Nadar Saraswathi College of Engineering & Technology

NSCET E-LEARNING PRESENTATION

LISTEN … LEARN… LEAD…

COMPUTER SCIENCE AND ENGINEERING

III YEAR / V SEMESTER

CS8501 – Theory of Computation

P.MAHALAKSHMI,M.E,MISTE

ASSISTANT PROFESSOR

Nadar Saraswathi College of Engineering & Technology,

Vadapudupatti, Annanji (po), Theni – 625531.

UNIT III

Context Free Grammar and Languages

Context Free Grammar (CFG)

The syntax of the programming language constructs can be illustrated by context free grammars or Backus Naur Form (BNF)

Definition

- ✓ Context free grammar has large set of classes.
- ✓ It is a recursive notation for defining a language.
- ✓ Context free grammar G is defined as

G = (V,T,P,S)

V – Finite set of variables / Non-terminals and each variable represents a language. Capital letters are used to denote the non terminals.

T – Finite set of symbols that form the strings of the language terminals. Lower case letters, digits, operator and punctuations are used to denote the terminals.

P – Finite set of production symbols or rules that represent recursive definition of a language

Ex: X -> abS | ba (or) X -> abS

X -> ba

S – Starting symbol and it represents the variable. (X)

Language of a grammar

Let G=(V, T, P, S) is a CFG. Language of G is denoted as L(G), which is the set of terminal strings that have been derived from the start symbol.

$$L(G) = \{w \text{ in } T \mid S \stackrel{*}{\Rightarrow} w\}$$

Where w is the input string, T is the terminal and S is the starting symbol.

Context Free Language

If language L is the language of some CFG, then L is said to be CFL.

Example 1:

Construct the CFG for the language having any number of a's over the set ∑={a}

Solution **aaa**

L = { ε , a, aa, aaa, aaaa,…….} A -> ε A -> aA

Production is A -> aA

S -> ε (or) S -> aS | ε -> aaA

S -> aS -> aaaA

->aaa ε

-> aaa

Example 2:

Construct the CFG for the regular expression (0+1)*

Solution

L = { ε, 0,1,01,10,100,001,10010,….}

Production is A-> 0A | 1A | ε

S ->0S | 1S | ε

V ={ S}

T = {0, 1, ε }

S = {S}

Example 3:

Try to recognize the language L for given CFG

G = [{S}, {a,b}, P, {S}]

Where P = S -> aSb

S -> ab

Solution

L={ab, aabb, aaabbb,……}

L ={ set of strings with equal number of a's and b's but a followed by b over the set ∑= {a,b}}

L= {$a^n b^n$ | where n≥1}

100

A -> 1A

-> 10A

-> 100A

-> 100 ε

S -> aSb -> a aSb b -> a a ab b b

Topic

Parse Tree

Derivation

The production rules are used to derive certain string from starting symbol.

It expands the starting symbol using one of its productions by replacing one of variables by the body of one of its production, until the input string is derived.

Derivation is represented by the symbol ⇒

Derivation Types

There are two types

i**) Leftmost Derivation**

To derive the input string, each step replaces the leftmost variable by one of its production.

ii) Rightmost Derivation

To derive the input string, each step replaces the rightmost variable by one of its production.

Parse Tree

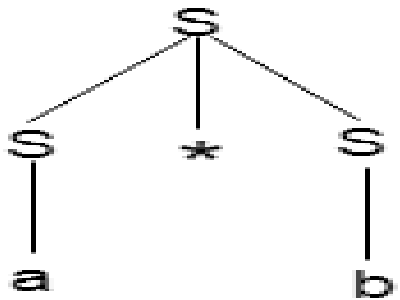
- ✓ Parse tree is a graphical representation for the derivation of the given production rules for a given CFG.
- ✓ It is the simplest way to show how the derivation can be done to obtain some string from given set of production rules.
- ✓ The parse tree is also called as derivation tree.

Properties of Parse tree

- ✓ The root node is always a node indicating start symbol
- ✓ The derivation is read from left to right
- ✓ The leaf nodes are always terminal nodes.
- ✓ The interior nodes are always the non-terminal nodes.

Example

S -> S * S | S + S
S -> a | b
w = a * b
S-> S * S
-> a * b



Example 1:

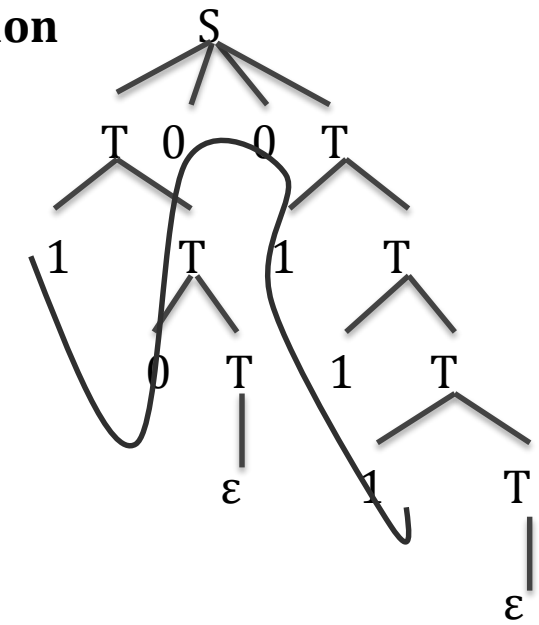
Derive the string 1000111 for leftmost and rightmost derivation and also construct parse tree using CFG.

G=(V,T,P,S) where V = {S, T} T = {0, 1} P = { S -> T00T
T -> 0T | 1T | ε }

Solution: Leftmost Derivation

w = **1000**111

S ⇒ **T** 0 0 T
⇒ 1 **T** 0 0 T
⇒ 1 0 **T** 0 0 T
⇒ 1 0 ε 0 0 T
-> 1 0 0 0 **T**
-> 1 0 0 0 1 **T**
-> 1 0 0 0 1 1 **T**
-> 1 0 0 0 1 1 **T**
-> 1 0 0 0 1 1 1 **T**
-> 1 0 0 0 1 1 1 ε
-> 1 0 0 0 1 1 1



Rightmost Derivation w = **1000**111

S -> T 0 0 **T**

-> T 0 0 1 **T**

-> T 0 0 1 1 **T**

-> T 0 0 1 1 1 **T**

-> T 0 0 1 1 1 ε

-> **T** 0 0 1 1 1

-> 1 **T** 0 0 1 1 1

-> 1 0 **T** 0 0 1 1 1

-> 1 0 ε 0 0 1 1 1

-> 1 0 0 0 1 1 1

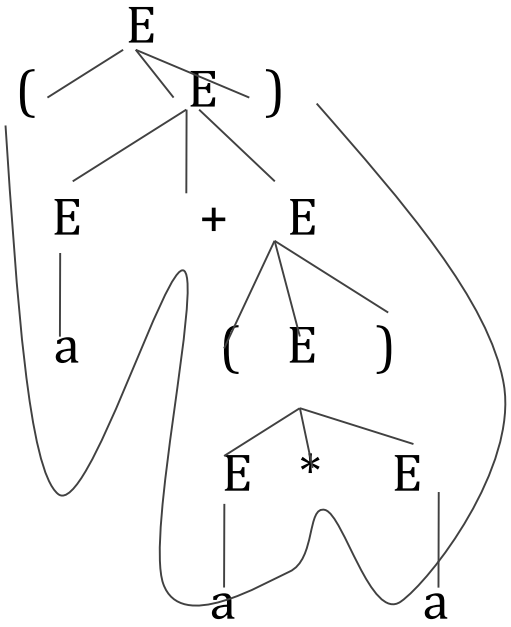
Derive the string (a+(a*a)) using leftmost derivation for the following production. Also draw parse tree.

E -> E + E | E * E | (E) | a

Solution

Leftmost Derivation (a+(a*a))

E -> (**E**)
-> (**E** + E)
-> (a + **E**)
-> (a + (**E**))
-> (a + (**E** * E))
-> (a + (a * **E**))
-> (a + (a * a))



Topic

Ambiguity in Grammars and Languages

Ambiguity in Grammars and Languages

- ✓ The parse tree has to be unique even though the derivation is leftmost or rightmost.
- ✓ But if there exists more than one parse trees for a given grammar, that means there could be more than one leftmost derivation or rightmost derivation possible and then that grammar is said to be ambiguous grammar.

Ambiguous – more than one parse tree for single input string.

Unambiguous Grammar

The grammar which has at most one parse tree for a given input string.

Unambiguous – At most one parse tree

Example 1:

Show that the following grammar is ambiguous.

E -> E + E | E * E

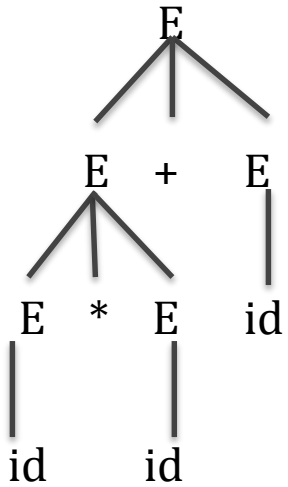
E -> id

E -> E * E -> E * E + E -> id * id + id

Solution

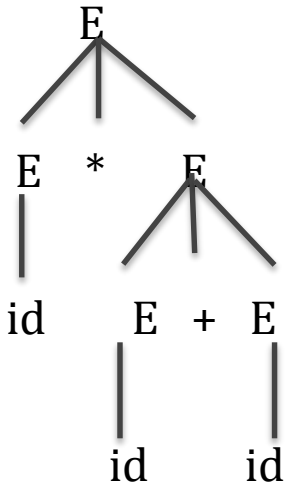
Leftmost derivation 1

E -> **E** + E
-> **E** * E + E
-> id * **E** + E
-> id * id + **E**
-> id * id + id



Leftmost derivation 2

E -> **E** * E
-> id * **E**
-> id * **E** + E
-> id * id + **E**
-> id * id + id

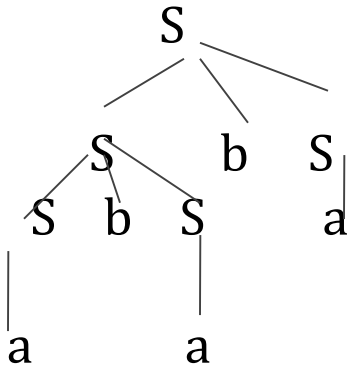


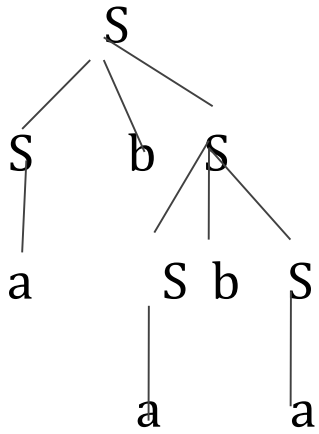
Example 2:

If G is the grammar S -> S b S | a. Show that G is ambiguous

S -> **S** b S
-> **S** b S b S
-> a b **S** b S
-> a b a b **S**
-> a b a b a

S -> **S** b S
-> a b **S**
-> a b **S** b **S**
-> a b a b a





Two different parse tree. Therefore the given grammar is ambiguous.

Topic

Pushdown Automata – Languages of Pushdown Automata and Deterministic Pushdown Automata

Definition of Pushdown Automata

Introduction

- ✓ The context free language is defined by the special type of automata namely push down automata.
- ✓ Push down automata is an extension of the non deterministic finite automata with ε transitions with the addition of a stack.
- ✓ The stack is used to store the string of stack symbols.
- ✓ The stack is used to read the symbol, push and pop only at the top of the stack.
- ✓ The pushdown automata can remember an infinite amount of string information.
- ✓ The pushdown automata can access the information on its stack in last in first out way.
- ✓ The pushdown automata can recognize only the context free language.
- ✓ The pushdown automata is more powerful than finite automata which has finite memory.

Formal Definition of Push Down Automata

- ✓ The push down automata or Non deterministic push down automata involves seven components as

$$P = (Q, \sum, \Gamma, \delta, q_0, Z_0, F)$$

Where,

Q -> Finite set of states

$\sum$ -> Finite set of input symbols

Γ -> Finite stack alphabet.

δ -> Transition Function

q_0 -> Starting state of PDA

Z0 -> Starting symbol of stack

F -> Set of accepting states of final state

Representation of Push Down Automata

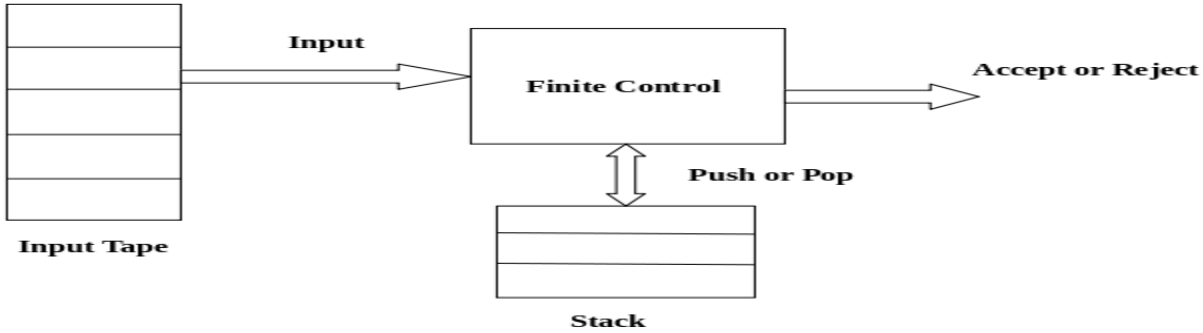


Fig: Pushdown Automata

The push down automata consists of the following

✓**Input tape:** The input tape is divided in many cells or symbols. The input head is read-only and may only move from left to right, one symbol at a time.

✓**Finite control:** The finite control has some pointer which points the current symbol which is to be read.

✓**Stack:** The stack is a structure in which we can push and remove the items from one end only. It has an infinite size. In PDA, the stack is used to store the items temporarily.

Transition function of Push Down Automata

The transition function of PDA depends on the current state, input symbol and the symbol on the top of the stack. The transition function δ take triple argument and output of δ is a finite set of pair of arguments of the form.

$$\delta(q, a, x) = (p, \gamma)$$

Where,

q -> State in Q

a -> Input symbol in ε or a=ε

x -> Stack symbol at the top x ϵ Γ

p -> New state after reading 'a'

γ -> Symbol that is pushed on to the stack that replaces the top symbol x on the stack. If γ = ε, then the stack is popped. If γ = x, then the stack is unchanged. If γ = yz, then x is replaced by z and y is pushed onto the stack.

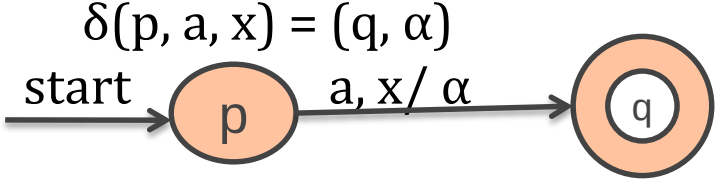
Graphical representation of PDA

The transition diagram is the graphical representation of the push down automata which has the following,

i) The nodes Corresponds to the state of PDA.

ii) An arrow labeled 'start' indicates the starting state and double circled states are the accepting states.

iii) An arc indicates the transition of PDA. Each arc is labeled as a, x/α from state p to state q means as δ(p, a, x) goes to (q, α)



Where,

a -> input symbol

x -> old top stack symbol

α -> New top stack symbol

Instantaneous Description of PDA

The execution status of the push down automata is represented by the instantaneous description (ID) of PDA. The instantaneous description records the state, stack contents and the input symbols.

The ID is defined 3 - tuples (q, a, z)

Where

q -> State of PDA'

a -> Remaining input

z -> Stack Contents.

If a PDA P = (Q, $\sum$, Γ, δ, q_0, Z_0, F) has the transition $\delta(p, a, x) = (q, \alpha)$, then for all the string w in $\sum^*$ and β in Γ^*, the ID is given by

$$(p, aw, x\beta) \vdash (p, w, \alpha\ \beta)$$

This means that by reading the input symbol 'a' at the state 'q' with x as top stack symbol replaces α for x and reaches the state 'p'.

Languages of Push down automata

The language of PDA is the set of all strings accepted by the PDA. Generally the string is accepted by PDA after consuming the input, the PDA should enter the final state. This method is accepting the language by reaching final state.

Else after reading the input, the stack should be empty, then the language is accepted by PDA. So the two methods of accepting a string in the push down automata is a follows.

i) Acceptance by final state

Let P = (Q, ∑, Γ, δ, q_0, Z_0, F) be a PDA, then the language L(p) accepted by final state is

$$\mathbf{L(p) = \{ w / (q_0, w, z_0) \vdash^* (q, \varepsilon, \alpha)\}}$$

Where q ϵ F and α is any stack symbol.

In this method the PDA initially in the start state and consumes w from the input, and enters an accepting state. Here after entering the final state, the content of the stack is not considered.

ii) Acceptance by empty stack

Let P = (Q, ∑, Γ, δ, q_0, Z_0, F) be a PDA, then the language N(p) accepted by final state is

$N(p) = \{ w / (q_0, w, z_0) \vdash^* (q, \varepsilon, \varepsilon)\}$

Where q ϵ F and

Deterministic Pushdown automata

The PDA P = (Q, ∑, Γ, δ, q_0, Z_0, F) is said to be deterministic push down automata(DPDA), if and only if the following conditions are satisfied.

i) δ(q, a, X) has at most one member for q in Q, a in ε, X in Γ

ii) δ(q, a, X) is non empty, then δ(q, ε, X) must be empty

✓The deterministic push down automata has atmost 1 move for the given state, input symbol and stack symbol.

✓The deterministic push down automata does not have a choice of move in the same state, using the same input and stack symbol.

✓The language accepted by deterministic push down automata is deterministic context free language.

Example 1:

Design the PDA to accept the language L={0^n1^n/n>=1} accepting the final state.

Solution 0000 1111

L={01,**0011**,000111,00001111……..}

Step1: Transition Function

$\delta(q_0, 0, Z_0) = (q_0, 0Z_0)$ //Push 0

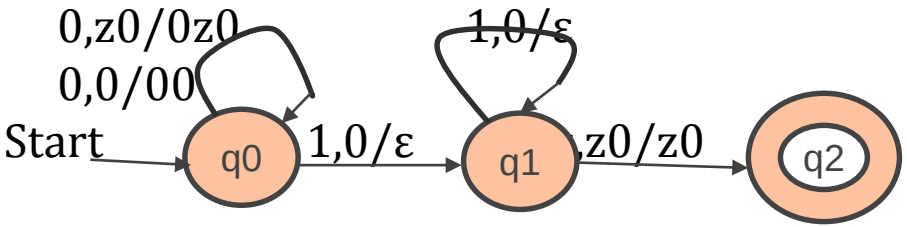
$\delta(q_0, 0, 0) = (q_0, 00)$ // Push 0

$\delta(q_0, 1, 0) = (q_1, \varepsilon)$ // Pop 0

$\delta(q_1, 1, 0) = (q_1, \varepsilon)$ // Pop 0

$\delta(q_1, \varepsilon, Z_0) = (q_2, Z_0)$ // Accept

Step2: Transition Diagram



Step3: Instantaneous Description of PDA

Consider the input string w=000111 w=001

$\delta(q_0 , \mathbf{0}00111, Z_0) \vdash \delta(q_0, \mathbf{0}0111, 0z_0)$

$\vdash \delta(q_0, \mathbf{0}111, 00z_0)$

$\vdash \delta(q_0, \mathbf{1}11, \mathbf{0}00z_0)$

$\vdash \delta(q_1, \mathbf{1}1 , \mathbf{0}0z_0)$

$\vdash \delta(q_1, \mathbf{1} , \mathbf{0}z_0)$

$\vdash \delta(q_1, \boldsymbol{\varepsilon} , 0z_0)$

$\vdash \delta(q_{2,} z_0)$

$\vdash$Accept

Example 1:

Construct the PDA to accept the set of all strings over the alphabets {0,1} such total number of 0's are more than total number of 1's.

Solution

L={001,100,01100,1100010,00001,10000,10101000,……..}

Step 1: Transition Function

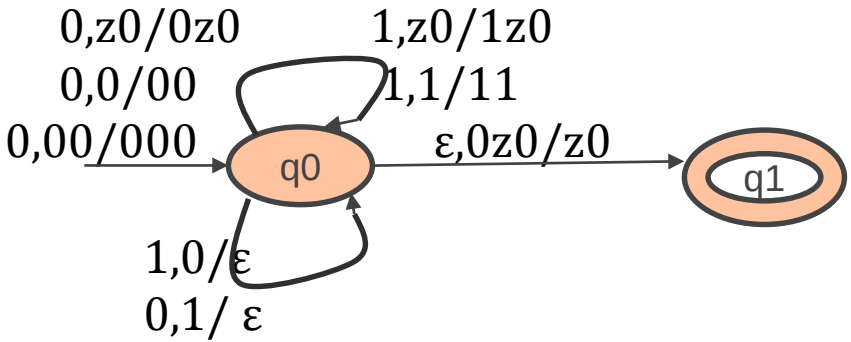
$\delta(q_0 , 0, Z_0) = (q_0,0z_0)$ $\quad\quad$ $\delta(q_0 , 1, Z_0) = (q_0,1z_0)$

$\delta(q_0 , 0, 0Z_0) = (q_0,00z_0)$ $\quad\quad$ $\delta(q_0 , 1, 1Z_0) = (q_0,11z_0)$

$\delta(q_0 , 1, 00Z_0) = (q_0,\varepsilon)$ $\quad\quad$ $\delta(q_0 , 0, 11Z_0) = (q_0, \varepsilon)$

$\delta(q_0 , \varepsilon, 0Z_0) = (q_1,Z_0)$

Step 1: Transition Diagram



Step3: Instantaneous Description of PDA

Consider the input string w=0100

$\delta(q_0 , \mathbf{0}100, \mathbf{Z}_0) \vdash \delta(q_0, 100 , 0z_0)$

$\vdash \delta(q_0, \mathbf{1}00 , \mathbf{0}z_0)$

$\vdash \delta(q_0, \mathbf{0}0 , z_0)$

$\vdash \delta(q_0, \mathbf{0} , 0z_0)$

$\vdash \delta(q_0, \varepsilon , 00z_0)$

$\vdash \delta(q1, z_0)$

$\vdash$ Accept

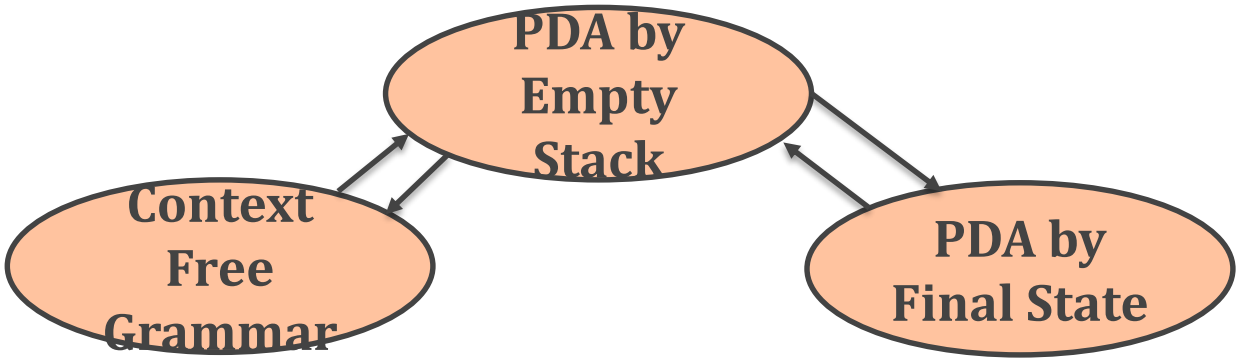
Topic

Equivalence of Pushdown automata and CFG

Introduction

The context free grammar is recognized by the push down automata. The languages defined by PDA's are the context free language. There are three classes of languages that are equivalent to each other and they are

1. The context-free languages (The language defined by CFG's).
2. The languages that are accepted by empty stack by some PDA.
3. The languages that are accepted by final state by some PDA.



Thus it is possible to convert pushdown automata into CFG and CFG to pushdown automata.

Converting Context Free Grammar to PDA

We can construct a Pushdown Automata that simulates the left most derivation of CFG 'G'. Let G be the context free grammar G=(V,T,P,S). To construct PDA P from G that accepts L(G) by empty stack N(P) is as follows

$$P = (\{q\}, T, V \cup T, \delta, q, z_0, s)$$

Where δ is the transition function that is defined for all the Non terminals and terminals of the grammar G.

Rules for Converting CFG to PDA

1. For such Nonterminal A in CFG

   $\delta(q, \varepsilon, A) = \{ (q, B) \} \rightarrow$ (If A -> B is a production of CFG)

2. For each terminal 'a' in CFG

   $\delta(q, a, a) = \{ (q, \varepsilon) \} \rightarrow$ (If aϵT in CFG)

Example 1:

Construct the PDA for the following grammar E -> E + E | E * E | a

Solution

Step1: Transition Function

Terminal = { +, *, a}

Nonterminal = { E }

i) For Nonterminal 'E' :

$\delta(q, \varepsilon, E) = \{ (q, E+E), (q,E*E), (q,a)\}$

ii) For Terminal '+', '*', 'a'

$\delta(q, +, +) = (q,\varepsilon)$

$\delta(q, *, *) = (q,\varepsilon)$

$\delta(q, a, a) = (q,\varepsilon)$

Step 2: Instantaneous Description

E->E+E -> a+a

W=a+a

$\delta(q, a+a, E) \vdash \delta(q, \mathbf{a}+a, E+E)$

$\vdash \delta(q, \mathbf{a}+a, E+E)$

$\vdash \delta(q, \mathbf{a}+a, \mathbf{a}+E)$ //pop a

$\vdash \delta(q, \mathbf{+}a, \mathbf{+}E)$ //pop +

$\vdash \delta(q, a, E)$

$\vdash \delta(q, \mathbf{a}, \mathbf{a})$ //pop

$\vdash \delta(q, \varepsilon, \varepsilon)$

Accept

Convert PDA to Context free grammar

The CFG and PDA has a strong relationship. Let $P = (Q, \sum, \Gamma, \delta, q_0, Z_0, F)$ is a PDA. There exists CFG G which is accepted by PDA P. The G can be defined as

$G=\{V,T,P,S\}$

Getting production rule P use the following algorithm.

Steps for getting production rules of CFG

i) If q_0 is start state in PDA

q_n is final state of PDA

Then **$[q_0\ Z\ q_n]$** becomes start state of CFG. Here Z represents the stack symbol.

ii) The production rule for the ID of the form

$\delta(q_i, a, Z_0) = (q_{i+1}, Z_1Z_2)$ can be obtained as

$[q_i\ Z_0\ q_{i+k}] \rightarrow a\ [q_{i+1}\ Z_1\ q_m]\ [q_m\ Z_2\ q_{i+k}]$

Where q_m – represents the Intermediate States.

Z_0, Z_1, Z_2 – Stack symbols

a – Input Symbols

iii) The production rule for the ID of the form

$\delta(q_i, a, Z_0) = (q_{i+1}, \varepsilon)$ can be obtained as

$[q_i \ \ Z_0 \ \ q_{i+1}] \rightarrow a$

Example 1:

Obtain CFG for the PDA given as below $A=(\{q_0,q_1\}, \{0,1\}, \{A,Z\}, \delta, Z, \{q_1\})$ where δ is as given below

$\delta(q_0, 0, Z) = (q_0, AZ)$

$\delta(q_0, 1, A) = (q_0, AA)$

$\delta(q_0, 0, A) = (q_1, \varepsilon)$

Solution

Step 1: Construct start symbol

S->[q0 Z q0] S->[q0 Z q1]

Step 2: Production rule for ID

i) $\delta(q_0, 0, Z) = (q_0, AZ)$ rule 2

$\delta(q_i, a, Z_0) = (q_{i+1}, Z_1Z_2)$

$qi=q_0$ $a=0$ $Z_0=Z$ $q_{i+1}=q0$ $Z_1=A$ $Z_2=Z$ $q_{i+k}=q0$ $Z=Z$

$q_m=0$ or 1(intermediate states q0,q1) $q_{i+k}=q1$

$[q_i\ Z_0\ q_{i+k}] \rightarrow a\ [q_{i+1}\ Z_1\ q_m]\ [q_m\ Z_2\ q_{i+k}]$

$[q0\ Z\ q0] \rightarrow 0[q0\ A\ q0]\ [q0\ Z\ q0] \mid 0[q0\ A\ q1]\ [q1\ Z\ q0]$

$[q0\ Z\ q1] \rightarrow 0[q0\ A\ q0]\ [q0\ Z\ q1] \mid 0[q0\ A\ q1]\ [q1\ Z\ q1]$

ii) $\delta(q_0, 1, A) = (q_0, AA)$ rule 2

$\delta(q_i, a, Z_0) = (q_{i+1}, Z_1Z_2)$

$qi=q0$ $a=1$ $Z0=A$ $qi+1=q0$ $Z1=A$ $Z2=A$ $qm=q0,q1$ $qi+k=q0,q1$

$[q_i\ Z_0\ q_{i+k}] \rightarrow a\ [q_{i+1}\ Z_1\ q_m]\ [q_m\ Z_2\ q_{i+k}]$

$[q0\ A\ q0] \rightarrow 1[q0\ A\ q0][q0\ A\ q0] \mid 1[q0\ A\ q1][q1\ A\ q0]$

$[q0\ A\ q1] \rightarrow 1[q0\ A\ q0]\ [q0\ A\ q1] \mid 1\ [q0\ A\ q1]\ [q1\ A\ q1]$

iii) $\delta(q_0, 0, A) = (q_1, \varepsilon)$ rule 3

$\delta(q_i, a, Z_0) = (q_{i+1}, \varepsilon)$ can be obtained as

qi=q0 a=0 Z0=A qi+1=q1

$[q_i \ Z_0 \ q_{i+1}] \rightarrow a$

[q0 A q1] → 0

Ans:

S->[q0 Z q0] S->[q0 Z q1]

[q0 Z q0] → 0[q0 A q0] [q0 Z q0] | 0[q0 A q1] [q1 Z q0]

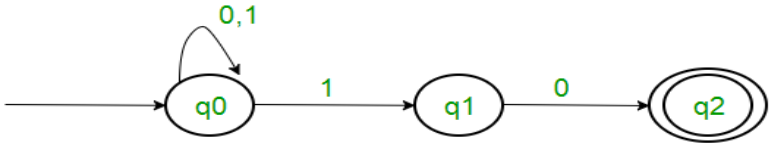
[q0 Z q1] → 0[q0 A q0] [q0 Z q1] | 0[q0 A q1] [q1 Z q1]

[q0 A q0] → 1[q0 A q0][q0 A q0] | 1[q0 A q1][q1 A q0]

[q0 A q1] → 1[q0 A q0] [q0 A q1] | 1 [q0 A q1] [q1 A q1]

[q0 A q1] → 0

Step 4: Process the NFA using union functions and write it for DA transition.
Step 5: The states in NFA are represented using braces {….} and in DFA using brackets […]
Step 6: Process all the states until there is no repetition of a new state.
Example: Convert the given NDFA into DFA



Solution
The starting state in NFA is {q0}
For DFA,
{q0}=[q0] in DFA -> A (New State)
Step 1: Process the input symbol {0,1}
δ(A,0) = δ(q0,0)
= [q0]
δ(A,0)= [q0] = A
δ(B,0)=δ({q0,q1},0)
= δ(q0,0) ∪ δ(q1,0)
= {q0 ∪ q2}=[**q0,q2]=C**

δ(A,1) = δ(q0,1)
=[q0,q1]
δ(A,1)= [q0,q1] = B (New State)
δ(B,1)=δ({q0,q1},1)
= δ(q0,1) ∪ δ(q1,1)
= {q0,q1} ∪ ϕ
= [q0,q1] =B

Topic

Types of Grammar

(or)

Chomsky Hierarchy of Language

Introduction

Chomsky Hierarchy represents the class of languages that are accepted by the different machine. The category of language in Chomsky's Hierarchy is as given below:

Type 0 known as Unrestricted Grammar.
Type 1 known as Context Sensitive Grammar.
Type 2 known as Context Free Grammar.
Type 3 Regular Grammar.

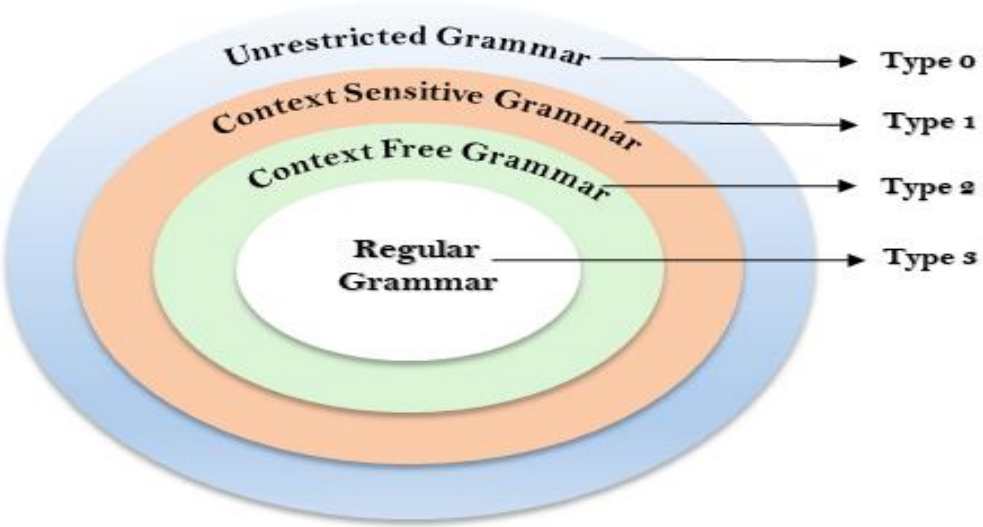


Fig: Chomsky Hierarchy

This is a hierarchy. Therefore every language of type 3 is also of type 2, 1 and 0. Similarly, every language of type 2 is also of type 1 and type 0, etc.

Grammar Type	Grammar Accepted	Language Accepted	Automaton
Type 0	Unrestricted grammar	Recursively enumerable language	Turing Machine
Type 1	Context-sensitive grammar	Context-sensitive language	Linear-bounded automaton
Type 2	Context-free grammar	Context-free language	Pushdown automaton
Type 3	Regular grammar	Regular language	Finite state automaton

Type - 3 Grammar

✓Type-3 grammars generate regular languages which can be described by regular expression. These languages can be modeled by NFA or DFA

✓Type-3 grammars must have a single non-terminal on the left-hand side and a right-hand side consisting of a single terminal or single terminal followed by a single non-terminal.

The productions must be in the form

$X \rightarrow a$ or $X \rightarrow aY$

where $X, Y \in N$ (Non terminal) and $a \in T$ (Terminal)

The rule $S \rightarrow \varepsilon$ is allowed if S does not appear on the right side of any rule.

Example

$X \rightarrow \varepsilon$

$X \rightarrow a \mid aY$

$Y \rightarrow b$

Type - 2 Grammar

Type-2 grammars generate context-free languages which can be represented by the context free grammar (CFG). The productions must be in the form

$$\mathbf{A \rightarrow \gamma}$$

where $A \in N$ (Non terminal)

and $\gamma \in (T \cup N)^*$ (String of terminals and non-terminals).

These languages generated by these grammars are be recognized by a non-deterministic pushdown automaton.

Example

$S \rightarrow X\ a$

$X \rightarrow a$

$X \rightarrow aX$

$X \rightarrow abc$

$X \rightarrow \varepsilon$

Type - 1 Grammar

Type-1 grammars generate context-sensitive languages. The languages generated by these grammars are recognized by a linear bounded automaton. The productions must be in the form **$\alpha A \beta \rightarrow \alpha \gamma \beta$**

where $A \in N$ (Non-terminal) and $\alpha, \beta, \gamma \in (T \cup N)^*$ (Strings of terminals and non-terminals)

Rules:

✓The context sensitive grammar may have more than one symbol on the left hand side of their production rules.

✓The number of symbols on the left-hand side must not exceed the number of symbols on the right-hand side.

✓The rule $S \rightarrow \varepsilon$ is allowed if S does not appear on the right side of any rule.

✓The Type 1 grammar should be Type 0. In type 1, Production is in the form of $V \rightarrow T$ Where the count of symbol in V is less than or equal to T.

Example

$AB \rightarrow AbBc$

$A \rightarrow bcA$

$B \rightarrow b$

Type - 0 Grammar (Unrestricted grammar)

Type-0 grammars generate recursively enumerable languages. The productions have no restrictions. They are any phase structure grammar including all formal grammars. They generate the languages that are recognized by a Turing machine.

The productions can be in the form of

$$\alpha \rightarrow \beta$$

where α - is a string of terminals and non-terminals with at least one non-terminal and α cannot be null.

β - is a string of terminals and non-terminals.

Example1

$S \rightarrow ACaB$

$Bc \rightarrow acB$

$CB \rightarrow DB$

$aD \rightarrow b$

Example2

$bAa \rightarrow aa$

$S \rightarrow c$