Quantum Series®
ENGINEERING

QUANTUM *Series*

Semester - 4 Computer Science & IT

Theory of Automata & Formal Languages

Session 2019-20 Even Semester

- Topic-wise coverage of entire syllabus in Question-Answer form.
- Short Questions (2 Marks)

Includes solution of following AKTU Question Papers:
2014-15 • 2015-16 • 2016-17 • 2017-18 • 2018-19

Scanned by TapScanner

CONTENTS

KCS 402 : THEORY OF AUTOMATA & FORMAL LANGUAGES

UNIT-1 : BASIC CONCEPT & AUTOMATA THEORY (1–1 A to 1–37 A)

Introduction to Theory of Computation- Automata, Computability & Complexity, Alphabet, Symbol, String, Formal Languages, Deterministic Finite Automaton (DFA)- Definition, Representation, Acceptability of a String and Language, Non Deterministic Finite Automaton (NFA), Equivalence of DFA and NFA, NFA with ε-Transition, Equivalence of NFA's with and without ε-Transition, Finite Automata with output- Moore Machine, Mealy Machine, Equivalence of Moore and Mealy Machine, Minimization of Finite Automata, Myhill-Nerode Theorem, Simulation of DFA and NFA.

UNIT-2 : REGULAR EXPRESSIONS & LANGUAGES (2–1 A to 2–24 A)

Regular Expressions, Transition Graph, Kleen's Theorem, Finite Automata and Regular Expression- Arden's theorem, Algebraic Method Using Arden's Theorem, Regular and Non-Regular Languages- Closure properties of Regular Languages, Pigeonhole Principle, Pumping Lemma, Application of Pumping Lemma, Decidability- Decision properties, Finite Automata & Regular Languages, Regular Languages & Computers, Simulation of Transition Graph & Regular language.

UNIT-3 : REGULAR & NON-REGULAR GRAMMARS (3–1 A to 3–28 A)

Context Free Grammar (CFG)-Definition, Derivations, Languages, Derivation Trees and Ambiguity, Regular Grammars-Right Linear and Left Linear grammars, Conversion of FA into CFG and Regular grammar into FA, Simplification of CFG, Normal Forms- Chomsky Normal Form (CNF), Greibach Normal Form (GNF), Chomsky Hierarchy, Programming problems based on the properties of CFGs.

UNIT-4 : PUSHDOWN AUTOMATA (4–1 A to 4–28 A)

Nondeterministic Pushdown Automata (NPDA)- Definition, Moves, A Language Accepted by NPDA, Deterministic Pushdown Automata (DPDA) and Deterministic Context free Languages (DCFL), Pushdown Automata for Context Free Languages, Context Free grammars for Pushdown Automata, Two stack Pushdown Automata, Pumping Lemma for CFL, Closure properties of CFL, Decision Problems of CFL, Programming problems based on the properties of CFLs.

UNIT-5 : TURING MACHINES (5–1 A to 5–35 A)

Basic Turing Machine Model, Representation of Turing Machines, Language Acceptability of Turing Machines, Techniques for Turing Machine Construction, Modifications of Turing Machine, Turing Machine as Computer of Integer Functions, Universal Turing machine, Linear Bounded Automata, Church's Thesis, Recursive & Recursively Enumerable language, Halting Problem, Post's Correspondance Problem, Introduction to Recursive Function Theory.

SHORT QUESTIONS (SQ-1A to SQ-19A)

SOLVED PAPERS (2014-15 TO 2018-19) (SP-1A to SP-18A)

Scanned by TapScanner

1 UNIT

Basic Concept and Automata Theory

CONTENTS

Part-1 : Introduction to Theory of Computation : Automata, Alphabets, Symbol, Strings, Formal Languages, Computability and Complexity ..... **1–2A to 1–4A**

Part-2 : Deterministic Finite Automaton (DFA) : Definition, Representation, Acceptability of a String and Language ..... **1–4A to 1–9A**

Part-3 : Non-Deterministic Finite Automaton (NFA) ..... **1–9A to 1–11A**

Part-4 : Equivalence of DFA and NFA, NFA with ε-Transition, Equivalence of NFA with and without ε-Transition ..... **1–11A to 1–21A**

Part-5 : Finite Automata with Output : Moore Machine, Mealy Machine, Equivalence of Moore and Mealy Machine ..... **1–21A to 1–30A**

Part-6 : Minimization of Finite Automata, Myhill-Nerode Theorem, Simulation of DFA and NFA ..... **1–30A to 1–36A**

PART-1

Introduction to Theory of Computation : Automata, Alphabets, Symbol, Strings, Formal Languages, Computability and Complexity.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.1. **What do you understand by term alphabets and string in automata theory ? Explain.**

Answer

Alphabet :

1. An alphabet is a finite, non-empty set of a symbol.
2. As a convention we use the Σ symbol to represent an alphabet. It contains the first lower letters of the Latin alphabet (a, b ...) or digit such as 0, 1 to represent symbols.
   For example :
   $$\Sigma = \{a, b, c\}$$
   It is the alphabet composed by the symbols a, b and c.

String :

1. A string (or word) is a sequence of symbols from some alphabets or digits.
2. For example, given the alphabet
   $$\Sigma = \{a, b, c\}$$
   We can build several strings using the symbols a, b and c :
   a
   abc
   aa
   abcabcabcabcabc

Que 1.2. **Explain the term language in reference to automata theory with example.**

Answer

1. A language is set of all strings chosen from some Σ^*, where Σ is particular alphabet.
2. If Σ is an alphabet and $L \subseteq \Sigma^*$, then L is a language over alphabet Σ.
3. A language over Σ need not include strings with all the symbols of Σ. So once we have established that L is a language over Σ, we also know that it is a language over any alphabet which is superset of Σ.

Scanned by TapScanner

4. The only important constraint that can be on language is all strings should be finite
5. Although they can have an infinite number of strings, but restricted to strings drawn from one fixed and finite alphabets.

For example :

1. English words are a set of strings over the alphabets that consist of all the letters
2. The language of all strings consisting of n 0's followed by n 1's for some $n \geq 0$ is
   {ε, 01, 0011, 000111, ...}
3. The set of strings over 0's and 1's with equal number of 0 and 1 is
   {ε, 01, 10, 0011, 1010, 0101, 1100, ...}.

Que 1.3. **Explain arithmetic expression with example.**

Answer

1. An arithmetic expression (AE) is valid combination of input symbols that are read by computer or human being and able to answer a desired result
2. The recursive definition may be defined for validity of an arithmetic expression. The definition includes

a. Any number (positive, negative or zero) is in AE.

b. If 'a' is in AE then

   i. (a) is also in AE.

   ii. $-a$ is also in AE.

c. If 'a' and 'b' are in AE, then the following will also be in AE.

   i. $a + b$

   ii. $a - b$

   iii. $a * b$

   iv. a/b

   v. $a \uparrow b$ or $a ** b$ (Exponentiation)

For example : If Σ is set of alphabets for a language *i.e.*

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, (,)\}$$

Then (4 +3) * 2 is a valid expression but
4 + 3) * 2 is not.

Que 1.4. **Define the grammar.**

Answer

A grammar or phrase-structured grammar is combination of four tuples and can be represented as $G(V, T, P, S)$. Where,

1. V is finite non-empty set of variables/non-terminals. Generally non-terminals are represented by capital letters like $A, B, C, \ldots\ldots, X, Y, Z$.

2. T is finite non-empty set of terminals. Sometimes also represented by Σ or V_T. Generally terminals are represented by $a, b, c, x, y, z, \alpha, \beta, \gamma$ etc.
3. P is finite set whose elements are in the form $\alpha \rightarrow \beta$. Where α and β are strings, made up by combination of V and T *i.e.*, $(V \cup T)$. α has at least one symbol from V. Elements of P are called productions or production rule or rewriting rules.
4. S is special variable/non-terminal known as starting symbol

PART-2

Deterministic Finite Automaton (DFA) : Definition, Representation, Acceptability of a String and Language.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.5. **What do you understand by Deterministic Finite Automata (DFA) and how it is represented ?**

OR

Define Deterministic Finite Automata (DFA).

Answer

1. A Finite Automata (FA) is said to be deterministic, if corresponding to an input symbol, there is single resultant state *i.e.*, there is only one transition.
2. A deterministic finite automata is set of five tuples and represented as

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

Where,

Q : A non-empty finite set of states present in the finite control $(q_0, q_1, q_2, \ldots)$.

Σ : A non-empty finite set of input symbols.

δ : It is a transition function that takes two arguments, a state and an input symbol, it returns a single state. The δ is represented as

$$\delta : Q * \Sigma \rightarrow Q$$

q_0 : It is starting state, one of the state in Q.

F : It is non-empty set of final states/accepting states from the set belonging to Q.

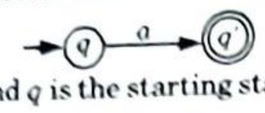
3. Let 'q' is the state and 'a' be the input symbol passed to the transition function.

Scanned by TapScanner

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

q' is the output of function, which may be same or new state.

The graphical representation of transition function is as follows :



The state q' is final state and q is the starting state.

Que 1.6. **Construct a DFA for the language that contains the strings ending with 0.**

Answer

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA

q_0 = initial state = $\{q_0\}$

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

F = final state = $\{q_f\}$

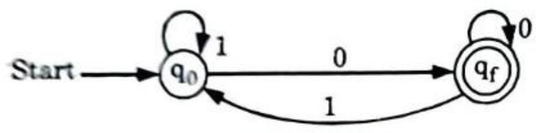


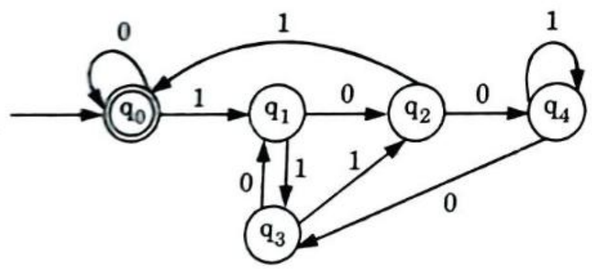
Fig. 1.6.1.

Que 1.7. **Define Deterministic Finite Automata (DFA) and design a DFA that accepts the binary number whose equivalent is divisible by 5.** **AKTU 2017-18, Marks 07**

Answer

DFA : Refer Q. 1.5, Page 1–4A, Unit-1.

Numerical :



Que 1.8. **Draw DFA for following over set $\Sigma = \{0, 1\}$**

i. $L = \{w : |w| \bmod 3 = 0\}$

ii. $L = \{w : |w| \bmod 3 > 1\}$

Answer

i.

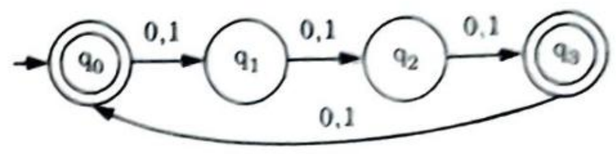


Fig. 1.8.1.

ii.

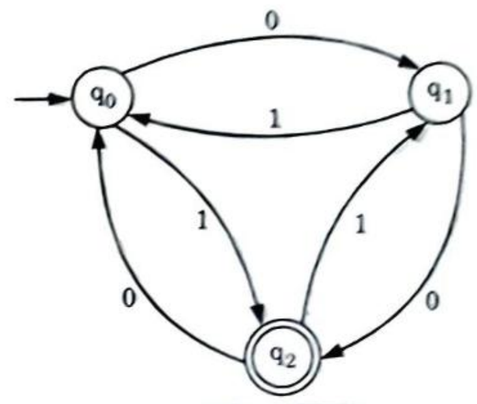


Fig. 1.8.2.

Que 1.9. **Design a FA which accepts set of strings containing exactly four 1's in every string over $\Sigma = \{0, 1\}$.** **AKTU 2014-15, Marks 05**

Answer

DFA should accept the strings such as 1111, 0101011, 011011000.....

Let DFA be $M = (Q, \Sigma, \delta, q_0, F)$

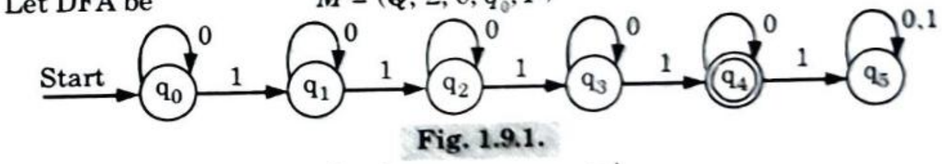


Fig. 1.9.1.

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

$F = \{q_4\}$

In this q_5 is dead state.

Que 1.10. **Design a DFA for the language**

$L = \{w$: every run of a's has either two three$\}$.

Answer

The transition diagram for language

Scanned by TapScanner

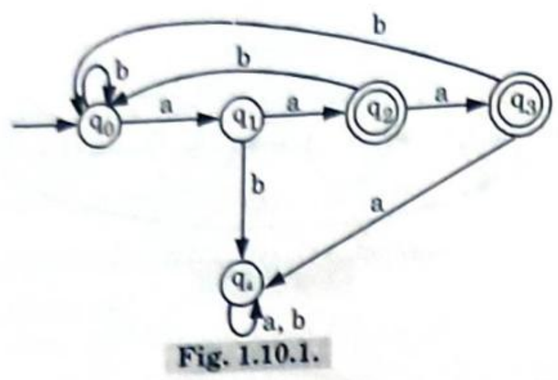


Fig. 1.10.1.

Que 1.11. **Draw the finite automata which accept all the strings containing both 11 and 010 as sub-strings.**

Answer

1. The set of strings for the language is 110101, 1100101, 101011, ...
2. The string will be accepted by the DFA if both the sub-sequence 11 and 010 are present.
3. Both 11 and 010 can follow each other in the language.
4. So, the DFA corresponding to the given language can be represented by the following transition diagram :

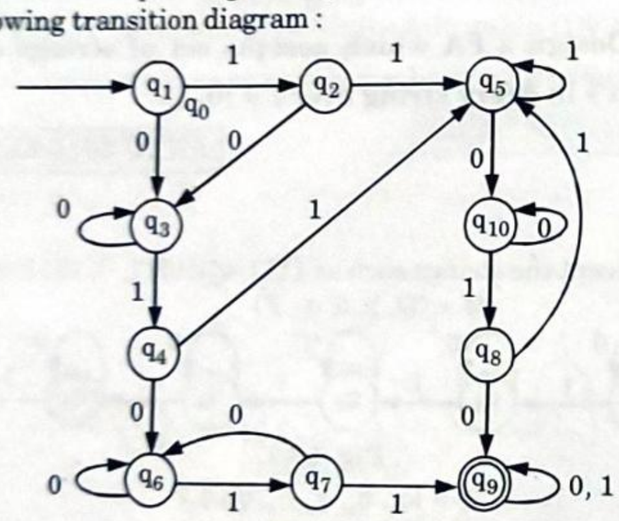


Fig. 1.11.1.

Que 1.12. **Design a finite automata which accepts the complement of the language accepted by the following automata :**

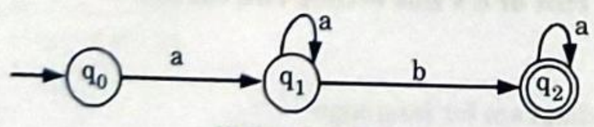


Fig. 1.12.1.

Answer

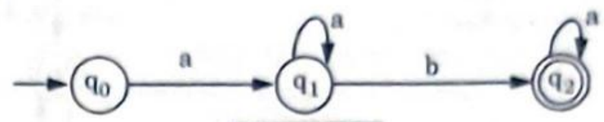


Fig. 1.12.2.

1. We know that if L is regular than complement of L that is $\overline{L}$ is also regular. That is
   $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that accepts language L; then DFA that accepts language $\overline{L}$ is
   $$\overline{M} = (Q_1, \Sigma, \delta, q_0, Q - F)$$
2. DFA which accepts the complement of language L is given by changing every non-final state to final state and every final state to non-final.
3. Thus, the required DFA is

Fig. 1.12.3.

Que 1.13. **Draw DFA of following over {0, 1} :**

i. All strings with even number of 0's and even number of 1's.

ii. All strings of length at most 5.

Answer

i. DFA should accept the strings such as 11, 00, 1010, 0101, 1111, 101101, ... etc. The transition diagram is given by

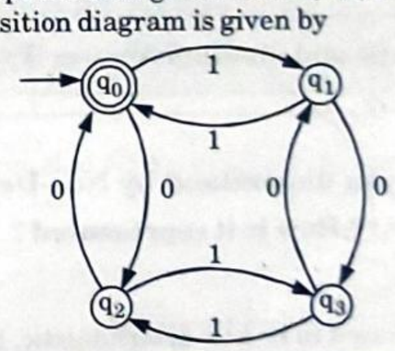


Fig. 1.13.1. Transition diagram.

ii. DFA should accept the string of length 5 such as 10110, 00000, 11111, 01101, ... etc. The transition diagram is given by

Scanned by TapScanner

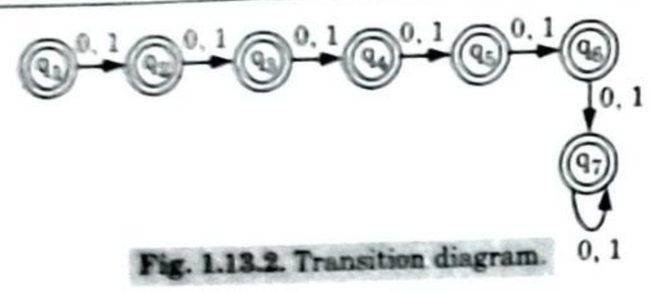


Fig. 1.13.2. Transition diagram.

Que 1.14. **What do you mean by language of a DFA ?**

Answer

1. The set of all strings that result in a sequence of state transition from start state to an accepting state.
2. Now we can define the language of a DFA $M = (Q, \Sigma, \delta, q_0, F)$. This language is denoted by $L(M)$, and defined by

$$L(M) = \{w \,/\, \hat{\delta}(q_0, w) \text{ is in } F\}$$

3. The language of M is the set of strings (w) that take the start state q_0 to one of the accepting states.
4. If L is $L(M)$ for some deterministic finite automata, then we say L is a regular language.

PART-3

Non-Deterministic Finite Automaton (NFA).

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.15. **What do you understand by Non-Deterministic Finite Automata (NFA/NDFA) ? How is it represented ?**

Answer

1. A finite automata is said to be non-deterministic, if there is more than one possible transition from one state on the same input symbol.
2. A Non-Deterministic Finite Automata is also a set of five tuples and represented as

$$M = (Q, \Sigma, \delta, q_0, F) \qquad \text{where,}$$

Q : A set of non-empty finite states.
Σ : A set of non-empty finite input symbols.
q_0 : Initial state of NFA and member of Q.

F : A non-empty finite set of final states and member of Q.
δ : It is transition function that takes a state from Q and an input symbol from Σ and returns a subset of Q. The δ is represented as

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$$

3. The graphical representation of δ is as follows :

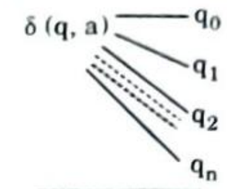


Fig. 1.15.1.

Que 1.16. **Construct a NFA for the language L which accept all the strings in which the third symbol from right end is always a over $\Sigma = \{a, b\}$.** **AKTU 2015-16, Marks 10**

Answer

Let NFA be

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

The δ is defined as follows :

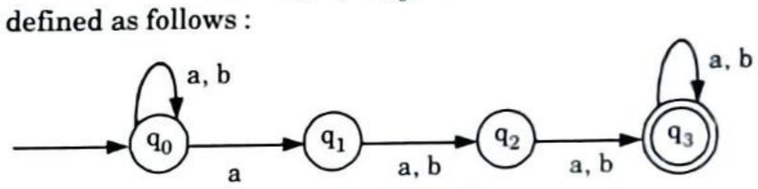


Fig. 1.16.1.

Que 1.17. **Design a NFA for the language L which accepts all strings over {0, 1} that have at least two consecutive 0's or 1's.**

Answer

By the analysis it is clear that NFA will accept the strings of the patterns like 00, 11, 101000, 101100, ...
The transition diagram is given by

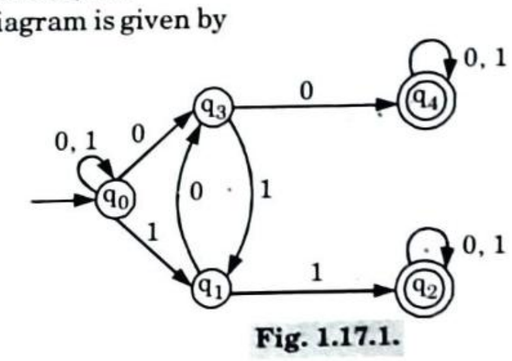


Fig. 1.17.1.

Scanned by TapScanner

Que 1.18. **What do you understand by language of a NFA ?**

Answer

1. An NFA accepts a string w if it is possible to make any sequence of choice of next state while reading the characters of w, and go from the start state to any accepting state.
2. The language of NFA $M = (Q, \Sigma, \delta, q_0, F)$ is defined by

$$L(M) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \phi\}$$

3. $L(M)$ is the set of strings w in Σ^* such that $\hat{\delta}(q_0, w)$ contains at least one accepting state.

PART-4

Equivalence of DFA and NFA, NFA with ε-Transition, Equivalence of NFA with and without ε-Transition.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.19. **What do you mean by NFA with ε-transition ? Why we need it ?**

Answer

If a finite automata is modified to permit transition without input symbols, along with zero, one or more transitions on input symbols, then we get NFA with ε-transitions.

We need it to concatenate two different languages.

For example : We have design NFA for accepting language

$$L = \{(ab)^* \cup aa^*\}$$

To solve this problem first divide the language

- $L = L_1 \cup L_2$, Where $L_1 = (ab)^*$ and $L_2 = aa^*$

First construct NFA for L_1

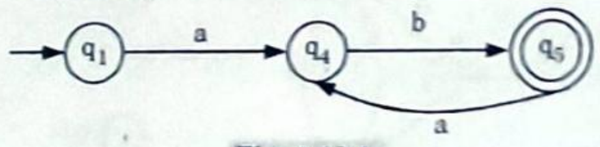


Fig. 1.19.1.

Secondly, we construct NFA for L_2

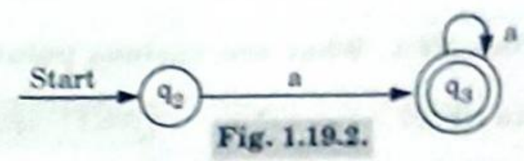


Fig. 1.19.2.

Now we combine two transition diagram *i.e.*, Fig. 1.19.1 and 1.19.2 as follows :

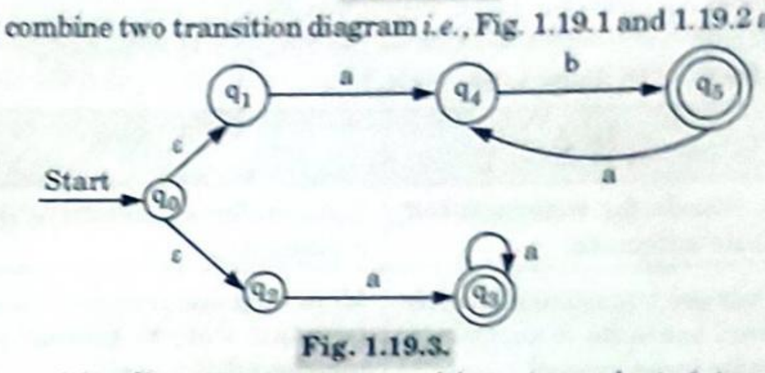


Fig. 1.19.3.

In this transition diagram we use ε-transitions, to reach at states q_1 and q_2.

Que 1.20. **Explain the working of ε-closure, with a suitable example.**

Answer

1. A string w in Σ^* will be accepted by NFA with ε-transition, if there exists at least one path corresponding 'w' which starts from initial state and ends at final state.
2. Since, this path may or may not contain ε-transition.
3. If the path contains ε-moves, then we need to define a function ε-closure (q), where 'q' is state of automata.
4. The function ε-closure is defined as follows :

ε-closure (q) = set of all those states of automata which can be reached from 'q' on a path labeled by ε *i.e.*, without consuming any input symbol. Consider the NFA

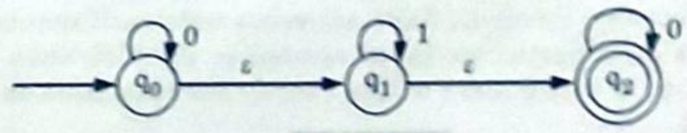


Fig. 1.20.1.

ε-closure $(q_0) = \{q_0, q_1, q_2\}$

ε-closure $(q_1) = \{q_1, q_2\}$

ε-closure $(q_2) = \{q_2\}$

Scanned by TapScanner

Que 1.21. **Define NFA. What are various points of difference between NFA and DFA ?**

AKTU 2018-19, Marks 10

Answer

NFA : Refer Q. 1.15, Page 1–9A, Unit-1.

S. No.	DFA	NFA
1.	It stands for deterministic finite automata.	It stands for non-deterministic finite automata.
2.	Only one transition is possible from one state to another on same input symbol.	More than one transition is possible from one state to another on same input symbol.
3.	Transition function δ is written as : $\delta : Q \times \Sigma \to Q$	Transition function δ is written as : $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$
4.	In DFA, ε-transition is not possible.	In NFA, ε-transition is possible.
5.	DFA cannot be converted into NFA.	NFA can be converted into DFA.

Que 1.22. **Can we convert a NFA into a DFA ?**

Answer

1. Yes, we can convert a NFA into DFA. For every NFA there exists an equivalent DFA.
2. The equivalence is defined in terms of language acceptance. Since NFA is nothing but a finite automata in which zero, one or more transition on an input symbols are permitted.
3. We can always construct finite automata which will simulate all moves of NFA on a particular input symbol in parallel, then get a finite automata in which there will be exactly one transition on every input symbol.

Que 1.23. **Explain the procedure for converting NFA to equivalent DFA.**

Answer

Procedure for converting NFA to equivalent DFA :

1. Let $M_2 = (Q_2, \Sigma, q'_0, \delta_2, F_2)$ be NFA that recognizes the languages L. Then the DFA $M = (Q, \Sigma, q_0, \delta, F)$ that satisfies the following conditions recognizes L :

$Q = 2^{Q_2}$, *i.e.*, is the set of all subsets of Q_2.

$q_0 = \{q'_0\}$

$\delta(q, a) = \bigcup_{p \in q} \delta(p, a)$ for each state q in Q and each symbol a in Σ and

$F = \{q \in Q \mid q \cap F_2 \neq \phi\}$.

2. To obtain a DFA $M = (Q, \Sigma, \delta, q_0, F)$ which accepts the same language as given NFA $M_2 = (Q_2, \Sigma, q'_0, \delta_2, F_2)$ does, we may proceed as follows :

Step 1 : Initially $Q = \phi$.

Step 2 : Put $\{q'_0\}$ into Q. $\{q'_0\}$ is the initial state of the DFA M.

Step 3 : Then for each state q in Q do the following :

a. add this new state.

b. add $\delta(q, a) = \bigcup_{p \in q} \delta(p, a)$ to δ, where the δ on the right hand side is that of NFA M_2.

Step 4 : Repeat step 3 till new states are there to add in Q, if there is new state found to add in Q, the process terminates. All the states of Q that contain accepting states of M_2 are accepting states of M.

Que 1.24. **Construct DFA equivalent to NFA where δ is defined in the following table :**

Q	$\delta(q, a)$	$\delta(q, b)$
A	A, B	C
B	A	B
C^*(final state)	–	A, B

AKTU 2015-16, Marks 10

Answer

$$\delta(A, a) = \{A, B\}$$
$$\delta(A, b) = \{C\}$$
$$\delta(\{A, B\}, a) = \delta(A, a) \cup \delta(B, a) = \{A, B\} \cup \{A\} = \{A, B\}$$
$$\delta(\{A, B\}, b) = \delta(A, b) \cup \delta(B, b) = \{C\} \cup \{B\} = \{B, C\}$$
$$\delta(C, a) = \{\phi\}$$
$$\delta(C, b) = \{A, B\}$$
$$\delta(\{B, C\}, a) = \delta(B, a) \cup \delta(C, a) = \{A\} \cup \{\phi\} = \{A\}$$
$$\delta(\{B, C\}, b) = \delta(B, b) \cup \delta(C, b) = \{B\} \cup \{A, B\} = \{A, B\}$$

Let,
$$A \to q_0$$
$$C \to q_1$$
$$\{A, B\} \to q_2$$
$$\{B, C\} \to q_3$$

Transition table for DFA,

Scanned by TapScanner

δ/Σ	a	b
$\rightarrow q_0$	q_2	q_1
$*q_1$	$\{\phi\}$	q_2
q_2	q_2	q_3
$*q_3$	q_0	q_2

Transition diagram for DFA,

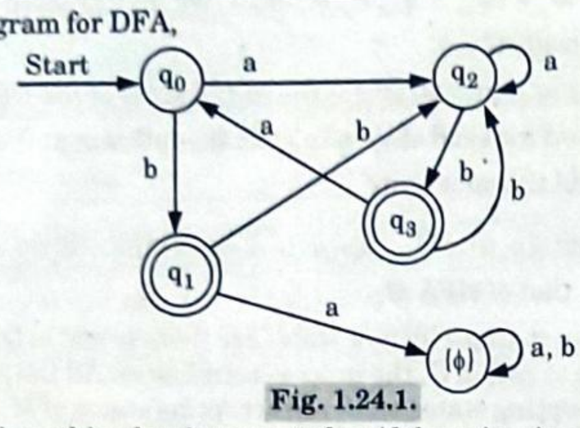


Fig. 1.24.1.

Here $\{\phi\}$ state is nothing but interpreted as if there is a input 'a' on state q, then there is no path for it and machine will be in dead stage.

Que 1.25. **Convert the following NFA $\{p, q, r, s\}, \{0, 1\}, \delta, p, \{q, s\}$ into DFA where δ is given by**

δ/Σ	0	1
$\rightarrow p$	q, s	q
$*q$	r	q, r
r	s	p
$*s$	ϕ	p

AKTU 2018-19, Marks 10

Answer

The transition table for DFA will be :

δ/Σ	0	1
p	$\{q, s\}$	$\{q\}$
q	$\{r\}$	$\{q, r\}$
r	$\{s\}$	$\{p\}$
s	–	$\{p\}$
$\{q, s\}$	$\{r\}$	$\{p, q, r\}$
$\{q, r\}$	$\{r, s\}$	$\{p, q, r\}$
$\{r, s\}$	$\{s\}$	$\{p\}$
$\{p, q, r\}$	$\{q, r, s\}$	$\{p, q, r\}$
$\{q, r, s\}$	$\{r, s\}$	$\{p, q, r\}$

Now let, $p \rightarrow A$
$q \rightarrow B$
$r \rightarrow C$
$s \rightarrow D$
$\{q, s\} \rightarrow E$
$\{q, r\} \rightarrow F$
$\{r, s\} \rightarrow G$
$\{p, q, r\} \rightarrow H$
$\{q, r, s\} \rightarrow I$

Therefore, transition table for DFA after relabeling and tuples will be :

δ/Σ	0	1
$\rightarrow A$	E	B
$*B$	C	F
C	D	A
$*D$	–	A
$*E$	C	H
$*F$	G	H
$*G$	D	A
$*H$	I	H
$*I$	G	H

$Q = \{A, B, C, D, E, F, G, H, I\}$
$\Sigma = \{0, 1\}$
$q_0 = \{A\}$
$F = \{B, D, E, F, G, H, I\}$

∴ Transition diagram will be :

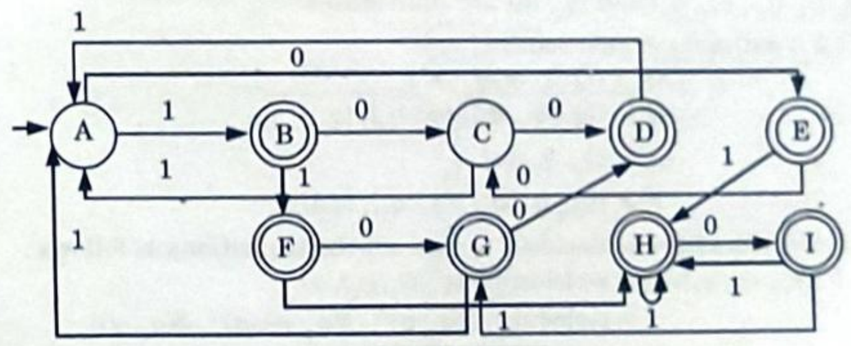


Fig. 1.25.1.

Que 1.26. **Construct a NFA without ε-moves corresponding to following ε-NFA. Explain each step.**

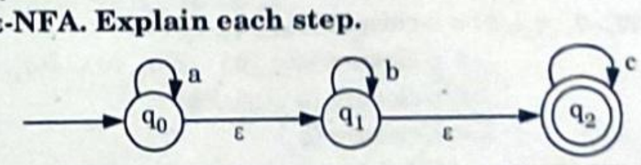


Fig. 1.26.1. NFA with ε-transition (ε-NFA).

Answer

Transition table for ε-NFA

Scanned by TapScanner

Σ/δ	a	b	c	ε
q_0	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_2	ϕ	q_2
q_2	ϕ	ϕ	q_2	ϕ

Step 1 : Find the states of NFA without ε-transition including initial state and final states as follows :

i. There will be the same number of states but the names can be constructed by writing the state name as the set of states in the ε-closure.

ii. Initial state will be ε-closure of initial state of NFA with ε-transition in Fig. 1.26.1 as

$\varepsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\}$ (New initial state for NFA, without ε-transition)

Rest of the states are

$\varepsilon\text{-closure}(q_1) = \{q_1, q_2\}$ (New state)

$\varepsilon\text{-closure}(q_2) = \{q_2\}$ (New state)

iii. The final states of NFA without ε-transition are all those new states which contain final state of NFA with ε-transition as member.

So $\{q_0, q_1, q_2\}$, $\{q_1, q_2\}$ and $\{q_2\}$ all are final states.

So if NFA without ε-transition is

$$M' = (Q', \Sigma, \delta', q_0', F')$$
$$Q' = (\{q_0, q_1, q_2\}, \{q_1, q_2\}, \{q_2\})$$
$$q_0' = \{q_0, q_1, q_2\}$$
$$F' = (\{q_0, q_1, q_2\}, \{q_1, q_2\}, \{q_2\})$$

Step 2 : Now we have to decide δ′ to find out the transitions as follows :

$$\begin{aligned}\delta'(\{q_0, q_1, q_2\}, a) &= \varepsilon\text{-closure}(\delta(q_0, q_1, q_2), a)\\ &= \varepsilon\text{-closure}(\delta(q_0, a) \cup \delta(q_1, a), a) \cup \delta(q_2, a))\\ &= \varepsilon\text{-closure}(q_0, \cup \phi \cup \phi)\\ &= \varepsilon\text{-closure}(q_0)\\ &= \{q_0, q_1, q_2\}\end{aligned}$$

Similarly

$$\begin{aligned}\delta'(\{q_0, q_1, q_2\}, b) &= \varepsilon\text{-closure}(\delta(q_0, q_1, q_2), b)\\ &= \varepsilon\text{-closure}(\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b))\\ &= \varepsilon\text{-closure}(\phi \cup q_1 \cup \phi)\\ &= \varepsilon\text{-closure}(q_1)\\ &= \{q_1, q_2\}\end{aligned}$$

$$\begin{aligned}\delta'(\{q_0, q_1, q_2\}, c) &= \varepsilon\text{-closure}(\delta(q_0, q_1, q_2), c)\\ &= \varepsilon\text{-closure}(\delta(q_0, c) \cup \delta(q_1, c) \cup \delta(q_2, c))\\ &= \varepsilon\text{-closure}(\phi \cup \phi \cup q_2)\\ &= \varepsilon\text{-closure}(q_2)\\ &= \{q_2\}\end{aligned}$$

Similarly we can write

$$\delta'(\{q_1, q_2\}, a) = \phi$$
$$\delta'(\{q_1, q_2\}, b) = \{q_1, q_2\}$$
$$\delta'(\{q_1, q_2\}, c) = \{q_2\}$$
$$\delta'(q_2, a) = \phi$$
$$\delta'(q_2, b) = \phi$$
$$\delta'(q_2, c) = \{q_2\}$$

So transition table for NFA without ε-transition will be as following :

δ′/Σ	a	b	c
→ * $\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
* $\{q_1, q_2\}$	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
* $\{q_2\}$	ϕ	ϕ	$\{q_2\}$

Let us say $\{q_0, q_1, q_2\}$ as q_x

$\{q_1, q_2\}$ as q_y

and $\{q_2\}$ as q_z

So transition diagram is as follows :

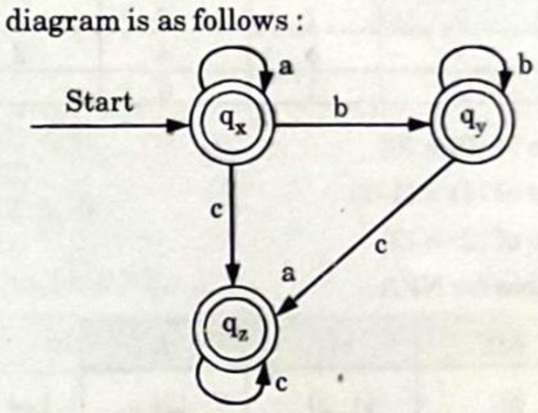


Fig. 1.26.2. NFA without ε-transitions.

Que 1.27.

i. Convert the NFA-ε to DFA.

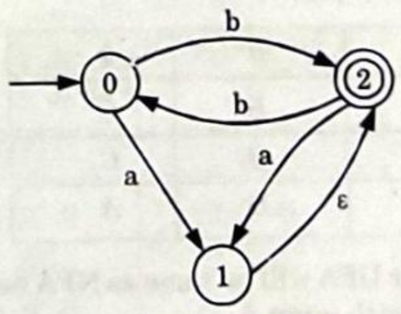


Fig. 1.27.1.

Scanned by TapScanner

ii. Check with the comparison method for testing equivalence of two FA given below.

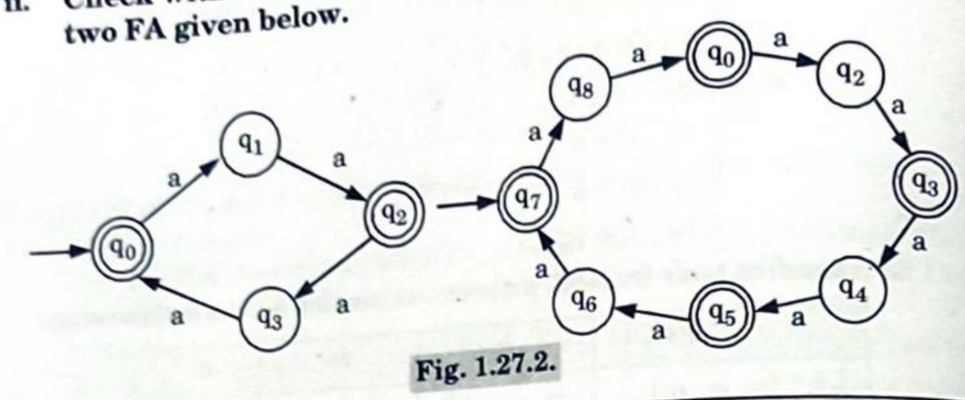


Fig. 1.27.2.

AKTU 2016-17, Marks 10

Answer

i. First we convert NFA-ε to NFA.

Transition table for NFA-ε :

δ/Σ	a	b	ε
0	1	2	
1	ϕ	ϕ	2
2	1	0	

ε-closure of (0) = {0}

ε-closure of (1) = {1, 2}

ε-closure of (2) = {2}

Transition table for NFA :

δ/Σ	a	b
{0}	{1, 2}	{2}
{1, 2}	{1, 2}	{2}
{2}	{1, 2}	{0}

Let {0} as *A*
{1, 2} as *B*
{2} as *C*

Transition table for NFA :

δ/Σ	a	b
A	*B*	*C*
B	*B*	*C*
C	*B*	*A*

Transition table for DFA will be same as NFA because there is only one transition from *A* with input a.
So DFA is given as

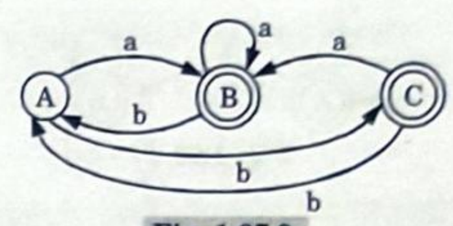


Fig. 1.27.3.

ii. Language accepted by both FAs

$L = \{w \in \{a\}/\text{string with even number of } a\}$

So, both, FAs are equivalent as they accept same set of string.

Que 1.28. Compute the epsilon-closure for the given NFA. Convert it into DFA.

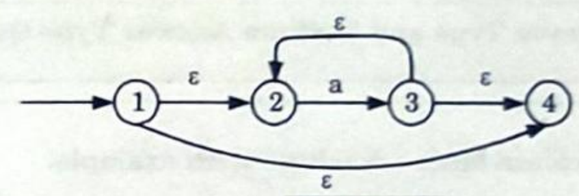


Fig. 1.28.1.

AKTU 2016-17, Marks 7.5

Answer

ε-closure of (1) = {1, 2, 4}
ε-closure of (2) = {2}
ε-closure of (3) = {2, 3, 4}
ε-closure of (4) = {4}

Transition table for ε-NFA :

δ/Σ	a	ε
1	ϕ	{2, 4}
2	3	ϕ
3	ϕ	{4, 2}
4	ϕ	ϕ

Transition table for NFA :

Let {1, 2, 4} as *A*
{2, 3, 4} as *B*

δ/Σ	a
{1, 2, 4}	{2, 3, 4}
{2}	{2, 3, 4}
{2, 3, 4}	{2, 3, 4}
{4}	ϕ

Scanned by TapScanner

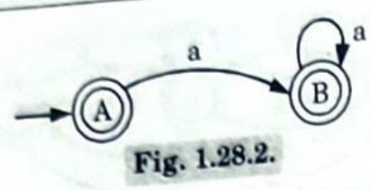


Fig. 1.28.2.

PART-5

Finite Automata with Output : Moore Machine, Mealy Machine, Equivalence of Moore and Mealy Machine.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.29. **Explain Moore machine with example.**

Answer

1. Moore machine is a finite automata in which output is associated with each state.
2. The output symbol at a given time depends only upon the present state of machine.
3. Mathematically, Moore machine is a six tuple machine and defined as

$$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$$

where,

Q : A non-empty finite set of states.

Σ : A non-empty finite set of input symbol.

Δ : A non-empty finite set of outputs.

δ : A transition function which takes two arguments as in finite automata, one is input state and another is input symbol. The output of this function is a single state and represented by

$$\delta : Q * \Sigma \rightarrow Q$$

λ : It is a mapping function, which maps Q to Δ, giving the output associated with each state. It is represented by $\lambda : Q \rightarrow \Delta$

q_0 : Initial state of machine.

4. When machine is in a particular state, it produces the output, irrespective of what was the input on which transition is made.

Representation of Moore machine :

Moore machine can be represented by transition table as well as transition diagram same as finite automata.

Let M be Moore machine having following transition table with $\Sigma = \{0, 1\}$

Present state	Next state at Input		Output
	$a = 0$	$a = 1$	
$\rightarrow q_0$	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

$$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$$
$$Q = \{q_0, q_1, q_2, q_3\}$$
$$\Sigma = \{0, 1\}$$
$$\Delta = \{0, 1\}$$
$$\lambda(q_0) = 0, \lambda(q_1) = 1, \lambda(q_2) = 0, \lambda(q_3) = 0$$

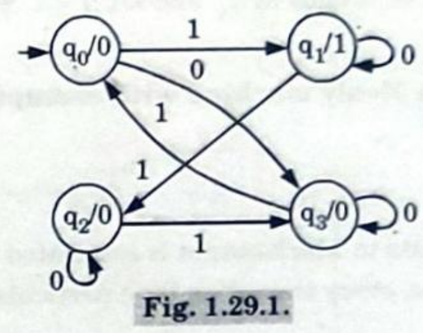


Fig. 1.29.1.

q_0 is the initial state.

Que 1.30. **Design a Moore machine which determines the residue Mod-3 for each binary string treated as binary integer.**

Answer

We have to design a Moore machine which prints the remainder, when decimal equivalent is divided by 3.

Therefore, $\Delta = \{0, 1, 2\}$

Since, 0, 1 and 2 are possible remainder, when a decimal number is divided by the 3.

Therefore, we need three states in Moore machine.

Let Moore machine $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$

$$Q = \{q_0, q_1, q_2\}$$
$$\Sigma = \{0, 1\}$$
$$\Delta = \{0, 1, 2\}$$
$$\lambda(q_0) = 0, \lambda(q_1) = 1, \lambda(q_2) = 2$$

q_0 is initial state.

The transition table is

Scanned by TapScanner

Present state	Next state at Input		Output
	$a = 0$	$a = 1$	
$\rightarrow q_0$	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_1	q_2	2

and transition diagram will be

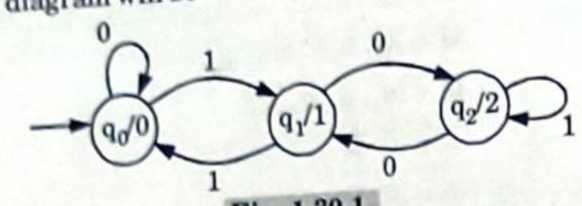


Fig. 1.30.1.

Let us assume input string is 111, decimal equivalent is 7.7/3 = 1 (remainder) when we pass 111 to M, it ends at q_1, and $\lambda(q_1) = 1$, which is remainder of 7/3.

Que 1.31. Explain Mealy machine with example.

Answer

Mealy machine :

1. It is finite automata in which output is associated with each transition.
2. In Mealy machine, every transition for a particular input symbol has a fixed output.
3. Mathematically, Mealy machine is a six tuple machine and defined as

Where, $M = \{Q, \Sigma, \Delta, \delta, \lambda, q_0\}$

Q : A non-empty finite set of state in M.

Σ : A non-empty finite set of input symbol.

Δ : A non-empty finite set of outputs.

q_0: Initial state of M.

δ : A transition function which takes two arguments, one is input state and another is input symbol. The output of this function is a single state and represented as $\delta : Q * \Sigma \rightarrow Q$

λ : It is mapping/machine function which maps $Q * \Sigma$ to Δ, giving the output associated with each transition. It is represented by

λ : $Q * \Sigma \rightarrow \Delta$

Thus, for this type of machine output depends on both current state and the current input symbol.

Representation of Mealy machine :

Mealy machine can be represented by transition table as well as transition diagram.

Let M be a Mealy machine having following transition table with $\Sigma = \{0, 1\}$.

Present state	For input $a = 0$		For input $a = 1$	
	State	Output	State	Output
q_1	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

$M = \{Q, \Sigma, \Delta, \delta, \lambda, q_0\}$

$Q = \{q_1, q_2, q_3, q_4\}$

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

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

$\lambda(q_1, 0) = 0$ $\quad$ $\lambda(q_1, 1) = 0$

$\lambda(q_2, 0) = 1$ $\quad$ $\lambda(q_2, 1) = 0$

$\lambda(q_3, 0) = 1$ $\quad$ $\lambda(q_3, 1) = 1$

$\lambda(q_4, 0) = 1$ $\quad$ $\lambda(q_4, 1) = 0$

The transition diagram

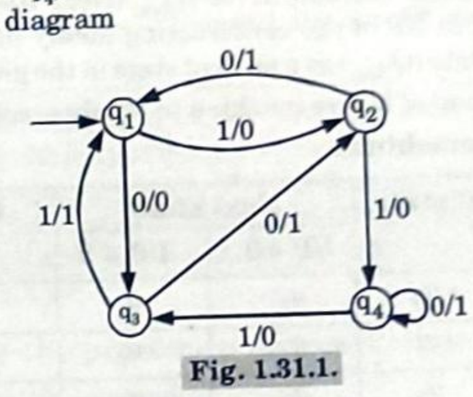


Fig. 1.31.1.

Here the output is associated with each input.

Que 1.32. Design a Mealy machine, which prints 1's complement of input bit string over alphabet $\Sigma = \{0, 1\}$.

Answer

If input string is 101110 then 1's complements of 101110 will be 010001, so, we have to design a Mealy machine which will point output 010001 for input string 101110.

Let $M_e = (Q, \Sigma, \Delta, \delta, \lambda', q_0)$

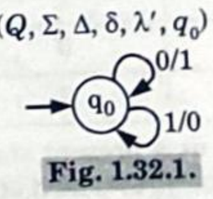


Fig. 1.32.1.

$Q = \{q_0\}$

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

Scanned by TapScanner

q_0 is also initial state.

Let us pass input $w = 011$ to M_e

$$q_0 \xrightarrow{0/1} q_0 \xrightarrow{1/0} q_0 \xrightarrow{1/0} q_0$$

Output string is $w' = 100$ which 1's complement of $w = 011$.

Que 1.33. Write the procedure for converting a Moore machine to a Mealy machine with an example.

Answer

1. Draw the tabular format of Mealy machine.
2. Put the present states of the Moore machine in the present state column of constructing Mealy machine.
3. Put the next states of Moore machine for corresponding present states and input combination, in the next state columns of the constructing Mealy machine for the same present states and input combination.
4. For the output, look into the present state column and output column of the Moore machine. The output for Q_{Next} (Next state for present state $Q_{Present}$ and input I/P of the constructing Mealy machine) will be the output of that state (Q_{Next}) as a present state in the given Moore machine.

Example : Conversion of Moore machine to Mealy machine.

Given Moore machine :

Present state	Next state		O/P
	I/P = 0	I/P = 1	
$\rightarrow q_0$	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_3	0

Mealy machine :

Present state	I/P = 0		I/P = 1	
	Next state	O/P	Next state	O/P
$\rightarrow q_0$	q_3	0	q_1	1
q_1	q_1	1	q_2	0
q_2	q_2	0	q_3	0
q_3	q_3	0	q_3	0

In the Moore machine, for q_1 as a present state the output is 1. So in the constructing Mealy machine for q_1 as a next state the output will be 1.

In the Moore machine, for q_2 as a present state the output is 0. Hence in the constructing Mealy Machine for q_2 as a next state the output will be 0.

Que 1.34. What are various points of difference between Moore and Mealy machine ? Explain the procedure to convert a Moore machine into Mealy machine. **AKTU 2018-19, Marks 10**

Answer

Difference between Moore and Mealy machine :

S. No.	Moore machine	Mealy machine
1.	Its output depends only on present state.	Its output depends on the transition (input) and present state.
2.	Its transition function is excess into Q (finite set of states).	Its transition function is excess into D (output alphabet).
3.	If input string is of length n the output string is of length $n + 1$.	If input string is of length n then the output string is of length n.
4.	At the different input on the same state, its output is same.	At the different input on the same state, its output is different.

Procedure to convert a Moore machine into Mealy machine : Refer Q. 1.33, Page 1–25A, Unit-1.

Que 1.35. Write the procedure for converting Mealy machine to Moore machine with example.

Answer

1. Draw the tabular format of Mealy machine.
2. Check the next state and output columns of the given Mealy machine.
3. If for same next state in the next state column, the output differs in the output column, break the state q_i into different number of states. The number is equal to the number of different outputs associated with q_i.
4. Put the states of the present state column in the new table. The states which are broken into number of different states put the broken states in the place of those states.
5. Change the next states in the next state columns according to the new set of states.
6. Put the output from the output column of the original Mealy machine in the new machine.
7. Draw the tabular format of Moore machine.

Scanned by TapScanner

8. Put the present states and next states from the constructed new Mealy machine to the constructed Moore machine.
9. For output check the next state and output column of the new constructed Mealy machine. For the state let q_i as a next state in the new constructed Mealy machine if output is 0 then for q_i as a present state in the constructing Moore machine; the output will be 0.
10. For Moore machine the output depends only on the present state. This means from the beginning state for ε-input we can get an output. If the output is 1, the new constructed Moore machine can accept zero length string, which was not accepted by given Mealy machine.
11. To make the Moore machine does not accept ε-string, we have to add an extra state q_b (new beginning state), whose state transactions will be identical with those of the existing beginning state but output is 0.

Example : Conversion of Mealy machine to Moore machine.

Given Mealy machine :

Present state	I/P = 0		I/P = 1	
	Next state	O/P	Next state	O/P
$\rightarrow q_0$	q_0	1	q_1	0
q_1	q_3	1	q_3	1
q_2	q_1	1	q_2	1
q_3	q_2	0	q_0	1

Check the next state and output columns of the given Mealy machine. For I/P 0 for q_1 as a next state output is 1. For I/P 1 for q_1 as a next state output is 0. Same thing happens for q_2 as a next state for input 0 and input 1. So the state q_1 will be broken as q_{10} and q_{11} and the state q_2 will be broken as q_{20} and q_{21}. After breaking the modified Mealy machine will be

Present state	I/P = 0		I/P = 1	
	Next state	O/P	Next state	O/P
$\rightarrow q_0$	q_0	1	q_{10}	0
q_{10}	q_3	1	q_3	1
q_{11}	q_3	1	q_3	1
q_{20}	q_{11}	1	q_{21}	1
q_{21}	q_{11}	1	q_{21}	1
q_3	q_{20}	0	q_0	1

For present state q_0 for input 1 the next state will be q_{10} because there is no q_1 in the modified Mealy machine. It has been broken into q_{10} and q_{11} depending on the output 0 and 1, respectively.

From this the Moore machine will be

Present state	Next state		O/P
	I/P = 0	I/P = 1	
$\rightarrow q_0$	q_0	q_{10}	0
q_{10}	q_3	q_3	1
q_{11}	q_3	q_3	1
q_{20}	q_{11}	q_{21}	1
q_{21}	q_{11}	q_{21}	1
q_3	q_{20}	q_0	1

For Moore machine beginning states q_0 and the output is 1. That means with null length input (no input) we are getting an output 1. Therefore, the Moore machine accepts 0 length sequence which is not acceptable for mealy machine. To overcome this situation we must add a new beginning state q_b with same transactions as q_0 but output 0. By including the new state the Moore machine will be :

Moore machine :

Present state	Next state		O/P
	I/P = 0	I/P = 1	
$\rightarrow q_b$	q_0	q_{10}	0
q_0	q_0	q_{10}	1
q_{10}	q_3	q_3	0
q_{11}	q_3	q_3	1
q_{20}	q_{11}	q_{21}	0
q_{21}	q_{11}	q_{21}	1
q_3	q_{20}	q_0	1

Que 1.36. **Describe Mealy and Moore machines with example. Convert the given Mealy machine as shown in Fig. 1.36.1 into Moore machine.**

Scanned by TapScanner

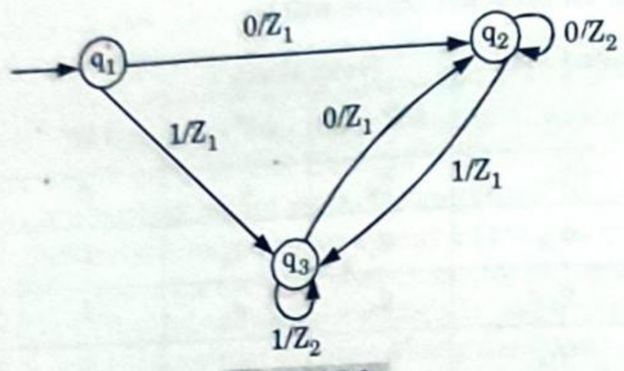


Fig. 1.36.1.

AKTU 2017-18, Marks 07

Answer

Mealy machine : Refer Q. 1.31, Page 1–23A, Unit-1.
Moore machine : Refer Q. 1.29, Page 1–21A, Unit-1.
Numerical :

1. Let us convert the transition diagram into the transition Table 1.36.1.
2. For the given problem : q_1 is not associated with any output; q_2 is associated with two different outputs Z_1 and Z_2; q_3 is associated with two different outputs Z_1 and Z_2.
3. Thus we must split q_2 into q_{21} and q_{22} with outputs Z_1 and Z_2, respectively and q_3 into q_{31} and q_{32} with outputs Z_1 and Z_2, respectively. Table 1.36.1 may be reconstructed as Table 1.36.2.

Table 1.36.1. Transition table

Present state	Next state			
	$a = 0$		$a = 1$	
	state	output	state	output
$\rightarrow q_1$	q_2	Z_1	q_3	Z_1
q_2	q_2	Z_2	q_3	Z_1
q_3	q_2	Z_1	q_3	Z_2

Table 1.36.2. Transition table of Moore machine

Present state	Next state		Output
	$z = 0$	$a = 1$	
$\rightarrow q_1$	q_{21}	q_{31}	
q_{21}	q_{22}	q_{31}	Z_1
q_{22}	q_{22}	q_{31}	Z_2
q_{31}	q_{21}	q_{32}	Z_1
q_{32}	q_{21}	q_{32}	Z_2

Fig. 1.36.1 gives the transition diagram of the required Moore machine.

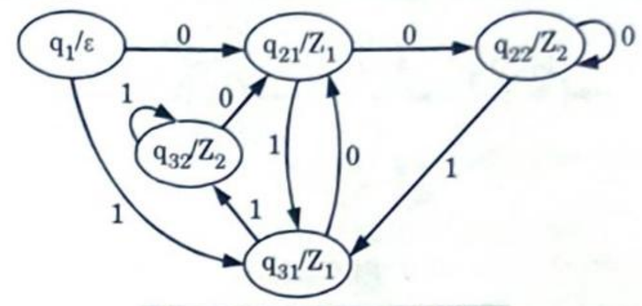


Fig. 1.36.1. Moore machine.

PART-6

Minimization of Finite Automata, Myhill-Nerode Theorem, Simulation of DFA and NFA.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.37. **What is the need of minimization of finite automata ?**

Answer

1. The language (regular expression) produced by a DFA is always unique.
2. However, the reverse, *i.e.*, a language produces a unique DFA is not true.
3. Hence for a given language there may be different DFAs.
4. By minimizing we can get a minimized DFA with minimum number of states and transitions which produces that particular language.
5. DFA determines how computers manipulate regular languages (expressions). DFA size determines space/time efficiency.
6. Hence from a DFA with minimized states need less time to manipulate a regular expression.

Que 1.37. **Define dead state, inaccessible state, equivalent state, distinguishable state and *k*-equivalence in case of finite automata.**

Answer

1. **Dead state :** A state q_i is called dead state if q_i is not a final state and for all the inputs to this state the transitions confined to that state. In mathematical notation, we can denote $q_i \notin F$ and $\delta(q_i, \Sigma) \rightarrow q_i$.

Scanned by TapScanner

2. **Inaccessible state :** The states which can never be reached from the initial state are called in accessible state.

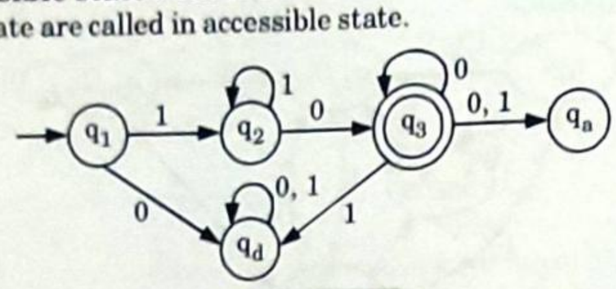


Fig. 1.38.1.

Here q_d is dead state and q_a is inaccessible state.

3. **Equivalent state :** Two states q_i and q_j of a finite automata M are called equivalent if $\delta(q_i, x)$ and $\delta(q_j, x)$ both produces final states or both of them produces non-final states for all $x \in \Sigma^*$. It is denoted by $q_i \equiv q_j$.
4. **Distinguishable state :** Two states q_i and q_j of a finite automata M are called distinguishable if for a minimum length string x, for $\delta(q_i, x)$ and $\delta(q_j, x)$ one produces final state and another produces non-final state or vice versa for all $x \in \Sigma^*$.
5. ***k*-equivalent :** Two states q_i and q_j of a finite automata M are called k-equivalent ($k \geq 0$) if $\delta(q_i, x)$ and $\delta(q_j, x)$ both produces final states or both of them produces non-final states for all $x \in \Sigma^*$ of length k or less.

Que 1.39. **Construct a minimum state automaton from the transition table given below.**

PS	NS	
	I/P = 0	*I/P* = 1
A	F	B
B	C	G
C	C	A
D	G	C
E	F	H
F	G	C
G	E	G
H	C	G

A is initial state and C is final state

Answer

1. In the finite automata the states are $\{A, B, C, D, E, F, G, H\}$.
   $S_0 : \{A, B, C, D, E, F, G, H\}$.

2. All of the states in S_0 are 0-equivalent.
3. In the finite automata there are two types of states final state and non-final states.
4. Hence divide the set of states into two parts Q_1 and Q_2.
   $Q_1 = \{C\}$, $Q_2 = \{A, B, D, E, F, G, H\}$,
   $S_1 : \{\{C\} \{A, B, D, E, F, G, H\}\}$.
5. States belong to same subset are 1-equivalent because they are in the same set for string length 1.
6. States belong to different subsets are 1-distinguishable.
7. The C is single state, hence it cannot be divided. Among the states $\{A, B, D, E, F, G, H\}$ for $\{B, D, F, H\}$ for input either 0 or 1, the next state belong to $\{C\}$ which is different subset. So,
   $S_2 : \{\{C\}, \{A, E, G\}, \{B, D, F, H\}\}$.
8. Similarly if we divide further,
   $S_3 : \{\{C\}, \{A, E\}, \{G\}, \{B, H\}, \{D, F\}\}$.
9. The subsets cannot be divided further. Hence these are the states of minimized DFA.
10. Let $\{C\} \rightarrow q_0$, $\{A, E\} \rightarrow q_1$, $\{G\} \rightarrow q_2$, $\{B, H\} \rightarrow q_3$, $\{D, F\} \rightarrow q_4$
11. Initial state was A, hence here initial states is $\{A, E\}$, *i.e.*, q_1.
12. Final state was C, hence here final state is $\{C\}$, *i.e.*, q_0.
13. The tabular representation of minimized DFA is

PS	NS	
	I/P = 0	*I/P* = 1
q_0	q_0	q_1
$\rightarrow q_1$	q_4	q_3
q_2	q_1	q_2
q_3	q_0	q_2
q_4	q_2	q_0

Que 1.40. **State and prove Myhill-Nerode theorem.**

Answer

The Myhill-Nerode Theorem : Let $L \subseteq \Sigma^*$ be a language. The following three statements are equivalent :

1. $L \subseteq \Sigma^*$ is regular, that is, $L = L(A)$ for some DFA A.
2. L is the union of some of the equivalent classes of a right-invariant equivalence relation on all strings taken from Σ^* of finite index.
3. Let an equivalence relation R_L on Σ^* be defined by :
   uR_Lv if and only if for all $w \in \Sigma^*$; $uw \in L \Leftrightarrow uv \in L$
   Then R_L is of finite index.

Scanned by TapScanner

The Myhill-Nerode theorem gives yet another characterization of regular languages. Its proof involves the construction of minimal automata, which is an important technique in its own right.

Proof :

1. Any equivalence relation R satisfying 2 could be shown is a refinement of R_L, that means, every equivalence class of R is entirely contained in some equivalence class of R_L. This way, the index of R_L cannot be greater than the index of R and therefore, it is finite.
2. Let us assume that uRv. Since R is right invariant, then for every $w \in \Sigma$, we have $uwRvw$, and therefore, uR_Lv. Thus, R is a refinement of uR_Lv.
3. We first prove that R_L is right invariant. Let us assume that uR_Lv. For any $y \in \Sigma^*$ we must show that uyR_Lvy, that means, for any $z \in \Sigma^*$, $uyz \in L \Leftrightarrow vyz \in L$. But this follows from the definition of R_L in 3, if we put $w = yz$.
4. We now construct a DFA A_{MV} accepting L. The basic technique is to use states as the equivalence classes of R_L (of which there is only a finite number). So,
   $A_{MN} = (Q, \Sigma, \delta, q_0, f) = (\{[w]_{RL} \mid w \in \Sigma^*\}, q_0, \delta_{MN}, F_{MN})$ we put
   i. $q_0 = [\varepsilon]_{RL}$
   ii. For all $w \in \Sigma^*$, $a \in \Sigma$ and $\delta_{MN}([w]_{RL}, a) = [wa]_{RL}$
   iii. $F_{MN} = \{[w]_{RL} \mid w \in L\}$
5. We first have to show that this is well-defined, especially that (*ii*) makes sense, that is, the definition of δ_{MN} is independent of the choice of representative w of the equivalence class $[w]_{RL}$. So we have to show that if wR_Lv, then $[w]_{RL} = \delta_{MN}([w]_{RL}, a) = \delta_{MN}[v]_{RL}, a) = [va]_{RL}$. But if wR_Lv, then waR_Lva because R_L is right-invariant, therefore $[wa]_{RL} = [va]_{RL}$. So δ_{MN} is well-defined.
6. Finally, we have to show that $L = L(A_{MN})$. This follows from $E_{MN}([\varepsilon]_{RL}, w) = [w]_{RL}$, where $E(q, x) = \varepsilon$ – closure $(\delta(q, x))$ and ε – closure (q) is set of all states reachable from q on input ε.

Que 1.41. **How Myhill-Nerode theorem can be applied in minimizing a DFA ?**

Answer

Step 1 : Build a two-dimensional matrix labelled by the states of the given DFA at the left and bottom side. The major diagonal and the upper triangular part will be put dash.

Step 2 : One of the three symbols, X, x or 0 will be put in the locations where is no dash.

a. Mark X at p, q in lower triangular part such that p is final state and Q is non-final state.
b. Make distinguished pair combination of the non-final states. If there are n number of non-final states there will be nC_2 number of distinguished pairs.
   Take a pair (p, q) and find (r, s), such that $r = \delta(p, a)$ and $s = \delta(q, a)$. If in the place of (r, s) there is X or x, in the place of (p, q) there will be x.

c. If (r, s) is neither X nor x, then (p, q) will be 0.
d. Repeat b and c for final states also.

Step 3 : The combination of states where there is 0, they will be the states of the minimized machine.

Que 1.42. **Minimize the automata given below**

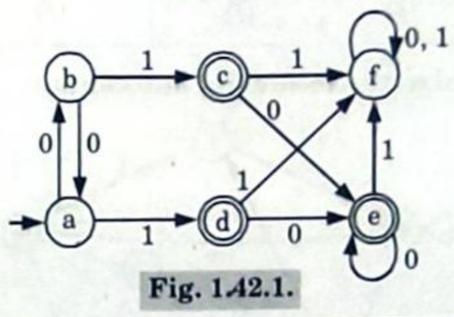


Fig. 1.42.1.

AKTU 2016-17, Marks 7.5

Answer

DFA minimization using equivalence theorem :

Transition table of DFA :

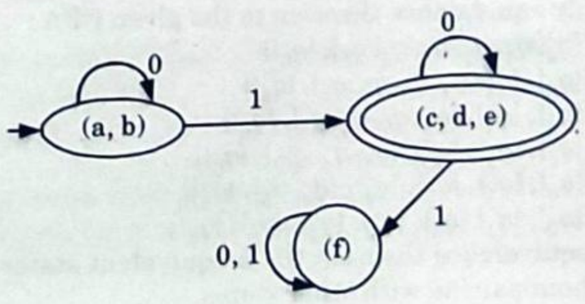
q	$\delta(q, 0)$	$\delta(q, 1)$
a	b	c
b	a	d
c	e	f
d	e	f
e	e	f
f	f	f

Let us apply the equivalence theorem to the given DFA :

$$P_0 = \{(c, d, e), (a, b, f)\}$$
$$P_1 = \{(c, d, e), (a, b), (f)\}$$
$$P_2 = \{(c, d, e), (a, b), (f)\}$$

Hence, $P_1 = P_2$.

There are three states in the reduced DFA. The reduced DFA is as follows :



Scanned by TapScanner

Transition table of DFA :

Q	δ(q, 0)	δ(q, 1)
(a, b)	(a, b)	(c, d, e)
(c, d, e)	(c, d, e)	(f)
(f)	(f)	(f)

Que 1.43. **Minimize the following automata :**

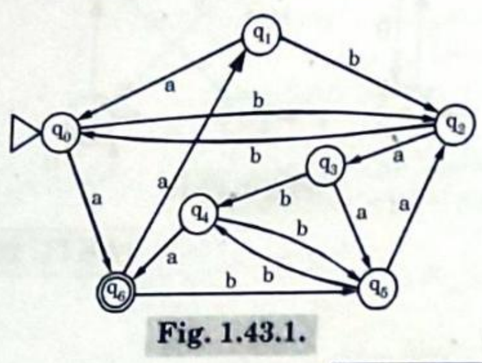


Fig. 1.43.1.

AKTU 2018-19, Marks 10

Answer

Transition table of DFA :

δ	a	b
→ q_0	q_6	q_2
q_1	q_0	q_2
q_2	q_3	q_0
q_3	q_5	q_4
q_4	q_6	q_5
q_5	q_2	q_4
*q_6	q_1	q_5

Let us apply equivalence theorem to the given DFA :

$P_0 = [\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{q_6\}]$

$P_1 = [\{q_0\}, \{q_1, q_2, q_3, q_4, q_5\}, \{q_6\}]$

$P_2 = [\{q_0\}, \{q_1\}, \{q_2, q_3, q_4, q_5\}, \{q_6\}]$

$P_3 = [\{q_0\}, \{q_1\}, \{q_2\}, \{q_3, q_4, q_5,\}, \{q_6\}]$

$P_4 = [\{q_0\}, \{q_1\}, \{q_2\}, \{q_3\}, \{q_4, q_5\}, \{q_6\}]$

$P_5 = [\{q_0\}, \{q_1\} \{q_2\}, \{q_3\}, \{q_4\}, \{q_5\}, \{q_6\}]$

By using equivalence theorem the 5-equivalent states contain all the states without pairing with other states.

So the given DFA is already minimized. So minimized DFA is

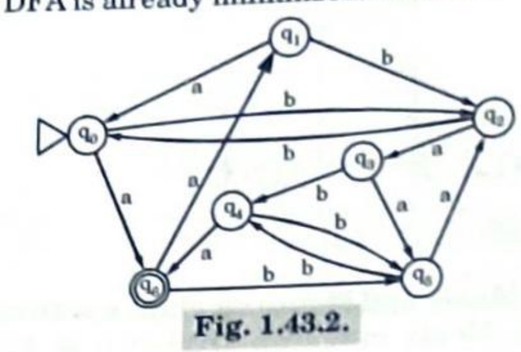


Fig. 1.43.2.

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. What do you understand by Deterministic Finite Automata (DFA) and how it is represented ?

Ans. Refer Q. 1.5.

Q. 2. Design a FA which accepts set of strings containing exactly four 1's in every string over Σ = {0, 1}.

Ans. Refer Q. 1.9.

Q. 3. What do you understand by Non-Deterministic Finite Automata (NFA/NDFA) ? How is it represented ?

Ans. Refer Q. 1.15.

Q. 4. Draw DFA of following over {0, 1} :

i. All strings with even number of 0's and even number of 1's.

ii. All strings of length at most 5.

Ans. Refer Q. 1.13.

Q. 5. Define NFA. What are various points of difference between NFA and DFA ?

Ans. Refer Q. 1.21.

Q. 6. Convert the following NFA {p, q, r, s}, {0, 1}, δ, p, {q, s} into DFA where δ is given by

Scanned by TapScanner

δ/Σ	0	1
$\rightarrow p$	q, s	q
$^*q$	r	q, r
r	s	p
$^*s$	ϕ	p

Ans. Refer Q. 1.25.

Q. 7. Describe Mealy and Moore machines with example. Convert the given Mealy machine as shown in Fig. 1 into Moore machine.

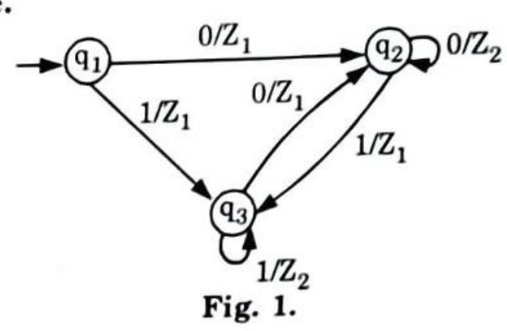


Fig. 1.

Ans. Refer Q. 1.36.

Q. 8. Minimize the automata given below

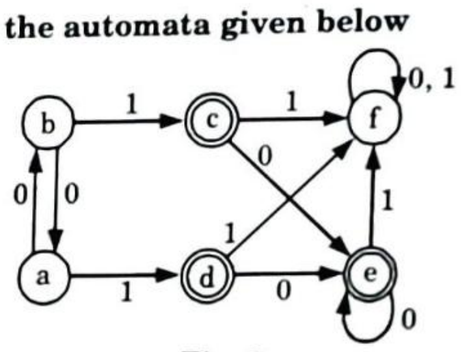


Fig. 2.

Ans. Refer Q. 1.42.

☺☺☺

2 UNIT

Regular Expressions and Languages

CONTENTS

Part-1 : Regular Expressions, Transition Graph ..... **2–2A to 2–4A**

Part-2 : Kleene's Theorem ..... **2–5A to 2–9A**

Part-3 : Finite Automata and Regular Expression : Arden's Theorem, Algebraic Method Using Arden's Theorem ..... **2–9A to 2–14A**

Part-4 : Regular and Non-Regular Languages : Closure Properties of Regular Languages ..... **2–14A to 2–17A**

Part-5 : Pigeonhole Principle, Pumping Lemma, Application of Pumping Lemma, Decidability: Decision Properties, Finite Automata and Regular Languages, Regular Language and Computers, Simulation of Transition Graph and Regular Language ..... **2–17A to 2–23A**

Scanned by TapScanner

PART-1

Regular Expressions, Transition Graph.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.1. **What do you mean by regular expression ?**

OR

Write the formal recursive definition of a regular expression.

Answer

1. Regular expression is a formula in a special language that is used for specifying simple classes of strings.
2. A string is a sequence of symbols; for the purpose of most text-based search techniques, a string is any sequence of alphanumeric characters (letters, numbers, spaces, tabs, and punctuation).

Formal recursive definition of regular expression :

Formally, a regular expression is an algebraic notation for characterizing a set of strings.

1. Any terminals, *i.e.*, the symbols belong to S are regular expression. Null string ($\wedge$ or ε) and null set (ϕ) are also regular expression.
2. If P and Q are two regular expressions then the union of the two regular expressions denoted by $P + Q$ is also a regular expression.
3. If P and Q are two regular expressions then their concatenation denoted by PQ is also a regular expression.
4. If P is a regular expression then the iteration (repetition or closure) denoted by P^* is also a regular expression.
5. If P is a regular expression then (P) is a regular expression.
6. The expressions got by repeated application of the rules from (1) to (5) over Σ are also regular expression.

Que 2.2. **Explain the basic operations and also write the operator precedence in regular expression.**

Answer

Basic operations in regular expression :

1. **Union :** If R_1 and R_2 are two regular expressions over Σ then $L(R_1 \cup R_2)$ is denoted by $L(R_1 + R_2)$.
   $L(R_1 \cup R_2)$ is a string from R_1 or a string from R_2.
   $L(R_1 + R_2) = L(R_1) \cup L(R_2)$.

2. **Concatenation :** If R_1 and R_2 are two regular expressions over Σ then $L(R_1 \cap R_2)$ is denoted by $L(R_1 R_2)$.
   $L(R_1 \cap R_2)$ is a string from R_1 followed by a string from R_2.
   $L(R_1 R_2) = L(R_1)\, L(R_2)$.
3. **Closure :** If R_1 and R_2 are two regular expressions over Σ then $L(R^*)$ is a string obtained by concatenating n elements for $n \geq 0$.

Operator precedence in regular expression :

Precedence	Operator	Description
Highest	(), (? :)	Parentheses and other grouping operators
↓	+, *, {m, n}	Repetition
	^ abc	Sequence
Lowest	|	Alternation

Que 2.3. **Discuss the identities for regular expression.**

Answer

1. Two regular expression P and Q are equivalent ($P = Q$) if P and Q represent the same set of strings.
2. The identities for regular expressions are useful for simplifying regular expressions.

I_1 : $\phi + R = R$

I_2 : $\phi R = R\phi = \phi$

I_3 : $\varepsilon R = R\varepsilon = R$

I_4 : $\varepsilon^* = \varepsilon$ and $\phi^* = \varepsilon$

I_5 : $R + R = R$

I_6 : $R^* R^* = R^*$

I_7 : $RR^* = R^*R$

I_8 : $(R^*)^* = R^*$

I_9 : $\varepsilon + RR^* = R^* = \varepsilon + R^*R$

I_{10} : $(PQ)^*P = P(QP)^*$

I_{11} : $(P + Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$

I_{12} : $(P + Q)R = PR + QR$ and $R(P + Q) = RP + RQ$

Que 2.4. **Simplify the regular expression $(r(r + s)^*)$.**

Answer

$\Rightarrow (r(r + s)^*)$

$\Rightarrow rr^* (sr^*)^*$ $\quad [(r + s)^* = r^* (sr^*)^*]$

$\Rightarrow r(\varepsilon + r)\, r^* (sr^*)^*$ $\quad [(\varepsilon + r)\, r^* = r^*]$

$\Rightarrow (r + rr)\, r^* (sr^*)^*$ $\quad [r + r\varepsilon$ and $\bullet$ distributes over $+]$

$\Rightarrow (r + rr)\, (r + s)^*$ $\quad [(r + s)^* = r^* (sr^*)^*]$

Que 2.5. **From the identities of regular expression prove that**

Scanned by TapScanner

i. $(1 + 100^*) + (1 + 100^*)(0 + 10^*)(0 + 10^*) = 10^* (0 + 10^*)^*$

ii. $\varepsilon + 1^* (011)^*[1^* (011)^*]^* = (1 + 011)^*$

Answer

i. $(1 + 100^*) + (1 + 100^*)(0 + 10^*)(0 + 10^*)^*$

L.H.S $= (1 + 100^*) [\varepsilon + (0 + 10^*)(0 + 10^*)^*]$ [According to $\varepsilon + RR^* = R^*$]

$= (1 + 100^*)(0 + 10^*)^*$

$= 1(\varepsilon + 00^*)(0 + 10^*)^*$

$= 10^*(0 + 10^*)^*$ = R.H.S.

ii. Take $1^* (011)^*$ as P then the LHS becomes $\varepsilon + PP^*$

$= P^*$

$= [1^*(011)^*]^* = (1 + 011)^*$ = R.H.S. [As $(P + Q)^* = (P^* Q^*)^*$].

Que 2.6. What are transition diagrams/transition graphs ?

Answer

A transition diagram/transition graphs is a graph which consists of series of states and there is a successful path beginning at some start state and ending at a final.

Thus, we can describe the transition graph as collection of following :

1. A finite set of states, one of them designated as the start state and some of which are designated as final state.
2. The starting state is represented by a circle having an arrow and final state is represented by two concentric circles. The diagrammatic representation is as follows :

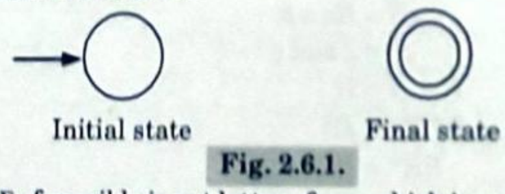


Fig. 2.6.1.

3. An alphabet Σ of possible input letters from which input string is formed.
4. A finite set of transitions (labeled edge) that shows how to go from one state to another state, based on reading specified substrings of input letters (possibly even the null string).
5. The null string may be represented by 'ε' or '∧'.
6. In transition diagram the initial state is represented as $\rightarrow (q_0)$ and final state as (q_f).

For example :

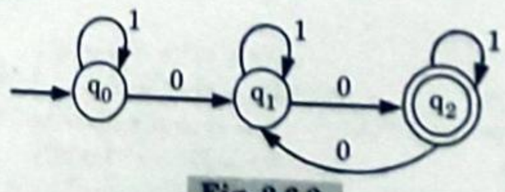


Fig. 2.6.2.

PART-2

Kleene's Theorem.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.7. State and prove Kleene's theorem with example.

AKTU 2016-17, Marks 10

Answer

Kleene's theorem states that any regular language is accepted by an FA and conversely that any language accepted by an FA is regular.

Theorem 1 (Part 1 of Kleene's theorem) : Any regular language is accepted by a finite automaton.

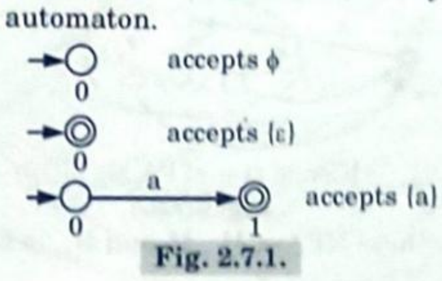


Fig. 2.7.1.

Proof :

Basis step : As shown in Fig. 2.7.1 the languages ϕ, {ε} and {a} for any symbol a in Σ are accepted by an FA.

Inductive step : We are going to show that for any languages L_1 and L_2 if they are accepted by FAs, then L_1L_2 , L_1L_2 and L_1^* are accepted by FAs. Since any regular language is obtained from {ε} and {a} for any symbol a in Σ by using union, concatenation and Kleene star operations, that together with the basis step would prove the theorem.

Suppose that L_1 and L_2 are accepted by FAs $M_1 = < Q_1, \Sigma, q_{1,0}, \delta_1, A_1 >$ and $M_2 = < Q_2, \Sigma, q_{2,0}, \delta_2, A_2 >$, respectively. We assume that $Q_1 \cap Q_2 = \phi$ without loss of generality since states can be renamed if necessary.

Then $L_1 \cup L_2$, $L_1 L_2$ and L_1^* are accepted by the FAs $M_u = < Q_u, \Sigma, q_{u,0}, \delta_u, A_u >$, $M_c = < Q_c, \Sigma, \delta_{c,0}, \delta_c, A_c >$ and $M_k = < Q_k, \Sigma, q_{k,0}, \delta_k, A_k >$, respectively, i.e.,

For $M_u = < Q_u, \Sigma, q_{u,0}, \delta_u, A_u >$:

$Q_u = Q_1 \cup Q_2 \cup \{q_{u,0}\}$, where $q_{u,0}$ is a state which is neither in Q_1 nor in Q_2.

$\delta_u = \delta_1 \cup \delta_2 \cup \{(q_{u,0}, \varepsilon, \{q_{1,0}, q_{2,0}\})\}$, that is $\delta_u(q_{u,0}, \varepsilon) = \{q_{1,0}, q_{2,0}\}$. i.e., $\delta_u(q_{u,0}, a) = \phi$ for all a in Σ.

$A_u = A_1 \cup A_2$

For $M_c = < Q_c, \Sigma, q_{c,0}, \delta_c, A_c >$:

Scanned by TapScanner

$Q_c = Q_1 \cup Q_2$

$q_{c,0} = q_{1,0}$

$\delta_c = \delta_1 \cup \delta_2 \cup \{(q, \varepsilon, \{q_{2,0}\}) \mid q \in A_1\}$

$A_c = A_2$

For $M_k = < Q_k, \Sigma, q_{k,0}, \delta_k, A_k >$:

$Q_k = Q_1 \cup \{q_{k,0}\}$, where $q_{k,0}$ is a state which is not in Q_1.

$\delta_k = \delta_1 \cup \{(q_{k,0}, \varepsilon, \{q_{1,0}\})\} \cup \{(q, \varepsilon, \{q_{k,0}\}) \mid q \in A_1\}$

$A_k = \{q_{k,0}\}$

These NFA-ε are illustrated in Fig. 2.7.2.

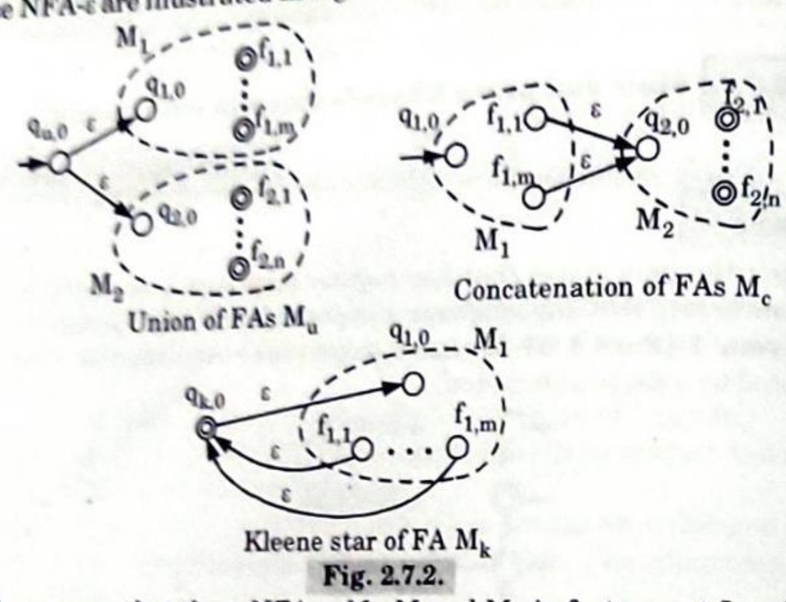
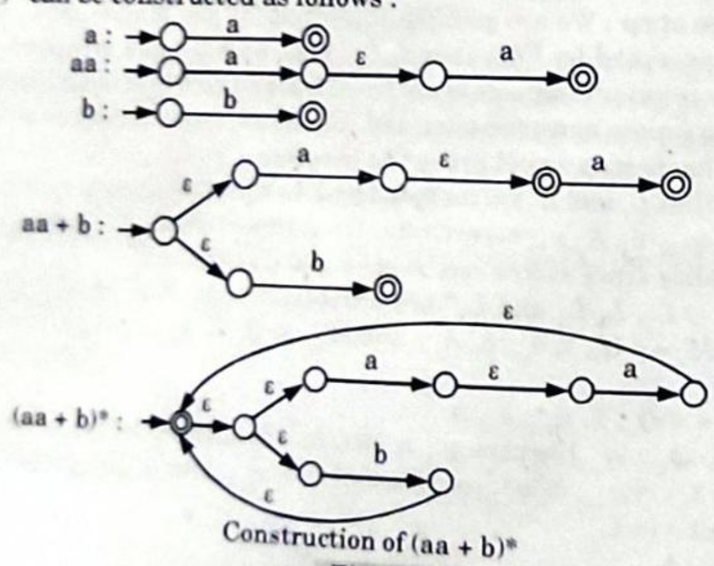


Fig. 2.7.2.

It can be proven, that these NFA-ε, M_u, M_c and M_k, in fact accept $L_1 \cup L_1$, L_2 and L_1^* respectively.

Example :

An NFA-ε that accepts the language represented by the regular expression $(aa + b)^*$ can be constructed as follows :



Construction of $(aa + b)^*$

Fig. 2.7.3.

Que 2.8. **For the given languages $L_1 = \phi$, $L_2 = \varepsilon$, and $L_3 = \{0, 1\}^*$. Compute $L_1 . L_2$ and $L_2 \cup L_3$.**

Answer

$L_1 = \phi, L_2 = \varepsilon, L_3 = \{0, 1\}^*$

Concatenation : $L_1 L_2 = \phi . \varepsilon = \{\varepsilon\}$

Union : $L_2 \cup L_3 = \varepsilon \cup \{0, 1\}^* = \{0, 1\}^*$

Que 2.9. **Let $L = \{0, 1\}\{0, 1\}^*$; construct NFA with ε-moves that accept L^2.**

Answer

The L includes strings here = {0, 1, 00, 01, 10, 11 .....}

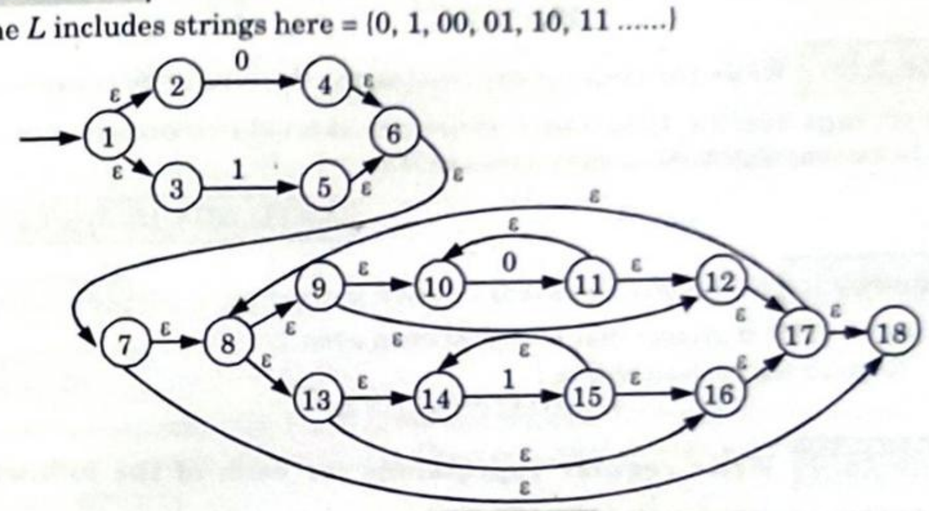


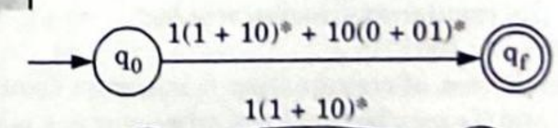
Fig. 2.9.1.

Que 2.10. **Draw finite automata recognizing the following language :**

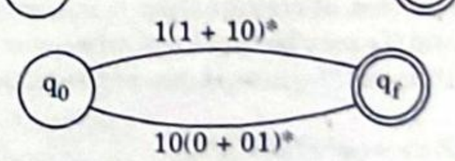
$1(1 + 10)^* + 10(0 + 01)^*$

Answer

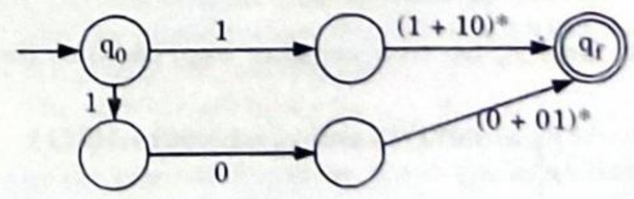
Step 1 :



Step 2 :



Step 3 :



Scanned by TapScanner

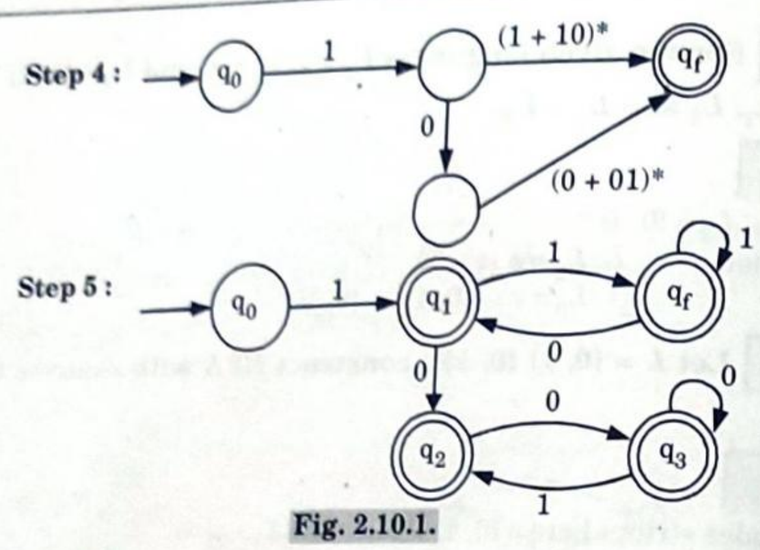


Fig. 2.10.1.

Que 2.11. **Write the regular expression for the language containing the strings over {0, 1} in which there are at least two occurrences of 1's between any two occurrences of 0's.**

AKTU 2014-15, Marks 10

Answer

If (11)* 0 (11)* it means that one 0 between even 1.

∴ Regular expression will be :

$$r = (0\,(11)^*\,0\,(11)^*\,0)^*$$

Que 2.12. **Write regular expressions for each of the following languages over the alphabet (0, 1) :**

i. **The set of all string in which every pair of adjacent zero's appear before any pair of adjacent one's.**

ii. **The set of all strings not containing 101 as substring.**

Answer

i. According to question, the pair (0011) may exists zero, one, two, ... times. Therefore the regular expression will be :

$$(0011)^*$$

ii. When we analyse the set of strings then it becomes clear that string may start with '0' and 0's may be repeated, wherever one is encountered then no single 0 must follow it so strings are like 0000010, 000001111001001.....

RE is = $(0^*1^*00)^*0^*1^*$

Que 2.13. **Let r_1 and r_2 be two regular expressions defined as follows :**

$$r_1 = (00^*1)^*1 \text{ and } r_2 = 1 + 0(0 + 10)^*11$$

Prove that $r_1 = r_2$.

Answer

$$r_1 = (00^*1)^*1$$

Set of string $= \{1, 011, 0011, 00011, 010101011, ...\}$

$$r_2 = 1 + 0\,(0 + 10)^*\,11$$

Set of string $= \{1, 011, 0011, 01011, 010101011, ...\}$

Since both regular expressions produce same set of string

So, $r_1 = r_2$

PART-3

Finite Automata and Regular Expression : Arden's Theorem, Algebraic Method Using Arden's Theorem.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.14. **State and prove Arden's theorem. Explain its application.**

Answer

Arden's theorem : Let P and Q are two regular expression over alphabet Σ. P does not contain null string ε. Then equation $R = Q + RP$ has a unique solution given by

$$R = Q.P^*$$

Proof : Let us consider the equation $R = Q + RP$ ...(2.14.1)

Put value of R in equation (2.14.1)

$$R = Q + (Q + RP)P \quad ...(2.14.2)$$

Put value of R in equation (2.14.2)

$$R = Q + QP + RP^2 \quad ...(2.14.3)$$

When we put the value of R again and again we get the following equation :

$R = Q + QP + QP^2 + QP^3 + ...$

$R = Q\,(\varepsilon + P + P^2 + P^3 + ...)$ [Since $\varepsilon + a + a^2 + ... = a^*$]

$R = Q.P^*$ [By definition of closure operation for regular expression]

Application :

1. Arden's theorem is used to determine the regular expression represented by transition diagram.
2. There are following assumptions corresponding to transition diagram :
   a. The transition system should not have any ε-move.
   b. It has only one starting state.
   c. The vertices are $q_1, q_2, q_3, ... q_n$.
   d. R_i is the regular expression representing the set of string accepted by the transition system, if q_i is the final state.

Scanned by TapScanner

e. w_{ij} denotes the regular expression representing the set of labels of edges from q_i to q_j. We can get the following set of equation in $q_1, q_2, \ldots q_n$.

$$q_1 = q_1 w_{11} + q_2 w_{21} + \ldots\ldots + q_n w_{n1} + \varepsilon$$
$$q_2 = q_1 w_{12} + q_2 w_{22} + \ldots\ldots + q_n w_{n2}$$
$$\ldots\ldots$$
$$\ldots\ldots$$
$$\ldots\ldots$$
$$q_n = q_1 w_{1n} + q_2 w_{2n} + \ldots\ldots + q_n w_{nn}$$

We solve these equations for q_i in terms of w_{ij}'s and it will be required regular expression. We add ε (null string) in the equation start with starting state q_i and we solve equation to find out q_i (final state) in terms of w_{ij}'s it is one of the regular expression for given deterministic finite automata.

Que 2.15. **Find the regular expression of given FA using Arden's theorem.**

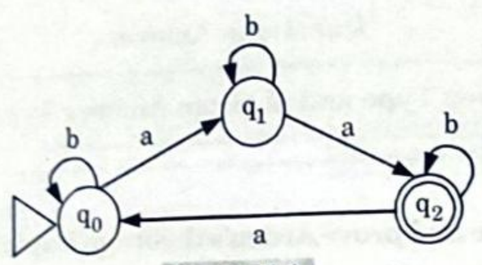


Fig. 2.15.1.

AKTU 2018-19, Marks 10

Answer

$$q_0 = q_0 b + q_2 a + \varepsilon \quad \ldots(2.15.1)$$
$$q_1 = q_1 b + q_0 a \quad \ldots(2.15.2)$$
$$q_2 = q_2 b + q_1 a \quad \ldots(2.15.3)$$

First solve equation (2.15.3)

According to theorem, $R = Q + RP$ gives $R = QP^*$

$$q_2 = q_1 a + q_2 b$$
$$Q = q_1 a \quad R = q_2 \quad P = b$$

∴ We will get $q_2 = q_1 ab^* \quad \ldots(2.15.4)$

Now solve equation (2.15.2)

$$q_1 = q_0 a + q_1 b$$
$$Q = q_0 a \quad R = q_1 \quad P = b$$

∴ We will get $q_1 = q_0 ab^* \quad \ldots(2.15.5)$

Put value of q_1 in equation (2.15.4),

$$q_2 = q_0 ab^* ab^*$$

Put value of q_2 in equation (2.15.1)

$$q_0 = q_0 b + q_0 ab^* ab^* a + \varepsilon = q_0 (b + ab^* ab^* a) + \varepsilon$$

According to Arden's theorem,

$$Q = \varepsilon \quad R = q_0 \quad P = (b + ab^* ab^* a)$$
$$R = QP^* = \varepsilon(b + ab^* ab^* a)^*$$
$$= (b + ab^* ab^* a)^* \quad [\because \ \varepsilon R = R]$$

Que 2.16. **Find the regular expression using Arden's theorem of FA given below.**

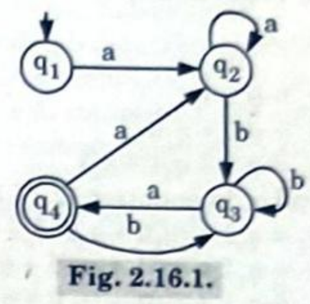


Fig. 2.16.1.

AKTU 2014-15, Marks 05

Answer

We directly apply the Arden's theorem as the graph does not contain any null (ε) moves.

We get the following equations :

$$q_1 = \varepsilon$$
$$q_2 = q_1 a + q_2 a + q_4 a$$
$$q_3 = q_2 b + q_3 b + q_4 b$$
$$q_4 = q_3 a$$

Therefore, $q_3 = q_2 b + q_3 b + q_3 ab = q_2 b + q_3 (b + ab)$

Applying Arden's theorem, $q_3 = q_2\, b\, (b + ab)^*$

Now, $q_2 = a + q_2\, a + q_3\, aa = a + q_2\, a + q_2\, b\, (b + ab)^*\, aa$

$$= a + q_2\, (a + b\, (b + ab)^*\, aa)$$

By Arden's theorem, $q_2 = a\, (a + b\, (b + ab)^*\, aa)^*$

$$q_4 = q_3\, a = a\, (a + b\, (b + ab)\, aa)^*\, b\, (b + ab)^*\, a$$

As q_4 is only final state, the RE corresponding to given FA is, $a(a + b\,(b + ab)^*\, aa)^*\, b\,(b + ab)^*\, a$.

Que 2.17. **State recursive definition of regular expression and construct a regular expression corresponding to the state transition diagram as shown in Fig. 2.17.1.**

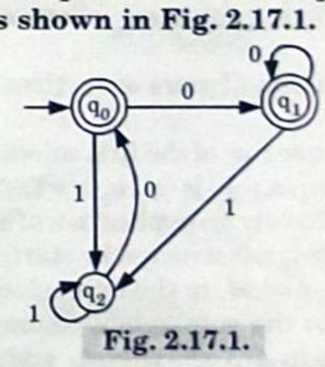


Fig. 2.17.1.

AKTU 2017-18, Marks 07

Scanned by TapScanner

Answer

Recursive definition of regular expression : Refer Q. 2.1, Page 2–2A, Unit-2.

Numerical :

$$q_0 = q_0 0 + \varepsilon \quad ...(2.17.1)$$
$$q_1 = q_0 0 + q_1 0 \quad ...(2.17.2)$$
$$q_2 = q_0 1 + q_1 1 + q_2 1 \quad ...(2.17.3)$$

Put the value of q_0 and q_1 in equation (2.17.3)

$$\begin{aligned} q_2 &= q_0 1 + (q_0 0 + q_1 0)\, 1 + q_2 1 \\ &= q_0 1 + q_0 01 + q_1 01 + q_2 1 \\ &= q_0 1 + q_0 01 + (q_0 0 + q_1 0)\, 01 + q_2 1 \\ &= q_0 1 + q_0 01 + q_0 001 + q_1 001 + q_2 1 \\ &= q_0 (1 + 01 + 001) + q_1 001 + q_2 1 \\ &= q_0 (1 + 01 + 001) + (q_0 0 + q_1 0) 001 + q_2 1 \\ &= q_0 (1 + 01 + 001) + q_0 0001 + q_1 0001 + q_2 1 \end{aligned}$$

If we solve further we get q_1 in every step. So, we cannot find the value of q_2 in term of q_0.

Hence, question cannot be solved.

Que 2.18. Explain the condition in which two machines M_1 and M_2 are said to be equivalent. Show that the following automatas are not equivalent.

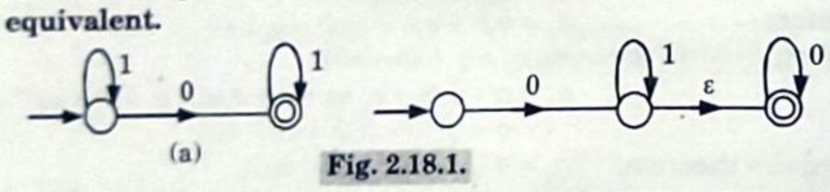


Fig. 2.18.1.

AKTU 2014-15, Marks 05

Answer

Comparison method :

1. Let M_1 and M_2 be two finite automata over Σ. We construct a comparison table consisting of $n + 1$ columns, where n is the number of input symbols.
2. The first column consists of pairs of vertices of the form (q, q'), where $q \in M_1$ and $q' \in M_2$.
3. If (q, q') appears in some row of the first column, then the corresponding entry in the a-column ($a \in \Sigma$) is (q_a, q_a'), where q_a and q'_a are reachable from q and q', respectively on application of a (*i.e.*, by a-paths).
4. The comparison table is constructed by starting with the pair of initial vertices q_{in}, q'_{in} of M_1 and M_2 in the first column.
5. The first elements in the subsequent columns are (q_a, q'_a), where q_a and q'_a are reachable by a-paths from q_{in} and q'_{in}.
6. We repeat the construction by considering the pairs in the second and subsequent columns which are not in the first column.

7. The row-wise construction is repeated. There are two cases :

Case 1 : If we reach a pair (q, q'_a) such that q is a final state of M_1, and q'_a is a non-final state of M_2 or vice versa, we terminate the construction and conclude that M_1 and M_2 are not equivalent.

Case 2 : Here the construction is terminated when no new element appears in the second and subsequent columns which are not in the first column (*i.e.*, when all the elements in the second and subsequent columns appear in the first column). In this case we conclude that M_1 and M_2 are equivalent.

Numerical :

Let M_1 and M_2 be the following automatas :

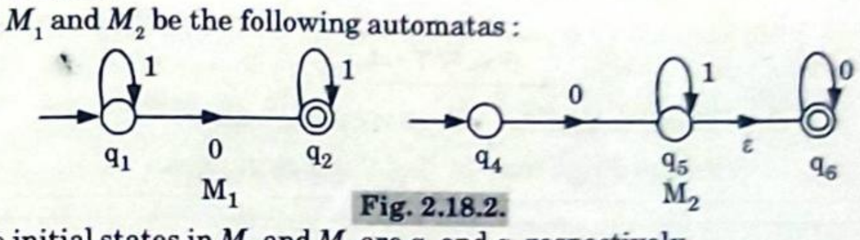


Fig. 2.18.2.

The initial states in M_1 and M_2 are q_1 and q_4 respectively.

Hence, the first element of the first column in the comparison table must be (q_1, q_4).

Table 2.18.1.

(q, q')	(q_0, q'_0)	(q_1, q'_1)
(q_1, q_4)	(q_2, q_5)	(q_2, q_5)
(q_2, q_5)	–	(q_2, q_5)

As q_2 is final state in M_1 and q_5 is non-final state in M_2, so M_1 and M_2 are not equivalent.

Que 2.19. From the given NFAs, decide whether the two accept the same language.

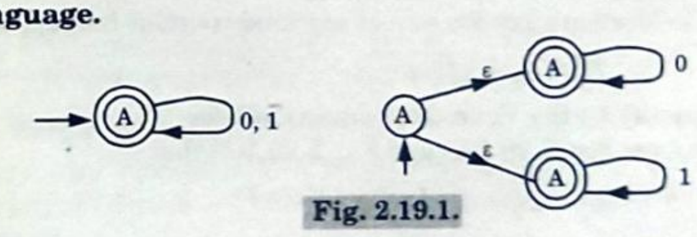


Fig. 2.19.1.

Answer

1. Let

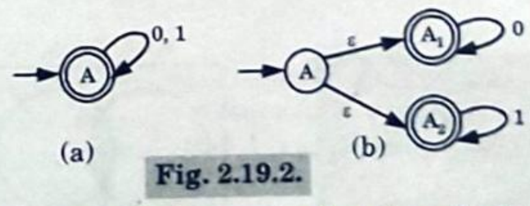


Fig. 2.19.2.

2. In Fig. 2.19.2(a) the NFA accept all string which is combination of alphabets

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

Scanned by TapScanner

3. In Fig. 2.19.2(b) the NFA has ε-move and we know that ε-transition is a null transition. So we can consider state A, A_1 and A_2 as same state i.e., A and we get

Fig. 2.19.3.

4. So the given NFAs (Fig. 2.19.1) accept same language.

PART-4

Regular and Non-Regular Languages :
Closure Properties of Regular Languages.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.20. **Prove that the language L_1 and L_2 are closed under intersection and complementation if they are regular.**

Answer

Intersection :

If L_1 and L_2 are regular languages, then $L_1 \cap L_2$ is also regular language. In other words the set of the regular language is closed under intersection.

Proof : By De-Morgan's law for sets of any kind (regular languages or not) :

$$L_1 \cap L_2 = \overline{(\bar{L}_1 + \bar{L}_2)}$$

This can be justify by the Venn diagrams as follows :

So, it is clear from Fig. 2.20.1(*a*) and Fig. 2.20.1(*b*) that

$$L_1 \cap L_2 = \overline{(\bar{L}_1 + \bar{L}_2)}$$

Because L_1 and L_2 are regular so $\bar{L}_1$ and $\bar{L}_2$ are regular. So is $\bar{L}_1 + \bar{L}_2$. And because $\bar{L}_1 + \bar{L}_2$ is regular then so is $\overline{(\bar{L}_1 + \bar{L}_2)}$ which means $\bar{L}_1 \cap \bar{L}_2$ is regular.

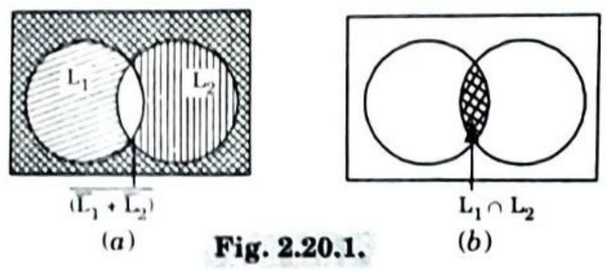


Fig. 2.20.1.

Complementation :

If L is regular then complement of L that is $\bar{L}$ is also regular. In other words, the set of regular languages is closed under complementation.

Proof :

1. To show closure under complementation, let $M = (Q, \Sigma, \delta, q_0, F)$ be DFA that accepts L, then DFA accepts $\bar{L}$.

$$\bar{M} = (Q, \Sigma, \delta, q_0, Q - F)$$

2. In DFA M accepts the language L in which some of the states of FA are final states and some are not.
3. In DFA $\bar{M}$ status of each state is reversed, that is if it is final state, make it a non-final state, and if it is non-final state, make it a final state.
4. If an input string formally ended in a non-final state, it now ends in a final state and vice versa.
5. This new machine accepts all input strings that were not accepted by the original FA (all the strings in $\bar{L}$) and rejects all the input strings that the FA used to accept (the strings in L). Therefore, machine accepts exactly the language $\bar{L}$. So, $\bar{L}$ is also regular language.

Que 2.21. **State whether following statement is true or false. Justify your answer :**

i. **If L and M are non-regular languages then intersection of L and M is also non-regular.**

ii. **If L and M are regular languages then $L - M$ is also regular language.**

Answer

i. True

Proof : Let L and M are two non-regular language.

The intersection of L and M is represented by $L \cap M$.

By De-Morgan's law $L \cap M = \overline{\bar{L} \cup \bar{M}}$

We know that complement of a non-regular language is non-regular and union of two non-regular languages is non-regular. Therefore $\overline{\bar{L} \cup \bar{M}}$ is non-regular.

Here $L \cap M$ is non-regular language.

ii. True

Proof : $L_1 - L_2 = L_1 \cap \bar{L}_2$

[Replace L_1 by L and L_2 by M]

We know that complement of a regular language is regular and we also know that the intersection of two regular languages is regular.

Hence, $L_1 \cap \bar{L}_2$ will be regular.

Therefore $L_1 - L_2$ is regular.

Scanned by TapScanner

Que 2.22. **Prove that the complement, homomorphism and inverse homomorphism, closure of a regular language is regular.**

AKTU 2016-17, Marks 10

Answer

1. **Regular languages are closed under complementation :** Refer Q. 2.20, Page 2–14A, Unit-2.
2. **Regular languages are closed under homomorphism, *i.e.*, if L is a regular language and h is a homomorphism, then $h(L)$ is also regular :**

   Proof :

   We will use the representation of regular languages in terms of regular expressions to prove this :

   Let E be the regular expression for L.

   Apply h to each symbol in E.

   Language of resulting RE is $h(L)$.

   Example :

   Let $h(0) = ab$; $h(1) = \varepsilon$.

   Let L be the language of regular expression $01^* + 10^*$.

   Then $h(L)$ is the language of regular expression $ab\varepsilon^* + \varepsilon(ab)^*$.

   $ab\varepsilon^* + \varepsilon(ab)^*$ can be simplified.

   $\varepsilon^* = \varepsilon$, so $ab\varepsilon^* = ab\varepsilon$.

   ε is the identity under concatenation *i.e.*, $\varepsilon E = E\varepsilon = E$ for any RE E.

   Thus, $ab\varepsilon^* + \varepsilon(ab)^* = ab\varepsilon + \varepsilon(ab)^* = ab + (ab)^*$.

   Finally, $L(ab)$ is contained in $L((ab)^*)$, so a RE for $h(L)$ is $(ab)^*$.
3. **Regular languages are closed under inverse homomorphism, *i.e.*, if L is regular and h is a homomorphism then $h^{-1}(L)$ is regular :**

   Proof :

   a. We will use the representation of regular languages in terms of DFA to argue this.

      Start with a DFA A for L.

   b. For a given a DFA M recognizing L, construct a DFA M' that accepts $h^{-1}(L)$.

      Construct a DFA B for $h^{-1}(L)$ with the same set of states, the same start state, the same final states and input alphabet (the symbols to which homomorphism h applies).

   c. The transitions for B are computed by applying h to an input symbol a and seeing where A would go on sequence of input symbols $h(a)$. Formally, $\delta_B(q, a) = \delta_A(q, h(a))$.

      Example :

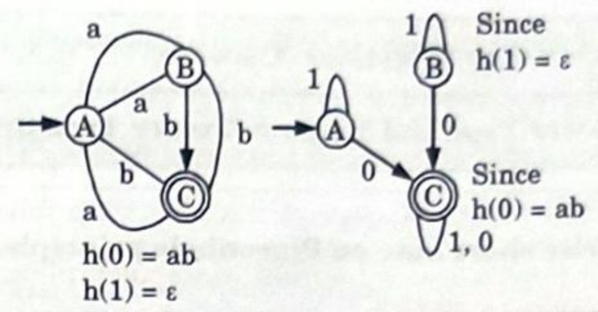


Fig. 2.22.1. Inverse homomorphism construction.

Que 2.23. **Discuss closure properties *i.e.*, concatenation, union, intersection, complement of regular languages.**

AKTU 2017-18, Marks 07

Answer

Union : If $L1$ and if $L2$ are two regular languages, their union $L1 \cup L2$ will also be regular.

For example :

Let $L1 = \{a^n \mid n \geq 0\}$ and $L2 = \{b^n \mid n \geq 0\}$

$L3 = L1 \cup L2 = \{a^n \cup b^n \mid n \geq 0\}$ is also regular.

Concatenation : If $L1$ and if $L2$ are two regular languages, their concatenation $L1.L2$ will also be regular.

For example :

Let $L1 = \{a^n \mid n \geq 0\}$ and $L2 = \{b^n \mid n \geq 0\}$

$L3 = L1.L2 = \{a^m . b^n \mid m \geq 0 \text{ and } n \geq 0\}$ is also regular.

Intersection : If $L1$ and if $L2$ are two regular languages, their intersection $L1 \cap L2$ will also be regular.

For example :

Let $L1 = \{a^m b^n \mid n \geq 0 \text{ and } m \geq 0\}$ and $L2 = \{a^m b^n \cup b^n a^m \mid n \geq 0 \text{ and } m \geq 0\}$

$L3 = L1 \cap L2 = \{a^m b^n \mid n \geq 0 \text{ and } m \geq 0\}$ is also regular.

Complement : If $L(G)$ is regular language, its complement $L'(G)$ will also be regular. Complement of a language can be found by subtracting strings which are in $L(G)$ from all possible strings. **For example :**

$$L(G) = \{a^n \mid n > 3\}$$
$$L'(G) = \{a^n \mid n <= 3\}$$

PART-5

Pigeonhole Principle, Pumping Lemma, Application of Pumping Lemma, Decidability : Decision Properties, Finite Automata and Regular Languages, Regular Language and Computers, Simulation of Transition Graph and Regular Language.

Scanned by TapScanner

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.24. **Write short note on Pigeonhole principle.**

Answer

Pigeonhole principle :

1. It states that "If n pigeons are assigned to m pigeonholes then at least one pigeonhole contains two or more pigeons $(m < n)$".
2. The pigeonhole principle is sometime useful in counting methods.

Proof :

1. Let m pigeonholes be numbered with the numbers 1 through m.
2. Beginning with the pigeon 1, each pigeon is assigned in order to the pigeonholes with the same number.
3. Since $m < n$ *i.e.*, the number of pigeonhole is less than the number of pigeons, $n - m$ pigeons are left without having assigned a pigeonhole.
4. Thus, at least one pigeonhole will be assigned to a more than one pigeon.
5. We note that the pigeonhole principle tells us nothing about how to locate the pigeonhole that contains two or more pigeons.
6. It only asserts the existence of a pigeon hole containing two or more pigeons.
7. To apply the principle one has to decide which objects will play the role of pigeon and which objects will play the role of pigeonholes.

Que 2.25. **State and prove the pumping lemma for regular language.**

Answer

1. Pumping Lemma is a powerful tool for proving certain language non-regular.
2. It is also useful in the development of algorithm to answer certain questions concerning, finite automata, such as whether the language accepted by a given FA is finite or infinite.

Statement : Let L be a regular language. Then there exist a constant n (which depends on L) such that for every string w in L, such that $|w| \geq n$, we can break w into three substrings, $w = xyz$, such that :

1. $y \neq \varepsilon$
2. $|xy| \leq n$
3. For all $i \geq 0$, the string xy^iz is also in L.

That is we always find a non-empty string y not too far from the beginning of w that can be "pumped"; that is repeating y any number of times keeps the resulting string in the language.

Proof :

1. Suppose L is regular. Then $L = L(M)$ for some DFA, M. Suppose M has n states.
2. Now, consider any string w of length n or more. Let $w = a_1, a_2, \ldots a_m$, where $m \geq n$ and each a_i is an input symbol, for $i = 0, 1, 2, \ldots, m$ define state p_i, to be $\hat{\delta}(q_0, a_1a_2 \ldots a_i)$, where δ is the transition function of M, and q_0 is the start state of M. That is p_i is the state, M is in after reading the first i symbols of w.
3. By the Pigeonhole principle, it is not possible for the $n + 1$ different p_i's for $i = 0, 1, \ldots n$ to be distinct, since there are only n different states.
4. Thus we can find two different integers i and j, with $0 \leq i < j \leq n$, such that $p_i = p_j$. Now we can break $w = xyz$ as follows :
   1. $x = a_1a_2 \ldots a_i$
   2. $y = a_{i+1}a_{i+2} \ldots a_j$
   3. $z = a_{j+1}a_{j+2} \ldots a_m$.

   That is x takes us to p_i once; y takes us from p_i back to p_i (since p_i is also p_j), and z is balance of w.
5. The relationship among the strings and states are discussed in Fig. 2.25.1.

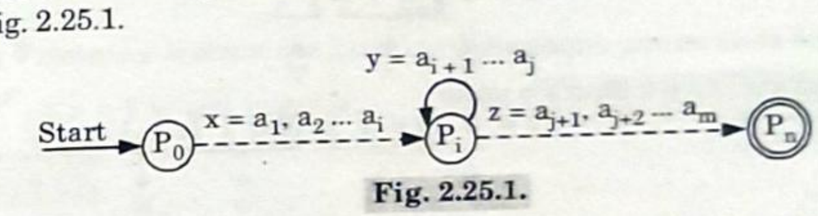


Fig. 2.25.1.

   x may be empty, in the case that $i = 0$. Also, z may be empty if $j = n = m$. However y cannot be empty, since i is strictly less than j.
6. Now consider what happens if the automaton M receives xy^iz for any $i \geq 0$. If $i = 0$, then automaton M goes from the start state q_0 to P_i on input x. Since p_i is also p_j, it must be that M goes from p_i to the accepting state in Fig. 2.20.2 on input z. Thus, in accepts xz.
7. If $i > 0$, then M goes from p_0 to p_i on input string x, circles from p_i to p_i, i times on input y_i, and then goes to the accepting state on input z. Thus, for any $i \geq 0$, xy^iz is also accepted by M; that is, xy^iz is in L.

Que 2.26. **Explain the application of pumping lemma.**

Answer

Application of the pumping lemma are :

The pumping lemma is extremely useful in proving that certain sets are non-regular. The general steps in its application are :

Step 1 : Assume L is a regular. Let n be the number of states in corresponding FA.

Step 2 : Choose a string w such that $|w| \geq n$. Use pumping lemma to write $w = xyz$, with $|yz| \leq n$ and $|y| > n$.

Step 3 : Find a suitable integer i such that $xy^iz \notin L$. This contradicts our assumption. Hence, L is not regular.

Scanned by TapScanner

Que 2.27. **State pumping lemma for regular languages. Use pumping lemma to prove that the language L, defined as follows, is not regular.**

$L = \{0^m 1^n \mid m$ and n are positive integers and $m \neq n\}$

Answer

Pumping lemma for regular language : Refer Q. 2.25, Page 2–18A, Unit-2.

Proof :

Let $L = \{0^m 1^n \mid m \neq n\}$ be a language. Assume that L is regular language. Now consider following case :

Case 1 :

Assume $z = 0001111 \in L$

By pumping lemma if $z = uvw$ then if we pump some string to make $z = uv^iw$ then if $z \in L$ then such a language is called regular.

Let

$$z = 0\ \underbrace{0\ 0\ 1\ 1}_{v}\ \underbrace{1\ 1}_{w}$$

$$\downarrow u$$

if $z = uv^iw$ and if $i = 2$ then $z = uvvw$

$$z = 0\ \underbrace{(0\ 0\ 1\ 1)}_{v}\ \underbrace{(0\ 0\ 1\ 1)}_{v}\ \underbrace{1\ 1}_{w}$$

$$\downarrow u$$

$\therefore$

$$z = 0^3 1^2 0^2 1^4 \notin 0^m 1^n \notin L$$

Case 2 :

Assume $L \in z$ such that

$$z = 0\ \underbrace{0\ 0}_{v}\ \underbrace{0\ 1\ 1\ 1\ 1\ 1\ 1}_{w}$$

$$\downarrow u$$

By pumping lemma, $z = uv^iw \in L$ for a regular language. If $i = 2$ then,

$$z = uvvw$$

$$z = 0\ \underbrace{0\ 0}_{v}\ \underbrace{0\ 0}_{v}\ \underbrace{0\ 1\ 1\ 1\ 1\ 1}_{w}$$

$$\downarrow u$$

$$z = 0^6 1^5 \in 0^m 1^n \text{ because } m = n.$$

From these cases, we get $z \notin L$. Thus our assumption of L being regular is wrong.

Hence given language $\{L = 0^m 1^n \mid m \neq n\}$ is not regular.

Que 2.28. **Prove that the language $L = \{0^n \mid n$ is prime$\}$ is not regular.**

OR

State Pumping Lemma for regular sets. Show that the set $L = \{a^p \mid p$ is a prime$\}$ is not regular. **AKTU 2015-16, 2017-18; Marks 10**

Answer

Pumping lemma : Refer Q. 2.25, Page 2–18A, Unit-2.

Numerical :

Step 1 : We suppose L is regular. Let q be the number of states in the finite automaton accepting L.

Step 2 : Let p be a prime number greater than p. Let $w = a^q$. By pumping lemma, w can be written as $w = xyz$, with $|xy| \leq q$ and $|y| > 0$. x, y, z are simply strings of a's. So, $y = a^m$ for some $m \geq 1$ (and $\leq q$).

Step 3 : Let $i = p + 1$. Then $|xy^iz| = |xyz| + |y^{i-1}| = p + (i-1)m = p + pm$. By pumping lemma, $xy^iz \in L$. But $|xy^iz| = p + pm = p(1+m)$, and $p(1+m)$ is not a prime. So, $xy^iz \in L$. This is a contradiction. Thus, L is not regular.

Que 2.29. **Prove that following are not regular languages :**

i. $\{0^n \mid n$ is perfect square$\}$.

ii. The set of strings of form $0^i 1^i$ such that the greatest common divisor of i and j is 1.

OR

Using Pumping lemma for regular languages prove that language $L = 0^{n^2}, n >= 1$ is not regular. **AKTU 2018-19, Marks 10**

Answer

i. $\{0^n \mid n$ is perfect square$\}$.

1. Let us suppose L be a regular, let p be a constant provided by pumping lemma. Let w be the string of 0^{p^2}.
2. This string is in L and its length is at least p. So we can write $|xy| \leq p$ and $y \neq \varepsilon$ and each xy^kz also in L.
3. Since $|xy| \leq p$, y consists of no more than p 0's and $y \neq \varepsilon$, so there is at least one '0' in y.
4. Now choose k to be 2 and the resulting string is $xyyz$.
5. We have to prove that the length of $xyyz$ is not perfect square. In fact we have to show that it lies most strictly in between two consecutive perfect squares, namely p^2 and $(p-1)^2$.
6. Let 'q' stand for length of y. Then the length of $xyyz$ is $p^2 + q$. So $p^2 < p^2 + q < (p+1)^2$.
7. The first inequality holds since $q > 0$. For second inequality, we compute $(p+1)^2 = p^2 + 2p + 1 > p^2 + q$. Since $q < p$, therefore $xyyz$ is not in L.
8. This contradicts the pumping lemma, so our assumption, that 'L' is regular is incorrect.

ii. Let $L = 0^i 1^j$ such that gcd $(i, j) = 1$.

We assume that this is a regular language. Now consider three cases

Case 1 : Let $z = \overbrace{0 \ldots 0}^{i}\ \overbrace{1 \ldots 1}^{j}$ be a string $\in L$.

According to pumping lemma if $z = uvw$ is a string $\in L$, then $z = uv^iw$, $\in L$.

Scanned by TapScanner

That means even if we pump some substring the string $z \in L$.
We map $z = uvw$ as this is a case in which v consists of substring 0^n1^n.
By pumping lemma $z = uv^iw$ if $i = 2$, $z = uvvw$

Then we get $\quad z = 0^{i-m}(01)^m 1^{j-m} \notin L = \{0^i\ 1^j\}$

Case 2 : Let, $\quad z = \overbrace{\underbrace{0\ldots0}_{uv}}^{i}\ \overbrace{\underbrace{1\ldots1}_{w}}^{j} \in L$

By pumping lemma $z = uv^iw$. If $i = 2$ then,

$$z = \overbrace{\underbrace{0\ldots0}_{u}\ldots\underbrace{0}_{v}}^{i}\ \overbrace{\underbrace{1\ldots1}_{v}\ldots\underbrace{1}_{w}}^{j}$$

$$z = uvvw$$

i.e., $\quad z = 0^{i+m}\,1^j$

But, gcd $(i+m, j) \neq 1$.

Hence, $\quad z \notin [L = \{0^i\ 1^j\}]$

Case 3 : Let,

$$z = \overbrace{\underbrace{0\ldots0}_{u}}^{i}\ \overbrace{\underbrace{1\ldots1}_{vw}}^{j}$$

By pumping lemma, $z = uv^iw$. If $i = 2$ then,

$$z = uvvw$$

i.e., $\quad z = 0^i\,1^{j+m}$

But gcd $(i, j+m) \neq 1$.

Hence, $\quad z \notin [L = \{0^i\ 1^j\}]$

From these 3 cases, we can prove that $L = 0^i1^j \mid \gcd(i, j) = 1$ is not a regular language.

Que 2.30. Prove that the language $L = \{0^n \mid n$ is perfect cube$\}$ is not regular. **AKTU 2014-15, Marks 05**

Answer

1. Let us assume L is regular, let p be a constant provided by the pumping lemma.
2. Let w be the string 0^{p^3}. This string is in L, and is of length at least p. So w can be written as xyz with $|xy| \leq p$ and y not ε.
3. Pumping lemma states that if $xyz \in L$ then xy^kz also in L. y contains at least one 0 in y since y is not ε.
4. Let us assume for $k = 2$, the resulting string is $xyyz$.
5. We have to show that the length of $xyyz$ is not a perfect cube. Let q be the length of y, then the length of $xyyz$ is $p^3 + q$.
6. So it suffices to show that $p^3 < p^3 + q < (p + 1)^3$. The first inequality holds since $q > 0$ (since y was not ε).
7. For second inequality, we compute $(p + 1)^3 = p^3 + 3p^2 + 3p + 1$, and this is greater than $p^3 + q$ since $q < p$ (since $|xy| \leq p$).

8. The claim is established, so $xyyz$ is not in L.
9. This contradicts the pumping lemma, so our original assumption, that L was regular is not incorrect.

Que 2.31. Explain decision problem and decidability properties in regular language.

Answer

1. A decision problem is a restricted type of an algorithmic problem where for each input there are only two possible outputs.
2. A decision problem is a function that associates with each input instance of the problem a truth value true or false.
3. A decision algorithm is an algorithm that computes the correct truth value for each input instance of a decision problem. The algorithm has to terminate on all inputs.
4. A decision problem is decidable if there exists a decision algorithm for it. Otherwise it is undecidable.

Decidability properties of regular languages :

1. **DFA membership :** Let $M = (Q, \Sigma, \delta, q_0, F)$ denote a deterministic finite automaton and x is a sequence of finite-domain variables $(x1, x2, \ldots, xn)$ with respective domains $D1, D2, \ldots, Dn \subseteq \Sigma$. Under a regular language membership constraint regular (x, M), any sequence of values taken by the variables of x must belong to the regular language recognized by M.
2. **DFA emptiness :** DFA is said to be empty if there is no path from initial state to final state.
3. **DFA equivalence :** Let $M_1 = (Q, \Sigma, \delta, q_0, F)$ and $M_2 = (Q', \Sigma', \delta', q_0', F')$ are two DFA. M_1 and M_2 are said to be equivalent if and only if $L(M_1) = L(M_2)$.
4. **DFA finiteness :** Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. String accept by M is said to be finite if and only if there is no cycle between any two state q_1 and q_2.

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. State and prove Kleene's theorem with example.

Ans. Refer Q. 2.7.

Q. 2. State and prove Arden's theorem. Explain its application.

Ans. Refer Q. 2.14.

Scanned by TapScanner

Q. 3. Find the regular expression of given FA using Arden's theorem.

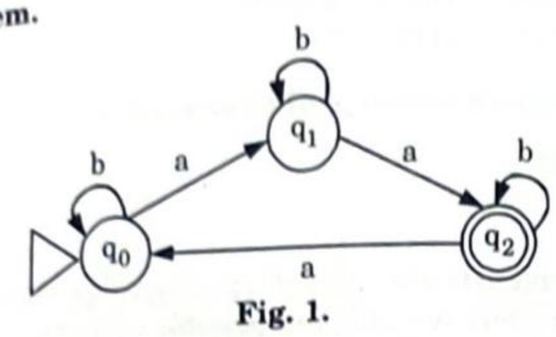
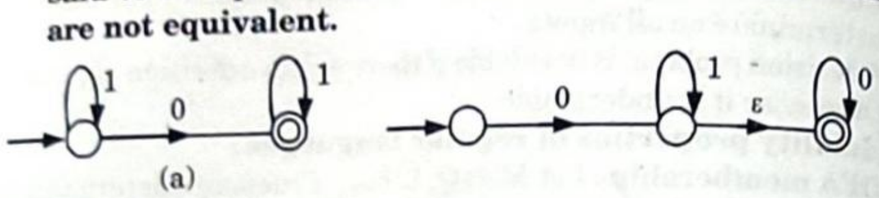


Fig. 1.

Ans. Refer Q. 2.15.

Q. 4. Explain the condition in which two machines M_1 and M_2 are said to be equivalent. Show that the following automatas are not equivalent.



(a)

Ans. Refer Q. 2.18.

Q. 5. Prove that the complement, homomorphism and inverse homomorphism, closure of a regular language is regular.

Ans. Refer Q. 2.22.

Q. 6. Discuss closure properties *i.e.*, concatenation, union, intersection, complement of regular languages.

Ans. Refer Q. 2.23.

Q. 7. Prove that the language $L = \{0^n \mid n$ is perfect cube$\}$ is not regular.

Ans. Refer Q. 2.30.

Q. 8. Prove that the language $L = \{0^n \mid n$ is prime$\}$ is not regular.

Ans. Refer Q. 2.28.

☺☺☺

3 UNIT

Regular and Non-Regular Grammars

CONTENTS

Part-1 :	Context Free Grammar (CFG) : Definition, Derivations, Languages	**3–2A to 3–4A**
Part-2 :	Derivation Trees and Ambiguity	**3–4A to 3–11A**
Part-3 :	Regular Grammars : Right Linear and Left Linear Grammars	**3–11A to 3–15A**
Part-4 :	Conversion of FA into CFG and Regular Grammar into FA	**3–15A to 3–16A**
Part-5 :	Simplification CFG, Normal Forms : Chomsky Normal Form (CNF), Greibach Normal Form (GNF), Chomsky Hierarchy	**3–17A to 3–26A**
Part-6 :	Programming Problem Based on the Properties of CFG	**3–26A to 3–27A**

Scanned by TapScanner

PART-1

Context Free Grammar (CFG) : Definition, Derivations, Language

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.1. What is context free grammar (CFG) ? Explain.

Answer

1. A CFG describes a language by recursive rules called productions.
2. A CFG can be described as a combination of four tuple and represented by $G(V, T, P, S)$.
   where,
   V : set of variables or non-terminal represented by $A, B, \ldots, Y, Z$.
   T : set of terminals represented by $a, b, c, \ldots\ldots x, y, z, +, -, *, (,)$ etc.
   S : starting symbol.
   P : set of productions.
3. The production used in CFG must be in the form of $A \to \alpha$, where A is a variable and α is string of symbols $(V \cup T)^*$.
4. The example of CGF is :

$$G = (V, T, P, S) \text{ where } V = \{E\},\ T = \{+, *, (,), id\}$$
$$S = \{E\} \text{ and production } P \text{ is given as}$$
$$P : \{E \to E + E, E \to E * E, E \to (E), E \to id\}$$

Que 3.2. Give an example of CFG for arithmetic expressions.

Answer

1. An example grammar that generates strings representing arithmetic expressions with the four operators +, –, *, /, and numbers as operands is :
   a. <expression> → number b. <expression> → (<expression>)
   c. <expression> → <expression> + <expression>
   d. <expression> → <expression> – <expression>
   e. <expression> → <expression> * <expression>
   f. <expression> → <expression> / <expression>
2. The only non-terminal symbol in this grammar is <expression>, which is also the starting symbol. The terminal symbols are {+, –, *, /, (,), number}.

3. The first rule (or production) states that an <expression> can be rewritten as (or replaced by) a number. In other words, a number is a valid expression.
4. The second rule says that an <expression> enclosed in parentheses is also an <expression>. This rule defines an expression in terms of expressions, an example of the use of recursion in the definition of CFG.
5. The remaining rules say that the sum, difference, product, or division of two <expression> is also an expression.

Que 3.3. How derivation is defined in CFG ?

Answer

1. A derivation is a sequence of tokens that is used to find out whether a sequence of string is generating valid statement or not.
2. We can define the notations to represent a derivation.
3. First we define two notations $\underset{G}{\Rightarrow}$ and $\overset{*}{\underset{G}{\Rightarrow}}$.
4. If $\alpha \to \beta$ is a production of P in CFG and a and b are strings in $(V_n \cup V_t)^*$, then
$$a\alpha b \underset{G}{\Rightarrow} a\beta b.$$
5. We say that the production $\alpha \to \beta$ is applied to the string $a\alpha b$ to obtain $a\beta b$ or we say that $a\alpha b$ directly drives $a\beta b$.
6. Now suppose $\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_m$ are string in $(V_n \cup V_t)^*$, $m \geq 1$ and $\alpha_1 \underset{G}{\Rightarrow} \alpha_2, \alpha_2 \underset{G}{\Rightarrow} \alpha_3, \alpha_3 \underset{G}{\Rightarrow} \alpha_4, \ldots, \alpha_{m-1} \underset{G}{\Rightarrow} \alpha_m$.
7. Then we say that $\alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_m$, *i.e.*, we say α_1 drives α_m in grammar G. If α drives by exactly i steps, we say $\alpha_1 \overset{i}{\underset{G}{\Rightarrow}} \beta$.

Que 3.4. Construct a CFG for the language
$$L = \{wCw^R \mid w \in (a, b)^*\}.$$

Answer

1. w is a string of any combination of a and b.
2. Hence, w^R is also a string of any combination of a and b, but it is a string which is the reverse of w.
3. C is a terminal symbol like a, b.
4. Hence, if we take C as a mirror, we will be able to see the reflection of w in the w^R part. Like C, $abCba$, $abbaCabba$ like this.
5. It means there is some generating symbol in the middle by replacing which it adds same terminal symbol before and after C.
6. As $w \in (a, b)^*$ so null symbol are also accepted in the place of a, b.
7. That means only C is accepted by this language set.

From the above points the production rules will be :

Scanned by TapScanner

$S \rightarrow aSb / bSb / C$.
The grammar will be $G = \{V_N, \Sigma, P, S\}$, where
$V_N : \{S\}$,
$\Sigma : \{a, b, C\}$,
$P : S \rightarrow aSb / bSb / C$,
$S : \{S\}$.

Que 3.5. **Construct a CFG for given regular expression $(011 + 1)^* (01)^*$.**

Answer

1. The regular expression consists of two parts $(011 + 1)^*$ and $(01)^*$.
2. Hence, from the start symbol, we are taking two non-terminals each of them is producing one part. $(011 + 1)$ can be written as $011/ 1$.
3. As it is any combination of 0s and 1s which is represented by *. So, null string is also included in the language set.
4. From the above discussion the production rules (P) of the grammar will be $S \rightarrow BC, B \rightarrow AB/\varepsilon, A \rightarrow 011/1, C \rightarrow DC/\varepsilon, D \rightarrow 01$.
   The grammar will be $G = \{V_N, \Sigma, P, S\}$, where
   $V_N : \{S, A, B, C, D\}, \Sigma : \{0, 1\}, S : \{S\}$.

Que 3.6. **Find the CFG for the language**

$L = \{a^n b^m \mid n + m$ is even$\}$. **AKTU 2014-15, Marks 05**

OR

Design CFG for the language consisting of all strings of even length over {a, b}. **AKTU 2015-16, Marks 10**

Answer

Let G be CFG for language

$$L = \{a^n b^m \mid n + m \text{ is even}\}$$

then $G = (V, T, P, S)$
where, $V = \{S, A\}, T = \{a, b\}$
and productions P are defined as :

$$S \rightarrow AaBb \mid AB, \quad A \rightarrow aaA \mid \varepsilon, \quad B \rightarrow bbB \mid \varepsilon$$

PART-2

Derivation Trees and Ambiguity.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.7. **Define parse tree. What are the conditions for constructing a parse tree from a CFG ? Why parse tree construction is only possible for CFG ?**

Answer

1. A parse tree is pictorial representation of derivation, so it is also known as derivation tree.
2. Parse tree is the tree representation of deriving a Context Free Language (CFL) from a given Context Free Grammar (CFG).
3. A parse tree is an ordered tree in which left-hand-side of a production represents a parent node and children nodes are represented by the production's right-hand-side.

Conditions for constructing a parse tree from a CFG :

i. Each vertex of the tree must have a label. The label is a non-terminal or terminal or null (ε).
ii. The root of the tree is the start symbol, *i.e.*, S.
iii. The label of the internal vertices is non-terminal symbols $\in V_N$.
iv. If there is a production $A \rightarrow X_1 X_2 \ldots X_K$. Then for a vertex, label A, the children of that node will be $X_1 X_2 \ldots X_K$.
v. A vertex n is called a leaf of the parse tree if its label is a terminal symbol $\in \Sigma$ or null (ε).

Parse tree construction is only possible for CFG. This is because the properties of a tree match with the properties of CFG.

Que 3.8. **What do you mean by left most derivation and right most derivation with example ?**

Answer

1. **Left most derivation :** The derivation $S \rightarrow s$ is called a left most derivation, if the production is applied only to the left most variable (non-terminal) at every step.
   For example : Let us consider a grammar G that consist of production rules $E \rightarrow E + E \mid E * E \mid id$.
   Firstly take the production

$$E \rightarrow E + E \rightarrow \underline{E} * E + E \quad \text{(Replace } E \rightarrow E * E\text{)}$$
$$\rightarrow id * \underline{E} + E \quad \text{(Replace } E \rightarrow id\text{)}$$
$$\rightarrow id * id + \underline{E} \quad \text{(Replace } E \rightarrow id\text{)}$$
$$\rightarrow id * id + id \quad \text{(Replace } E \rightarrow id\text{)}$$

2. **Right most derivation :** A derivation $S \rightarrow s$ is called a right most derivation, if production is applied only to the right most variable (non-terminal) at every step.
   For example : Let us consider a grammar G having production.

$$E \rightarrow E + E \mid E * E \mid id.$$

Scanned by TapScanner

Start with production

$$E \to E * \underline{E} \quad \text{(Replace } E \to E + E\text{)}$$
$$\to E * E + \underline{E} \quad \text{(Replace } E \to id\text{)}$$
$$\to E * \underline{E} + id \quad \text{(Replace } E \to id\text{)}$$
$$\to E * id + id \quad \text{(Replace } E \to id\text{)}$$
$$\to id * id + id$$

Que 3.9. **Define derivation tree. Show the derivation tree for string '*aabbbb*' with the following grammar $S \to AB/\varepsilon$, $A \to aB$, $B \to Sb$.**

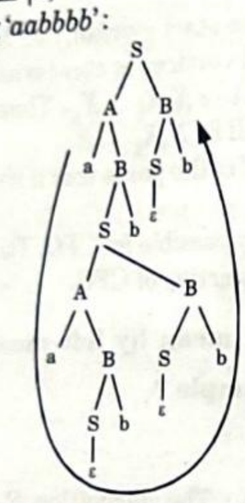
AKTU 2018-19, Marks 10

Answer

Derivation tree (parse tree) : Refer Q. 3.7, Page 3–4A, Unit-3.

Given grammar : $S \to AB \mid \varepsilon$, $A \to aB$, $B \to Sb$

Derivation tree for string '*aabbbb*':



Que 3.10. **Consider the grammar with the production. Compute the string *aababbb* with the left most and right most derivation. Draw the derivation tree.**

AKTU 2016-17, Marks 10

Answer

Grammar $G = (\{S\}, \{a, b\}, P, S)$

Production rules :

1. $S \to aSS$
2. $S \to b$

Left most derivation :

$$S \Rightarrow a\underline{S}S \quad \text{(Rule 1)}$$
$$\Rightarrow aa\underline{S}SS \quad \text{(Rule 1)}$$
$$\Rightarrow aab\underline{S}S \quad \text{(Rule 2)}$$
$$\Rightarrow aaba\underline{S}SS \quad \text{(Rule 1)}$$
$$\Rightarrow aabab\underline{S}S \quad \text{(Rule 2)}$$

$$\Rightarrow aababb\underline{S} \quad \text{(Rule 2)}$$
$$\Rightarrow aababbb \quad \text{(Rule 2)}$$

Right most derivation :

$$S \Rightarrow a\underline{S}S \quad \text{(Rule 1)}$$
$$\Rightarrow a\underline{S}b \quad \text{(Rule 2)}$$
$$\Rightarrow aaS\underline{S}b \quad \text{(Rule 1)}$$
$$\Rightarrow aaSaS\underline{S}b \quad \text{(Rule 1)}$$
$$\Rightarrow aaSa\underline{S}bb \quad \text{(Rule 2)}$$
$$\Rightarrow aa\underline{S}abbb \quad \text{(Rule 2)}$$
$$\Rightarrow aababbb \quad \text{(Rule 2)}$$

Derivation tree :

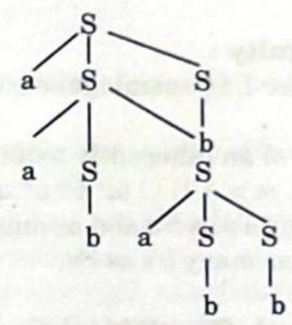


Fig. 3.10.1.

Que 3.11. **Explain ambiguity in context free grammar.**

Answer

A Context Free Grammar (CFG) G is ambiguous if there is at least one string in $L(G)$ having two or more distinct derivation tree.

For example : Let us consider CFG having productions $S \to S + S \mid a$. The string $a + a + a$ has two left most derivations.

$$S \to \underline{S} + S \to \underline{S} + S + S \to a + \underline{S} + S$$
$$\to a + a + \underline{S} \to a + a + a$$
$$S \to \underline{S} + S \to a + \underline{S} \to a + \underline{S} + S$$
$$\to a + a + \underline{S} \to a + a + a$$

The derivation tree will be

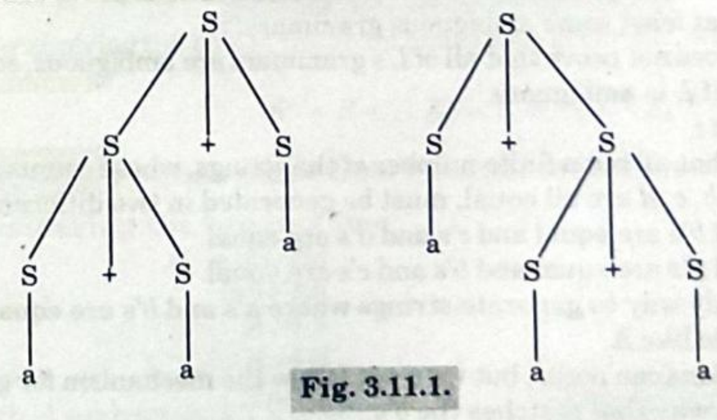


Fig. 3.11.1.

Scanned by TapScanner

The above grammar is ambiguous because it has two left most derivation tree and similarly we can show two right most derivation trees.

Que 3.12. Explain inherently ambiguous context tree grammar with example.

Answer

Statement : "If L is a context free language for which there exists an unambiguous grammar, then L is said to be unambiguous. If every grammar that generates L is ambiguous, then the language is said to be inherently ambiguous".

Proving inherent ambiguity :

1. If even one grammar for L is unambiguous, then L is an unambiguous language.
2. Let us see an example of an inherently ambiguous CFL.
   $L = \{a^n b^n c^m d^m \mid n > 1, m >= 1\} \cup \{a^n b^m c^m d^n \mid n >= 1, m >= 1\}$
3. Thus, there are as many a's as b's and as many c's and d's, or there are as many a's as d's and as many b's as c's.
   A grammar for L
   $S \rightarrow AB \mid C, A \rightarrow aAb \mid ab, B \rightarrow cBd \mid cd, C \rightarrow aCd \mid aDd, E \rightarrow bDc \mid bc$

Parse trees for *aabbccdd* :

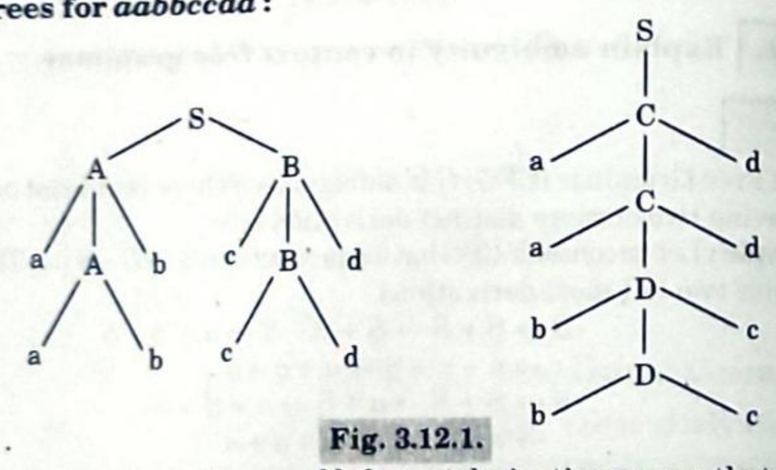


Fig. 3.12.1.

1. The previous parse trees and left most derivations prove that language L has at least some ambiguous grammars.
2. This does not prove that all of L's grammars are ambiguous, so we do not know if L is ambiguous.

Proof idea :

We argue that all but a finite number of the strings, whose counts of the four symbols a, b, c, d are all equal, must be generated in two different ways :

1. a's and b's are equal and c's and d's are equal.
2. a's and d's are equal and b's and c's are equal.
3. The only way to generate strings where a's and b's are equal is with a variable like A.
4. Variations can occur, but we must follow the mechanism for generating a's in a way that matches the b's.

5. This idea carries for the variables B, C, and D.
6. This argument will show that no matter what modifications we make to the basic grammar, it will generate at least some of the strings of the form $a^n b^n c^n d^n$.
7. Following this proof idea will show that L is inherently ambiguous, because no matter how the grammar is changed, it will remain ambiguous.

Que 3.13. How to convert ambiguous grammar into unambiguous grammar ?

Answer

To convert ambiguous grammar into unambiguous grammar, we use left and right recursion.

If the grammar contain arithmetic operator, we will apply the following rule :

a. If the grammar has left associative operator (such as +, –, *, /) then induce the left recursion.
b. If the grammar has right associative operator (exponential operator) then induces the right recursion.

Let consider an example :

$$E \rightarrow E + E \mid E * E \mid id$$

is an ambiguous grammar. We will design the parse tree for $id + id * id$ as follows :

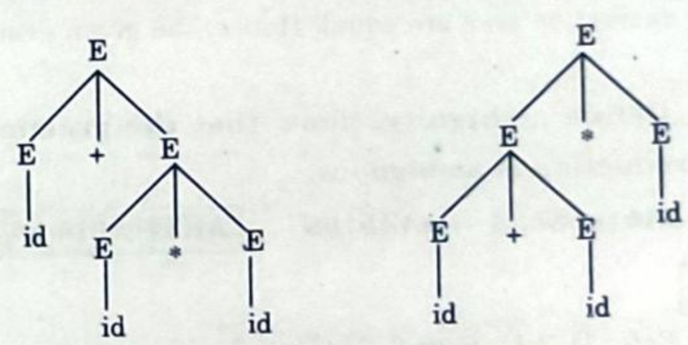


Fig. 3.13.1. Ambiguous grammar.

After applying the rule in the given ambiguous grammar, we get unambiguous grammar as :

$$E \rightarrow E + T, E \rightarrow T, E \rightarrow T * F, T \rightarrow F, F \rightarrow id$$

Que 3.14. Define parse tree. Find parse tree for the string *abbcde* considering the productions.

$$S \rightarrow aAcBe$$
$$A \rightarrow Ab$$
$$A \rightarrow b$$
$$B \rightarrow d$$

Is this ambiguous ? Justify. **AKTU 2017-18, Marks 07**

Scanned by TapScanner

Answer

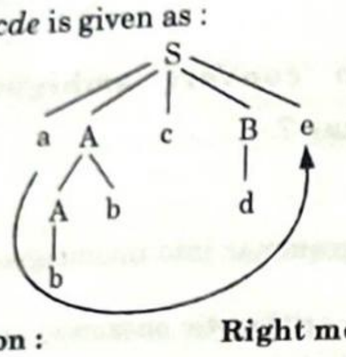
Parse tree : Refer Q. 3.7, Page 3–4A, Unit-3.

Numerical : Given : $S \to aAcBe$

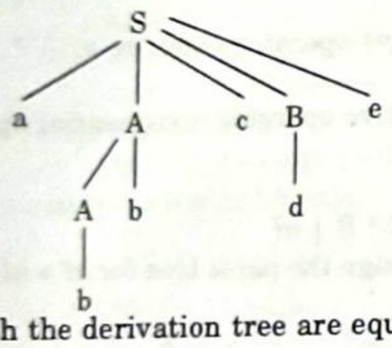
$A \to Ab \mid b$

$B \to d$

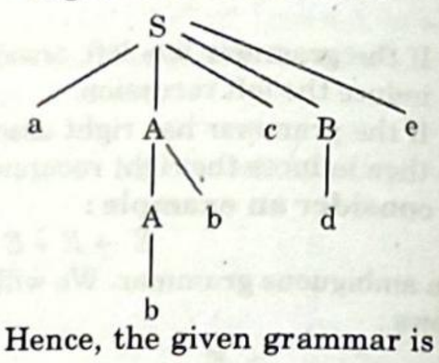
Parse tree for string *abbcde* is given as :



Left most derivation :



Right most derivation :



As, both the derivation tree are equal. Hence, the given grammar is not ambiguous.

Que 3.15. **Define ambiguity. Show that the grammar *G* with following production is ambiguous.**

$S \to a \mid aAb \mid abSb, A \to aAAb \mid bS$ **AKTU 2018-19, Marks 10**

Answer

Ambiguity : Refer Q. 3.11, Page 3–7A, Unit-3.

Numerical :

Given grammar :

$S \to a \mid aAb \mid abSb$

$A \to a\,AAb \mid bS$

Let us consider a string "*abab*", if the string forms two left most derivation tree or two right most derivation tree then it will be ambiguous grammar.

i. **Left most derivation :**

$S \to abSb \qquad (\because \; S \to a)$

$\to abab$

ii. **Again left most derivation :**

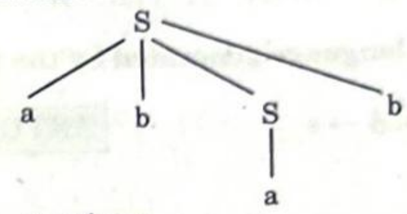
$S \to aAb$

$\to a\,\underline{A}\,b \qquad (\because \; A \to bS)$

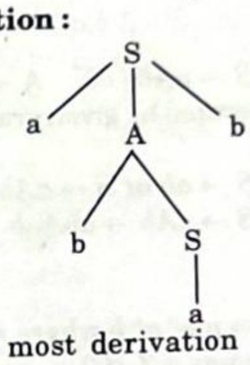
$\to ab\,\underline{S}\,b \qquad (\because \; S \to a)$

$\to abab$

Parse tree for (i) derivation :



Parse tree for (ii) derivation :



Since, there are two left most derivation tree, therefore grammar is ambiguous.

PART-3

Regular Grammars : Right Linear and Left Linear Grammars.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.16. **Write short note on right linear and left linear grammar.**

Answer

Left linear grammar :

In a left linear grammar, all productions have one of the two forms:

$$V \to VT^* \quad \text{or} \quad V \to T^*$$

That is, the left-hand side must consist of a single variable, and the right-hand side consists of an optional single variable followed by any number of terminals.

For example :

$$S \to Sa/Ab, \quad A \to Bc, \quad B \to d$$

Right linear grammar :

In a right linear grammar, all productions have one of the two forms :

$$V \to T^*V \quad \text{or} \quad V \to T^*$$

Scanned by TapScanner

That is, the left-hand side must consist of a single variable, and the right hand side consists of any number of terminals followed by a single variable.

For example :

$$S \to aS/bA, \quad A \to cB, \quad B \to d$$

Que 3.17. **Find the language generated by the following grammar.**

AKTU 2014-15, Marks 05

$S \to aAb/ab, A \to bAa, A \to \varepsilon$

Answer

Given grammar is as follows :

$$S \to aAb / ab, \quad A \to bAa, \quad A \to \varepsilon$$

Let us analyze the strings generated by given grammar. The minimum string which can be generated is ab.

$$S \to ab \text{ or } S \to aAb \to ab \qquad (\text{By } A \to \varepsilon$$
$$S \to aAb \to abAab$$
$$\vdots$$
$$\to a\, b^n\, a^n\, b \text{ where } n \geq 0$$

So, the language of grammar is $L(G)$.

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

Que 3.18. **Find context free grammar for following languages with ($n >= 0, m >= 0, k >= 0$).**

i. $L = \{a^n b^n c^k \mid k >= 3\}$

ii. $L = \{a^m b^n c^k \mid n = m \text{ or } m <= k\}$.

Answer

i. Let us assume CFG for L is G

$$G = \{V_n, V_t, P, S\}$$
$$V_n = \{S, X, Y, Z\}$$
$$V_t = \{a, b, c\}$$
$$S \to XY$$
$$X \to aXb \mid ab \mid \varepsilon$$
$$Y \to ccZ$$
$$Z \to cZ \mid \varepsilon$$

ii. Let us assume $L = L_1 \cup L_2$

where $L_1 = \{a^m b^n c^k \mid n = m\}$

and $L_2 = \{a^m b^n c^k \mid m <= k\}$

Let us consider L_1 first, let CFG for L_1 be G_1

$$G_1 = \{V_n, V_t, P_1, S_1\}$$
$$V_n = \{S_1, A, B\}$$
$$V_t = \{a, b, c\}$$

P_1 is defined as follows

$$S_1 \to AB$$
$$A \to aAb/\varepsilon$$

$$B \to cB/\varepsilon$$

Let as assume that CFG for L_2 is G_1

$$G_2 = \{V_n, V_t, P_2, S_2\}$$
$$V_n = \{S_2, X, Y\}$$
$$V_t = \{a, b, c\}$$

P_2 is defined as follows

$$S_2 \to aS_2c / aYc$$
$$Y \to C / Yc \mid \varepsilon$$
$$C \to bC / \varepsilon$$

We can define CFG for L (since $L = L_1 \cup L_2$)

Let us assume that CFG for L is G

$$G = \{V_n, V_t, P, S\}$$
$$V_n = \{S_1, S_2, A, B, X, Y\}$$
$$V_t = \{a, b, c\}$$

P is define as

$$S \to S_1/S_2$$
$$S_1 \to AB$$
$$A \to aAb/\varepsilon$$
$$B \to cB/\varepsilon$$
$$S_2 \to aS_2c / aYc$$
$$Y \to X / Yc\varepsilon$$
$$X \to bX/\varepsilon$$

Que 3.19. **Discuss inherent ambiguity of context free languages with suitable example. Construct the context free grammar that accepts language $L = \{a^i b^j c^k \mid i = j \text{ or } j = k; i, j, k$ are positive integers$\}$.**

AKTU 2017-18, Marks 07

Answer

Inherent ambiguity of context free grammar : Refer Q. 3.12, Page 3–8A, Unit-3.

Numerical :

Let us assume $L = L_1 \cup L_2$

where $L_1 = \{a^i b^j c^k \mid i = j\}$

and $L_2 = \{a^i b^j c^k \mid j = k\}$

Let us consider L_1 first, let CFG for L_1 be G_1

$$G_1 = \{V_1, T_1, P_1, S_1\}$$
$$V_1 = \{S_1, A, B\}$$
$$T_1 = \{a, b, c\}$$

P_1 is defined as follows :

$$S_1 \to AB, \quad A \to aAb/\varepsilon, \quad B \to cB/\varepsilon$$

Now let us assume that CFG for language L_2 is G,

$$G_2 = \{V_2, T_2, P_2, S_2\}$$
$$V_2 = \{S_2, C, D\}$$
$$T_2 = \{a, b, c\}$$

Productions are defined as follows :

Scanned by TapScanner

$$S_2 \to CD, \quad C \to aC/\varepsilon, \quad D \to bDc/\varepsilon$$

By the help G_1 and G_2, we can define CFG for the language L. Let it G.

$$G = (V, T, P, S)$$
$$V = \{S, S_1, S_2, A, B, C, D\}$$
$$T = \{a, b, c\}$$

Productions are defined as follows :

$$S \to S_1/S_2, \quad S_1 \to AB, \quad A \to aAb/\varepsilon, \quad B \to cB/\varepsilon,$$
$$S_2 \to CD, \quad C \to aC/\varepsilon, \quad D \to bDc/\varepsilon$$

Que 3.20. **Define context free grammar. Find a context fre grammar for the following language :**

AKTU 2014-15, Marks 05

$L = \{a^n b^{2n} c^m \mid n, m >= 0\}$

Answer

Context Free Grammar (CFG) : Refer Q. 3.1, Page 3–2A, Unit-3.

Numerical :

1. If string starts with 'a' then number of a's must follow b's and the numb of b's are twice than the number of a's.
2. If string does not start with a after any b or c, and no b after any c. Let us assume CFG for language is G.

$$G = (V_n, V_t, P, S)$$
$$V_n = \{S, A, B\}$$
$$V_t = \{a, b, c\}$$

P is defined as follows :

$$S \to AB, \quad A \to aAbb/\varepsilon, \quad B \to cB/\varepsilon$$

Que 3.21. **Construct a CFG for the following language :**

AKTU 2014-15, Marks 10

$L = \{a^m b^n \mid m \neq n\}$.

Answer

If $m \neq n$, then there are only two cases possible.

Case 1 : $m > n$

$\therefore$ Language L on condition $m > n$ is L_1

and $$L_1 = \{a^m b^n : m > n\}$$

Let G_1 be the CFG for the language L_1,

$$G_1 = (V_1, T_1, P_1, S_1)$$
$$V_1 = \{S_1, A, B\}$$
$$T_1 = \{a, b\}$$

then productions P_1 are defined as :

$$S_1 \to AB, \quad B \to aBb/\varepsilon, \quad A \to aA/a$$

Case 2 : $m < n$

Let L on condition $m < n$ is L_2

and $$L_2 = \{a^m b^n : m < n\}$$

Let CFG for L_2 be G_2

$$G_2 = (V_2, T_2, P_2, S_2)$$
$$V_2 = \{S_2, C, D\}$$
$$T_2 = \{a, b\}$$

P_2 are defined as follows :

$$S_2 \to CD, \quad C \to aCb/\varepsilon, \quad D \to bD/b$$

By the help of G_1 and G_2, we can write CFG for L as

$$S \to S_1/S_2, \quad S_1 \to AB, \quad A \to aA/a, \quad B \to aBb/\varepsilon$$
$$S_2 \to CD, \quad C \to aCb/\varepsilon, \quad D \to bD/b$$

where S is start symbol of CFG for L.

PART-4

Conversion of FA into CFG and Regular Grammar into FA.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.22. **Explain the conversion of FA into CFG.**

Answer

Construction of Regular Grammar Equivalent to a given DFA M :

1. Let $M = (\{q_0, q_1, \ldots, q_n\}, \Sigma, \delta, q_0, F)$ be the given DFA.
2. To construct an equivalent grammar G, our productions of G should correspond to transitions in the DFA. Also there should be a provision for terminating the derivation, once a transition terminating at some final state is encountered.
3. So, the grammar G can be constructed as follows :

$$G = (\{A_0, A_1, \ldots A_n\}, \Sigma, P, A_0)$$

where the set of productions P are constructed by

i. $A_i \to aA_j$, is included in P if $\delta(q_i, a) = a_j$, and $q_i \notin F$.

ii. $A_j \to aA_j$, and $A_i \to a$ are included in P if $\delta(q_i, a) = q_j$ and $q_j \in F$.

4. The construction of the grammar is as follow :
   i. The grammar will have n non-terminals if there are n states in a DFA, *i.e.* each state of the DFA is represented by one non-terminal.
   ii. If q_0 is a start of the DFA, then the non-terminal corresponding to q_0 is the start-symbol of the grammar.
   iii. If there are m transitions (edges) from some state to other then we add m productions. However, if the transition leads to a final state of DFA, we add additional production to provide termination of derivation.

Que 3.23. **Construct a regular grammar for the finite automata shown in Fig. 3.23.1.**

Scanned by TapScanner

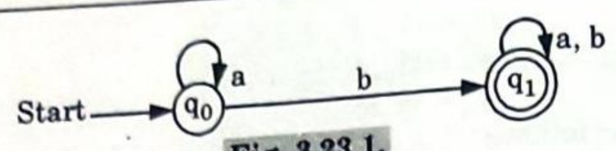


Fig. 3.23.1.

Answer

The grammar G equivalent to the FA shown in Fig. 3.23.1, will be

$$G = (V_N, \Sigma, P, A_0), \text{ where}$$
$$V_N = \{A_0, A_1\}$$
$$\Sigma = \{a, b\}$$

A_0 is the start symbol

and $P = \{A_0 \to aA_0,\ A_0 \to bA_1 \mid b,\ A_1 \to aA_1 \mid a,\ A_1 \to bA_1 \mid b\ \}$

1. Since there are only two states in the FA, we will have two non-terminals, A_0 and A_1, A_0 represents q_0 and A_1 represents q_1.
2. Since q_0 is the start state, A_0 will be start symbol.
3. Since each edge is either labeled as a or b, we will have a and b as the set of terminals.
4. There are four transitions in the FA, three of them are to the final state q_1, and one of them to the non-final state. So we will have total 7 productions, two each for the three transitions leading to final state and 1 for the transition to non-final state.

Que 3.24. **Construct a *NFA* corresponding to the following grammar.**

$$S \to aA \mid bB, A \to aS \mid a, B \to bS \mid b$$

Answer

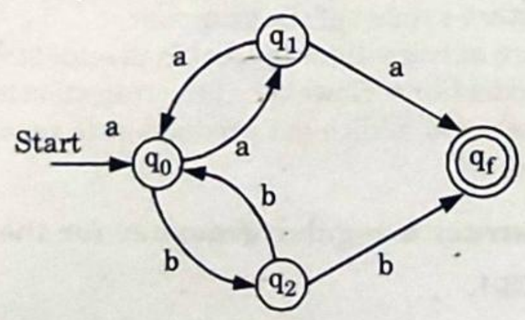
The NFA accepting the language generated by the above grammar will be

$$M = (\{q_0, q_1, q_2, q_f\}, \{a, b\}, \delta, q_0, \{q_f\})$$

where q_0 corresponds to S, q_1 to A and q_2 to B q_f is the new final state. q_0 is the initial state.

$S \to aA$ induces transition from q_0 to q_1 on a.
$S \to bB$ induces transition from q_0 to q_2 on b.
$A \to aS$ induces transition from q_1 to q_0 on a.
$A \to a$ induces transition from q_1 to q_f on a.
$B \to bS$ induces transition from q_2 to q_0 on b.
$B \to b$ induces transition from q_2 to q_f on b.

The NFA is given by



PART-5

Simplification of CFG, Normal Forms : Chomsky Normal Form (CNF), Greibach Normal Form (GNF), Chomsky Hierarchy.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.25. **Explain the simplification of Context Free Grammar (CFG).**

Answer

Let L be a non-empty context free language, then it can be generated by a CFG G with the following properties :

1. **Eliminating useless symbols :**
   a. In this, we identify those symbols, which do not play any role in the derivation of any string w in $L(G)$ (these symbols are called useless symbols) and then eliminate the identified production, which contains useless symbols from the Context Free Grammar (CFG).
   b. Reduction of a given grammar G, involves following steps :
      i. Identified non-generating symbols in given CFG and eliminate those productions which contains non-generating symbols.
      ii. Identified non-reachable symbols in grammar and eliminate those productions which contain non-reachable symbols.

   Then, after this process, CFG will have only useful symbols.
2. **Removal of unit production :**
   a. A unit production is production of the form :
      Non-terminal → One non-terminal
   b. Unit production increases the cost of derivation in a grammar.

   Following algorithm can be used to eliminate the unit production.

   Algorithm :

   While (there exist a unit productions, $A \to B$)
   {
   select a unit production $A \to B$, such that there exist a production $B \to \alpha$, where α is a terminal.
   For (every non-unit production, $A \to \alpha$)
   Add production $A \to \alpha$ to the grammar
   Eliminate $A \to B$ from the grammar
   }
3. **Removal of ε-productions and nullable non-terminal :**
   a. In a CFG, a non-terminal A is nullable variable if there is a production $A \to \varepsilon$ or there is a derivation that start with A and leads to ε.

Scanned by TapScanner

b. To eliminate ε-productions from a grammar G we use the following technique :
   i. First we look for all productions, whose right side contains $A \to \varepsilon$.
   ii. Replace each occurrence of A in each of these productions to to obtain the non ε-productions.
   iii. Now these resultant non ε-production must be added to the grammar.

Que 3.26. **For given CFG, find equivalent CFG with no useless variables :**

$$S \to AB \mid AC, A \to aAb \mid bAa \mid a, B \to bbA \mid aaB \mid AB$$
$$C \to abCa \mid aDb, D \to bDa \mid aC$$

Answer

Since, non-terminal C and D do not derives any terminal so C and D are useless symbol. Remove the production which contains C and D.
So, the CFG without useless symbol is given as follows :

$$S \to AB, \quad A \to aAb \mid bAa \mid a, \quad B \to bbA \mid aaB$$

Que 3.27. **Consider the following grammar and remove ε-productions** $S \to aSa, \quad S \to bSb/\varepsilon$

Answer

Here $S \to \varepsilon$ is the ε-production, so let us replace occurrence of S by ε in the following productions

$$S \to aSa, \quad S \to bSb$$

We will get following new productions

$$S \to aa/bb$$

add these productions in the grammar and remove $S \to \varepsilon$ from the grammar

$$S \to aSa / aa / bb / bSb$$

It is ε-production free grammar.

Que 3.28. **Explain Chomsky Normal Form (CNF) with example. Write the steps to convert CFG to CNF.**

OR

Reduce the given grammar $G = (\{S, A, B\}, \{a, b\}, P, S)$ to Chomsky Normal Form.
Where P is defined as :
$S \to bA \mid aB, \quad A \to bAA \mid aS \mid a, \quad B \to bBB \mid bS \mid b$

AKTU 2017-18, Marks 07

Answer

Chomsky Normal Form :
If a CFG has only production of the form

Non-terminals → string of exactly two non-terminals or
Non-terminals → one terminal then grammar is said to be in CNF.
For example :
Let us consider a grammar
$G = (\{S, A, B\}, \{a, b\}, P, S)$, where S is the start symbol and P is given by

$$S \to bA / aB, \quad A \to bAA / aS / a, \quad B \to aBB / bS / b$$

Now let us find an equivalent CNF of it.
As we know that right side in CNF either contains two non-terminals or one terminal.
So, it is clear that in the first production above, we have to replace terminal 'b' by a non-terminal say C_b and 'a' by C_a, hence grammar becomes

$$S \to C_bA / C_aB, \quad A \to C_bAA / C_aS / a$$
$$B \to C_aBB / C_bS / b, \quad C_a \to a, \quad C_b \to b$$

Now first rule is in CNF. In second rule C_bAA is not in CNF. So we can replace AA by a non-terminal say D and similarly BB by E, we obtain

$$S \to C_bA / C_aB, \ A \to C_bD / C_aS / a, \ B \to C_aE / C_bS / b$$
$$C_a \to a, \quad C_b \to b, \quad D \to AA, \quad E \to BB$$

This grammar is equivalent CNF of the given grammar.
Steps to convert a given CFG to CNF :
Step 1 : If the start symbol S occurs on some right side, create a new start symbol S' and a new production $S' \to S$.
Step 2 : Remove null productions (Using the null/ production lecture).
Step 3 : Remove unit productions (Using the unit production removal).
Step 4 : Replace each production $A \to B_1 \ldots\ldots B_n$, where $n > 2$, with $A \to B_1C$ where $C \to B_2, \ldots\ldots, B_n$ repeat this step for all production having two or more symbols on the right side.
Step 5 : If the right side of any production is in the form $A \to cB$ where c is a terminal and A and B are non-terminals, then the production is replaced by $A \to XB$ and $X \to c$ repeat this step for every production which is of the form $A \to cB$.

Que 3.29. **Convert the given grammar in Chomsky Normal Form (CNF) $S \to ABa \quad A \to aab \quad B \to Ac$.**

Answer

Step 1 : Consider the production $S \to ABa$, not in CNF. Let us consider, another productions $D_1 \to a$, $D_2 \to AB$. So production will be

$$S \to D_2D_1 \text{ and } D_1 \to a \text{ and } D_2 \to AB$$

Step 2 : Consider the second production

$$S \to aab \text{ not in CNF.}$$

Let us consider a production $D_3 \to b$ and substitute a by D_1 using production $D_1 \to a$. So production will be :

$$A \to D_1D_1D_3, \text{ it is also not in CNF.}$$

Therefore, consider a new production $D_4 \to D_1D_3$
Thus, $A \to D_1D_4$ and $D_4 \to D_1D_3$ and $D_1 \to a$ and $D_3 \to b$

Scanned by TapScanner

Step 3 : Consider the third production $B \to Ac$ is not in CNF. Consider a production $D_5 \to c$. The production will be :

$B \to AD_5$ which is in CNF.

Therefore, the productions for given grammar in CNF form will be,

$$S \to D_2D_1, \quad A \to D_1D_4, \quad B \to AD_5$$

$$D_1 \to a, \quad D_2 \to AB, \quad D_3 \to b, \quad D_4 \to D_1D_3, \quad D_5 \to c$$

Que 3.30. Explain GNF and the process of converting given CFG into GNF.

Answer

GNF : A grammar is said to be in GNF if every productions of the grammar is of the form

non-terminal → (single terminal) (string of non-terminals)

non-terminal → single terminal

For converting a CFG into GNF, we use two lemmas which are as follow :

Lemma I : If $G = (V_N, \Sigma, P, S)$ be a CFG and if $A \to A\alpha$ and $A \to \beta_1/\beta_2/\ldots/\beta_n$ belongs to the productions rules (P) of G. Then a new grammar $G' = (V_N, \Sigma, P', S)$ can be constructed by replacing $A \to \beta_1/\beta_2 \ldots/\beta_n$ in $A \to A\alpha$, which will produce

$A \to \beta_1\alpha/\beta_2\alpha \ldots/\beta_n\alpha$

This production belongs to P' in G'. Then $L(G)$ is equivalent to $L(G')$

Lemma II : Let $G = (V_N, \Sigma, P, S)$ be a CFG and the set of A productions belongs to P be $A \to A\alpha_1/A\alpha_2 \ldots/A\alpha_m/\beta_1/\beta_2/\ldots/\beta_n$.

Introduce a new non-terminal X. Let $G = (V'_N, \Sigma, P', S)$ where $V'_N = (V_N \cup X)$ and P can be formed by replacing the A-productions by

$A \to \beta_1 | \beta_2 | \beta_3 | \ldots | \beta_n$

$A \to \beta_1X | \beta_2X | \beta_3X | \ldots | \beta_nX$

$X \to \alpha_1 | \alpha_2 | \alpha_3 | \ldots | \alpha_n$

$X \to \alpha_1X | \alpha_2X | \ldots | \alpha_nX$

Process for conversion a CFG into GNF :

Step I : Eliminate null productions and unit productions from the grammar and convert the grammar into CNF.

Step II : Rename the non-terminals of V_N as $(A_1, A_2 \ldots A_n)$, with start symbol A_1.

Step III : Using Lemma I modify the productions. Such that left-hand-side variable subscript is less than or equal to right-hand-side starting variable subscript, that is in mathematical notation it can be said all the productions will be in the form $A_i \to A_jV$ where $i \le j$.

Step IV : By repeating application of Lemma I and Lemma II, all the production of the modified grammar will come into GNF.

Que 3.31.

i. **Convert the following CFG into CNF**

$S \to XY \mid Xn \mid p$

$X \to mX \mid m$

$Y \to Xn \mid o$

ii. **Convert the following CFG into CNF** $S \to ASA \mid aB$, $A \to B \mid S, B \to b \mid \varepsilon$

AKTU 2016-17, Marks 10

Answer

i. In CNF R.H.S. contains either two non-terminal or one terminal. So, in the first production replace terminal 'n' by non-terminal 'C_n' and 'p' by 'C_p' and similarly 'm' by 'C_m' and 'o' by 'C_o'.

Hence, the grammar becomes

$S \to XY | XC_n | C_p, \quad X \to C_mX | C_m, \quad Y \to XC_n | C_o$

$C_n \to n, \quad C_p \to p, \quad C_m \to m, \quad C_o \to o$

Now the grammar is in CNF.

ii. Given $S \to ASA \mid aB, A \to B \mid S, B \to b \mid \varepsilon$

1. Since S appears in R.H.S, we add a new state S_0 and $S_0 \to S$ is added to the production set and it becomes :

   $S_0 \to S, S \to ASA \mid aB, A \to B \mid S, B \to b \mid \varepsilon$

2. Now we will remove the null productions :

   $B \to \varepsilon$

   After removing $B \to \varepsilon$, the production set becomes :

   $S_0 \to S, S \to ASA \mid aB \mid a, A \to B \mid S \mid \varepsilon, B \to b$

   After removing $A \to \varepsilon$, the production set becomes :

   $S_0 \to S, S \to ASA \mid aB \mid a \mid AS \mid SA \mid S, A \to B \mid S, B \to b$

3. Now we will remove the unit productions.

   After removing $S \to S$, the production set becomes :

   $S_0 \to S, S \to ASA \mid aB \mid a \mid AS \mid SA, A \to B \mid S, B \to b$

   After removing $S_0 \to S$, the production set becomes :

   $S_0 \to ASA \mid aB \mid a \mid AS \mid SA, S \to ASA \mid aB \mid a \mid AS \mid SA$

   $A \to B \mid S, B \to b$

   After removing $A \to B$, the production set becomes :

   $S_0 \to ASA \mid aB \mid a \mid AS \mid SA, S \to ASA \mid aB \mid a \mid AS \mid SA$

   $A \to S \mid b$

   $B \to b$

   After removing $A \to S$, the production set becomes :

   $S_0 \to ASA \mid aB \mid a \mid AS \mid SA, S \to ASA \mid aB \mid a \mid AS \mid SA$

   $A \to b \mid ASA \mid aB \mid a \mid AS \mid SA, B \to b$

4. Now we will find out more than two variables in the R.H.S.

   Here, $S_0 \to ASA, S \to ASA, A \to ASA$ violates two non-terminals in R.H.S.

   Let us consider another production $X \to SA$, so the production will become

   $S_0 \to AX \mid aB \mid a \mid AS \mid SA$

   $S \to AX \mid aB \mid a \mid AS \mid SA$

   $A \to b \mid AX \mid aB \mid a \mid AS \mid SA$

   $B \to b, \quad X \to SA$

Scanned by TapScanner

5. Productions $S_0 \to aB, S \to aB, A \to aB$ are not in CNF. So consider another production $Y \to a$ thus the production become $S_0 \to YB$, $S \to YB, A \to YB$

Therefore, the production for given CFG in CNF form will be given as :

$S_0 \to AX \mid YB \mid a \mid AS \mid SA$
$S \to AX \mid YB \mid a \mid AS \mid SA$
$A \to b \mid AX \mid YB \mid a \mid AS \mid SA$
$B \to b$
$X \to SA$
$Y \to a$

Que 3.32. **Convert the following grammar in GNF : $S \to AB, A \to BS / a, B \to SA / b$**

AKTU 2018-19, Marks 10

Answer

Given grammar :

$$S \to AB, \quad A \to BS \mid a, \quad B \to SA \mid b$$

Let rename S as A_1, A as A_2, B as A_3 then given grammar becomes,

$$A_1 \to A_2 A_3$$
$$A_2 \to A_3 A_1 \mid a$$
$$A_3 \to A_1 A_2 \mid b$$

Let start with A_3. The rule for A_3 is

$$A_3 \to A_1 A_2 \mid b$$

Now replace A_1 by rule of A_1

$$A_3 \to A_2 A_3 A_2 \mid b$$
$$A_3 \to A_3 A_1 A_3 A_2 \mid A_3 A_2 \mid b$$

According to lemma I if

$$A \to A\alpha_1 \mid A\alpha_2 \mid \ldots\ldots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots\ldots \mid \beta_n$$
$$A \to \beta_1 Z \mid \beta_2 Z \mid \ldots\ldots \mid \beta_n Z$$
$$Z \to \alpha_1 Z \mid \alpha_2 Z \mid \ldots\ldots \mid \alpha_n Z$$

We can map this lemma to A_3 rule as

$$A = A_3, \quad \alpha_1 = A_1 A_3 A_2, \quad \beta_1 = aA_3 A_2, \beta_2 = b$$

Then we get,

$$A_3 \to aA_3 A_2 \mid b$$
$$A_3 \to aA_3 A_2 Z \mid bZ$$
$$Z \to A_1 A_3 A_2$$
$$Z \to A_1 A_3 A_2 Z$$

Thus now we have obtained A_3 being in GNF. Now, consider production for A_2

$$A_2 \to A_3 A \mid a$$

Replace the value of A_3 by its recently GNF rule

$$A_2 \to aA_3 A_2 ZA_1 \mid bZA_1 \mid aA_3 A_2 A_1 \mid bA_1 \mid a$$

Now consider production for A_1

$$A_1 \to A_2 A_3$$

Replace the value of A_2 and A_3

$$A_1 \to aA_3 A_2 ZA_1 A_3 \mid bZA_1 A_3 \mid aA_3 A_2 A_1 A_3 \mid bA_1 A_3 \mid aA_3$$

Now, consider Z rule,

$$Z \to A_1 A_3 A_2 Z$$
$$Z \to A_1 A_3 A_2$$

Replace A_1 with rule of A_1 in GNF

$$Z \to aA_3 A_2 ZA_1 A_3 A_3 A_2 Z \mid bZA_1 A_3 A_3 A_2 Z \mid$$
$$aA_3 A_2 A_1 A_3 A_3 A_2 Z \mid bA_1 A_3 A_3 A_2 Z \mid aA_3 A_3 A_2 Z$$
$$Z \to aA_3 A_2 ZA_1 A_3 A_3 A_2 \mid bZA_1 A_3 A_3 A_2 \mid aA_3 A_2 A_1 A_3 A_3 A_2$$
$$bA_1 A_3 A_3 A_2 \mid aA_3 A_3 A_2$$

Now rewrite the rules by converting back $A_1 = S$

$$A_2 = A \; A_2 = B$$

$$S \to aBAZSB \mid bZSB \mid aBASB \mid bSB \mid aB$$
$$A \to aBAZS \mid bZS \mid aBAS \mid bS \mid a$$
$$Z \to aBAZSBBAZ \mid bZSBBAZ \mid aBASBBAZ \mid bSBBAZ \mid aBBAZ$$
$$Z \to aBAZSBBA \mid bZSBBA \mid aBASBBA \mid bSBBA \mid aBBA$$

is equivalent GNF.

Que 3.33. **What is difference between Chomsky Normal Form (CNF) and Greibach Normal Form (GNF) ?**

Answer

S. No.	CNF	GNF
1.	A CFG G is in CNF if every production is of the form $a \to A$, or $A \to BC$, and $S \to \varepsilon$ is in G if $\varepsilon \in L(G)$.	A CFG is in GNF if every production is of the form $A \to a\alpha$, where $\alpha \in V^*_N$ and $a \in \Sigma$ (α may be ε) and $S \to \varepsilon$ is in G if $\varepsilon \in L(G)$.
2.	When ε is in $L(G)$, we assume that S does not appear on the R.H.S. of any production.	When $\varepsilon \in L(G)$, we assume that S does not appear on the R.H.S. of any production.
3.	For example, consider G whose productions are $S \to AB \mid \varepsilon, A \to a, B \to b$ is in CNF.	For example, G given by $S \to aAB \mid \varepsilon$, $A \to bC$, $B \to b$, $C \to c$ is in GNF.

Que 3.34. **Explain the parse tree with an example. Reduce the CFG into GNF whose productions are**

$S \to aSb$

$S \to ab$

AKTU 2015-16, Marks 10

Answer

Parse tree : Refer Q. 3.7, Page 3–4A, Unit-3.

Scanned by TapScanner

For example : Refer Q. 3.10, Page 3–6A, Unit-3.

Numerical :

Step 1 : Let us find CNF of given GNF as follows :

$$S \to ASB$$
$$S \to AB$$
$$A \to a$$
$$B \to b$$

Step 2 : Rename the non-terminals as A_1, A_2, A_3, by putting $S = A_1, A = A_2, B = A_3$, then CNF can be written as follows :

$$A_1 \to A_2A_1A_3$$
$$A_1 \to A_2A_3$$
$$A_2 \to a$$
$$A_3 \to b$$

Step 3 : $A_1 \to A_2A_1A_3$ and $A_1 \to A_2A_3$ are not in GNF, so let us replace them as follows :

$$A_1 \to aA_1A_3$$
$$A_1 \to aA_3$$

So, GNF is,

$$A \to aA_1A_3 \mid aA_3$$
$$A_2 \to a$$
$$A_3 \to b$$

Que 3.35. **Convert the CFG into GNF.**

$S \to aSbA$

$A \to Sa/a$

AKTU 2014-15, Marks 05

Answer

First we convert this grammar into CNF.

We introduce $C \to a$

$B \to b$

Now the grammar will be :

$$S \to CSBA, \quad C \to a$$
$$A \to SC/a, \quad B \to b$$

This grammar is in CNF.

Let, rename S as A_1, A as A_2, B as A_3 and C as A_4.

Then,

$$A_1 \to A_4A_1A_3A_2$$
$$A_2 \to A_1A_4/a$$
$$A_4 \to a$$
$$A_3 \to b$$

All productions are fine except $A_2 \to A_1A_4$.

Using $A_1 \to A_4A_1A_3A_2$ in A_2

we get, $A_2 \to A_4A_1A_3A_2A_4$

$\therefore$ $A_2 \to A_4A_1A_3A_2A_4/a$

Now, our grammar looks like :

$$A_1 \to A_4A_1A_3A_2$$
$$A_2 \to A_4A_1A_3A_2A_4/a$$
$$A_3 \to b$$

$$A_4 \to a$$

Now, substituting A_4 in A_1 and A_2

we get,

$$A_1 \to aA_1A_3A_2$$
$$A_2 \to aA_1A_3A_2A_4/a$$
$$A_3 \to b$$
$$A_4 \to a$$

which is in GNF.

Que 3.36. **Explain the Chomsky hierarchy of languages. Determine the type of the following grammar :**

$S \to aAb/\varepsilon, A \to aA/Ab/a/b$

AKTU 2014-15, Marks 05

Answer

Chomsky hierarchy of languages :

Chomsky has suggested four different classes of phrase structure grammar as follows :

1. **Type-0 grammar (Unrestricted grammar) :**
   a. Type-0 grammar are constructed with no restrictions on replacement rule, except that a non-terminal.
   b. A non-terminal must appear in the string on the left side. The language generated is called recursive enumerable language.
   c. Thus, a type-0 grammar is :
      i. An alphabet Σ of terminal symbols.
      ii. An alphabet $\forall$ of non-terminals, including a start symbol.
      iii. A set of production rule $\alpha \to \beta$, where α and β are from $(\Sigma \cup V)^*$, 'α' contains at least one non-terminal and there is no restriction on 'β'.
   d. The type-0 grammar is recognized by turing machines.
2. **Type-1 grammar (Context-sensitive grammar) :**
   a. A grammar is said to be type-1 grammar or context-sensitive grammar if it follows the following conditions :
      i. Each production in the form $\alpha \to \beta$, and length of 'α' is less than or equal to length of β, *i.e.*, there are no empty production, those in which right side is empty string ε.
      ii. Each production of the form $\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$, with $\beta \neq \varepsilon$.
   b. The turing machine can be constructed to recognize the context-sensitive language generated by context-sensitive grammar (CSG).
3. **Type-2 grammar (Context free grammar) :**
   a. A grammar is said to be type-2 grammar, if the production is in the form of $A \to \alpha$, where A is a non-terminal and 'α' is a sentential form *i.e.*, $\alpha \in (V \cup T)^*$ *i.e.*, α can be ε.

Scanned by TapScanner

b. The left hand side of production must contain only one non-terminal.

c. The type-2 grammar can be recognized by push down automata.

4. Type-3 grammar (Regular grammar) :

a. A grammar is said to be type-3 grammar if the production is in the form $A \rightarrow a$ or $A \rightarrow aB$, *i.e.*, the left hand side of each production should contain only one non-terminal or first symbol on right hand side must be a terminal and may be followed by a non-terminal.

b. The language generated by this grammar is recognized by finite state machine.

c. These regular languages can also be expressed by simpler expressions called regular expression.

Numerical :

Given : $S \rightarrow aAb / \varepsilon$

$A \rightarrow aA / Ab / a / b$

The given grammar is a type-2 grammar or context free grammar as it contains only type-2 productions. Also, it is of the form :

$A \rightarrow \alpha$, where $A \in V_N$ and $\alpha \in (V_N \cup \Sigma)^*$.

PART-6

Programming Problem Based on the Properties of CFGs.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.37. **Discuss closure property of CFL.**

Answer

Context free languages (CFLs) are closed under several common operations.

Union :

1. If $L1$ and $L2$ are two CFG, their union $L1 \cup L2$ will also be context free.

   Example :

   $L1 = \{ a^n b^n c^m \mid m >= 0 \text{ and } n >= 0 \}$ and $L2 = \{ a^n b^m c^m \mid n >= 0 \text{ and } m >= 0 \}$

   $L3 = L1 \cup L2 = \{ a^n b^n c^m \cup a^n b^m c^m \mid n >= 0, m >= 0 \}$ is also context free.

2. $L1$ says number of a's should be equal to number of b's and $L2$ says number of b's should be equal to number of c's.

3. Their union says either of two conditions to be true. So it is also CFG.

Concatenation :

1. If $L1$ and $L2$ are two context free languages, their concatenation $L1.L2$ will also be context free.

   Example :

   $L1 = \{ a^n b^n \mid n >= 0 \}$ and $L2 = \{ c^m d^m \mid m >= 0 \}$

   $L3 = L1.L2 = \{a^n b^n . c^m d^m \mid m >= 0 \text{ and } n >= 0\}$ is also context free.

2. $L1$ says number of a's should be equal to number of b's and $L2$ says number of c's should be equal to number of d's.

3. Their concatenation says first number of a's should be equal to number of b's, then number of c's should be equal to number of d's.

4. So, we can create a PDA which will first push for a's, pop for b's, push for c's then pop for d's. So it can be accepted by pushdown automata, hence context free.

Kleene closure : If $L1$ is context free, its Kleene closure $L1^*$ will also be context free.

Example :

$L1 = \{ a^n b^n \mid n >= 0 \}$

$L1^* = \{ a^n b^n \mid n >= 0 \}^*$ is also context free.

Intersection with a regular language :

1. The intersection of a CFG and a regular language is always context free.

2. To show this, assume we have a PDA M accepting the CFG and a DFA N accepting the regular language.

3. Use the product construction to create a PDA which simulates both machines in parallel. This works because only M needs to manipulate the stack; N never we the stack.

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. Find the regular grammar for the language $L = \{a^n b^m \mid n + m \text{ is even}\}$.

Ans. Refer Q. 3.6.

Q. 2. Consider the grammar with the production. Compute the string *aababbb* with the left most and right most derivation. Draw the derivation tree.

Ans. Refer Q. 3.10.

Scanned by TapScanner

Q. 3. Define ambiguity. Show that the grammar G with following production is ambiguous.
$S \rightarrow a \mid aAb \mid abSb, A \rightarrow aAAb \mid bS$

Ans. Refer Q. 3.15.

Q. 4. Find the language generated by the following grammar :
$S \rightarrow aAb/ab, A \rightarrow bAa, A \rightarrow \varepsilon$

Ans. Refer Q. 3.17.

Q. 5. Find context free grammar for following languages with $(n >= 0, m >= 0, k >= 0)$.

i. $L = \{a^n b^n c^k \mid k >= 3\}$

ii. $L = \{a^m b^n c^k \mid n = m \text{ or } m <= k\}$.

Ans. Refer Q. 3.18.

Q. 6. Discuss inherent ambiguity of context free languages with suitable example. Construct the context free grammar that accepts language $L = \{a^i b^j c^k \mid i = j \text{ or } j = k; i, j, k$ a positive integers$\}$.

Ans. Refer Q. 3.19.

Q. 7. Convert the following grammar in GNF : $S \rightarrow AB, A \rightarrow BS/a, B \rightarrow SA/b$

Ans. Refer Q. 3.32.

Q. 8. Explain the parse tree with an example. Reduce the CFG into GNF whose productions are
$S \rightarrow aSb$
$S \rightarrow ab$

Ans. Refer Q. 3.34.

Q. 9. Convert the CFG into GNF.
$S \rightarrow aSbA$
$A \rightarrow Sa/a$

Ans. Refer Q. 3.35.

Q. 10. Explain the Chomsky hierarchy of languages. Determine the type of the following grammar :
$S \rightarrow aAb/\varepsilon, A \rightarrow aA/Ab/a/b$

Ans. Refer Q. 3.36.

☺☺☺

4 UNIT

Push Down Automata and Properties of Context Free Languages

CONTENTS

Part-1 : Non-Deterministic Pushdown Automata (NPDA) : Definition, Moves, A Language Accepted by NPDA, Deterministic Pushdown Automata (DPDA) and Deterministic Context Free Languages (DCFL) ..... **4–2A to 4–9A**

Part-2 : Pushdown Automata for Context Free Language ..... **4–9A to 4–15A**

Part-3 : Context Free Grammar for Pushdown Automata ..... **4–15A to 4–17A**

Part-4 : Two Stack Pushdown Automata ..... **4–18A to 4–21A**

Part-5 : Pumping Lemma for CFL, Closure Properties of CFL, Decision Problems of CFL, Programming Problems based on the Properties of CFLs ..... **4–22A to 4–27A**

Scanned by TapScanner

PART-1

Non-Deterministic Push Down Automata (NPDA) : Definition Moves, A Language Accepted by NPDA, Deterministic Push Down Automata (DPDA) and Deterministic Context Free Languages (DCFL).

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.1. **What is Push Down Automata (PDA) ? Explain with diagram.**

Answer

1. PDA is the machine format of Context Free Language (CFL).
2. It is one type of finite state machine (FSM) which is used to accept only CFLs.

Diagram :

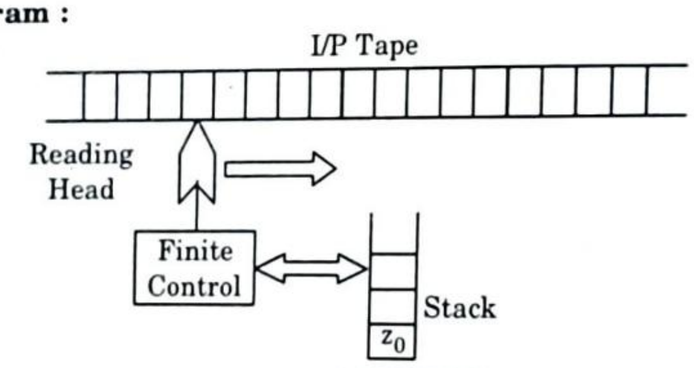


Fig. 4.1.1.

a. Input tape :

i. Input tape contains the input symbols.

ii. The tape is divided into a number of squares. Each square contains a single input character.

iii. The string placed in the input tape is traversed from left to right.

iv. Two end sides of the input string contain infinite number of blank symbol.

b. Reading head :

i. The head scans each square in the input tape and reads the input from the tape.

ii. The head moves from left to right.

iii. The input scanned by the reading head is sent to the finite control of the PDA.

c. Finite control :

i. Finite control can be considered as a control unit of a PDA.

ii. The reading head scans the input from the input tape and sends it to finite control.

iii. Depending on the input take from input tape and input from stack top, finite control decides in which state the PDA will move and which stack symbol it will push to the stack or pop from the stack or do nothing on the stack.

d. Stack :

i. A stack is a temporary storage of stack symbols.

ii. Every move of PDA indicates one of the followings to the stack.

1. One stack symbol may be added to the stack.
2. One stack symbol may be deleted from the top of the stack.

iii. Stack is the component of PDA which differentiates it from finite automata.

iv. In stack there is always a symbol z_0 which denotes the bottom of the stack.

Que 4.2. **Explain definition of push down automata and why it is named push down ?**

Answer

A push down automata is a system, which is mathematically defined as follows :

$$P = (Q, \Sigma, \Gamma, \delta, s, F)$$

Where, Q : non-empty finite set of states

Σ : non-empty finite set of input symbols

Γ : is finite set of pushdown symbols

s : is the initial state, $s \in Q$

F : is the set of final states and $F \subseteq Q$

δ : it is a transition function or transition relation which maps. $(Q \times \Sigma^* \times \Gamma^*) \rightarrow (Q \times \Gamma^*)$

Why push down :

1. Push is an operation related to stack. By this operation one symbol is added to the stack top.
2. In finite automata, states act as a form of primitive memory. States memorize the non-terminals encountered at the time of derivation of a string.

Scanned by TapScanner

3. Hence only state is suitable for traversing a regular expression as in the case of finite automata.

Example :

a. Let's consider a case of $L = \{a^n b^n$, where $n \geq 1\}$. It is not a regular expression but a CFL. Here n is any number.

b. In the string there is equal number of 'a' and 'b'. In the string 'a' will occur before 'b'.

c. In case of traversing $a^n b^n$, the machine has to remember n number of a's to traverse equal number of 'b'. The 'n' is any number.

d. Therefore, to memorize number of a's in the string the machine requires infinite number of states, *i.e.*, the machine will not be finite state machine.

e. To remove this difficulty we need to add an auxiliary memory in the form of stack.

f. For each occurrence of 'a' in the string one symbol is pushed into the stack.

g. For each occurrence of b (after finishing 'a') one symbol will be popped from the stack. By this process the matching of 'a' will be done. By adding a stack to a finite automata, PDA is generated.

Que 4.3. **Design PDA for palindrome strips.**

AKTU 2014-15, Marks 05

Answer

PDA for palindrome :

We define the PDA M as follows :

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

where δ is defined by

$$\delta(q_0, a, Z_0) = \{(q_0, aZ_0), (q_1, Z_0)\}$$
$$\delta(q_0, b, Z_0) = \{(q_0, bZ_0), (q_1, Z_0)\}$$
$$\delta(q_0, a, a) = \{(q_0, aa), (q_1, a)\}$$
$$\delta(q_0, b, a) = \{(q_0, ba), (q_1, a)\}$$
$$\delta(q_0, a, b) = \{(q_0, ab), (q_1, b)\}$$
$$\delta(q_0, b, b) = \{(q_0, bb), (q_1, b)\}$$
$$\delta(q_0, \varepsilon, Z_0) = \{(q_1, Z_0)\}$$
$$\delta(q_0, \varepsilon, a) = \{(q_1, a)\}$$
$$\delta(q_0, \varepsilon, b) = \{(q_1, b)\}$$
$$\delta(q_1, a, a) = \{(q_1, \varepsilon)\}$$
$$\delta(q_1, b, b) = \{(q_1, \varepsilon)\}$$
$$\delta(q_1, \varepsilon, Z_0) = \{(q_2, Z_0)\}$$

Que 4.4. **What are the condition to declare a string accepted by PDA ?**

OR

Define Pushdown Automata (PDA). Differentiate PDA by empty stack and final state by giving their definitions.

AKTU 2015-16, Marks 10

Answer

PDA : Refer Q. 4.1, Page 4–2A, Unit-4.

There are two ways to declare a string accepted by a PDA :

1. **Accepted by empty stack (store) :**

   a. In each PDA, there is a stack attached.

   b. In the stack at the bottom there is a symbol called stack bottom symbol.

   c. In each move of the PDA, one symbol called stack symbol is either pushed in or popped from the stack. But the symbol z_0 still remains in the stack.

   d. A string w may be declared accepted by empty stack after processing all the symbol of w, if the stack is empty after reading the rightmost input character of the string w.

   e. In mathematical notation, we can say $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ be a PDA, the string w is declared accepted by empty stack if
   $\{w \in \Sigma^* / (q_0, w, z_0)^* \rightarrow (q, \varepsilon, \varepsilon)$ for some $q \in Q\}$.

   f. In general, we can say that a string is accepted by a PDA by empty stack if both the following conditions are fulfilled.

      i. The string is finished (Totally traversed)

      ii. The stack is empty.

2. **Accepted by final state :**

   a. A string w may be declared accepted by final state if after total traversal of the string w the PDA enters into its final state.

   b. In mathematical notation, we can say $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ be a PDA, the string w is declared accepted by final state if
   $\{w \in \Sigma^* / (q_0, w, z_0)^* \rightarrow (q_f, \varepsilon, z_0)$ for $q_f \in F$ and z_0 is the stack bottom symbol$\}$.

   c. In general, we can say that a string is accepted by a PDA by final state if both the following conditions are fulfilled.

      i. The string is finished (Totally traversed)

      ii. The PDA enters into its final state.

Scanned by TapScanner

Que 4.5. **Define Push Down Automata (PDA). Design a PDA for the following language :**

$L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$

AKTU 2014-15, Marks 10

Answer

PDA : Refer Q. 4.1, Page 4–2A, Unit-4.

Numerical :

We have PDA M as follows :

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{a, b, c\}, \{z_0, x\}, \delta, q_0, z_0, \{q_1, q_3\})$$

δ is given by :

$$\delta(q_0, \varepsilon, z_0) = \{(q_1, z_0), (q_2, z_0), (q_3, z_0)\}$$
$$\delta(q_1, c, z_0) = (q_1, z_0)$$
$$\delta(q_2, a, z_0) = (q_0, xz_0)$$
$$\delta(q_2, a, x) = (q_2, xx)$$
$$\delta(q_2, b, x) = (q_4, b, x) = (q_4, \varepsilon)$$
$$\delta(q_4, \varepsilon, z_0) = (q_1, z_0)$$
$$\delta(q_3, a, z_0) = (q_3, z_0)$$
$$\delta(q_3, b, z_0) = (q_5, xz_0)$$
$$\delta(q_5, b, x) = (q_5, xx)$$
$$\delta(q_5, c, x) = (q_6, c, x) = (q_6, \varepsilon)$$
$$\delta(q_6, \varepsilon, z_0) = (q_3, \varepsilon) \rightarrow \text{Accepted}$$

Que 4.6. **How many types of PDA are there ? Explain each of them.**

Answer

There are two types of PDA :

1. **Deterministic PDA (DPDA) :**
   a. A PDA is said to be a DPDA if all derivations in the design give only single move.
   b. If a PDA being in a state with a single input and single stack symbol gives a single move, then the PDA is called Deterministic PDA.
2. **Non-deterministic PDA (NPDA) :**
   a. A PDA is called non-deterministic if one of the derivations generates more than one move.
   b. If a PDA being in a state with a single input and single stack symbol gives more than one move for any of its transitional functions, then the PDA is called non-deterministic PDA.

Que 4.7. **Construct a PDA that accepts $L = \{ww^R \mid w = (a + b)^*\}$**

AKTU 2016-17, Marks 10

Answer

From analysis of problem we find that we have to design a PDA which accepts all palindrome of even length. The string (palindrome) must content only a, b and ε the strings accepted by PDA will be like bb, aa, $abba$, $baab$ etc. Now we can define PDA and the δ given by

$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b, \varepsilon\}$
$q_0 = \{q_0\}$
$Z_0 = \{X\}$
$\Gamma = \{X, A, B\}$
$F = \{q_2\}$

$$\delta(q_0, a, X) \rightarrow (q_0, AX)$$
$$\delta(q_0, b, X) \rightarrow (q_0, BX)$$
$$\delta(q_0, a, A) \rightarrow (q_0, AA)$$
$$\delta(q_0, a, A) \rightarrow (q_1, \varepsilon)$$
$$\delta(q_0, a, B) \rightarrow (q_0, AB)$$
$$\delta(q_0, b, A) \rightarrow (q_0, BA)$$
$$\delta(q_0, b, B) \rightarrow (q_0, BB)$$
$$\delta(q_0, b, B) \rightarrow (q_1, \varepsilon)$$
$$\delta(q_1, a, A) \rightarrow (q_1, \varepsilon)$$
$$\delta(q_1, b, B) \rightarrow (q_1, \varepsilon)$$
$$\delta(q_1, \varepsilon, X) \rightarrow (q_2, \varepsilon) \rightarrow \text{Accepted}$$

Que 4.8. **Show that if L is a language of Deterministic PDA (DPDA) and R is regular then $L \cap R$ is a language of DPDA.**

Answer

1. To run the PDA for L and the DFA for R in parallel, we construct a DFA for the intersection of two regular languages by running both their DFAs in parallel and accepting only if both machines accept.
2. Formally, suppose the PDA for L has finite states Q_1, stack alphabet Γ, transition function δ_1, and accepting states F_1, and the DFA for R has similarly Q_2, δ_2, and F_2.
3. Assume for simplicity that the PDA for L reads a input symbol on every step.
4. Then the PDA for $L \cap R$ will have states $Q = Q_1 \times Q_2$ and transition function $\delta : Q \times \varepsilon \times \Gamma \rightarrow Q \times \Gamma$, where the new state is $(\delta_1(q_1, a), \delta_2(q_2, a))$ and we use δ_1 to determine what, if anything, we push on the stack.
5. Our set of accepting states is $F = F_1 \times F_2 \subset Q$. The resulting PDA is deterministic if the original PDA for L is regular, so $L \cap R$ is a CFL or DCFL if L is regular.

Scanned by TapScanner

Que 4.9. **Construct a PDA to accept the language $L = \{a^n b^n c^m d^m$, where $m, n \geq 1\}$ by empty stack and by final state.**

Answer

1. Here number of 'a' number of 'b' are same and number of 'c' and number of 'd' are same state change will occur in transition from 'a' to 'b', 'b' to 'c' and 'c' to 'd'.
2. Upon traversing 'a', stack symbol z_1 will be pushed to the stack. Number of 'b' are same as number of 'a'. Number of z_1 pushed to the stack will be equal to number of 'a', *i.e.*, number of 'b'.
4. For traversal of each 'b' in the input tape one z_1 will be popped from the stack.
5. When first 'c' will be traversed the stack top will be z_0. Upon traversing 'c', stack symbol z_2 will be pushed to the stack. Upon traversing 'd' those z_2's will be popped from the stack.
6. When first 'c' will be traversed the stack top will be z_0. Upon traversal 'c', stack symbol z_2 will be pushed to the stack. Upon traversing 'd' those z_2's will be popped from the stack.

By these processes all the string will be traversed. The PDA for accepting the string $a^n b^n c^m d^m / m, n \geq 1$ will be

$Q = \{q_0, q_1, q_2, q_3, q_f\}$

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

$\Gamma = \{z_0, z_1, z_2\}$

$q_0 = \{q_0\}$

$z_0 = \{z_0\}$

$F = \{q_f\}$

The transition function of constructing PDA will be

$\delta(q_0, a, z_0) = (q_0, z_1 z_0)$

$\delta(q_0, a, z_1) = (q_0, z_1 z_1)$

$\delta(q_0, b, z_1) = (q_1, \varepsilon)$

$\delta(q_1, b, z_1) = (q_1, \varepsilon)$

$\delta(q_1, c, z_0) = (q_2, z_2 z_0)$

$\delta(q_2, c, z_2) = (q_2, z_2 z_2)$

$\delta(q_2, d, z_2) = (q_3, \varepsilon)$

$\delta(q_3, d, z_2) = (q_3, \varepsilon)$

$\delta(q_3, \varepsilon, z_0) = (q_3, \varepsilon)$ accepted by empty stack,

$\delta(q_3, \varepsilon, z_0) = (q_f, z_0)$ accepted by final state.

Que 4.10. **What is difference between deterministic and non-deterministic PDA ? Give example of each.**

Answer

S. No.	Deterministic PDA	Non-deterministic PDA
1.	A deterministic PDA is one for which every string has a unique path.	A non-deterministic PDA is one for which at certain time, we may have to select a particular path among the existing ones.
2.	There is at most one legal sequence of transitions that can be followed for any input.	There can be more than one sequence of transition that can be followed for any input.
3.	DPDA contains one-stack.	Non DPDA has two-stack.
4.	There is only one-stack in DPDA for energy state.	Every state has its own stack.
5.	The example of DPDA is PDA accepting string like wcw^R where 'w' is a string and w^R is reverse of string.	The example of NDPDA is a PDA accepting string like ww^R where w^R is reverse of w.

PART-2

Pushdown Automata for Context Free Language.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.11. **Obtain PDA to accept all strings generated by the language $\{a^n b^m a^n, m, n >= 1\}$.** **AKTU 2015-16, Marks 10**

Answer

Let PDA be $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$.

Here δ is defined as follows :

1. $(q_0, a, \varepsilon) = (q_0, a)$
2. $(q_0, a, a) = (q_0, a)$
3. $(q_0, b, a) = (q_1, a)$
4. $(q_1, b, a) = (q_1, a)$
5. $(q_1, a, a) = (q_2, \varepsilon)$

Scanned by TapScanner

6. $(q_2, a, a) = (q_2, \varepsilon)$
7. $(q_2, \varepsilon, \varepsilon) = (f, \varepsilon)$

$$Q = \{q_0, q_1, q_2, f\}$$
$$\Sigma = \{a, b\}, \Gamma = \{a, b\}$$
$$F = \{f\}$$

Here we are reading all a's first and putting them on the stack. After it all b's are read without making any change on stack. Now again we are reading a's from input tape and one 'a' from stack is popped for every single 'a' of input tape. If all a's are popped and there is no 'a' on input tape then final state is achieved.

Que 4.12. **Construct PDA to accept.**

$$L = \{0^n 1^n : n \geq 0\}$$

AKTU 2016-17, Marks 7.5

Answer

1. By the analysis of language, it is clear that PDA for the given language will accept the set of strings in which every string start with '0' followed by any number of 0's and followed by same number of 1's.
2. The PDA should accept the string like 01, 0011, 00011 ..... and reject ε, 00, 11, 001, 0101, 1100 etc.

Let the PDA

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

Here, consider the stack starting symbol is Z.

The tuples for PDA

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, q_0, \{Z\}, \delta, Z, q_2)$$

i.e.,
$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{0, 1\}$
$\Gamma = \{A, Z\}$
$q_0 = q_0$
$Z_0 = Z$
$F = q_2$

$\delta(q_0, 0, Z) = (q_0, 0Z)$
$\delta(q_0, 0, 0) = (q_0, 00)$
$\delta(q_0, 1, 0) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, 0) = (q_2, \varepsilon) \rightarrow$ Accepted

Que 4.13. **Construct PDA for following :**

$L = \{a^n c\, b^{2n} \mid n >= 1\}$
over alphabet $\Sigma = \{a, b, c\}$. Specify the acceptance state.

Answer

This PDA has input set $\{a, b, c\}$ and the number of b's following total number of a's is double and the a's and b's are separated by single c.

Step 1 : Initially we will read string of a's and push those a's onto the stack.

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, a, a) = (q_0, aa)$$

Step 2 : Now we will read single c. We will not perform any push or pop operation at this step.

$$\delta(q_0, c, a) = (q_1, a)$$

At this step the state is changed to q_1.

Step 3 : Being in state q_1 if we read input symbol b then we will read two consecutive b's without pushing or popping anything, simply read two b's and

$$\delta(q_1, b, a) = (q_2, a)$$
$$\delta(q_2, b, a) = (q_3, a)$$

Then read the third consecutive b and pop single a from the stack. This means checking presence of two b's against single a.

$$\delta(q_3, b, a) = (q_3, \varepsilon)$$

Step 4 : When the input read is a blank character and stack is empty.

Then we reach to accept $\delta(q_3, \varepsilon, z_0) = (q_4, \varepsilon)$.
The state q_4 is accept state.

Que 4.14. **Construct a deterministic PDA for the following language :**

$$L = \{x \in (a, b)^* \mid n_a(x) \neq n_b(x)\}$$

where $n_a(x)$: **number of a's in the string x**
$n_b(x)$: **number of b's in the string x**

Answer

The tuples will be :

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}, \Sigma = \{a, b\}$$
$$\Gamma = \{z_0, x\}, q_0 = \{q_0\}, z_0 = \{z_0\}, F = \{q_1, q_3\}$$
$$\delta(q_0, \varepsilon, z_0) = \{(q_1, z_0), (q_2, z_0), (q_3, z_0)\}$$
$$\delta(q_1, a, z_0) = (q_1, xz_0)$$
$$\delta(q_1, a, x) = (q_1, xx)$$
$$\delta(q_1, b, x) = (q_3, b, x) = (q_3, \varepsilon)$$
$$\delta(q_3, \varepsilon, z_0) = (q_1, z_0)$$
$$\delta(q_2, a, z_0) = (q_2, z_0)$$
$$\delta(q_2, b, z_0) = (q_4, xz_0)$$
$$\delta(q_4, b, x) = (q_4, x\,x) = (q_4, \varepsilon)$$
$$\delta(q_5, \varepsilon, z_0) = (q_2, \varepsilon)$$

Que 4.15. **Design PDA for Language WcW^R, $W \in (a, b)^*$.**

AKTU 2018-19, Marks 10

Scanned by TapScanner

OR

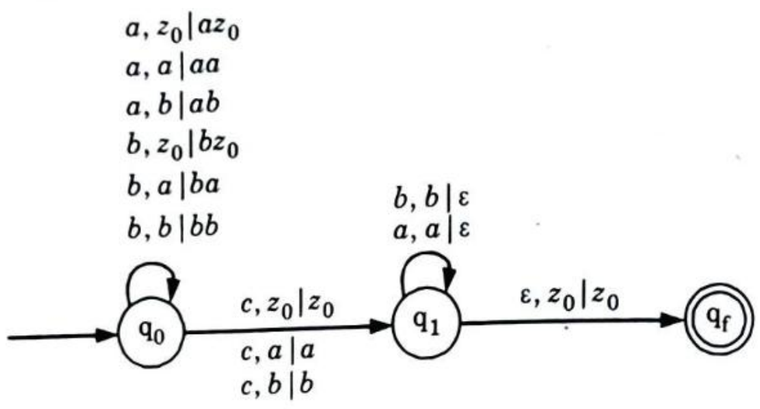
What is Push Down Automata (PDA) ? Design the PDA for the language $L = \{wcw^R \mid w \in \{a, b\}^*\}$ **AKTU 2017-18, Marks 07**

Answer

PDA : Refer Q. 4.1, Page 4–2A, Unit-4.

$L = \{WcW^R \mid W \in \{a, b\}^*\}$

PDA for L is given as :



Que 4.16. **How to convert an equivalent PDA of a context free grammar ?**

Answer

Step I : Convert the PDA into Greibach Normal Form (GNF).

Step II : First the start symbol S of the CFG is put to the stack by transition function

$\delta(q_0, \varepsilon, z_0) \rightarrow (q_1, Sz_0)$

Step III : For a production in the form $\{NT_1\} \rightarrow$ {Single T} {String of NT}, the transitional function will be

$\delta(q_1, T, NT_1) \rightarrow (q_1,$ String of $NT)$.

Step IV : For a production in the form $\{NT_1\} \rightarrow$ {Single T}, the transitional function will be $\delta(q_1, T, NT_1) \rightarrow (q_1, \varepsilon)$.

Step V : For accepting a string two transitional functions are added, one for accepted by empty stack and one for accepted by final state.

Que 4.17. **Convert the following CFG into PDA :**

$S \rightarrow aSa/aA/Bb, \quad A \rightarrow aA/a, \quad B \rightarrow Bb/A$ **AKTU 2014-15, Marks 05**

Answer

The PDA for given CFG will be :

$$P = (Q, \Sigma, \Gamma, \delta, q_o, Z_o, F)$$
$$Q = \{q\}$$
$$\Sigma = \{a, b\}$$
$$\Gamma = \{S, A, B, a, b\}$$
$$q_o = \{q\}$$
$$Z_o = \{S\}$$
$$F = \{q\}$$

The δ will be defined as :

$$\delta(q, \varepsilon, S) = (q, aSa)$$
$$\delta(q, \varepsilon, S) = (q, aA)$$
$$\delta(q, \varepsilon, S) = (q, Bb)$$
$$\delta(q, \varepsilon, A) = (q, aA)$$
$$\delta(q, \varepsilon, A) = (q, a)$$
$$\delta(q, \varepsilon, B) = (q, Bb)$$
$$\delta(q, \varepsilon, B) = (q, A)$$
$$\delta(q, a, a) = (q, \varepsilon)$$
$$\delta(q, b, b) = (q, \varepsilon)$$

Que 4.18. **Construct a PDA from the following CFG.**

$G = (\{S, X\}, \{a, b\}, P, S)$ where the productions are :

$S \rightarrow XS \mid \varepsilon, A \rightarrow aXb \mid Ab \mid ab$ **AKTU 2016-17, Marks 10**

Answer

Given : $G = (\{S, X\}, \{a, b\}, P, S)$

where the productions are

$S \rightarrow XS \mid \varepsilon, A \rightarrow aXb \mid Ab \mid ab$

Let the equivalent PDA,

$$P = (\{q\}, \{a, b\}, \{a, b, X, S\}, \delta, q, S)$$

where $\delta \rightarrow$

$$\delta(q, \varepsilon, S) = \{(q, XS), (q, \varepsilon)\}$$
$$\delta(q, \varepsilon, X) = \{(q, aXb), (q, Xb), (q, ab)\}$$
$$\delta(q, a, a) = \{(q, \varepsilon)\}$$
$$\delta(q, 1, 1) = \{(q, \varepsilon)\}$$

Que 4.19. **Consider the CFG $(\{S, A, B\}, \{a, b\}, P, S)$ where productions P are as follows :**

$S \rightarrow aABB/aAA$, $A \rightarrow aBB/a$, $B \rightarrow bBB/A$. Convert the given grammar to PDA that accept the same language by empty stack.

AKTU 2015-16, Marks 10

Scanned by TapScanner

Answer

The PDA is given by $P(\{q\}, \{a, b\}, \{a, b, S, A, B\}, \delta, q, s,)$ where δ is given by

$$\delta(q, \varepsilon, S) \rightarrow \{(q, aABB), (q, aAA)\}$$
$$\delta(q, \varepsilon, A) \rightarrow \{(q, aBB), (q, a)\}$$
$$\delta(q, \varepsilon, B) \rightarrow \{(q, bBB), (q, A)\}$$
$$\delta(q, a, a) \rightarrow \{(q, \varepsilon)\}$$
$$\delta(q, b, b) \rightarrow \{(q, \varepsilon)\}$$

Que 4.20. **Construct a PDA M equivalent to grammar with following productions :**
$S \rightarrow aAA, A \rightarrow aS \mid bS \mid a$
Also, check whether the string '*abaaaa*' is in *M* or not.

AKTU 2018-19, Marks 10

Answer

Let the PDA for given CFG be
$M = \{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$
$Q = \{q\}$
$\Sigma = \{a, b\}$
$\Gamma = \{S, A, a, b\}$
$q_0 = \{q\}$
$z_0 = \{S\}$
$F = \{q\}$
δ will be defined as :
$\delta(q, \varepsilon, S) = \{(q, aAA)\}$
$\delta(q, \varepsilon, A) = \{(q, aS), (q, bS), (q, a)\}$
$\delta(q, a, a) = \{(q, \varepsilon)\}$
$\delta(q, b, b) = \{(q, \varepsilon)\}$
Check : Given string is '*abaaaa*'
$(q, abaaaa, S)$

$$\vdash (q, abaaaa, aAA) \vdash (q, baaaa, AA)$$
$$\vdash (q, baaaa, bSA) \vdash (q, aaaa, SA)$$
$$\vdash (q, aaaa, aAAA) \vdash (q, aaa, AAA)$$
$$\vdash (q, aaa, aAA) \vdash (q, aa, AA)$$
$$\vdash (q, aa, aA) \vdash (q, a, A)$$
$$\vdash (q, a, a) \vdash (q, \varepsilon, \varepsilon)$$

Thus, the string '*abaaaa*' is in *M*.

Que 4.21. **Construct PDA equivalent to the following CFG production : $S \rightarrow aAA, A \rightarrow aS \mid bS \mid a$**

AKTU 2017-18, Marks 07

Answer

PDA for given CFG will be :

$$P = (Q, \Sigma, T, \delta, q_0, Z_0, F)$$
$$Q = (q), \Sigma = (a, b), T = (S, A, a, b), q_0 = (q)$$
$$Z_0 = (S), F = (q)$$

The δ will be defined as :

$$\delta(q, \varepsilon, S) = (q, aAA)$$
$$\delta(q, \varepsilon, A) = (q, aS)$$
$$\delta(q, \varepsilon, A) = (q, bS)$$
$$\delta(q, \varepsilon, A) = (q, a)$$
$$\delta(q, \varepsilon, a) = (q, \varepsilon)$$
$$\delta(q, \varepsilon, b) = (q, \varepsilon)$$

PART-3

Context Free Grammar for Pushdown Automata.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.22. **Convert the given PDA *M* to equivalent context free grammar. The PDA *M* is defined as $M(\{q_0, q_1\}, \{0, 1\}, \{X, Z_0\}, \delta, q_0, Z_0, \{q_1\})$ where δ is given as follows :**

$$(q_0, 1, Z_0) = (q_0, XZ_0)$$
$$(q_0, 1, X) = (q_0, XX)$$
$$(q_0, 0, X) = (q_0, X)$$
$$(q_0, \varepsilon, X) = (q_1, \varepsilon)$$
$$(q_1, \varepsilon, X) = (q_1, \varepsilon)$$
$$(q_1, 0, X) = (q_1, XX)$$
$$(q_1, 0, Z_0) = (q_1, \varepsilon)$$

Answer

Step 1 : Applying Rule 1 : $S \rightarrow [q_0, Z_0, q_0] \mid [q_0, Z_0, q_1]$

Step 2 : $\delta(q_0, 1, Z_0) = (q_0, XZ_0)$ the production will be

$$[q_0, Z_0, q_0] \rightarrow 1[q_0, X, q_0]\ [q_0, Z_0, q_0]$$
$$[q_0, Z_0, q_0] \rightarrow 1[q_0, X, q_1]\ [q_1, Z_0, q_0]$$
$$[q_0, Z_0, q_1] \rightarrow 1[q_0, X, q_0]\ [q_0, Z_0, q_1]$$
$$[q_0, Z_0, q_1] \rightarrow 1[q_0, X, q_1]\ [q_1, Z_0, q_1]$$

Step 3 : $\delta(q_0, 1, X) = (q_0, XX)$ production will be

$$[q_0, X, q_0] \rightarrow 1[q_0, X, q_0]\ [q_0, X, q_0]$$

Scanned by TapScanner

$[q_0, X, q_0] \to 1[q_0, X, q_1]\ [q_1, X, q_0]$
$[q_0, X, q_1] \to 1[q_0, X, q_0]\ [q_0, X, q_1]$
$[q_0, X, q_1] \to 1[q_0, X, q_1]\ [q_1, X, q_1]$

Step 4 : $\delta(q_0, 0, X) = (q_0, X)$ production will be

$[q_0, X, q_0] \to 0[q_0, X, q_0]$
$[q_0, X, q_0] \to 0[q_0, X, q_1]$
$[q_0, X, q_1] \to 0[q_0, X, q_1]$
$[q_0, X, q_1] \to 0[q_1, X, q_1]$

Step 5 : $(q_0, \varepsilon, X) = (q_1, \varepsilon)$

$[q_0, X, q_1] \to \varepsilon$

Step 6 : $\delta(q_1, \varepsilon, X) = (q_1, \varepsilon)$

$[q_1, X, q_1] \to \varepsilon$

Step 7 : $\delta(q_1, 0, X) = (q_1, XX)$

$[q_1, X, q_0] \to 0[q_1, X, q_0]\ [q_0, X, q_0]$
$[q_1, X, q_0] \to 0[q_1, X, q_1]\ [q_1, X, q_0]$
$[q_1, X, q_1] \to 0[q_1, X, q_0]\ [q_0, X, q_1]$
$[q_1, X, q_1] \to 0[q_1, X, q_1]\ [q_1, X, q_1]$

Step 8 : $\delta(q_1, 0, Z_0) = (q_1, \varepsilon)$

$[q_1, Z_0, q_1] \to \varepsilon$
$[q_0, Z_0, q_0] \to 1[q_0, X, q_1]\ [q_1, Z_0, q_1]/\varepsilon$
$[q_0, X, q_1] \to 1[q_0, X, q_1]\ [q_1, X, q_1]\ /0[q_1, X, q_1]$
$[q_1, X, q_1] \to 1$
$[q_0, Z_0, q_0] \to 0[q_0, Z_0, q_0]$
$[q_1, Z_0, q_1] \to \varepsilon$
$[q_1, X, q_1] \to 0[q_0, X, q_1]\ [q_0, X, q_1]$

Let,
$(q_0, Z_0, q_0) = S$
$(q_0, X, q_1) = A$
$(q_1, Z_0, q_1) = B$
$(q_1, X, q_1) = D$
$(q_1, Z_0, q_1) = E$

CFG for given PDA

$S \to 1AE \mid \varepsilon \mid 0S$
$A \to 1AD \mid 0D$
$D \to 1 \mid 0AD$
$E \to \varepsilon$

Que 4.23. **Consider given PDA :**

$M = (\{q_0\}, \{0, 1\}, \{a, b, z_0\}, \delta, q_0, z_0, \phi)$ δ is defined as :

$\delta(q_0, a, z_0) = \{(q_0, az_0)\}$

$\delta(q_0, 1, z_0) = \{(q_0, bz_0)\}$
$\delta(q_0, 0, z_0) = \{(q_0, aa)\}$
$\delta(q_0, 1, b) = \{(q_0, bb)\}$
$\delta(q_0, 0, b) = \{(q_0, \varepsilon)\}$
$\delta(q_0, 1, a) = \{(q_0, \varepsilon)\}$
$\delta(q_0, \varepsilon, z_0) = \{(q_0, \varepsilon)\}$

Convert given PDA M to corresponding CFG.

Answer

Step 1 :

$S \to [q_0, z_0, q_0]$

Step 2 : For move

$\delta(q_0, a, z_0) = (q_0, az_0)$ we get the following production

$(q_0, z_0, q_0) \to a\,(q_0, z_0, q_0)\,(q_0, a_0, q_0)$

Step 3 : For move

$\delta(q_0, 1, z_0) = (q_0, bz_0)$ we get the following production

$(q_0, z_0, q_0) \to 1\,(q_0, z_0, q_0)\,(q_0, b, q_0)$

Step 4 : For move

$\delta(q_0, 0, z_0) = (q_0, a\,a)$

$(q_0, z_0, q_0) \to 0\,(q_0, a, q)\,(q_0, a, q_0)$

Step 5 : $\delta(q_0, 1, b) = (q_0, bb)$

$(q_0, b, q_0) \to 1\,(q_0, b, q_0)\,(q_0, b, q_0)$

Step 6 : For move

$\delta(q_0, 0, z_0) = (q_0, \varepsilon)$

$(q_0, b, q_0) \to 0$

Let $(q_0, z_0, q_0) = A$
$(q_0, a, z_0) = B$
$(q_0\, b\, z_0) = C$

Step 7 : $(q_0, a, z_0) = 1$

$S \to A$
$A \to aAB \mid 1AC \mid 0BB \mid \varepsilon$
$B \to 1$
$C \to 1\,CC \mid 0$

Step 8 : $(q_0, z_0, z_0) = \varepsilon$

CFG for given PDA

$S \to aSB \mid 1SC \mid 0BB \mid \varepsilon$
$B \to 1$
$C \to 1\,CC \mid 0$

Scanned by TapScanner

PART-4

Two Stack Pushdown Automata.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.24. Explain two-stack PDA with example.

Answer

1. A two-stack push down automaton (Two-stack PDA) is similar to a PDA, but it has two stacks instead of one.
2. In each transition, we must denote the pop and push action on both stacks.
3. A two-stack pushdown automaton to be a six tuple

$$M = (K, \Sigma, \Gamma, \Delta, S, F)$$

where, K is a finite set of states,

Σ is an alphabet (the input symbols),

Γ is an alphabet (the stack symbols),

$S \in K$ is the initial state,

$F \subseteq K$ is the set of final states, and

Δ, the transition relation, is a finite subset of $(K \times (\Sigma \cup \{e\}) \times \Gamma^* \times \Gamma^*) \times (K \times \Gamma^* \times \Gamma^*)$, where the third parameter pops the first stack, the fourth parameter pops the second stack, the fifth parameter pushes a symbol onto the first stack, and the sixth parameter pushes a symbol onto the second stack.

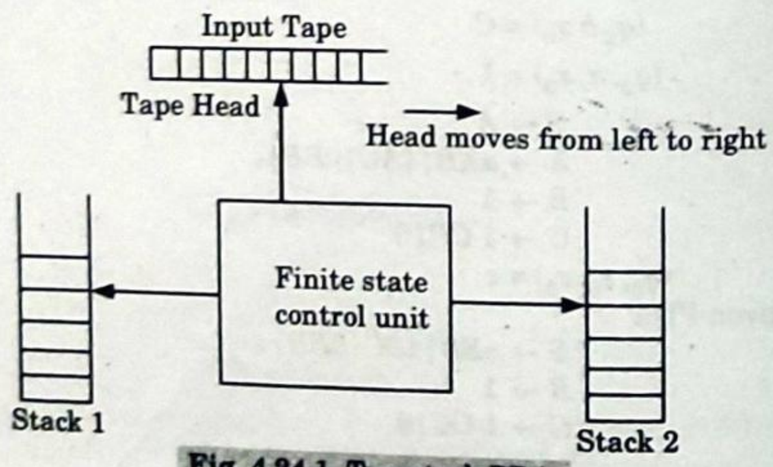


Fig. 4.24.1. Two stack PDA.

4. A move of two-stack PDA includes :
   a. Change state.
   b. The input symbol read.
   c. Replace the symbol of each stack with a string of zero or more stack symbol.
5. There are certain languages which cannot be accepted by pushdown automata as there is only one stack. This is especially when we want to perform two checks on the input string.
6. But if we use two stacks there then those languages can be accepted by PDA.

For example :

1. Let consider a language $L = \{a^n b^n c^n \mid n \geq 1\}$. Now if we use one stack here then the language cannot be accepted by PDA.
2. Suppose with one stack we will push all a's onto the stack on reading them. Then on reading b's we will pop each a. This helps in making a check on equal number of a's and b's.

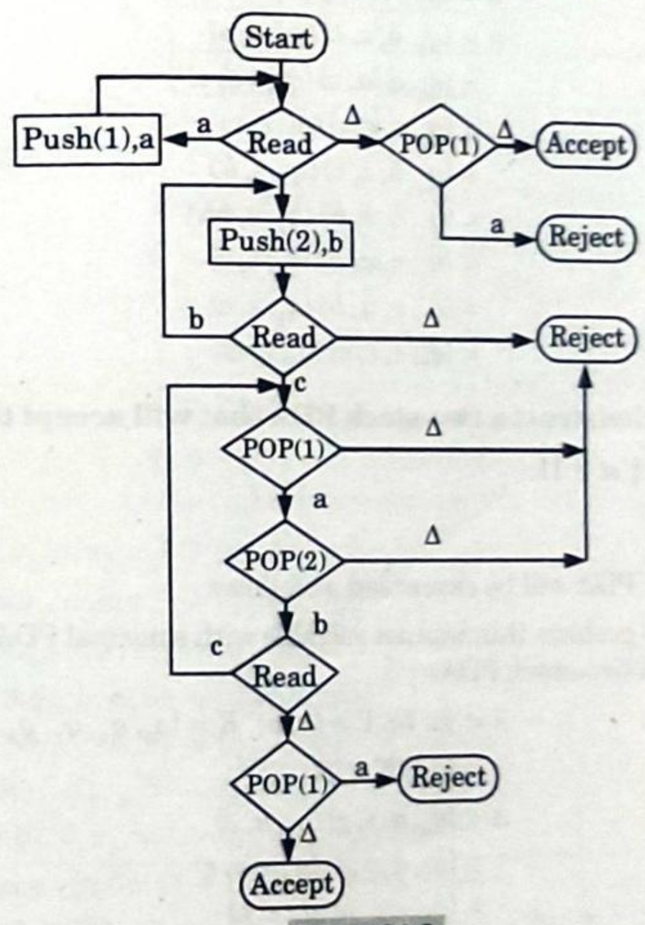


Fig. 4.24.2.

3. **But now when we read out c's the single stack is empty and we cannot make out the equality of a's and b's with c's.**

Scanned by TapScanner

4. Thus it is not possible for us to design a PDA with one stack which accepts language $L = a^n b^n c^n$.
5. But if we use two stacks one for storing all a's. Other for storing all the b's then on reading each c we can pop each a from first stack and each b from second stack. This helps in identifying whether or not there are equal number of a's, b's and c's coming in order.
6. This also means that PDA with two stacks has more power than PDA with one stack.
7. The PDA with two stacks accepting $a^n b^n c^n$ is given in Fig. 4.24.2.

Que 4.25. **Construct a two-stack PDA that will accept the language $L = \{a^n b^n c^n \mid n \in N\}$.**

Answer

The two stack PDA will be expressed as follows :

$$\Sigma = \{a, b, c\}, \Gamma = \{a, b\}, K = \{q_0, q_1, q_2, f\},$$
$$s = \{q_0\}, F = \{f\}$$
$$\Delta = (q_0, a, \varepsilon, \varepsilon)\,(q_0, a, \varepsilon)$$
$$= (q_0, a, a, \varepsilon)\,(q_0, aa, \varepsilon)$$
$$= (q_0, \varepsilon, \varepsilon, \varepsilon)\,(q_1, \varepsilon, \varepsilon)$$
$$= (q_1, b, \varepsilon, \varepsilon)\,(q_1, \varepsilon, b)$$
$$= (q_1, b, \varepsilon, b)\,(q_1, \varepsilon, bb)$$
$$= (q_1, c, a, b)\,(q_2, \varepsilon, \varepsilon)$$
$$= (q_2, c, a, b)\,(q_2, \varepsilon, \varepsilon)$$
$$= (q_2, \varepsilon, \varepsilon, \varepsilon)\,(q_2, \varepsilon, \varepsilon)$$

Que 4.26. **Construct a two-stack PDA that will accept the language $L = \{a^n b^n a^n b^n \mid n \geq 1\}$.**

Answer

The two-stack PDA will be expressed as follows :

This is another problem that was not solvable with a normal PDA, but solvable with a TM and two-stack PDA.

$$\Sigma = \{a, b\}, \Gamma = \{a, b\}, K = \{q_0, q_1, q_2, q_3, f\},$$
$$s = \{q_0\}, F = \{f\}$$
$$\Delta = \{q_0, a, \varepsilon, \varepsilon)\,(q_0, a, \varepsilon)$$
$$= (q_0, a, a, \varepsilon)\,(q_0, aa, \varepsilon)$$
$$= (q_0, \varepsilon, \varepsilon, \varepsilon)\,(q_1, \varepsilon, \varepsilon)$$
$$= (q_1, b, \varepsilon, \varepsilon)\,(q_1, \varepsilon, b)$$
$$= (q_1, b, \varepsilon, b)\,(q_1, \varepsilon, bb)$$
$$= (q_1, a, a, \varepsilon)\,(q_2, \varepsilon, \varepsilon)$$

Que 4.27. **Design a two stack PDA for the language $L = a^n b^m c^n d^m$ where $n, m \geq 1$.**

AKTU 2018-19, Marks 10

OR

Differentiate between deterministic PDA (DPDA) and non-deterministic PDA (NPDA) with suitable example. Also discuss two stack PDA with example.

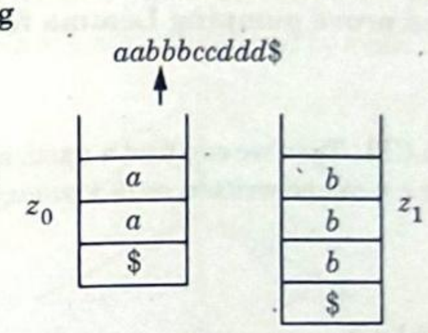
AKTU 2017-18, Marks 07

Answer

$$L = a^n b^m c^n d^m \quad n, m \geq 1$$

The PDA accepts the string like *abcd*, *aabccd*, *aabbccdd*, *aabbbccddd* etc.

Consider a string



Let the PDA :

$$P = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$$
$$Q = \{q_0, q_1, q_2, q_3\}$$
$$\Sigma = \{a, b\}$$
$$\Gamma = \{z\}$$
$$z = \{z_0, z_1\}$$
$$q_0 = \{q_0\}$$
$$F = \{q_f\}$$
$$\delta(q_0, a, z_0, z_1) \rightarrow (q_1, az_0, z_1)$$
$$\delta(q_1, a, a, z_1) \rightarrow (q_1, aa, z_1)$$
$$\delta(q_1, b, a, z_1) \rightarrow (q_2, aa, bz_1)$$
$$\delta(q_2, b, a, b) \rightarrow (q_2, aa, bb)$$
$$\delta(q_2, c, a, b) \rightarrow (q_3, \varepsilon, bb)$$
$$\delta(q_3, d, z_1, b) \rightarrow (q_f, \varepsilon, \varepsilon)$$
$$\delta(q_3, \$, z_0, z_1) \rightarrow (q_f, \varepsilon, \varepsilon) \rightarrow \text{Accepted}$$

Difference : Refer Q. 4.10, Page 4–8A, Unit-4.

Two stack PDA : Refer Q. 4.24, Page 4–18A, Unit-4.

Scanned by TapScanner

PART-5

Pumping Lemma for CFL, Closure Properties of CFL, Decision Problems of CFL, Programming Problems based on the Properties of CFLs.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.28. **State and prove pumping Lemma for CFL.**

Answer

Statement : Let L be a CFL. Then we can find a natural number n such that

a. Every $z \in L$ with $z \geq n$ can be written as $w = uvwxy$, for some strings u, v, w, x, y.

b. $|vx| \geq 1$

c. $|vwx| \leq n$.

d. $uv^kwx^ky \in L$ for all $k \geq 0$.

Proof :

1. We can prove that if G is a context free grammar, then we can find a context free grammar G_1 having no null productions such that $L(G_1) = L(G) - \{\varepsilon\}$.
2. If null string (Λ or ε) belongs to the language set L, we will consider $L - \{\varepsilon\}$ and will construct CNF of the grammar G generating $L - \{\varepsilon\}$ [If null string does not belong to L, we will construct CNF of G generating L].
3. Let number of non-terminals (V_N) of the grammar is m. $|V_N| = m$ and $n = 2^m$. To prove that n is the required number, we start with a string z where $z \in L$, $|z| \geq 2^m$.
4. Construct a derivation tree or parse tree T of the string z.
5. If the length of the longest path in T is almost m, then we can write the length of the yield of T, *i.e.*, $|z| \leq 2^{m-1}$. But $|z| \geq 2^m$. So T has a path, say Γ of length greater than or equal to $m + 1$.
6. Path length of greater than or equal to in $m + 1$ needs at least $m + 2$ vertices, where the last vertex is the leaf.
7. Thus in Γ all the labels except the last one are variables. As $|N| = m$, in $m + 1$ labels of the parse tree T, some labels are repeated.
8. Let v_1 and v_2 be the vertices with repeated label (let A) obtained at the time of upward traversal from leaf of Γ.

9. Between v_1 and v_2, let v_1 is nearer to root. In Γ, the portion of the path from v_1 to leaf has only one label repeated (let A), so its length is almost $m + 1$.
10. Let T_1 and T_2 are two subtrees taking root as v_1 and v_2 respectively. Let z_1 and w are the yields of T_1 and T_2, respectively.
11. As Γ is the longest path in T, v_1 to leaf is the longest path of T_1 and its length is almost in $m + 1$. So the length of z_1 (yield of T_1), *i.e.*, $|z_1| \leq 2^m$.
12. As z is the yield of T and T_1 is a proper subtree of T. Yield of T_1, is z_1. So yield of T, *i.e.*, z can be written as $z = uz_1y$.

    As z_1 is the yield of T_1, and T_2 is a proper subtree of T_1. Yield of T_2 is w. So yield of T_1, *i.e.*, z_1 can be written as $z_1 = vwx$.
13. T_2 is a proper subtree of T_1, so $|vwx| > |w|$. So $|vx| \geq 1$.
14. Thus we can write $z = uvwxy$ with $|vwx| \leq n$ and $|vx| \geq 1$.

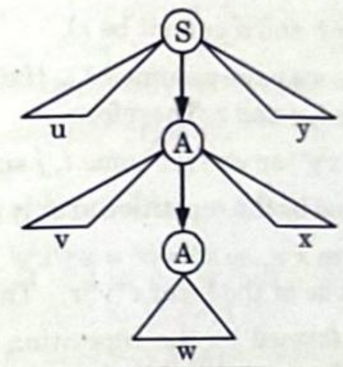


Fig. 4.28.1.

15. As T is tree with root S(S tree) and T_1, T_2 are tree with root B(B tree), we can write $S \overset{*}{\Rightarrow} uAy$, $A \overset{*}{\Rightarrow} vAx$ and, $A \overset{*}{\Rightarrow} w$.
16. As $S \overset{*}{\Rightarrow} uAy \Rightarrow uwy$, $uv^0wx^0y \in L$. For $k \geq 1$, $S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uv^kAx^ky \overset{*}{\Rightarrow} uv^kwx^ky \in L$. By this the theorem is proved.

Que 4.29. **Prove that $L = \{0^p$ / where p is prime$\}$ is not a CFL.**

Answer

Step I : Suppose $L = L(G)$ is context free. Let n be a natural number obtained from the pumping lemma for CFL.

Step II : Let p is a number $> n$, $z = 0^p \in L$. By using pumping lemma for CFL we can write $z = uvwxy$, where $|vr| \geq 1$ and $|vwx| \leq n$.

Step III : Let $k = 0$. From the pumping lemma for CFL, we can write uv^0wx^0y, *i.e.*, $uwy \in L$. As uvy is in the form 0^p, where p is prime; so $|uwy|$ is also a prime number. Let take it as q. Let

Scanned by TapScanner

$|vx| = r$. Then $|uv^qwx^qy| = q + qr = q(1 + r)$. This is not prime as it has factors q, $(1 + r)$ including 1 and $q(1 + r)$. So $uv^qwx^qy \notin L$. This is a contradiction. Therefore L is not context free.

Que 4.30. **Find out whether the language $L = \{x^n y^n z^n \mid n \geq 1\}$ is context free or not.**

AKTU 2016-17, Marks 05

Answer

The language L says that 'x' cannot appear after 'y' and 'z' cannot appear before 'y' and so on.

Step 1 : Assume L is context free and we get a contradiction. Let n be the natural number obtained by using pumping lemma.

Step 2 : Let $Z = x^n y^n z^n$, then $|Z| = 3n > n$. We can write $Z = abcde$, where $|bd| \geq 1$ (*i.e.*, both b and d cannot be ε).

Step 3 : $abcde = x^n y^n z^n$. As we have assumed $1 \leq |bd| \leq n$, then b or d cannot contain all three symbols x, y and z. Therefore,

i. b or d is of the form $x^i y^j$ (or $y^i z^j$) for some i, j such that $i + j \leq n$, or

ii. b or d is a string formed by the repetition of only one symbol among x, y, z.

When b or d is of the form $x^i y^j$, so $x^2 = b^2 = x^i y^j x^i y^j$. As b^2 is a sub-string of ab^2cd^2e so ab^2cd^2e cannot be of the form $x^m y^m z^m$. Therefore $ab^2cd^2e \notin L$.

When both b and d are formed by the repetition of a single symbol, for example $d = x^i$ and $b = y^j$ for some i and j, where $i, j \leq n$ then the string uwy will contain the remaining symbols, say e. Also, e^n will be a sub-string of ace as d does not occur in b or d. The number of occurrence of one of the other two symbols in ace is less than n, and n is the number of occurrence of e. Therefore $ab^2cd^2e = ace \notin L$.

Thus, for any choice of b or d we get a contradiction. Hence, L is not context free language.

Que 4.31. **Given context free grammar, how do you determine that grammar as :**

i. Empty or non-empty

ii. Finite or non-finite

iii. Whether a string X belong to language of grammar.

Answer

i. **Empty or non-empty :** Prove that for a given CFG, there is an algorithm to determine whether or not it can generate any word or not, or it is empty or not.

Proof :

1. We already know about nullable non-terminals, and how to decide whether the start symbol S is nullable :

$$S \overset{*}{\Rightarrow} \varepsilon ?$$

2. We also know that every CFG that does not generate ε can be convert ε-production.
3. Now let us assume that ε is not a word generated by CFG.
4. In this case we can convert the CFG to CNF, preserving the entire language.
5. If there is a production of the form

$$S \rightarrow a$$

where a is terminal, then a is word in the language if there is no such production, we then propose the following algorithm :

Step 1 :

a. For each non-terminal N that has some productions of the form

$$V_n \rightarrow a$$

where a is terminal or string of terminals, we choose one of them and put a in all other productions having V_n at right side, this eliminating non-terminal V_n altogether.

b. We have changed the grammar so that it no longer accepts the same language.

c. It may no longer be in CNF.

d. That is fine with us.

e. Every word that can be generated from the new grammar could have been generated by the old CFG.

f. If old CFG generated any words, then the new one does also.

Step 2 :

Repeat step 1 until it eliminates S or it eliminates no new non-terminals. If S has been eliminated, then the CFG produces some words, if not then not.

ii. **Finite or non-finite :**

There is an algorithm to decide whether a given CFG generates an infinite language or finite language.

Proof :

1. We assume that grammar contains no ε-productions (if there is any useless or ε-production in the grammar then we have to remove those productions) or no unit-production, and no useless symbols.
2. Now, let us assume that grammar has production like,

$$X \overset{*}{\Rightarrow} xXy$$

3. Since, grammar is assumed to have no ε-production and no unit production x and y cannot both be simultaneously empty.
4. Since X is neither nullable nor a useless symbol, we have :

$$S \overset{*}{\Rightarrow} uXv \overset{*}{\Rightarrow} w$$

Scanned by TapScanner

$$X \overset{*}{\Rightarrow} z$$

and

where u, v and z are in V^*_t. But then

$$S = \overset{*}{\Rightarrow} uXv \overset{*}{\Rightarrow} ux^n Xy^n v \overset{*}{\Rightarrow} ux^n zy^n v$$

is possible for all n, so that $L(G)$ is infinite.

5. If no variable can ever repeat, then the length of any derivation is bounded by $|V_n|$. In that case, $L(G)$ is finite.
6. Thus, to get an algorithm for determining whether $L(G)$ is finite, we need only to determine whether the grammar has some repeating variables.
7. This can be done simply by drawing a dependency graph for the variables in such a way that there is an edge (X, Y) whenever there is a corresponding production

$$X \rightarrow xYy$$

8. Then any variable which is at the base of a cycle is a repeating one.
9. Consequently, the grammar has a repeating variable if and only if the dependency graph has a cycle.
10. Since, we have an algorithm for deciding whether a grammar has a repeating variable, we have an algorithm for determining whether or not $L(G)$ is infinite.

iii. CYK algorithm is used to determine whether a string X belong to language of grammar.

The CYK algorithm : Let the input be a string S consisting of n characters : $a_1 \ldots a_n$. Let the grammar contain r non-terminal symbols $R_1 \ldots R_r$. This grammar contains the subset R_s which is the set of start symbols. Let $P[n, n, r]$ be an array of booleans. Initialize all elements of P to false.

For each $i = 1$ to n

For each unit production $R_j \rightarrow a_i$, set $P[i, l, j]$ = true.

For each $i = 2$ to n - - Length of span

For each $j = 1$ to $n - i + 1$ - - Start of span

For each $k = 1$ to $i - 1$ - - Partition of span

For each production $R_A \rightarrow R_B R_C$

If $P[j, k, B]$ and $P[j + k, i - k, C]$ then set $P[j, i, A]$ = true

If any of $P[1, n, x]$ is true (x is iterated over the set s, where s are all the indices for R_s).

Then S is a member of language

Else S is not member of language.

Que 4.32. **Discuss the decidability property of CFG.**

OR

Given a context free grammar G. Write an algorithm (if it exist) to determine whether $L(G)$ is infinite or not.

Answer

Decidability property : Following are the decidability properties :

i. **Decide whether grammar is empty or non-empty :** Refer Q. 4.31, Page 4–24A, Unit-4.

ii. **Decide whether grammar is finite or non-finite :** Refer Q. 4.31, Page 4–24A, Unit-4.

iii. **Decide whether a string X belong to language of grammar :** Refer Q. 4.31, Page 4–24A, Unit-4.

Algorithm :

1. Make the grammar simplified by removing unit productions and useless symbols.
2. Check whether there is any self-embedded non-terminal or not by following steps :
   a. Underline some non-terminal say X which is at L.H.S of production rule.
   b. Then put dot over all the X which are at R.H.S throughout the grammar. The dotted X and underlined X are treated as two different symbols.
   c. Put dot over any non-terminal at L.H.S whose R.H.S contains any dotted symbols. Dot this non-terminal throughout the grammar.
   d. If underlined X gets dotted then X is called self-embedded otherwise it is not self-embedded.
3. If the grammar contains any self-embedded non-terminal then it generates infinite languages otherwise it generates finite languages.

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. Define Pushdown Automata (PDA). Differentiate PDA by empty stack and final state by giving their definitions.

Ans. Refer Q. 4.4.

Scanned by TapScanner

Q. 2. Define Push Down Automata (PDA). Design a PDA for the following language :

$$L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$$

Ans. Refer Q. 4.5.

Q. 3. Obtain PDA to accept all strings generated by the language $\{a^m b^m a^n, m, n >= 1\}$.

Ans. Refer Q. 4.11.

Q. 4. Convert the following CFG into PDA :

$S \rightarrow aSa/aA/Bb, \quad A \rightarrow aA/a, \quad B \rightarrow Bb/A$

Ans. Refer Q. 4.17.

Q. 5. Construct a PDA from the following CFG.

$G = (\{S, X\} \{a, b\}, P, S)$ where the productions are :

$S \rightarrow XS \mid \varepsilon, A \rightarrow aXb \mid Ab \mid ab$

Ans. Refer Q. 4.18.

Q. 6. Consider the CFG $(\{S, A, B\} \{a, b\}, \{P, S\})$ where productions P are as follows :

$S \rightarrow aABB/aAA$, $A \rightarrow aBB/a$, $B \rightarrow bBB/A$. Convert the given grammar to PDA that accept the same language by empty stack.

Ans. Refer Q. 4.19.

Q. 7. Construct a PDA M equivalent to grammar with following productions :

$S \rightarrow aAA, A \rightarrow aS \mid bS \mid a$

Also, check whether the string '*abaaaa*' is in M or not.

Ans. Refer Q. 4.20.

Q. 8. Design a two stack PDA for the language $L = a^n b^m c^n d^m$ where $n, m \geq 1$.

Ans. Refer Q. 4.27.

Q. 9. State and prove pumping Lemma for CFL.

Ans. Refer Q. 4.28.

☺☺☺

5 UNIT

Turing Machines and Recursive Function Theory

CONTENTS

Part-1 :	Basic Turing Machine Model, Representation of Turing Machines, Language Acceptability of Turing Machines	**5–2A to 5–7A**
Part-2 :	Techniques for Turing Machine Construction	**5–7A to 5–10A**
Part-3 :	Modification of Turing Machine, Turing Machine as Computer of Integer Functions	**5–10A to 5–16A**
Part-4 :	Universal Turing Machine, Linear Bounded Automata, Church Thesis, Recursive and Recursively Enumerable Language, Halting Problem	**5–16A to 5–24A**
Part-5 :	Post's Correspondence Problem Introduction to Recursive Function Theory	**5–24A to 5–34A**

Scanned by TapScanner

PART-1

Basic Turing Machine Model, Representation of Turing Machines, Language Acceptability of Turing Machines.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

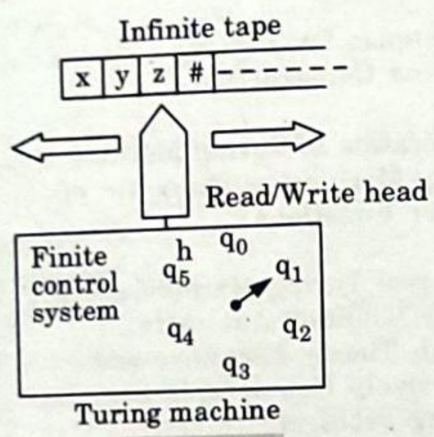
Que 5.1. **Explain Turing machine with block diagram.**

OR

Explain the basic model of a Turing machine.

Answer

TM act as a physical computing device that can be represented as a diagram as shown in Fig. 5.1.1.



Turing machine

Fig. 5.1.1.

TM consists of following component :

1. **Tape :**
   a. A tape has an end on the left but is infinite on the right side.
   b. The tape is divided into squares or cells, with each cell capable of holding one of the tape symbols including the blank symbol #.
   c. At any time, there can be finitely many cells of the tape that can contain non-blank symbols. The set of tape symbols is denoted by Γ.
   d. A finite sequence of input symbols which is placed in the left-most cells of the tape.
2. **Finite control :**
   a. A finite control is in any one of the finite number of states.
   b. The states in TM can be divided in three categories that is :

   i. The initial state, the state of the control just at the time when TM starts its operations. The initial state is denoted by q_0.
   ii. The halt state, which is the state in which TM stops all further operations. The halt state is generally denoted by 'h'.
   iii. Other states are considered as intermediate states.
3. **Tape head :**
   a. With the help of tape head, machine can read and write on the tape.
   b. It moves one cell left or right at each step.
   c. These directions are denoted by L and R.
4. **Action :**
   a. The course of action is called a move of the Turing machine.
   b. The move is a function of current state of the finite control and the tape symbol being scanned.
   c. Move of the turing machine will change the state, write a tape symbol in the cell scanned and move the tape head left or right.

Que 5.2. **Explain the formal definition of turing machine.**

Answer

TM is defined by using 7-tuple

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

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 a subset of Γ.

δ : The transition function. The arguments of $\delta(q, X)$ are a state q and a tape symbol X.

q_0 : The start state, a member of Q, in which the finite control is found initially.

B : The blank symbol. This symbol is in Γ but not in Σ; *i.e.*, it is not an input symbol. The blank appears initially in all but the finite number of initial cells that hold input symbols.

F : The set of final or accepting states, a subset of Q.

Que 5.3. **Write a short note on the following :**

i. **Instantaneous description**
ii. **Transition diagram**
iii. **Role of Turing machine**

Answer

i. **Instantaneous description :**
   1. Instantaneous description (ID) describes the current situation of the TM.

Scanned by TapScanner

2. Instantaneous Description (ID) in the case of a TM (Turing Machine) is the information in respect of :
   a. Contents of all the cells of the tape, starting from the right most cell upto at least the last cell, containing a non-blank symbol and containing all cells upto the cell being scanned.
   b. The cell currently being scanned by the machine and
   c. The state of the machine.
3. A snapshot of Turing machine is shown in Fig. 5.3.1.

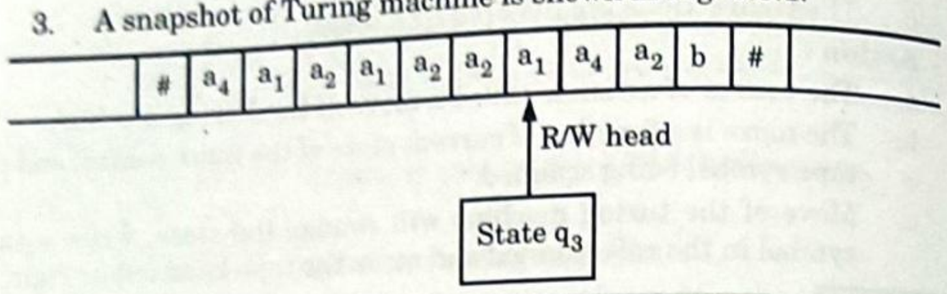


Fig. 5.3.1. A snapshot of turing machine.

4. The present symbol under the R/W head is a_1. The present state is q_3. So a_1 is written to the right of q_3, the non-blank symbol to the left of a_1 from the string $a_4a_1a_2a_1a_2a_2$, which is written to the left of q_3.
5. The sequence of non-blank symbols to the right of a_1 is a_4a_2. Thus the ID is as given in Fig. 5.3.2.

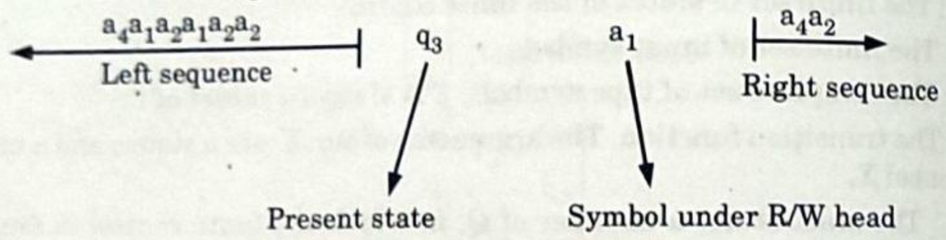


Fig. 5.3.2. Representation of ID.

ii. Transition diagram :

1. A transition diagram of the next-move functions δ of a TM is a graphical representation consisting of a finite number of nodes and directed-labeled arcs between the nodes.
2. Each node represents a state of the TM and a label on an arc from one state (say q_1) to another state (q_2) represents the information about the required input symbol say 'a' for the transition from q_1 to q_2 to take place and the action on the part of the control of TM.
3. The action part consists of :
   a. The symbol say 'b' to be written in the current cell.
   b. The movement of the tape head.
4. The label of an arc is generally written as $a/(b, m)$, where m is L, R or N.

For example :

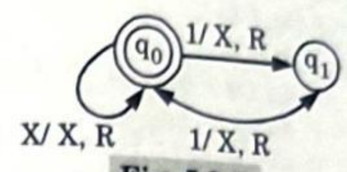


Fig. 5.3.3.

iii. Role of turing machine :

1. **As accepting device for language :** This is similar to the role played by FAs and PDAs.
2. **As a computer function :** In this role, 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 strings :** In this role it outputs the strings of a language, one at a time, in some systematic order that is as a list.

Que 5.4. **Construct a Turing machine which accepts the regular expression, $L = \{0^n 1^n \mid n >= 1\}$.** **AKTU 2015-16, Marks 15**

Answer

To design a Turing machine which accepts a set of strings in which every string starts with '0', followed by any number of 0's, every string ends in equal number of 1's and 0's. No '0' is encountered after first '1' is read. It will not accept the strings like ε, 0, 00, 11, 011, 00111, 001 and so on.

Let required Turing machine is :

$$T_m = \{Q, \Sigma, \Gamma, \delta, q_0, h\}$$
$$Q = \{q_0, q_1, q_2, q_3, h\}$$
$$\Sigma = \{0, 1\}$$
$$\Gamma = \{0, 1, \#\}$$

q_0 is initial state and h is halt state.

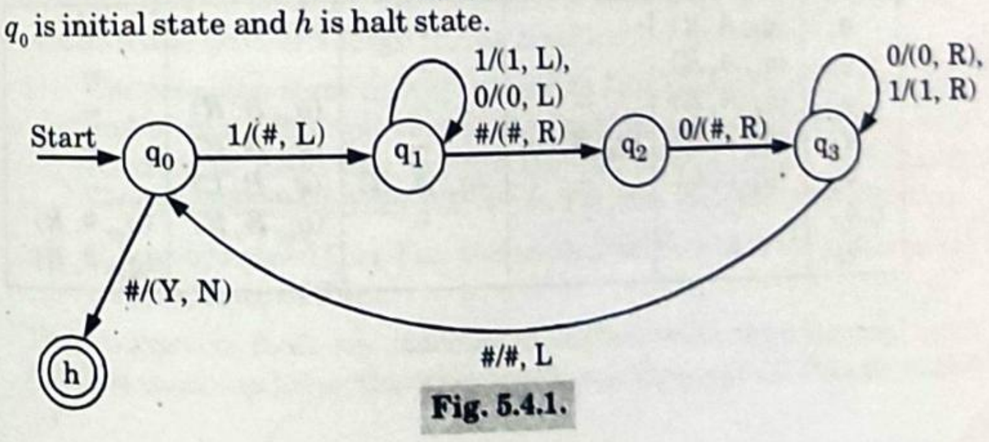


Fig. 5.4.1.

Scanned by TapScanner

The moves which are unaddressed, leads towards rejecting states.

Let us pass the string $w = \# 0q_0 1 \#$

$$\#01q_0 \vdash \#0q_1\# \vdash \#q_1 0\# \vdash \#0q_2\# \vdash \#\#\#q_3 \vdash \#\#q_0\# \vdash \#Yh\#$$

Que 5.5. **Design Turing machine for the language**

$$L = \{a^{n+2} b^n \mid n > 0\}.$$

AKTU 2014-15, Marks 10

Answer

For language $L = \{a^{n+2} b^n \mid n > 0\}$,

a. First two 'a' symbols will be simply replaced by 'A' symbol.

b. Now, if the leftmost symbol in the given input string w is 'a' then replace it by 'A' and move right till we encounter a leftmost 'b' in w. Change it to 'B' and move backwards.

c. Repeat (b) with the leftmost 'a'. If we move back and forth and no a or b remains, move to a final state.

d. Strings which are not in the form $a^{n+2}b^n$ do not reach final state.

We construct the Turing machine M as follows :

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

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$$

$$F = \{q_6\}$$

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

$$\Gamma = \{a, b, A, B, \#\}$$

Transition diagram for this Turing machine is given in Fig. 5.5.1.

The transition table will be :

Table 5.5.1.

	a	b	A	B	#
q_0	(q_1, A, R)	–	–	–	–
q_1	(q_2, A, R)	–	–	–	–
q_2	(q_3, A, R)	–	–	(q_5, B, R)	–
q_3	(q_3, a, R)	(q_4, B, L)	–	(q_3, B, R)	–
q_4	(q_4, a, L)	–	(q_2, A, R)	(q_4, B, L)	–
q_5	–	–	–	(q_5, B, R)	$(q_6, \#, R)$
q_6	–	–	–	–	$(q_6, \#, R)$

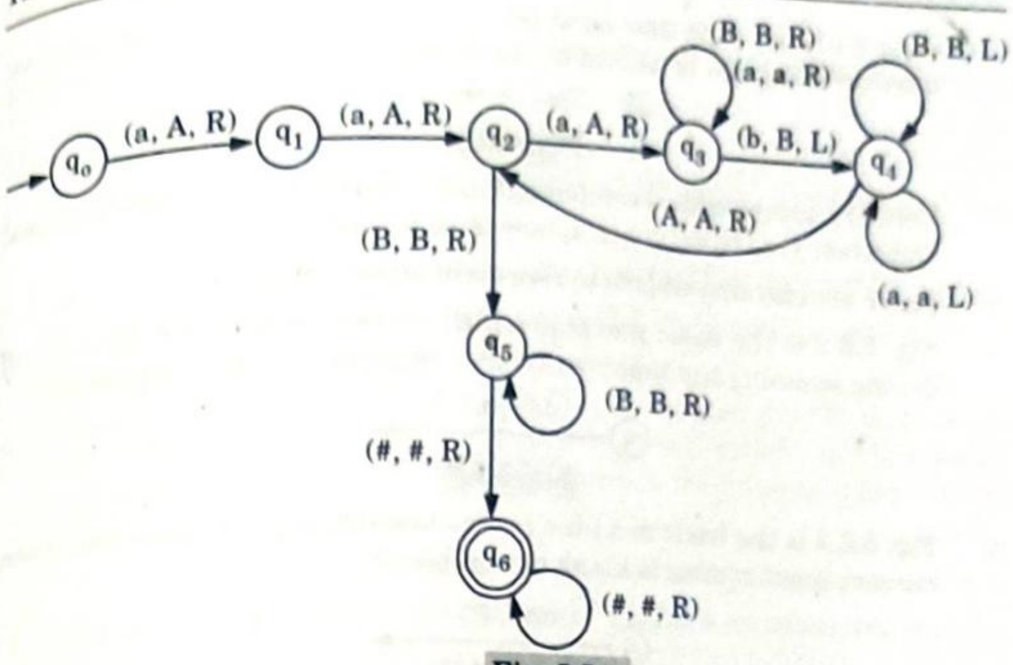


Fig. 5.5.1.

PART-2

Techniques for Turing Machine Construction.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.6. **How to construct complex turing machine ?**

Answer

Complex turing machine can be constructed by using simple turing machines.

1. Suppose Tm_1 is a turing machine that is not known to hang. Then Tm_1 can be made part of a large turing machine Tm as follows :
   i. Tm prepares some string as input to Tm_1, placing it near the right end of non-blank portion of the tape, passes control to Tm_1 with read/write head just beyond the end of that string and finally retrieves control from Tm_1 when Tm_1 has finished its computing.
   ii. It is guaranteed that Tm_1 will never interfere with the operational computation of Tm.
2. The connection from one machine to another will not be pursued until the first machine halts; the other machine is then started from its initial

Scanned by TapScanner

state with the tape and head position as they were left by the first machine. This can be shown by the following figure :

$$> \; Tm_1 \longrightarrow Tm_2$$

Fig. 5.6.1.

Here Tm_1 machine start computing (that is initial state in finite automata) and when Tm_1 halts, control, now goes to machine Tm_2.

3. There are certain symbols to represent simple machines.

i. Fig. 5.6.2 is the basic machine which moves towards left by one cell during scanning any input (σ) symbol. State change is also possible.

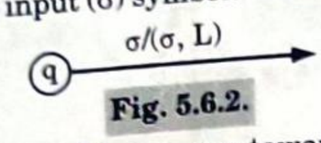


Fig. 5.6.2.

ii. Fig. 5.6.3 is the basic machine moves towards right by one cell, when current input symbol is blank or non-blank (σ).

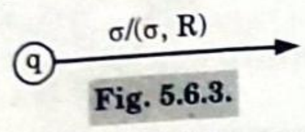


Fig. 5.6.3.

iii. Fig. 5.6.4 is the basic machine, which moves towards left till the first blank symbol is encountered.

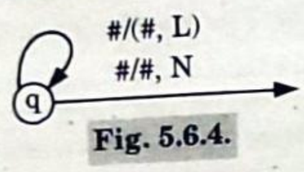


Fig. 5.6.4.

iv. Fig. 5.6.5 is the basic machine, which moves towards right until the first blank symbol is encountered.

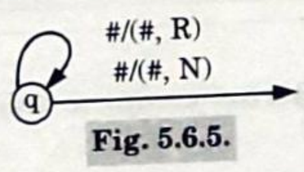


Fig. 5.6.5.

v. Fig. 5.6.6 is the basic machine, which moves towards left until the first non-blank symbol is encountered.

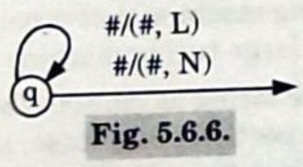


Fig. 5.6.6.

vi. Fig. 5.6.7 is the basic machine, which moves towards right until the first non-blank symbol is encountered.

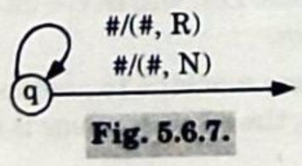


Fig. 5.6.7.

Que 5.7. **Explain techniques for turing machine construction.**

Answer

1. In the definition of a *TM* we defined $\delta(q, a)$ as (q', y, D) where $D = L$ or R.
2. So the head moves to the left or right after reading an input symbol.
3. Suppose, we want to include the option that the head can continue to be in the same cell for some input symbol.
4. Then we define $\delta(q, a)$ as (q', y, S). This means that the *TM*, on reading the input symbol a, changes the state to q' and writes y in the current cell in place of a and continues to remain in the same cell. In terms of IDs,

$$wqax \vdash wq'yx$$

This move can be simulated by the standard *TM* with two moves, namely

$$wqax \vdash wyq''x \vdash wq'yx$$

That is, $\delta(q, a) = (q', y, S)$ is replaced by $\delta(q, a) = (q'', y, R)$ and $\delta(q'', X)$ (q'', y, L) for any tape symbol X.

Thus in this model $\delta(q, a) = (q', y, D)$ where $D = L, R$ or S.

Storage in the state :

1. We are using a state, whether it is of a *FA* or PDA or *TM*, to 'remember' things.
2. We can use a state to store a symbol as well. So the state becomes a pair (q, a) where q is the state (in the usual sense) and a is the tape symbol stored in (q, a).
3. So the new set of states becomes $Q \times \Gamma$.

Multiple track turing machine :

1. In the case of TM, a single tape was used.
2. In a multiple track TM, a single tape is assumed to be divided into several tracks.
3. Now the tape alphabet is required to consist of k-tuples of tape symbols, k being the number of tracks.
4. Hence the only difference between the standard TM and the TM with multiple tracks is the set of tape symbols. In the case of the standard Turing machine, tape symbols are elements of Γ; in the case of TM with multiple tracks it is Γ^k. The moves are defined in a similar way.

Subroutines :

1. Subroutines are used in computer languages, when some task has to be done repeatedly.
2. We can implement subroutines using Turing machine.

Scanned by TapScanner

3. First, a TM program for the subroutine is written. This will have an initial state and a 'return' state.
4. After reaching the return state, there is a temporary halt. For using a subroutine, new states are introduced.
5. When there is a need for calling the subroutine, moves are effected to enter the initial state for the subroutine (when the return state of the subroutine is reached) and to return to the main program of TM.

PART-3

Modification of Turing Machine, Turing Machine as Computer of Integer Fucntions.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.8. **Write a short note on multitape Turing machines.**

Answer

1. A Turing machine with several input tapes is said to be a multitape Turing machine.
2. In a multitape Turing machine each tape is controlled by its own independent read/write head.
3. A multitape Turing machine with k tapes is shown in Fig. 5.8.1.

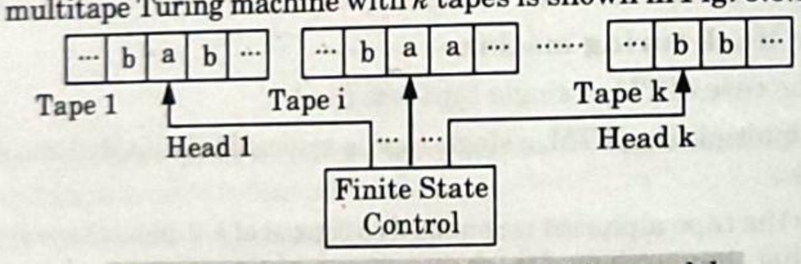


Fig. 5.8.1. A multitape Turing machine model.

4. An n-tape Turing machine can be defined as $M = (Q, \Sigma, \delta, \Gamma, q_0, \#, H)$, where, all the parameters have the same meanings as in the basic definition of a Turing machine, but the transition function δ is defined as
$$\delta : Q \times \Gamma^n \to Q \times \Gamma^n \times \{L, R, \text{stop}\}^n$$
5. To show the equivalence between a multitape and a standard Turing machine, we argue that a standard Turing machine A can be simulated by any given multitape Turing machine M, and conversely, that any standard Turing machine can simulate a multitape Turing machine.

Que 5.9. **Write a short note on multidimensional Turing machines.**

Answer

1. A Turing machine is said to be a multidimensional Turing machine if its input tape can be viewed as extending infinitely in more than one dimension.
2. The formal definition of a two-dimensional Turing machine involves a transition function δ of the form
$$\delta : Q \times \Gamma \to Q \times \Gamma \{L, R, U, D, \text{stop}\}$$
3. All symbols have their usual meanings except U and D. The symbols U and D specify movement of read/write head in up and down directions, respectively. A diagram of a two-dimensional Turing machine is given by Fig. 5.9.1.

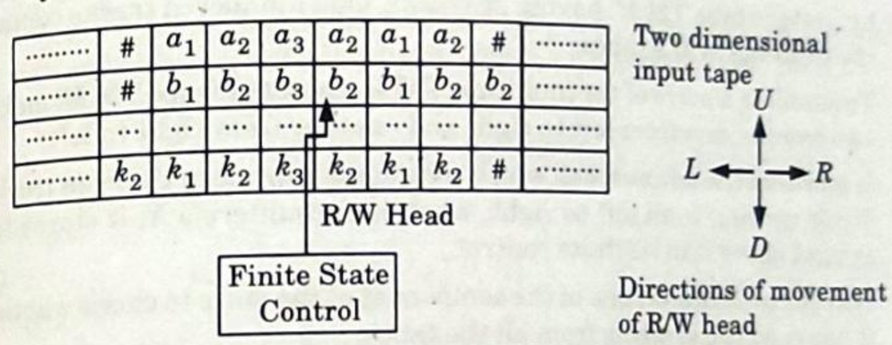


Fig. 5.9.1. A multidimensional Turing machine model.

Que 5.10. **Write a short note on multihead Turing machines.**

Answer

1. A multihead Turing machine $M = (Q, \Sigma, \delta, \Gamma, q_0, \#, H)$, can be viewed as a Turing machine with a single tape and a single finite state control (also called control unit) but with several independent read/write heads.
2. The transition function of a multihead Turing machine can be defined as
$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, \text{stop}\}^n$$
where $\Gamma = \Gamma \times \Gamma \times \Gamma \times \Gamma \times \Gamma \times \Gamma \ldots \times \Gamma$, and n is the number of read/write heads.
3. A move of the multihead Turing machine depends upon the state and the symbol scanned by each head.
4. In one move, the read/write heads may move independently left, right or may remain stationary.
5. A five-head Turing machine (a multihead Turing machine with five heads) is shown by Fig. 5.10.1.

Scanned by TapScanner

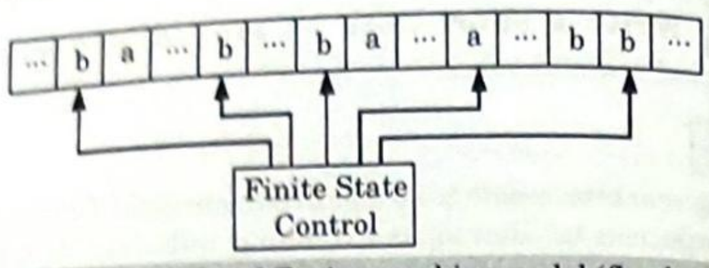


Fig. 5.10.1. A multihead Turing machine model (five heads).

Que 5.11. **Prove that single tape machines can simulate multi tape machines.**

AKTU 2016-17, Marks 10

Answer

1. Let $M = (K, \Sigma, \Gamma, \delta, q_0, F)$ be a k-tape Turing machine. It can be simulated by a single tape TM M' having $2k$ tracks. Odd numbered tracks contain the contents of M's tapes.
2. To simulate a move of the multitape TM M, the single tape TM M' makes two sweeps, one from left to right and another from right to left.
3. It starts on the leftmost cell which contains a X in one of the even tracks. While moving from left to right, when it encounters a X, it stores the symbol above it in its finite control.
4. It keeps a counter as one of the component of the state to check whether it has read the symbols from all the tapes.

		...	A	...	
		...	X	...	
		...		...	B
		...		...	X
	C	...		...	
	X	...		...	

Fig. 5.11.1.

5. After the left to right move is over, depending on the move of M determined by the symbols read, the single tape TM M' makes a right to left move changing the corresponding symbols on the odd tracks and positioning the X's properly in the even numbered tracks.
6. To simulate one move of M, M' roughly takes a number of steps equal to twice the distance between the leftmost and the rightmost cells containing a X in an even numbered the leftmost and the rightmost cells containing a X in an even numbered track.
7. When M' starts all X's will be in one cell.

8. After i move the distance between the leftmost and rightmost X can be at most $2i$. Hence to simulate n moves of M, M' roughly takes $\sum_{i=1}^{n} 2(2i) = \sum_{i=1}^{n} 4i = O(n^2)$ steps. If M reaches a final state, M' accepts and halts.

Que 5.12. **Design a Turing machine that computes a function $f(m, n) = m + n$, *i.e.*, addition of two integers.**

Answer

1. The Turing machine will have 0^m10^n on its tape initially. Therefore, it will start with the leftmost 0, go on scanning 0's moving right keeping them as it, till it get 1, it replaces this 1 by 0, and then again go on scanning 0's, and moving right keeping them as it is.
2. When it gets a B (blank), it moves left, and replaces the rightmost 0 by B, and halts, thereby leaving finally $m + n$ 0's on the tape. Therefore, the moves of the Turing machine are :

	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 0, R)$	—
q_1	$(q_1, 0, R)$	—	(q_2, B, L)
q_0	(q_3, B, R)	—	—
q_3	—	—	—

Therefore, the turing matching $M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_3\})$, where δ is given above. The transition diagram corresponding to the table is shown in Fig. 5.12.1.

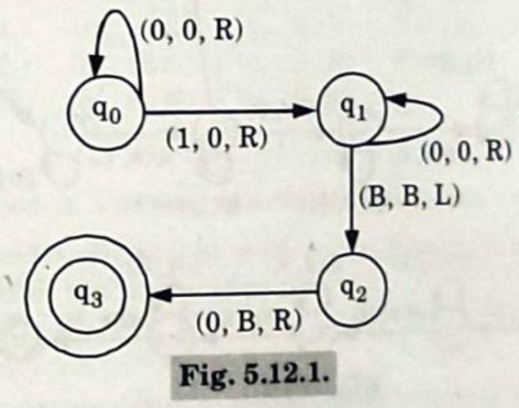


Fig. 5.12.1.

Que 5.13. **What do you understand by instantaneous description of a Turing machine ? Design a Turing machine that computes the**

Scanned by TapScanner

integer function f which multiplies two given positive integers defined as follows :

$$f(m, n) = m * n$$

Answer

Instantaneous description of Turing machine : Refer Q. 5.3(i), Page 5–3A, Unit-5.

Numerical :

1. We will represent m and n by number of Γs :

$$m = 2 \ \#II\#$$
$$n = 3 \ \#III\#$$

2. We know that $m * n$ means addition of m, n times.
3. So, let us assume $m = 2$ and $n = 3$, then these numbers will be represented on input tape as follows :

$$\underbrace{\#II\#}_{m}\underbrace{III\#}_{n} \overset{TM}{\vdash} \underbrace{\#IIIIII\#}_{m*n}$$

6. The machine halts after leaving read/write head to extreme left.
4. We will read first I from n and replace it by d then complete m is copied at extreme left of complete string, the same process is repeated for rest of the Γs of n.
5. At the end of the process Γs in m are copied n times and m and n are erased.

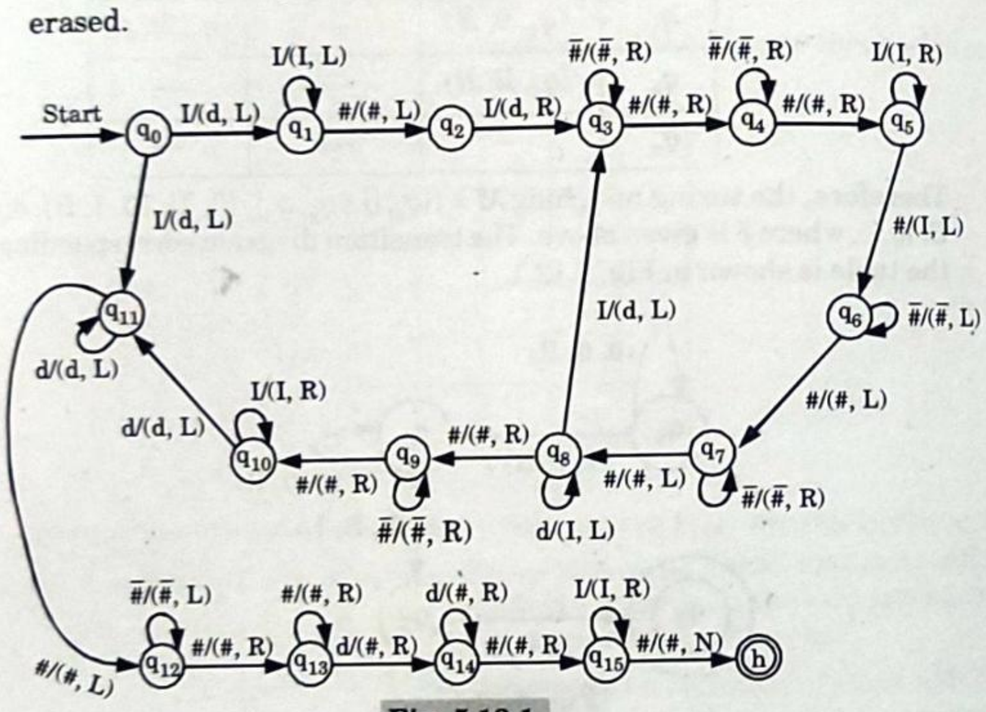


Fig. 5.13.1.

Que 5.14. **Let T_1 and T_2 be two Turing machines; compute the functions f_1 and f_2 from N to N (where N is a natural number) respectively, construct a Turing machine that computes the function min (f_1, f_2).**

Answer

Given that T_1 and T_2 be Turing machine to compute natural number function f_1 and f_2 respectively. We have to construct a TM for computing the function min (f_1, f_2).

For a series of natural number, the min (f_1, f_2) works as a proper subtraction for Turing machine.

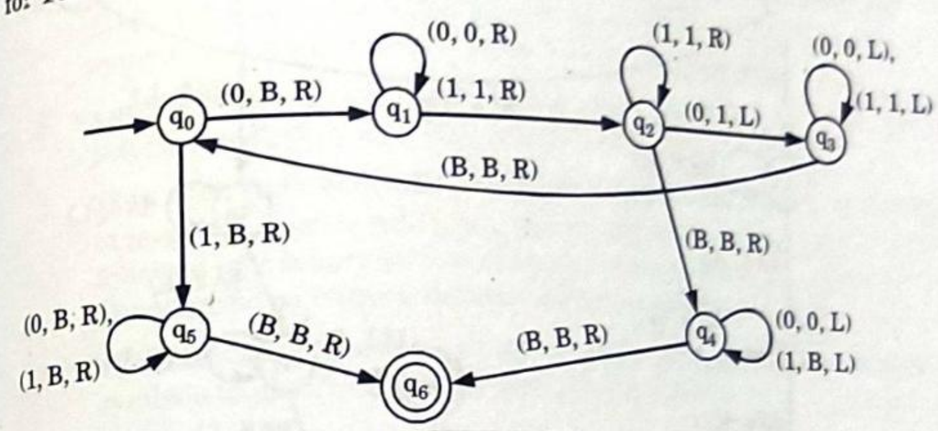


Fig. 5.14.1.

The tuples will be as follows :

δ	0	1	B
q_0	(q_1, B, R)	(q_5, B, R)	–
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	–
q_2	$(q_3, 1, L)$	$(q_2, 1, R)$	(q_4, B, R)
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, B, R)
q_4	$(q_4, 0, L)$	(q_4, B, L)	(q_6, B, R)
q_5	(q_5, B, R)	(q_5, B, R)	(q_6, B, R)
q_6	–	–	–

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$
$\Sigma = \{0, 1\}$
$\Gamma = \{0, 1, B\}$
$q_0 = \{q_0\}$
$B = \{B\}$
$F = \{q_6\}$

Que 5.15. **Design a Turing machine that can compute proper subtraction *i.e.*, $m\$n$, where m and n are positive integers, $m\$n$ is defined as $m - n$ if $m > n$ and 0 if $m \le n$.**

Answer

Let m and n can be represented by number of Γs that is if $m = 4$, it will be represented by#III# on input tape.

Let us analyse the logic of Turing machine.

Scanned by TapScanner

Case 1 : $m > n$

$$\underbrace{\#III\#}_{m=3}\ \underbrace{\#II\#}_{n=2} \vdash \underset{m-n=1}{\#I\#}$$

Case 2 : $m \le n$

$$\underbrace{\#III\#}_{m=2}\ \underbrace{III\#}_{n=2} \vdash \underset{m-n=0}{\#I\#}$$

$$\underbrace{\#II\#}_{m=2}\ \underbrace{III\#}_{n=3} \vdash \underset{m-n=0}{\#\#}$$

The transition diagram is as follows :

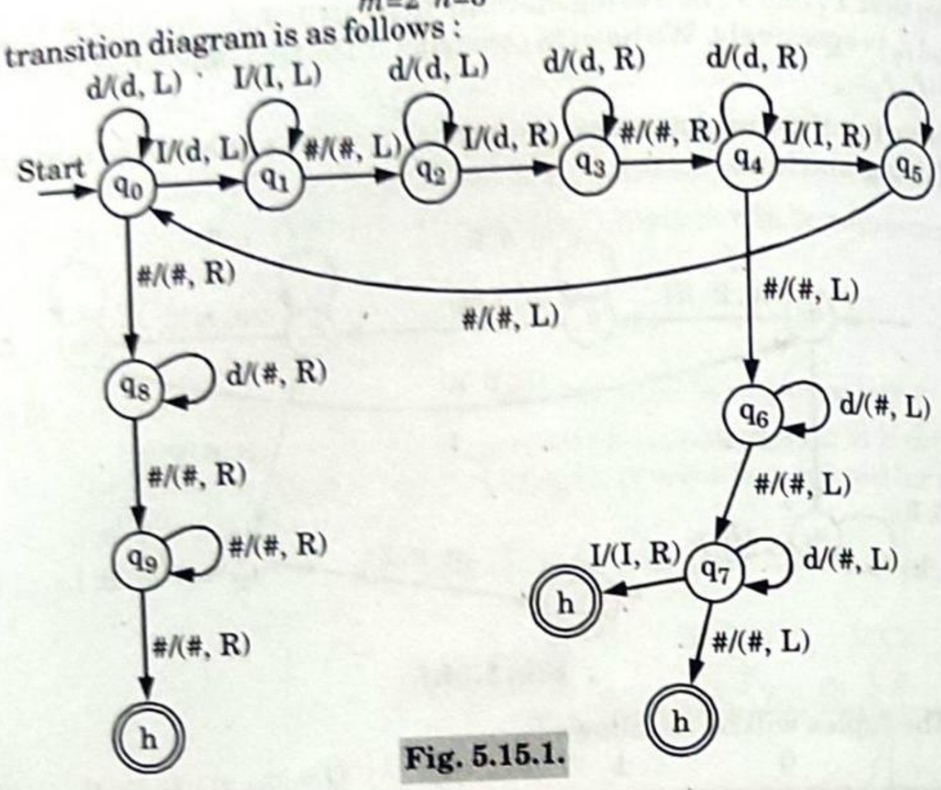


Fig. 5.15.1.

PART-4

Universal Turing Machine, Linear Bounded Automata, Church Thesis, Recursive and Recursively Enumerable Language, Halting Problem.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.16. **Explain Universal Turing machine with example.**

AKTU 2014-15, Marks 05

Answer

1. Universal Turing machines 'U' takes two arguments, a description of a machine TM, and a description of an input string w.
2. We want U to have the following property : U halts on input "TM" "w" if and only if TM halts on input w.

$$U(\text{“}m\text{”}\ \text{“}w\text{”}) = \text{“}m(w)\text{”}$$

3. It is the functional notation of Universal Turing machine.
4. We define a language whose strings are all legal representation of Turing machines.
5. A string representing a Turing machine state is of the form $\{q\}\{0, 1\}^*$; that is, the letter q followed by binary string. Similarly, a tape symbol is always represented as a string in $\{q\}\{0, 1\}^*$.
6. Let TM = (Q, Σ, δ, S) be a Turing machine, and k and l be smallest integers such that $2^k \ge |Q|$, and $2l \ge |\Sigma| + 2$.
7. Then each state in Q will be represented by q followed by a binary string of length k; each symbol in Σ will be likewise represented as letter a followed by a binary string of length k; each symbol in Σ will be likewise represented as letter a followed by a binary string of l bits.
8. The head directions that left (L) and right (R) are included in input tape symbols to justify "+2" term in the definition of l.

For example :

Consider the turing machine

$$m = (Q, \Sigma, \delta, S) \text{ where}$$

$$Q = \{S, q, h\}$$

$\Sigma = \{\#, b, a\}$, and δ is given in following table.

States	Symbol	δ
S	a	$(q, \#)$
S	$\#$	$(h, \#)$
S	b	(S, R)
q	a	(S, a)
q	$\#$	(S, R)
q	b	(q, R)

Since there are three states, three symbols in Σ, we have $k = 2$ and $l = 3$. These are the smallest integers such that $2^k \ge 3$ and $2^l \ge 3 + 2$. The states and symbols are represented as follows :

Scanned by TapScanner

State / Symbol	Representation
S	$q00$
q	$q01$
h	$q11$
#	$a000$
b	$a001$
L	$a010$
R	$a011$
a	$a100$

Thus representation of $baa\#a$ is

"$baa\#a$" = $a001a100a100a100a000a100$

The representing "TM" of the turing machine TM is the following strings :

"TM" = ($q00$, $a100$, $q01$, $a000$), ($q00$, $a000$, $q11$, $a000$), ($q00$, $a001$, $q00$, $a011$), ($q01$, $a100$, $q00$, $a011$), ($q01$, $a000$, $q00$, $a011$), ($q01$, $a001$, $q01$, 011)

The pictorial representation of Universal Turing machine is :

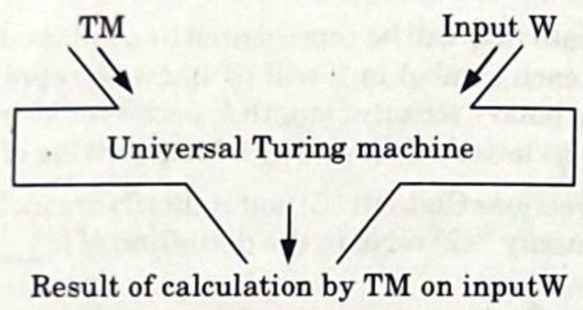


Fig. 5.16.1. Result of calculation by TM on input w.

Que 5.17. **Explain Linear Bounded Automata (LBA).**

Answer

1. A linear bounded automata (LBA) is a non-deterministic Turing machine satisfying the following :
   a. It input alphabets includes two more special symbols ϕ and \$, used as left end marker, and right end marker symbols respectively.
   b. It has no moves to the left of ϕ and to the right of \$. And it is not allowed print another symbol over ϕ or \$.
2. Therefore, LBA is a Turing machine in which the tape on which it computes is restricted to the portion containing input w, plus the left and right end makers. That means there is bound which is linear function of the length of input, as per as use of tape is concerned, hence called as linear bounded automata.
3. Formally therefore, LBA is :

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

where,

Q is finite set of states of Turing machine.

Γ is finite set of symbols called as tape symbols.

Σ is subset of Γ called as input symbols containing ϕ and \$.

q_0 is the initial state.

ϕ, and \$ are the left and right end markers respectively.

δ defines a mapping from $Q \times \Gamma \rightarrow Q \times \Gamma \times (L, R)$.

Therefore, δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times (L, R)$ and $L(M) = \{w \mid w$ is in $(\Sigma - \{\phi, \$\})^*$ and $q_0\, \phi\, w\$ \vdash^* \alpha_1 p \alpha_2$ where p is in $F\}$.

4. The class of languages accepted by LBA are called as context sensitive languages.

Que 5.18. **Write short notes on the following :**

a. **Halting problem**

b. **Church's thesis**

c. **Recursively enumerable language** **AKTU 2015-16, Marks 15**

OR

Discuss the halting problem of a Turing machine.

AKTU 2014-15, Marks 05

Answer

a. **Halting problem :**
   1. The description of a Turing machine M and an input w, does M, when started in the initial configuration $q_0 w$, perform a computation that eventually halts.
   2. Now we ask whether M applied to w, or simply (M, w), halts or does not halt.
   3. The domain of this problem is to be taken as the set of all Turing machines and all w; that is, we are looking for a single Turing machine that, given the description of an arbitrary M and w, will predict whether or not the computation of M applied to w will halt.

b. **Church's thesis :**
   1. Church thesis states that any algorithm that can be performed on any computing machine can be performed on a Turing machine as well.
   2. An algorithm requiring polynomial time was considered as an efficient algorithm, a strengthened version of the Church-Turing thesis was enunciated.
   3. Any algorithmic process can be simulated efficiently by a Turing machine.

Scanned by TapScanner

4. But a challenge to the strong Church-Turing thesis arose from analog computation.
5. Certain types of analog computers had solved some problems efficiently whereas these problems had no efficient solution on a Turing machine.
6. But when the presence of noise was taken into account, the power of the analog computers disappeared.

Strong Church–Turing thesis :

1. Any algorithmic process can be simulated efficiently using a non-deterministic turing machine.
2. Computers are physical objects, and computations are physical processes.
3. What computers can or cannot compute is determined by the law of physics alone, and not by pure mathematics.

c. Recursively enumerable language :

1. TM defines two classes of languages such as recursively enumerable language and recursive language.
2. When a string belonging to S runs on TM then it results in three states :
   a. Acceptance of string.
   b. Loops forever on receiving string.
   c. Rejects the given string.
3. The language L is called recursively enumerable language if there is a turing machine T that accepts every valid strings in L and either rejects or loops for every strings in the complement of L $(\bar{L})$.
4. Recursive enumerable language is also called recognizable, semidecidable, Turing-acceptable or Turing-recognizable because a TM can recognize a string in the language and not able to decide if a string is not in the language.

Recursive language :

1. A language L over input set Σ is called recursive if there is a TM that accepts every valid strings in L and rejects every strings in $\bar{L}$.
2. Recursive language is also called decidable language because a TM can decide membership in these languages.
3. Every recursive language is recursively enumerable language.
4. Not all recursively enumerable languages are recursive. That means every TM which accepts the enumerable language must have some strings for which it loops forever.

Que 5.19. Prove that every recursively enumerable language where complement is closed must be recursive.

Answer

1. Consider a Turing machine M made up of two Turing machines M_1 and M_2.
2. The machine M_2 is complement of machine M_1.
3. We can also denote that $L(M) = L(M_1)$ and $L(M_2)$.
4. Both M_1 and M_2 are simulated in parallel by machine M.
5. Machine M is a two tape TM, which can be made one tape TM for the ease of simulation.
6. This one tape then will consist of tape of machine M_1 and machine M_2.
7. The states of M consist of all the states of machine M_1 and all the states of machine M_2.
8. The machine M made up of M_1 and M_2 is as shown in Fig. 5.19.1.

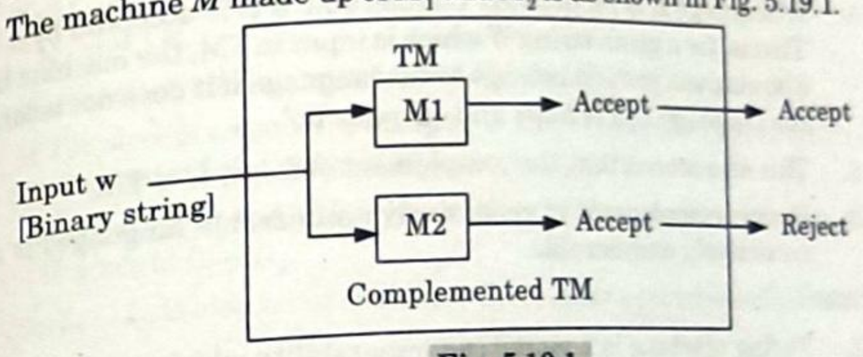


Fig. 5.19.1.

9. If the input w of language L is given to M then M_1 will eventually accept and therefore M will accept L and halt.
10. If w is not in L, then it is in L'. So M_2 will accept and therefore M will halt without accepting.
11. Thus on all inputs, M halts. Thus $L(M)$ is exactly L. Since M always halts we can conclude that $L(M)$ means L is a recursive language.
12. Thus a recursive language can be recursively enumerable but a recursively enumerable language is not necessarily be recursive.

Que 5.20. State True/False with reason :

i. **Every language described by regular expression can be recognized by DFA.**
ii. **Every R.E.L. can be generated by CFL.**
iii. **The Halting problem of TM is decidable.**
iv. **Complement of R.E.L. is also R.E.L.**
v. **Every CFL can be recognized by TM.**

Answer

i. True :
   1. The language denoted by regular expression is called regular language.

Scanned by TapScanner

2. The non-deterministic finite automata with ε transitions accept such a language.
3. The NFA with ε can be converted to NFA without ε.
4. Then such a NFA can easily be converted to DFA.
5. Hence, it is said that every language described by regular expression can be recognized by deterministic finite automata.

ii. False : The recursively enumerable languages are accepted by Turing machines. Hence it has larger class of languages than the class of context free languages.

iii. False : The problem of "whether the machine will halt on particular input or not" is a halting problem and it is an undecidable problem.

iv. True :
1. A language L is recursively enumerable if it is accepted by a TM. That is, for a given string W which is input to TM, the machine halts and outputs 'yes' if it belongs to the language if it does not belong to the language L, TM halts and outputs 'no'.
2. This also means that the complement of L *i.e.*, $L' \in$ TM.
3. Hence, complement of recursively enumerable languages is also recursively enumerable.

v. True :
1. Turing machine is a model of computability which accepts a large class of languages.
2. The context free language is a language generated by context free grammar and accepted by PDA.
3. Hence, CFL is also accepted by TM.

Que 5.21. **Prove the following properties of R.E.L :**

a. Complement of a recursive language is recursive.
b. Union of two recursive language is recursive.
c. Union of two R.E.L is R.E.

Answer

a. Complement of a recursive language is recursive :

Proof :

i. Let L be a recursive language and TM be Turing machine that halts on all inputs and accepts L.

ii. Let us construct a Turing machine TM' from TM so that if TM enters a final state on input w, then TM' halts without accepting.

iii. If TM halts without accepting, TM' enters a final state.

iv. Since one of these two events occurs, TM' is an algorithm. So clearly T(TM') is the component of L and thus the complement of L is recursive language. Fig. 5.21.1 shows constructions of TM' from TM.

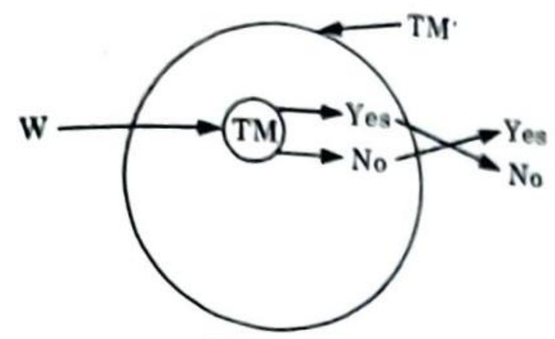


Fig. 5.21.1.

b. Union of two recursive language is recursive :

Proof :

i. Let L_1 and L_2 be two recursive languages accepted by Turing machines TM_1 and TM_2.

ii. We construct a Turing machine TM, which first simulate TM_1.

iii. If TM_1 accepts, then TM accepts. If TM_1 rejects then TM simulates TM_2 and accepts if and only if TM_2 accepts.

iv. Since both TM_1 and TM_2 is algorithm so TM is guaranteed to halt. Clearly TM accepts $L_1 \cup L_2$.

So $L_1 \cup L_2$ is also recursive since there exist a turing machine TM for it.

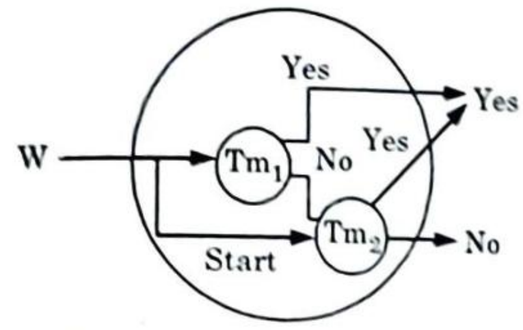


Fig. 5.21.2. Construction for union.

c. Union of two R.E.L is R.E. :

Proof :

i. We know that enumerating Turing machine have one input tape and one output tape.

ii. Let L_1 and L_2 are recursively enumerable languages and their enumerative turing machines are TM_1 and TM_2 respectively.

iii. Let us design a Turing machine TM which can simulate TM_1 and TM_2 simultaneously on separate tape.

iv. If either accepts, then TM accepts as follows :

Scanned by TapScanner

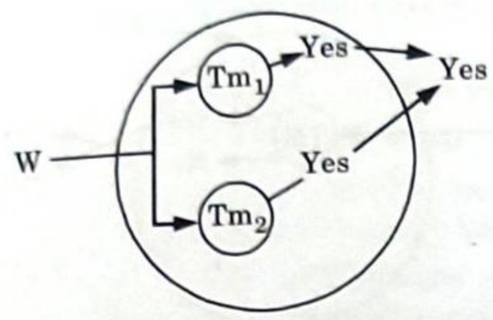


Fig. 5.21.3. Construction for union.

Que 5.22. **Prove that recursively enumerable languages are closed under intersection.**

Answer

1. Let L_1 and L_2 be recursive languages.
2. Then there are Turing machines M_1 and M_2, such that $L(M_1) = L_1$, $L(M_2) = L_2$ and both M_1 and M_2 halt on every input string.
3. We construct another Turing machine M based on M_1 and M_2 to simulate union, intersection, and complement.
4. If there is a TM, M which accepts the language of $L_1 \cup L_2$ and $L_1 \cap L_2$, then recursive languages are closed under each operation.

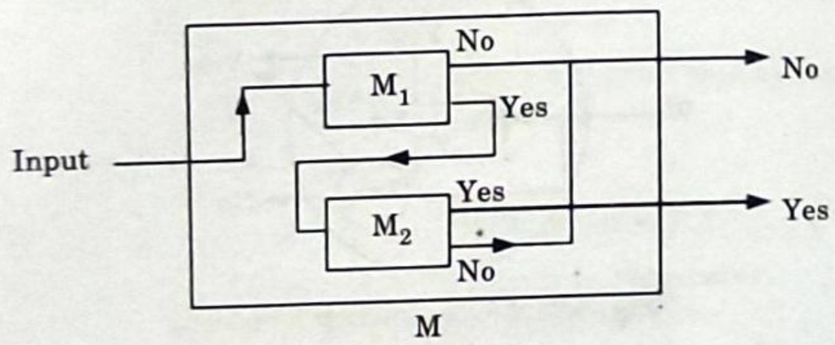


Fig. 5.22.1.

PART-5

Post's Correspondence Problem, Introduction to Recursive Function Theory.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.23. **What do you understand by undecidable problem ? State the halting problem and prove that halting problem is undecidable.**

OR

Prove that the halting problem is undecidable.

AKTU 2016-17, Marks 10

Answer

Undecidable problem : A problem is undecidable if there exist no algorithm that takes as input an instance of the problem and determine whether the answer to that instance is "yes" or "no".

Halting problem : Refer Q. 5.18, Page 5–19A, Unit-5.

Halting problem is undecidable :

a. We assume that $HALT_{TM}$ is decidable, and get a contradiction.
b. Let M_1 be the TM such that $T(M_1) = HALT_{TM}$ and let M_1 halt eventually on all (M, w).
c. We construct a TM M_2 as follows :
   1. For M_2, (M, w) is an input.
   2. The TM, M_1 acts on (M, w).
   3. If M_1 rejects (M, w) then M_2 rejects (M, w).
   4. If M_1 accepts (M, w), simulate the TM M on the input string w until M halts.
   5. If M has accepts w, M_2 accepts (M, w); otherwise M_2 rejects (M, w).
d. When M_1 accepted (M, w) (in step 4), the Turing machine M halts on w.
e. In this case either an accepting state q or a state q' such that $\delta(q', a)$ is undefined till some symbol a in w is reached.
f. In the first case (the first alternative of step 5) M_2 accepts (M, w).
g. In the second case (the second alternative of step 5) M_2 rejects (M, w).
h. It follows from the definition of M_2 that M_2 halts eventually.

Also, $T(M_2) = \{(M, w) \mid \text{The turing machine accept } w\}$
$= A_{TM}$

This is a contradiction since A_{TM} is undecidable.

Que 5.24. **Write short note on Post Correspondence Problem (PCP).**

AKTU 2014-15, Marks 05

Answer

1. The post correspondence problem is another undecidable problem that turns out to be a very helpful tool for proving problems in logic or in formal language theory to be undecidable.

Scanned by TapScanner

2. Let Σ be an alphabet with at least two letters. An instance of the post corresponding problem (PCP) is given by two sequences $U = (u_1, u_2, ..., u_m)$ and $V = (v_1, v_2, ..., v_m)$ of strings $u_i, v_i \in \Sigma^*$.
3. The problem is to find whether there is a (finite) sequence $(i_1, i_2, ..., i_p)$, with $i_j \in \{1, 2, ..., m\}$ for $i_j = 1, 2, ... p$ so that, $p \geq 1$

$$u_{i1}, u_{i2} u_{i3} ...u_{ip} = v_{i1} v_{i2} ... v_{ip}.$$

4. Equivalently, on instance of the PCP is a sequence of pairs

$$\begin{pmatrix} u_1 \\ \vdots \\ v_1 \end{pmatrix}, ..., \begin{pmatrix} u_m \\ \vdots \\ v_m \end{pmatrix}$$

5. The sequence $i_1, i_2 ... i_p$ is said to be solution to this instance of PCP.

Que 5.25. Does the PCP with two lists $X = (10, 011, 101)$, $Y = (101, 11, 011)$ have a solution ? **AKTU 2014-15, Marks 10**

Answer

$$X = (10, 011, 101)$$
$$Y = (101, 11, 011)$$

1. Let us assume that this instance of PCP has solution $i_1, i_2 .... i_p$.
2. Clearly $i_1 = 1$ since no string beginning with $w_2 = 011$ can equal a string beginning with $x_2 = 11$, no string beginning with $w_3 = 101$ can equal a string beginning with $x_3 = 011$.

	X	Y
i	w_i	x_i
1	10	101
2	011	11
3	101	011

3. We write the string from list X with the corresponding string from Y. So we have

$$10$$
$$101$$

The next selection from X must begin with a one.

4. Thus, $i_2 = 1$ or $i_2 = 3$. But $i_2 = 1$ will not do, since no string beginning with $w_1 \wedge_1 w_1 = 1010$ can equal a string beginning with $x_1 \wedge_1 x_1 = 101101$.
5. With $i_2 = 3$ we have

$$10101$$
$$101011$$

since the string from list Y again exceeds the string from list X by single symbol 1, a similar argument shows that $i_3 = i_4 ..... = 3$.

6. Thus, there is only one sequence of choices that generates compatible strings, and for this sequence string Y is always one character longer.
7. Thus, the given PCP has no solution.

Que 5.26. Define PCP. Let $A = \{1, 110, 0111\}$ and $B = \{111, 001, 11\}$ and $\Sigma = \{0.1\}$. Find the solution of PCP. **AKTU 2018-19, Marks 10**

Answer

PCP : Refer Q. 5.24, Page 5–25A, Unit-5.

Numerical :

$$A = \{1, 110, 0111\}$$
$$B = \{111, 001, 11\}$$

i	x_i	y_i
1	1	111
2	110	001
3	0111	11

Consider x-series substring as :

$x_1 x_2 x_3$

Corresponding x-series substring as :

11100111

Consider y-series substring as :

$y_1 y_2 y_3$

Corresponding y-series substring as :

11100111

Here, both series are same. In this instance we say that PCP has solution as (1, 2, 3).

Que 5.27. Explain Modified PCP (MPCP).

Answer

In the modified PCP, there is the additional requirement on a solution that the first pair on the list X and list Y must be the first pair in the solution. More formally, an instance of MPCP is two lists

$$X = w_1, w_2, ..., w_k$$

Scanned by TapScanner

and $Y = x_1, x_2, x_3, \ldots, x_k$,

and a solution is a list of 0 or more integers $i_1, i_2, i_3, \ldots, i_p$, such that

$$w_1, w_{i1}, w_{i2}, \ldots, w_{im} = x_1, x_{i1}, x_{i2}, \ldots, x_{ip}.$$

Que 5.28. Define Post's Correspondence Problem (PCP) and Modified PCP with its application. Find any three CPC solutions of the lists $x = (b, bab^3, ba)$ and $y = (b^3, ba, a)$.

AKTU 2017-18, Marks 07

Answer

PCP : Refer Q. 5.24, Page 5–25A, Unit-5.
Modified PCP : Refer Q. 5.27, Page 5–27A, Unit-5.
Application : The ambiguity of CFG is undecidable.
Numerical :
We have to determine whether or not there exists a sequence of substrings of x such that the string formed by this sequence and the string formed by the sequence of corresponding substrings of y are identical. The required sequence is given by $i_1 = 2$, $i_2 = 1$, $i_3 = 1$, $i_4 = 3$, *i.e.*, (2, 1, 1, 3). The corresponding strings are

$$\underset{x_2}{\boxed{bab^3}}\ \underset{x_1}{\boxed{b}}\ \underset{x_1}{\boxed{b}}\ \underset{x_3}{\boxed{ba}} = \underset{y_2}{\boxed{ba}}\ \underset{y_1}{\boxed{b^3}}\ \underset{y_1}{\boxed{b^3}}\ \underset{y_3}{\boxed{a}}$$

Thus the PCP has a solution.

Que 5.29. Explain recursive function theory.

Answer

1. The theory of recursive function is just converse of Church's hypothesis.
2. Recursive function theory begins with some very elementary functions that are intuitively effective. Then it provides a few methods for building more complicated functions from simpler functions. Some of elementary function :

a. **Initial function :**
   i. The initial functions are the elementary functions whose values are independent of their smaller arguments.
   ii. The following functions comprise the class of recursive functions :
   The zero function : $z(x) = 0$
   The zero function returns zero regardless of its argument.
   The successor function : $s(x)$ = successor of x (roughly, "$x+1$")
   The successor functions take only one argument each.
   The identify function : $id(x) = x$
   But the identity function is designed to take any number of arguments.

b. **Composition :** We will start with successor function,

$$s(x) = x + 1$$

Then we may replace its argument, x, with a function. If we replace the argument, x, with the zero function.

$$z(x)$$

then the result is the successor of zero,

$$s(z(x)) = 1$$

$$s(s(z(x))) = 2 \text{ and so on.}$$

In this way, with the help of the initial functions we can describe the natural numbers. This building operation is called "composition". It should be clear that when composition is applied to computable functions, only computable functions will result.

c. **Primitive recursion :**
   i. The second building operation is called primitive recursion.
   ii. Primitive recursion is a method of defining new functions from old function.
   iii. The function h is defined through functions f and g by primitive recursion when

$$h(x, 0) = f(x)$$

Then $f(5) = 5 * f(4)$

$$= 5 * 24 = 120$$

   iv. Primitive recursion is like mathematical induction. The first equation defines the basis, and the second defines the induction step.

d. **Minimization :**
   i. If $g(x)$ is a function that computes the least x such that $f(x) = 0$, then g is computable.
   ii. g is produced from f by minimization.
   iii. But we can build g by minimization only if $f(x)$ is already known to be computable.

Que 5.30. What is recursive function ? Explain classification of recursive function.

Answer

Recursion function : A recursive function is a function defined by a finite set of rules that for various arguments specify the function in terms of variables, non-negative integer constants, the successor function, the function itself, or an expression built from these by composition of functions.

Class of recursive functions : The classification of recursive functions is as follows :

Scanned by TapScanner

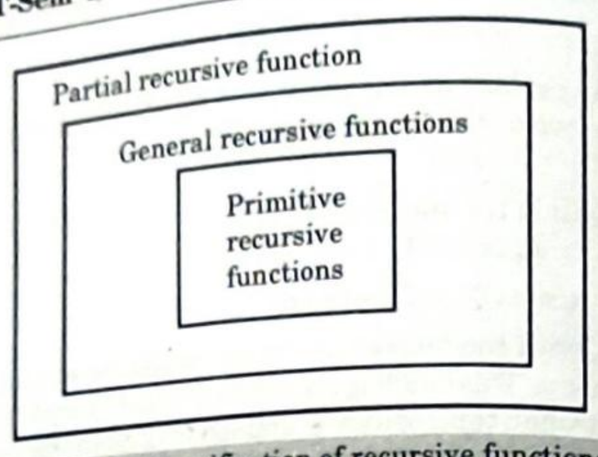


Fig. 5.30.1. Classification of recursive functions.

1. **Partial recursive function :** The function is called partial recursive function if it can be obtained by applying composition, primitive recursion and minimization as building operations.
2. **General recursive function :**
   a. The function is called general recursive if it is obtained by applying composition, primitive recursive and an unbounded minimization that happen to terminate.
   b. The general recursive function is a larger class than partial recursive functions.
3. **Primitive recursive function :**
   a. The function is called primitive recursive if it is obtained by applying composition, primitive recursion and unbounded minimization that does not terminate.
   b. The set of general recursive function is the same as the set of Turing computable functions.
   c. The example of general recursive function is an Ackerman's function. The Ackermann's function can be defined as follows :

$$A(0, y) = y + 1$$
$$A(x + 1, 0) = A(x, 1)$$
$$A(x + 1, y + 1) = A(x, A(x + 1, y))$$

Then we can calculate $A(x, y)$ for every pair of (x, y).

Que 5.31. **Discuss tractable and non tractable problems.**

AKTU 2014-15, Marks 10

Answer

1. A problem is said to be tractable if there exists an algorithm of polynomial complexity for all instances and solves the problem.
2. When the space required for implementing the steps of the particular algorithm is reasonable, we can say that the problem is tractable, that is solvable in practice.

3. Conversely, a problem is said to be intractable, if the optimal algorithm for solving the problem cannot solve all of its instances in polynomial time.
4. The problems are intractable if the time required for any of the algorithm is at least $f(n)$, where f is an exponential function of n.

Que 5.32. **Design a TM to recognize all strings consisting of an odd number of α's.**

AKTU 2016-17, Marks 10

Answer

The turing machine M can be constructed by the following moves :

i. Let q_1 be the initial state.

ii. If M is q_1; on scanning α, it enters the state q_2 and writes B(blank).

iii. If M is q_2; on scanning α, it enters the state q_1 and writes B(blank).

iv. From the previous 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_2'	Present state 'q_1'
A	BRq_2	BRq_1

Que 5.33. **Design the Turing machine that accepts the language of even integers written in binary.**

AKTU 2014-15, Marks 10

Answer

The Turing machine for given language will be :

$$TM = (Q, \Sigma, \Gamma, \delta, q_o, B, F)$$
$$= (\{q_o, q_f\}, \{0, 1\}, \{0, 1, B\}, \delta, q_o, B, \{q_f\})$$

Language of even integers written in binary will contain a set of strings that always ends with zero (0).

The transition diagram for given string will be :

Scanned by TapScanner

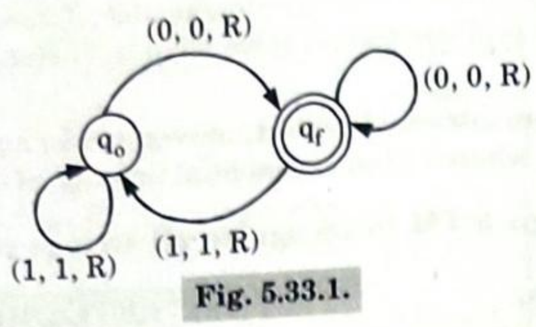


Fig. 5.33.1.

The transition table will be :

Table 5.33.1.

Q \ Input	0	1	B
q_0	$(q_f, 0, R)$	$(q_0, 1, R)$	(q_f, B, R)
q_f	$(q_f, 0, R)$	$(q_0, 1, R)$	(q_f, B, R)

Que 5.34. Explain the modification done in finite automata (FA) to make it :

i. PDA

ii. Turing machine

AKTU 2014-15, Marks 05

Answer

i. PDA :

1. If we add an extra memory element in finite automata to accept the context free languages, known as push down automata. We use a stack corresponding to memory element.
2. The push down automata is essentially a finite automata with control of both input tape and a stack to store what it has read.
3. A stack is last in, first out list, *i.e.*, symbol can be entered or removed from top of list.
4. Thus, a PDA must consist of an input tape, a finite control and a stack. The pictorial representation of PDA is as follows :

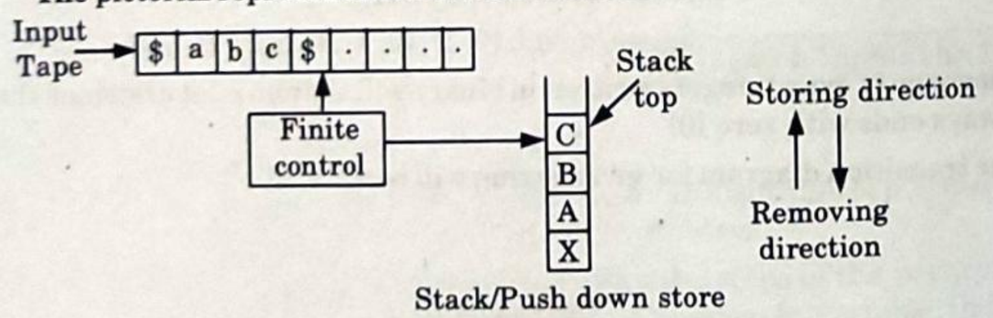


Fig. 5.34.1.

ii. Turing machine :

1. If we make tape size infinite rather than finite in finite automata and movement of tape head is done on left or right to accept any language and use a stack corresponding to memory element.
2. Turing machine consist of input tape, finite control, tape head and action.
3. We can visualize a TM as a physical computing device that can be represented as a diagram as shown in Fig. 5.35.2.

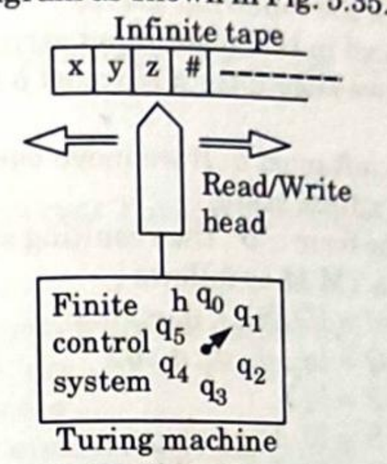


Fig. 5.34.2.

Que 5.35. Design a turing machine for language containing palindromes of *a*'s and *b*'s.

AKTU 2018-19, Marks 10

Answer

Turing machine for language containing palindrome of *a* and *b* :

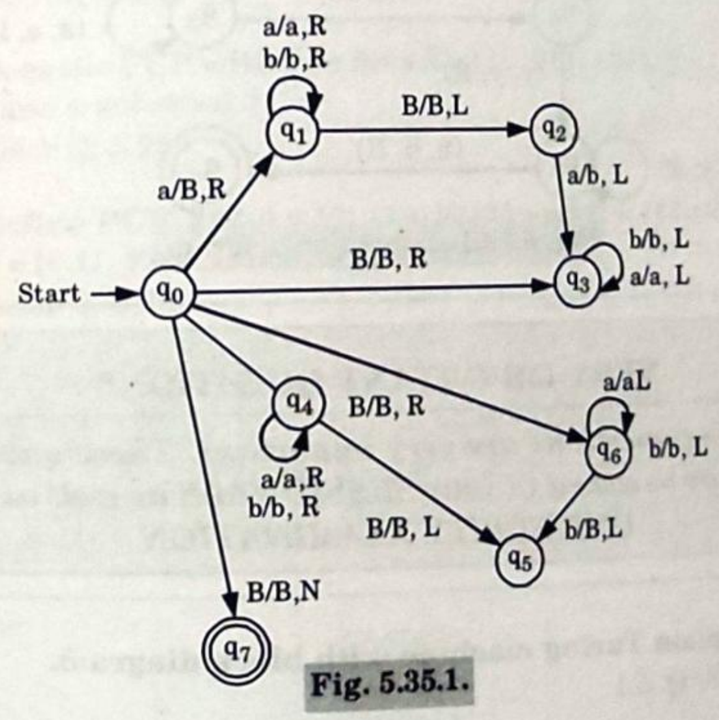


Fig. 5.35.1.

Scanned by TapScanner

Que 5.36. **Define Turing Machine (TM). Construct the TM for the language $L = \{a^n b^n \mid n > 0\}$.** **AKTU 2017-18, Marks 07**

Answer

Turing Machine : Refer Q. 5.2, Page 5–3A, Unit-5.

Numerical : We require the following moves :

a. If the leftmost symbol in the given input string w is a, replace it by x and move right till we encounter a leftmost b in w. Change it to y and move backwards.

b. Repeat (a) with the left most a. If we move back and forth and no a or b remains, move to a final state.

c. For strings not in the form $a^n b^n$, the resulting state has to be non-final.

Now, we construct a TM M as follows :

$$M = (Q, S, G, d, q_0, b, F)$$

where
$$Q = \{q_0, q_1, q_2, q_3, q_f\}$$
$$F = \{q_f\}$$
$$S = \{0, 1\}$$
$$G = \{0, 1, x, y, b\}$$

The transition diagram is given in Fig. 5.36.1. M accepts $\{a^n b^n \mid n > 0\}$.

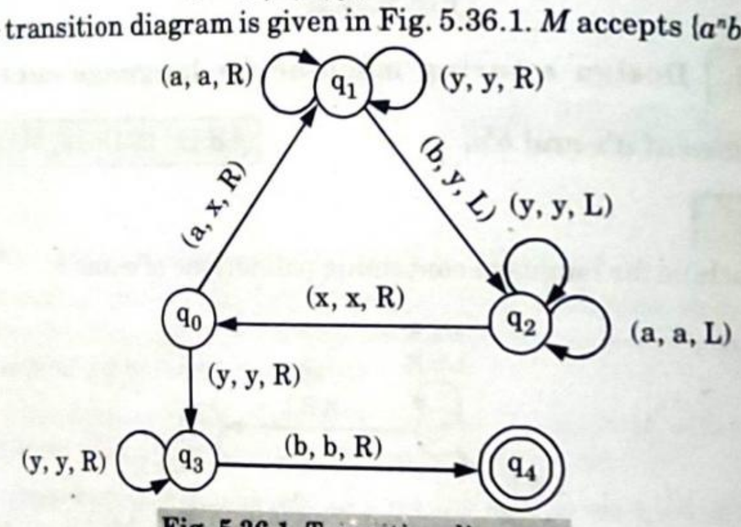


Fig. 5.36.1. Transition diagram.

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. Explain Turing machine with block diagram.

Ans. Refer Q. 5.1.

Q. 2. Construct a Turing machine which accepts the regular expression, $L = \{0^n 1^n \mid n >= 1\}$.

Ans. Refer Q. 5.4.

Q. 3. Design Turing machine for the language
$$L = \{a^{n+2} b^n \mid n > 0\}.$$

Ans. Refer Q. 5.5.

Q. 4. Prove that single tape machines can simulate multi tape machines.

Ans. Refer Q. 5.11.

Q. 5. Explain Universal Turing machine with example.

Ans. Refer Q. 5.16.

Q. 6. Write short notes on the following :

a. Halting problem

b. Church's thesis

c. Recursively enumerable language

Ans. Refer Q. 5.18.

Q. 7. What do you understand by undecidable problem ? State the halting problem and prove that halting problem is undecidable.

Ans. Refer Q. 5.23.

Q. 8. Write short note on Post Correspondence Problem (PCP).

Ans. Refer Q. 5.24.

Q. 9. Does the PCP with two lists $X = (10, 011, 101)$, $Y = (101, 11, 011)$ have a solution ?

Ans. Refer Q. 5.25.

Q. 10. Define PCP. Let $A = \{1, 110, 0111\}$ and $B = \{111, 001, 11\}$ and $\Sigma = \{0.1\}$. Find the solution of PCP.

Ans. Refer Q. 5.26.

Q. 11. Explain the modification done in finite automata (FA) to make it :

i. PDA

ii. Turing machine

Ans. Refer Q. 5.34.

☺☺☺

Scanned by TapScanner