

Regular and Context free Grammar

Shruthi Kumaravel - 106119116

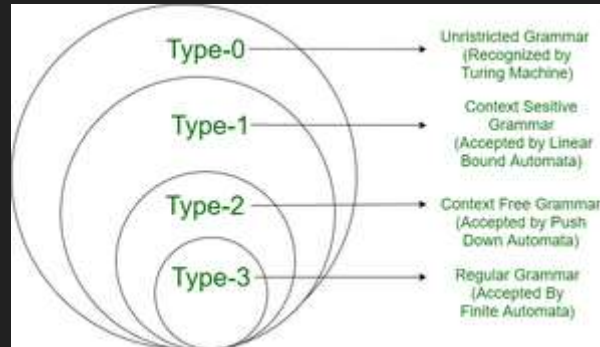
Grammar

A Grammar 'G' can be formally described using four tuples as $G = (V, T, S, P)$:

- V = Set of Variables or Non-Terminal Symbols
- T = Set of Terminal Symbols
- S = Start Symbol
- P = Production Rules for Terminals and Non-Terminals

Language generated by a grammar refers to the set of all strings generated by the rules

According to Chomsky's classification of grammar,



Example – language generated from grammar

Example: $G = (\{S, A, B\}, \{a, b\}, S, \{S \rightarrow AB, A \rightarrow a, B \rightarrow b\})$

$$V = \{S, A, B\}$$

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

$$S = S$$

$$P = S \rightarrow AB, A \rightarrow a, B \rightarrow b$$

Eg. $S \rightarrow AB$
 $\rightarrow aB$
 $\rightarrow \underline{ab}$

Example – language generated from grammar

Example 3: $G_3 = (\{S, A, B\}, \{a, b\}, S, \{S \rightarrow AB, A \rightarrow aA \mid a, B \rightarrow bB \mid b\})$

$$\begin{aligned} S &\rightarrow AB \\ &\rightarrow ab \end{aligned}$$

$$\begin{aligned} S &\rightarrow AB \\ &\rightarrow aAbB \\ &\rightarrow aabbb \end{aligned}$$

$$\begin{aligned} S &\rightarrow AB \\ &\rightarrow aAb \\ &\rightarrow aab \end{aligned}$$

$$\begin{aligned} S &\rightarrow AB \\ &\rightarrow abB \\ &\rightarrow abbb \end{aligned}$$

$$L(G_3) = \{ab, a^2b^2, a^2b, ab^2, \dots\}$$

$$= \{a^m b^n \mid m \geq 0 \text{ and } n \geq 0\}$$

Regular Grammar

Regular Grammar or Type 3 grammar

Regular Languages - languages which are accepted by finite automation

A grammar is regular iff a single nonterminal and is a single terminal or a single terminal followed by a single nonterminal, that is a production is of the form $X \rightarrow a$ or $X \rightarrow aY$, where X and Y are nonterminals and a is a terminal.

Regular equations are usually used to describe the structure of lexical construct (identical keywords, constants etc.) but it cannot specify recursive structure of the program

Regular Grammar is divided into two types

A grammar is said to be Right Linear if all productions are of the form

$$A \rightarrow xB$$

$$A \rightarrow x$$

where $A, B \in V$ and $x \in T$

A grammar is said to be Left Linear if all productions are of the form

$$A \rightarrow Bx$$

$$A \rightarrow x$$

where $A, B \in V$ and $x \in T$

Operations on Regular Languages

UNION

$$- A \cup B = \{x \mid x \in A \text{ or } x \in B\}$$

CONCATENATION

$$- A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$$

STAR

$$- A^* = \{x_1 x_2 x_3 \dots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$$

$$A \cup B = \{pq, r, t, uv\}$$

$$A \circ B = \{pqt, pquv, rt, ruv\}$$

$$A^* = \{\epsilon, pq, r, pqr, rpq, pqpq, rr, rpqpq, rrr, \dots\}$$

Properties of Regular Languages

(A) Closure Properties

- Complementation
- Union
- Intersection
- Concatenation
- Kleene Star

(B) Decision Properties

- Membership($w \in L$) – if DFA accepts, membership approved
- Emptiness($L = \varnothing$?) – if no path in DFA
- Equivalence($L1 = L2$?) – if minimal DFAs are equal
- Subset($L1 \subset L2$?) - If $(L1 - L2) = \varnothing$ and $(L2 - L1) \neq \varnothing$
- Infinte(L infinite?) - If DFA(L) has at least one loop

Gate question

Given the language $L = \{ab, aa, baa\}$, which of the following strings are in L^* ?

GATE 2012

1) abaabaabaa ✓

3) baaaabaaaab ✗

2) aaabaaaa ✓

4) baaaabaa ✓

(A) 1, 2 and 3

(B) 2, 3 and 4

(C) 1, 2 and 4

(D) 1, 3 and 4

ANSWER (C)

Advantages

- Lexical rules are quite simple in Regular Grammar.
- Notations in Regular Grammar are easy to understand.
- It is easy to construct efficient recognizer from Regular Expressions.
- There is proper procedure for lexical and syntactical analysis for Regular Grammar.
- Regular Expressions are most useful for describing the structure of lexical construct such as identifiers, constant etc.

Limitations

- Regular Grammars are less productive than Context Free Grammar.
- Regular Languages are the most restricted types of languages that is accepted by finite automata.
- Regular Expressions are not closed under infinite intersections.
- All languages are not regular.

Context Free Grammar

A context-free Language (CFL) - accepted by pushdown automata

In CFG, the start symbol is used to derive the string. You can derive the string by repeatedly replacing a non-terminal by the right hand side of the production, until all non-terminal have been replaced by terminal symbols.

.

Classification of CFGs

CFGs can be classified based on :

1) Number of strings it generates.

- If CFG is generating finite number of strings, then CFG is Non-Recursive (or the grammar is said to be Non-recursive grammar)
- If CFG can generate infinite number of strings then the grammar is said to be Recursive grammar

2) Number of derivation trees

- If there is only 1 derivation tree then the CFG is unambiguous.
- If there are more than 1 left most derivation tree or right most derivation or parse tree , then the CFG is ambiguous

Example - CFG

For generating a language that generates equal number of a's and b's in the form $a^n b^n$, the Context Free Grammar will be defined as $G = \{ (S, A), (a, b), (S \rightarrow aAb, A \rightarrow aAb | \epsilon) \}$

$$\begin{aligned} S &\rightarrow a \underline{A} b \\ &\rightarrow a a \underline{A} b b \quad (\text{by } A \rightarrow aAb) \\ &\rightarrow a a a \underline{A} b b b \quad (\quad " \quad) \\ &\Rightarrow a a a b b b \quad (\text{by } A \rightarrow \epsilon) \\ &\Rightarrow \underline{a^3} \underline{b^3} \Rightarrow a^n b^n \end{aligned}$$

Derivation

- Derivation is a sequence of production rules. It is used to get the input string through these production rules. During parsing we have to take two decisions.
 - We have to decide the non-terminal which is to be replaced.
 - We have to decide the production rule by which the non-terminal will be replaced.
- We have two options to decide which non-terminal to be replaced with production rule i.e. left most or right most

Example – Left most

- Production rules

- $S = S + S$
- $S = S - S$
- $S = a \mid b \mid c$

- Input : $a - b + c$

- LMD

- $S = S + S$
- $S = S - S + S$
- $S = a - S + S$
- $S = a - b + S$
- $S = a - b + c$

Example – Right Most Derivation

- Production rules

- $S = S + S$
- $S = S - S$
- $S = a \mid b \mid c$

- Input : $a - b + c$

- LMD

- $S = S - S$
- $S = S - S + S$
- $S = S - S + c$
- $S = S - b + c$
- $S = a - b + c$

Advantatges

- Context Free Grammar(CFG) is most useful in describing the nested chain structure or syntactic structure such as balanced parenthesis, if else etc. and these can't be define by Regular Expression.
- Context Free Grammar is more powerful than Regular Grammar because it allows a wider range of rules than Regular Grammar i.e. they can generate a wider range of structures than Regular Grammar.
- CFG provide a precise mathematical definition that clearly rules out certain types of language.
- The formal definition means that Context Free Grammars are computationally TRACTABLE i.e. it is possible to write a computer program which determines whether sentences are grammatical or not.
- They provide a convenient visual notation for the structure of sentences (the tree).

Limitations

- Lexical rules are difficult in case of Context free grammar.
- Notations in Context free grammar are quite complex.
- By using the context free grammar, it is very difficult to construct the recognizer.
- There is no specific guideline for lexical and syntactic analysis in case of Context free grammar.

THANK YOU



PUMPING LEMMA

NIRANJANA D P

106119086

PUMPING LEMMA (Regular Language)

- Pumping Lemma for Regular Language is used to prove that a Language is not Regular.
- But it cannot be used to prove that a Language is Regular.

THEOREM

- If A is a Regular Language, then A has a Pumping Length ' P ' such that any string ' S ' where $|S| \geq P$ may be divided into 3 parts $S = x y z$ such that the following conditions must be true:
 - $x y^i z \in A$ for every $i \geq 0$
 - $|y| > 0$
 - $|x y| \leq P$

Language is not Regular (For RL)

- Assume that A is Regular
- It has to have a Pumping Length (say P)
- All strings longer than P can be pumped $|S| \geq P$
- Now find a string 'S' in A such that $|S| \geq P$
- Divide S into x y z
- Show that $x y^i z \notin A$ for some i
- Then consider all ways that S can be divided into x y z
- Show that none of these can satisfy all the 3 pumping conditions at the same time
- S cannot be Pumped == CONTRADICTION

Example

Q. Using Pumping Lemma prove that language $A = \{an, bn \mid n \geq 0\}$ is not Regular

- Assume that A is Regular
- Pumping Length = P
- Consider string $S = a^p b^p$
- Divide S into 3 parts x, y, z
- Lets take $p = 7$, then $S = aaaaaa bbbbbb$
- Case 1: $\underbrace{aa}_x \underbrace{aaaaa}_y \underbrace{bbbbbb}_z$ $x y^i z \mid i = 2, x y^2 z$, then $S = aa \{aaaa aaaa\} bbbbbb$
 $11 \neq 7$
- Case 2: $\underbrace{aaaaaaabb}_{x} \underbrace{bbbbb}_y \underbrace{b}_{z}$ $x y^i z \mid i = 2, x y^2 z$, then $S = aaaaaaabb \{bbbbb bbbb\} b$
 $7 \neq 11$

Therefore, the conditions are not satisfied.

Hence A is not a regular language

PUMPING LEMMA (Context Free Language)

- Pumping Lemma for Context Free Language is used to prove that a Language is not Context Free.

THEOREM

- If A is a Context Free Language, then A has a Pumping Length ' P ' such that any string ' S ' where $|S| \geq P$ may be divided into 5 parts $S = u v x y z$ such that the following conditions must be true:
 - $u v^i x y^i z \in A$ for every $i \geq 0$
 - $|v y| > 0$
 - $|v x y| \leq P$

Language is not Regular (For CFL)

- Assume that A is Regular
- It has to have a Pumping Length (say P)
- All strings longer than P can be pumped $|S| \geq P$
- Now find a string 'S' in A such that $|S| \geq P$
- Divide S into $u v x y z$
- Show that $u v^i x y^i z \notin A$ for some i
- Then consider all ways that S can be divided into $u v x y z$
- Show that none of these can satisfy all the 3 pumping conditions at the same time
- S cannot be Pumped == CONTRADICTION

Q. For $\Sigma = \{a, b\}$, let us consider the regular language $L = \{x \mid x = a^2 + 3k \text{ or } x = b^{10} + 12k, k \geq 0\}$ Which one of the following can be a pumping length (the constant guaranteed by the pumping lemma) for L ?

A. 3

B. 5

C. 9

D. 24

Q. For $\Sigma = \{a, b\}$, let us consider the regular language $L = \{x \mid x = a^2 + 3k \text{ or } x = b^{10} + 12k, k \geq 0\}$ Which one of the following can be a pumping length (the constant guaranteed by the pumping lemma) for L ?

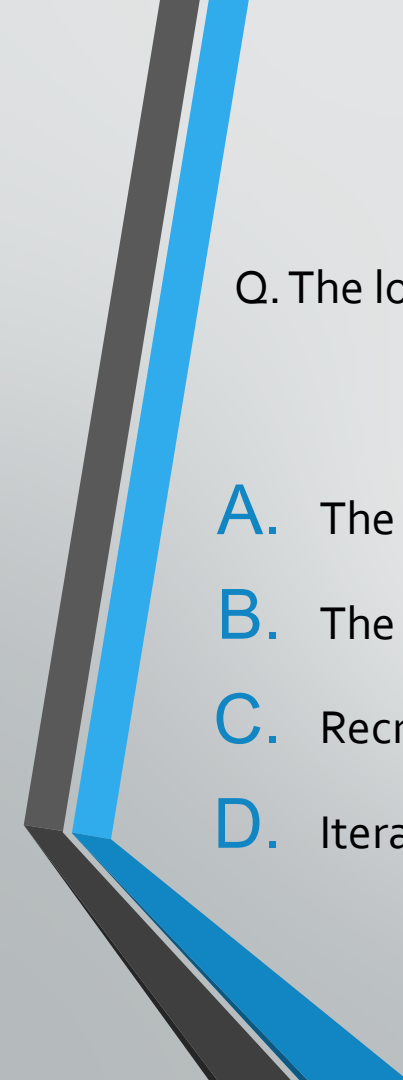
A. 3

B. 5

C. 9

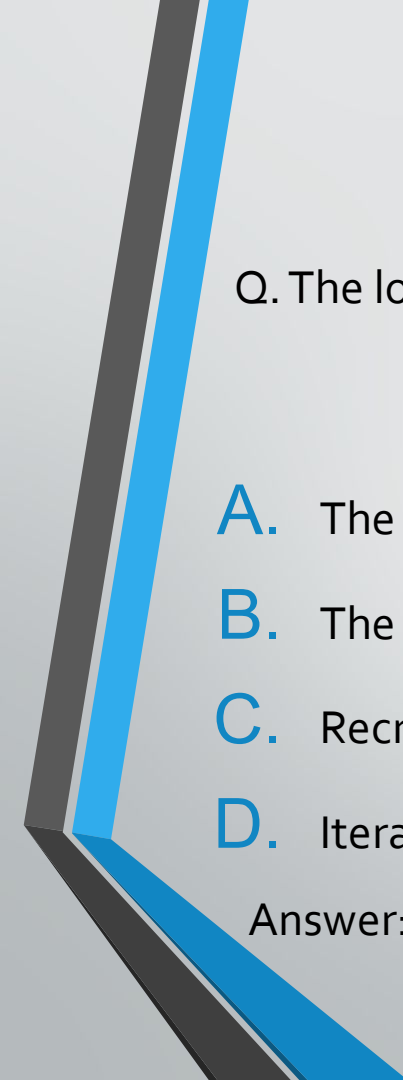
D. 24

Answer: D



Q. The logic of pumping lemma is a good example of

- A. The pigeonhole principle
- B. The divide and conquer technique
- C. Recursion
- D. Iteration



Q. The logic of pumping lemma is a good example of

- A. The pigeonhole principle
- B. The divide and conquer technique
- C. Recursion
- D. Iteration

Answer: A

Q. A language L satisfies the Pumping Lemma for regular languages, and also the Pumping Lemma for context-free languages. Which of the following statements about L is TRUE?

- A. L is necessarily a regular language.
- B. L is necessarily a context-free language, but not necessarily a regular language.
- C. L is necessarily a non-regular language.
- D. None of the above

Q. A language L satisfies the Pumping Lemma for regular languages, and also the Pumping Lemma for context-free languages. Which of the following statements about L is TRUE?

- A. L is necessarily a regular language.
- B. L is necessarily a context-free language, but not necessarily a regular language.
- C. L is necessarily a non-regular language.
- D. None of the above

Answer: D



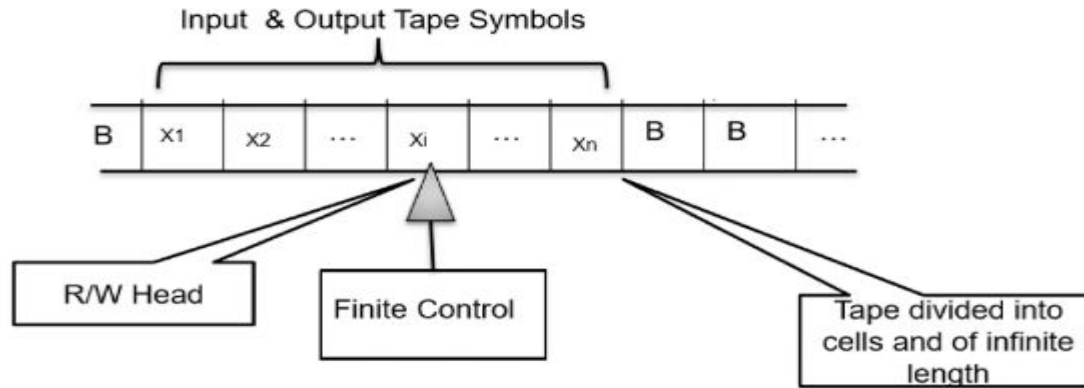
THANK YOU

TURING MACHINE

ISHWARYA A - 106119054

What is Turing Machine?

- A Turing Machine is a computational model which consists of an infinite length tape divided into cells on which input is given.
- It consists of a head which reads the input tape. A state register stores the state of the Turing machine.



TM is described as a 7-tuple $(Q, X, \Sigma, \delta, q_0, B, F)$ where –

- Q is a finite set of states
- X is the tape alphabet
- Σ is the input alphabet
- δ is a transition function; $\delta : Q \times X \rightarrow Q \times X \times \{\text{Left_shift}, \text{Right_shift}\}$.
- q_0 is the initial state
- B is the blank symbol
- F is the set of final states

Designing a Turing Machine

Design a TM to recognize all strings consisting of an odd number of α 's.

Solution

The Turing machine M can be constructed by the following moves –

- Let q_1 be the initial state.
- If M is in q_1 ; on scanning α , it enters the state q_2 and writes B (blank).
- If M is in q_2 ; on scanning α , it enters the state q_1 and writes B (blank).
- From the above moves, we can see that M enters the state q_1 if it scans an even number of α 's, and it enters the state q_2 if it scans an odd number of α 's. Hence q_2 is the only accepting state.

Hence,

$$M = \{\{q_1, q_2\}, \{1\}, \{1, B\}, \delta, q_1, B, \{q_2\}\}$$

where δ is given by –

Tape alphabet symbol	Present State ' q_1 '	Present State ' q_2 '
α	BR q_2	BR q_1

Example

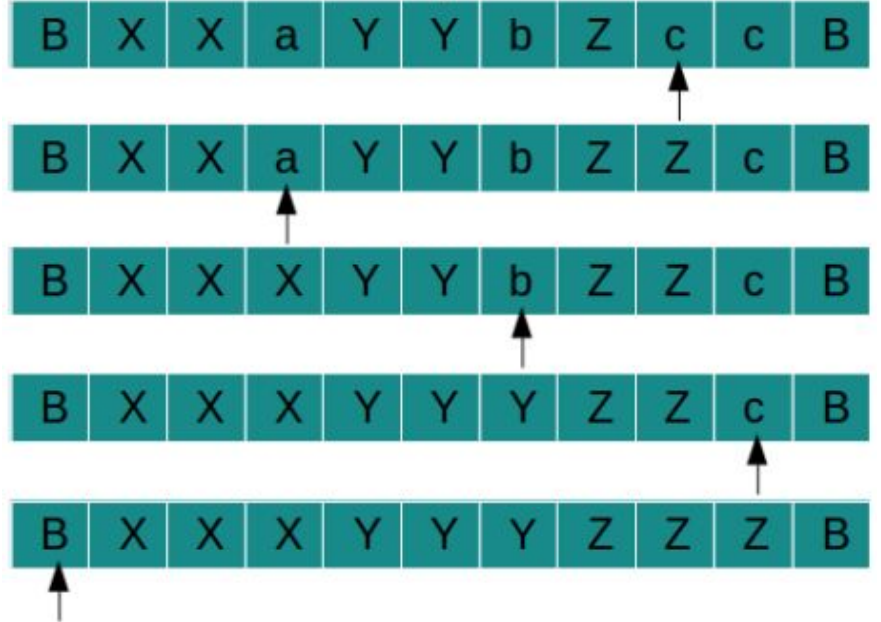
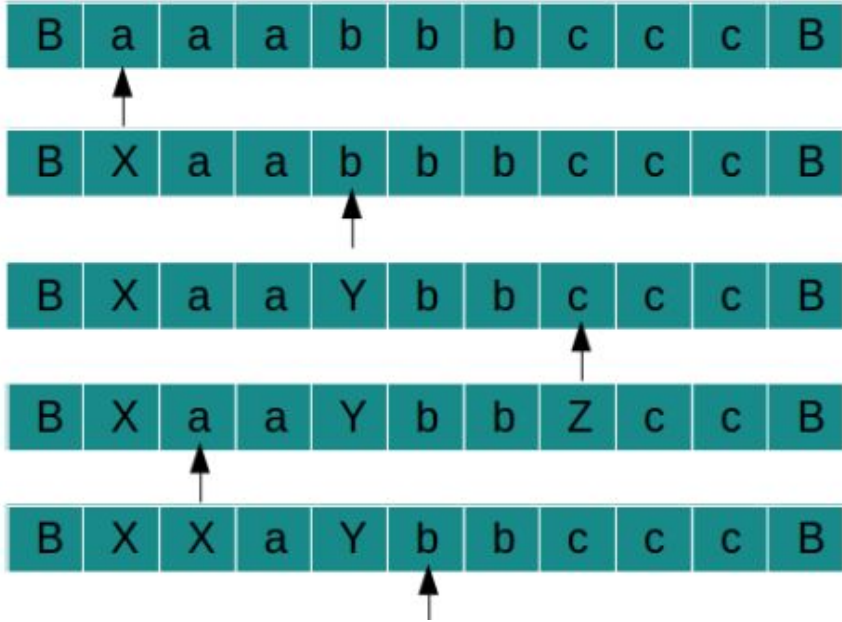
Construct a TM for the language $L = \{a^n b^n c^n\}$ where $n \geq 1$

Solution:

$L = \{ \{a^n b^n c^n\} | n \geq 1 \}$ represents language where we use only 3 character, i.e., a, b and c. In this, some number of a's followed by an equal number of b's and then followed by an equal number of c's. Any type of string which falls in this category will be accepted by this language.

- Mark 'a' then move right.
- Mark 'b' then move right
- Mark 'c' then move left
- Come to far left till we get 'X'
- Repeat above steps till all 'a', 'b' and 'c' are marked
- At last if everything is marked that means string is accepted.

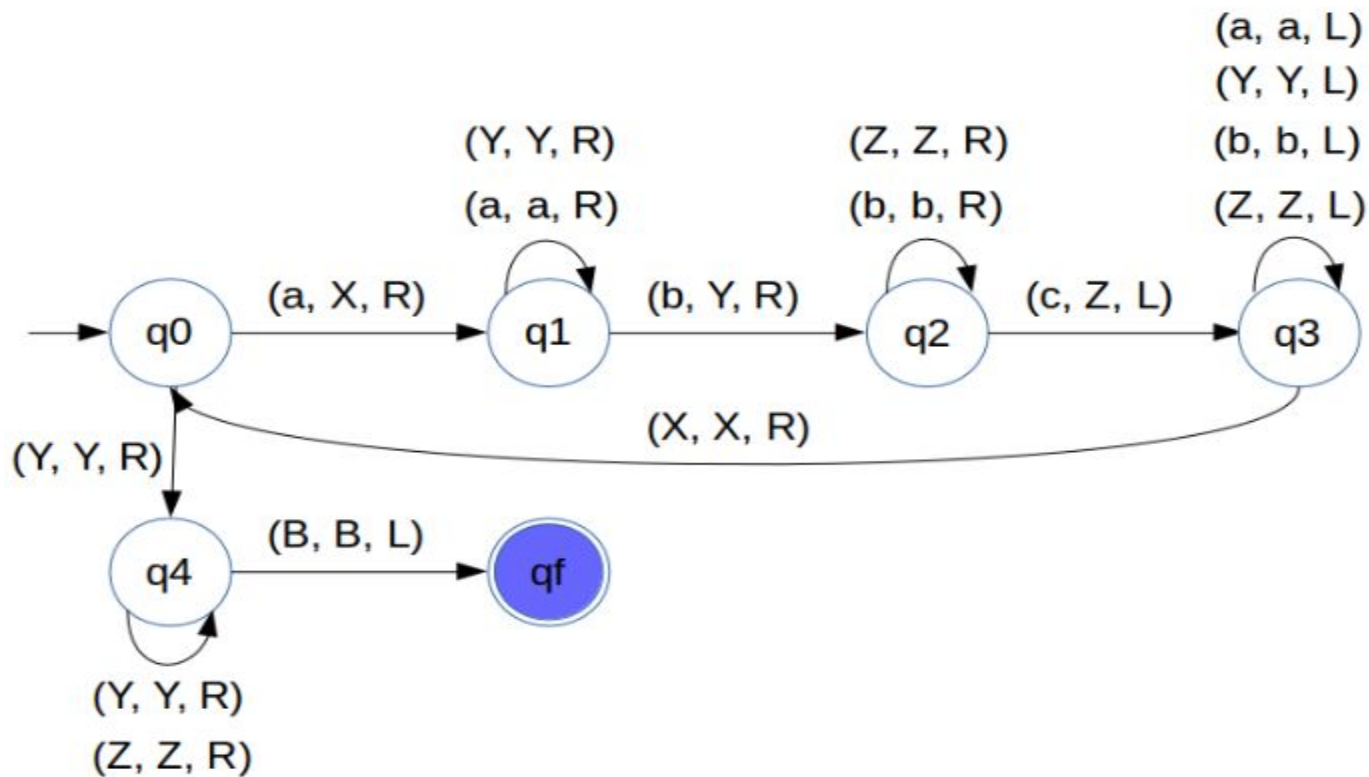
Example - aaabbbccc



Example

states	a	b	c	X	Y	Z	B
q0	(q1,X,R)	-	-	-	(q4,Y,R)	-	-
q1	(q1,a,R)	(q2,Y,R)	-	-	(q1,Y,R)	-	-
q2	-	(q2,b,R)	(q3,Z,L)	-	-	(q2,Z,R)	-
q3	(q3,a,L)	(q3,b,L)	-	(q0,X,R)	(q3,Y,L)	(q3,Z,L)	-
q4	-	-	-	-	(q4,Y,R)	(q4,Z,R)	(q5,B,R)
q5 (qf)	-	-	-	-	-	-	-

Example

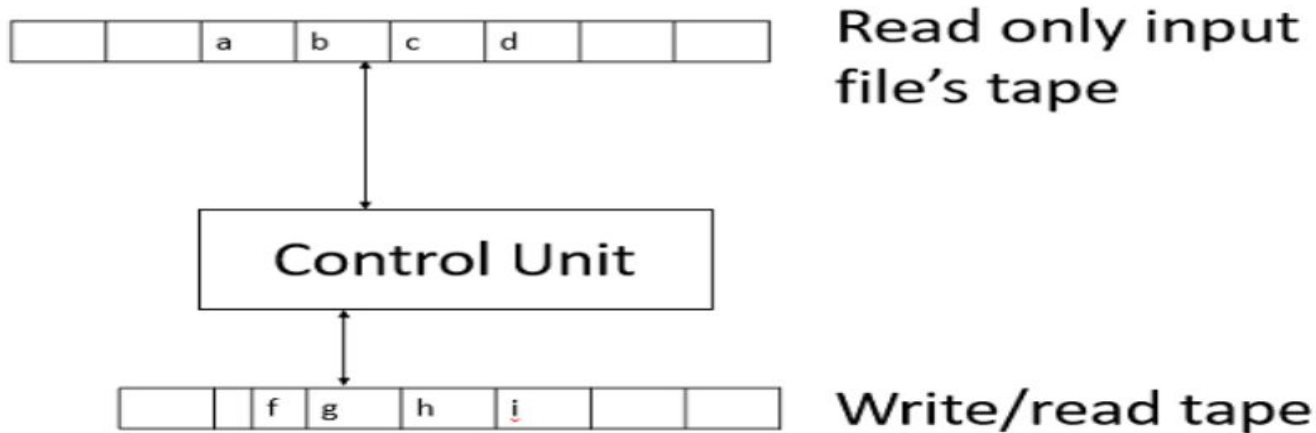


TURING MACHINE VARIATIONS

Off-line Turing Machine

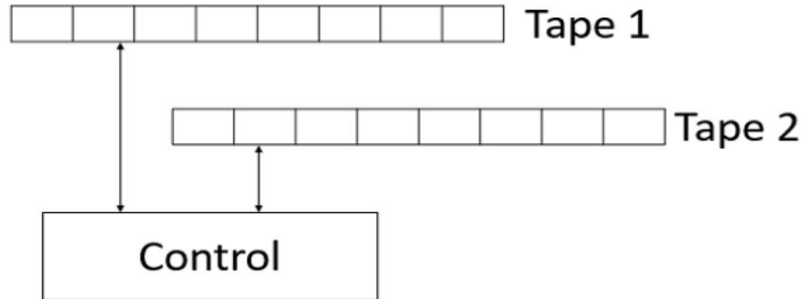
It has two tapes, which are as follows –

- One tape is read-only and contains the input.
- The other is read-write and is initially blank.



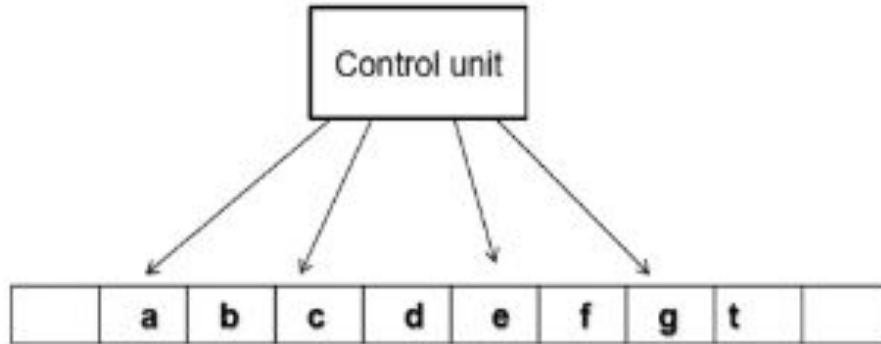
Multitape Turing Machine

- A Turing machine (TM) with several tapes is called a multi tape Turing machine.
- Every tape's have their own Read/Write head
- For N-tape Turing Machine
 - $M = \{ (Q, X, \Sigma, \delta, q_0, B, F) \}$
 - $\delta = Q \times X^N \rightarrow Q \times X^N \times \{L, R\}^N$
- Each TM has its own read-write head, but the state is common for all.



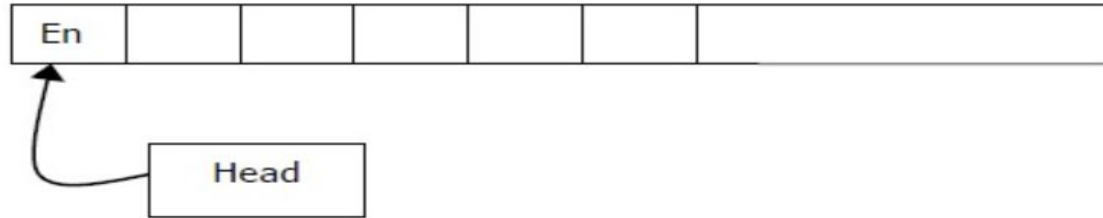
Multihead Turing Machine

- Multihead TM has a number of heads instead of one
- Each head independently read/write symbols and move right/left or be stationary.



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.

Non - Deterministic Turing Machine

- For every input at a state, there can be multiple paths/actions performed by the TM, meaning the transitions are not deterministic.
- An input is accepted if there is at least one node of the tree which is an accept configuration, otherwise it is not accepted.
- If all branches of the computational tree halt on all inputs, the non-deterministic Turing Machine is called a Decider and if for some input, all branches are rejected, the input is also rejected.

THANK YOU

UNDECIDABILITY

KIRUTHIGA S - 106119060

Undecidable Problems

A problem is undecidable if there is no Turing machine which will always halt in finite amount of time to give answer as 'yes' or 'no'. An undecidable problem has no algorithm to determine the answer for a given input.

Examples:

- Ambiguity of context-free languages
- Equivalence of two context-free languages
- Everything or completeness of CFG
- Regularity of CFL, CSL, REC and REC

Decidable Problems

A problem is decidable if we can construct a Turing machine which will halt in finite amount of time for every input and give answer as 'yes' or 'no'. A decidable problem has an algorithm to determine the answer for a given input.

Examples:

- Equivalence of two regular languages
- Finiteness of regular language
- Emptiness of context free language

Recursive language:

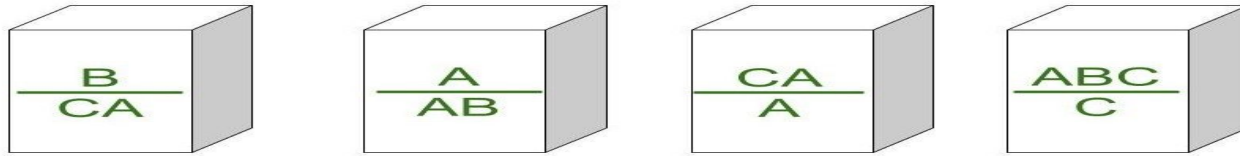
A language 'L' is said to be recursive if there exists a Turing machine which will accept all the strings in 'L' and reject all the strings not in 'L'. The Turing machine will halt every time and give an answer(accepted or rejected) for each and every string input. A language 'L' is decidable if it is a recursive language. All decidable languages are recursive languages and vice-versa.

Recursively enumerable language:

A language 'L' is said to be a recursively enumerable language if there exists a Turing machine which will accept (and therefore halt) for all the input strings which are in 'L' but may or may not halt for all input strings which are not in 'L'. By definition , all REC languages are also RE languages but not all RE languages are REC languages.

Post Correspondence Problem:

In this problem we have N number of Dominos (tiles). The aim is to arrange tiles in such order that string made by Numerators is same as string made by Denominators.



We have two lists both containing N words, aim is to find out concatenation of these words in some sequence such that both lists yield same result.

Eg. $A=[aa, bb, abb]$ and $B=[aab, ba, b]$

Now for sequence 1, 2, 1, 3 first list will yield aabbbaabb and second list will yield same string aabbbaabb. So the solution to this PCP becomes 1, 2, 1, 3. That is, there is no particular algorithm that determines whether any Post Correspondence System has solution or not.

The Halting problem

Can we have an algorithm that will tell that the given program will halt or not. In terms of Turing machine, will it terminate when run on some machine with some particular given input string.

We cannot design a generalized algorithm that can accurately predict whether or not a given program will ever halt.

The only way to find out is to run the algorithm and see if it halts.

State Entry Problem

Given a turing machine, an input string and a state in that turing machine, tell whether the turing machine will enter to that state or not during the executing of the input string.

Halting problem can be reduced to this problem, and hence it is also undecidable.

Rice Theorem

A property is trivial if either it is not satisfied by any r.e. language, or if it is satisfied by all r.e. languages. Otherwise it is non-trivial.

Theorem: Every non-trivial property of recursively enumerable languages is undecidable.

If P is a non-trivial property, then LP is undecidable.

- Thus $\{ \langle M \rangle \mid L(M) \in P \}$ is not decidable (unless P is trivial)

Reducibility and Undecidability

Language A is reducible to language B (represented as $A \leq B$) if there exists a function f which will convert strings in A to strings in B as:

$$w \in A \iff f(w) \in B$$

Theorem 1: If $A \leq B$ and B is decidable then A is also decidable.

Theorem 2: If $A \leq B$ and A is undecidable then B is also undecidable.

Which of the following languages are undecidable? Note that indicates encoding of the Turing machine M.

$$L_1 = \{ \langle M \rangle \mid L(M) = \Phi \}$$

$$L_2 = \{ \langle M, w, q \rangle \mid M \text{ on input } w \text{ reaches state } q \text{ in exactly 100 steps} \}$$

$$L_3 = \{ \langle M \rangle \mid L(M) \text{ is not recursive} \}$$

$$L_4 = \{ \langle M \rangle \mid L(M) \text{ contains at least 21 members} \}$$

Sol:

L_1 is undecidable as emptiness problem of Turing machine is undecidable. L_3 is undecidable since there is no algorithm to check whether a given TM accept recursive language. L_4 is undecidable as it is similar to membership problem.

Only L_2 is decidable. We can check whether a given TM reach state q in exactly 100 steps or not. Here we have to check only upto 100 steps, so here is not any case of going to infinite loop.

Which of the following problems are decidable?

1. Does a given program ever produce an output?
2. If L is context free language then L' is also context free?
3. If L is regular language then L' is also regular?
4. If L is recursive language then L' is also recursive?

A. 1,2,3,4

B. 1,2

C. 2,3,4

D. 3,4

Sol: As regular and recursive languages are closed under complementation, option 3 and 4 are decidable problems.

Context free languages are not closed under complementation, option 2 is undecidable.

Option 1 is also undecidable as there is no TM to determine whether a given program will produce an output.

Consider three decision problems P_1 , P_2 and P_3 . It is known that P_1 is decidable and P_2 is undecidable. Which one of the following is TRUE?

- A. P_3 is undecidable if P_2 is reducible to P_3
- B. P_3 is decidable if P_3 is reducible to P_2 's complement
- C. P_3 is undecidable if P_3 is reducible to P_2
- D. P_3 is decidable if P_1 is reducible to P_3

Sol: Option A is true

Option B,C,D are false

Which of the following problems is undecidable?

- A. Membership problem for CFGs.
- B. Ambiguity problem for CFGs.
- C. Finiteness problem for FSAs.
- D. Equivalence problem for FSAs.

Sol:

Whether a given CFG is ambiguous, this problem is undecidable. The reason is there is no algorithm exist for this. Remaining all are decidable.