RGPVNOTES.IN

Program : **B.Tech**

Subject Name: **Theory of Computation**

Subject Code: **CS-501**

Semester: **5th**

Subject Notes
CS501- Theory of Computation
B. Tech, CSE-5th Semester

Syllabus: Push down Automata: example of PDA, deterministic and non-deterministic PDA, conversion of PDA into context free grammar and vice versa, CFG equivalent to PDA, Petri net model.

Objective: The goal is to make a pushdown automaton that will accept all of the input strings that the context-free grammar accepts.

Unit-IV: Push down Automata:

A pushdown automaton is a way to implement a context-free grammar in a similar way we design DFA for a regular grammar. A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information.

Basically, a pushdown automaton is: "Finite state machine" + "a stack"

A pushdown automaton has three components:

- An input tape,
- A control unit, and
- A stack with infinite size.

The stack head scans the top symbol of the stack.

A stack does two operations:

- Push: a new symbol is added at the top.
- Pop: the top symbol is read and removed.

A PDA may or may not read an input symbol, but it has to read the top of the stack in every transition.

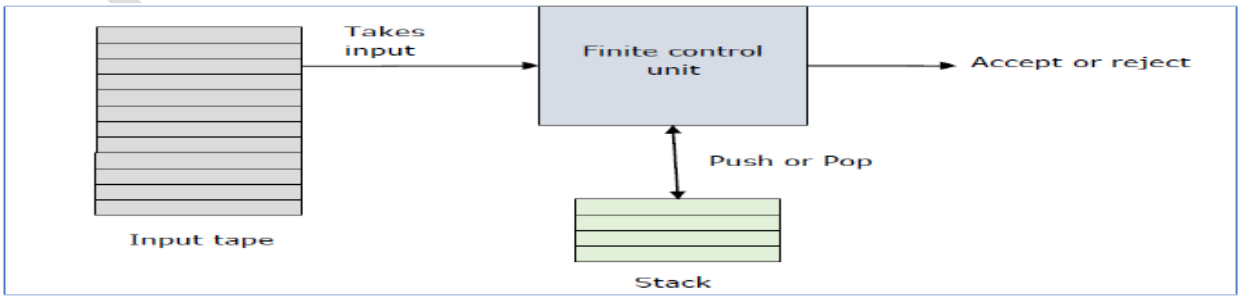

Figure 4.1: Pushdown Automata

Applications of PDA:

1. Online Transaction process system.
2. Used in compiler design (parser design for syntactic analysis)
3. Tower of Hanoi (Recursive Solution)

Formal Definition of PDA:

A deterministic pushdown automaton is a 7 -tuples M = (Q, Σ, Γ, δ, q0, Z0, F), where

- Q is a finite set of states,
- Σ is a finite set of tape alphabet or input symbol
- Γ is a finite set of stack alphabet
- q0 is initial state, q0 is an element of Q
- Z0 is Initial symbol on top of stack
- F set of final state which is sub set of Q.
- δ is transition function which maps (Q x Σ x Γ) into Q x Γ*

Transition of PDA:

Transition function of PDA is denoted as δ (q, a, X) = (p, Y) which specified transition in PDA is function of three components:

1. Present state of PDA, q
2. Symbol of input alphabet, a, being read by PDA.
3. Symbol at top of stack, X.

In every transition

- PDA enters into new state p or remain into same state p = q
  - if a = ε than no symbol is consumed
  - If Y = zX, symbol z is pushed into the stack at the top.
  - If Y = ε, Symbol X at the top of stack is popped.
- If Y = X, No change of symbol at top of stack.

Example: The following diagram shows a transition in a PDA from a state q_1 to state q_2, labeled as "a,b/c"

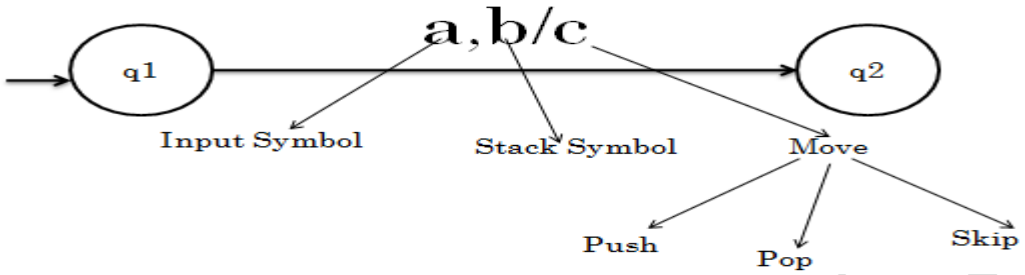

Figure 4.2: Example of PDA

Representation of PDA:

Corresponding to transition function δ (q, a, X) = (p, Y) transition diagram will have

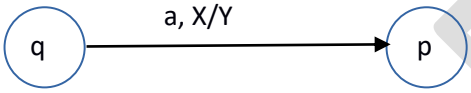

Figure 4.3: Representation of PDA

Terminologies related to PDA:

Instantaneous Description:

The instantaneous description (ID) of a PDA is represented by a triplet (q, w, s) where

- q is the state
- w is unconsumed input
- s is the stack contents

Turnstile Notation:

The "turnstile" notation is used for connecting pairs of ID's that represent one or many moves of a PDA. The process of transition is denoted by the turnstile symbol "⊢".

Consider a PDA (Q, Σ, S, δ, q0, I, F). A transition can be mathematically represented by the following turnstile notation:

$$(p, aw, T\beta) \vdash (q, w, \alpha b)$$

This implies that while taking a transition from state p to state q, the input symbol 'a' is consumed, and the top of the stack 'T' is replaced by a new string 'α'.

Note: If we want zero or more moves of a PDA, we have to use the symbol (⊢*) for it.

Example: Define the pushdown automata for language {anbn | n > 0}

Solution: M = where Q = {q0, q1} and Σ = {a, b} and Γ = {A, Z} and &delta is given by

&delta (q0, a, Z) = {(q0, AZ)}

&delta (q0, a, A) = {(q0, AA)}

&delta (q0, b, A) = {(q1, ∈)}

&delta (q1, b, A) = {(q1, ∈)}

&delta (q1, ∈, Z) = {(q1, ∈)}

Let us see how this automaton works for aabb

Deterministic PDA:

A PDA is said to be deterministic if and only if following conditions are met

1. δ (q, a, X) has at most one transition.
2. δ (q, ε, X) = Φ

Acceptance of PDA:

There are two different ways to define PDA acceptability.

Final State Acceptability:

In final state acceptability, a PDA accepts a string when, after reading the entire string, the PDA is in a final state. From the starting state, we can make moves that end up in a final state with any stack values. The stack values are irrelevant as long as we end up in a final state.

For a PDA (Q, Σ, S, δ, q0, I, F), the language accepted by the set of final states F is:

$$L(PDA) = \{w \mid (q0, w, I) \vdash^* (q, \varepsilon, x), q \in F\}$$

for any input stack string x.

Empty Stack Acceptability:

Here a PDA accepts a string when, after reading the entire string, the PDA has emptied its stack.

For a PDA (Q, Σ, S, δ, q0, I, F), the language accepted by the empty stack is:

$$L(PDA) = \{w \mid (q0, w, I) \vdash^* (q, \varepsilon, \varepsilon), q \in Q\}$$

Example: Construct a PDA that accepts L= $\{0^n1^n \mid n \geq 0\}$

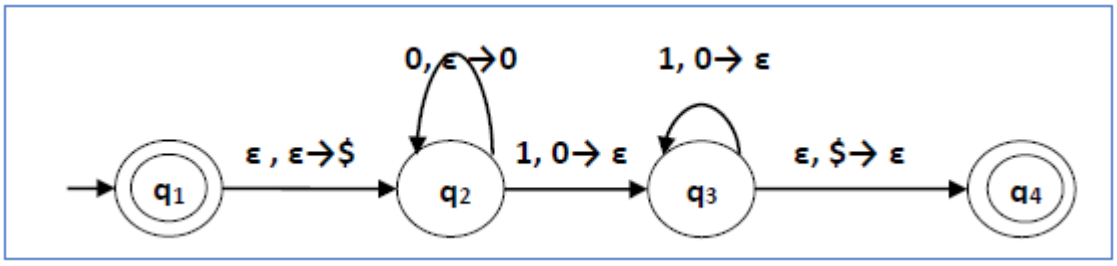

Figure 4.4: Example of PDA

This language accepts L = {ε, 01, 0011, 000111, ... }

Here, in this example, the number of 'a' and 'b' have to be same.

- Initially we put a special symbol '$' into the empty stack.
- Then at state q2, if we encounter input 0 and top is Null, we push 0 into stack. This may iterate. And if we encounter input 1 and top is 0, we pop this 0.
- Then at state q3, if we encounter input 1 and top is 0, we pop this 0. This may also iterate. And if we encounter input 1 and top is 0, we pop the top element.
- If the special symbol '$' is encountered at top of the stack, it is popped out and it finally goes to the accepting state q4.

Example: Construct a PDA that accepts the language L over {0, 1} by empty stack which accepts all the string of 0's and 1's in which a number of 0's are twice of number of 1's.

Solution: There are two parts for designing this PDA:

- If 1 comes before any 0's
- If 0 comes before any 1's.

We are going to design the first part i.e. 1 comes before 0's. The logic is that read single 1 and push two 1's onto the stack. Thereafter on reading two 0's, POP two 1's from the stack. The δ can be

1. δ (q0, 1, Z) = (q0, 11, Z) Here Z represents that stack is empty
2. δ (q0, 0, 1) = (q0, ε)

Now, consider the second part i.e. if 0 comes before 1's. The logic is that read first 0, push it onto the stack and change state from q0 to q1. [Note that state q1 indicates that first 0 is read and still second 0 has yet to read].

Being in q1, if 1 is encountered then POP 0. Being in q1, if 0 is read then simply read that second 0 and move ahead. The δ will be:

1. δ (q0, 0, Z) = (q1, 0Z)
2. δ (q1, 0, 0) = (q1, 0)
3. δ (q1, 0, Z) = (q0, ε) (indicate that one 0 and one 1 is already read, so simply read the second 0)
4. δ (q1, 1, 0) = (q1, ε)

Now, summarize the complete PDA for given L is:

1. δ (q0, 1, Z) = (q0, 11Z)
2. δ (q0, 0, 1) = (q1, ε)
3. δ (q0, 0, Z) = (q1, 0Z)
4. δ (q1, 0, 0) = (q1, 0)
5. δ (q1, 0, Z) = (q0, ε)
6. δ (q0, ε, Z) = (q0, ε) ACCEPT state

Non-deterministic Pushdown Automata

The non-deterministic pushdown automata are very much similar to NFA. We will discuss some CFGs which accepts NPDA.

The CFG which accepts deterministic PDA accepts non-deterministic PDAs as well. Similarly, there are some CFGs which can be accepted only by NPDA and not by DPDA. Thus, NPDA is more powerful than DPDA.

Example: Design PDA for Palindrome strips.

Solution: Suppose the language consists of string L = {aba, aa, bb, bab, bbabb, aabaa, ...]. The string can be odd palindrome or even palindrome. The logic for constructing PDA is that we will push a symbol onto the stack till half of the string then we will read each symbol and then perform the pop operation. We will compare to see whether the symbol which is popped is similar to the symbol which is read. Whether we reach to end of the input, we expect the stack to be empty.

This PDA is a non-deterministic PDA because finding the mid for the given string and reading the string from left and matching it with from right (reverse) direction leads to non-deterministic moves. Here is the ID.

1. $\delta(q1, a, Z) = (q1, aZ)$
2. $\delta(q0, b, Z) = (q1, bZ)$
3. $\delta(q0, a, a) = (q1, aa)$
4. $\delta(q1, a, b) = (q1, ab)$
5. $\delta(q1, a, b) = (q1, ba)$
6. $\delta(q1, b, b) = (q1, bb)$

Pushing the symbols onto the stack

7. $\delta(q1, a, a) = (q2, \varepsilon)$
8. $\delta(q1, b, b) = (q2, \varepsilon)$
9. $\delta(q2, a, a) = (q2, \varepsilon)$
10. $\delta(q2, b, b) = (q2, \varepsilon)$
11. $\delta(q2, \varepsilon, Z) = (q2, \varepsilon)$

Popping the symbols on reading the same kind of symbol

Simulation of abaaba

1. $\delta(q1, abaaba, Z)$ Apply rule 1
2. $\vdash \delta(q1, baaba, aZ)$ Apply rule 5
3. $\vdash \delta(q1, aaba, baZ)$ Apply rule 4
4. $\vdash \delta(q1, aba, abaZ)$ Apply rule 7
5. $\vdash \delta(q2, ba, baZ)$ Apply rule 8
6. $\vdash \delta(q2, a, aZ)$ Apply rule 7

7. ⊢ δ(q2, ε, Z) Apply rule 11
8. ⊢ δ(q2, ε) Accept

PDA & Context Free Grammar:

If a grammar G is context-free, we can build an equivalent nondeterministic PDA which accepts the language that is produced by the context-free grammar G. A parser can be built for the grammar G.

Also, if P is a pushdown automaton, an equivalent context-free grammar G can be constructed where

L(G) = L(P)

Algorithm to find PDA corresponding to a given CFG:

Step 1: Convert the given productions of CFG into GNF.

Step 2: The PDA will only have one state {q}.

Step 3: The initial symbol of CFG will be the initial symbol in the PDA.

Step 4: For non-terminal symbol, add the following rule:

1. δ(q, ε, A) = (q, α)

Where the production rule is A → α

Step 5: For each terminal symbols, add the following rule:

1. δ (q, a, a) = (q, ε) for every terminal symbol

Example 1:

Convert the following grammar to a PDA that accepts the same language.

1. S → 0S1 | A
2. A → 1A0 | S | ε

Solution:

The CFG can be first simplified by eliminating unit productions:

1. S → 0S1 | 1S0 | ε

Now we will convert this CFG to GNF:

1. S → 0SX | 1SY | ε
2. X → 1
3. Y → 0

The PDA can be:

R1: δ (q, ε, S) = {(q, 0SX) | (q, 1SY) | (q, ε)}

R2: δ (q, ε, X) = {(q, 1)}

R3: δ (q, ε, Y) = {(q, 0)}

R4: δ (q, 0, 0) = {(q, ε)}

R5: δ (q, 1, 1) = {(q, ε)}

Example 2:

Construct PDA for the given CFG, and test whether 0104 is acceptable by this PDA.

1. S → 0BB
2. B → 0S | 1S | 0

Solution:

The PDA can be given as:

1. A = {(q), (0, 1), (S, B, 0, 1), δ, q, S,?}

The production rule δ can be:

R1: δ (q, ε, S) = {(q, 0BB)}

R2: δ (q, ε, B) = {(q, 0S) | (q, 1S) | (q, 0)}

R3: δ (q, 0, 0) = {(q, ε)}

R4: δ (q, 1, 1) = {(q, ε)}

Testing 010^4 i.e. 010000 against PDA:

1. δ (q, 010000, S) ⊢ δ (q, 010000, 0BB)
2. ⊢ δ (q, 10000, BB) R1
3. ⊢ δ(q, 10000,1SB) R3
4. ⊢ δ(q, 0000, SB) R2
5. ⊢ δ(q, 0000, 0BBB) R1
6. ⊢ δ(q, 000, BBB) R3
7. ⊢ δ(q, 000, 0BB) R2
8. ⊢ δ(q, 00, BB) R3
9. ⊢ δ(q, 00, 0B) R2
10. ⊢ δ(q, 0, B) R3
11. ⊢ δ(q, 0, 0) R2
12. ⊢ δ(q, ε) R3
13. ACCEPT

Thus 010^4 is accepted by the PDA.

Example 3:

Draw a PDA for the CFG given below:

1. S → aSb
2. S → a | b | ε

Solution:The PDA can be given as:

P = {(q), (a, b), (S, a, b, z0), δ, q, z0, q}

The mapping function δ will be:

R1: δ(q, ε, S) = {(q, aSb)}

R2: δ(q, ε, S) = {(q, a) | (q, b) | (q, ε)}

R3: δ(q, a, a) = {(q, ε)}

R4: δ(q, b, b) = {(q, ε)}

R5: δ(q, ε, z0) = {(q, ε)}

Simulation: Consider the string aaabb

1. δ(q, εaaabb, S) ⊢ δ(q, aaabb, aSb) R3
2. ⊢ δ(q, εaabb, Sb) R1
3. ⊢ δ(q, aabb, aSbb) R3
4. ⊢ δ(q, εabb, Sbb) R2
5. ⊢ δ(q, abb, abb) R3
6. ⊢ δ(q, bb, bb) R4
7. ⊢ δ(q, b, b) R4
8. ⊢ δ(q, ε, z0) R5
9. ⊢ δ(q, ε)
10. ACCEPT

Algorithm to find CFG corresponding to a given PDA

Input − A CFG, G = (V, T, P, S)

Output − Equivalent PDA, P = (Q, ∑, S, δ, q_0, I, F) such that the non- terminals of the grammar G will be $\{X_{wx} \mid w,x \in Q\}$ and the start state will be $A_{q0,F}$.

Step 1 − For every w, x, y, z ∈ Q, m ∈ S and a, b ∈ ∑, if δ (w, a, ε) contains (y, m) and (z, b, m) contains (x, ε), add the production rule X_{wx} → a X_{yz}b in grammar G.

Step 2 − For every w, x, y, z ∈ Q, add the production rule X_{wx} → $X_{wy}X_{yx}$ in grammar G.

Step 3 – For w ∈ Q, add the production rule $X_{ww} \rightarrow \varepsilon$ in grammar G.

Example : **Obtain CFG for the given PDA below:**

$\delta(q_0,a,z_0)$ -> (q_1,az_0)

$\delta(q_1,a,a)$ -> (q_1,aa)

$\delta(q_1,b,a)$ -> (q_2,λ)

$\delta(q_2,b,a)$ -> (q_2,λ)

$\delta(q_2,\lambda,z_0)$ -> (q_f,λ)

Sol. The productions of the grammar are as follows: -

$S \rightarrow [q_0, z_0, q]$

1) $[q_0, z_0, q] \rightarrow a\,[q_1, a, p]\,[p, z_0, q]$

2) $[q_1, a, q] \rightarrow a\,[q_1, a, p]\,[p, z_0, q]$

3) $[q_1, a, q] \rightarrow b$

4) $[q_2, a, q] \rightarrow b$

5) $[q_2, z_0, q] \rightarrow \lambda$

Where p & q are q_0 to q_f all combinations.

Petri nets Models

Petri nets are a basic model of parallel and distributed systems (named after Carl Adam Petri). The basic idea is to describe state changes in a system with transitions.

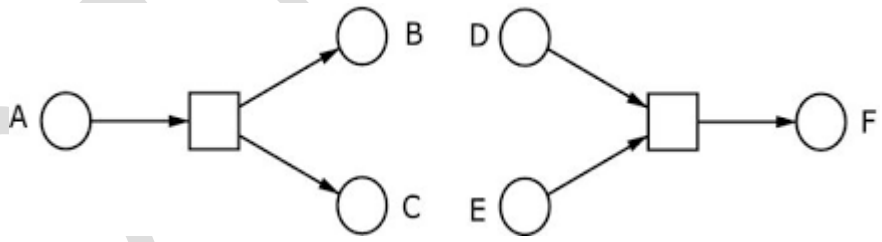

Figure 4.5: Perti-net

Petri nets contain places Circle and rectangle and transitions that may be connected by directed arcs. Places symbolize states, conditions, or resources that need to be met/be available before an action can be carried out. Transitions symbolize action.

Places may contain tokens that may move to other places by executing ("firing") actions. A token on a place means that the corresponding condition is fulfilled or that a resource is available:

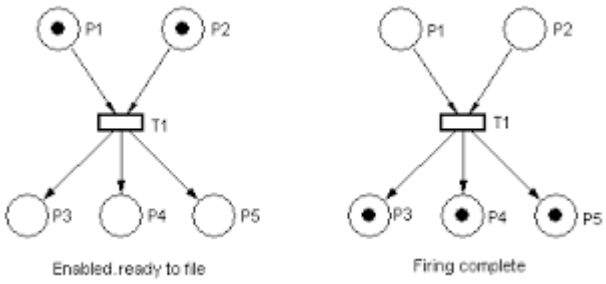

Figure 4.6: Petri net

In the example, transition t may "fire" if there are tokens on places p1 and p2. Firing t will remove those tokens and place new tokens on p3, p4 and p5

A Petri net is a tuple N = h P, T, F, W, m0i, where

• P is a finite set of places,

• T is a finite set of transitions,

• the places P and transitions T are disjoint ($P \cap T = \emptyset$),

• $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation,

• $W : ((P \times T) \cup (T \times P)) \rightarrow IN$ is the arc weight mapping

(where $W(f) = 0$ for all $f \in / F$, and $W(f) > 0$ for all $f \in F$), and

• m0: $P \rightarrow IN$ is the initial marking representing the initial distribution of tokens.

Example: Dining philosophers

There are philosophers sitting around a round table. There are forks on the table, one between each pair of philosophers.

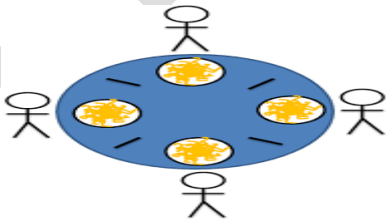

Figure 4.7: Dining philosophers

The philosophers want to eat spaghetti from a large bowl in the center of the table.

Dining philosophers: Petri net

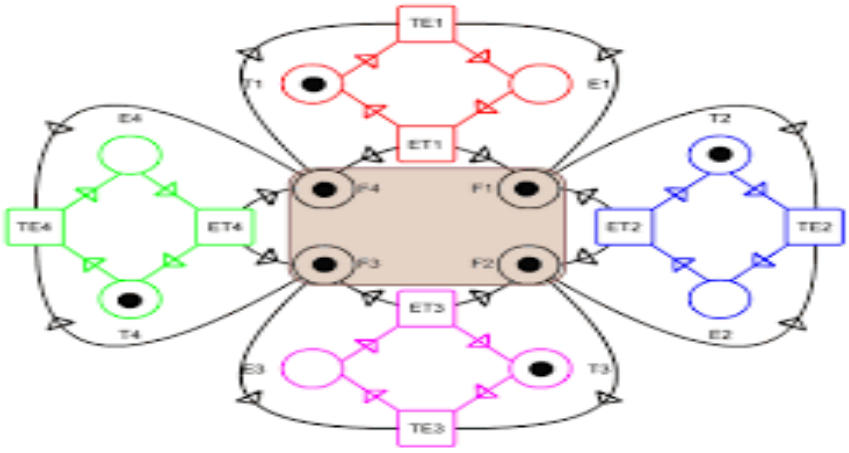

Figure 4.7: Dining philosophers with Petri-net

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in