keadaan awal di mana mesin dimulai.
- Keadaan Akhir (Final States): NFA memiliki satu atau beberapa keadaan akhir yang menunjukkan bahwa mesin telah mengenali sebuah string.

C. Fungsi Transisi:

- Fungsi transisi NFA adalah pemetaan yang menghubungkan keadaan saat ini dengan himpunan keadaan berikutnya berdasarkan simbol input yang sedang dibaca.
- Dalam NFA, terdapat beberapa keadaan yang dapat dicapai secara non-deterministik untuk simbol input yang sama.

D. Non-Determinism:

Non-determinism adalah kemampuan NFA untuk melakukan transisi non-deterministik saat membaca simbol input tertentu.Ketika ada beberapa transisi yang mungkin dari suatu keadaan dengan simbol input yang sama, NFA dapat memilih salah satu atau lebih transisi tersebut.

E. Penerimaan Bahasa oleh NFA:

NFA dapat digunakan untuk mengenali bahasa-bahasa formal, seperti bahasa regular.Sebuah string dikatakan diterima oleh NFA jika terdapat setidaknya satu jalur yang memungkinkan mesin berakhir di salah satu keadaan akhir setelah membaca seluruh string input.

F. Konversi NFA ke DFA:

- NFA dapat dikonversi menjadi Deterministic Finite Automata (DFA) yang setara untuk mencapai determinisme.
- Konversi NFA ke DFA melibatkan pembuatan tabel transisi dan identifikasi keadaan baru dalam DFA yang mewakili kombinasi keadaan dari NFA.

G. Keterbatasan NFA:

- NFA dapat mengenali bahasa-bahasa yang dapat dikenali oleh bahasa regular, tetapi dalam beberapa kasus, DFA diperlukan untuk mengenali bahasa yang sama.

- NFA tidak secara langsung mendukung operasi komposisi atau konkatenasi pada bahasa formal.

BAB VII
EKUIVALENSI NON-DETERMINISTIC FA(NFA) KE DETERMINISTIC FA(DFA)

Materi tentang ekivalensi antara Non-Deterministic Finite Automata (NFA) dan Deterministic Finite Automata (DFA) menjelaskan bagaimana NFA dapat dikonversi menjadi DFA yang setara.

A. Ekivalensi NFA dan DFA:

NFA dan DFA adalah dua jenis mesin yang digunakan untuk mengenali bahasa-bahasa formal.Kedua mesin memiliki kemampuan yang setara dalam hal mengenali bahasa, artinya setiap bahasa yang dapat dikenali oleh NFA juga dapat dikenali oleh DFA, dan sebaliknya.Dengan demikian, NFA dan DFA secara teoritis setara dalam hal daya ekspresifitas.

B. Konversi NFA ke DFA:

NFA dapat dikonversi menjadi DFA yang setara untuk mencapai determinisme.Langkah-langkah umum dalam konversi NFA ke DFA adalah sebagai berikut:

- Buat DFA dengan himpunan keadaan yang baru.
- Tentukan keadaan awal DFA dengan menghitung ε-Closure dari keadaan awal NFA.
- Untuk setiap simbol input pada DFA:
- Hitung transisi untuk setiap keadaan pada DFA dengan menggabungkan semua transisi yang mungkin dari NFA.
- Hitung ε-Closure dari setiap keadaan pada DFA yang dihasilkan.
- Setiap himpunan keadaan pada DFA yang dihasilkan adalah keadaan pada DFA yang baru.
- Tentukan keadaan akhir DFA dengan memasukkan keadaan akhir dari NFA yang terkandung dalam himpunan keadaan pada DFA yang dihasilkan.

C. Tabel Transisi DFA:

Konversi NFA ke DFA melibatkan pembuatan tabel transisi baru.Tabel transisi DFA menyajikan semua kemungkinan transisi antara keadaan-keadaan pada DFA dengan simbol input.Setiap sel dalam tabel transisi menunjukkan keadaan yang dihasilkan ketika DFA berada pada keadaan tertentu dan membaca simbol input tertentu.

D. Keuntungan dan Keterbatasan DFA:

- DFA memiliki keuntungan dibandingkan NFA karena deterministik dalam operasinya.
- DFA memberikan hasil yang unik dan pasti untuk setiap string input.
- Namun, DFA memiliki keterbatasan dalam hal kompleksitas pengenalan bahasa-bahasa yang lebih rumit, seperti bahasa konteks-bebas atau bahasa tak terbatas.

BAB VIII
ε-MOVE & ε-CLOSURE PADA NON DETERMINISTIC FINITE AUTOMATA

Pada Non-Deterministic Finite Automata (NFA), ε-Move (ε-pindah) dan ε-Closure (ε-penutupan) adalah konsep penting yang berkaitan dengan transisi non-deterministik yang melibatkan simbol ε (epsilon).

A. ε-Move (ε-Pindah):

ε-Move adalah kemampuan NFA untuk melakukan transisi non-deterministik tanpa membaca simbol input (transisi kosong) dengan simbol ε.Saat ada ε-Move dari suatu keadaan, mesin dapat berpindah ke keadaan lain tanpa mengonsumsi simbol apa pun.ε-Move memungkinkan NFA untuk melakukan perpindahan yang tidak tergantung pada simbol input saat ini.

B. ε-Closure (ε-Penutupan):

ε-Closure dari suatu keadaan dalam NFA adalah himpunan semua keadaan yang dapat dicapai dari keadaan tersebut melalui satu atau lebih ε-Move.ε-Closure menggambarkan semua keadaan yang dapat diakses dari keadaan awal dengan mempertimbangkan ε-Move.ε-Closure juga mencakup keadaan awal itu sendiri.

C. Penerapan ε-Closure:

ε-Closure dapat digunakan untuk menghitung semua keadaan yang dapat dicapai dari satu keadaan dalam NFA dengan mempertimbangkan ε-Move.Biasanya, algoritma yang disebut "ε-Closure" atau "ε-Closure Construction" digunakan untuk menghitung ε-Closure dari setiap keadaan dalam NFA.

D. Signifikansi ε-Closure:

ε-Closure memainkan peran penting dalam pengenalan bahasa oleh NFA.ε-Closure memungkinkan NFA untuk mengenali jalur-jalur yang melibatkan transisi non-deterministik dengan ε-Move.Dengan menggunakan ε-Closure, NFA dapat menghitung semua keadaan yang dapat dicapai saat membaca simbol input tertentu.

E. Konversi NFA dengan ε-Move ke NFA Tanpa ε-Move:

- NFA dengan ε-Move dapat dikonversi menjadi NFA tanpa ε-Move atau DFA yang setara.
- Konversi ini melibatkan penggabungan ε-Closure dengan transisi yang sesuai pada DFA yang dihasilkan.

BAB IX
EKUIVALENSI NFA DENGAN ε-MOVE KE NFA TANPA ε-MOVE

Dalam konteks Non-Deterministic Finite Automata (NFA), ekivalensi antara NFA dengan ε-Move (ε-NFA) dan NFA tanpa ε-Move (NFA biasa) adalah konsep yang penting. Ekivalensi ini berarti bahwa keduanya memiliki kemampuan yang setara dalam mengenali bahasa-bahasa formal.

A. NFA dengan ε-Move (ε-NFA):

NFA dengan ε-Move adalah NFA yang memiliki kemampuan untuk melakukan transisi non-deterministik dengan simbol ε (epsilon), yang disebut ε-Move.ε-Move memungkinkan NFA untuk berpindah ke keadaan lain tanpa membaca simbol input apa pun.ε-NFA dapat mengenali bahasa-bahasa formal yang melibatkan perpindahan dengan ε-Move.

B. NFA tanpa ε-Move (NFA biasa):

NFA tanpa ε-Move adalah NFA yang tidak memiliki transisi dengan simbol ε (epsilon).Pada NFA biasa, transisi hanya terjadi ketika simbol input yang relevan dibaca.NFA biasa dapat mengenali bahasa-bahasa formal tanpa mempertimbangkan ε-Move.

C. Ekivalensi NFA dengan ε-Move dan NFA tanpa ε-Move:

NFA dengan ε-Move dan NFA tanpa ε-Move adalah dua representasi yang ekivalen dalam hal kemampuan pengenalan bahasa formal.Meskipun memiliki mekanisme yang berbeda dalam melakukan transisi, keduanya dapat mengenali bahasa yang sama.Setiap bahasa yang dapat dikenali oleh ε-NFA juga dapat dikenali oleh NFA biasa, dan sebaliknya.

D. Konversi ε-NFA ke NFA biasa:

ε-NFA dapat dikonversi ke NFA biasa atau DFA yang setara untuk mencapai determinisme.Konversi ini melibatkan menggabungkan ε-Closure dengan transisi yang sesuai pada NFA atau DFA yang dihasilkan.Langkah-langkah konversi termasuk menghilangkan ε-Move, menghitung ε-Closure, dan mengubah fungsi transisi ε-NFA menjadi fungsi transisi NFA biasa yang hanya beroperasi pada simbol input.

E. Keterbatasan NFA dengan ε-Move:

- NFA dengan ε-Move memiliki kekuatan ekspresif yang lebih besar dibandingkan dengan NFA biasa.
- Bahasa-bahasa dengan struktur yang kompleks, seperti bahasa konteks-bebas atau bahasa tak terbatas, mungkin memerlukan NFA dengan ε-Move untuk dikenali.

BAB X
FINITE STATE AUTOMATA TRANSDUCER(MOORE MACHINE)

Finite State Automata Transducer, juga dikenal sebagai Moore Machine, adalah model komputasi yang merupakan perluasan dari Finite State Automata (FSA) dengan kemampuan menghasilkan output selain melakukan transisi ke keadaan lain.

A. Definisi Moore Machine:

Moore Machine adalah model FSA yang memiliki keluaran (output) yang terkait dengan setiap keadaan.Setiap keadaan pada Moore Machine memiliki keluaran tertentu yang dikaitkan dengannya.Keluaran dihasilkan berdasarkan keadaan saat ini tanpa mempertimbangkan simbol input yang sedang dibaca.

B. Komponen Moore Machine:

- Input: Moore Machine membaca simbol-simbol input satu per satu.
- State: Moore Machine memiliki himpunan keadaan yang mungkin, dan mesin berada dalam satu keadaan pada suatu waktu.
- Output: Setiap keadaan pada Moore Machine memiliki keluaran yang terkait dengannya. Output dihasilkan saat mesin berada di keadaan tersebut.
- Transition Function: Moore Machine memiliki fungsi transisi yang menggambarkan perpindahan dari satu keadaan ke keadaan lain berdasarkan simbol input.
- Output Function: Moore Machine memiliki fungsi keluaran yang mengaitkan keluaran dengan setiap keadaan.

C. Representasi Moore Machine:

Moore Machine dapat direpresentasikan menggunakan diagram keadaan atau tabel transisi.Diagram keadaan Moore Machine menunjukkan keadaan-keadaan sebagai lingkaran dan transisi di antara mereka sebagai panah dengan simbol input yang relevan.Tabel transisi Moore Machine menggambarkan keadaan awal, transisi, simbol input, dan keluaran yang terkait dengan setiap keadaan.

D. Penggunaan Moore Machine:

Moore Machine digunakan dalam berbagai aplikasi, termasuk pengenalan pola, pengolahan sinyal, pengenalan suara, dan bahasa alami.

Moore Machine juga berguna dalam komunikasi antara sistem dan lingkungan eksternal, di mana keluaran dapat digunakan untuk memberikan respons atau tindakan.

E. Perbedaan dengan Mealy Machine:

Moore Machine berbeda dengan Mealy Machine, yang menghasilkan keluaran berdasarkan transisi yang terjadi dan simbol input yang sedang dibaca.

Pada Moore Machine, keluaran hanya bergantung pada keadaan saat ini, sedangkan pada Mealy Machine, keluaran bergantung pada transisi dan simbol input.

BAB XI
POHON PENURUNAN, PARSING, AMBIGUITAS

Pohon Penurunan (Parse Tree) adalah struktur pohon yang menggambarkan urutan produksi yang digunakan dalam parsing sebuah string pada tata bahasa tertentu. Pohon penurunan digunakan dalam analisis sintaksis (parsing) untuk memvisualisasikan bagaimana string input diproduksi menggunakan aturan produksi dari tata bahasa. Berikut ini adalah materi tentang Pohon Penurunan, Parsing, dan Ambiguitas:

A. Pohon Penurunan:

Pohon penurunan adalah representasi visual dari urutan aturan produksi yang digunakan dalam parsing suatu string.Pohon penurunan menggambarkan bagaimana aturan produksi diaplikasikan untuk mengubah simbol-simbol non-terminal menjadi string terminal.

Setiap node dalam pohon penurunan mewakili satu simbol non-terminal atau terminal, sedangkan cabang-cabang dari setiap node menunjukkan penggunaan aturan produksi.

B. Parsing:

Parsing adalah proses analisis sintaksis yang menggunakan tata bahasa formal untuk memeriksa kesesuaian struktur string input dengan tata bahasa yang didefinisikan.Parsing dapat dilakukan menggunakan berbagai metode, seperti parsing top-down (mulai dari simbol awal dan melihat ke simbol terminal) atau parsing bottom-up (mulai dari simbol terminal dan membangun struktur hingga mencapai simbol awal).Saat parsing dilakukan, pohon penurunan dibangun untuk menggambarkan urutan aturan produksi yang digunakan.

C. Ambiguitas:

Ambiguitas terjadi ketika sebuah tata bahasa memiliki lebih dari satu pohon penurunan yang valid untuk suatu string input.Ambiguitas dapat menyebabkan ketidakjelasan dalam arti dari string input dan mempersulit proses parsing.

Ambiguitas dapat terjadi karena beberapa aturan produksi yang bersifat ambigu atau karena kurangnya informasi dalam tata bahasa.

D. Mengatasi Ambiguitas:

Ambiguitas dalam parsing dapat diatasi dengan melakukan pemrosesan tambahan, seperti menambahkan aturan produksi tambahan, mengubah urutan aturan produksi, atau menggunakan tata bahasa yang lebih spesifik.Beberapa metode parsing, seperti parsing LL(1) dan parsing LR(1), dirancang untuk mengatasi ambiguitas dan memastikan parsing yang unik.

Pohon penurunan, parsing, dan ambiguitas merupakan konsep penting dalam analisis sintaksis dan memahami struktur tata bahasa. Pohon penurunan memberikan visualisasi yang jelas tentang bagaimana string input dihasilkan menggunakan aturan produksi.

BAB XII
PENGANTAR ATURAN PRODUKSI REKURSIF KIRI

Aturan Produksi Rekursif Kiri adalah konsep dalam teori bahasa dan automata yang digunakan untuk menggambarkan tata bahasa rekursif kiri. Aturan produksi rekursif kiri adalah aturan produksi di mana simbol non-terminal pada sisi kiri dapat menghasilkan derivasi yang tidak berkurang dalam satu langkah. Berikut adalah materi pengantar tentang aturan produksi rekursif kiri:

A. Definisi Aturan Produksi Rekursif Kiri:

Aturan produksi rekursif kiri adalah aturan produksi di mana simbol non-terminal pada sisi kiri dapat langsung menghasilkan simbol non-terminal yang sama atau derivasi lain yang berawal dengan simbol non-terminal yang sama.

Misalnya, jika terdapat aturan produksi $A \rightarrow A\alpha$, di mana A adalah simbol non-terminal dan α adalah deretan simbol non-terminal dan/atau terminal, maka aturan produksi tersebut rekursif kiri.

B. Contoh Aturan Produksi Rekursif Kiri:

- Berikut adalah contoh aturan produksi rekursif kiri dalam tata bahasa aritmatika sederhana:
  - $E \rightarrow E + T$
  - $E \rightarrow E - T$
  - $E \rightarrow T$
- Pada contoh di atas, aturan produksi $E \rightarrow E + T$ dan $E \rightarrow E - T$ adalah contoh aturan produksi rekursif kiri karena simbol non-terminal E pada sisi kiri dapat menghasilkan derivasi yang berawal dengan simbol non-terminal E.

C. Penggunaan Aturan Produksi Rekursif Kiri:

Aturan produksi rekursif kiri digunakan untuk mendefinisikan bahasa yang memiliki struktur rekursif kiri.Dalam analisis sintaksis, aturan produksi rekursif kiri digunakan dalam metode parsing seperti Recursive Descent Parsing yang sesuai dengan struktur rekursif kiri.

D. Konversi Aturan Produksi Rekursif Kiri:

Aturan produksi rekursif kiri dapat diubah menjadi aturan produksi rekursif kanan atau aturan produksi tidak rekursif melalui teknik penggantian atau faktorisasi.

Konversi aturan produksi rekursif kiri menjadi rekursif kanan atau aturan produksi tidak rekursif diperlukan dalam beberapa metode parsing yang membutuhkan aturan produksi tidak rekursif atau rekursif kanan.

Aturan Produksi Rekursif Kiri adalah konsep penting dalam teori bahasa dan automata, terutama dalam analisis sintaksis. Memahami aturan produksi rekursif kiri membantu dalam pemodelan tata bahasa rekursif kiri dan pemilihan metode parsing yang sesuai.

BAB XIII
TEKNIK PENGHILANGAN REKURSIF KIRI

Teknik penghilangan rekursif kiri adalah metode yang digunakan dalam teori bahasa dan automata untuk mengubah aturan produksi rekursif kiri menjadi aturan produksi rekursif kanan atau aturan produksi tidak rekursif. Tujuan dari teknik ini adalah untuk memastikan bahwa tata bahasa tidak mengandung rekursi kiri yang dapat mengakibatkan masalah dalam analisis sintaksis.

A. Definisi Rekursi Kiri:

Rekursi kiri terjadi ketika suatu simbol non-terminal pada sisi kiri aturan produksi dapat langsung menghasilkan derivasi yang berawal dengan simbol non-terminal yang sama.

Misalnya, jika terdapat aturan produksi A → Aα, maka aturan produksi tersebut rekursi kiri karena simbol non-terminal A pada sisi kiri menghasilkan derivasi yang dimulai dengan A.

B. Langkah-langkah Penghilangan Rekursif Kiri:

- Terdapat beberapa langkah yang dapat diikuti untuk menghilangkan rekursi kiri dalam tata bahasa:
  - Identifikasi aturan produksi yang rekursif kiri.
  - Faktorisasi aturan produksi rekursif kiri dengan memisahkan bagian yang mengandung rekursi kiri.
  - Buat simbol non-terminal baru untuk mewakili langkah yang diambil dalam faktorisasi.
  - Ganti aturan produksi rekursif kiri dengan aturan produksi rekursif kanan atau aturan produksi tidak rekursif menggunakan simbol non-terminal baru yang dibuat.

C. Contoh Penghilangan Rekursif Kiri:

- Misalkan terdapat aturan produksi berikut:
  - A → Aα1 | Aα2 | ... | Aαn | β1 | β2 | ... | βm
- Untuk menghilangkan rekursi kiri, langkah-langkah berikut dapat diikuti: a. Faktorisasi aturan produksi menjadi:
  - A → β1A' | β2A' | ... | βmA'
  - A' → α1A' | α2A' | ... | αnA' | ε b. Aturan produksi A' → ε ditambahkan untuk menangani kasus ketika derivasi berakhir.

D. Pentingnya Penghilangan Rekursi Kiri:

Penghilangan rekursi kiri penting dalam analisis sintaksis karena beberapa metode parsing, seperti Recursive Descent Parsing, membutuhkan aturan produksi tidak rekursif atau rekursif kanan.

Rekursi kiri dapat menyebabkan masalah seperti looping tak berhingga dalam parsing jika tidak dihilangkan.

Teknik penghilangan rekursi kiri adalah langkah penting dalam memodelkan tata bahasa yang dapat dianalisis secara sintaksis dengan benar. Dengan mengidentifikasi aturan produksi yang rekursif kiri dan mengikuti langkah-langkah penghilangan yang tepat, rekursi kiri dapat diatasi untuk memastikan analisis sintaksis yang efektif.

BAB XIV
CONTOH KASUS PENGHILANGAN REKURSIF KIRI

Mari kita lihat contoh kasus penghilangan rekursi kiri menggunakan langkah-langkah yang telah disebutkan sebelumnya. Misalkan kita memiliki tata bahasa sederhana dengan simbol non-terminal A yang menghasilkan aturan produksi rekursif kiri berikut:

1. A → Aα1 | Aα2 | ... | Aαn | β1 | β2 | ... | βm

   Langkah-langkah penghilangan rekursi kiri adalah sebagai berikut:

2. Faktorisasi aturan produksi: A → β1A' | β2A' | ... | βmA' A' → α1A' | α2A' | ... | αnA' | ε

   Dalam langkah ini, kita membuat simbol non-terminal baru A' untuk mewakili langkah-langkah yang diambil dalam faktorisasi.

3. Perhatikan bahwa pada langkah faktorisasi di atas, terdapat ε pada aturan produksi A'. Ini ditambahkan untuk menangani kasus ketika derivasi berakhir. Jika tidak ada ε dalam tata bahasa, maka langkah ini tidak diperlukan.

Setelah langkah-langkah penghilangan rekursi kiri selesai, aturan produksi rekursif kiri telah diubah menjadi aturan produksi tidak rekursif atau rekursif kanan. Dalam contoh ini, aturan produksi menjadi:

A → β1A' | β2A' | ... | βmA' A' → α1A' | α2A' | ... | αnA' | ε

Dengan langkah-langkah ini, rekursi kiri dalam tata bahasa telah dihilangkan, dan tata bahasa tersebut siap untuk digunakan dalam analisis sintaksis yang membutuhkan aturan produksi tidak rekursif atau rekursif kanan.

BAB XV
PENGANTAR PENYEDERHANAAN ATURAN PRODUKSI CONTEXT FREE GRAMMAR

Penyederhanaan aturan produksi dalam Konteks Bebas adalah proses mengurangi kompleksitas tata bahasa dengan menghilangkan aturan produksi yang tidak perlu atau redundan. Tujuan dari penyederhanaan adalah untuk menghasilkan tata bahasa yang lebih sederhana dan mudah dipahami tanpa mengubah bahasa yang dihasilkan oleh tata bahasa awal

A. Definisi Aturan Produksi Context-Free Grammar (CFG):

Aturan produksi dalam CFG mengikuti format A → α, di mana A adalah simbol non-terminal dan α adalah deretan simbol non-terminal dan/atau terminal.Aturan produksi menentukan bagaimana simbol-simbol dalam bahasa diterjemahkan atau dihasilkan.

B. Tujuan Penyederhanaan Aturan Produksi CFG:

Memperbaiki kualitas tata bahasa dengan menghilangkan aturan produksi yang tidak perlu atau redundan.Mengurangi kompleksitas dan ukuran tata bahasa untuk membuatnya lebih mudah dipahami dan dianalisis.

C. Teknik Penyederhanaan Aturan Produksi CFG:

- Faktorisasi: Jika ada aturan produksi yang memiliki prefiks umum, faktorisasi dapat digunakan untuk menggabungkan prefiks tersebut menjadi satu aturan produksi baru.
- Penggantian: Jika ada aturan produksi yang tidak efisien atau tidak diperlukan, dapat diganti dengan aturan produksi baru yang lebih sederhana atau langsung dihilangkan.
- Penghapusan Simbol Non-Terminal Tidak Terpakai: Jika ada simbol non-terminal yang tidak pernah terlibat dalam derivasi string terminal, maka simbol non-terminal tersebut dapat dihapus.

D. Contoh Penyederhanaan Aturan Produksi CFG: Misalkan kita memiliki tata bahasa berikut dengan aturan produksi yang kompleks:

- S → aSb | ε
- S → aA | aB
- A → aS
- B → bS

Dalam kasus ini, aturan produksi bisa disederhanakan dengan menggunakan teknik penyederhanaan:

- Faktorisasi aturan produksi S → aSb | ε menjadi:
  - S → aSB'
  - S → ε
  - B' → Sb

Setelah penyederhanaan, tata bahasa menjadi lebih sederhana dan lebih mudah dipahami:

- S → aSB' | ε
- B' → Sb
- S → aA | aB
- A → aS
- B → bS

Penting untuk melakukan penyederhanaan aturan produksi dalam CFG guna memperbaiki kualitas dan kompleksitas tata bahasa. Dengan menerapkan teknik penyederhanaan, tata bahasa menjadi lebih sederhana, mudah dipahami, dan lebih efisien dalam proses analisis sintaksis.

BAB XVI
PENGHILANGAN PRODUKSI KOSONG ATURAN PRODUKSI CONTEXT FREE GRAMMAR

Penghilangan produksi kosong adalah salah satu tahap dalam penyederhanaan Context-Free Grammar (CFG) di mana aturan produksi yang menghasilkan string kosong (ε) dihapus atau diubah menjadi bentuk yang lebih sederhana. Penghilangan produksi kosong penting dalam analisis sintaksis karena produksi kosong dapat mempengaruhi proses parsing dan menyebabkan ambiguitas dalam tata bahasa

A. Definisi Produksi Kosong:

Produksi kosong (ε-produksi) adalah aturan produksi yang menghasilkan string kosong (ε).Misalnya, jika terdapat aturan produksi A → ε, maka aturan produksi tersebut adalah produksi kosong.

B. Tujuan Penghilangan Produksi Kosong:

Menghilangkan produksi kosong untuk mengurangi kompleksitas dan ambiguitas dalam tata bahasa.Membuat CFG lebih efisien dalam analisis sintaksis dan memperbaiki kualitas tata bahasa.

C. Langkah-langkah Penghilangan Produksi Kosong:

Terdapat beberapa langkah yang dapat diikuti untuk menghilangkan produksi kosong dalam CFG:

- Identifikasi aturan produksi yang menghasilkan string kosong.
- Buat himpunan semua simbol non-terminal yang menghasilkan string kosong secara langsung atau tidak langsung.
- Dalam setiap aturan produksi yang mengandung simbol non-terminal dari himpunan tersebut, hilangkan simbol non-terminal tersebut atau gantikan dengan kombinasi simbol non-terminal lain.
- Ulangi langkah b dan c hingga tidak ada lagi produksi kosong dalam CFG.

D. Contoh Penghilangan Produksi Kosong: Misalkan kita memiliki tata bahasa berikut dengan aturan produksi yang menghasilkan produksi kosong:

- S → ε
- S → A
- A → aB
- B → bC
- C → ε

Dalam kasus ini, aturan produksi yang menghasilkan produksi kosong adalah S → ε dan C → ε. Langkah-langkah penghilangan produksi kosong adalah sebagai berikut:

a) Identifikasi aturan produksi yang menghasilkan string kosong: S → ε dan C → ε.
b) Buat himpunan simbol non-terminal yang menghasilkan string kosong: {S, C}.
c) Hilangkan produksi kosong dari aturan produksi yang mengandung simbol non-terminal dari himpunan tersebut:
    - S → A
    - A → aB
    - B → b

Setelah penyederhanaan, tata bahasa menjadi lebih sederhana dan tidak mengandung produksi kosong:

- S → A
- A → aB
- B → b

Penghilangan produksi kosong adalah langkah penting dalam penyederhanaan CFG. Dengan menghilangkan produksi kosong, tata bahasa menjadi lebih sederhana dan lebih mudah dipahami. Hal ini juga membantu dalam analisis sintaksis yang efisien dan menghindari ambiguitas yang dapat terjadi karena produksi kosong.

BAB XVII
PENGHILANGAN PRODUKSI UNIT ATURAN PRODUKSI CONTEXT FREE GRAMMAR

Penghilangan produksi unit adalah proses dalam penyederhanaan Context-Free Grammar (CFG) di mana aturan produksi yang hanya mengandung satu simbol non-terminal di sisi kanan aturan produksi dihapus atau digantikan dengan aturan produksi yang lebih sederhana. Penghilangan produksi unit bertujuan untuk mengurangi kompleksitas dan ambiguitas dalam tata bahasa.

A. Definisi Produksi Unit:

Produksi unit adalah aturan produksi dalam CFG di mana hanya terdapat satu simbol non-terminal di sisi kanan aturan produksi.

Misalnya, jika terdapat aturan produksi A → B, maka aturan produksi tersebut adalah produksi unit.

B. Tujuan Penghilangan Produksi Unit:

Menghilangkan produksi unit untuk mengurangi kompleksitas dan ambiguitas dalam tata bahasa.Membuat CFG lebih efisien dalam analisis sintaksis dan memperbaiki kualitas tata bahasa.

C. Langkah-langkah Penghilangan Produksi Unit:

Terdapat beberapa langkah yang dapat diikuti untuk menghilangkan produksi unit dalam CFG:

- Identifikasi aturan produksi yang merupakan produksi unit.
- Untuk setiap aturan produksi yang merupakan produksi unit A → B, tambahkan semua aturan produksi B → α ke aturan produksi A, kecuali jika B → α adalah produksi unit.
- Ulangi langkah b hingga tidak ada lagi produksi unit dalam CFG.

D. Contoh Penghilangan Produksi Unit: Misalkan kita memiliki tata bahasa berikut dengan aturan produksi yang mengandung produksi unit:

- S → A
- A → B
- B → C
- C → a

Dalam kasus ini, aturan produksi yang merupakan produksi unit adalah S → A, A → B, dan B → C. Langkah-langkah penghilangan produksi unit adalah sebagai berikut:

a) Identifikasi aturan produksi yang merupakan produksi unit: S → A, A → B, dan B → C.

b) Gantikan aturan produksi yang merupakan produksi unit dengan aturan produksi yang sesuai:

- S → C
- S → a
- A → C
- A → a
- B → a
- C → a

Setelah penyederhanaan, tata bahasa menjadi lebih sederhana dan tidak mengandung produksi unit:

- S → C | a
- A → C | a
- B → a
- C → a

Penghilangan produksi unit adalah langkah penting dalam penyederhanaan CFG. Dengan menghilangkan produksi unit, tata bahasa menjadi lebih sederhana dan tidak ambigu. Hal ini juga membantu dalam analisis sintaksis yang efisien dan memperbaiki kualitas tata bahasa.

BAB XVIII
PENGHILANGAN PRODUKSI USELESS PADA ATURAN PRODUKSI CONTEXT FREE GRAMMAR

Penghilangan produksi yang tidak berguna atau produksi yang tidak digunakan (useless production) adalah langkah dalam penyederhanaan Context-Free Grammar (CFG) di mana aturan produksi yang tidak berkontribusi pada pembentukan string terminal yang valid dihapus. Langkah ini bertujuan untuk mengurangi kompleksitas dan memperbaiki kualitas CFG dengan menghapus aturan produksi yang tidak diperlukan.

A. Definisi Produksi Useless:

Produksi yang tidak berguna adalah aturan produksi dalam CFG yang tidak berkontribusi pada pembentukan string terminal yang valid.

Misalnya, jika terdapat aturan produksi $A \rightarrow \varepsilon$ dan simbol non-terminal A tidak pernah muncul di sisi kiri aturan produksi lainnya, maka aturan produksi $A \rightarrow \varepsilon$ adalah produksi yang tidak berguna.

B. Tujuan Penghilangan Produksi Useless:

Menghilangkan produksi yang tidak berguna untuk mengurangi kompleksitas dan ambiguitas dalam tata bahasa.Membuat CFG lebih efisien dalam analisis sintaksis dan memperbaiki kualitas tata bahasa.

C. Langkah-langkah Penghilangan Produksi Useless:

Terdapat beberapa langkah yang dapat diikuti untuk menghilangkan produksi yang tidak berguna dalam CFG:

- Identifikasi simbol non-terminal yang dapat mencapai string terminal secara langsung atau tidak langsung melalui derivasi.
- Identifikasi simbol non-terminal yang dapat mencapai akhiran (string terminal) langsung atau tidak langsung melalui derivasi.
- Hapus aturan produksi yang mengandung simbol non-terminal yang tidak mencapai string terminal atau akhiran.
- Ulangi langkah a, b, dan c hingga tidak ada lagi produksi yang tidak berguna dalam CFG.

D. Contoh Penghilangan Produksi Useless: Misalkan kita memiliki tata bahasa berikut dengan aturan produksi yang mengandung produksi yang tidak berguna:

- $S \rightarrow A$
- $A \rightarrow aB$
- $B \rightarrow \varepsilon$
- $C \rightarrow b$

Dalam kasus ini, aturan produksi yang tidak berguna adalah B → ε dan C → b. Langkah-langkah penghilangan produksi yang tidak berguna adalah sebagai berikut:

a) Identifikasi simbol non-terminal yang dapat mencapai string terminal secara langsung atau tidak langsung melalui derivasi: S, A, B.
b) Identifikasi simbol non-terminal yang dapat mencapai akhiran langsung atau tidak langsung melalui derivasi: A.
c) Hapus aturan produksi yang mengandung simbol non-terminal yang tidak mencapai string terminal atau akhiran: B → ε.
d) Setelah penghilangan produksi yang tidak berguna, tata bahasa menjadi:
    - S → A
    - A → aB
    - C → b

Setelah penyederhanaan, tata bahasa menjadi lebih sederhana dan tidak mengandung produksi yang tidak berguna:

- S → A
- A → aB
- C → b

Penghilangan produksi yang tidak berguna adalah langkah penting dalam penyederhanaan CFG. Dengan menghilangkan produksi yang tidak berguna, tata bahasa menjadi lebih sederhana dan lebih mudah dipahami. Hal ini juga membantu dalam analisis sintaksis yang efisien dan menghindari ambiguitas yang dapat terjadi karena produksi yang tidak berguna.

BAB XIX
CONTOH PENYEDERHANAAN ATURAN PRODUKSI CONTEXT FREE GRAMMAR

Berikut ini adalah contoh penyederhanaan aturan produksi Context-Free Grammar (CFG):

Misalkan kita memiliki CFG dengan aturan produksi sebagai berikut:

- S -> aSb
- S -> ε
- S -> SSS
- S -> AB
- A -> aa
- B -> b

Langkah-langkah penyederhanaan aturan produksi CFG:

1. Langkah 1: Penghilangan Produksi Kosong Dalam CFG kita memiliki produksi kosong S -> ε. Kita dapat menghapus produksi ini dan mengubah aturan produksi lain yang mengandung S menjadi S -> aSb | ab.

   Setelah penyederhanaan, aturan produksi menjadi:

   - S -> aSb | ab
   - S -> SSS
   - S -> AB
   - A -> aa
   - B -> b

2. Langkah 2: Penghilangan Produksi Unit Dalam CFG, kita memiliki beberapa produksi unit seperti S -> AB, A -> aa, dan B -> b. Kita dapat menggantikan produksi unit ini dengan produksi yang sesuai.

   Setelah penghilangan produksi unit, aturan produksi menjadi:

   - S -> aSb | ab
   - S -> SSS
   - A -> aa

- B -> b

3. Langkah 3: Penghilangan Produksi Yang Tidak Berguna Dalam CFG, kita melihat bahwa produksi S -> SSS tidak berguna karena tidak ada cara untuk menghasilkan string terminal dari produksi ini. Kita dapat menghapus produksi ini.

   Setelah penghilangan produksi yang tidak berguna, aturan produksi menjadi:

   - S -> aSb | ab
   - A -> aa
   - B -> b

Setelah melakukan langkah-langkah penyederhanaan, CFG menjadi lebih sederhana dengan aturan produksi yang lebih padat.

BAB XX
PENGANTAR CHOMSKY NORMAL FORM(CNF)

Chomsky Normal Form (CNF) adalah bentuk normal atau bentuk standar untuk Context-Free Grammar (CFG). CNF memiliki aturan produksi yang terbatas dan terstruktur dengan cara tertentu. Penggunaan CNF dalam CFG membantu dalam analisis sintaksis dan algoritma parsing, serta mempermudah pemahaman dan manipulasi struktur bahasa.

A. Definisi CNF:

CNF adalah bentuk normal untuk CFG di mana setiap aturan produksi memiliki bentuk yang terbatas.Setiap aturan produksi dalam CNF adalah salah satu dari dua bentuk berikut:

- Non-terminal -> Terminal Terminal (bentuk produksi terminal).
- Non-terminal -> Non-terminal Non-terminal (bentuk produksi non-terminal).

B. Aturan CNF:

Setiap aturan produksi dalam CNF harus mematuhi aturan berikut:

- Aturan produksi ε (epsilon) dilarang dalam CNF.
- Non-terminal yang tidak digunakan dalam aturan produksi tidak diperbolehkan.
- Satu-satunya simbol non-terminal yang dapat menghasilkan ε adalah simbol awal (start symbol).
- Non-terminal tidak boleh menghasilkan langsung ke simbol terminal.

C. Tujuan CNF:

Menggunakan CNF membantu dalam analisis sintaksis dan algoritma parsing, seperti CYK (Cocke-Younger-Kasami) parsing.CNF mempermudah pemahaman dan manipulasi struktur bahasa.CNF menghilangkan ambiguitas dalam CFG dan memungkinkan analisis yang lebih efisien.

D. Langkah-langkah mengubah CFG menjadi CNF:

Untuk mengubah CFG menjadi CNF, kita dapat mengikuti langkah-langkah berikut:

- Hilangkan produksi ε (jika ada) dengan memperhatikan aturan CNF.
- Hilangkan produksi unit dengan menggunakan aturan CNF.
- Konversi produksi yang mengandung lebih dari dua simbol non-terminal menjadi produksi dengan dua simbol non-terminal menggunakan simbol baru.
- Jika ada produksi yang mengandung terminal, gantikan terminal dengan simbol non-terminal yang sesuai menggunakan simbol baru.

Chomsky Normal Form (CNF) adalah bentuk normal yang berguna dalam pemrosesan bahasa alami, analisis sintaksis, dan algoritma parsing.

BAB XXI
PEMBENTUKAN CHOMSKY NORMAL FORM DARI CONTEXT FREE GRAMMAR

Untuk mengubah sebuah Context-Free Grammar (CFG) menjadi Chomsky Normal Form (CNF), kita perlu mengikuti serangkaian langkah-langkah. Berikut adalah langkah-langkah umum yang dapat diikuti untuk membentuk CNF dari CFG:

1. Langkah 1: Hilangkan produksi ε (epsilon):
    - Identifikasi dan hapus semua aturan produksi yang menghasilkan ε (epsilon).
    - Jika ada aturan produksi yang mengandung simbol non-terminal di sisi kanan dan menghasilkan ε (epsilon), gantilah dengan semua kombinasi dari simbol non-terminal tersebut tanpa ε (epsilon).
2. Langkah 2: Hilangkan produksi unit:
    - Identifikasi dan hapus semua aturan produksi unit, yaitu aturan produksi di mana hanya ada satu simbol non-terminal di sisi kanan.
    - Jika terdapat aturan produksi A -> B, di mana A dan B adalah simbol non-terminal, gantilah setiap kemunculan A di aturan produksi lain dengan B.
3. Langkah 3: Konversi produksi dengan lebih dari dua simbol non-terminal:
    - Identifikasi setiap aturan produksi yang memiliki lebih dari dua simbol non-terminal di sisi kanan.
    - Untuk setiap aturan produksi yang memiliki X1X2...Xn (n > 2) di sisi kanan, buatlah aturan produksi baru dengan menggunakan simbol non-terminal baru Z.
    - Bagi aturan produksi menjadi aturan produksi berturut-turut dengan hanya dua simbol non-terminal, misalnya X1X2...Xn = X1Y1, Y1X2Y2, Y2X3Y3, ..., Y(n-2)X(n-1)Y(n-1), Y(n-1)Xn.
4. Langkah 4: Gantikan terminal dengan simbol non-terminal:
    - Jika ada aturan produksi yang memiliki terminal (simbol dari alfabet) di sisi kanan, gantilah terminal tersebut dengan simbol non-terminal baru.
    - Buatlah aturan produksi baru untuk memetakan simbol non-terminal baru ke terminal yang sebelumnya ada dalam aturan produksi.

Setelah mengikuti langkah-langkah di atas, CFG akan berada dalam bentuk Chomsky Normal Form (CNF) yang memenuhi aturan-aturan CNF. Penting untuk dicatat bahwa langkah-langkah ini tidak mengubah bahasa yang diterima oleh CFG, tetapi hanya mengubah representasi CFG menjadi bentuk normal yang lebih terstruktur dan memudahkan analisis sintaksis serta pemrosesan bahasa.

BAB XXII
ALGORITMA COCKE-YOUNGER-KASAMI(CFG-CNF)

Algoritma Cocke-Younger-Kasami (CYK) adalah algoritma parsing yang digunakan untuk memeriksa apakah suatu string dapat dihasilkan oleh sebuah Context-Free Grammar (CFG) yang telah diubah menjadi Chomsky Normal Form (CNF). Algoritma ini bekerja dengan membangun tabel parse dan melakukan pencarian bottom-up menggunakan dynamic programming. Berikut adalah langkah-langkah algoritma CYK:

1. Langkah 1: Persiapan
    - Ubah CFG menjadi CNF menggunakan langkah-langkah yang dijelaskan sebelumnya.
    - Ambil string input yang akan di-parse.
2. Langkah 2: Inisialisasi Tabel Parse
    - Buat tabel parse berukuran N x N, di mana N adalah panjang string input.
    - Setiap sel (i, j) dalam tabel parse akan berisi himpunan non-terminal yang dapat menghasilkan substring dari i sampai j dalam string input.
3. Langkah 3: Pengisian Tabel Parse
    - Untuk setiap panjang substring mulai dari 1 hingga N: a. Untuk setiap posisi mulai i dari 1 hingga N - panjang + 1: b. Inisialisasi himpunan non-terminal kosong untuk sel (i, i + panjang - 1) dalam tabel parse. c. Untuk setiap pemisahan substring yang mungkin pada posisi p dari i hingga i + panjang - 2:
        - Ambil himpunan non-terminal dari sel (i, p) dan (p + 1, i + panjang - 1) dalam tabel parse.
        - Untuk setiap aturan produksi X -> YZ dalam CNF, di mana Y ada di himpunan non-terminal (i, p) dan Z ada di himpunan non-terminal (p + 1, i + panjang - 1), tambahkan X ke himpunan non-terminal (i, i + panjang - 1) dalam tabel parse.
4. Langkah 4: Pemeriksaan String
    - Periksa apakah simbol awal (start symbol) ada di himpunan non-terminal (1, N) dalam tabel parse.
    - Jika simbol awal ada di himpunan non-terminal (1, N), berarti string dapat dihasilkan oleh CFG.

Algoritma CYK akan menghasilkan tabel parse yang memungkinkan kita untuk melihat himpunan non-terminal yang dapat menghasilkan setiap substring dalam string input. Jika simbol awal (start symbol) ada di himpunan non-terminal untuk seluruh string, maka string dianggap dapat dihasilkan oleh CFG. Jika simbol awal tidak ada di himpunan non-terminal (1, N), maka string tidak dapat dihasilkan oleh CFG.

Algoritma CYK menggunakan pendekatan bottom-up dan dynamic programming, sehingga efisien untuk parsing bahasa yang dihasilkan oleh CFG dalam bentuk Chomsky Normal Form (CNF).