PUSHDOWN AUTOMATA

The context free languages have a type if automation that defines them. This automaton is called "push down automaton" which can be thought as an ϵ-NFA with the addition of a stack. The presence of a stack means that, the push down automaton can "remember" an infinite amount of information. However, the pushdown automaton can only access the information on its stack in a last in first on way.

Thus, PDA is an abstract machine determined by following three things;

- Input tape
- finite state control
- A stack.

Each moves of the machine is determined by things;

- the current state
- read input symbol
- Symbol on the top of stack.

The moves consist of;

- changing state | staying on same state
- replacing the stack top by string of zero or more symbols.

Popping the top symbol off the stack means replacing it by ϵ, pushing Γ on the stack means replacing stack's top, say x, by y x s assuming the left end of string corresponds to stacks top.

The single move of the machine contain only stack operation either push or pop.

Replacing the stack symbol x lay the string α can be accomplished by a sequence of basic moves (a pop follower by a sequence of 0 or more pushes).

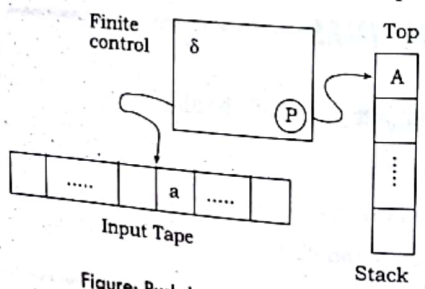


Figure: Pushdown Automaton

Formal Definition: Pushdown Automata (Non-deterministic)

A PDA is defined by seven tuples;

$(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

Where,

Q- Finite set of states.

Σ- Finite set of input symbols | alphabets.

Γ- Finite set of stack alphabet

q_0- Start state of PDA $q_0 \in Q$

Z_0- Initial stack symbol; $Z_0 \in \Gamma$

F- Set of final states; $F \subseteq Q$

δ- Transition function that maps;

$Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to Q \times \Gamma^*$

Thus, moves of PDA can be interpreted as;

$\delta(q, a, z) = \{(P_1, r_1) (p_2, r_2) \ldots\ldots\ldots\ldots (p_m, r_m)\}$ here $q, p_i \in Q$, $a \in \Sigma \cup \{\lambda\}$ & $z \in \Gamma$, $r_i \in \Gamma^*$.

Representation of PDA

PDA can be described by using two techniques:

- Transition Table
- Transition diagram

Transition Table

Consider a context free language defined by the grammar $a^n b^n$. Then the transition table used to represent this language is given as:

Move number	State	Input	Stack Symbol	Move(s)
1	q_0	A	Z_0	(q_0, aZ_0)
2	q_0	A	A	(q_0, aa)
3	q_0	ϵ	A	(q_1, a)
4	q_1	B	A	(q_1, ϵ)
5	q_1	ϵ	Z_0	(q_2, Z_0)

Here $q_0$is the starting state of PDA, q_2 is the final state of the PDA and Z_0 is the initial stack symbol.

Description

Move number 1: If the PDA is in state q_0 and read input a and top of the stack is Z_0 then it remains in the state q_0 and push the input symbol a into the top of the stack. It can be denoted as:

$$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$$

Move number 2: If the PDA is in state q_0 and read input a and top of the stack is a then it remains in the state q_0 and push the input symbol a into the top of the stack. It can be denoted as:

$$\delta(q_0, a, a) = \{(q_0, aa)\}$$

Move number 3: If the PDA is in state q_0 and read nothing (ϵ) and top of the stack is a then it switches to the state q_1 and no changed in the content of stack. It can be denoted as:

$$\delta(q_0, \epsilon, a) = \{(q_1, a)\}$$

Move number 4: If the PDA is in state q_1 and read input b and top of the stack is a then it switches to the state q_1 and pop the symbol from the top of the stack. It can be denoted as:

$$\delta(q_1, b, a) = \{(q_1, \epsilon)\}$$

Move number 5: If the PDA is in state q_1 and read nothing (ϵ) and top of the stack is Z_0 then it switches to the state q_2 and the stack elements remains unchanged. It can be denoted as:

$$\delta(q_1, \epsilon, Z_0) = \{(q_2, Z_0)\}.$$

How the string aaabbb is processed by this PDA.

- Initially, the PDA is in state q_0 and stack symbol is Z_0, it then reads the first symbol a which leads it to the state q_0 and push the symbol 'a' into the stack. Hence the content of the stack is aZ_0.
- Now it reads the second symbol 'a' and top of stack is 'a' so it still remains in the state q_0 and push the element 'a' into the stack. Hence the content of the stack is aaZ_0.
- It then reads the third symbol 'a' and top of stack is 'a' so it still remains in the state q_0 and push the element 'a' into the stack. Hence the content of the stack is $aaaZ_0$.
- It then reads nothing 'ϵ' and top of stack is 'a', so it switches to the state q_1 and the content of the stack remains unchanged. Thus the content of the stack is $aaaZ_0$.

- It then reads the fourth symbol 'b' and top of stack is 'a', so it remains to the state q_1 and pop the top element from the stack. Thus the content of the stack is aaZ_0.
- It then reads the fifth symbol 'b' and top of stack is 'a', so it remains to the state q_1 and pop the top element from the stack. Thus the content of the stack is aZ_0.
- It then reads the sixth symbol 'b' and top of stack is 'a', so it remains to the state q_1 and pop the top element from the stack. Thus the content of the stack is Z_0.
- It then reads nothing (ϵ) and top of stack is Z_0, so it switches to the state q_2 and no operation is performed in stack. Thus the content of the stack is Z_0.

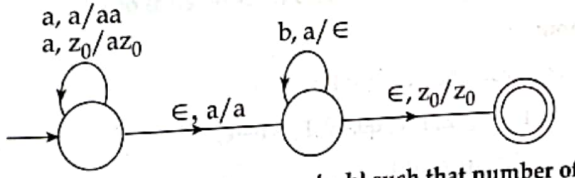
Since, after processing the whole string, PDA reaches to the state q_2 which is accepting state. Hence the string aaabbb is accepted by the PDA.

Transition Diagram

We can use transition diagram to represent a PDA, where

- Any state is represented by a node in diagram.
- Any arrowlabeled with "start" indicates the start state and doubly circled states are accepting / final states.
- The arc corresponds to transition of PDA as:
  - Arc labeled as a, X | α means;$\delta(q, a, x) = (p, \alpha)$ for arc from state p to q.

The PDA of $a^n b^n$ can represented using transition diagram as shown below:



Example 1: A PDA accepting a string over {a, b} such that number of a's and b's are equal. i.e. L={w | w ε {a, b}* and a's and b's are equal}.

Solution:

PDA to describe the above language is given by

$P = \{Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\}$ where,

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

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

$\Gamma = \{a, b, z_0\}$

$Z_0 = Z_0$

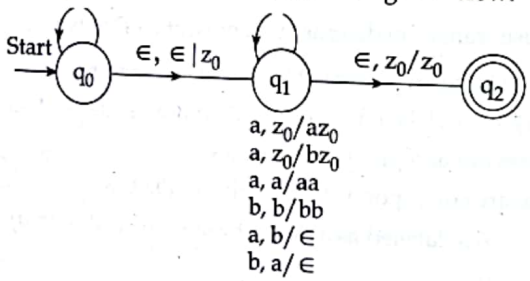
$q_0 = q_0$

$F = \{q_2\}$

And δ is defined as;

1. $\delta(q_1, a, z0) = (q_1, az_0)$
2. $\delta(q_1, b, z0) = (q_1, bz_0)$
3. $\delta(q_1, a, a) = (q_1, aa)$
4. $\delta(q_1, b, b) = (q_1, bb)$
5. $\delta(q_1, a, b) = (q_1, \in)$
6. $\delta(q_1, b, a) = (q_1, \in)$
7. $\delta(q_1, \in, z0) = (q_2, \in)$

So, the transition diagram for this PDA is shown in figure below:



Example 2: Construct a PDA accepting Language;

$L = \{ww^R \mid w \in (0+1)^*$ and w^R is reversal of w$\}$

Solution:

Let us construct the PDA for L as;

$P = \{Q, \Sigma, \Gamma, \delta, q0, z0, F\}$ where,

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

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

$\Gamma = \{a, b, z_0\}$

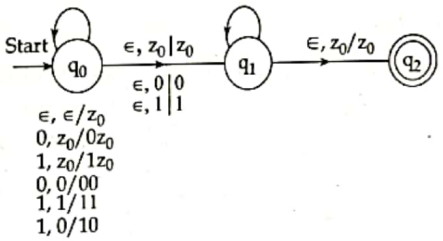
$Z_0 = Z_0$

$q_0 = q_0$

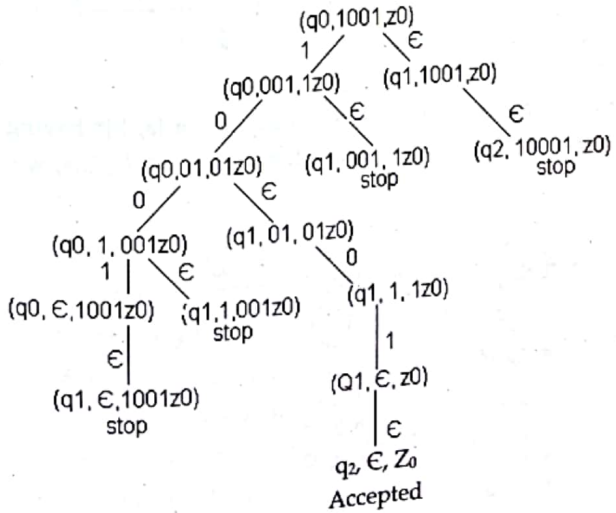
$F = \{q_2\}$

Now, δ is defined as;

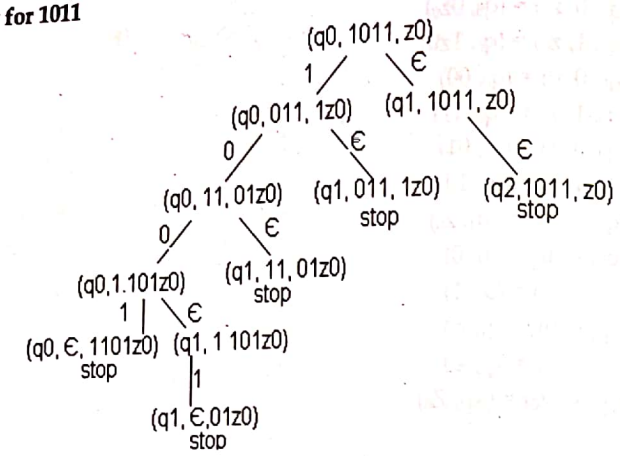
1. $\delta(q_0, 0, z_0) = (q_0, 0z_0)$
2. $\delta(q_0, 1, z_0) = (q_0, 1z0)$
3. $\delta(q_0, 0, 0) = (q_0, 00)$
4. $\delta(q_0, 1, 1) = (q_0, 11)$
5. $\delta(q_0, 0, 1) = (q_0, 01)$
6. $\delta(q_0, 1, 0) = (q_0, 10)$
7. $\delta(q_0, \in, z_0) = (q_1, z_0)$
8. $\delta(q_0, \in, 0) = (q_1, 0)$
9. $\delta(q_0, \in, 1) = (q_1, 1)$
10. $\delta(q_1, 0, 0) = (q_1, \in)$
11. $\delta(q_1, 1,1) = (q_1, \in)$
12. $\delta(q_1, \in, z_0) = (q_2, Z_0)$



Acceptance of any string can also be shown using parse tree. Consider any string 1001;

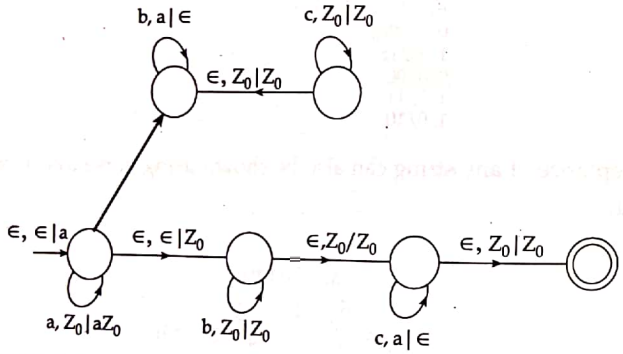


Now for 1011

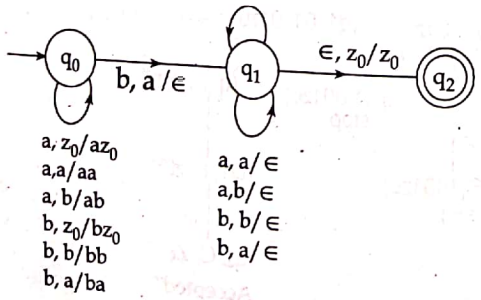


Hence, is not accepted.

Example 3: PDA that recognises the language $\{a^i b^i c^k \mid i,j,k \geq 0$ and $i=j$ or $i=k\}$.



Example 4: PDA that accepts the set L of all strings in {a, b}* having an odd length and whose middle symbol is b, i.e., L = {vbw : v ∈ {a, b}*, w ∈ {a, b}*, |v| = |w|}.



Instantaneous Description of PDA

Any configuration of a PDA can be described by a triplet (q, w, γ).

Where,

q- is the state.

w- is the remaining input.

γ- is the stack contents.

Such a description by triple is called an instantaneous description of PDA (ID).

Instantaneous description is helpful for describing changes in the state, input, and stack of the PDA.

Let P = {Q, Σ, Γ, δ, q_0, z_0, F} be a PDA. Then, we define a relation ⊢, "yields" as;

$(q, aw, z\alpha) \vdash (p, w, \beta\alpha)$ if $\delta(q, a, z) = (p, \beta)$, and a may be ϵ)

This move reflects the idea that, by consuming a (which may be ϵ) from the input, and replacing z on the top of stack by β, we can go from state q to state p. Note that what remains on the input w and what is below the top of stack, β, do not influence the action the PDA.

For the PDA described earlier accepting language ww^R, [see example 2] the accepting sequence of ID's for string 1001 can be shown as;

$(q_0, 1001, Z_0) \vdash (q_0, 001, 1Z_0)$

$\vdash (q_0, 01, 01Z_0)$

$\vdash (q_1, 01, 01Z_0)$

$\vdash (q_1, 1, 1Z_0)$

$\vdash (q_1, \epsilon, Z_0)$

$\vdash (q_2, \epsilon, Z_0)$ Accepted

Therefore $(q_0, 1001, z_0) \vdash^* (q_1, 01, 01z_0) \vdash^* (q_2, \epsilon, Z_0)$.

Language of a PDA

Language of a PDA refers to the set of strings that is accepted by the PDA. The acceptance of any string by a PDA can be defined in two ways;

1) **Acceptance by final state:**

- According to this the Language of PDA is set of all string that causes PDA to enter in its accepting state after consuming all the alphabets of the string.

- Given a PDA P, the language accepted by final state, L(P) is;

  $\{w \mid (q, w, z_0) \vdash^* (q', \in, \gamma)\}$ where $q' \in F$ and $\gamma \in \Gamma^*$.

2) **Acceptance by empty stack:**

- According to this the Language of PDA is set of all string that causes PDA to enter in its accepting state after consuming all the alphabets of the string.
- Given a PDA P, the language accepted by empty stack, L(P), is

  $\{w \mid (q, w, z_0) \vdash^* (q', \in, \in)\}$ where $P \in Q$.

DETERMINISTIC PUSH DOWN AUTOMATA (DPDA)

A PDA is said to be deterministic if for every input alphabet 'a' (a may be ∈) and top of stack symbol,PDA has either unique transition(i.e moves to only one state.) or its transition is not defined.

Formally, A pushdown automata $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ is deterministic pushdown automata if the following two conditions are satisfied;

1) For any $q \in Q, X \in \Gamma$,

   If $\delta(q, a, X) \neq \phi$ for any $a \in \Sigma$ then $\delta(q, \in, X) = \phi$

   i.e. if $\delta(q, a, X)$ is non-empty for some a, then $\delta(q, \in, X)$ must be empty.

2) For any $q \in Q, a \in \Sigma \cup \{\in\}$ and $X \in \Gamma$,

   $\delta(q, a, X)$ has at most one element.

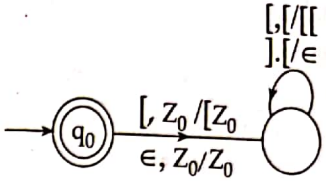
Example 5: Construct DPDA that accepts balanced parenthesis. i.e. equal no. of opening and closing braces. e.g: [[ ]], [ ][ ] etc.

Solution:

The Transition table for the above DPDA is given as:

Move number	State	Input	Stack Symbol	Move(s)
1	q_0	[	Z_0	$(q_1, [Z_0)$
2	q_1	[	[	$(q_1, [[)$
3	q_1	]	[	$(q_1, \in)$
4	q_1	∈	Z_0	(q_0, Z_0)

The transition diagram for this shown below:



Example 6: A DPDA accepting strings of balanced ordered parenthesis.

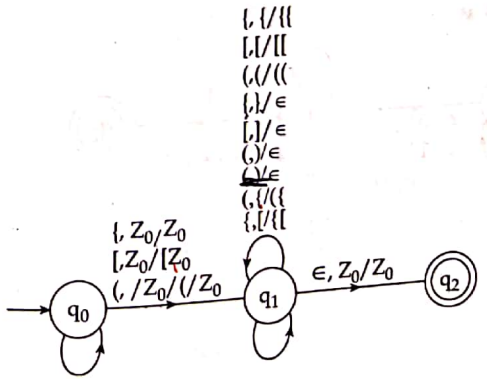
Solution:

A DPDA P can be constructed as:

$$P = \{Q = (q_0, q_1, q_2), \Sigma = \{\{,\}, (,), [,]\}, \Gamma = \Sigma \cup \{z_0\}, \delta, q_0, z_0, \{q_2\}\}$$

Where δ is defined as:

1. $\delta(q_0, \{, z_0) = (q_1, \{z_0)$
2. $\delta(q_0, [, z_0) = (q_1, [z_0)$
3. $\delta(q_0, (, z_0) = (q_1, (z_0)$
4. $\delta(q_1, \{, \{) = (q_1, \{\{)$
5. $\delta(q_1, [, [ ) = (q_1, [[ )$
6. $\delta(q_1, (, ( ) = (q_1, (( )$
7. $\delta(q_1, \{, \}) = (q_1, \in)$
8. $\delta(q_1,], [ ) = (q_1, \in)$
9. $\delta(q_1,), ( ) = (q_1, \in)$
10. $\delta(q_1, (, \{) = (q_1, (\{)$
11. $\delta(q_1, \{, [ ) = (q_1, \{[ )$
12. $\delta(q_1, \in, z_0) = (q_2, \in)$



Evaluate [[ ]]

$(q_0, [[\ ]], z_0) \vdash (q_1, [\ ]], [z_0)$

$\vdash (q_1,]], [[z_0)$

$\vdash (q_1,], [z_0)$

$\vdash (q1, \in, z_0)$

$\vdash (q_2, \in, \in)$ Accepted

Evaluate [[ ]]

$(q_0, [[\ ]], z_0) \vdash (q_1, [\,]], [z_0)$

$\vdash (q_1,]], [[z_0)$

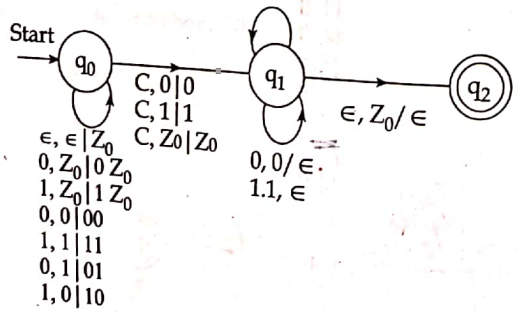
Stop (as $(q_1,], \{\ )$ is not defined) so not accepted.

Example 7: A PDA accepting language $L = \{w\ c\ w^R \mid w \in (0+1)^*\}$ is constructed as;

$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, z_0\}, \delta, q_0, z_0, \{q_2\})$ where δ is defined as;

1. $\delta(q_0, \in, \in) = (q_0, z_0)$
2. $\delta(q_0, 0, z_0) = (q_0, 0z_0)$
3. $\delta(q_0, 1, z_0) = (q_0, 1z_0)$
4. $\delta(q_0, 0, 0) = (q_0, 00)$
5. $\delta(q_0, 1, 1) = (q_0, 11)$
6. $\delta(q_0, 0, 1) = (q_0, 01)$
7. $\delta(q_0, 1, 0) = (q_0, 10)$
8. $\delta(q_0, C, 0) = (q_1, 0)$
9. $\delta(q_0, C, 1) = (q_1, 1)$
10. $\delta(q_0, C, z_0) = (q_1, z0)$
11. $\delta(q_1, 0, 0) = (q_1, \in)$
12. $\delta(q_1, 1, 1) = (q_1, \in)$
13. $\delta(q_1, \in, z_0) = (q_2, \in)$

The general notation is as:

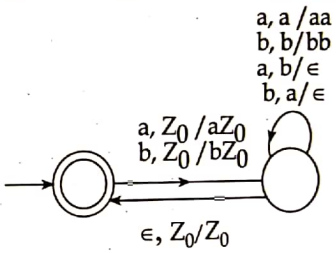


Example 8: Construct a DPDA that accepts all the string having equal no. of a's and b's.

Solution:

The transition table for above DPDA can be given as below

Move number	State	Input	Stack Symbol	Move(s)
1	q_0	a	Z_0	(q_1, aZ_0)
2	q_0	b	Z_0	(q_1, bZ_0)
3	q_1	a	a	(q_1, aa)
4	q_1	b	b	(q_1, bb)
5	q_1	a	b	$(q_1, \in)$
6	q_1	b	a	$(q_1, \in)$
7	q_1	$\in$	Z_0	(q_0, Z_0)



Finding the sequence of ID's for the string abbaaabb.

$(q_0, abbaaabb, Z_0) \vdash (q_1, bbaaabb, aZ_0)$

$\vdash (q_1, baaabb, Z_0)$

$\vdash (q_0, baaabb, Z_0)$

$\vdash (q_1, aaabb, b\ Z_0)$

$\vdash (q_1, aabb, Z_0)$

$\vdash (q_0, aabb, Z_0)$

$\vdash (q_1, abb, a\ Z_0)$

$\vdash (q_1, bb, aa\ Z_0)$

$\vdash (q_1, b, a\ Z_0)$

$\vdash (q_1, \in, Z_0)$

$\vdash (q_0, \in, Z_0)$ Hence accepted.

Non Deterministic PDA (NPDA)

A DFA (or NFA) is not powerful enough to recognize many context-free languages because a DFA can't count. But counting is not enough -- consider a language of palindromes, containing strings of the form WWR. Such a language requires more than an ability to count; it requires a stack.

A nondeterministic pushdown automaton (npda) is basically an nfa with a stack added to it.

We start with the formal definition of an NFA, which is a 5-tuple, and add two things to it:

Γ is a finite set of symbols called the *stack alphabet*, and

$z \in \Gamma$ is the stack start symbol

We also need to modify δ, the transition function, so that it manipulates the stack

A nondeterministic pushdown automaton or npda is a 7-tuple

$M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$

Q is a finite set of states,

Σ is a the input alphabet,

Γ is the stack alphabet,

δ is a transition function,

$q_0 \in Q$ is the initial state,

z

$\in \Gamma$ is the stack start symbol, and

$F \subseteq Q$ is a set of final states.

Transition Functions for NPDAs

The transition function for an npda has the form

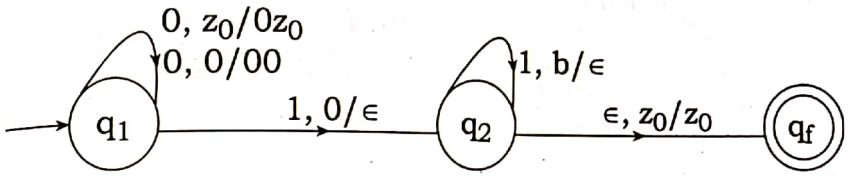
D: $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow$ finite subsets of $Q \times \Gamma^*$

Construction of PDA by Final State

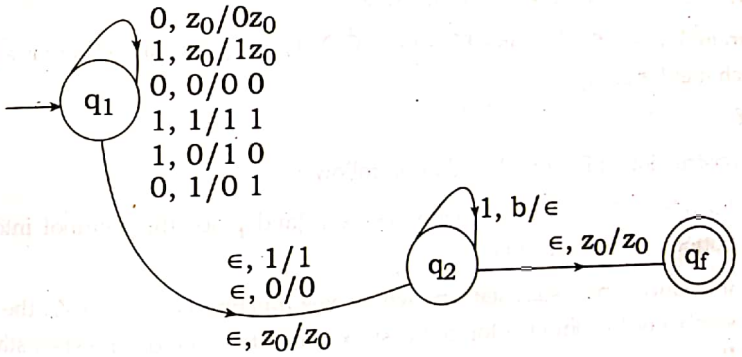
Let P= (Q,Σ,Γ, δ, q0, Z0, F)

be a PDA. The language accepted by P by final state is: $L(P) = \{w \mid (q0, w, Z0) \vdash^* (q, \in, \alpha), q \in F\}$ for some state q in F and any input stack string α. Starting in the initial ID with w waiting on the input, P consumes w from the input and enters an accepting state. The contents of the stack at that time is irrelevant.

Example 1: PDA with final state with equal number of zeros (0's) followed by equal numbers ones (1's).



Example 2: PDA with final state that accept the palindrome

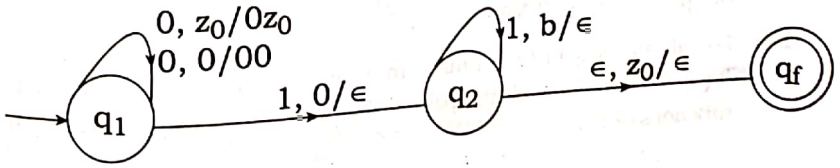


Construction of PDA by Empty Stack

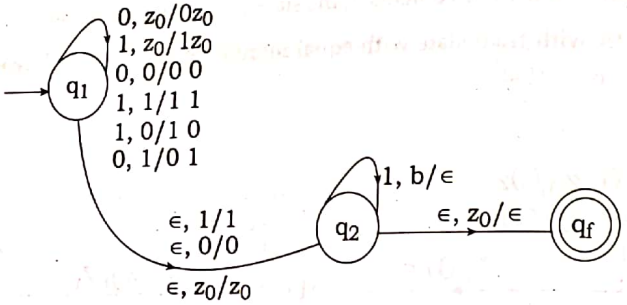
PDA- Acceptance by empty stack:

Let P= (Q,Σ,Γ, δ, q0, Z0, F)be a PDA. The language accepted by P by empty stack is:$N(P) = \{w \mid (q0, w, Z0) \vdash^* (q, \in, \in)\}$where qis any state N(P) is the set of inputs w that P can consume an at the same time empty the stack.

Example 3: PDA with empty stack with equal number of zeros (0's) followed by equal numbers ones (1's).



Example 4: PDA with empty stack that accept the palindrome.



Conversion of PDA accepting by Empty stack to accepting by final state

Let us consider the PDA P_N that accepts a language L by empty stack and P_F is its equivalent PDA that accepts L by final state.

Theorem: If $L = L(P_N)$ for some PDA $P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0)$, then there is a PDA P_F such that $L = L(P_F)$.

Proof:

The construction of P_F from P_N is done as follows:

1. Introduce the a new symbol X_0 ($X_0 \notin \Gamma$)and place this symbol into the bottom of the stack of P_F.
2. Introduce a new start state, p_0, whose sole function is to push Z_0, the start symbol of P_N, onto the top of the stack and enter state q_0 (the start state of P_N).
3. Copy the other states and transitions functions of P_N in P_F.
4. Finally add another new state p_f, which is the accepting state of P_F ; this PDA transfers to state p_f whenever it discovers that P_N would have emptied its stack.

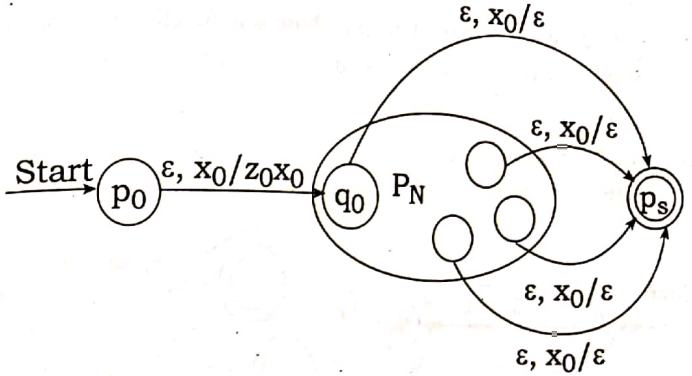
Thus the new constructed P_F has the components:

$$P_N = (Q \cup \{p_0, p_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta_F, p_0, X_0, \{p_f\})$$

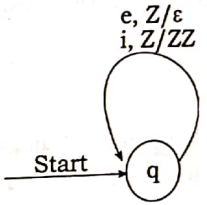
Where δ_F is defined by:

1. $\delta_F(p_0, \in, X_0) = (q_0, Z_0X_0)$
2. For all states q in Q, inputs a in Σ or a= $\in$, and stack symbols Y in Γ, $\delta_F(q, a, Y)$ contains all the pairs in $\delta_N(q, a, Y)$. i.e. Copy all the transition functions of P_N.

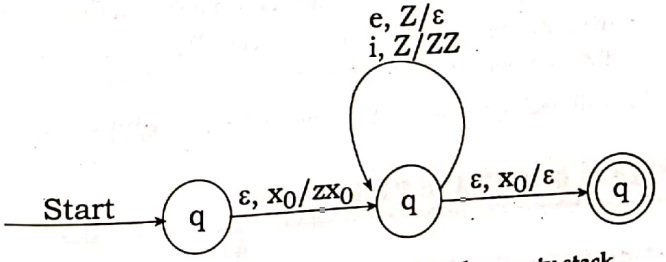
3. Add another transition function $\delta_F(q, \in, X_0) = (p_f, \in)$ for every state q in Q.



Example 12: Convert the following PDA that accepts the string using empty stack into its equivalent PDA that accepts the string by entering into the accepting state.



Solution:



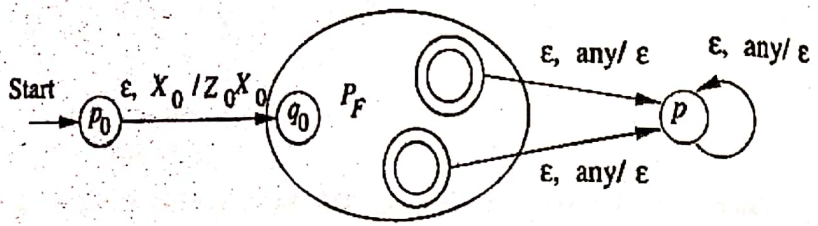
Conversion of PDA accepting by final state to accepting by empty stack

Theorem: Let L be $L(P_F)$ for some PDA $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$. Then there is a PDA P_N such that $L = N(P_N)$.

Proof:

The construction of $P_N$from P_F is done as follows:

1. Introduce the a new symbol X_0 ($X_0 \notin \Gamma$) and place this symbol into the bottom of the stack of P_N.
2. Introduce a new start state, p_0, whose sole function is to push Z_0, the start symbol of P_F, onto the top of the stack and enter state q_0 (the start state of P_F).
3. each accepting state of P_F, add a transition on ϵ to a new state p. When in state p, P_N pops its stack and does not consume any input.



Thus the new constructed P_N has the components:

$$P_N = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{X_0\}, \delta_N, p_0, X_0)$$

Where δ_N is defined by:

1. $\delta_N(p_0, \epsilon, X_0) = \{(q_0, Z_0X_0)\}$.
2. For all states q in Q, inputs a in $\sum$ or a= ϵ, and stack symbols Y in Γ, $\delta_N(q, a, Y)$ contains all the pairs in $\delta_F(q, a, Y)$. i.e. Copy all the transition functions of P_N.
3. For all accepting states q in F and stack symbols Y in Γ or $Y = X_0$, $\delta_N(q, \epsilon, Y) = (p, \epsilon)$. By this rule, whenever P_F accepts, P_N can start emptying its stack without consuming any more input.
4. For all stack symbols Y in Y in Γ or $Y = X_0$, $\delta_N(q, \epsilon, Y) = (p, \epsilon)$.

EQUIVALENCE OF CFG AND PDA

Converting CFG into its Equivalent PDA

Given a CFG, G = (V, T, P,S), we can construct a push down automaton, P which accepts the language generated by the grammar G, i.e. L(M) = L (G).

The PDA M can be defined as;

M = ({q}, T, V U T, δ, q, S, Φ)

Where,

Q = {q} is only the state in the PDA.

Σ = T

Γ = V U T (i.e. PDA uses terminals and variables of G as stack symbols)

z_0 = S (initial stack symbol is start symbol in grammar)

F = Φ

δ can be defined as;

i. $\delta(q, \epsilon, A) = \{q, \alpha) / A \rightarrow \alpha$ is a production P of G.
ii. $\delta(q, a, a) = \{(q, \epsilon)$, for all a ε T}

Alternatively, we can define push down automata for the CFG with two states p & q, where p being start state. Here idea is that the stack symbol initially is supposed to be ∈, and at first PDA starts with state p and reading ∈, it inserts start symbol 'S' of CFG into stack and changes the state to q.

Then all transitions occur in state q.

i.e. the PDA can be defined as;

P= ({p,q}, T, V U T, δ, p, {q}); stack top is ∈.

Then δ can be defined as;

$\delta(p, \epsilon, \epsilon) = \{(q, S) / S$ is start symbol of grammar G.

$\delta(p, \epsilon, A) = \{(q, \alpha) / A \rightarrow \alpha$ is a production P of G.

$\delta(p, a, a) = \{(q, \epsilon)$, for all a ε Σ.

Example 13: Consider an example;

Let G = (V, T, P and S) where, productions in G denoted by P are:

$E \rightarrow T \mid E + T$

$T \rightarrow F \mid T^*F$

$F \rightarrow a \mid (E)$

We can define a PDA equivalent to this grammar as;

$M = (\{q_0\}, \{a, *, +, (,)\}, \{a, *, +, (,), E, T, F\}, \delta, q_0, E, \Phi\}$

Where δ can be defined as;

$\delta(q_0, \epsilon, E) = \{(q_0, T), (q_0, E + T)\}$

$\delta(q_0, \epsilon, T) = \{(q_0, F), (q_0, T^*F)\}$

$\delta(q_0, \epsilon, F) = \{(q_0, a), (q_0, (E))\}$

$\delta(q_0, a, a) = \{(q_0, \in)\}$

$\delta(q_0, *, *) = \{(q_0, \in)\}$

$\delta(q_0, +, +) = \{(q_0, \in)\}$

$\delta(q_0, (, () = \{(q_0, \in)\}$

$\delta(q_0,),)) = \{(q_0, \in)\}$

Now with this PDA, M, let us trace out acceptance of a + (a*a);

$(q_0, a + (a*a), E) \vdash (q_0, a + (a*a), E + T)$

$\vdash (q_0, a + (a*a), T + T)$

$\vdash (q_0, a + (a*a), F + T)$

$\vdash (q_0, a + (a*a), a + T)$

$\vdash (q_0, + (a*a), + T)$

$\vdash (q_0, (a*a), T)$

$\vdash (q_0, (a*a), F)$

$\vdash (q_0, (a*a), (E))$

$\vdash (q_0, a*a), E))$

$\vdash (q_0, a*a), T))$

$\vdash (q_0, a*a), T*F)$

$\vdash (q_0, a*a), F*F)$

$\vdash (q_0, a*a), a*F)$

$\vdash (q_0, *a), *F)$

$\vdash (q_0, a), F))$

$\vdash (q_0, a), a))$

$\vdash (q_0,),))$

$\vdash (q_0, \in, \in)$ Accepted (acceptance by empty stack).

In CFG

$E \rightarrow E + T$

$\rightarrow E + T$

$\rightarrow T + T$

$\rightarrow F + T$

$\rightarrow a + T$

$\rightarrow a + F$

$\rightarrow a + (E)$

$\rightarrow a + (T*F)$

$\rightarrow a + (F*F)$

$\rightarrow a + (a*F)$

$\rightarrow a + (a*a)$

Example 14: Convert the grammar defined by following production into PDA;

S→ 0S1 | A

A→ 1S0 | S | ∈

Solution:

Let G = (V, T, P and S) defined by following productions;

S→ 0S1 | A

A→ 1S0 | S | ∈

PDA equivalent top this grammar as

$M = (\{q_0\}, \{0, 1\}, \{0, 1, S, A\}, \delta, q_0, S, \Phi)$

Where; $Q = \{q_0\}$

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

$\Gamma = \{0, 1, S, A\}$

$z_0 = S$

$F = \Phi$

And δ is defined as;

$\delta(q_0, \in, S) = \{(q_0, 0S1), (q_0, A)\}$

$\delta(q_0, \in, A) = \{(q_0, 0S1), (q_0, S), (q_0, \in)\}$

$\delta(q_0, 0, 0) = \{(q_0, \in)\}$

$\delta(q_0, 1, 1) = \{(q_0, \in)\}$

Example 15: Construct a PDA equivalent to following grammar defined by;

S→aAA

A→aS | bS | a

Solution:

Let G = (V, T, P and S) be the grammar defined by the production;

S→aAA

A→aS | bS | a

Now, we can define the PDA equivalent to the grammar as;

$M = (\{q_0\}, \{a, b\}, \{a, b, S, A\}, \delta, q_0, S, \Phi)$

Where δ is defined as

$\delta(q_0, \in, S) = \{q_0, aAA\}$

$\delta(q_0, \in, A) = \{(q_0, aS), (q_0, bS), (q_0, a)\}$

$\delta(q_0, a, a) = \{q_0, \in\}$

$\delta(q_0, b, b) = \{q_0, \in\}$

Now, we trace acceptance of aaabaaaaa

$(q_0, aaabaaaaa, S) \vdash (q_0, aaabaaaaa, aA)$

$\vdash (q_0, aabaaaaa, A)$

$\vdash (q_0, aabaaaaa, AA)$

$\vdash (q_0, aabaaaaa, aSA)$

$\vdash (q_0, abaaaaa, SA)$

$\vdash (q_0, abaaaaa, aAAA)$

$\vdash (q_0, baaaaa, AAA)$

$\vdash (q_0, baaaaa, bSAA)$

$\vdash (q_0, aaaaa, SAA)$

$\vdash (q_0, aaaaa, aAAAA)$

$\vdash (q_0, aaaa, AAAA)$

$\vdash (q_0, aaaaa, aAAA)$

$\vdash (q_0, aaa, AAA)$

$\vdash (q_0, aaa, aAA)$

$\vdash (q_0, aa, AA)$

$\vdash (q_0, aa, aA)$

$\vdash (q_0, a, A)$

$\vdash (q_0, a)$

In CFG

$S \rightarrow aAA \rightarrow aaSA$

$\rightarrow aaaAAA$

$\rightarrow aaabSAA$

$\rightarrow aaabaAAAA$

$\rightarrow aaabaaAAA$

$\rightarrow aaabaaaAA$

$\rightarrow aaabaaaaA$

$\rightarrow aaabaaaaa$

Converting PDA into its equivalent CFG

Given a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$; $F = \Phi$, we can obtain an equivalent CFG, $G = (V, T, P \text{ and } S)$ which generates the same language as accepted by the PDA M.

The set of variables in the grammar consists of;

- The special symbol S, which is start symbol.
- All the symbols of the form $[p \times q]$; $p,q \in Q$ and $X \in \Gamma$

i.e. $V = \{S\} \cup \{[p\ X\ q]\}$

The terminal in the grammar $T = \Sigma$

The production of G is as follows;

- For all states $q \in Q$, $S \rightarrow [q_0, z_0, q]$ is a production of G.
- For any states $q, r \in Q$, $X \in \Gamma$ and $a \in \Sigma \cup \{\in\}$

If $\delta(q, a, X) = (p, \in)$ then $[p\ X\ q] \rightarrow a$

- For any states $q, r \in Q$, $X \in \Gamma$ and $a \in \Sigma \cup \{\in\}$,

If $\delta(q, a, x) = (r, Y_1Y_2 \ldots\ldots Y_k)$; where $Y_1, Y_2, \ldots\ldots Y_k \in \Gamma$ and $k \geq 0$

Then for all lists of states $r_1, r_2, \ldots\ldots\ldots r_k$, G has the production $[p\ X\ q] \rightarrow a\ [rY_1r_1][r_1Y_2r_2]\ldots\ldots\ldots\ldots[r_{k-1}Y_kr_k]$

This production says that one way to pop X and go from stack q to state r_k is to read "a" (which may be $\in$), then use some input to pop Y_1 off the stack while going from state r to r_1, then read some more input that pops Y_2 off the stack and goes from r_1 to r_2 and so on.....

Note: Each of the variables [q X r] represents an event in PDA.

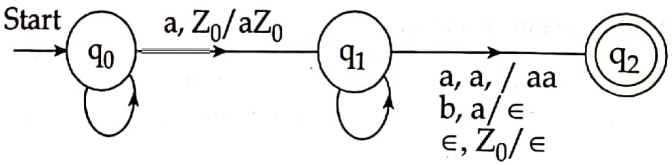
- The next popping of some symbol x from the stack,

- A change in state from some r at beginning to a q when x has finally been replaced by ∈ on the stack.

Example 16: PDA that recognizes a language

$L = \{a^n b^n \mid n > 0\}$ be defined as;

1. $\delta(q_0, a, z_0) = (q_1, az_0)$
2. $\delta(q_1, a, a) = (q_1, aa)$
3. $\delta(q_1, b, a) = (q_1, \in)$
4. $\delta(q_1, \in, z_0) = (q_1, \in)$



Let G = (V, T, P and S) be the equivalent CFG for the given PDA where,

V= {S} U {[p × q] / p, q ε Q, x ε Γ}

S= is the start state.

T= Σ

And P is defined by following production;

1. $S \rightarrow [q_0 z_0 q_0] \mid [q_0 z_0 q_1]$
   i.e. $S \rightarrow [q_0 z_0 r_2]$; for r_2 in $\{q_0, q_1\}$
2. From the fact that $\delta(q_0, a, z_0)$ contains (q_1, az_0), we get production
   $[q_0 z_0 r_2] \rightarrow a[q_1 ar_1][r_1 z_0 r_2]$; for r_i in $\{q_0, q_1\}$
3. From the fact that $\delta(q_1, a, a)$ contains (q_1, aa), we get productions
   $[q_1 ar_2] \rightarrow a[q_1 ar_1][r_1 a r_2]$ for r_i in $\{q_0, q_1\}$
4. From the fact that $\delta(q_1, b, a)$ contains $(q_1, \in)$, we get
   $[q_1 a q_1] \rightarrow b$
5. From the fact that $\delta(q_1, \in, z_0)$ contains $(q_1, \in)$, we get
   $[q_1 z_0 q_1] \rightarrow \in$

Now acceptance of aaabbb can be shown as;

$S \rightarrow [q_0 z_0 r_2]$

$\rightarrow a [q_1 a r_1] [r_1 z_0 r_2]$

$\rightarrow aa [q_1 a r_1] [r_1 a r_2] [r_1 z_0 r_2]$

$\rightarrow aaa[q_1 a r_1] [r_1 ar_2] [r_1 ar_2] [r_1 z_0 r_2]$

$\rightarrow aaab[r_1 ar_2] [r_1 ar_2] [r_1 z_0 r_2]$

$\rightarrow aaabb[r_1 ar_2] [r_1 z_0 r_2]$

$\rightarrow aaabbb[r_1 z_0 r_2]$

$\rightarrow aaabbb \in = aaabbb$

Example 17: Convert the PDA $P = (\{p, q\}), \{0, 1\}, \{x, z_0\}, \delta, q, z_0)$ to a CFG if δ is given by

- $\delta(q, 1, z_0) = (q, xz_0)$
- $\delta(q, 1, x) = (q, xx)$
- $\delta(q, 0, x) = (q, x)$
- $\delta(q, \in, z_0) = (q, \in)$
- $\delta(q, 1, x) = (p, \in)$
- $\delta(q, 0, z_0) = (q, z_0)$

Now, the equivalent grammar can be written as;

G = (V, Σ, P, S) where P consists of productions as;

$S \rightarrow [qz_1r_2]$; r_2 in {p, q}

$[qz_0r_2] \rightarrow 1[q \times r_1] [r_1 z_0r_2]$; for r_i in {p,q}

$[q \times r_2] \rightarrow 1[q \times r_1] [r_1 \times r_2]$; for r_i in {p,q}

$[q \times r_2] \rightarrow 0[p \times r_2]$; for r_2 in {p,q}

$[q \times q] \rightarrow \in$

$[p \times p] \rightarrow 1$

$[pz_0r_2] \rightarrow 0[qz_0r_2]$; for r_2 in {p,q}

Example 18: Convert the following PDA into CFG.

P = ({q}, {i, e}, {X,Z}, δ, q, Z), where δ is given by:

$\delta(q, i, Z) = \{(q, XZ)\}$, $\delta(q, e, X) = \{(q, \in)\}$ and $\delta(q, \in, Z) = \{(q, \in)\}$

Solution:

Equivalent productions are:

$S \rightarrow [qZq]$

$[qZq] \rightarrow i[qXq][qZq]$

$[qXq] \rightarrow e$

$[qZq] \rightarrow \varepsilon$

If [qZq] is renamed to A and [qXq] is renamed to B, then the CFG can be defined by:

$G = (\{S, A, B\}, \{i, e\}, \{S \rightarrow A, A \rightarrow iBA \mid \varepsilon, B \rightarrow e\}, S)$

Example 19: Convert PDA to CFG. PDA is given by P = ({p,q}, {0,1}, {X, Z}, δ, q, Z)), Transition function δ is defined by:

$\delta(q, 1, Z) = \{(q, XZ)\}$

$\delta(q, 1, X) = \{(q, XX)\}$

$\delta(q, \varepsilon, X) = \{(q, \varepsilon)\}$

$\delta(q, 0, X) = \{(p, X)\}$

$\delta(p, 1, X) = \{(p, \varepsilon)\}$

$\delta(p, 0, Z) = \{(q, Z)\}$

Solution:

Add productions for start variable

$S \rightarrow [qZq] \mid [qZp]$

For $\delta(q, 1, Z) = \{(q, XZ)\}$

$[qZq] \rightarrow 1[qXq][qZq]$

$[qZq] \rightarrow 1[qXp][pZq]$

$[qZp] \rightarrow 1[qXq][qZp]$

$[qZp] \rightarrow 1[qXp][pZp]$

For $\delta(q, 1, X) = \{(q, XX)\}$

$[qXq] \rightarrow 1[qXq][qXq]$

$[qXq] \rightarrow 1[qXp][pXq]$

$[qXp] \rightarrow 1[qXq][qXp]$

$[qXp] \rightarrow 1[qXp][pXp]$

For $\delta(q, \varepsilon, X) = \{(q, \varepsilon)\}$

$[qXq] \rightarrow \varepsilon$

For $\delta(q, 0, X) = \{(p, X)\}$

$[qXq] \rightarrow 0[pXq]$

$[qXp] \rightarrow 0[pXp]$

For $\delta(p, 1, X) = \{(p, \varepsilon)\}$

$[pXp] \rightarrow 1$

For $\delta(p, 0, Z) = \{(q, Z)\}$

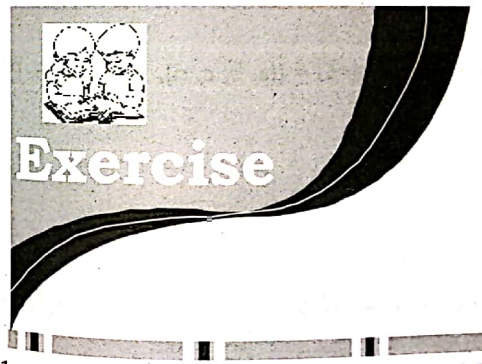
$[pZq] \rightarrow 0[qZq]$

$[pZp] \rightarrow 0[qZp]$

Renaming the variables [qZq] to A, [qZp] to B, [pZq] to C, [pZp] to D, [qXq] to E [qXp] to

F, [pXp] to G and [pXq] to H, the equivalent CFG can be defined by:

G = ({S, A, B, C, D, E, F, G, H}, {0,1}, R, S). The productions of R also are to be renamed accordingly.



1. Describe instantaneous description of PDA. Configure a PDA for balanced parentheses, i.e. $\sum = \{ [, (,),] \}$.
2. Configure a Pushdown automaton accepting the language, $L = \{wCw^R \mid w \in (a,b)^*\}$. Show instantaneous description of strings abbCbba and baCba.
3. Configure a DPDA accepting the language $L = \{ (^n C)^n \}$. Show the instantaneous description of ((C)) and (((C)).
4. Construct a PDA L = {w ∈ {0, 1}* starts and end with same symbol }.
5. Construct a PDA that accepts string of the form {w ∈ {0, 1}* contains more 1's than 0's}.
6. Construct a PDA with recognizes the language of arithmetical expression with the following $\sum = \{0, 1, +, *, (,)\}$.

7. Configure a Push Down Automaton, with 7-tuples, for the language defined by following grammar. Also draw the state diagram.

   ]

   $S \rightarrow aSbS \mid bSaS \mid \epsilon$

8. Consider a PDA that accepts by empty stack, P= ({p, q}, {(,)}, {Z}, δ, p, z); with δ defined as $\delta(p, (, z) = (p, (z)$, $\delta(p, (, ( ) = (p, (( )$, $\delta(p,), ( ) = (p, \epsilon)$, $\delta(p, \epsilon, z) = (q, \epsilon)$. Now construct an equivalent CFG. Also show the derivation of (( )) using the productions of so constructed CFG

9. Construct a push down automaton for the following grammar. Also draw the state diagram.

   $S \rightarrow 1A \mid 0B \mid \epsilon$

   $A \rightarrow 1AA \mid 0S \mid 0$

   $B \rightarrow 0BB \mid 1 \mid A$

10. Construct a PDA from the following CFG

    G = ({S, X}, {a, b}, P, S)

    Where the production are

    $S \rightarrow XS \mid \epsilon$

    $A \rightarrow aX \mid Ab \mid ab$

11. Obtain the PDA corresponding to the grammar G = ({a, b, c, d}, {s, A, B}, S, P) with the following production rule:

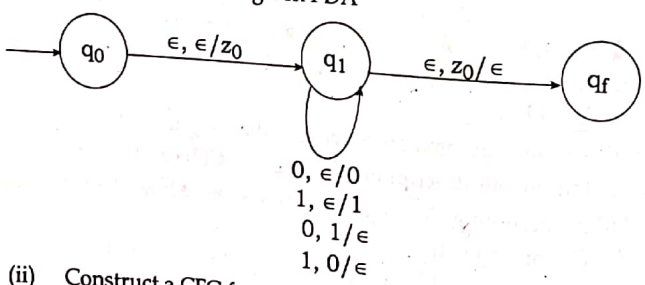
    $S \rightarrow aSB \mid bA \mid b \mid d$

    $A \rightarrow bA \mid b$

    $B \rightarrow C$

12. Conversion PDA to CFG

    (i) Construct CFG for the given PDA



    (ii) Construct a CFG for the following language:

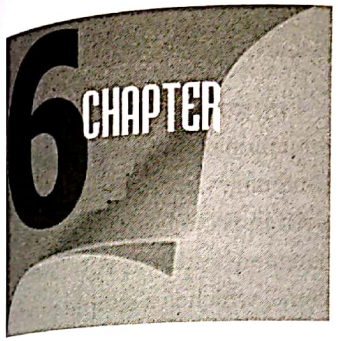
    (a) $L = \{a^n b^n \mid n \geq 0\}$

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

    (c) $L = \{a^{n+1} b^n \mid n > 0\}$

    (d) $L = \{a^{2n+1} b^n \mid n > 0\}$

❑❑❑



TURING MACHINE

CHAPTER OUTLINES

After studying this Chapter you should be able to:

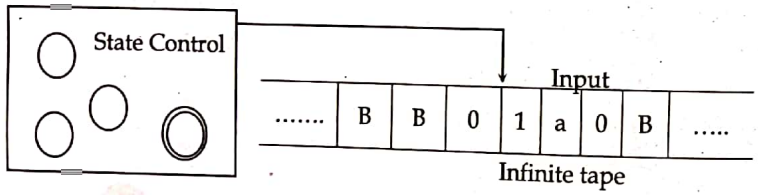
- Introduction to Turing Machine
- Instantaneous Description for TM
- Turing Machine as a Language Recognizer
- Turing Machine with Storage in the State
- Turing Machine as Enumerator of Strings of a Language
- Turing Machine with Multiple Tracks
- Equivalence of One-tape and Multi-tape TM's
- Church Thesis and Algorithm
- Encoding of Turing Machine

INTRODUCTION TO TURING MACHINE

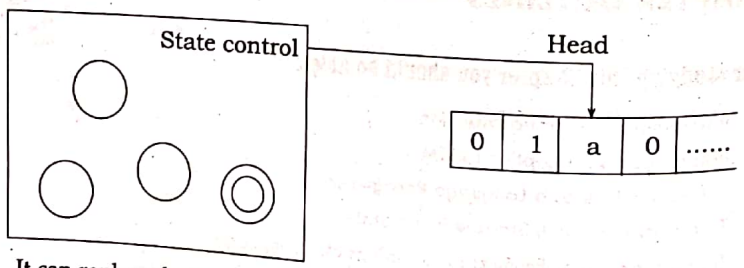
Introduction

In the previous chapters, we have seen several computational devices that can be used to accept or generate regular and context-free languages. Even though these two classes of languages are fairly large, we have seen in Section 3.8.2 that these devices are not powerful enough to accept simple languages such as $A = \{a^n b^n c^n : n \geq 0\}$. In this chapter, we introduce the Turing machine, which is a simple model of a real computer. Turing machines can be used to accept all context-free languages, but also languages such as A. Furthermore, every problem that can be solved on a real computer can also be solved by a Turing machine (this statement is known as the Church-Turing Thesis).

A Turing Machine is a finite automaton with a two-way access to an infinite tape.

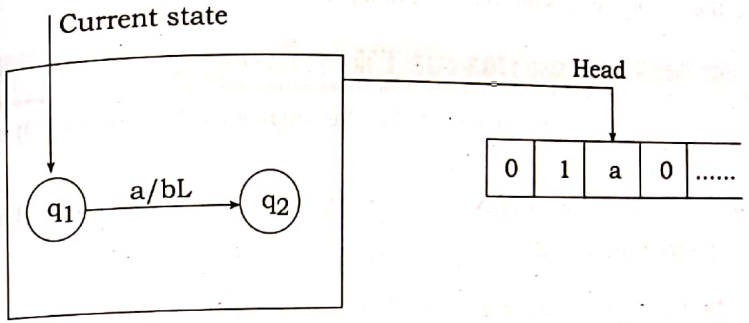


- Initially, thefew cells contain the input, and the other cells are blank.
- At each point in the computation, the machine sees its current state and the symbol at the head

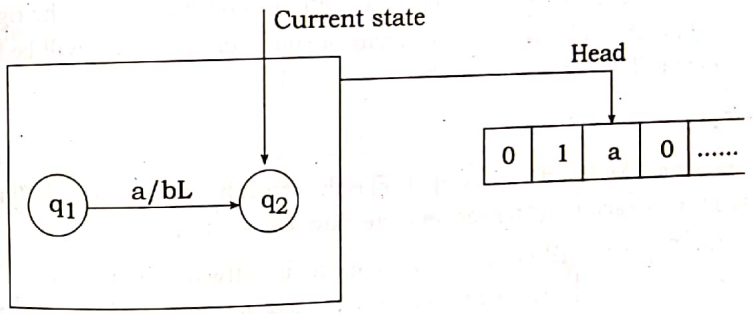


- It can replace the symbol on the tape, change state, and move the head left or right.

Example 1: Consider the following state where it reads symbol a.



Let it replace a with b, and move head left which can be shown in figure below as:



Formal Definition of Turing Machine

A Turing Machine TM is defined by the seven-tuples, $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where,

Q = the finite set of states of the finite control

Σ = the finite set of input symbols

Γ = the complete set of tape symbols Σ is always subset of Γ.

q_0 = the start state; $q_0 \varepsilon Q$

B = the blank symbol; $B \varepsilon \Gamma$ but B does not belong to Σ.

F = the set of final or accepting states; $F \subseteq Q$

δ = the transition function defined by

$Q \times \Gamma \rightarrow Q \times \Gamma \times (R, L, S)$; where R, L, S is the direction of movement ofhead-left, or right or stationary.

i.e. $\delta(q, x) = (P, Y, D)$; which means TM in state q and current tape symbol x, moves to next state P, replacing tape symbol x with Y and move the head either direction or remains at same cell of input tape.

INSTANTANEOUS DESCRIPTION FOR TM

The configuration of a TM is described by Instantaneous description (ID) of TM as like PDA.

A string $x_1x_2.....x_{i-1}qx_ix_{i+1}.....x_n$ represents the I.D. of TM in which;

- q is the state of T M .
- the tape head scanning the i^{th} symbol from the left.
- $x_1x_2.....x_n$ is the portion of tape between the leftmost and rightmost non-blank.(If the head is to the left of leftmost non blank or to the right of rightmost non-blank then some prefix or suffix of $x_1x_2.....x_n$ will be blank and i will be 1 or n respectively.)

Moves of T M

The moves of TM, $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is described by the notation $\vdash$, "yield", for single move and by $\vdash^*$ for zero or more moves as in PDA.

a. For $\delta(q, x_i) = (P, Y, L)$ i.e. next move is leftward then, $x_1x_2.....x_{i-1}qx_ix_{i+1}.....x_n \vdash x_1x_2....x_{i-2}\ P\ x_{i-1}Yx_{i+1}.xn$ reflects the change of state from q to p and the replacement of symbol x_i with Y and then head is positioned at i-1 (next scan is x_{i-1}).

   i. If i=1, M moves to the left of x_1 i.e. $qx_1x_2.....x_n \vdash pBYx_2.....x_n$

   ii. If i=n, Y = B, then M moves to state p and system B written over x_n joins the infinite sequence of trailing blanks which does not appear in next ID as $x_1x_2.....x_{n-1}qxn \vdash x_1x_2.....x_{n-2}px_{n-1}$

b. If $\delta(q, x_i) = (P, Y, R)$ i.e. next move is rightward then, $x_1x_2.....x_{i-1}qx_i.....x_n \vdash x_1x_2....x_{i-1}\ Y\ p\ x_{i+1}.....x_n$, which reflects that the symbol xi is replaced with Y and head has moved to cell i+1 with change in state from p to q.

   - If i= n, then i+1 cell holds blank which is not part of previous ID; i.e. $x_1x_2.....x_{n-1} \vdash x_1x_2..... x_{n-1}\ YpB.$
   - If i=1, Y=B, then the symbol B written over xi joins the infinite sequence of leading blanks and does not appear in next ID; i.e. $x_1x_2.....x_n \vdash px_2x_3.....x_n.$

Example 2: Consider an example; A TM accepting $\{0^n1^n / n \geq 1\}$

- Given finite sequence of 0's and 1's on its tape preceded and followed by blanks.
- The TM will change 0 to an X and then a 1 to Y_1 until all 0's and 1's are matches.
- Starting at left end of the input, it repeatedly changes a 0 to an X and moves to the right over whatever 0's and Y's it sees until comes to a 1.
- It changes 1 to a Y, and moves left, over Y's and 0's until it finds X. At that point, it looks for a 0 immediate to the right. If finds one 0 then changes it to X and repeats the process changing a matching 1 and Y.

Now, TM will have;

$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$

The transition rule for the move of M is described by following transition table;

	0	1	X	Y	B
q_0	(q_1, X, R)			(q_3, Y, R)	
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	
q_3				(q_3, Y, R)	(q_4, B, R)
q_4					

Now, the acceptance of 0011 by the TM, M1 is described by following sequence of moves;

$q_0\,0011 \vdash Xq_1011 \vdash X0q_111 \vdash Xq_20Y1$
$\vdash q_2X0Y1$
$\vdash Xq_00Y1$
$\vdash XXq_1Y1$
$\vdash XXYq_11$
$\vdash XXq_2YY$
$\vdash Xq_2XYY$
$\vdash XXq_0YY$
$\vdash XXYq_3Y$
$\vdash XXYYq_3B$
$\vdash XXYYBq_4B$ Halt and accept.

For string 0110

$q_0 0110 \vdash X q_1 110 \vdash q_2 XY10 \vdash X q_0 Y10$

$\vdash XY q_3 10$

Halt and reject; since q_3 has no move on symbol 1.

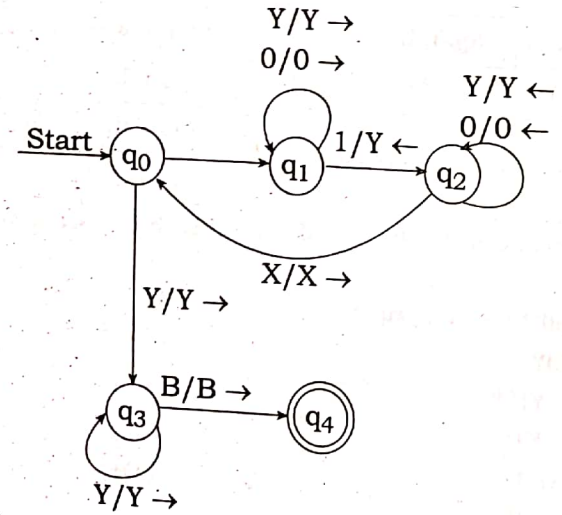
Transition diagram for a TM

A transition diagram of TM consists of,

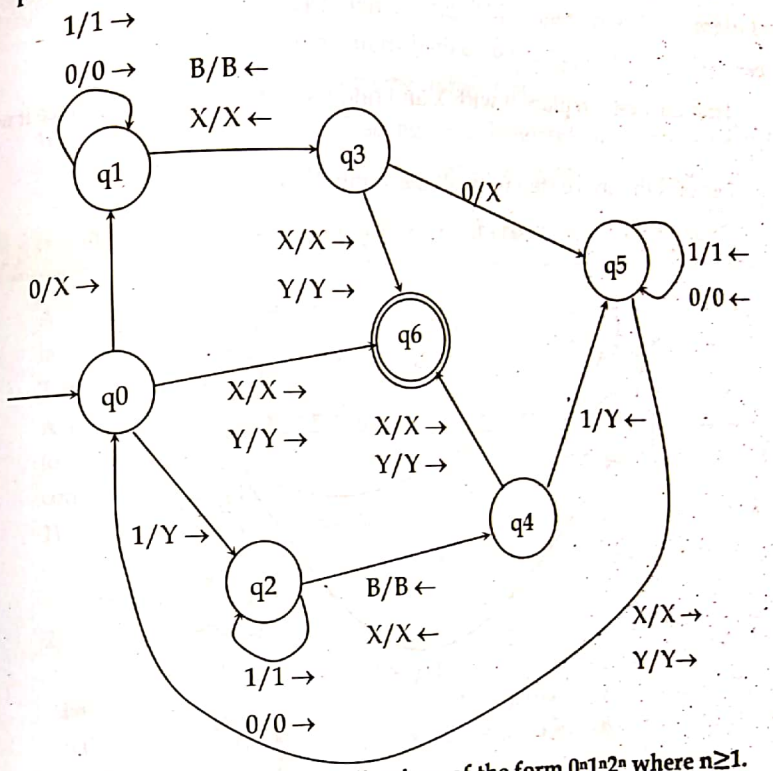
- A set of nodes representing states of TM.
- An arc from any state, q to p is labeled by the items of the form X / YD, where X and Y are tape symbols, and D is a direction, either L or R. that is, whenever δ(q, x) = (P, Y, D), we find the label X / YD on the arc from q to p.

However, in diagram, the direction D is represented by ← for left (L) and → for right (R)

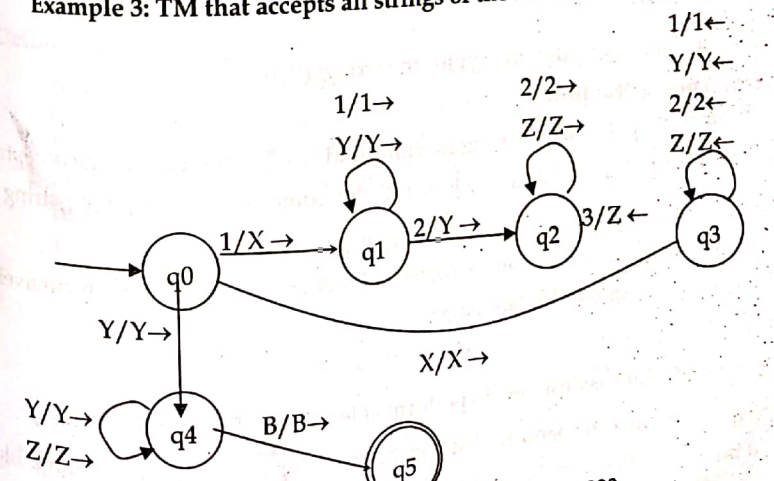
Thus, transition diagram for the TM for L = {0^n1^n / n ≥ 1} as;



Example 2: Transition diagram for the TM which accepts the set of all palindromes over {0,1}



Example 3: TM that accepts all strings of the form $0^n1^n2^n$ where n≥1.



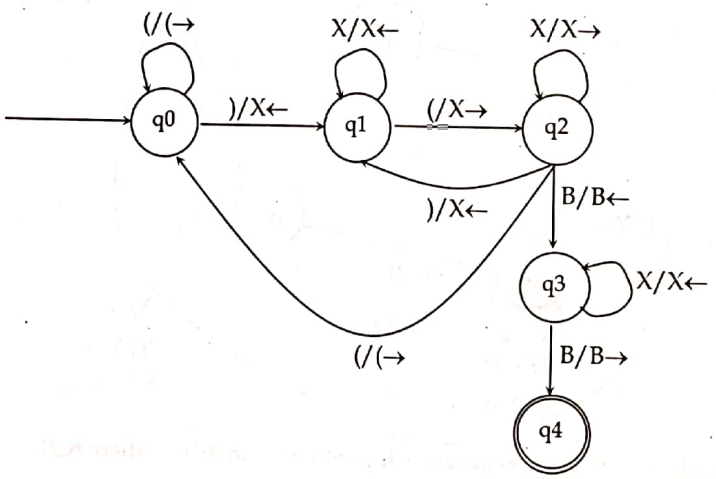
Show the instantaneous description for the string 112233.

Example 4: Design a TM that accepts well formed string of parenthesis. i.e. L = {(()),()(()),(())(),.....}.

Solution:

Idea:

- Find the first), replace it with X and find its corresponding (replace it with X.
- Perform the above step until all the symbols are scanned.
- Finally if the tape consist of only X or B then accept otherwise not.



Show the instantaneous description for the string: (()()).

Language of Turing Machine

If T= (Q, Σ, Γ, δ, q_0, B, F) is a Turing machine and w ε Σ *, then language accepted by T, L (T) = {w | w ε Σ* and $q_0 w \vdash^* \alpha\ p\ \beta$} for some pεF and any tape string α and β.

The set of languages that can be accepted using TM are called recursively enumerable languages or RE languages.

Role of TM

The Turing Machine is designed to perform at least the following three roles;

1) As a language recognizer: TM can be used for accepting a language like Finite Automaton and Pushdown Automata.

2) As a Computer of function: A TM represents a particular function. Initial input is treated as representing an argument of the function. And the final string on the tape when the TM enters the halt state; treated as representative of the value obtained by an application of the function to the argument represented by the initial string.

3) As an enumerator of string of a language: It outputs the strings of a language, one at a time in some systematic order that is as a list.

TURING MACHINE AS A LANGUAGE RECOGNIZER

A Turing Machine can be used a language reconizer to accept strings of certain language. For example, A TM accepting $\{0^n1^n / n \geq 1\}$ as mentioned previously.

Turing Machine as a Computing a Function:

A Turing Machine can be used to compute functions. For such TM, we adopt the following policy to input any string to the TM which is an input of the computation function.

1) The string w is presented into the form BwB, where B is a blank symbol, and placed onto the tape; the head of TM is positioned at a blank symbol which immediately follows the string w.
2) The TM is said to halt on input w if we can reach to halting state after performing some operation.

i.e. If TM = (Q, Σ, Γ, δ, q_0, B, {q_a}) is a turing machine . Then this TM is said to be halt on input w if and only if $BwqB$ yields to $B\alpha q_a B$, for some αεΓ* i.e. $(qBwB) \vdash (B\alpha q_a B)$.

Definition

A function $f(x) = y$ is said to be computable by a TM (Q, ∑, Γ, δ , $q0$, B, {q_a}) if $(q0\ ,BxB) \vdash^* (q_a,\ ByB)$ where x may be in some alphabet $\sum_1$ * and y may be in some alphabet $\sum_2$ * and $\sum_1, \sum_2 \subseteq \sum$. It means that if we give input x to the Turing Machine TM , it gives output as a string if it computes the function $f(x) = y$.

Example 5 : Design a TM which compute the function f(x) = x +1 for each x that belongs to the set of natural numbers.

Solution:

Given function f(x) = x + 1. Here we represent input x on the tape by a number of 1's on the tape.

For example x = 1, input will be B1B,

for x = 2 , input will be B11B,

for x = 3, input will be B111B and so on.

Similarly output can be seen by the number of 1s on the tape when machine halts.

Let TM = $(Q,\sum,\Gamma, \delta, q_0, B,\{q_a\})$ where Q = {q0,qa} Γ ={1,B} and halt state qa . The transition function is defined as

Q	B	1
q_0	(q_0,B,R)	$(q_1,1,R)$
q_1	$(q_a,1,R)$	$(q_1,1,R)$
q_a	-	-

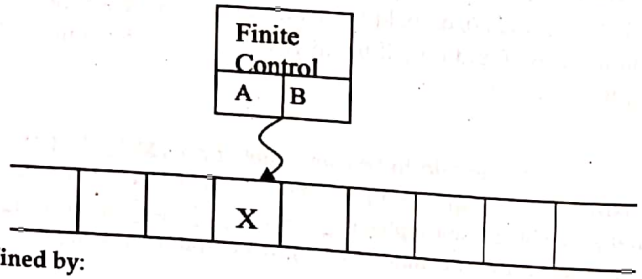
Let input x = 4, that is input tape contains input as B1111B.

So (q0B1111B) ⊢ (Bq01111B) ⊢(B1$q_1$1111B) ⊢(B11$q_1$11B) ⊢(B111$q_1$1B) ⊢(B1111q_1B) ⊢(B11111q_a B) Which means output is 5.

Turing Machine with Storage in the State

In Turing machine, any state represents the position in the computation. But a state can also be used to hold a finite amount of data. The finite control of machine consists of a state q and some data portion. In this case a state is considered as a tuple – (state,data).

Following figure illustrates the model.



δ is defined by:

δ ([q, A], X) = ([$q1$, X],Y, R) means q is the state and data portion of q is A. The symbol scanned on the tape is copied into the second component of the state and moves right entering state q1 and replacing tape symbol by Y.

Turing Machine as Enumerator of Strings of a Language

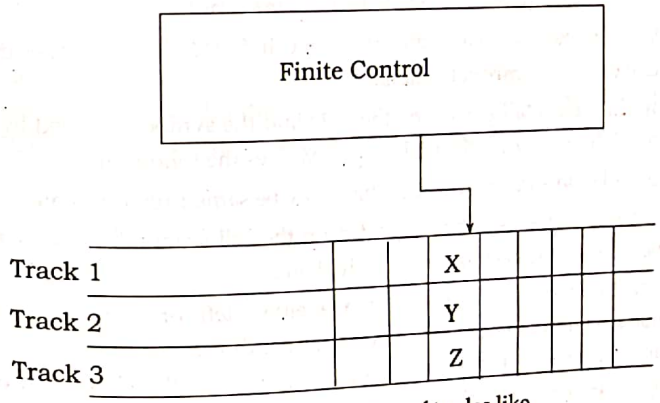
We have viewed turing machines as recognizers of languages and computers of functions on the non-negative integers. There is a third useful view of turing machines, as generating devices.

Consider a multi-tape turing machine TM that uses on tape as an output tape, which a symbol, once written can never be changed, and those whose tape head never moves left. Suppose also that on the output tape, TM writes strings over some alphabet $\sum$, separated by a marker symbol #. We can define L(TM), the language generated by TM to be set of w in $\sum^*$ such that w is eventually printed between a pair of #'s on the output tape.

Thus the language of this type of Turing machine is called Turing Enumerable languages.

Turing Machine with Multiple Tracks

The tape of TM can be considered as having multiple tracks. Each track can hold one symbol, and the tape alphabet of the TM consists of tuples, with one component for each track. Following figure illustrates the TM with multiple tracks;



The tape alphabet Γ is a set consisting of tuples like,

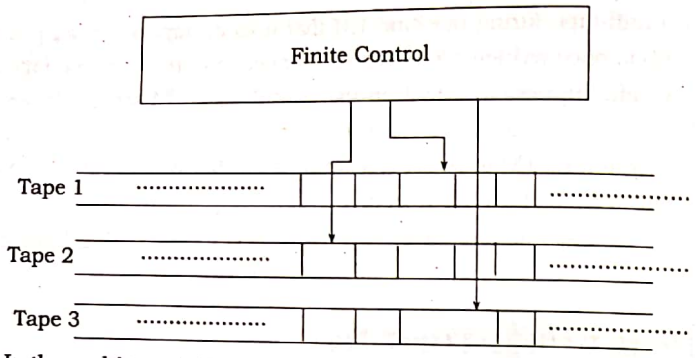
Γ = {(X, Y, Z), ...}

The tape head moves up and down scanning symbols in the tape at one position.

Multi-tape Turing Machine

Modern computing devices are based on the foundation of TM computation models. To simulate the real computers, a TM can be viewed as multi-tape machine in which there is more than one tape. However, adding extra tape adds no power to the computational model, only the ability to accept the language is concerned.

A multi-tape TM consists of finite control and finite number of tapes. Each tape is divided into cells and each cell can hold any symbol of finite tape alphabets. The set of tape symbols include a blank and the input symbols.



In the multi-tape TM, initially;

- Input (finite sequence of input symbols) w is placed on the first tape.
- All other cells of the tapes hold blanks.
- TM is in initial state.
- The head of the first tape is at the left end of the input.
- All other tape heads are at some arbitrary cell. Since all other tapes except first tape consists completely blank.

A move of multi- tape TM depends on the state and the symbol scanned by each of the tape head. In one move, the multi-tape TM does the following:

- The control enters in a new state, which may be same previous state.
- On each step, a new symbol is written on the cell scanned, these symbols may be same as the symbols previously there.
- Each of the tape head make a move either left or right or remains stationary. Different head may move different direction independently i.e. if head of first tape moves leftward; at same time other head can move another direction or remains stationary.

The initial configuration (initial ID) of multi-tape TM with n-tapes is represented as;

$(q_0, ax, B, B, \ldots\ldots\ldots\ldots B)$; n+1 tuples.

Where w= ax is an input string and head first tape is scanning first symbol of w.

So, in general, it can be rewritten as;

$(q, x_1a_1y_1, x_2a_2y_2, \ldots\ldots\ldots\ldots\ldots\ldots x_na_ny_n)$

Where each x_i are the portion of string on tapes before the current head position, each a_i are the symbol currently scanning in each tapes and each y_i are the portion of string on tapes just rightward to the current head position q is the control state.

EQUIVALENCE OF ONE-TAPE AND MULTI-TAPE TM'S

Theorem: Every language accepted by a multitape TM is recursively enumerable.

or

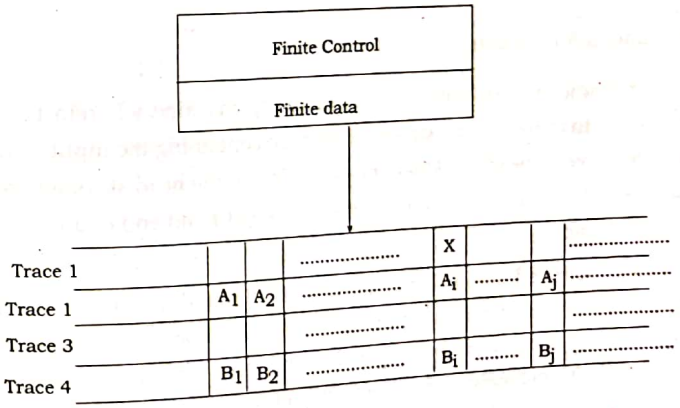
Any languages that are accepted by a multi-tape TM are also accepted by one tape Turing Machine.

(Since One tape accepts only the recursively enumerable language. Hence if multi-tape TM accepts the recursively enumerable then we can convert it into its equivalent One-tape TM . Thus to prove above theorem we have to describe the way to convert mult-tape TM to One tape TM.)

Proof:

- Let L is a language accepted by a k-tape TM, M. Now, can simulate M with a one-tape TM, N whose tape consists of 2k tracks.
- Half of these tracks holds the tapes of M, and the other half of the tracks each hold only a single marker that indicates where the head for the corresponding tape of M is currently located.

i.e. To simulate 2-tape turing machine using 1-tape turing machine we need a tape having 4-tracks. The second and fourth tracks hold the contents of the first and second tapes of M, track first holds the position of the head of tape 1, and track third holds the position of the second tape head.



Now, to simulate a move of M,

- N's head must visit the k-head markers.
- After visiting each head marker and storing the scanned symbol in a component of its finite control, N knows what tape symbols are being scanned by each of M's head.
- N also knows the state of M, which it stores in N's own finite control. Thus N knows what move M will make.

- M now revisits each of the head markers on its tape, changes the symbol in track representing corresponding tapes N and moves the head marker left or right, if necessary.

Finally, N changes the state of M as recorded in its own finite control. Hence N has simulated one move of M.

We select N's accepting states, all those states that record M's state as one of the accepting a state of M. Hence, whatever M accepts N also accepts.

Non-deterministic Turing Machine

A non-deterministic Turing Machine (NTM), $T = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is defined exactly the same as an ordinary TM, except the value of transition function δ. In NTM, the values of the transition function δ are subsets, rather than a single element of the set $Q \times \Gamma \times \{R,L,S\}$. Here, the transition function δ is such that for each state q and tape symbol x, $\delta(q, x)$ is a set of triples.

$$\{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (q_k, Y_k, D_k)$$

where k is any finite integer.

The NTM can choose, at each step, any of the triples to be the next move.

Restricted Turing Machine

Linear Bounded Automaton

Linear Bound Automation is a type of turing machine wherein the tape is not permitted to move off the portion of the tape containing the input. If the machine tries to move its head off either end of the input, the head stays where it is,in the same way that the head will not move off the left-hand end of an ordinary turing machine's tape.

A Linear bounded automaton is a TM with a limited amount of memory. It can only solve problems requiring memory that can fit within the tape used for the input. Using a tape alphabet larger than the input alphabet allows the available memory to be increased up to a constant factor.

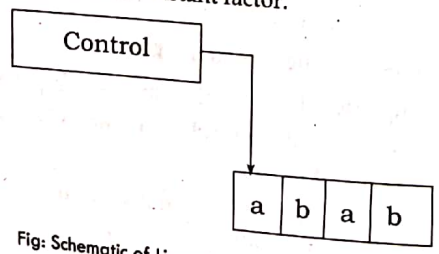
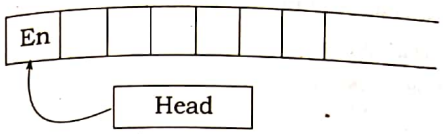


Fig: Schematic of Linear Bounded Automata

Semi-Infinite Tape Turing Machine

A Turing Machine with a semi-infinite tape has a left end but no right end. The left end is limited with an end marker.



It is a two-track tape –

- **Upper track** – It represents the cells to the right of the initial head position.
- **Lower track** – It represents the cells to the left of the initial head position in reverse order.

The infinite length input string is initially written on the tape in contiguous tape cells.

The machine starts from the initial state q_0 and the head scans from the left end marker 'End'. In each step, it reads the symbol on the tape under its head. It writes a new symbol on that tape cell and then it moves the head either into left or right one tape cell. A transition function determines the actions to be taken.

It has two special states called **accept state** and **reject state**. If at any point of time it enters into the accepted state, the input is accepted and if it enters into the reject state, the input is rejected by the TM. In some cases, it continues to run infinitely without being accepted or rejected for some certain input symbols.

Note – Turing machines with semi-infinite tape are equivalent to standard Turing machines.

Multi Stack Machines

- Generalizations of the PDAs
- TMs can accept languages that are not accepted by any PDA with one stack.
- But PDA with two stacks can accept any language that a TM can accept.

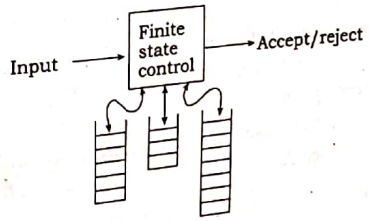


Figure: A machine with three sacks

- A k-stack machine is a deterministic PDA with k stacks.
- It obtains its input from an input source rather than having the input placed in a tape.
- It has a finite control, which is in one of a finite set of states.
- It has a finite stack alphabet, which it uses for all its stacks.
- A move of a multistack machine is based on:
- The state of the finite control
- The input symbol read, which is chosen from the finite input alphabet
- The top stack symbol on each stack
- In one move:
- a multistack machine can change to a new state $q \in Q$;
- and replace the top symbol of each stack with a string of zero or more stack symbols $X \in \Gamma^*$.
- The typical transition function of k-stack machine looks similar to:
- In state q, with Xi on top of ith stack,
- For i= 1, 2, ..., k, the machine may consume input a, go to state p, and replace Xi on top of the ith stack by string yi, for i=1,2, ..., k.
- To make it easier for a multistack machine to simulate a TM, we introduce a special symbol called the endmarker, represented by $.
- The role of the endmarker is To let us know when we have consumed all the available input.
- The endmarker appears only at the end of the input and is not part of it.

Counter Machines

Counter Machine are offline TMs (is a multi-tape TM whose input tape is read only) whose storage tapes are semi-infinite,and whose tape alphabets contain only two symbols Z and B(Blank).

Furthermore the symbol Z,which serves as a bottom of stack marker,appears initially on the cell scanned by the tape head and may never appear on any other cell.

An integer i can be stored by moving the tape head i cells to the right of Z. A stored number can be incremented or decremented by moving the tape head right or left. We can test whether a number is zero by checking whether Z is scanned by the head, but we cannot directly test whether two numbers are equal. Instantaneous description of a counter machine can be described by the state, the input tape contents, the position of the input head, and the distance of the storage heads from the symbol Z.

The counter machine can really only stores a count on each tape and tell if that count is zero.

CHURCH THESIS AND ALGORITHM

It is a mathematically un-provable hypothesis about the computability. This hypothesis simplify states that – **"Any algorithmic procedure that can be carried out at all (by human, a team of human or a computer) can be carried out by a TM."**

This statement was first formulated by Alonzo Church a logician, in 1930s and it is referred to as Church Thesis or Church-Turing thesis. It is not a mathematically precise statement so un-provable.

According to Church, **"No computational procedure will be considered an algorithm unless it can be represented by a Turing Machine."**

After adopting Church-Turing Thesis, we are giving a precise meaning of the term:

An algorithm is a procedure that can be executed on a Turing Machine.

Another use of Church-Thesis is that- When we want to describe a solution to a problem, we will often satisfied with a verbal description of the algorithm, translating it into detailed Turing Machine implementations.

Universal Turing Machine

A Turing machine is created to execute a specific algorithm. If we have a Turing machine for computing one function , then for computing different function or doing some other calculation, a different TM will be required.

Originally electronic computers were limited in a similar way , and changing the computation to be performed , requires rewriting the machine. But the modern computer are general purpose, hence to simulate modern computer Alan Turing proposed the concept of Universal Turing Machine. It can simulate arbitrary Turing machine. i.e. This single machine can perform the function of any other Turing machine.

Definition:

A *universal* Turing machine is a Turing machine T_u that works as follows.

It is assumed to receive an input string of the form $e(T)e(w)$, where T is an arbitrary TM, w is a string over the input alphabet of T, and e is an encoding function whose values are strings in $\{0, 1\}^*$. The computation performed by T_u on this input string satisfies these two properties:

1. T_u accepts the string $e(T)e(w)$ if and only if T accepts w.
2. If T accepts w and produces output y, then T_u produces output $e(y)$.

ENCODING OF TURING MACHINE

For the representation of any arbitrary TM T, and an input string w over an arbitrary alphabet as strings $e(T)$, $e(w)$ over some fixed alphabet , a notational system should be formulated . Encoding the TM T and input string w in to $e(T)$ and $e(w)$, it must not destroy any information , for the encoding of TM , we use the alphabet $\{0,1\}$ although the TM may have a much larger alphabet .

For encoding , we start by assigning positive integers to each state , each tape symbol and each of three directions S,L *and* R in the TM we want to encode. We assume two fixed infinite sets Q = { q_1, q_2,.....} and S = { a_1, a_2, ...} so that for any TM

$T_1 = (Q_1,\Sigma,\Gamma,\delta ,q0 , B, F)$, we have $Q_1 \subseteq Q$ and $\Gamma \subseteq S$.

Hence once we have a subscript attached to every possible state and tape symbols, we can represent a state or a symbol by a string of 0's of the appropriate length. 1's are used as separators.

The Encoding Function e

First, associate to each tape symbol, to each state and to each of the three directions , a string

of 0's. Let

$s(B) = 0$

$s(a_i) = 0^i+1$ *for each* $a_i \in S$

$s(q_i) = 0^i+2$ *for each* $q_i \in S$

$s(S) = 0$

$s(L) = 00$

$s(R) = 000$

Then each move of TM , described by formula $\delta (q, a) = (p,b, D)$ is encoded as

$e(m) = s(q)1s(a)1s(p)1s(b)1s(D)1$

and for any TM T, with initial state q, T is encoded as:

$e(T) = s(q)1e(m1)1e(m2)1.....1e(mk)1$

where $m1$, $m2$,.....mk are the distinct moves of T arranged in some arbitrary order. Finally,

any string $w = w1w2w3.....wk$, for each $wk \in S$ is encoded as:

$e(w) = 1s(w1)1s(w2)1.....1s(wk)1$

An Example: Let TM T is given as:

$T = (\{q1,q2,q3\},\{a,b\},\{a,b, B\},\delta ,q1, B, F)$ where δ is defined by the following moves.

$m1 = \delta (q1,b) = (q3, a, R)$

$m2 = \delta (q3,a) = (q1,b, R)$

$m3 = \delta (q3,b) = (q2 ,a, R)$

$m4 = \delta (q3, B) = (q3,b, L)$

Encoding :

$s(q1) = 000$

$s(q2) = 0000$

$s(q3) = 00000$

$s(a1) = 00$ considering $a1 = a$ *and* $a2 = b$

$s(a2) = 000$

$s(B) = 0$

$s(S) = 0$

$s(L) = 00$

$s(R) = 000$

Now

$e(m1) = s(q1)1s(b)1s(q3)1s(a)1s(R)1 = 000100010000010010001$

$e(m2) = s(q3)1s(a)1s(q1)1s(b)1s(R)1 = 000001001000100010001$

$e(m3) = s(q3)1s(b)1s(q2)1s(a)1s(R)1 = 000001000100000100100001$

$e(m4) = s(q3)1s(B)1s(q3)1s(b)1s(L)1 = 000001010000010001001$

Now The code for T will be:

$e(T) = s(q1)1e(m1)1e(m2)1e(m3)1e(m4)1$

= 000100010001000001001000110000010010001000100011000001000100001001000110010001100 0001010000100010011

for this machine the input (T,w) where w= ab , the code will be,

e(w) = 1s(a)1s(b)1 = 10010001

Code for (T,W) is:

000100010001000001001000110000010010001000100011000001000100001001000110000 010100000100010011100010001

1. Construct a Turing Machine accepting a string ab.
2. Construct a Turing machine accepting a language of strings ending with aba.
3. Construct a Turing Machine accepting the language, L={ $\{^n\}^n$ }. Also show the transition diagram of the machine.

4. How Turing Machine can be simulated as having storage space in its finite control. Illustrate with an example.
5. How a multi-tape Turing Machine works? Is it possible to simulate a multi-tape TM with just a single tape? If yes, Justify.
6. Construct a Turing Machine accepting the language, $L=\{ a^n b^n \mid n>=1\}$. Also show the transition diagram of the machine.
7. Construct a Turing Machine accepting the language, $L= \{ a^n b^{2n} \mid n>=1\}$. Also show the transition diagram of the machine.
8. Design a TM for language $L= \{x^{2n}y^n / n>=0\}$. Show instantaneous description for xxxxyy and xxyy.
9. Construct a Turing Machine accepting the strings represented by ab*+ba*. Also draw the equivalent transition diagram.
10. Construct a TM for language $L= \{a^nb^nc^n / n>=0\}$. Show instantaneous description for aabbcc and abcc.
11. Design a TM that recognizes the language $L = \{0^{2n} \mid n \geq 0\}$
12. Design a Turing Machine to calculate the
    (i) 1- complement of a binary number
    (ii) 2- complement of a binary number
13. Construct a non-deterministic Turing Machine, which recognizes the language $\{ww/w \in \{a, b\}^*\}$.
14. Let M be the Turing Machine defined by

δ	B	a	b	c
q_0	q_1, B, R			
q_1	q_1, B, R	q_1, a, R	q_1, b, R	q_2, C, L
q_2		q_2, b, L	q_2, a, L	

(a) Trace the computation for the input string abcab.
(b) Trace the first six translations of the computation for the input string abab.
(c) Give the state diagram of M.
(d) Describe the result of a computation in M.

❑❑❑