Basrah University

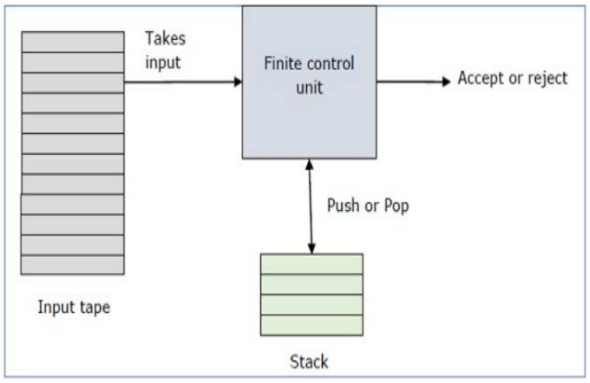
Faculty of Education for Pure Sciences

Computer Department

Pushdown Automata (PDA)

A pushdown automaton is a way to implement a context-free grammar (CFG) the similar way we have designed 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 finite control unit.
- a stack with infinite size.



The stack head scans the top symbol of the stack, and does two operations: –

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

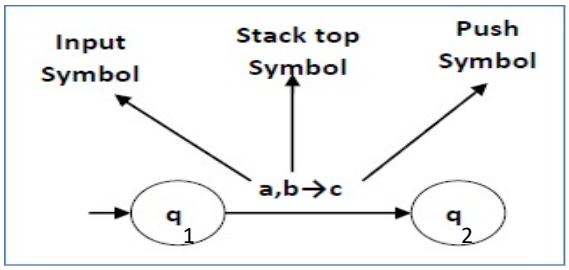
A PDA may or may not read an input symbol, but it has to read the top of the stack in every transition. A PDA is formally defined by 7 tuples as shown below:

P= (Q, ∑, Γ, δ, q_0, z_0, F): –

- **Q** is the finite set of states
- **∑ (Sigma)** is the finite set of input alphabet (input tape)
- **Γ** is the finite stack alphabet
- **δ** (Delta) is the transition function
- **q_0** is the start state ($q_0 \in Q$)
- **Z_0** is the start stack top symbol ($z_0 \in \Gamma$)
- **F** is the set of final/accepting states ($F \in Q$)

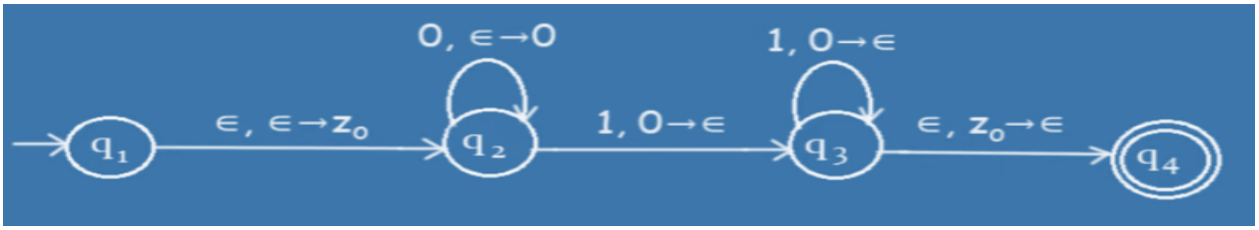
δ Takes as an argument a triple (q, a, x) where:
i- q is a state in Q
ii- a is either an Input Symbol in Σ or a= ε
iii-X is a Stack Symbol, that is a member of Γ

The following diagram shows a transition in a PDA from a state q_1 to state q_2, labeled as a, b → c

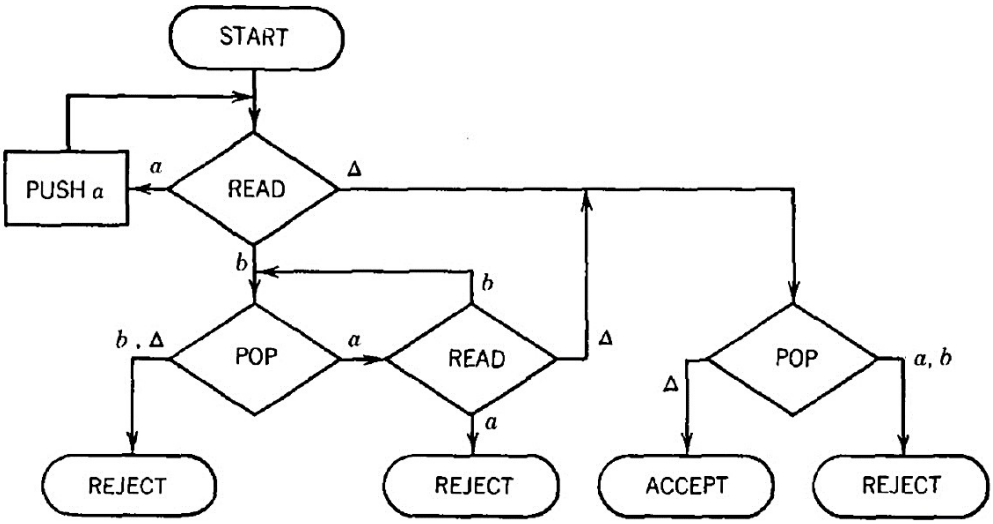


This means at state **q_1**, if we encounter an input string **'a'** and top symbol of the stack is **'b'**, then we pop **'b'**, push **'c'** on top of the stack and move to state **q_2**.

Example: Construct a PDA that accepts L= $\{0^n 1^n \mid n \geq 0\}$: number of 0s equals number of 1s



Example: The language accepted by this PDA is exactly: $\{a^n b^n, n = 0, 1, 2, \ldots\}$: number of as equals number of bs



Before we begin to analyze this machine in general, let us see it in operation on the input string aaabbb. We begin by assuming that this string has been put on the TAPE. We always start the operation of the PDA with the STACK empty as shown:

TAPE

a	a	a	b	b	b	Δ	…

We have to PUSH the first part of string into the STACK and then POP contents of the STACK when we start reading the second part of the string. For example, aaabbbΔ We will PUSH all "aaa" into the stack then we will POP when we start reading "bbb".

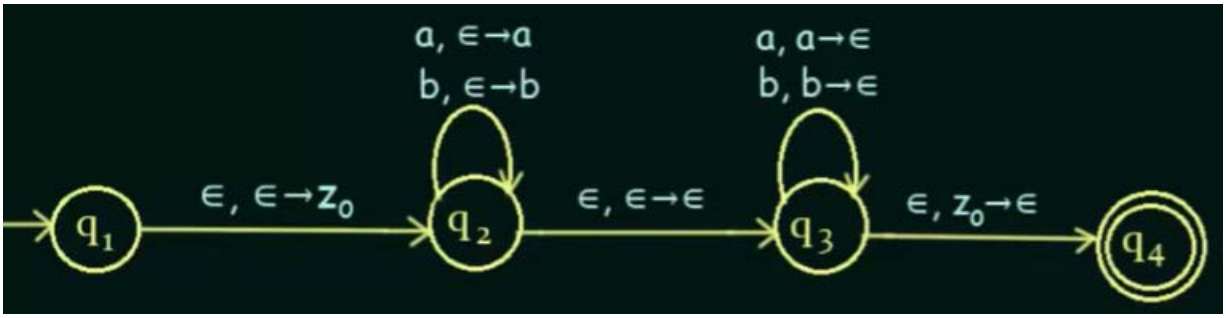
We can divide string reading into two stages. When we get the first part "aaa" we will PUSH them into the STACK, and when we read the second part "bbb" we will POP from the stack. We should get "aaa" and the STACK will be empty and the string is reached to the space symbol, so if we read the space symbol we have to POP from the stack, if we get space symbol that is means the string is accepted.

State	Stack	Tape
START	Δ	aaabbbΔ
READ	Δ	aaabbbΔ
PUSH a	aΔ	aaabbbΔ
READ	aΔ	aaabbbΔ
PUSH a	aaΔ	aaabbbΔ
READ	aaΔ	aaabbbΔ
PUSH a	aaaΔ	aaabbbΔ
READ	aaaΔ	aaabbbΔ
POP	aaΔ	aaabbbΔ
READ	aaΔ	aaabbbΔ
POP	aΔ	aaabbbΔ
READ	aΔ	aaabbbΔ
POP	Δ	aaabbbΔ
READ	Δ	aaabbb**Δ**
POP	-	aaabbb**Δ**
ACCEPT		

Example: Construct a PDA that accepts **Even** Palindromes of the form L={ WW^R | W = $(a+b)^+$ }

PALINDROMES: A word of sequence that reads the same backwards as forwards for example: -

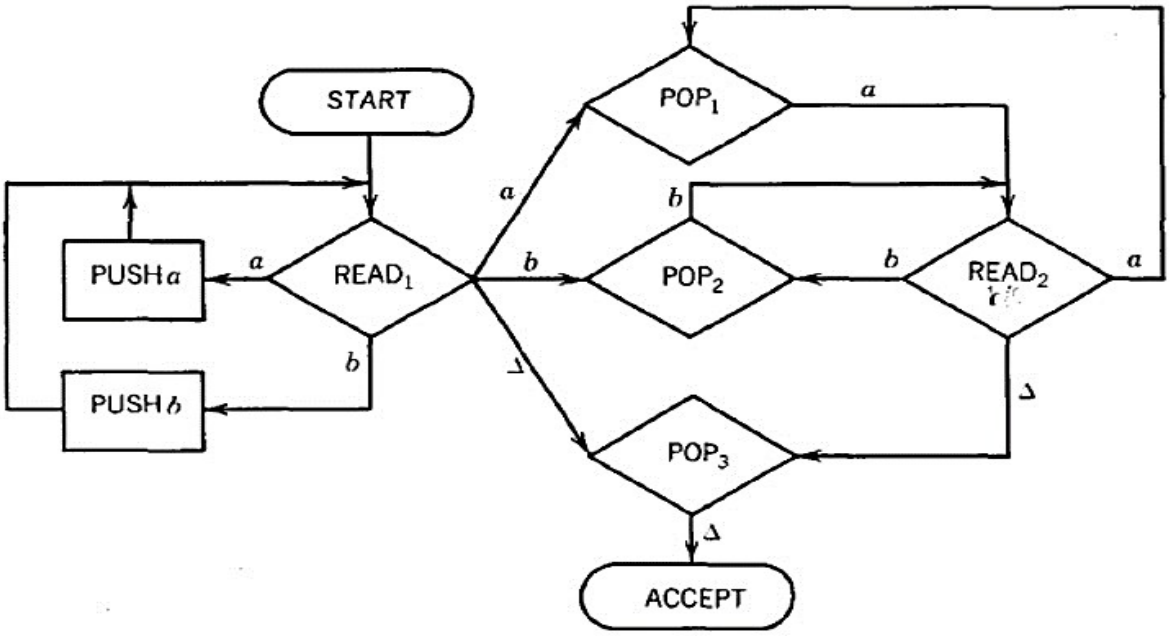
NOON, 123321, ABBA, RACEECAR, NO LEMON NO MELOM



Example: EVENPALINDROME = {s reverse(s), where s is in (a + b)*}

{Λ, aa, bb, aaaa, abba, baab, babbab, bbbbaaaaaa, ... }

We will check the string “babbab” is ACCEPT or not in the following machine:



State	Stack	Tape
START	Δ	babbabΔ
READ	Δ	**b**abbabΔ
PUSH b	bΔ	**b**abbabΔ
READ	bΔ	**ba**bbabΔ
PUSH a	baΔ	**ba**bbabΔ
READ	baΔ	**bab**babΔ
PUSH b	babΔ	**bab**babΔ
READ	babΔ	**babb**abΔ
POP	baΔ	**babb**abΔ
READ	baΔ	**babba**bΔ
POP	bΔ	**babba**bΔ
READ	bΔ	**babbab**Δ
POP	Δ	**babbab**Δ
READ	Δ	**babbabΔ**
POP	-	**BabbabΔ**
ACCEPT		

Theorem: A language is context free if only if same pushdown Automata recognizes it.

Proof

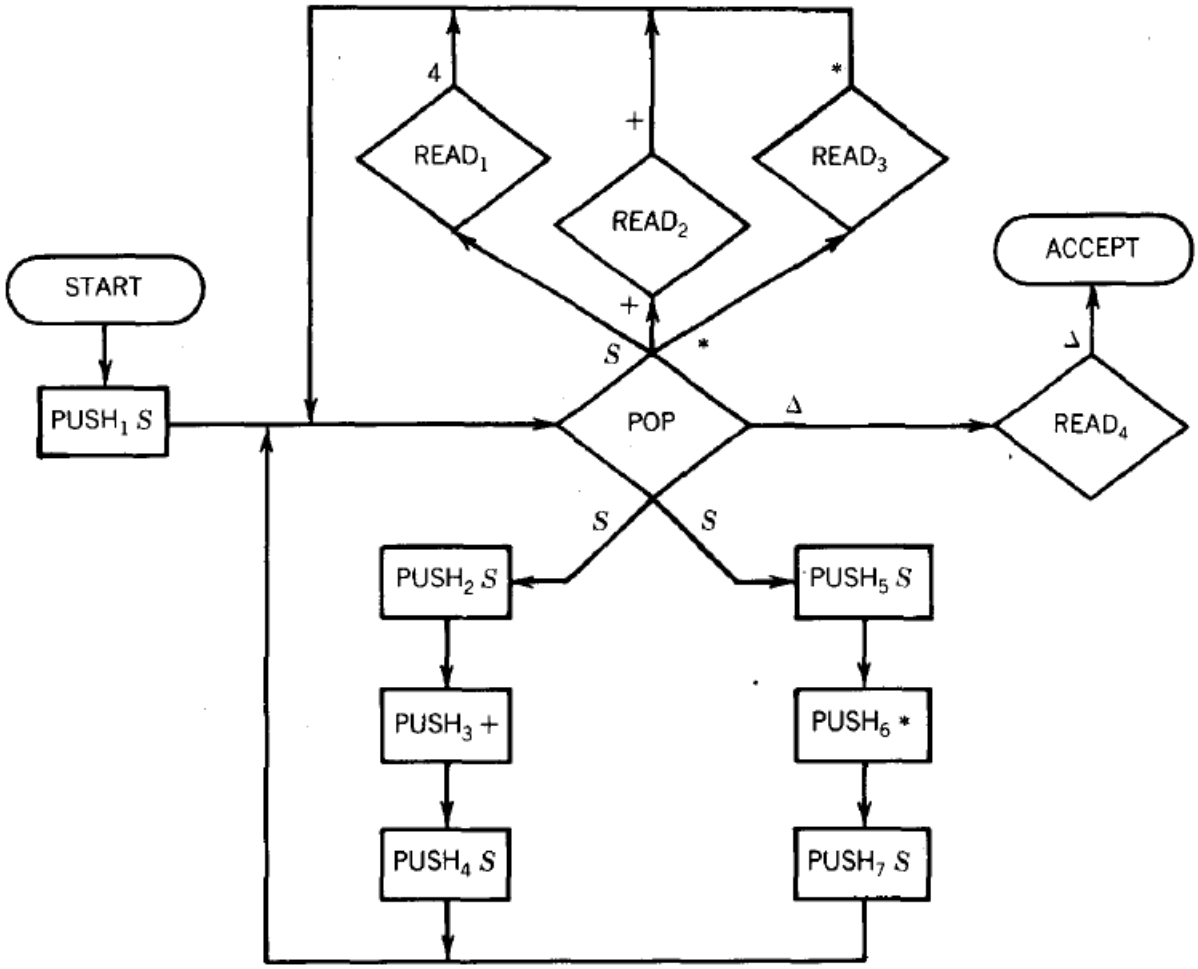
Part1: Given a CFG, show how to construct a PDA that recognizes it.

Part2: Given a PDA, show how to construct a CFG that recognizes the same language.

Example: Consider the language generated by the CFG:

S→ S + SI S * S|4

The terminals are +, *, and 4 and the only nonterminal is S. The following PDA accepts this language: the string 4 + 4 * 4



STATE	STACK	TAPE
START	Δ	4 + 4 * 4
$PUSH_1$ S	S	4 + 4 * 4
POP	Δ	4 + 4 * 4
$PUSH_2$ S	S	4 + 4 * 4
PUSH $_3$ +	+ S	4 + 4 * 4
$PUSH_4$ S	S + S	4 + 4 * 4
POP	+ S	4 + 4 * 4
$READ_1$	+S	+ 4 * 4
POP	S	+ 4 * 4
$READ_2$	S	4 * 4
POP	Δ	4 * 4
$PUSH_5$ S	S	4 * 4
PUSH $_6$ *	*S	4 * 4
$PUSH_7$ S	S * S	4 * 4
POP	* S	4 * 4
$READ_1$	* S	* 4
POP	S	* 4
$PEAD_3$	S	4
POP	Δ	4
$READ_1$	Δ	Δ
POP	Δ	Δ
$READ_4$	Δ	Δ
ACCEPT		

Turing Machine

Turing machine was invented in 1936 by Alan Turing. It is an accepting device which accepts Recursive Enumerable Language generated by type 0 grammar. There are various features of the Turing machine:

1- It has an external memory which remembers arbitrary long sequence of input.
2- It has unlimited memory capability.
3- The model has a facility by which the input at left or right on the tape can be read easily.
4- The machine can produce a certain output based on its input. Sometimes it may be required that the same input has to be used to generate the output. So, in this machine, the distinction between input and output has been removed. Thus, a common set of alphabets can be used for the Turing machine.

Formal definition of Turing machine

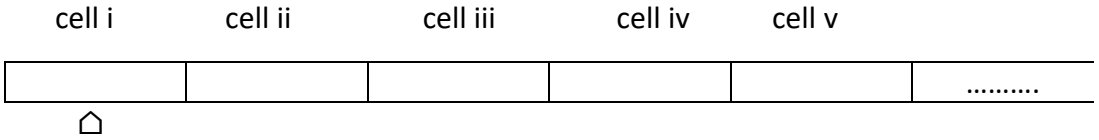
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q0, qaccept, qreject)$, where
Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$ is the transition function,
5. $q0 \in Q$ is the start state,
6. $qaccept \in Q$ is the accept state, and
7. $qreject \in Q$ is the reject state, where $qreject \neq qaccept$.

Turing machine (TM) is collection of six things:

1.An alphabet of input letters, which for clarity's sake does not contain the blank symbol △

2.A TAPE divided into a sequence of numbered cells each containing one character or a blank. The input word is presented to the machine one letter per cell beginning in the left-most cell, called cell i. The rest of the TAPE is initially filled with blanks, △'s.



3.A TAPE HEAD that can in one step read the contents of a cell on the TAPE, replace it with some other character, and reposition itself to the next cell to the right or to the left of the one it has just read. At the start of the processing, the TAPE HEAD always begins by reading the input in cell i. The TAPE HEAD can never move left from cell i. If it is given orders to do so, the machine crashes.

4.An alphabet, Γ, of characters that can be printed on the TAPE by the TAPE HEAD. This can include ΣI. Even though we allow the TAPE HEAD to print a △ we call this erasing and do not include the blank as a letter in the alphabet Γ.

5. A finite set of states including exactly one START state from which we begin execution (and which we may reenter during execution) and HALT states (the final state there are two final states, the accept state and the reject state) that cause execution to terminate when we enter them. The other states have no functions, only names:

q1, q2, q3, …. or 1, 2, 3,….

6. A program, which is a set of rules that tell us, on the basis of the letter the TAPE HEAD has just read, how to change states, what to print and where to move the TAPE HEAD. We depict the program as a collection of directed edges connecting the states. Each edge is labeled with a triplet of information:

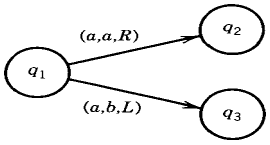
(letter, letter, direction)

The first letter (either △ or from Σ or Γ) is the character the TAPE HEAD reads from the cell to which it is pointing. **The second letter** (also △ or from Γ) is what the TAPE HEAD prints the cell before it leaves**. The third component**, the direction, tells the TAPE HEAD whether to move one cell to the right, R, or one cell to the left, L.

No stipulation is made as to whether every state has an edge leading from it for every possible letter on the TAPE. If we are in a state and read a letter that offers no choice of path to another state, we *crash;* that means we terminate execution unsuccessfully. To terminate execution of a certain input successfully we must be led to a HALT state. The word on the input TAPE is then said to be *accepted* by the TM. A crash also occurs when we are in the first cell on the TAPE and try to move the TAPE HEAD left.

By definition, all Turing machines are **deterministic.** This means that there is no state *q* that has two or more edges leaving it labeled with the same first letter. For example,

is not allowed.



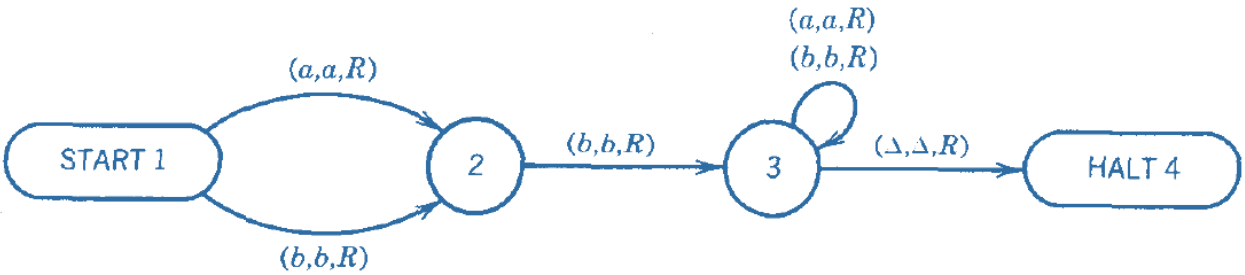
Example: Find TM that can accepts the language defined by the regular expression: (a+b)b(a+b*)

The following is the TAPE from a Turing machine about to run on the input aba

cell i	cell ii	cell iii	cell iv	cell v	cell vi
a	b	a	△	△	△
⌂					

TAPE HEAD

The program for this TM is given as a directed graph with labeled edges as shown below



Notice that the loop at state 3 has two labels. The edges from state 1 to state 2 could have been drawn as one edge with two labels. We start, as always, with the TAPE HEAD reading cell i and the program in the start state, which is here labeled state 1. We depict this as

1

aba

The number on top is the number of the state we are in. Below that is the current meaningful contents of the string on the TAPE up to the beginning of the infinite run of blanks. It is possible that there may be△ inside this string. We underline the character in the cell that is about to be read. At this point in example, the TAPE HEAD reads the letter a and we follow the edge (a, a, R) to state 2. The instructions of this edge to the TAPE HEAD are "read an a, print an a, move right". The TAPE now looks like this:

cell i	cell ii	cell iii	cell iv	cell v	cell vi
a	b	a	△	△	△
	⌂				

TAPE HEAD

Notice: that we have stopped writing the words "TAPE HEAD" under the indicator under the TAPE. It is still the TAPE HEAD nonetheless.

We can record the execution process by writing:

1 2

aba→ aba

At this point we are in **state 2**. Since we are reading the **b** in **cell ii**, we must take the ride to **state 3** on the edge labeled (b,b,R). The TAPE HEAD replaces the **b** with **b** and moves right one cell. The idea of replacing a letter with itself may seem silly, but it unifies the structure of Turing machines.

We could instead have constructed a machine that uses two different types of instructions: either print or move, not both at once. Our system allows us to formulate two possible meanings in a single type of instruction.

(a, a, R) means move, but do not change the TAPE cell

(a, b, R) means move and change the TAPE cell

This system does not give us a one-step way of changing the contents of the TAPE cell without moving the TAPE HEAD, but we shall see that this too can be done by our TM's. Back to machine. We are now up to

1 2 3

Aba→ aba→ aba

The TAPE now looks like this.

cell i	cell ii	cell iii	cell iv	cell v	cell vi
a	b	a	△	△	△

⌂

TAPE HEAD

We are in **state3** reading an **a**, so we loop. That means we stay in **state 3** but we move the TAPE HEAD to cell iv.

3 3

aba→ aba△

This is one of those times when we must indicate a △ as part of the meaningful contents of the TAPE. We are now in **state3** reading △, so we move to **state 4**.

3

aba△→HALT

The input string aba has been accepted by this TM. This particular machine did not change any of the letters on the TAPE, so at the end of the run the TAPE still reads aba△ . . . . This is not a requirement for the acceptance of a string, just a phenomenon that happened this time.

In summary, the whole execution can be depicted by the following execution chain, also called a process chain, or a trace of execution, or simply a trace:

1 → 2 → 3 → 3

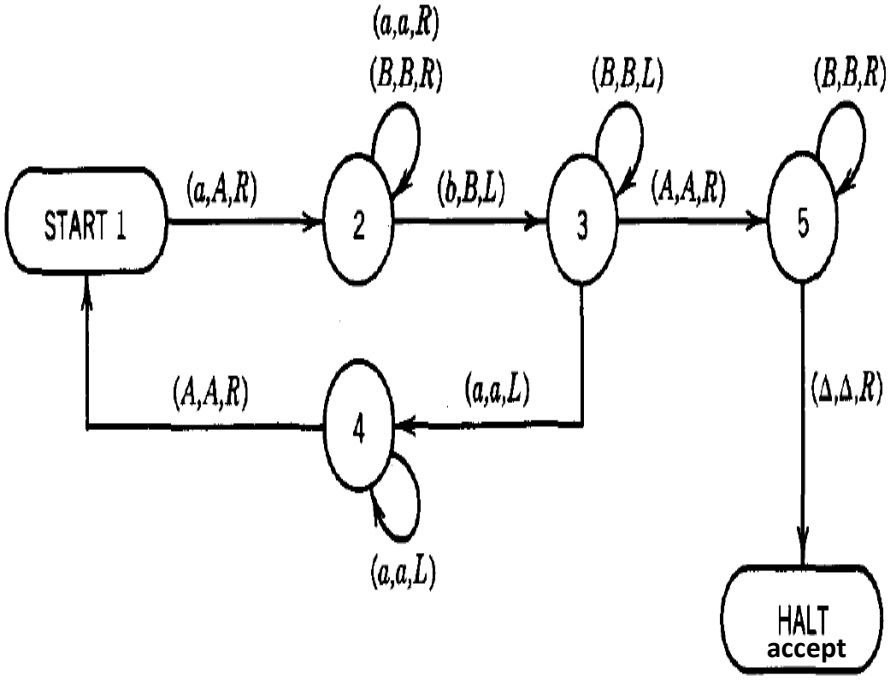
aba aba aba aba△→ HALT

The language of words accepted by this machine is: All words over the alphabet {a,b} in which the second letter is a b.This is a regular language because it can also be defined by the regular expression:(a + b)b(a + b)*

Stats	Tape head position
Start 1	**a**ba
2	a**b**a
3	ab**a**
3	aba△
HALT4	aba△

Example: Find TM that can accepts the language $\{a^n b^n\}$, trace the acceptance of the string aaabbb

States	Tape head position
Start 1	**a**aabbb
2	A**a**abbb
2	Aa**a**bbb
2	Aaa**b**bb
3	Aa**a**Bbb
4	A**a**aBbb
4	**A**aaBbb
Start1	A**a**aBbb
2	Aa**a**Bbb
2	Aaa**B**bb
2	AaaB**b**b
3	AAa**B**Bb
3	AA**a**BBb
4	A**A**aBBb
Start1	AA**a**BBb
2	AAA**B**Bb
2	AAAB**B**b
2	AAABB**b**
3	AAAB**B**B
3	AAA**B**BB
3	AA**A**BBB
5	AAA**B**BB
5	AAAB**B**B
5	AAABB**B**
HALT	AAABBB△



Accept in this TM