Contents

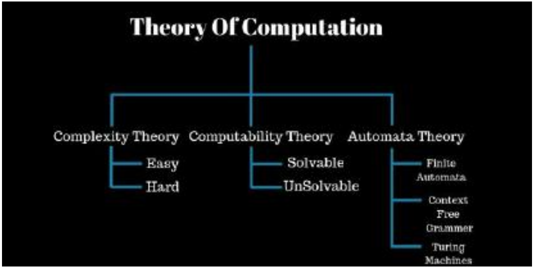
Contents

Contents

Contents

Theory of Computation: Introduction

- Theory of Computation is the branch of Computer Science which deals with how efficiently problems can be solved on model of computation using an algorithm
- The domain is further classified into 3 sub-domains:
  - Automata theory and languages
  - Computability theory
  - Complexity theory



Preliminaries

- Propositions (or Statements)
- Connectives (Propositional connectives or Logical connectives)
  - NOT (Negation)$\neg P$
  - AND (Conjunction) $P \wedge Q$
  - OR (Disjunction) $P \vee Q$
  - If.. Then... (Implication)
  - If and only If
- Tautology- A tautology or a universally true formula is a well defined formula whose truth value is T for all possible assignments of truth values to the propositional variables.
  Example-$P \vee \neg P$
- Contradiction- A contradiction (or absurdity) is well formed formula whose truth value is F for all possible assignments of truth values to propostion variables.
  Example-$P \wedge \neg P$
- Equivalence

Preliminaries

- Equivalence- Two well formed α and β in propositional variables $P_1, P_2, ...P_n$ are equivalent (or logically equivalent) if the formula $\alpha \leftrightarrow \beta$ is a tautology.

Example

$$(P \implies (Q \vee R)) \equiv ((P \implies Q) \vee (P \implies R))$$

Preliminaries

- Logical Identities
  - Idempotent laws- $P \vee P \equiv P$, $P \wedge P \equiv P$
  - Commutative laws- $P \vee Q \equiv Q \vee P$, $P \wedge Q \equiv Q \wedge P$
  - Associative laws- $P \vee (Q \vee R) \equiv (P \vee Q) \vee R$
    $P \wedge (Q \wedge R) \equiv (P \wedge Q) \wedge R$
  - Distributive laws- $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$
    $P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$
  - Absorption laws- $P \wedge (P \vee Q) \equiv P$
    $P \vee (P \wedge Q) \equiv P$
  - De-morgan's Laws- $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$
    $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$
  - Contrapositive- $P \Rightarrow Q \equiv \neg Q \Rightarrow \neg P$
    $P \Rightarrow Q \equiv \neg P \vee Q$
  - Double negation- $P \equiv \neg(\neg P)$

Questions

- Show that:
  $(P \wedge Q) \vee (P \wedge \neg Q) \equiv P$
- Show that:
  $(P \implies Q) \wedge (R \implies Q) \equiv (P \vee R) \implies Q$

Mathematical Preliminaries

Set:

- A **set** is well defined collection of objects.
  Example-Set of all students in ASET,
  Collection of all books in library.
- Individual objects are called **Members or Elements** of the set.
- Capital letter usually represent Set such as A,B,C,...
- Small letters represent Elements of any set such as a,b,c,....
- If a is an element of set $A \implies a \in A$
- Ways of describing set
  - Listing its element with no repetition
    Example: $\{15, 30, 45, 60, 75, 90\}$
  - Describing properties of elements of set
    Example: $\{n \mid nisapositiveintegerdivisibleby15andlessthan100\}$
  - By recursion
    Example: Set of all natural numers leaving remainder 1 when divided by 3 can be written as
    $\{a_n \mid a_0 = 1, a_{n+1} = a_n + 3\}$

Mathematical Preliminaries

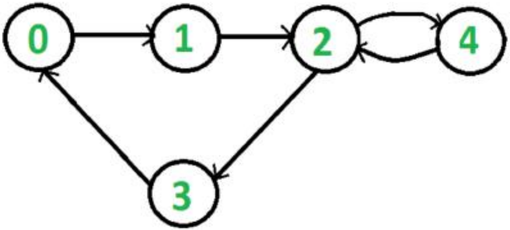
Subsets and Operations on Sets

- A set A is said to be subset of B i.e. $(A \subseteq B)$, if every element of A is also an element of B.
- If two sets A and B are *equal* i.e. $(A = B)$ $\implies A \subseteq B$ and $B \subseteq A$.
- Empty set: A set with no elements.
- **Operations on sets**:
  - $A \cup B$: $\{x \mid x \in A \text{ or } x \in B\}$ called union of A and B.
  - $A \cap B$: $\{x \mid x \in A \text{ and } x \in B\}$ called intersection of A and B.
  - $A - B$: $\{x \mid x \in A \text{ and } x \notin B\}$ called complement of B in A.

Mathematical Preliminaries

Graph

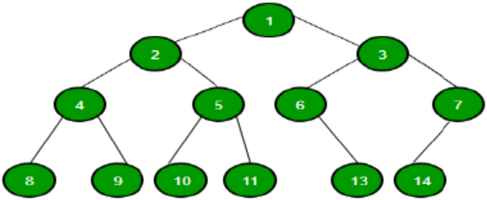
- A graph (or undirected graph) consists of:
  - a non-empty set V of **vertices**
  - a set E called set of **edges**.
  - a **map** ϕ which assigns to every edge a unique unordered pair of vertices.
- A directed graph or (digraph) consists of
  - a non-empty set V of **vertices**
  - a set E called set of **edges**
  - a **map** ϕ which assigns to every edge a unique ordered pair of vertices.



Mathematical Preliminaries

Tree

- A graph is called a tree if it is connected and has no circuits



- Properties of tree:
  - A tree is connected **graph with no circuits** or loops
  - there is **one and only one path** between every pair of vertices.
  - if a connected graph has n **vertices** $\Longrightarrow$ **has** $n-1$ **edges**, implies a tree

Automata

- Automata is defined as a system where energy, material and information are transformed, transmitted and used for performing some function without direct human participation.
- Example: Automatic machine tools, automatic packing machines, automatic photoprinting machines

Figure: Automatic machine tools

Figure: Automatic photoprinting machine

Discrete Automata

- Model of discrete automaton:

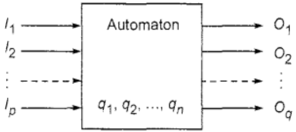


Figure: Model of a discrete automaton

- Characteristics of Automata
  - Input-At a discrete instant of **time** $t_1, t_2, \ldots\ldots t_m$, **input values** $I_1, I_2 \ldots. I_p$ take finite number of fixed values from **input alphabet** Σ are applied as input to model.
  - Output- $O_1, O_2, \ldots O_q$ are output of model, each of which can take finite number of fixed values from an Output.
  - States- At any instant of time, the automaton can be in one of the states $q_1, q_2, \ldots q_n$
  - State relation- The next state of an automaton at any instant of time is determined by present state and present input.
  - Output relation- On **reading** an input symbol, automaton moves to next state which is given by state relation.

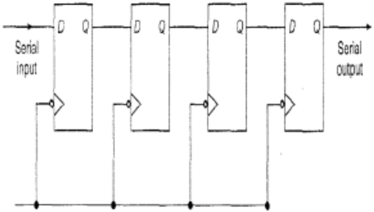
Discrete Automata

- Automaton without Memory:An automata in which the output depends only on input.
- Automaton with finite Memory: An automaton in which the output depends on states as well as input.
  - Moore Machine: An automaton in which the output depends only on states of machine.
  - Mealy Machine: An automaton in which output depends on the state as well as on the input at any instant of time.

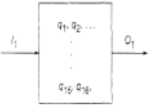
Discrete Automata

Any sequential machine behaviour can be represented by an automaton.

Example: Consider a 4-bit serial shift register as a finite state machine.



- $2^4 = 16$ states (0000, 0001, ..., 1111)
- 1 serial Input and 1 serial output
- Input alphabet, $\Sigma = \{0,1\}$
- Can be represented as



- Here, output depends on both Input and state $\therefore$ **Mealy machine**.

Finite Automaton

- A finite automaton can be represented by a **finite 5-tuple** $(Q, \Sigma, \delta, q_0, F)$, where:
  - Q is a finite non-empty set of **states**.
  - Σ is a finite non-empty set of inputs called **Input** alphabet.
  - δ is a function which maps $Q \times \Sigma \rightarrow Q$ called **Direct transition function**
    It describes change of states during transition.
    It is represented by transition table $\backslash$ diagram.
  - $q_0 \in Q$ is **Initial state**
  - $F \subseteq Q$ is the set of **Final states**
- The transition function which maps $Q \times \Sigma^*$ into Q is called **Indirect transition function**.

Finite Automata

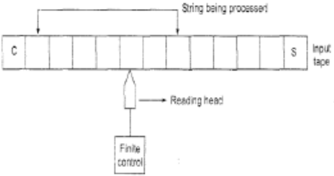
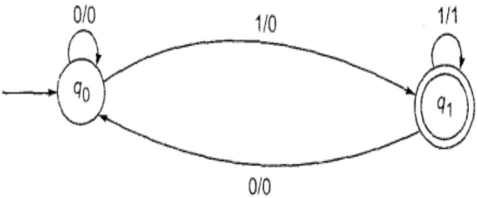


Figure: Block diagram of Finite Automata

- Input tape: Each square contains a single symbol from input alphabet Σ.
  - ¢ and $ are end markers of Tape
  - Absence of end markers indicates that the tape is of infinite length
- Reading Head: Examines only 1 □ at a time
  - can move from Left → Right *or* Right → Left
- Finite Control: Input to a finite control will usually be a symbol under read head.
  Following Outputs:
  - A motion to R-head along the tape on next □

Transition Systems

- A transition system or transition graph is finite directed labelled graph in which each vertex (node) is represented by a state and edges are labelled with input/output.



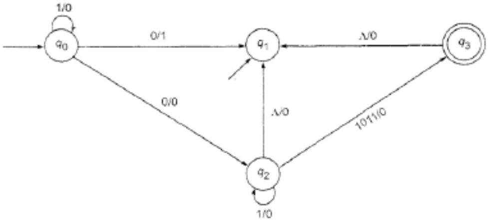
- A transition system is a 5-tuple $(Q, \Sigma, \delta, Q_0, F)$ where:
  - Q, Σ, F are finite non-empty set of states, input alphabet and set of final states respectively.
  - $Q_0 \subseteq Q$ and is non-empty
  - δ is a finite subset of $Q \times \Sigma^* \times Q$

Transition system

- A transition system accepts a string w in Σ^* if:
  - There exists a path which originates from some initial state, goes along the arrows and terminates at some final state, and
  - The path value obtained by concatenation of all edge-labels of the path is equal to w

Example

Consider the given transition system:



Determine the initial states, final states and acceptability of 101011, 111010.

Initial states: q_0 and q_1; Final State: q_3

Path value $q_0 q_0 q_2 q_3$ for 101011 $\Longrightarrow$ **accepted by system**

But, 111010 not accepted

Transition function

- Every finite automaton (Q, Σ, δ, q_0, F) can be viewd as a transition system (Q, Σ, δ',Q_0,F) if we take $Q_0 = \{q_0\}$ and $\delta' = \{(q,w,\delta(q,w)) \mid q \in Q,\ w \in \Sigma^*\}$
- But, a transition system need not be a finite automaton.
- Example: *A transition system may contain more than one initial state.*
- **Properties of Transition Functions:**
  1. $\delta(q, \wedge) = q$ is a finite automaton
     $\Longrightarrow$ State of the system can be changed only by an input symbol.
  2. For all strings w and input symbol a:
     $\delta(q, aw) = \delta(\delta(q, a), w)$
     $\delta(q, wa) = \delta(\delta(q, w), a)$

Exercise:
Prove that for any transition function δ and for any two input string x and y
$\delta(q, xy) = \delta(\delta(q, x), y)$

Acceptability of a String by a finite automaton

- A string 'x' is accepted by a finite automaton M $= (Q, \Sigma, \delta, q_0, F)$ if $\delta(q_0, x) = q$ for some $q \in F$

Example

Consider the finite state machine whose transition function δ is given below in form of a transition table. Here, Q=$\{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, F $= \{q_0\}$.
Give the entire sequence of states for the input string 110101.

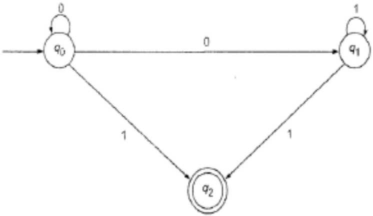
	Input	
State	0	1
$\rightarrow$ (q_0)	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Answer: $\delta(q_0, \mathbf{1}10101) = \delta(q_1, \mathbf{1}0101)$

$$
\begin{aligned}
&= \delta(q_0, \mathbf{0}101) \\
&= \delta(q_2, \mathbf{1}01) \\
&= \delta(q_3, \mathbf{0}1) \\
&= \delta(q_1, \mathbf{1}) \\
&= \delta(q_0, \wedge) \\
&= q_0
\end{aligned}
$$

Hence, $q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_0 \xrightarrow{0} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_1 \xrightarrow{1} q_0$ Accepted

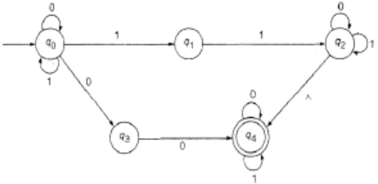
Non-Deterministic Finite State Machines



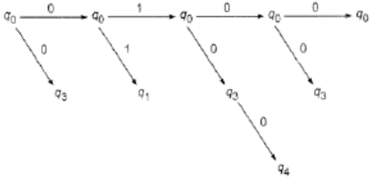
- A non-deterministic finite automaton (NDFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where
  Q is a finite non-empty set of states
  Σ is a finite non-empty set of inputs
  δ is a transition function mapping from $Q \times \Sigma$ into 2^Q which is power set of Q, the set of all subsets of Q
  $q_0 \in Q$ is the initial state
  $F \subseteq Q$ is the set of final states.

Non-Deterministic Finite State Machines

- Consider a Non-deterministic automaton as under:



Determine the sequence of states for input string 0100



$\delta(q_0, 0100) = \{q_0, q_3, q_4\}$
Since q_4 is the final state. $\therefore$ input string 0100 is accepted by the system.

- A string $w \in \Sigma^*$ is accepted by NDFA "M". If $\delta(q_0, w)$ contains some final state.

Equivalence of DFA and NDFA

- A DFA can simulate the behaviour of NDFA by increasing the number of states
- DFA (Q, Σ, δ, q_0, F) can be viewed as NDFA (Q, Σ, δ', q_0, F)
- Any NDFA is a more general machine without being more powerful.

  $\implies$ For every NDFA, there exists a DFA which simulates the behaviour of NDFA. **Alternatively, if L is a set accepted by NDFA, then there exists a DFA which also accepts L.**

Example

Contruct a deterministic automaton equivalent to M=($\{q_0, q_1\}$, $\{0, 1\}$, δ, q_0, $\{q_0\}$) where δ is defined as under:

State	Input	
	0	1
$\rightarrow$ (q_0)	q_0	q_1
q_1	q_1	q_0, q_1

NDFA → DFA

Example 1

Contruct a deterministic automaton equivalent to M=($\{q_0, q_1\}$, $\{0, 1\}$, δ, q_0, $\{q_0\}$)
where δ is defined as under:

State	Input	
	0	1
$\rightarrow$ (q_0)	q_0	q_1
q_1	q_1	q_0, q_1

Solution: For the deterministic automaton M_1:

- The states are subsets of $\{q_0, q_1\}$
  $\implies \phi$, $[q_0]$, $[q_1]$, $[q_0, q_1]$
- $[q_0]$ is initial state.
- $[q_0]$ and $[q_0, q_1]$ are final states as these are the only states containing q_0
- δ is defined by state table as under:

State	Input	
	0	1
$[\phi]$	$[\phi]$	$[\phi]$
$[q_0]$	$[q_0]$	$[q_1]$
$[q_1]$	$[q_1]$	$[q_0, q_1]$
$[q_o, q_1]$	$[q_o, q_1]$	$[q_o, q_1]$

NDFA $\rightarrow$ DFA

Example 2

Find a deterministic acceptor equivalent to:
M= $(\{q_0, q_1, q_2\}, \{a,b\}, \delta, q_0, \{q_2\})$
where δ is given by

State	Input	
	a	b
$\rightarrow q_0$	q_0, q_1	q_2
q_1	q_0	q_1
(q_2)	ϕ	q_0, q_1

Solution: The deterministic automaton M_1 equivalent to M is defined as follows:
$M_1 = (2^Q, \{a,b\}, \delta, [q_0], F')$
where, $F' = \{[q_2], [q_0, q_2], [q_1, q_2], [q_0, q_1, q_2]\}$

State	Input	
	a	b
$[q_0]$	$[q_0, q_1]$	$[q_2]$
$[q_1]$	$[q_0]$	$[q_1]$
$[q_2]$	ϕ	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_1, q_2]$
$[q_1, q_2]$	$[q_0]$	$[q_0, q_1]$

NDFA $\rightarrow$ DFA

Example 3

Construct a deterministic finite automaton equivalent to
M=($\{q_0, q_1, q_2, q_3\}$,{a,b},δ,q_0,$\{q_3\}$)
where δ is as under:

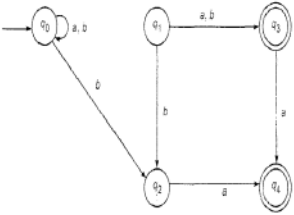
State	Input	
	a	b
$\rightarrow q_0$	q_0, q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_3
(q_3)		q_2

Solution: Let Q = $\{q_0, q_1, q_2, q_3\}$, then the deterministic automaton M_1 equivalent to M is given by M_1=(2^Q,{a,b},δ,[q_0],F)
where, F consists of:
$\{[q_3],[q_0, q_3],[q_1, q_3],[q_2, q_3],[q_0, q_1, q_3],[q_0, q_2, q_3],[q_1, q_2, q_3],[q_0, q_1, q_2, q_3]\}$
and δ is defined by state table as under:

State	Input	
	a	b
$[q_0]$	$[q_o, q_1]$	$[q_0]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_1]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_3]$
$[q_0, q_1, q_3]$	$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$
$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$

NDFA → DFA

1. Construct a DFA equivalent to NDFA 'M' whose transition diagram is given as:



2. Construct a DFA equivalent to NDFA with initial state q_0 whose transition table is defined as

State	a	b
q_0	q_1, q_3	q_2, q_3
q_1	q_1	q_3
q_2	q_3	q_2
(q_3)	—	—

Constructing required DFA

Example 1

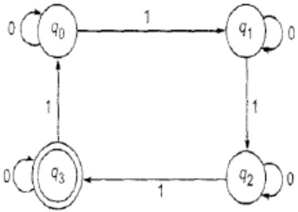
Construct a DFA accepting all strings 'w' over $\{0,1\}$ such that the number of 1's in 'w' is 3mod4.

Solution: Let the required DFA, as the condition on strings of T(M) doesn't at all involve 0,

$\Longrightarrow$ M doesnot change the state on input 0.

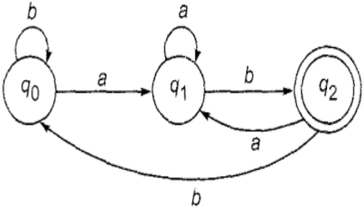
If 1 appears in w (4k+3) times, M can come back to initial state, after reading 4 1's and to a final state after reading 3 1's.

The required DFA:



Constructing required DFA

1. Construct a DFA accepting all strings over {a,b} ending in *ab*.



2. Construct a DFA equivalent to NDFA for:

State	Input		
	0	1	$\wedge$
$\rightarrow q_0$	q_0, q_3	q_0, q_1	
q_1	q_2		
q_2	q_2	q_2	q_4
q_3	q_4		
$\circled{q_4}$	q_4	q_4	

Constructing required DFA

1. M=($\{q_1, q_2, q_3\}$,{0,1}, δ,q_1,$\{q_3\}$) is a NDFA where δ is given by:
   $\delta(q_1, 0)=\{q_2, q_3\}$
   $\delta(q_1, 1)=\{q_1\}$
   $\delta(q_2, 0)=\{q_1, q_2\}$
   $\delta(q_2, 1)=\phi$
   $\delta(q_3, 0)=\{q_2\}$
   $\delta(q_3, 1)=\{q_1, q_2\}$
   Construct equivalent DFA.

Finite Automata with Outputs

- Moore Machine is a 6-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$
  where Q is a finite set of states
  Σ is the input alphabet
  Δ is the output alphabet
  δ is the transition function $Q \times \Sigma$ **into Q**
  λ is the output function **Q into** Δ
  q_0 is the initial state

Example:

Initial state q_0 is marked with an arrow. The table defines δ and λ:

Present State	Next State a=0	Next State a=1	Output λ
$\rightarrow$ (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

Determine transition states and output string for input string 0111.
Solution: Transition states:
$q_0 \xrightarrow{0\backslash 0} q_3 \xrightarrow{1\backslash 0} q_0 \xrightarrow{1\backslash 0} q_1 \xrightarrow{1\backslash 1} q_2$ 0
OutputString: 00010

Finite Automata with Outputs

- Mealy Machine is a 6-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$
  where Q is a finite set of states
  Σ is the input alphabet
  Δ is the output alphabet
  δ is the transition function $Q \times \Sigma$ **into Q**
  λ is the output function mapping **Q** $\times \Sigma$ **into** Δ
  q_0 is the initial state

Example:

Consider a mealy machine for q_1 as initial state.

Present State	a=0 State	a=0 Output	a=1 State	a=1 Output
$\rightarrow q_1$	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

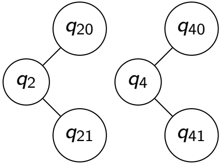
Determine the transition of states and corresponding output string for input string 0011.

Solution: $q_1 \xrightarrow{0\backslash 0} q_3 \xrightarrow{0\backslash 1} q_2 \xrightarrow{1\backslash 0} q_4 \xrightarrow{1\backslash 0} q_3$
Output String: 0100

Procedure of transforming Mealy machine into Moore machine

- Consider the mealy machie described by given transition table. Construct a moore machine which is equivalent to given mealy machine.

Present State	a=0 State	a=0 Output	a=1 State	a=1 Output
$\rightarrow q_1$	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



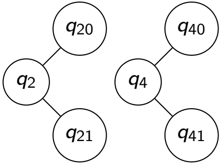
- **Solution:**

Present State	a=0 State	a=0 Output	a=1 State	a=1 Output
$\rightarrow q_1$				
q_{20}				
q_{21}				
q_3				
q_{40}				
q_{41}				

Procedure of transforming Mealy machine into Moore machine

- Consider the mealy machie described by given transition table. Construct a moore machine which is equivalent to given mealy machine.

Present State	a=0 State	a=0 Output	a=1 State	a=1 Output
$\rightarrow q_1$	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



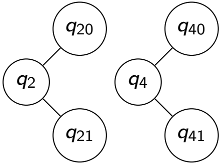
- **Solution:**

Present State	a=0 State	a=0 Output	a=1 State	a=1 Output
$\rightarrow q_1$	q_3	0	q_{20}	0
q_{20}	q_1	1	q_{40}	0
q_{21}	q_1	1	q_{40}	0
q_3	q_{21}	1	q_1	1
q_{40}	q_{41}	1	q_3	0
q_{41}	q_{41}	1	q_3	0

Procedure of transforming Mealy machine into Moore machine

- Consider the mealy machie described by given transition table. Construct a moore machine which is equivalent to given mealy machine.

Present State	a=0		a=1	
	State	Output	State	Output
$\rightarrow q_1$	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



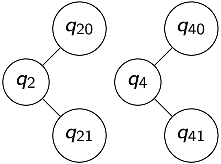
- **Solution:**

Present	a=0	a=1	Output
q_1	q_3	q_{20}	
q_{20}	q_1	q_{40}	
q_{21}	q_1	q_{40}	
q_3	q_{21}	q_1	
q_{40}	q_{41}	q_3	
q_{41}	q_{41}	q_3	

Procedure of transforming Mealy machine into Moore machine

- Consider the mealy machie described by given transition table. Construct a moore machine which is equivalent to given mealy machine.

Present State	a=0		a=1	
	State	Output	State	Output
$\rightarrow q_1$	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



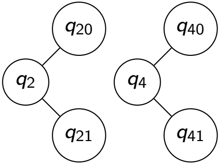
- **Solution:**

Present	a=0	a=1	Output
q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	
q_{21}	q_1	q_{40}	
q_3	q_{21}	q_1	
q_{40}	q_{41}	q_3	
q_{41}	q_{41}	q_3	

Procedure of transforming Mealy machine into Moore machine

- Consider the mealy machie described by given transition table. Construct a moore machine which is equivalent to given mealy machine.

Present	a=0		a=1	
State	State	Output	State	Output
$\rightarrow q_1$	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



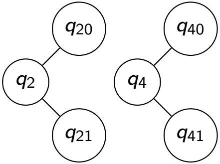
- **Solution:**

Present	a=0	a=1	Output
q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	
q_3	q_{21}	q_1	
q_{40}	q_{41}	q_3	
q_{41}	q_{41}	q_3	

Procedure of transforming Mealy machine into Moore machine

- Consider the mealy machie described by given transition table. Construct a moore machine which is equivalent to given mealy machine.

Present State	a=0 State	a=0 Output	a=1 State	a=1 Output
$\rightarrow q_1$	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



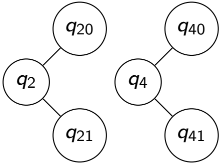
- **Solution:**

Present	a=0	a=1	Output
q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	
q_{40}	q_{41}	q_3	
q_{41}	q_{41}	q_3	

Procedure of transforming Mealy machine into Moore machine

- Consider the mealy machie described by given transition table. Construct a moore machine which is equivalent to given mealy machine.

Present State	a=0 State	a=0 Output	a=1 State	a=1 Output
$\rightarrow q_1$	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



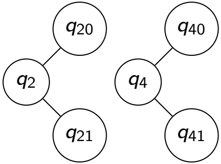
- **Solution:**

Present	a=0	a=1	Output
q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	
q_{41}	q_{41}	q_3	

Procedure of transforming Mealy machine into Moore machine

- Consider the mealy machie described by given transition table. Construct a moore machine which is equivalent to given mealy machine.

Present State	a=0 State	a=0 Output	a=1 State	a=1 Output
$\rightarrow q_1$	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



- **Solution:**

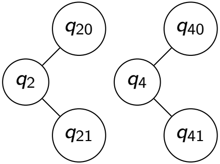
Present	a=0	a=1	Output
q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	

Procedure of transforming Mealy machine into Moore machine

- Consider the mealy machie described by given transition table. Construct a moore machine which is equivalent to given mealy machine.

Present State	a=0 State	a=0 Output	a=1 State	a=1 Output
$\rightarrow q_1$	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

- **Solution:**

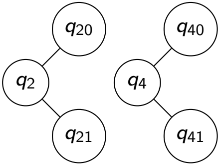


Present	a=0	a=1	Output
q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

Procedure of transforming Mealy machine into Moore machine

- Consider the mealy machie described by given transition table. Construct a moore machine which is equivalent to given mealy machine.

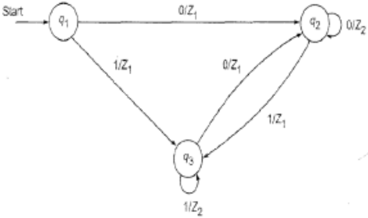
Present	a=0		a=1	
State	State	Output	State	Output
$\rightarrow q_1$	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0



- **Solution:**

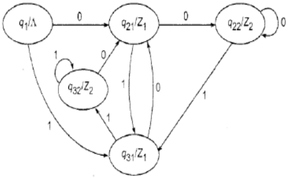
Present	a=0	a=1	Output
$\rightarrow q_0$	q_3	q_{20}	0
q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

Convert the given mealy machine into equivalent moore machine

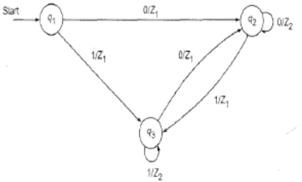


Present state	Next state			
	a = 0		a = 1	
	state	output	state	output
$\to q_1$	q_2	Z_1	q_3	Z_1
q_2	q_2	Z_2	q_3	Z_1
q_3	q_2	Z_1	q_3	Z_2

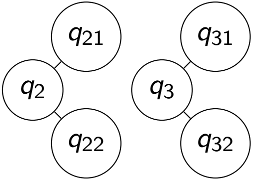
Present state	Next state		Output
	a = 0	a = 1	
$\to q_1$	q_{21}	q_{31}	
q_{21}	q_{22}	q_{31}	Z_1
q_{22}	q_{22}	q_{31}	Z_2
q_{31}	q_{21}	q_{32}	Z_1
q_{32}	q_{21}	q_{32}	Z_2



Procedure of transforming Mealy machine into Moore machine



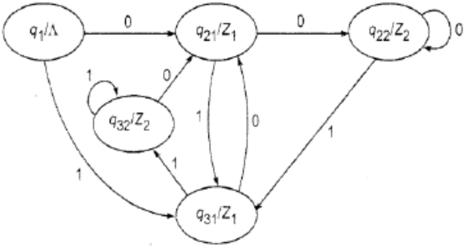
Present State	a=0 state	a=0 Output	a=1 State	a=1 Output
$\rightarrow q_1$	q_2	Z_1	q_3	Z_1
q_2	q_2	Z_2	q_3	Z_1
q_3	q_2	Z_1	q_3	Z_2



Present State	a=0 state	a=0 Output	a=1 State	a=1 Output
q_1	q_{21}	Z_1	q_{31}	Z_1
q_{21}	q_{22}	Z_2	q_{31}	Z_1
q_{22}	q_{22}	Z_2	q_{31}	Z_1
q_{31}	q_{21}	Z_1	q_{32}	Z_2
q_{32}	q_{21}	Z_1	q_{32}	Z_2

Procedure of transforming Mealy machine into Moore machine

Present State	Next State		Output
	a=0	a=1	
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



Procedure of transforming Moore machine into Mealy machine

- Consider the moore machine described by the transition table given:

Present State	Next State a=0	Next State a=1	Output
$\rightarrow q_1$	q_1	q_2	0
q_2	q_1	q_3	0
q_3	q_1	q_3	1

Construct the corresponding mealy machine.

- **Solution:**

Present State	a=0 state	a=0 Output	a=1 State	a=1 Output
$\rightarrow q_1$	q_1	0	q_2	0
q_2	q_1	0	q_3	1
q_3	q_1	0	q_3	1

Now, Find identical rows and remove one of them

Present State	a=0 state	a=0 Output	a=1 State	a=1 Output
$\rightarrow q_1$	q_1	0	q_2	0
q_2	q_1	0	q_2	1

Minimization of Finite Automata

- Equivalence: Two states q_1 and q_2 are equivalent(denoted by $q_1 \equiv q_2$), if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states or both of them are non-final states for all x $\in \Sigma^*$.
- Precisely, Two states q_1 and q_2 are k-equivalent (k $\geq$ 0), if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states *or* both non-final states for all string x of length k or less.
- If $\delta(q_1, w)$ and $\delta(q_2, w)$ are equivalent then
  - for $|$w $|=$ 0, the states are 0-equivalent.
  - for $|$w $|=$ 1, the states are 1-equivalent.
  - for $|$w $|=$ 2, the states are 2-equivalent.

    .

    ..
  - for $|$w $|=$ n, the states are n-equivalent.
- Properties of Equivalence relations:
  - If a relation is equivalence or k-equivalence, then they are reflexive, symmetric and transitive.
  - If q_1 and q_2 are k-equivalent for all k $\geq$ 0, then they are equivalent.
  - If q_1 and q_2 are (k+1)-equivalent, then they are k-equivalent.

Minimization of Finite Automata

- Construct a minimum state automaton equivalent to the given finite automaton

Solution:

1. Draw transition table

State $\backslash\Sigma$	0	1
$\rightarrow q_0$	q_1	q_5
q_1	q_6	q_2
q_2	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

Minimization of Finite Automata

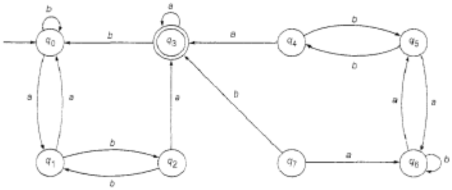
2. Find 0-equivalent set
   π_0=$[q_0, q_1, q_3, q_4, q_5, q_6, q_7][q_2]$
3. Find 1-equivalent set
   π_1=$[q_0, q_6, q_4][q_1, q_7][q_5, q_3][q_2]$
4. find 2-equivalent set
   π_2=$[q_0, q_4][q_6][q_2][q_1, q_7][q_3, q_5]$
5. find 3-equivalent set
   π_3=$[q_0, q_4][q_6][q_2][q_1, q_7][q_3, q_5]$

Therefore, M'=(Q',{ 0,1 } ,δ,q_0',F')
where Q'={$[q_2]$,$[q_0, q_4]$,$[q_6]$,$[q_1, q_7]$,$[q_3, q_5]$}
q_0'=$[q_0, q_4]$, F'=$[q_2]$

State $\backslash\Sigma$	0	1
$[q_0, q_4]$	$[q_1, q_7]$	$[q_3, q_5]$
$[q_1, q_7]$	$[q_6]$	$[q_2]$
$[q_2]$	$[q_0, q_4]$	$[q_2]$
$[q_3, q_5]$	$[q_2]$	$[q_6]$
$[q_6]$	$[q_6]$	$[q_0, q_4]$

Minimization of Finite Automata

- Construct a minimum state automaton equivalent to the given finite automaton



- **Solution:**

State \ Σ	a	b
$\rightarrow q_0$	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
(q_3)	q_3	q_0
q_4	q_3	q_5
q_5	q_6	q_4
q_6	q_5	q_6
q_7	q_6	q_3

Minimization of Finite Automata

- π_0={{q_3}{$q_0, q_1, q_2, q_4, q_5, q_6, q_7$}}
- π_1={{q_3}{q_0, q_1, q_5, q_6}{q_2, q_4},{q_7}}
- π_2={{q_3}{q_0, q_6}{q_1, q_5}{q_2, q_4}{q_7}}
- π_3={{q_3}{q_0, q_6}{q_1, q_5}{q_2, q_4}{q_7}}

$\therefore$ Q'= {{q_3}{q_0, q_6}{q_1, q_5}{q_2, q_4}{q_7}}
q_0'={q_0, q_6}
F'={q_3}
Now, make δ' and transition diagram.

Minimization of Finite Automata

- Construct minimum state automaton equivalent to given automata M:

State $\backslash\Sigma$	a	b
$\rightarrow q_0$	q_0	q_3
q_1	q_2	q_5
q_2	q_3	q_4
q_3	q_0	q_5
q_4	q_0	q_6
q_5	q_1	q_4
$\textcircled{$q_6$}$	q_1	q_3

Language: Introduction

- Language
  - Formal (Syntactic Languages)
  - Informal (Semantic Languages)
- Alphabet
  - String A concatenation of finite symbols from the alphabet is called a string.
    Example: If Σ={a,b} then a, abab, aaabb, abababababaaaaaabaab, etc.
  - Empty String or Null String $\wedge$ or Λ or ϕ
    $\implies$ A string with no symbols
- Words $\implies$ strings belonging to some language
  Example: If Σ={x} then a language L can be defined as
  L={x^n:n=1,2,3,....} or
  L={x,xx,xxx,....}
  Here, x, xx, xxx ,.... are the words of L.
- All words are strings but not all strings are words.

Language: Introduction

- Length of String: |S| $\Longrightarrow$ number of letters in the string.
  Example: Σ={a,b}
  If S= ababa, then |S|=5
- Reverse of String: S^r $\Longrightarrow$ Obtained by writing letters of 'S' in reverse order.
  Example:
  - If s=abc over Σ={a,b,c}
    Then, Rev(s) or s^r=cba
  - Σ={B,aB,bab,d}
    s=BaBbabBd
    s^r=dBbabaBB

Language: Definition

- Descriptive Definition
- Recursive Definition
- Using Regular Expression
- Using Finite Automata, etc.

Descriptive definition of Language

The language is defined describing the conditions imposed on its words.
Example:

1. The language L of strings of odd length, defined over Σ={a}
   $\implies$ L={a, aaa,aaaaa, ...}
2. The language L of strings that doesnot start with 'a' defined over Σ={a,b,c}
   $\implies$ L={b,c,ba, bb,bc,ca,cb,cc}
3. The language L of strings of length 2 over Σ={0,1,2}
   $\implies$ L={00,01,02,10,22,12,...}
4. The language L of strings ending in 0 over Σ={0,1}
   L={0,00,10,000,010,100,...}
5. The language L of Strings with number of "a"(s) equal to number of "b"(s) over Σ={a,b}
   L={∧, ab,aabb,abab,baba,abba,....}
6. Language **Even-Even** of string with even number of a(s) and even number of b(s) over Σ={a,b}
   L={∧, aa,bb,aaaa,aabb,abab,...}
7. Language Integer of strings over Σ=-,0,1,2,3,4,5,6,7,8,9
   L={......, -2,-1,0,1,2,....}
8. Language $\{a^n b^n\}$ over Σ={a,b} or $\{a^n b^n$:n=1,2,3,....}
   L={ab,aabb,aaabbb, ...}
9. Palindrome over Σ={a,b}
   L={∧, a,b, aa,bb, aaa,aba,bab,bbb,....}

GRAMMAR

- A grammar is (V_N, Σ, P, S)
  where V_N is a non-empty set whose elements are called variables.
  Σ finite non-empty set whose elements are called terminals.
  S is aspecial symbol called Start Symbol.
  P are set of Production rules.
- $V_N \cap \Sigma = \phi$

Example

G=$\{V_N, \Sigma, P, S\}$ is a Grammar, where
V_N=\{<sentence>, <noun>, <verb>, <adverb>\}, Σ=\{Ram,Sam,ran,sang,fast\},
S=<sentence>
P consists of following productions:
<sentence>→<noun><verb>
<sentence>→<noun><verb><adverb>
<noun>→ Ram
<noun>→ Sam
<verb>→ ran
<verb>→ sang
<adverb>→ fast

Grammar

- If G=({S},{0,1},{S$\rightarrow$ 0S1,S$\rightarrow \wedge$},S). Find L(G).
  Solution: S$\rightarrow$0S1 $\rightarrow$ 00S11 $\rightarrow$ 000S111.....$0^n S 1^n$
  $0^n \wedge 1^n = 0^n 1^n \in$ L(G) for n $\geq$ 0
  $\therefore$ L(G)={$0^n 1^n$ |n $\geq$ 0}
- If G=({S},{a},{S $\rightarrow$ SS},S), Find the language generated by G.
  Solution: L(G)=ϕ
- Let G=({S,C},{a,b},P,S), where P consists of S $\rightarrow aCa$, $C \rightarrow aCa$ |b. Find L(G).
  Solution: S $\Longrightarrow$ aCa $\Longrightarrow$ aba . So, aba $\in$ L(G)
  S $\Longrightarrow$ aCa *(using S$\rightarrow$ aCa)*
  $\Longrightarrow$ $a^n C a^n$ *(using S$\rightarrow$ aCa (n-1) times)*
  $\Longrightarrow$ $a^n b a^n$ *(using C $\rightarrow$ b)*
  Hence, $a^n b a^n \in$ L(G), where n$\geq$ 1
  $\therefore$ L(G)={$a^n b a^n$ |n $\geq$ 1}

Grammar

Exercise

Construct a grammar G so that L(G) = $\{a^n ba^m \mid n, m \geq 1\}$

- If G is S $\rightarrow$ aS |bS |a |b, Find L(G).
  Solution: L(G)=$\{a, b\}^+$

Exercise 1

If G is S $\rightarrow$ aS |a, then show that L(G)=$\{a\}^+$

- Let L be the set of all palindromes over {a,b}. Construct a grammar G generating L.
  Solution: $\wedge$, a, b, or axa and bxb are palindromes.
  $\therefore$ P consists of
  S $\rightarrow \wedge$
  S $\rightarrow$ a, S $\rightarrow$ b
  S $\rightarrow$ aSa, S $\rightarrow$ bSb
  Thus, G=({S},{a,b},P,S)

Grammar

- Construct a Grammar generating L=$\{wcw^T \mid w \in \{a, b\}^*\}$
  Solution: Let G=({S},{a,b,c},P,S)
  where P is defined as
  S $\rightarrow$ aSa |bSb |c
- Find a grammar generating L=$\{a^n b^n c^i \mid n \geq 1, i \geq 0\}$
  Solution: L=$L_1 \cup L_2$, $L_1=\{a^n b^n \mid n \geq 1\}$
  $L_2=\{a^n b^n c^i \mid n \geq 1, i \geq 1\}$
  Let "P" be as follows:
  S $\rightarrow$ A
  A $\rightarrow$ ab |aAb
  S $\rightarrow$ Sc
  Let G=({S,A}, {a,b,c}, P,S) for n $\geq$ 1, i $\geq$ 0
  $S \xrightarrow{*} Sc^i \rightarrow Ac^i \rightarrow a^{n-1}Ab^{n-1}c^i \rightarrow a^{n-1}abb^{n-1}c^i = a^n b^n c^i$
  L(G)=$\{a^n b^n c^i \mid n \geq 1, i \geq 0\}$

Grammar

- Find a grammar generating
  $\{a^j b^n c^n \mid n \geq 1, j \geq 0\}$
  Solution: Let G=({S,A},{a,b,c},P,S)
  where "P" consists of:
  S $\rightarrow$ aS |A
  A $\rightarrow$ bAc |bc
- Let G =({S,A},{0,1,2},P,S) where P consists of
  S $\rightarrow$ 0SA2 |S $\rightarrow$ 012
  2A $\rightarrow$ A2
  1A $\rightarrow$ 11. Show that L(G)=$\{0^n 1^n 2^n \mid n \geq 1\}$
  Solution: S $\xrightarrow{*} 0^{n-1}S(A2)^{n-1}$ by applying S $\rightarrow$ 0SA2 (n-1) times
  $\rightarrow 0^n 12(A2)^{n-1}$ by applying S $\rightarrow$ 012
  $\xrightarrow{*} 0^n 1 A^{n-1} 2^n$ by applying 2A $\rightarrow$ A2 several times
  $\xrightarrow{*} 0^n 1^n 2^n$ by applying 1A $\rightarrow$ 11 (n-1 times)
  $\therefore 0^n 1^n 2^n \in$ L(G) for all n $\geq$ 1

Grammar

- Construct grammar G generating $\{a^n b^n c^n \mid n \geq 1\}$
  Solution: G=({S,B,C},{a,b,c},P,S)
  where P consists of: S $\rightarrow$ aSBC |aBC, CB $\rightarrow$ BC, aB $\rightarrow$ ab,
  bB $\rightarrow$ bb, bC $\rightarrow$ bc, cC $\rightarrow$ cc
  S $\implies$ aBC $\implies$ abC $\implies$ abc
- Construct a grammar G generating $\{xx \mid x \in \{a, b\}^*\}$
  Solution: Let G is as follows: G=({S,D,E,F,A,B},{a,b},P,S)
  where P consists of :
  S $\rightarrow$ DEF
  DE $\rightarrow$ aDA, DE $\rightarrow$ bDB
  AF $\rightarrow$ EaF, BF $\rightarrow$ EbF
  Aa $\rightarrow$ aA, Ab $\rightarrow$ bA, Ba $\rightarrow$ aB, Bb$\rightarrow$bB
  aE $\rightarrow$ Ea, bE$\rightarrow$ Eb
  DE $\rightarrow$ $\wedge$, F $\rightarrow$ $\wedge$

Grammar

- Let G=({S,A,B},{a,b},P,S) where P consists of
  $S \rightarrow aABa$, $A \rightarrow baABb$, $B \rightarrow Aab$, $aA \rightarrow baa$, $bBb \rightarrow abab$.
  Test whether w = baabbabaaabbaba is in L(G).
  Solution: $S \rightarrow \underline{aA}Ba$
  $\Longrightarrow baa\underline{B}a$
  $\Longrightarrow baa\underline{A}aba$
  $\Longrightarrow baab\underline{aA}Bbaba$
  $\Longrightarrow baabbaa\underline{B}baba$
  $\Longrightarrow baabba\underline{aA}abbaba$
  $\Longrightarrow baabbabaaabbaba = w$
  $\therefore w \in L(G)$
- If the grammar G is given by the productions $S \rightarrow aSa \mid bSb \mid aa \mid bb \mid \wedge$, show that:
  - L(G) has no strings of odd length
  - Any string in L(G) is of length 2n, $n \geq 0$
  - The number of strings of length 2n is 2^n

Chomsky Classification of Languages

- Chomsky classified grammar into 4 types i.e. (type 0-3)
- A type 0 grammar is any phrase structure grammar without any restrictions
  $\implies$ All grammar we have considered till now are type 0 grammar
- In a production of form $\phi A\psi \to \phi\alpha\psi$
  Example:
  - **ab**A**bcd** $\to$ **ab**AB**bcd**
    $\phi \implies ab$
    $\alpha \implies AB$
    $\psi \implies bcd$
  - **A**C $\to$ A
    $\phi \implies A$
    $\alpha \implies \wedge$
    $\psi \implies \wedge$

Chomsky Classification of Languages

- C $\to \wedge$
  $\phi \implies \wedge$
  $\alpha \implies \wedge$
  $\psi \implies \wedge$

Chomsky Classification of Languages

- A production of the form $\phi A\ \psi \to \phi\alpha\psi$ is called Type-1 production or Context-sensitive Language, if $\alpha \neq \wedge$
  $\implies$ In type-1 production erasing 'A' is not allowed
  Example:
  - aAbCD → abcDbcD is a type 1 production.
    A is replaced by bcD $\neq \wedge$
  - AB → AbBc is a type 1 production.
  - A → abA is a type 1 production.
- A grammar is called type 1 or Context sensitive or Context-dependent if all its productions are type 1 productions.
- The production S → $\wedge$ is also allowed in type 1 grammar, but in this case S does not appear on the right-hand side of any production.
- The language generated by a type-1 grammar is called a type-1 or context-sensitive language

Chomsky Classification of Languages

- A grammar G= $(V_N, \Sigma$,P,S) is monotonic (or length-increasing) if every production in P is of the form $\alpha \to \beta$ with $|\alpha| \leq |\beta|$or S $\to \wedge$. In second case, S does not appear on right-hand side of any production in P.
- Type-2: Context free Grammar generates context free language
- A Type-2 production is a production of the form A $\to \alpha$ where A $\in V_N$ and $\alpha \in (V_N \vee \Sigma)^*$
- In other words, in Type-2 $\Longrightarrow$
  - It should be in Type-1
  - L.H.S. production should have only 1 variable i.e. $|A| = 1$ and there is no restriction on α
    Example: S $\to$ Aa, A $\to$a, B $\to$ abc, A $\to \wedge$ are type-2 productions.

Chomsky Classification of Languages

- A production of the form A $\rightarrow$ a or A $\rightarrow$ aB, where A,B $\in V_N$ and a$\in \Sigma$ is called a type-3 production.
- A grammar is called a type-3 or **Regular Grammar** if all its productions are type-3 productions.
- A production S $\rightarrow \wedge$ is allowed in type-3 grammar, but in this case S does not appear on the right-hand side of any production

Chomsky Classification of Languages

1. Find the highest type number which can be applied to the following productions:
   - $S \rightarrow Aa$, $A \rightarrow c \mid Ba$, $B \rightarrow abc$
   - $S \rightarrow ASB \mid d$, $A \rightarrow aA$
   - $S \rightarrow aS \mid ab$
2. Differentiate between Recursive Set and Recursively Enumerable Set
3. Prove that Context-sensitive language is recursive.
4. Prove that there exists a recursive set which is not a contxt-sensitive language over $\{0,1\}$.
5. Let $G=(\{A,B,S\},\{0,1\},P.S)$ where P consists of $S \rightarrow 0AB$, $A0 \rightarrow S0B$, $A1 \rightarrow SB1$, $B \rightarrow SA$, $B \rightarrow 01$. Show that $L(G)= \phi$.
6. Find the language generated by grammar $S \rightarrow AB$, $A \rightarrow A1 \mid 0$, $B \rightarrow 2B \mid 3$. Can the above language be generated by a grammar of higher type?

Chomsky Classification of Languages

7. Construct a grammar which generates all even integer upto 998.
8. Construct CFG to generate the following:
   - $\{0^m 1^n \mid m \neq n,\ m,n \geq 1\}$
   - $\{a^l b^m c^n \mid$ one of l,m,n equals 1 and remaining tqo are equal$\}$
   - $\{a^l b^m c^n \mid l+m=n\}$
   - The set of all strings over {0,1} containing twice as many 0's and 1's
9. Show that G_1=({S},{a,b},P_1,S) where P_1={S $\rightarrow$ aSb |ab} is equivalent to G_2=({S,A,B,C},{a,b},P_2,S), where P_2 consists of S $\rightarrow$AC, C$\rightarrow$SB, S $\rightarrow$AB, A$\rightarrow$a,B $\rightarrow$b
10. What are the applications of different grammar types?

Regular Expression

- A language is regular if there exists a finite acceptor for it
  $\therefore$ Every regular language can be described as DFA or NDFA
- Regular Expression: Algebraic description of languages
- Let Σ be a given alphabet, then:
  1. $\phi, \wedge$ and a $\in \Sigma$ are all regular expressions, called **Primitive regular expressions**.
  2. If R_1 and R_2 are regular expressions, so are $R_1 + R_2, R_1.R_2, {R_1}^*$ and (R_1)
  3. A string is a regular expression if and only if it can be derived from primitive regular expressions by a finite number of applications of the rules in (2)
- **Example:** For Σ={a,b,c}, the string $(a + b.c)^*.(c + \phi)$ is a regular expression
  while $(a + b+)$ is not a regular expression.

Language Associated with Regular Expression

The language L(R) denoted by any regular expression 'R' is defined by following rules

1. ϕ is a R.E. denoting empty set
2. $\wedge$ is a R.E. denoting $\{\wedge\}$
3. For every $a \in \Sigma$, **a** is a R.E. denoting $\{\mathbf{a}\}$
   If R_1 and R_2 are R.E. , then
4. $L(R_1+R_2)=L(R_1)\vee L(R_2)$
5. $L(R_1.R_2)=L(R_1)L(R_2)$
6. $L((R_1))=L(R_1)$
7. $L(R_1^*)=(L(R_1))^*$

Example: For $\Sigma=\{a,b\}$, the expression
$R=(a+b)^*(a+bb)$ is regular
$\implies L(R)=\{a,bb,aa,abb,ba,bbb,....\}$
$\implies$ L(R) is the set of all strings on $\{a,b\}$, terminated by either 'a' or 'bb'.

Language Associated with Regular Expression

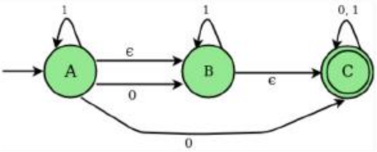
- R=$(aa)^*(bb)^*b$
  denotes the set of all strings with even number of a's followed by an odd number of b's
  L(R)=$\{a^{2n}b^{2m+1}$ |n≥0, m≥0}
- For Σ ={0,1}. Give a regular expression 'R' such that
  L(R)={w∈ Σ^*|w has atleast one pair of consecutive zeroes}
  Solution: R=$(0+1)^*00(0+1)^*$
- Find R.E. for language
  L=$\{w \in \{0,1\}^*$ |w has no pair of consecutive zeroes}
  Solution: R=$(1+01)^*(0+\wedge)$
  R=$(1^*011^*)^*(0+\wedge)+1^*(0+\wedge)$
- Find all strings in $L((a+b)^*b(a+ab)^*)$ of length less than 4
- Find R.E. for set $\{a^nb^m$:(n+m) is even}

Identities for Regular Expression

1. $\phi + \text{R} = \text{R}$
2. $\phi\text{R} = \text{R}\ \phi = \phi$
3. $\Lambda\text{R} = \text{R}\Lambda = \text{R}$
4. $\Lambda^* = \Lambda$ and $\phi^* = \Lambda$
5. $\text{R} + \text{R} = \text{R}$
6. $R^*R^* = R^*$
7. $\text{R}R^* = R^*\text{R}$
8. $(R^*)^* = R^*$
9. $\Lambda + \text{R}R^* = R^* = \Lambda + R^*R$
10. $(PQ)^*P = \text{P}(QP)^*$
11. $(P + Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$
12. $(P + Q)R = PR + QR$ and $R(P + Q) = RP + RQ$

ϵ-NFA

$\Longrightarrow$ Moving without reading a symbol from Input Tape.



State/Input	0	1	ϵ
A	B,C	A	B
B	-	B	C
C	C	C	-

Epsilon Closure

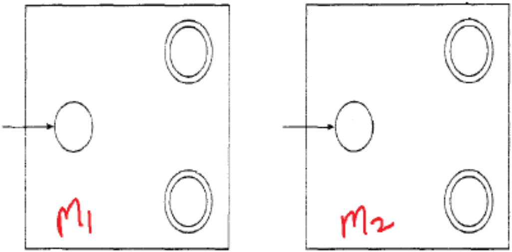
ϵ-closure for a given state X is a set of States which can be reached from states X with only (null) or ϵ moves including the state X itself.

Example: ϵ closure (A)={A,B,C}

ϵ closure (B)={B,C}

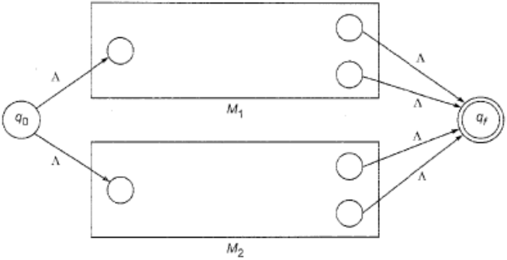
ϵ closure (C)={C}

Automata and Regular Expression

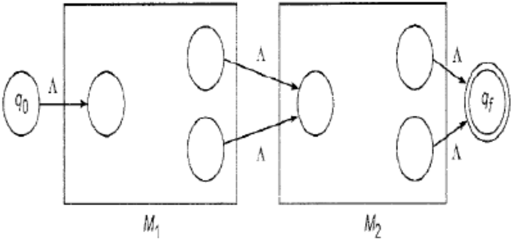


- Automaton for $L(R_1 + R_2)$

Automata and Regular Expression

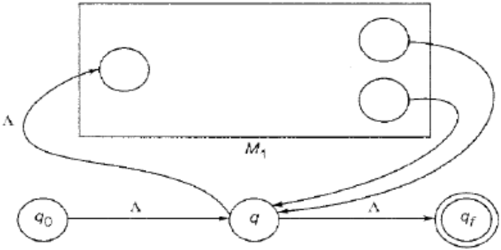


- Automaton for $L(R_1.R_2)$



Automata and Regular Expression

- Automaton for L($R_1{}^*$)

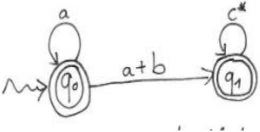


Automata and Regular Expression

- Generalized Transition Graph: A transition graph whose edges are labelled as R.E.
- **Example:** L(R)=$(a^* + a^*(a+b)c^*)$
- Equivalence of Generalized Transition Graph:
  Let R be a regular expression. Then, there exists some NFA that accepts L(R). Consequently, L(R) is a regular language.
- Find NDFA which accepts L(R) where R=$(a+bb)^*(ba^*+\Lambda)$

Automata and Regular Expression

- The Strings denoted by such regular expressions are a subset of the language accepted by GTG, with full language being the union of all such generated subsets
- **Example:** The language accepted by the following GTG is



$\mathrm{L}(a^* + a^*(a+b)c*)$

- State Elimination method
- Arden's Theorem

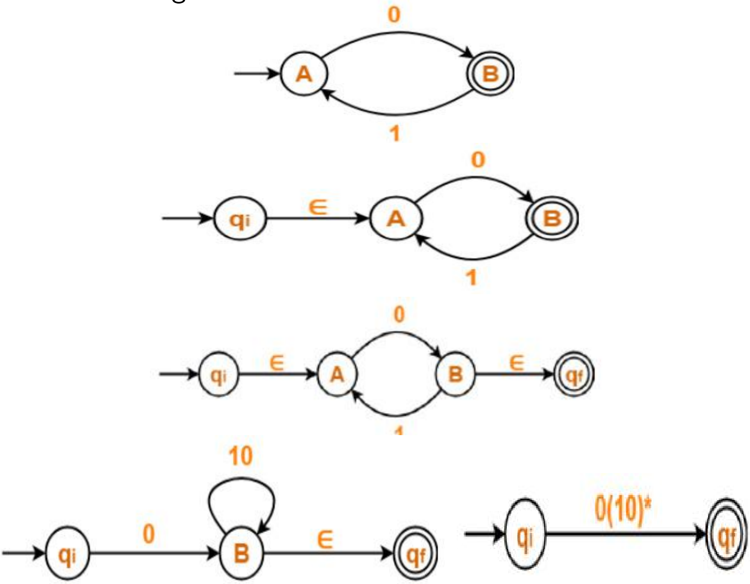
Automata and Regular Expression

State Elimination method

1. Initial State should not have any incoming edge
2. Final State should not have any outgoing edge
3. Only 1 final state
4. Eliminate each non-initial/final vertex one by one

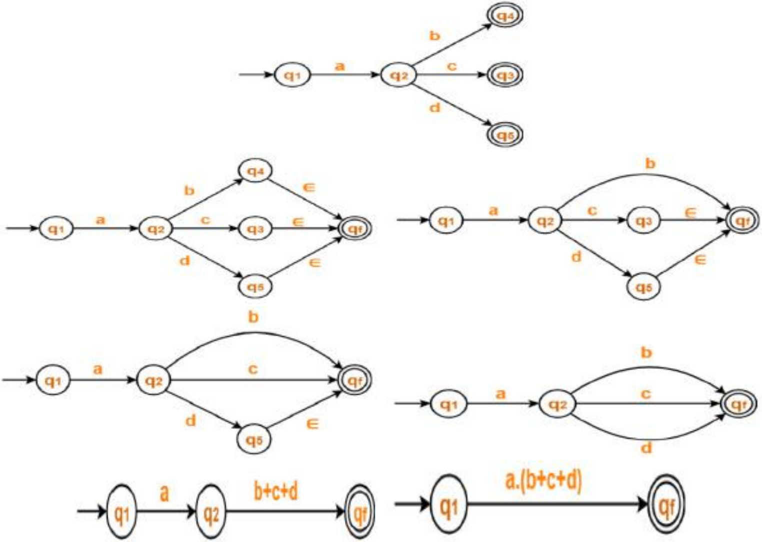
State Elimination Method

1. Find R.E. for given DFA



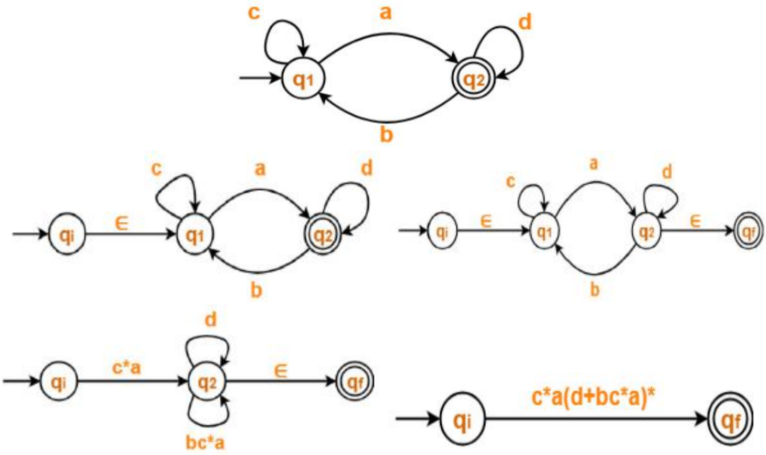
State Elimination Method

2. Find R.E. for given DFA



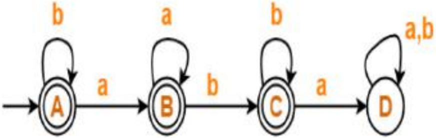
State Elimination Method

3. Find R.E. for given DFA

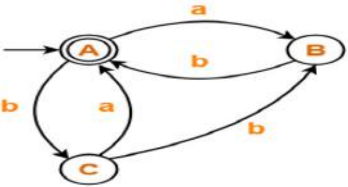


State Elimination Method

1. Find R.E. for the given DFA



2. Find R.E. for the given DFA



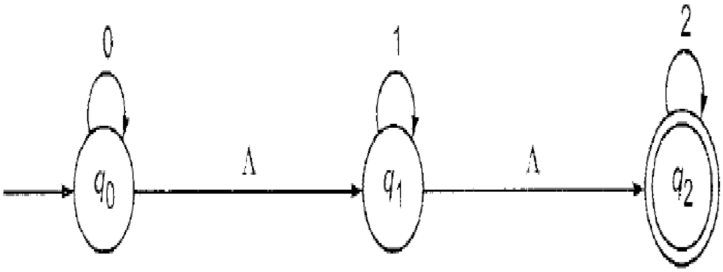
Elimination of $\epsilon/\wedge$ moves

Suppose we want to replace a $\wedge$-move from vertex v_1 to vertex v_2
Then proceed as follows:

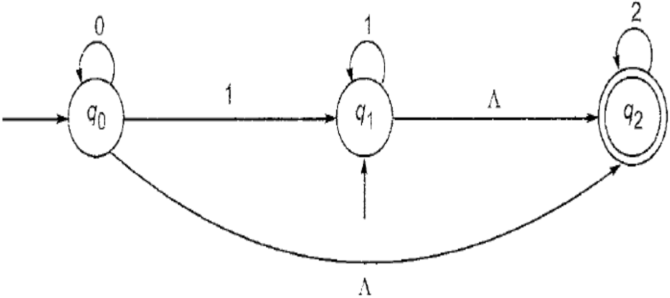
1. Find all the edges starting from v_2
2. Duplicate all these edges starting from v_1, without changing the edge labels.
3. If v_1 is an initial state, make v_2 also as initial state
4. If v_2 is a final state. make v_1 also as the final state

Elimination of $\epsilon/\wedge$ moves

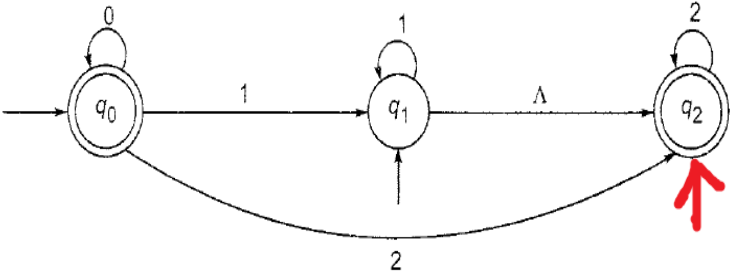
Example:



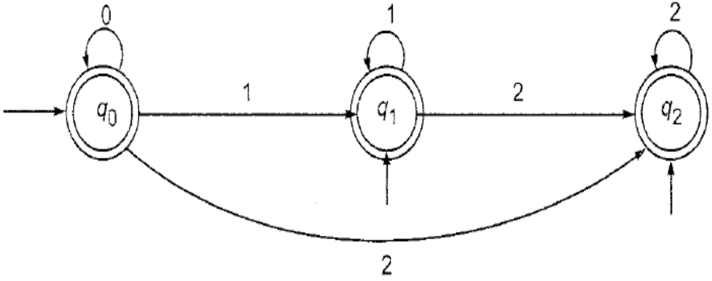
Elimination of $\epsilon/\wedge$ moves



Elimination of ϵ/Λ moves

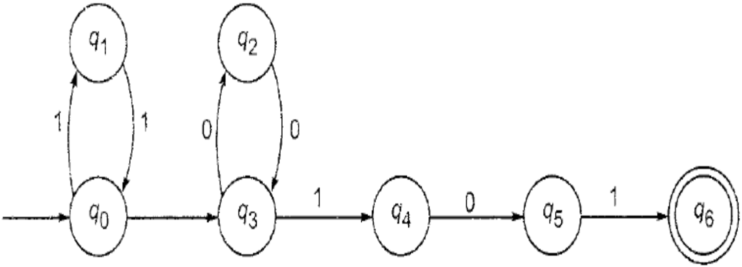


Elimination of $\epsilon/\wedge$ moves



Elimination of $\epsilon/\wedge$ moves

Example:



Conversion of ϵ-NFA to DFA

Steps:

1. Take ϵ closure of initial state as beginning state
2. Find states that can be traversed from present state for each input symbol
3. If any new state is found, repeat step 2 till we get no new state in the transition table.
4. Mark states containing final states as new final state.

State/Input	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}	{C}	{C}

Now, create transition diagram

Arden's Theorem

Let P and Q be two regular expressions over Σ. **If P does not contain** Λ, then the following equation in R i.e. R=Q+RP has a unique solution R=QP^*

Proof:

$$R = Q + RP \tag{1}$$

putting "R=Q+RP" in equation 1
R=Q+QP+RPP
putting R recursively again and again
we get:
R=Q+QP+QP^2+QP^3+ ...
R= Q $(\Lambda + P + P^2 + P^3 + \ldots)$
R=QP^*

Arden's Theorem

Assumptions:

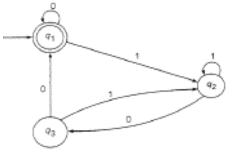
- The transition diagram must not have null transitions.
- It must only 1 initial state, let v_1
- Its vertices are $v_1, \ldots, v_n$.
- V_i, the R.E. represents the set of strings accepted by the system even though v_i, is a final state.
- α_{ij} denotes the R.E. representing the set of labels of edges from v_i to v_j. When there is no such edge, $\alpha_{ij} = \phi$. Consequently, we can get the following set of equations in $V_1, \ldots, V_n$:
  $V_1 = V_1\alpha_{11} + V_2\alpha_{21} + \cdots + V_n\alpha_{n1} + \wedge$
  $V_2 = V_1\alpha_{12} + V_2\alpha_{22} + \cdots + V_n\alpha_{n2}$
  $\vdots$
  $V_n = V_1\alpha_{1n} + V_2\alpha_{2n} + \cdots + V_n\alpha_{nn}$

Arden's Theorem

- By repeatedly applying substitutions and Arden's theorem, we can express V_i in terms of α_{ij}.
- For getting the set of strings recognized by the transition system, we have to take the "union" of all V_i corresponding to final states

Arden's Theorem

1. Find R.E. for the following DFA



$$q_1 = q_1 0 + q_3 0 + \wedge \quad (2)$$

$$q_2 = q_1 1 + q_2 1 + q_3 1 \quad (3)$$

$$q_3 = q_2 0 \quad (4)$$

putting equation 4 in equation 3

$$q_2 = q_1 1 + q_2 1 + q_3 1$$
$$= q_1 1 + q_2 1 + (q_2 0) 1$$
$$= q_1 1 + q_2 (1 + 01)$$

Arden's Theorem

Applying Arden's Theorem

$$q_2 = q_1 1(1+01)^* \tag{5}$$

putting 5 in equation 2

$$\begin{aligned} q_1 &= q_1 0 + q_3 0 + \wedge \\ &= q_1 0 + q_2 00 + \wedge \\ &= q_1 0 + (q_1 1(1+01)^* 00) + \wedge \\ & q_1(0 + 1(1+01)^* 00) + \wedge \end{aligned}$$

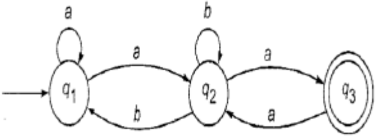
using arden's theorem again

$$\begin{aligned} q_1 &= \wedge(0+1(1+01)^*00)^* \\ & (0+1(1+01)^*00)^* \end{aligned}$$

As q_1 is the final state. $\therefore$ r$= (0+1(1+01)^*00)^*$

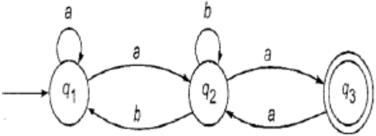
Arden's Theorem

- Consider the transition system below. Prove that the strings recognized are $(a + a(b + aa)^* b)^* a(b + aa)^* a$



Arden's Theorem

- Consider the transition system below. Prove that the strings recognized are $(a + a(b + aa)^* b)^* a(b + aa)^* a$



Solution:

$$q_1 = q_1 a + q_2 b + \wedge \tag{6}$$

$$q_2 = q_1 a + q_2 b + q_3 a \tag{7}$$

$$q_3 = q_2 a \tag{8}$$

Putting equation 8 in equation 7

$$q_2 = q_1 a + q_2 b + q_2 aa$$

$$q_2 = q_1 a + q_2(b + aa)$$

$$q_2 = q_1 a(b + aa)^*$$

Arden's Theorem

Now putting q_2 in equation 6

$$\begin{aligned} q_1 &= q_1 a + q_2 b + \wedge \\ &= q_1 a + q_1 a(b+aa)^* b + \wedge \\ &= q_1(a + a(b+aa)^* b) + \wedge \\ &= \wedge(a + a(b+aa)^* b)^* \\ &= (a + a(b+aa)^* b)^* \end{aligned}$$

putting this in q_2

$$\begin{aligned} q_2 &= (a + a(b+aa)^* b)^* a + q_2 b + q_2 aa \\ &= (a + a(b+aa)^* b)^* a + q_2(b+aa) \\ &= (a + a(b+aa)^* b)^* a(b+aa)^* \end{aligned}$$

putting q_2 in q_3

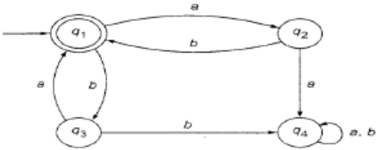
$$q_3 = (a + a(b+aa)^* b)^* a(b+aa)^* a \tag{9}$$

Arden's Theorem

- Since q_3 is the final state.
  $\therefore$ r$= (a + a(b + aa)^* b)^* a(b + aa)^* a$

Arden's Theorem

- Prove that the finite automaton whose transition diagram below accepts the set of all strings over alphabet {a,b} with an equal number of **a's** and **b's**, such that each prefix has atmost has atmost one more **a** than the **b's** and atmost one more **b** than the **a's**



Arden's Theorem

Solution:

$$q_1 = q_2 b + q_3 a + \wedge \tag{10}$$

$$q_2 = q_1 a \tag{11}$$

$$q_3 = q_1 b \tag{12}$$

$$q_4 = q_2 a + q_3 b + q_4 a + q_4 b \tag{13}$$

putting q_2 and q_3 in q_1

$$q_1 = q_1 ab + q_1 ba + \wedge$$

$$q_1 = q_1(ab + ba) + \wedge$$

applying Arden's theorem

$$q_1 = \wedge(ab + ba)^*$$

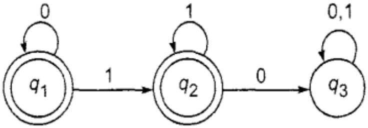
$$q_1 = (ab + ba)^*$$

Arden's Theorem

- Now, the prefix can be even or odd in length. For Prefix x of even length, the number of a's and b's shall be equal as x is a substring formed by ab's and ba's. For prefix x of odd length, then we can write 'x' as ya or yb. As y has even number of symbols, which implies x has one more a than b or vice-versa

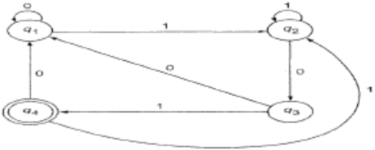
Arden's Theorem

- Describe in English the set accepted by finite automaton whose transition diagram is as under:



Arden's Theorem

- Construct a regular expression corresponding to the state diagram described as under:



- Give R.E. for representing the set L of strings in which every 0 is immediately followed by atleast two 1's. Prove that R.E. r= $\wedge + 1^*(011)^*(1^*(011)^*)^*$ also describes the same set of strings.
- Prove $(1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1) = 0^*1(0+10^*1)^*$

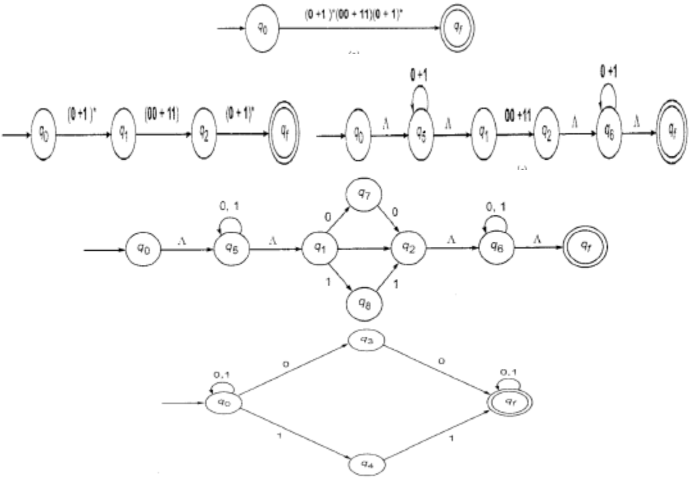
Construction of FA equivalent to given RE

The method for constructing a finite automaton equivalent to a given regular expression is called the subset method which involves four steps.

1. Construct a transition system equivalent to the given regular expression using $\wedge$-moves.
2. Construct the transition table for the transition graph obtained in step 1.
3. Construct the DFA equivalent to NDFA.
4. Reduce the number of states if possible.

- Construct FA equivalent to Regular Expression.
  $(0+1)^*(00+11)(0+1)^*$
  Solution:

Construction of FA equivalent to given RE



State/Σ	0	1
q_0	q_0, q_3	q_0, q_4
q_3	q_f	
q_4		q_f
q_f	q_f	q_f

Construction of FA equivalent to given RE

converting to DFA

State/Σ	0	1
q_0	q_0, q_3	q_0, q_4
q_0, q_3	q_0, q_3, q_f	q_0, q_4
q_0, q_4	q_0, q_3	q_0, q_4, q_f
q_0, q_3, q_f	q_0, q_3, q_f	q_0, q_4, q_f
q_0, q_4, q_f	q_0, q_3, q_f	q_0, q_4, q_f

reducing

State/Σ	0	1
q_0	q_0, q_3	q_0, q_4
q_0, q_3	q_0, q_3, q_f	q_0, q_4
q_0, q_4	q_0, q_3	q_0, q_3, q_f
q_0, q_3, q_f	q_0, q_3, q_f	q_0, q_3, q_f

Construction of FA equivalent to given RE

1. Construct DFA with reduced states equivalent to R.E. $(10+(0+11)0^*1)$.
2. Construct transition system equivalent to R.E.
   - $(ab+c^*)^*b$
   - $a+bb+bab^*a$
   - $(a+b)^*abb$
3. Prove that
   $(a^*ab+ba)^*a^* = (a+ab+ba)^*$
4. Construct a finite automata accepting all strings over {0,1} ending in 010 or 0010.
5. Construct a regular grammar which can generate the set of all strings starting with a letter (A to Z) followed by a string of letters or digits (0 to 9).

2-way DFA

- 2-way DFA allows the read head to move left or right on the input
- Two end-markers
- Needs only 1 accept or reject state.
- A tuple $M = \{Q, \Sigma, \vdash, \dashv, \delta, s, t, r\}$
  where Q is the set of states
  Σ is the input alphabet set
  $\vdash$ is the left end marker
  $\dashv$ is the right end marker
  δ is $Q \times (\Sigma \cup \{\vdash, \dashv\}) \rightarrow Q \times \{L, R\}$
  s is start state
  t is the accept state
  r is reject state such that r $\neq$ t

2-Way DFA

- Determine the acceptability of 101001 for the following:

State/ Σ	0	1
$\rightarrow q_0$	$(q_0$,R)	$(q_1$, R)
q_1	$(q_1$,R)	$(q_2$, L)
q_2	$(q_0$,R)	$(q_2$, L)

where Q=$\{q_0, q_1, q_2\}$, s=q_0, t=q_1, r=q_2

Pumping Lemma for Regular Sets

- Let M = (Q, Σ, δ, q_0, F) be a finite automaton with 'n' states. Let L be the regular sets accepted by M.
- Let w $\in$ L and $|w| \geq$ n, then $\exists$ x,y,z such that w=xyz, y $\neq \wedge$ and $xy^iz \in$ L for each i $\geq$ 0.
- **Applications of Pumping Lemma:** Used to prove that certain set are not regular.
- **Steps to prove that given set is not regular:**
  1. Assume L is regular. Let 'n' be the number of states in corresponding FA.
  2. Choose a string 'w' such that $|w| \geq$ n. Use pumping lemma to write w=xyz with $|xy| \leq$ n and $|y| >$ 0
  3. Fing a suitable integer i such that $xy^iz \notin$ L. This contradicts our assumption. Hence, L is not regular.

Pumping Lemma for Regular Sets

Show that the set L $=\{a^{i^2} \mid i \leq 1\}$ is not regular

Solution:

- Let L is regular
  Let 'n' be number of states in FA accepting L.
- Let w=$a^{n2} \implies |w| = n^2 > n$
  by pumping lemma, w=xyz with $|xy| \leq$ n and $|y| > 0$
- Consider xy^2z
  $|xy^2z| = |x| + 2|y| + |z| > |x| + |y| + |z| \because |y| > 0$
  $\implies n^2 = |xyz| = |x| + |y| + |z| < |xy^2z|$
  As $|xy| \leq$ n, $|y| \leq$ n
- $\therefore |xy^2z| = |x| + 2|y| + |z| \leq n^2 + n < n^2 + n + n + 1.$
  Hence, $|xy^2z|$ lies between n^2 and $(n+1)^2$ but not equal to any one of them.
  $\therefore |xy^2z|$ is not a perfect square and so $xy^2z \notin$ L.
  $\therefore$ this is a contradiction. This implies not Regular

Pumping Lemma for Regular Sets

Show that L= $\{a^p \mid p \text{ is a prime}\}$ is not regular.

Solution:

1. Let L is regular. Let 'n' be number of states in finite automata accepting L.
2. Let 'p' be a prime number greater than 'n'.
   Let w=a^p
   by pumping lemma, w=xyz with $|xy| \leq$ n and $|y| > 0$
   x, y, z are simply strings of a's.
   So, y= a^m for some m $\geq$ 1 (and $\leq$ n)
3. Let i= p+1, then
   $|xy^iz| = |xyz| + |y^{i-1}|$=p+(i-1)m=p+pm=p(1+m) which is not prime.
   $\therefore xy^iz \notin$ L. $\implies$ contradiction.
   So, L is not regular.

Pumping Lemma for Regular Sets

1. Show that L=$\{0^i 1^i \mid i \geq 1\}$ is not regular.
2. Show that L= $\{ww \mid w \in \{a, b\}^* \}$ is not regular.
3. Is L $= \{a^{2n} \mid n \geq 1\}$ regular ?

Regular Sets and Regular Grammar

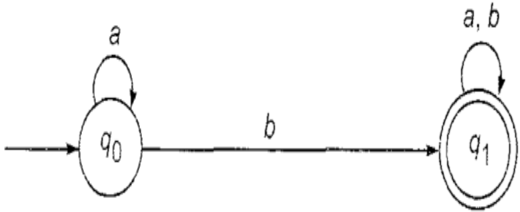
We can construct a Regular Grammar from a Regular Sets and vice versa.

Construction of a Regular Grammar from a Regular Sets

- We can show that $L(G) = T(M)$ by using the construction of P such that:
  $A_i \rightarrow aA_j$ iff $\delta(q_i, a) = q_j$
  $A_i \rightarrow a$ iff $\delta(q_i, a) \in F$
- Construct a regular grammar G generating the regular set represented by
  P$= a^*b(a+b)^*$.

Regular Sets and Regular Grammar

Solution:



Let G=($\{q_0, q_1\}$,{a,b},P,q_0)
where P is given by:

$q_0 \rightarrow aq_0$

$q_0 \rightarrow bq_1, \; q_0 \rightarrow b$

$q_1 \rightarrow aq_1, \; q_1 \rightarrow bq_1$

$q_1 \rightarrow a, q_1 \rightarrow b$

Regular Sets and Regular Grammar

Construction of a Regular Set from a Regular Grammar

- We define M as:
  1. Each production $A_i \to aA_j$ induces a transition from q_i to q_j with label a i.e $\delta(q_i, a) = q_j$,
  2. Each production $A_i \to a$ induces a transition from q_i to q_f with label a i.e $\delta(q_i, a) = q_f \in F$
  3. $S \to \wedge$, corresponding transition is from q_0 to q_f with a label $\wedge$ or q_0 is also a final state.
- Let G=({A,B},{a,b},P,A) where P consists of
  A $\to$ aB, B $\to$ bB
  B $\to$ a, B $\to$ bA
  Construct a transition system M accepting L(G).

Regular Sets and Regular Grammar

- If a regular grammar G is given by $S \to aS \mid a$. Find M accepting L(G).
- Construct a DFA equivalent to grammar
  $S \to aS \mid bS \mid aA$
  $A \to bB$
  $B \to aC$, $C \to \wedge$

Context Free Grammar

- **Finite Automata** accepts all regular languages.
  - Simple languages such as
    - $a^n b^n : n = 0, 1, 2, ...$
    - $w : w$ is a Palindrome

    are not regular and thus no finite automata accepts them.
- Context Free Languages are a larger class of languages that encompasses all regular languages and many others including above examples.
- Languages generated by context free grammar are called Context free languages.
- Context free grammar are more expressive than finite automata: If a language L is accepted by a finite automata, then L can be generated by a context-free grammar, *While opposite is not true*

Context-Free Grammars

- A Context-free grammar is a 4-tuple (V_n, Σ, P, S)
  - V_n is set of *Variables*.
  - Σ is set of terminals.
  - P is set of Productions.
  - S is the start symbol.
- A Grammar G is Context free, if every production is of the form $A \to \alpha$, where $A \in V_N$ and $\alpha \in (V_N \cup \Sigma)^*$

Example:CFG

- Contruct a CFG generating all integers (with sign).

Solution: Let $G = (V, \Sigma, P, S)$

where $V = \{S, < sign >, < digit >, < integer >\}$

$\Sigma = \{0, 1, 2, 3, ...9, +, -\}$

P consists of:

S $\rightarrow < sign >< integer >$

$< sign > \rightarrow +|-$

$< integer > \rightarrow < digit >< integer > \mid < digit >$

$< digit > \rightarrow 0|1|2|\ldots|9$

Derivation for -42

S$\rightarrow < sign >< integer >$

$\Longrightarrow$ -$< integer >$

$\Longrightarrow$ -$< digit >< integer >$

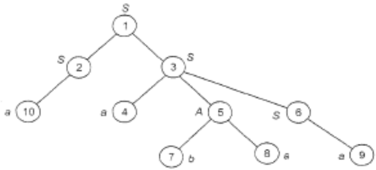
$\Longrightarrow$ -4$< digit >$

$\Longrightarrow$ -42

Derivation Trees

- Trees are used for derivation of CFG.
- Definition: A derivation tree (or a parse tree) for a CFG $G = (V, \Sigma, P, S)$ is a tree satisfying:
  - Every vertex has a label which is variable/terminal/$\wedge$.
  - The root has label S.
  - The label of the internal vertex is a variable.
  - If vertices $n_1, n_2, ...n_k$ written with labels $X_1, X_2, ...X_k$ are sons of vertex 'n' with label A, then $A \rightarrow X_1X_2...X_k$ is a production in P.
  - A vertex 'n' is a leaf if its label is $a \in \Sigma$ or $\wedge$; 'n' is the only son of its father if its label is $\wedge$
- Let G = ({S,A},{a,b},P,S) where P consists of

$$S \rightarrow aAS|a|SS, A \rightarrow SbA|ba$$



Derivation Trees

- Yield of Derivation Tree:is a concatenation of labels of the leaves without repetition in the left to right ordering. Example: aabaa
- Subtree of a Derivation Tree T is a tree:
  - whose root is some vertex 'v' of T,
  - whose vertices are descendants of 'v' together with their labels.
  - whose edges are those connecting the descendants of v.

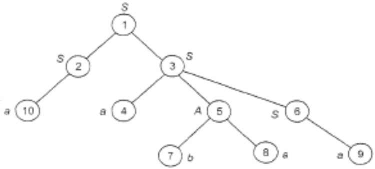


Figure: Derivation Tree T

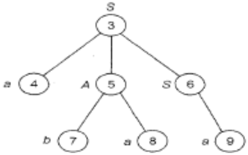
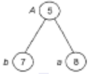


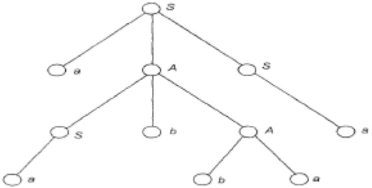
Figure: Sub tree of Tree T



Derivation Trees

Theorem 1:
Let G=(V,Σ, P, S) be a CFG. Then, $S \stackrel{*}{\Longrightarrow} \alpha$ if and only if there is a derivation tree for G with yield α.

- Example: Consider G whose productions are $S \rightarrow aAS|a, A \rightarrow SbA|SS|ba$. Show that $S \stackrel{*}{\Longrightarrow} aabbaa$ and Construct a derivation tree whose yield is $aabbaa$.
- Case 1:
  $S \Longrightarrow aAS \Longrightarrow aSbAS \Longrightarrow aabAS \Longrightarrow a^2bbaS \Longrightarrow aabbaa$
- Case 2:
  $S \Longrightarrow aAS \Longrightarrow aAa \Longrightarrow aSbAa \Longrightarrow aSbbaa \Longrightarrow aabbaa$
- Case 3:
  $S \Longrightarrow aAS \Longrightarrow aSbAS \Longrightarrow aSbAa \Longrightarrow aabAa \Longrightarrow aabbaa$



Derivation Trees

- Left most derivation: A derivation $A \overset{*}{\Longrightarrow} w$ is called left-most derivation, if we apply production only to the left most variable at every step.
- Right most derivation: A derivation $A \overset{*}{\Longrightarrow} w$ is a right most derivation, if we apply production to right most variable at each step.

Theorem

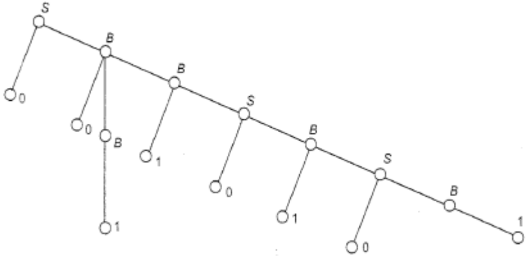
If $A \overset{*}{\Longrightarrow} w$ in G, then there is a left most derivation of w.

Derivation Trees

- Example: Let G be the grammar $S \to 0B|1A$, $A \to 0|0S|1AA$, $B \to 1|1S|0BB$. For the string 00110101, find:
  - the leftmost derivation
  - the rightmost derivation
  - The derivation tree
- Leftmost derivation:

  $S \implies 0B \implies 00BB \implies 001B \implies 0011S \implies 0^2 1^2 0B \implies 0^2 1^2 01S \implies 0^2 1^2 010B \implies 0^2 1^2 0101$
- Rightmost derivation:

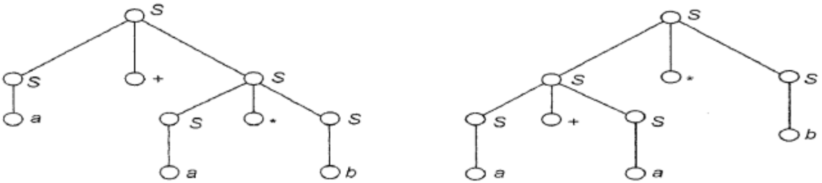
  $S \implies 0B \implies 00BB \implies 00B1S \implies 00B10B \implies 0^2 B101S \implies 0^2 B1010B \implies 0^2 B10101 \implies 0^2 110101$
- Derivation tree:



Ambiguity in CFG

In books selected information is given.

- A terminal string $w \in L(G)$ is ambiguous if there exist two or more derivation trees for 'w' (or there exist two or more left most derivation of w).
- Example: $G = (\{S\}, \{a, b, +, *\}, P, S)$, where P consists of $S \rightarrow S + S | S * S | a | b$. We have two derivation trees for a+a*b:

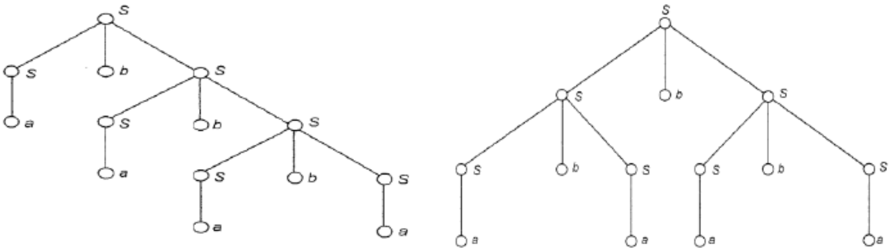


- $S \implies S + S \implies a + S \implies a + S * S \implies a + a * S \implies a + a * b$
- $S \implies S * S \implies S + S * S \implies a + S * S \implies a + a * S \implies a + a * b$

Ambiguity in CFG

Example:

- If G is grammar $S \rightarrow SbS|a$. Show that the given grammar is ambigious.
- For $w = abababa$



- Thus, G is ambiguous.

Simplification of CFG

- We eliminate following in order to simplify a grammar:
  - Useless variables
  - Unit productions
  - Null Productions

Eliminating Useless Variables

- Those variables which are not deriving any terminal string and which are not reachable are known as Useless Variables.
- Example: $S \rightarrow AB$
  $A \rightarrow a|B$
  $B \rightarrow b|C$
  $C \rightarrow aC$
  $D \rightarrow b$
- C is not deriving any terminal, C is not deriving any terminal $\implies$ useless variable, $\therefore$ remove C
  $S \rightarrow AB$, $A \rightarrow a|B$, $B \rightarrow b$, $D \rightarrow b$
  D is not included in S $\implies$ remove D
  $\therefore S \rightarrow AB$, $A \rightarrow a|B$, $B \rightarrow b$

Eliminating Unit Productions

- Production of the form $A \rightarrow B$ is known as Unit Production
- Example: $S \rightarrow A,\ A \rightarrow B,\ B \rightarrow C,\ C \rightarrow D,\ D \rightarrow a$
  appears as chainlike process
  Instead $S \rightarrow a$ will serve the purpose
  $\therefore$ required grammar is $S \rightarrow a$

Eliminating Unit Productions

- Example: $S \to Aa|B$, $B \to A|bb$, $A \to a|bc|B$
  Starting from last to first
  $A \to B$ is a unit production, replace B by its R.H.S.
  $A \to a|bc|A|bb$
  $B \to a|bc|A|bb$
  $S \to Aa|a|bc|A|bb$
  Now, remove unit productions
  $A \to a|bc|bb$
  $B \to a|bc|bb$
  $S \to Aa|bc|a|bb$
  Now, S does not has B, $\implies$ remove B
  $S \to Aa|bc|a|bb$
  $A \to a|bc|bb$

Eliminating Null Productions

- Any production of the form $A \to \epsilon$ is called Null Production.
- Example: $S \to aAb$, $A \to aAb|\epsilon$
  Replace A with ϵ in each production containing A and add it to grammar without ϵ
  $S \to aAb|ab$
  $A \to aAb|ab$

Simplifying CFG

- Example: Construct reduced grammar equivalent to grammar G whose productions are:
  $S \rightarrow AB|CA$, $B \rightarrow BC|AB$, $A \rightarrow a$, $C \rightarrow aB|b$
  Here, B is not deriving any terminals $\therefore$ remove B
  $S \rightarrow CA$, $A \rightarrow a$, $C \rightarrow b$
  $\therefore$ New Grammar $G' = (\{S, A, C\}, \{a, b\}, P, S)$
- Question: Find the reduced grammar equivalent to G.
  $S \rightarrow aAa$, $A \rightarrow bBB$, $B \rightarrow ab$, $C \rightarrow aB$

Normal forms of CFG

- Chomsky Normal Form
- Greibach Normal Form

Chomsky Normal Form

A CFG is in Chomsky normal form if all productions are of the form:
$A \to BC$ or $A \to a$ and $S \to \wedge$ if $\wedge \in L(G)$
where A, B, C are variables and a is a terminal. When $\wedge$ is in $L(G)$, we assume that S does not appear on the R.H.S. of any production.

- Example: The Grammar in the form:
  $S \to AS|a$, $A \to SA|b$
- Reduction to Chomsky normal form
  1. Elimination of null productions and unit productions
  2. Elimination of terminals on R.H.S.
  3. Restricting the number of variables on R.H.S.

Chomsky Normal Form

- Example: Convert the grammar G with Productions as:
  $S \to ABa$, $A \to aab$, $B \to Ac$ to chomsky normal form
  Let us assume few new variables:
  $X \to a$, $Y \to b$, $Z \to c$
  gives $S \to ABX$, $A \to XXY$, $B \to AZ$
  Now, Assuming $Q \to XX$, $P \to AB$
  gives, $S \to PX$, $A \to QY$, $B \to AZ$, $X \to a$, $Y \to b$, $Z \to c$,
  $Q \to XX$, $P \to AB$
- Convert the given grammar to CNF
  - S $\to$ aSb |ab
  - S $\to$ aAB |Bb
    A $\to$ a , B $\to$ b
  - S $\to$ bA |aB
    A $\to$ bAA |aS |a
    B $\to$ aBB |bS |b

Greibach Normal Form

- A CFG is said to be in Greibach Normal form, if all the productions have the form $A \to aX$ (or $A \to a$ when X is $\wedge$), where $a \in \Sigma$ and $X \in V^*$ (X may be $\wedge$) and $S \to \wedge$ if $\wedge \in L(G)$ and S does not appear on the R.H.S. of any production
- Example: $S \to aAB|bBB|bB|a$
  $A \to aA|bB|b$, $B \to b$

Lemma 1:
Let $G = (V, \Sigma, P, S)$ be a CFG. Let $A \to B\gamma$ be an A-production in P. Let the B-productions be $B \to \beta_1|\beta_2|\ldots|\beta_k$. Define $P_1 = (P - \{A \to B\gamma\}) \cup \{A \to \beta_i\gamma | 1 \leq i \leq k\}$. Then $G_1 = (V, \Sigma, P_1, S)$ is a context-free grammar equivalent to G.

Lemma 2:

Greibach Normal Form

Let $G = (V, \Sigma, P, S)$ be a CFG. Let the set of A-productions be $A \to A\alpha_1|A\alpha_2|\ldots|A\alpha_r|\beta_1|\beta_2|\ldots|\beta_s|$ (β_i's do not start with A). Let Z be a new variable. Let $G_1 = (V \cup \{Z\}, \Sigma, P_1, S)$, where P_1 is defined as follows:

1. The set of A-productions in P_1 are
   $A \to \beta_1|\beta_2|\ldots|\beta_s$
   $A \to \beta_1 Z|\beta_2 Z|\ldots|\beta_s Z$
2. The set of Z-productions in P_1 are
   $Z \to \alpha_1|\alpha_2|\ldots|\alpha_r$
   $Z \to \alpha_1 Z|\alpha_2 Z|\ldots|\alpha_r Z$
3. The productions for the other variables are as in P.

Then G_1 is a CFG and equivalent to G.

Greibach Normal Form

- Reduction to Greibach normal form
  1. Elimination of null productions and unit productions
  2. Elimination of terminals on R.H.S. except the first leftmost terminal.
  3. Make all production starting with a terminal if not.
- Example: Convert the grammar G into equivalent GNF:
  $S \to abSb|aa$
  Solution: Let $A \to a$, $B \to b$
  $\therefore S \to aBSB|aA$, $A \to a$, $B \to b$
- Convert the grammar S $\to$ ab |aS |aaS into GNF
  Solution: Let B $\to$ b, A $\to$ a, S $\to$ aB |aS |aAS
- Exercise:
  1. Convert the grammar $S \to AA|a, A \to SS|b$ into GNF.
  2. Convert the grammar $S \to AB, A \to BS|b, B \to SA|a$ into GNF.
  3. Convert the grammar $E \to E + T|T, T \to T * F|F, F \to (E)|a$ into GNF.

Pumping Lemma for Context Free Language

Let L be an infinite context free language. Then, there exists some positive integer n such that:

1. Every $z \in L$ with $|z| \geq n$ can be written as $uvwxy$ for some strings u, v, w, x, y.
2. $|vx| \geq 1$
3. $|vwx \leq n$
4. $uv^k wx^k y \in L$ for all $k \geq 0$

Application: We use the pumping lemma to show that a language L is not a context free language.

Procedure: We assume that L is context-free. By applying the pumping lemma we get a contradiction. The procedure can be carried out by using the following steps:

Step 1 Assume L is context-free. Let n be the natural number obtained by using the pumping lemma.

Pumping Lemma for Context Free Language

Step 2 Choose $z \in L$ so that $|z| \geq n$. Write $z = uvwxy$ using the pumping lemma.

Step 3 Find a suitable k so that $uv^kwx^ky \notin L$. This is a contradiction, and so L is not context-free.

Pumping Lemma for Context Free Language

Ques: Show that the language
L=$\{a^n b^n c^n$: n $\geq$ 0$\}$ is not context free.
Solution: Let L be Context free
Let w be a string in L
For n=4, string becomes aaaabbbbcccc
By Pumping Lemma, w=uvxyz
w=aaaabbbbcccc
Case 1: If vxy contain only a, b or c, then on pumping. It won't be in L
Case 2: If string contains any two either ab or bc, then pumped string will contain $a^k b^l c^m$ with $k \neq l \neq m$ or it will not be in the order, so does not belong to L
$\therefore$ L is not context free.

- Show that following L are not Context free
  1. L=$\{ww : w \in \{a, b\}*\}$
  2. L=$\{a^n b^j : n = j^2\}$
  3. L= $\{a^{n!} : n \geq 0\}$

Pumping Lemma for Context Free Language

Ques: Show that $L = \{a^p | p \text{ is a prime }\}$ is not a context-free language

Solution:

1. Let L be Context free, Let w be a string in L
2. Let n be the natural number obtained by using the pumping lemma.
3. Let p be a prime number greater than n, Then $z = a^p \in L$. We can write $z = uvwxy$.
4. Prove for some k that $uv^k wx^k y \notin L$ means $|uv^k wx^k y|$ is not prime.
5. By pumping lemma, $uv^0 wx^0 y = uwy \in L$. So $uxy|$ is a prime number, say q.
6. Let $|vx| = r$. Then, $|uv^q wx^q y| = q + qr$.
7. As $q(1 + r)$ is not a prime, means $uv^q wx^q y \notin L$.
8. This is a contradiction. Therefore, L is not context-free.

Pumping Lemma for Context Free Language

- Show that following L are not Context free
  1. L=$\{ww : w \in \{a, b\}^*\}$
  2. L=$\{a^n b^j : n = j^2\}$
  3. L= $\{a^{n!} : n \geq 0\}$

Decision Algorithms

Some decision algorithms for context-free languages and regular sets.

1. Algorithm for deciding whether a context free language L is empty.
2. Algorithm for deciding whether a context-free language L is finite.
3. Algorithm for deciding whether a regular language L is empty.
4. Algorithm for deciding whether a regular language L is infinite.

Linear Grammar

- A grammar in which atmost one variable can occur on right side of any production without restriction on the size of this grammar, is known as Linear Grammar.
- Right Linear Grammar- A grammar $G = (V.T, P, S)$ is said to be right linear, if all the productions are of the form:
  $A \rightarrow xB$, $A \rightarrow x$
  where $A, B \in V, x \in T^*$
- Left Linear Grammar- A grammar is said to be left linear if all the productions are of the form:
  $A \rightarrow Bx$, $A \rightarrow x$
  where $A, B \in V, x \in T^*$
- Linear Grammar- A grammar is said to be linear grammar if all the productions are of the form:
  $A \rightarrow vBw$, $A \rightarrow w$
  where $A, B \in V; v, w \in T^*$

Linear Grammar

- A regular grammar is always linear but not all linear grammars are regular.
- A regular grammar is one that is either right linear or left linear
- In a regular grammar, atmost one variable appears on right side of any production. Further, that variable must consistently be either on rightmost or leftmost symbol of right side of any production.
- Example: $G_1 = (\{S\}, \{a, b\}, P_1, S)$ where P_1 given as $S \rightarrow abS|a$ is right linear.
- Example: $G_2 = (\{S, S_1, S_2\}, \{a, b\}, P_2, S)$ with productions $S \rightarrow S_1ab, S_1 \rightarrow S_1ab, S_1 \rightarrow S_2, S_2 \rightarrow a$ is left linear

Linear Grammar

- $G = (\{S, A, B\}, \{a, b\}, P, S)$ with productions
  $S \rightarrow A$, $A \rightarrow aB|\wedge$, $B \rightarrow Ab$ is not regular
  $\because$ even if each production is right linear or left linear, but grammar itself is neither right linear nor left linear $\therefore$ not regular

Linear Grammar

1. Construct a finite automata that accepts the language generated by grammar
   $V_0 \to aV_1,\ V_1 \to abV_0\ |$b
2. Construct a right linear grammar for L(aab^*a)
3. Convert the following regular expression into equivalent regular grammar
   - $(a+b)^*a$
   - a^*+b+b^*

Pushdown Automata

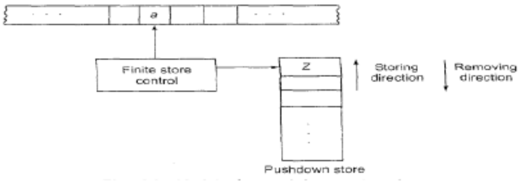


Figure: Model of Pushdown Automata

- A Non-deterministic Pushdown Automata is a 7-tuple $(Q, \Sigma, \tau, \delta, q_0, Z_0, F)$
  where , $Q=$ finite non-empty set of states
  $\Sigma=$ finite non-empty set of input symbols
  $\tau=$ finite non-empty set of pushdown symbols
  $q_0=$ initial state
  $Z_0=$ initial symbol on pushdown store
  $F=$ set of final states
  $\delta \implies Q \times (\Sigma \cup \{\wedge\}) \times \tau \rightarrow Q \times \tau^*$

Pushdown Automata

$\delta \implies Q \times (\Sigma \cup \{\wedge\}) \times \tau \to Q \times \tau^*$

- Each move of the control unit is determined by the current input symbol as well as by the symbol currently on the top of the stack.
- The result of the move is a new state of control unit and a change in the top of the stack.

Instantaneous Description (ID) Let $A = (Q, \Sigma, \tau, \delta, q_0, Z_0, F)$ be a pda. An instantaneous description (ID) is (q, w, α), where $q \in Q, w \in \Sigma^*$ and $\alpha \in \tau^*$.

- An initial ID is $({}_q o, w, Z_0)$. This means that initially the pda is in the initial state q_0, the input string to be processed is w and the PDS has only one symbol, namely Z_0.
- In an ID $(q, \wedge, Z)$, In this case the pda makes a $\wedge$-move.

Pushdown Automata

- A move relation, denoted by $\vdash$ between IDs is defined as

$$(q, a_1 a_2 \ldots a_n, Z_1 Z_2 \ldots Z_m) \vdash (q', a_2 \ldots a_n, \beta Z_2 \ldots Z_m)$$

  if $\delta(q, a_1, Z_1) = (q', \beta)$
- if $(q_1, x, \alpha) \vdash^* (q_2, \wedge, \beta)$ then for every $y \in \Sigma^*$, $(q_1, xy, \alpha) \vdash^* (q_2, y, \beta)$
- Conversely, if $(q_1, xy, \alpha) \vdash^* (q_2, y, \beta)$ for some $y \in \Sigma^*$, then $(q_1, x, \alpha) \vdash^* (q_2, \wedge, \beta)$
- if $(q_1, x, \alpha) \vdash^* (q_2, \wedge, \beta)$ then for every $\gamma \in \tau^*$, $(q_1, x, \alpha\gamma) \vdash^* (q_2, \wedge, \beta\gamma)$

NPDA: Example

- Consider a NPDA as under:
  $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a, b\}$
  $\tau = \{a, Z_0\}$, Z_0, $F = \{q_3\}$ and
  $\delta(q_0, a, Z_0) = (q_0, aZ_0)$
  $\delta(q_0, a, a) = (q_0, aa)$
  $\delta(q_0, b, a) = (q_1, \wedge)$
  $\delta(q_1, b, a) = (q_1, \wedge)$
  $\delta(q_1, \wedge, Z_0) = (q_f, Z_0)$
  What can we say about the action of this automaton?

Language accepted by a PDA

Acceptance of input strings by pda is of two way:

1. Acceptance by Final State
2. Acceptance by Null Store

Let M $=(Q, \Sigma, \tau, \delta, q_0, Z_0, F)$ be a non-deterministic push-down automata. The language accepted by M is the set
$L(M) = \{w \in \Sigma^* | (q_0, w, Z_0) \vdash_M^* (q', \Lambda, \alpha)\}$
where $q' \in F$ and $\alpha \in \tau^*$

Example: Construct a NPDA for the language
$L = \{w \in \{a, b\}^* | n_a(w) = n_b(w)\}$

Solution: $Q = \{q_0, q_f\}, \Sigma = \{a, b\}, \tau = \{a, b, Z\}, F = \{q_f\}$
Let M= $\{\mathrm{Q}, \Sigma, \tau, \delta, q_0, Z, F\}$
$\delta(q_0, a, Z) = (q_0, aZ)$
$\delta(q_0, b, Z) = (q_0, bZ)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, b) = (q_0, bb)$
$\delta(q_0, a, b) = (q_0, \Lambda)$

Language accepted by a PDA

$\delta(q_0, b, a) = (q_0, \Lambda)$
$\delta(q_0, \Lambda, Z) = (q_f, Z)$
Let us assume $w = baab$ to process
$(q_0, baab, Z) \vdash$
$(q_0, aab, bZ) \vdash$
$(q_0, ab, Z) \vdash$
$(q_0, b, aZ) \vdash$
$(q_0, \Lambda, Z) \vdash$
(q_f, Λ, Z)

Language accepted by PDA

Let A $=(Q, \Sigma, \tau, \delta, q_0, Z_0, F)$ be a non-deterministic push-down automata. The language accepted by null store or empty store A is the set $N(A) = \{w \in \Sigma^* | (q_0, w, Z_0) \vdash_A^* (q, \Lambda, \Lambda)\}$
where $q \in Q$

Theorem

If $A = (Q, \Sigma, \tau, \delta, q_0, Z_0, F)$ is a pda accepting L by empty store. we can find a pda $B = (Q', \Sigma, \tau', \delta', q_0', Z_0, F')$ accepting L by final state: i.e. $L = N(A) = T(B)$.

Theorem

If $A = (Q, \Sigma, \tau, \delta, q_0, Z_0, F)$ accepts L by final state, we can find a pda B accepting L by empty store: i.e. $L = T(A) = N(B)$.

Language accepted by PDA

Question: Construct a NPDA for accepting the language
L=$\{ww^R$: w $\in \{a, b\}^+$ }
Solution: M=(Q,Σ, τ, δ, q_0, z, F)
where Q= $\{q_0, q_1, q_2\}$, Σ={a,b}
τ= {a,b,z}, F=$\{q_2\}$
$\delta(q_0, a, a) = (q_0, aa)$, $\delta(q_0, b, a) = (q_0, ba)$
$\delta(q_0, a, b) = (q_0, ab)$, $\delta(q_0, b, b) = (q_0, bb)$
$\delta(q_0, a, z) = (q_0, az)$, $\delta(q_0, b, z) = (q_0, bz)$
For middle:
$\delta(q_0, \Lambda, a) = (q_1, a)$, $\delta(q_0, \Lambda, b) = (q_1, b)$
For matching w^R against contents of stack
$\delta(q_1, a, a) = (q_1, \Lambda)$, $\delta(q_1, b, b) = (q_1, \Lambda)$
Finally, $\delta(q_1, \Lambda, z) = (q_2, z)$
Let the String assumed be 'abba':
$(q_0, abba, z) \vdash (q_0, bba, az) \vdash (q_0, ba, baz) \vdash (q_1, ba, baz)$
$\vdash (q_1, a, az) \vdash (q_1, \Lambda, z) \vdash (q_2, z)$

Language accepted by a PDA

1. Construct a NPDA that accepts the language
   L=$\{a^n b^m$: n$\geq$0, n$\neq$m$\}$
2. Find NPDA on Σ=$\{$a,b,c$\}$ that accepts the language
   L=$\{w_1 c w_2$: $w_1, w_2 \in \{a, b\}^*$, $w_1 \neq {w_2}^R\}$

CFG to PDA

Theorem

If L is a context-free language, then we can construct a pda A accepting L by empty store, i.e. $L = N(A)$.

Construction of pda A Let $L = L(G)$, where $G = (V, \Sigma, P, S)$ is a context-free grammar. We construct a pda A as $A = (\{q\}, \Sigma, V \cup \Sigma, \delta, q, S, \phi)$, where δ is defined by the following rules:

R_1 For every $A \to \alpha$ in P, $\delta(q, \wedge, A) = \{(q, \alpha)\}$

R_2 For every a in Σ, $\delta(q, a, a) = \{(q, \wedge)\}$

CFG to PDA

- Construct a PDA that accepts the language generated by the grammar with productions
  $S \to aSA|a,\ A \to bB,\ B \to b$
  Solution: Step-1 The given productions are:
  $S \to aSA|a$
  $A \to bB$
  $B \to b$
  δ is defined by the following rules:
  S-productions
  $\delta(q, \wedge, S) = \{(q, aSA), (q, a)\}$
  A-productions
  $\delta(q, \wedge, A) = \{(q, bB)\}$
  B-productions
  $\delta(q, \wedge, B) = \{(q, b)\}$
  Productions for Σ
  $\delta(q, a, a) = \{(q, \wedge)\}$

CFG to PDA

$\delta(q, b, b) = \{(q, \wedge)\}$
Appearance of Λ on top of stack implies completion of derivation and PDA is put to final state by
$\delta(q, \Lambda, Z) = (q_f, \Lambda)$

CFG to PDA

Question: Consider the grammar

S $\rightarrow$ aA, A $\rightarrow$aABC |bB |a, B $\rightarrow$b, C $\rightarrow$c

Solution: Putting Start Symbol on stack

$\delta(q_0, \Lambda, Z) = (q_1, SZ)$

Final Production: $\delta(q_1, \Lambda, Z) = (q_f, Z)$

Now, according to the productions

$\delta(q_1, a, S) = (q_1, A)$, $\delta(q_1, a, A) = (q_1, ABC)$,
$\delta(q_1, b, A) = (q_1, B)$, $\delta(q_1, a, A) = (q_1, \Lambda)$, $\delta(q_1, b, B) = (q_1, \Lambda)$,
$\delta(q_1, c, C) = (q_1, \Lambda)$

Let the derivation be

S $\rightarrow$ aA $\rightarrow$ a**aAB**C $\rightarrow$ aa**a**BC $\rightarrow$ aaa**b**C $\rightarrow$ aaab**c**

Therefore, the sequence of moves by M for the processing of "aaabc" is:

$(q_0, aaabc, Z) \vdash (q_1, aaabc, SZ) \vdash (q_1, aabc, AZ)$
$\vdash (q_1, abc, ABCZ) \vdash (q_1, bc, BCZ) \vdash (q_1, c, CZ) \vdash (q_1, \Lambda, Z)$
$\vdash (q_f, \Lambda, Z)$

CFG to PDA

Question Construct a PDA 'A' equivalent to the following Context free grammar:
S $\rightarrow$ 0BB, B $\rightarrow$ 0S |1S |0.
Test whether 010000 is in N(A).
Solution: Let A =({q},{0,1},{S,B,0,1},δ,q,S,ϕ)
δ is defined by following rules:
$\delta(q, \Lambda, Z) = (q, SZ)$
$\delta(q, 0, S) = (q, BB)$
$\delta(q, 0, B) = (q, S)$
$\delta(q, 1, B) = (q, S)$
$\delta(q, 0, B) = (q, \Lambda)$
$\delta(q, \Lambda, Z) = (q_f, \Lambda)$
Let the derivation be:
S $\rightarrow$ 0BB $\rightarrow$ 01SB $\rightarrow$ 010BBB $\rightarrow$ 010000
Acceptability of 010000:
(q,010000,Z) $\vdash$ (q,010000,SZ) $\vdash$ (q,10000,BB) $\vdash$ (q,0000,SB) $\vdash$
(q,000,BBB) $\vdash$ (q, 00,BB) $\vdash$ (q,0,B) $\vdash$ (q,Λ,Z) $\vdash$ (q_f, Λ)

PDA to CFG

Theorem

If $A = (Q, \Sigma, \tau, \delta, q_0, Z_0, F)$ is a pda then there exists a context-free grammar G such that $L(G) = N(A)$.

Construction of G for CFG

We define $G = (V, \Sigma, P, S)$, where $V = \{S\} \cup \{[q, Z, q'] | q, q' \in Q, Z \in \tau\}$ i.e. any element of V is either the new symbol S acting as the start symbol for G or an ordered triple whose first and third elements are states and the second element is a pushdown symbol. The productions in P are induced by moves of pda as follows:

- R_1 S-productions are given by $S \to [q_0, Z_0, q]$ for every $q \in Q$.
- R_2 Each transition erasing a pushdown symbol given by $\delta(q, a, Z) = (q', \Lambda)$ induces the production $[q, Z, q'] \to a$

PDA to CFG

R_3 Each transition not erasing a pushdown symbol giving by $\delta(q, a, Z) = (q_1, Z_1 Z_2 \ldots Z_m)$ induces the production $[q, Z, q'] \rightarrow a[q_1, Z_1, q_2][q_2, Z_2, q_3] \ldots [q_m, Z_m, q']$, where each of the states $q', q_2, \ldots, q_m$ can be any state in Q

PDA to CFG

1. $S \to [q_0, Z_0, q_i]$
2. $\delta(q, a, Z) = (q', \Lambda)$
   $[q, Z, q'] \to a$
3. $\delta(q, \Lambda, Z) = (q', \Lambda)$
   $[q, Z, q'] \to \Lambda$
4. $\delta(q, a, Z) = (q', b)$
   $[q, Z, q_i] \to a[q', b, q_i]$
5. $\delta(q, a, Z) = (q', bX)$
   $[q, Z, q_i] \to a[q', b, \ldots][\ldots, X, q_i]$
6. $\delta(q, a, Z) = (q', bXY)$
   $[q, Z, q_i] = a[q', b, \ldots][\ldots, X, \ldots][\ldots, Y, q_i]$

PDA to CFG

Question: Construct a Context free grammar G which accepts N(A), where A=($\{q_0, q_1\}$,{a,b},$\{Z_0$,Z$\}$, δ, q_0, Z_0, ϕ) and δ is given by:

$\delta(q_0, b, Z_0) = (q_0, ZZ_0)$, $\delta(q_0, \Lambda, Z_0) = (q_0, \Lambda)$
$\delta(q_0, b, Z) = (q_0, ZZ)$, $\delta(q_0, a, Z) = (q_1, Z)$
$\delta(q_1, b, Z) = (q_1, \Lambda)$, $\delta(q_1, a, Z_0) = (q_0, Z_0)$

Solution: Let G=$(V_N, \{a, b\}, P, S)$

$V_N = \{S, [q_0, Z_0, q_0], [q_0, Z_0, q_1], [q_1, Z_0, q_0], [q_1, Z_0, q_1],$
$[q_0, Z, q_0], [q_0, Z, q_1], [q_1, Z, q_0], [q_1, Z, q_1]\}$

The Productions are:

Initial: S $\rightarrow [q_0, Z_0, q_0], [q_0, Z_0, q_1]$

For $\delta(q_0, b, Z_0) = (q_0, ZZ_0)$

$[q_0 Z_0 \ldots] \rightarrow b[q_0 Z \ldots][\ldots Z_0 \ldots]$
$[q_0 Z_0 \ldots] \rightarrow b[q_0 Z \ldots][\ldots Z_0 \ldots]$
$[q_0 Z_0 \ldots] \rightarrow b[q_0 Z \ldots][\ldots Z_0 \ldots]$
$[q_0 Z_0 \ldots] \rightarrow b[q_0 Z \ldots][\ldots Z_0 \ldots]$

PDA to CFG

Question: Construct a Context free grammar G which accepts N(A), where A=($\{q_0, q_1\}$,{a,b},$\{Z_0$,Z$\}$, δ, q_0, Z_0, ϕ) and δ is given by:

$\delta(q_0, b, Z_0) = (q_0, ZZ_0)$, $\delta(q_0, \Lambda, Z_0) = (q_0, \Lambda)$
$\delta(q_0, b, Z) = (q_0, ZZ)$, $\delta(q_0, a, Z) = (q_1, Z)$
$\delta(q_1, b, Z) = (q_1, \Lambda)$, $\delta(q_1, a, Z_0) = (q_0, Z_0)$

Solution: Let G=$(V_N, \{a, b\}, P, S)$

$V_N = \{S, [q_0, Z_0, q_0], [q_0, Z_0, q_1], [q_1, Z_0, q_0], [q_1, Z_0, q_1],$
$[q_0, Z, q_0], [q_0, Z, q_1], [q_1, Z, q_0], [q_1, Z, q_1]\}$

The Productions are:

Initial: S $\rightarrow [q_0, Z_0, q_0], [q_0, Z_0, q_1]$

For $\delta(q_0, b, Z_0) = (q_0, ZZ_0)$

$[q_0 Z_0 q_0] \rightarrow b[q_0 Z q_0][q_0 Z_0 q_0]$
$[q_0 Z_0 q_0] \rightarrow b[q_0 Z q_1][q_1 Z_0 q_0]$
$[q_0 Z_0 q_1] \rightarrow b[q_0 Z q_0][q_0 Z_0 q_1]$
$[q_0 Z_0 q_1] \rightarrow b[q_0 Z q_1][q_1 Z_0 q_1]$

PDA to CFG

For $\delta(q_0, \wedge, Z_0) = (q_0, \wedge)$

$[q_0 Z_0 q_0] \rightarrow \wedge$

$(q_0, b, Z) = (q_0, ZZ)$

$[q_0 Z \ldots] \rightarrow b[q_0 Z \ldots][\ldots Z \ldots]$

$[q_0 Z \ldots] \rightarrow b[q_0 Z \ldots][\ldots Z \ldots]$

$[q_0 Z \ldots] \rightarrow b[q_0 Z \ldots][\ldots Z \ldots]$

$[q_0 Z \ldots] \rightarrow b[q_0 Z \ldots][\ldots Z \ldots]$

PDA to CFG

For $\delta(q_0, \wedge, Z_0) = (q_0, \wedge)$

$[q_0 Z_0 q_0] \to \wedge$

$(q_0, b, Z) = (q_0, ZZ)$

$[q_0 Z q_0] \to b[q_0 Z q_0][q_0 Z q_0]$

$[q_0 Z q_0] \to b[q_0 Z q_1][q_1 Z q_0]$

$[q_0 Z q_1] \to b[q_0 Z q_0][q_0 Z q_1]$

$[q_0 Z q_1] \to b[q_0 Z q_1][q_1 Z q_1]$

For $(q_0, a, Z) = (q_1, Z)$

$[q_0 Z \ldots] \to a[q_1 Z \ldots]$

$[q_0 Z \ldots] \to a[q_1 Z \ldots]$

PDA to CFG

For $\delta(q_0, \wedge, Z_0) = (q_0, \wedge)$
$[q_0 Z_0 q_0] \to \wedge$
$(q_0, b, Z) = (q_0, ZZ)$
$[q_0 Z q_0] \to b[q_0 Z q_0][q_0 Z q_0]$
$[q_0 Z q_0] \to b[q_0 Z q_1][q_1 Z q_0]$
$[q_0 Z q_1] \to b[q_0 Z q_0][q_0 Z q_1]$
$[q_0 Z q_1] \to b[q_0 Z q_1][q_1 Z q_1]$
For $(q_0, a, Z) = (q_1, Z)$
$[q_0 Z q_0] \to a[q_1 Z q_0]$
$[q_0 Z q_1] \to a[q_1 Z q_1]$
For $\delta(q_1, b, Z) = (q_1, \wedge)$
$[q_1, Z, q_1] \to b$
For $\delta(q_1, a, Z_0) \to (q_0, Z_0)$
$[q_1, Z_0, \ldots] \to a[q_0 Z_0 \ldots]$
$[q_1, Z_0, \ldots] \to a[q_0 Z_0 \ldots]$

PDA to CFG

For $\delta(q_0, \wedge, Z_0) = (q_0, \wedge)$

$[q_0 Z_0 q_0] \rightarrow \wedge$

$(q_0, b, Z) = (q_0, ZZ)$

$[q_0 Z q_0] \rightarrow b[q_0 Z q_0][q_0 Z q_0]$

$[q_0 Z q_0] \rightarrow b[q_0 Z q_1][q_1 Z q_0]$

$[q_0 Z q_1] \rightarrow b[q_0 Z q_0][q_0 Z q_1]$

$[q_0 Z q_1] \rightarrow b[q_0 Z q_1][q_1 Z q_1]$

For $(q_0, a, Z) = (q_1, Z)$

$[q_0 Z q_0] \rightarrow a[q_1 Z q_0]$

$[q_0 Z q_1] \rightarrow a[q_1 Z q_1]$

For $\delta(q_1, b, Z) = (q_1, \wedge)$

$[q_1, Z, q_1] \rightarrow b$

For $\delta(q_1, a, Z_0) \rightarrow (q_0, Z_0)$

$[q_1, Z_0, q_0] \rightarrow a[q_0 Z_0 q_0]$

$[q_1, Z_0, q_1] \rightarrow a[q_0 Z_0 q_1]$

PDA to CFG

1. Let M=($\{q_0, q_1\}$,{a,b},{a,Z_0}, δ, q_0, Z_0, ϕ)
   where productions are
   $\delta(q_0, a, Z_0) = (q_0, aZ_0)$
   $\delta(q_0, a, a) = (q_0, aa)$
   $\delta(q_0, b, a) = (q_1, \wedge)$
   $\delta(q_1, b, a) = (q_1, \wedge)$
   $\delta(q_1, \wedge, Z_0) = (q_1, \wedge)$
   Find grammar G.
2. Find a context free grammar that generates the language accepted by NPDA
   M=($\{q_0, q_1\}$,{a,b},{A,Z},δ,q_0,Z,$\{q_1\}$) with transitions
   $\delta(q_0, a, Z) = (q_0, AZ)$
   $\delta(q_0, b, A) = (q_0, AA)$
   $\delta(q_0, a, A) = (q_1, \Lambda)$

PDA to CFG

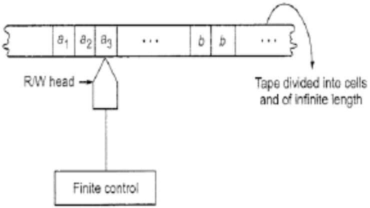
Example: Construct a PDA accepting $\{a^n b^m a^n | m, n \geq 1\}$ by null store. Construct the corresponding CFG accepting the same set.
Solution: The PDA 'A' accepting $\{a^n b^m a^n | m, n \geq 1\}$ is defined as
A=($\{q_0, q_1\}$,{a,b},{a,Z_0}, δ, q_0, Z_0, ϕ)
where δ is defined by:
$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, a)$
$\delta(q_1, b, a) = (q_1, a)$
$\delta(q_1, a, a) = (q_1, \wedge)$
$\delta(q_1, \wedge, Z_0) = (q_1, \wedge)$
Let the required grammar G=($V_N, \{a, b\}, P, S$)
$V_N =$
$(S, [q_0 Z_0 q_0], [q_0 Z_0 q_1], [q_1 Z_0 q_0], [q_1 Z_0 q_1], q_0 a q_0], [q_0 a q_1], [q_1 a q_0], [q_1 a q_1])$
...

Exercise

1. What do you understand by LL(k) grammar? Explain with a suitable example.
2. What do you understand by Parsing? How Top-Down Parsing is different from Bottom-Up Parsing? Explain with suitable example.
3. What is left factoring? How is it different from Left recursion?
4. Construct a PDA accepting the set of ll even-length palindromes over $\{a,b\}$ by the empty store.

Turing Machine

- A Turing Machine's storage can be visualized as a single, one dimensional array of cells, each of which can hold a single symbol.
- This array extends infinitely in both directions.
- Information can be read and changed in any order, such storage device is called **Tape**.
- Turing Machine has neither an input file nor any special output mechanism, whatever input and output is required will be done on machine's tape.

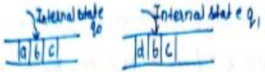


Turing Machine

- A Turing machine M is defined by
  M=(Q, $\Sigma, \tau, \delta, q_0, \square, F$)
  where, Q $=$ set of internal states
  Σ $=$ Input alphabet; $\Sigma \subseteq \tau - \{\square\}$
  τ $=$ finite set of symbols called tape alphabet
  δ $=$ transition function
  $Q \times \tau \rightarrow Q \times \tau \times \{L, R\}$
  $\square \in \tau$ $=$ special symbol called blank
  $q_0 \in Q$ $=$initial state
  $F \subseteq Q$ $=$ set of final states

Turing Machine

- The current state of the control unit and the current tape symbol being read determines the new state of the control unit and new tape symbol which replaces the old one and move the head L or R.
- $\delta(q_0, a) = (q_1, b, R)$



- The acceptability of a string is decided by the reachability from the initial state to some final state. So the final states are also called the accepting states.
- A Turing machine is said to **halt** whenever it reaches a configuration for which δ is not defined.
  $\implies$ No transitions are defined for any final state, so the Turing machine will halt whenever it enters a final state.

Turing Machine

- Consider the Turing machine defined by: Q=$\{q_0, q_1\}$, Σ=\{0,1\}, $\tau = \{0, 1, \Box\}$, F =$\{q_1\}$ and
  $\delta(q_0, 0) = (q_0, 1, R)$
  $\delta(q_0, 1) = (q_0, 0, R)$
  $\delta(q_0, \Box) = (q_1, \Box, L)$
- Q=$\{q_0, q_1, q_2\}$, Σ=\{a,b\}, $\tau = \{a, b, \Box\}$
  Let F be empty. Define δ by:
  $\delta(q_0, a) = (q_1, a, R)$
  $\delta(q_0, b) = (q_1, a, R)$
  $\delta(q_0, \Box) = (q_1, \Box, L)$
  $\delta(q_1, a) = (q_0, a, L)$
  $\delta(q_1, b) = (q_0, b, L)$
  $\delta(q_1, \Box) = (q_2, \Box, R)$

Representation of Turing Machines

Turing machine can be represented by tree ways:

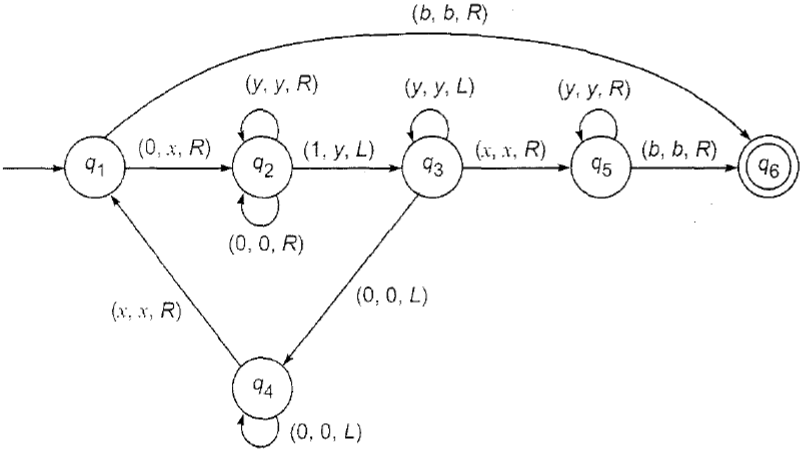
- Instantaneous Descriptions (ID) using move-relations
  - $\delta(q_1, x_i) = (q_2, y, R)$
    $x_1x_2 \dots q_1x_i \dots x_n \vdash x_1x_2 \dots yq_2x_{i+1} \dots x_n$
  - $\delta(q_1, x_i) = (q_2, y, L)$
    $x_1x_2 \dots q_1x_i \dots x_n \vdash x_1x_2 \dots q_2x_{i-1}y \dots x_n$
  - $\delta(q_1, x_n) = (q_2, y, R)$
    $x_1x_2 \dots q_1x_n \vdash x_1x_2 \dots yq_2\square$
    Because the tape is of infinite length having $\square$.
  - $\delta(q_1, x_1) = (q_2, y, L)$
    $q_1x_1x_2 \dots x_n \vdash q_2yx_2 \dots x_n$
    Because we prevent the machine from going off the left-hand end of the tape
- Transition table

Representation of Turing Machines

Present state	Tape symbol		
	b	0	1
$\rightarrow q_1$	$1Lq_2$	$0Rq_1$	
q_2	bRq_3	$0Lq_2$	$1Lq_2$
q_3		bRq_4	bRq_5
q_4	$0Rq_5$	$0Rq_4$	$1Rq_4$
(q_5)	$0Lq_2$		

- Transition diagram (transition graph)

Representation of Turing Machines



Language acceptability of Turing Machine

- Consider the Turing machine $M = (Q, \Sigma, \tau, \delta, q_0, \Box, F)$. A string $w \in \Sigma^*$ is said to be accepted by M if $q_o w \vdash^* \alpha_1 q \alpha_2$ for some $q \in F$ and $\alpha_1, \alpha_2 \in \tau^*$
- M does not accept w if the machine M either halts in a non-accepting state or does not halt.
- There are other equivalent definitions of acceptance by the Turing machine, we will not discuss them now.

Design of TM

The basic guidelines for designing a Turing machine:

- The fundamental objective in scanning a symbol by the R/W head is to know what to do in the future. The machine must remember the past symbols scanned. The Turing machine can remember this by going to the next unique state.
- The number of states must be minimized. This can be achieved by changing the states only when there is a change in the written symbol or when there is a change in the movement of the R/W head.

Design of Turing Machine

Question: For $\Sigma = \{a,b\}$, design a Turing machine that accepts
L=$\{a^n b^n : n \geq 1\}$
Solution:

- Start with left most 'a', replace it by 'x'
- Travel righ to find left most 'b', replace it by 'y'.
- Move left again to find left most 'a', replace by 'x, then again right to left most 'b', replace by 'y'.
- Continue moving right and left till no 'a' and 'b' remains, then the string must be in L.

Q=$\{q_0, q_1, q_2, q_3, q_4\}$, F=$\{q_4\}$
$\Sigma = \{a, b\}$, $\tau = \{a, b, x, y, \square\}$
$\delta(q_0, a) = (q_1, x, R) \implies$ replaces 'a' by 'x'
$\delta(q_1, a) = (q_1, a, R) \implies$ move right
$\delta(q_1, y) = (q_1, y, R) \implies$ move right
$\delta(q_1, b) = (q_2, y, L) \implies$ 'a' paired with 'b'
Move left to find 'x'
$\delta(q_2, y) = (q_2, y, L) \implies$ move left
$\delta(q_2, a) = (q_2, a, L) \implies$ move left
$\delta(q_2, x) = (q_0, x, R) \implies$ placed at first 'a'
Check for all 'a' and 'b' are replaced
$\delta(q_0, y) = (q_3, y, R)$
$\delta(q_3, y) = (q_3, y, R)$

Design of Turing Machine

$\delta(q_3, \Box) = (q_4, \Box, R)$
For input 'aabb'

$q_0 aabb \vdash xq_1 abb \vdash xaq_1 bb \vdash xq_2 ayb \vdash q_2 xayb \vdash xq_0 ayb \vdash xxq_1 yb \vdash xxyq_1 b \vdash$

$xxq_2 yy \vdash xq_2 xyy \vdash xxq_0 yy \vdash xxyq_3 y \vdash xxyyq_3 \vdash xxyy\Box q_4 \Box$

Design of Turing machine

Question: Design a Turing Machine that accepts
L=$\{a^n b^n c^n : n \geq 1\}$

Turing Computable

A function 'f' with domain 'D' is said to be Turing computable or just computable, if there exists some Turing machine M=(Q, $\Sigma, \tau, \delta, q_0, \Box$, F) such that $q_0 w \vdash_M^* q_f f(w), q_f \in F$ for all w$\in D$.

Turing Computable

Example

Given two positive integers 'x' and 'y'. Design a Turing machine that computes x+y

Solution: Using Unary notation in which any positive integer 'x' is represented by w(x) $\in \{1\}^+$ such that |w(x)|=x.

So, the required machine is: $q_0 w(x) 0 w(y) \vdash * q_f w(x+y) 0$

Steps: Move the separating 0 to right end of w(y)

Let M=(Q,$\Sigma, \tau, q_0, \Box, F$)

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

$\delta(q_0, 1) = (q_0, 1, R)$, $\delta(q_0, 0) = (q_1, 1, R)$, $\delta(q_1, 1) = (q_1, 1, R)$,

$\delta(q_1, \Box) = (q_2, \Box, L)$, $\delta(q_2, 1) = (q_3, 0, L)$, $\delta(q_3, 1) = (q_3, 1, L)$

$\delta(q_3, \Box) = (q_4, \Box, R)$

Adding 111 to 11

$q_0 111011 \vdash 1 q_0 11011 \vdash 11 q_0 1011 \vdash 111 q_0 011 \vdash 1111 q_1 11 \vdash \ldots$

Variation of TM

- Turing Machine with Stationary Head
- Storage in the State
- Multiple Track Turing Machine
- Subroutines
- Multitape Turing Machines
- Nondeterministic Turing Machines

Universal TM

A universal Turing machine is a Turing machine T_u that works as follows:

- It is assumed to receive an input string of the form $e(T)e(z)$, where T is an arbitrary TM, z is a string over the input alphabet of T, and e is an encoding function whose values are strings in $\{0,1\}^*$. The computation performed by T_u on this input string satisfies two properties:
  1. T_u accepts the string $e(T)e(z)$ if and only if T accepts z.
  2. If T accepts z and produces output y, then T_u produces output $e(y)$.

Encoding Function

We assume that there is an infinite set $\mathbb{S} = \{a_1, a_2, a_3, \dots\}$ of symbols, where $a_1 = \Delta = blank$, such that the tape alphabet of every Turing machine T is a subset of $\mathbb{S}$.
If $T = (Q, \Sigma, \tau, q_0, \delta)$ is a TM and z is a string, we define the strings $e(T)$ and $e(z)$ as follows:

- First we assign numbers to each state, tape symbol, and tape head direction of T.
- Each tape symbol, including Δ, is an element a_i of $\mathbb{S}$, and it is assigned the number $n(a_i) = i$.
- The accepting state h_a, the rejecting state h_r, and the initial state q_0 are assigned the numbers $n(h_a) = 1$, $n(h_r) = 2$, and $n(q_0) = 3$.
- The other elements $q \in Q$ are assigned distinct numbers $n(q)$, each at least 4.

Encoding Function

- We don't require the numbers to be consecutive, and the order is not important.
- The three directions R, L, and S are assigned the numbers $n(R) = 1, n(L) = 2$, and $n(S) = 3$
- For each move m of T of the form $\delta(p, a) = (q, b, D)$

$$e(m) = 1^{n(p)}01^{n(a)}01^{n(q)}01^{n(b)}01^{n(D)}0$$

- List the moves of T in any order as $m_1, \ldots, m_k$. and define $e(T)$ as:

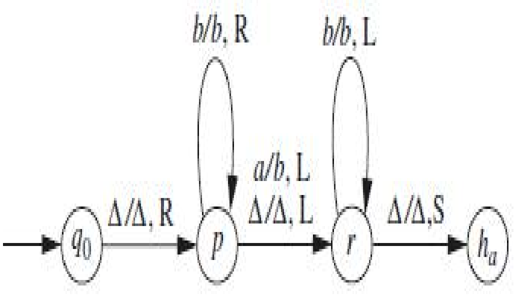
$$e(T) = e(m_1)0e(m_2)0\ldots)0(m_k)0$$

- If $z = z_1 z_2 \ldots z_j$ is a string, where each $z_i \in \mathbb{S}$

$$e(z) = 01^{n(z_1)}01^{n(z_2)}0\ldots01^{n(z_j)}0$$

Encoding Function

- The input to UTM will be $e(T)e(z)$
- Example: Let T be the TM shown in below figure, which transforms an input string of a's and b's by changing the leftmost a, if there is one, to b.



Encoding Function

- Solutions:We assume for simplicity that $n(a) = 2$ and $n(b) = 3$. By definition, $n(q_0) = 3$, and we let $n(p) = 4$ and $n(r) = 5$.
- If m is the move determined by the formula $\delta(q_0, \Delta) = (p, \Delta, R)$, then

$$e(m) = 1^3 0101^4 01010 = 111010111101010$$

- if we encode the moves in the order they appear in the diagram, left to right
$e(T) = 1110101111010100111101110111101110100$
$11110110111110111011001111010111110101100$
$1111101110111110111011001111101010101011100$
- Let the string to be checked for acceptance is Δaab

$$e(z) = 0101101101110$$

Encoding Function

- Input to the UTM is $e(T)e(z)$
  $= 111010111010100111101110111101110100$
  $1111011011111011101100111101011111010100$
  $111110111011111011101100111110101010111000101101101110$
- We will use three tapes. The first tape is for input and output and originally contains the string $e(T)e(z)$, where T is a TM and z is a string over the input alphabet of T.
- The second tape will correspond to the working tape of T , during the computation that simulates the computation of T on input z.
- The third tape will have only the encoded form of T's current state.
- T_u starts by transferring the string $e(z)$, except for the initial 0, from the end of tape 1 to tape 2, beginning in square 3.

Encoding Function

- Since T begins with its leftmost square blank, T_u writes 10, the encoded form of Δ, in squares 1 and 2.
- Square 0 is left blank, and the tape head begins on square 1.
- The second step is for T_u to write 111, the encoded form of the initial state q_0, on tape 3, beginning in square 1.
- Δ111010111010100111011101111011101001110110111110111011
  Δ10110110111Δ
  Δ111Δ
- Now we simulate the UTM by finding the pattern for state q from tape 3 followed by code of the $0e(z_i)0$ from tape 2 under R/W head.
- When pattern is found, copy 1st part as state on tape 3, replace $e(z_i)$ by 2nd part from tape 1 and move the R/W head as per the encoded value of direction in part 3 on tape 1.

Encoding Function

- We repeat the above two steps until we found state $h_a = 1$
- In the last, when T halt with acceptance means on the 3rd tape $h_a = 1$, we erase the contents of 1st tape and copy the encoded output of UTM on 2nd tape to the 1st tape.

Church-Turing Thesis

To say that the Turing machine is a general model of computation means that any algorithmic procedure that can be carried out at all, by a human computer or a team of humans or an electronic computer, can be carried out by a TM. This statement was first formulated by Alonzo Church in the 1930s and is usually referred to as Church's thesis, or the Church-Turing thesis. It is not a mathematically precise statement that can be proved, because we do not have a precise definition of the term algorithmic procedure. By now, however, there is enough evidence for the thesis to have been generally accepted. Here is an informal summary of some of the evidence.

Church-Turing Thesis

- The nature of the model makes it seem likely that all the steps crucial to human computation can be carried out by a TM. Humans normally work with a two-dimensional sheet of paper, and a human computer may perhaps be able to transfer his attention to a location that is not immediately adjacent to the current one, but enhancements like these do not appear to change the types of computation that are possible. A TM tape could be organized so as to simulate two dimensions; one likely consequence would be that the TM would require more moves to do what a human could do in one.
- Various enhancements of the TM model have been suggested in order to make the operation more like that of a human computer, or more convenient, or more efficient. The multitape TM discussed is an example. In each case, it has been shown that the computing power of the device is unchanged.

Church-Turing Thesis

- Other theoretical models of computation have been proposed. These include abstract machines such as the ones with two stacks or with a queue, as well as machines that are more like modern computers. In addition, various notational systems (programming languages, grammars, and other formal mathematical systems) have been suggested as ways of describing or formulating computations. Again, in every case, the model has been shown to be equivalent to the Turing machine.
- Since the introduction of the Turing machine, no one has suggested any type of computation that ought to be included in the category of "algorithmic procedure" and cannot be implemented on a TM.

Characteristic Function

For a language $L \subseteq \Sigma^*$, the characteristic function of L is the function $\chi_L : \Sigma^* \rightarrow \{0, 1\}$ defined by

$$\chi_L(x) = \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{if } x \notin L \end{cases}$$

- Computing the function χ_L and accepting the language L are two approaches to the question of whether an arbitrary string is in L or not.
- A TM computing χ_L indicates whether the input string is in L by producing output 1 or output 0.
- A TM accepting L indicates the same thing by accepting or not accepting the input.

Accepting a Language and Deciding a Language

- A Turing machine T with input alphabet Σ accepts a language $L \subseteq \Sigma^*$ if $L(T) = L$.
- T decides L if T computes the characteristic function $\chi_L : \Sigma^* \rightarrow \{0, 1\}$.
- A language L is recursively enumerable if there is a TM that accepts L, and L is recursive if there is a TM that decides L.
- Recursively enumerable languages are sometimes referred to as Turing-acceptable, and recursive languages are sometimes called Turing-decidable, or simply decidable.
- Every recursive language is recursively enumerable.
- The main difference is that in recursively enumerable language the machine halts for input strings which are in language L. but for input strings which are not in L, it may halt or may not halt. When we come to recursive language it always halt whether it is accepted by the machine or not.

Accepting a Language and Deciding a Language

- If $L \subseteq \Sigma^*$ is accepted by a TM T that halts on every input string, then L is recursive.
- If $L \subseteq \Sigma^*$ is accepted by a TM T that halts on every input string x when $x \in L$ and may or may not halt when $x \notin L$ then L is recursively enumerable.

Unrestricted Grammars

- The unrestricted grammars correspond to recursively enumerable languages in the same way that CFGs correspond to languages accepted by PDAs and regular grammars to those accepted by DFAs
- An unrestricted grammar is a 4-tuple $G = (V, \Sigma, P, S)$, where V and Σ are disjoint sets of variables and terminals, respectively. S is an element of V called the start symbol, and P is a set of productions of the form $\alpha \to \beta$ where $\alpha, \beta \in (V \cup \Sigma)^*$ and α contains at least one variable
- For every unrestricted grammar G, there is a Turing machine T with $L(T) = L(G)$.
- For every Turing machine T with input alphabet Σ, there is an unrestricted grammar G generating the language $L(T) \subseteq \Sigma^*$.

Unrestricted Grammars

- A context-sensitive grammar (CSG) is an unrestricted grammar in which no production is length-decreasing.
- In other words, every production is of the form $\alpha \rightarrow \beta$, where $|\beta| \geq |\alpha|$.
- No variable is allowed in β whose value is null.
- A language is a context-sensitive language (CSL) if it can be generated by a context-sensitive grammar.
- Example Write the production for the language $L = \{a^n b^n c^n | n \geq 1\}$

Unrestricted Grammars

- Solution: $S \rightarrow SABC|\mathbb{A}BC$,
  $BA \rightarrow AB$,
  $CA \rightarrow AC$,
  $CB \rightarrow BC$,
  $\mathbb{A} \rightarrow a$,
  $aA \rightarrow aa$,
  $aB \rightarrow ab$,
  $bB \rightarrow bb$,
  $bC \rightarrow bc$,
  $cC \rightarrow cc$.

Linear Bounded Automata

- This model is important because (a) the set of context-sensitive languages is accepted by the model and (b) the infinite storage is restricted in size but not in accessibility to the storage in comparison with the Turing machine model.
- It is called the linear bounded automaton (LBA) because a linear function is used to restrict (to bound) the length of the tape.
- A linear bounded automaton is a non-deterministic Turing machine which has a single tape whose length is not infinite but bounded by a linear function of the length of the input string.
- The models can be described formally by the following set format:
  $M = (Q, \Sigma, \tau, \delta, q_0, b, \S, \$, F)$

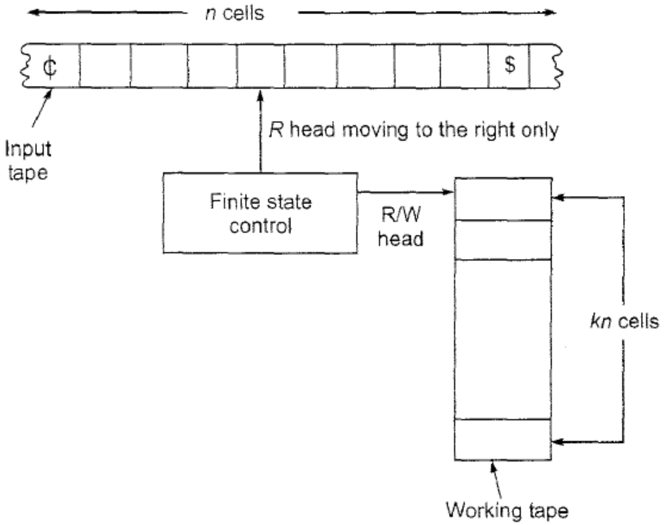
Linear Bounded Automata

- All the symbols have the same meaning as in the basic model of Turing machines with the difference that the input alphabet Σ contains two more special symbols $\S$ and \$ also.
- $\S$ is called the left-end marker which is entered in the leftmost cell of the input tape and prevents the R/W head from getting off the left end of the tape.
- \$ is called the right-end marker which is entered in the rightmost cell of the input tape and prevents the R/W head from getting off the right end of the tape.
- Both the end-markers should not appear on any other cell within the input tape, and the R/W head should not print any other symbol over both the end-markers.
- Let us consider the input string w with $|w| = n - 2$.

Linear Bounded Automata

- The input string w can be recognized by an LBA if it can also be recognized by a Turing machine using no more than kn cells of input tape, where k is a constant specified in the description of LBA.
- The value of k does not depend on the input string but is purely a property of the machine.
- Whenever we process any string in LBA, we shall assume that the input string is enclosed within the end-markers § and $.
- The model of LBA can be represented by the block below diagram:

Linear Bounded Automata



Linear Bounded Automata

- There are two tapes: one is called the input tape, and the other is working tape.
- On the input tape the head never prints and never moves to the left.
- On the working tape the head can move in any direction Left or Right and can modify the contents in any way, without any restriction.
- In the case of LBA, an ID is denoted by (q, w, i), where $q \in Q, w \in \tau$ and i is some integer between 1 and n.
- The transition of IDs is similar except that i changes to $i - 1$ if the R/W head moves to the left and to $i + 1$ if the head moves to the right.

Linear Bounded Automata

- The language accepted by LBA is defined as the set

$$\{w \in \{\Sigma - \{\S, \$\}\}^* | (q_0, \S w\$, 1) \vdash^* (q, \alpha, i)\}$$

  for some $q \in F$ and for some integer i between 1 and n, $\alpha \in \tau^*$.
- As a null string can be represented either by the absence of input string or by a completely blank tape, an LBA may accept the null string.
- A linear bounded automaton M accepts a string w if, after starting at the initial state with R/W head reading the left-endmarker, M halts over the right-endmarker in a final state. Otherwise, w is rejected
- The set of strings accepted by non-deterministic LBA is the set of strings generated by the context sensitive grammars, excluding the null strings.

Linear Bounded Automata

- If L is a context-sensitive language, then L is accepted by a linear bounded automaton and vice versa.
- Exercise; Design the LBA for the language $L = \{a^n b^n c^n | n \geq 1\}$

Construction of Grammar Corresponding to TM

- For understanding the construction. we have to note that a transition of ID corresponds to a production.
- We enclose IDs within brackets. So acceptance of w by M corresponds to the transformation of initial ID $[q_0, \S w\$]$ into $[q_f b]$.
- Also, the 'length' of ID may change if the R/W head reaches the left-end or the right-end, i.e. when the left-hand side or the right-hand side bracket is reached.
- So we get productions corresponding to transition of IDs with (i) Left move (ii) Right move, and (iii) end-markers.
- Right move:

Construction of Grammar Corresponding to TM

1.

$$\delta(q_i, a_j) = (q_l, a_k, R)$$

ID $a_m q_i a_j a_{m+1} \vdash a_m a_k q_l a_{m+1}$
leads to the production

$$q_i a_j \rightarrow a_k q_l$$

2. When at the right-end and right move. When the R/W head moves to the right of], the length increases. That is

$$\delta(q_i,]) = (q_i, \Box, R)$$

ID $a_m q_i] \vdash a_m q_i \Box]$
Corresponding to this we have a production

$$q_i] \rightarrow q_i \Box]$$

for all $q_i \in Q$

Construction of Grammar Corresponding to TM

3. When $\square$ occurs to the left of], it can be deleted. This is achieved by the production

$$a_j\square] \rightarrow a_j]$$

for all $a_j \in \tau$

- Left move:

1.

$$\delta(q_i, a_j) = (q_l, a_k, L)$$

ID $a_m q_i a_j \vdash q_l a_m a_k$ leads to the production

$$a_m q_i a_j \rightarrow q_l a_m a_k$$

for all $a_m \in \tau$

Construction of Grammar Corresponding to TM

2. When at the left-end and left move

$$\delta(q_i, a_j) = (q_l, a_k, L)$$

ID $[q_i a_j \vdash [q_l \Box a_k$
leads to the production

$$[q_i a_j \rightarrow [q_l \Box a_k$$

3. When $\Box$ occurs next to the left-bracket, it can be deleted. This is achieved by including the production

$$[\Box \rightarrow [$$

- Introduction of end-markers: For introducing end-markers for the input string, the following productions are included, where q_0 is the initial state and q_f is the final state:
  1. $a_i \rightarrow [q_0 \S a_i$ for $a_i \in \tau, a_i \neq \Box$
  2. $a_i \rightarrow a_i \$]$ for $a_i \in \tau, a_i \neq \Box$

Construction of Grammar Corresponding to TM

3. For removing the brackets from $[q_f \square]$, we include the production

$$[q_f \square] \rightarrow S$$

- To get the required grammar, reverse the arrows of the productions obtained above.
- The productions we get can be called inverse productions.
- The new grammar is called the generative grammar.

Construction of Grammar Corresponding to LBA

- A linear bounded automaton M accepts a string w if, after starting at the initial state with R/W head reading the left-end marker, M halts over the right-end marker in a final state. Otherwise, w is rejected.
- The production rules for the generative grammar are constructed as in the case of Turing machines.
- The following additional productions are needed in the case of LBA:
  1. $a_i q_f \$ \rightarrow q_f \$$ for all $a_i \in \tau$
  2. $\S q_f \$ \rightarrow \S q_f$
  3. $\S q_f \rightarrow q_f$

Exercise: Find the grammar generating the set accepted by a linear bounded automaton M whose transition table is given below:

Construction of Grammar Corresponding to LBA

Present state	Tape input symbol			
	¢	\$	0	1
$\rightarrow q_1$	$¢Rq_1$		$1Lq_2$	$0Rq_2$
q_2	$¢Rq_4$		$1Rq_3$	$1Lq_1$
q_3		$\$Lq_1$	$1Rq_3$	$1Rq_3$
(q_4)		Halt	$0Lq_4$	$0Rq_4$

CYK Algorithm

- The algorithm is called the CYK algorithm, after its originators J. Cocke, D. H. Younger, and T. Kasami.
- The algorithm works only if the grammar is in Chomsky normal form and succeeds by breaking one problem into a sequence of smaller ones in the following way.
- Assume that we have a grammar G = (V, T, S, P) in Chomsky normal form and a string $w = a_1 a_2 \dots a_n$.
- Define substrings $w_{ij} = a_i \dots a_j$
- Define subsets of V as $V_{ij} = \{A \in V : A \Rightarrow^* w_{ij}\}$
- Clearly, $w \in L(G)$ if and only if $S \in V_{1n}$.
- To compute V_{ii}, observe that $A \in V_{ii}$ if and only if G contains a production $A \to a_i$.
- Therefore, V_{ii} can be computed for all $1 \leq i \leq n$ by inspection of w and the productions of the grammar.

CYK Algorithm

- To compute V_{ij} for $i < j$, A derives w_{ij} if and only if there is a production $A \to BC$, with $B \Rightarrow^* w_{ik}$ and $C \Rightarrow^* w_{k+1j}$ for some k with $i \leq k < j$.
- In other words, $V_{ij} = \cup_{k=i \dots j-1}\{A : A \to BC, B \in V_{ik}, C \in V_{k+1j}$ $\Rightarrow \{A_1 | A_1 \to \{V_{ii}\}\{V_{i+1j}\}\} \cup \{A_2 | A_2 \to \{V_{ii+1}\}\{V_{i+2j}\}\} \cup \dots \cup \{A_l | A_l \to \{V_{ik}\}\{V_{k+1j}\}\} \cup \dots \cup \{A_{n-1} | A_{n-1} \to \{V_{ij-2}\}\{V_{j-1j}\}\} \cup \{A_n | A_n \to \{V_{ij-1}\}\{V_{jj}\}\}$
- Compute all the V_{ij} using the above eq. as:
  1. Compute $V_{11}, V_{22}, \dots, V_{nn}$
  2. Compute $V_{12}, V_{23}, \dots, V_{n-1n}$
  3. Compute $V_{13}, V_{24}, \dots, V_{n-2n}$
  4. ...
  5. Compute V_{1n}
- If $S \in V_{1n}$ then $w \in L(G)$ otherwise $w \notin L(G)$

CYK Algorithm

Exercise: Determine whether the string $w = aabbb$ is in the language generated by the grammar:

$$S \to AB$$

$$A \to BB|a$$

$$B \to AB|b$$

Some basic definition

- When a Turing machine reaches a final state, it halts.
- We can also say that a Turing machine M halts when M reaches a state q and a current symbol a to be scanned so that $\delta(q, a)$ is undefined.
- There are TMs that never halt on some inputs in any one of these ways.
- So we make a distinction between the languages accepted by a TM that halts on all input strings and a TM that never halts on some input strings.
- Recursively Enumerable: A language $L \subseteq \Sigma^*$ is recursively enumerable if there exists a TM M, such that $L = T(M)$.
- Recursive: A language $L \subseteq \Sigma^*$ is recursive if there exists some TM M that satisfies the following two conditions:
  1. If $w \in L$ then M accepts w, that is. reaches an accepting state on processing w and halts.

Some basic definition

2. If $w \notin L$ then M eventually halts, without reaching an accepting state.

- Decidable:A problem with two answers (Yes/No) is decidable if the corresponding language is recursive. In this case, the language L is also called decidable.
- Undecidable:A problem/language is undecidable if it is not decidable.
- A decidable problem is called a solvable problem and an undecidable problem an unsolvable problem by some authors.
- $A_{DFA} = \{(B, w) | B$ accepts the input string $w\}$
- $A_{CFG} = \{(G, w)|$ The context-free grammar G accepts the input string $w\}$
- $A_{CSG} = \{(G, w)|$ The context-sensitive grammar G accepts the input string $w\}$
- $A_{TM} = \{(M, w)|$ The TM M accepts $w\}$

Some basic definition

- A_{DFA} is decidable.
- A_{CFG} is decidable.
- A_{CSG} is decidable.
- A_{TM} is undecidable.

Turing machine halting Problem

- The reduction technique is used to prove the undecidability of halting problem of Turing machine
- We say that problem A is reducible to problem B if a solution to problem B can be used to solve problem A.
- If A is reducible to B and B is decidable then A is decidable. If A is reducible to B and A is undecidable, then B is undecidable.
- Theorem $HALT_{TM} = \{(M, w) |$ The Turing machine M halts on input $w\}$ is undecidable.
  Proof: We assume that $HALT_{TM}$ is decidable, and get a contradiction. Let M_1 be the TM such that $T(M_1) = HALT_{TM}$ and let M_1 halt eventually on all (M, w). 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).

Turing machine halting Problem

4. If M_1 accepts (M, w), simulate the TM M on the input string w until M halts.
5. If M has accepted w, M_2 accepts (M, w); otherwise M_2 rejects (M, w).

- When M_1 accepts (M, w) (in step 4), the Turing machine M halts on w.
- 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.
- In the first case (the first alternative of step 5) M_2 accepts $(M.w)$.
- In the second case (the second alternative of step 5) M_2 rejects (M, w).
- It follows from the definition of M_2 that M_2 halts eventually.
- $TM_2 = \{(M, w) |$ The Turing machine accepts $w\} = A_{TM}$
- This is a contradiction since A_{TM} is undecidable.

Post correspondence problems (PCP)

- The Post Correspondence Problem (PCP) was first introduced by Emil Post in 1946.
- The problem over an alphabet Σ belongs to a class of yes/no problems and is stated as follows:
- Consider the two lists $x = (x_1 \ldots x_n), y = (y_1 \ldots y_n)$ of non-empty strings over an alphabet $\Sigma = \{0, 1\}$.
- The PCP is to determine whether or not there exist $i_1, \ldots, i_m$, where $1 \leq i_j \leq n$ such that

$$x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m}$$

- The indices i_j's need not be distinct and m may be greater than n. Also, if there exists a solution to PCP, there exist infinitely many solutions.

Modified Post correspondence problems

- If the first substring used in PCP is always x_1 and y_1 then the PCP is known as the Modified Post Correspondence Problem.

Partial and Total Functions

- A Partial Function f from X to Y ($f : X \to Y$) is a rule which assigns to every element of X at most one element of Y.
- Example: if R denotes the set of all real numbers, the rule f from R to R given by $f(r) = +\sqrt{r}$; is a partial function since $f(r)$ is not defined as a real number when r is negative.
- A Total Function f from X to Y is a rule which assigns to every element of X a unique element of Y.
- Example:The rule f from R to R given by $f(r) = |r|$ is a total function since $f(r)$ is defined for every real number r.
- We consider total functions f from X^k to X, where $X = \{0, 1, 2, 3, \dots\}$ or $X = \{a, b\}^*$.
- We denote $\{0, 1, 2, \dots\}$ by N and $\{a, b\}$ by Σ.
- X^k is the set of all k-tuples of elements of X.
- For example, $f(m, n) = m - n$ defines a partial function from N to itself as $f(m, n)$ is not defined when $m - n < 0$.

Partial and Total Functions

- But $g(m,n) = m + n$ defines a total function from N to itself.
- A partial or total function f from X^k to X is also called a function of k variables and denoted by $f(x_1, X_2, \ldots, X_k)$.
- For example, $f(x_1, x_2) = 2x_1 + x_2$ is a function of two variables: $f(1,2) = 4; 1$ and 2 are called arguments and 4 is called a value.
- $g(w_1, w_2) = w_1 w_2$ is a function of two variables $w_1, w_2 \in \Sigma^* : g(ab, aa) = abaa, ab, aa$ are called arguments and $abaa$ is a value.

Primitive Recursive functions

- The initial functions over N are given as:
  1. Zero function Z defined by $Z(x) = 0$
  2. Successor function S defined by $S(x) = x + 1$
  3. Projection function U_i^n defined by $U_i^n(x_1, \ldots, x_n) = x_i$
  4. As $U_1^1(x) = x$ for every x in N. U_1^1 is simply the identity function. So U_i^n is also termed a generalized identity function.
- The initial functions over Σ are given as:
  1. nil(x) defined by $nil(x) = \wedge$
  2. cons a(x) defined by $cons\ a(x) = ax$
  3. cons b(x) defined by $cons\ b(x) = bx$

Primitive Recursive functions

- Example:

$$Z(7) = 0$$

$$S(4) = 5$$

$$U_2^3\{2, 5, 7\} = 5$$

$$nil(aabb) = \wedge$$

$$cons\ \ a(aabb) = aaabb$$

$$cons\ \ b(aabb) = baabb$$

Primitive Recursive functions

- Composition of a function: If $f_1, f_2, \ldots, f_k$ are partial functions of n variables and g is a partial function of k variables, then the composition of g with $f_1, f_2, \ldots, f_k$ is a partial function of n variables defined by

$$g(f_1(x_1, x_2, \ldots, x_n), f_2(x_1, x_2, \ldots, x_n), \ldots, f_k(x_1, x_2, \ldots, x_n))$$

- The composition of g with $f_1, f_2, \ldots, f_n$ is total when $g, f_1, f_2, \ldots, f_n$ are total.
- Example: Let $f_1(x, y) = x + y$, $f_2(x, y) = 2x$, $f_3(x, y) = xy$ and $g(x, y, z) = x + y + z$ be functions over N. Find the composition of g with f_1, f_2, f_3
- Solution: The composition of g with f_1, f_2, f_3 is given by $h(x, y) = g(f_1(x, y), f_2(x, y), f_3(x, y)) = (x + y) + (2x) + (xy)$ $= x + y + 2x + xy$

Primitive Recursive functions

- A function $f(x)$ over N is defined by recursion if there exists a constant k (a natural number) and a function $h(x, y)$ such that $f(0) = k, f(n+1) = h(n, f(n)$
- Example: Define $n!$ by recursion.
- Solution: Let $f(0) = 1$ and $f(n+1) = h(n, f(n))$, where $h(x, y) = S(x) * y$.
  So $f(n)$ will be
  $f(n) = h(n-1, f(n-1)) = S(n-1) * f(n-1) = n * f(n-1)$
- A function f of $n+1$ variables is defined by recursion if there exists a function g of n variables, and a function h of $n+2$ variables, and f is defined as follows:
  $f(x_1, x_2, \ldots, x_n, 0) = g(x_1, x_2, \ldots, x_n)$
  $f(x_1, x_2, \ldots, x_n, y+1) =$
  $h(x_1, x_2, \ldots, x_n, y, f(x_1, x_2, \ldots, x_n, y))$

Primitive Recursive functions

- A total function f over N is called primitive recursive
  (i) if it is anyone of the three initial functions, or
  (ii) if it can be obtained by applying composition and recursion a finite number of times to the set of initial functions.
- A total function is primitive recursive if it can be obtained by applying composition and recursion a finite number of times to primitive recursive functions $f_1, f_2, \ldots, f_m$. Each f_i is obtained by applying composition and recursion a finite number of times to initial functions.

Primitive Recursive functions

- A function $f(x)$ over Σ is defined by recursion if there exists a 'constant' string $w \in \Sigma^*$ and functions $h_1(x, y)$ and $h_2(x.y)$ such that

$$f(\wedge) = w$$

$$f(ax) = h_1(x, f(x))$$

$$f(bx) = h_2(x, f(x))$$

h_1 and h_2 may be functions in one variable.

Primitive Recursive functions

- A function $f(x_1, x_2, \dots, x_n)$ over Σ is defined by recursion if there exists a function $g(x_1, x_2, \dots, x_{n-1})$, $h_1(x_1, x_2, \dots, x_{n+1})$, $h_2(x_1, x_2, \dots, x_{n+1})$ such that

$$f(\wedge, x_2, \dots, x_n) = g(x_2, \dots, x_n)$$

$$f(ax_1, x_2, \dots, x_n) = h_1(x_1, x_2, \dots, x_n, f(x_1, x_2, \dots, x_n))$$

$$f(bx_1, x_2, \dots, x_n) = h_2(x_1, x_2, \dots, x_n, f(x_1, x_2, \dots, x_n))$$

  h_1 and h_2 may be functions of m variables, where $m < n + 1$.
- A total function f over Σ is primitive recursive
  (i) if it is anyone of the three initial functions, or
  (ii) if it can be obtained by applying composition and recursion a finite number of times to the initial functions.

Recursive functions

- Let $g(x_1, x_2, \ldots, x_n, y)$ be a total function over N. The function g is a regular function if there exists some natural number y_0 such that $g(x_1, x_2, \ldots, x_n, y_0) = 0$ for all values $x_1, x_2, \ldots, x_n \in N$.
- Example: $g(x, y) = min(x, y)$ is a regular function since $g(x, 0) = 0$ for all $x \in N$.
- But $f(x, y) = |x - y|$ is not regular since $f(x, y) = 0$ only when $x = y$, and so we cannot find a fixed y such that $f(x, y) = 0$ for all x in N.
- A function $f(x_1, x_2, \ldots, x_n)$ over N is defined from a total function $g(x_1, x_2, \ldots, x_n, y)$ by minimization if
  (a) $f(x_1, x_2, \ldots, x_n)$ is the least value of all y's such that $g(x_1, x_2, \ldots, x_n, y) = 0$ if it exists. The least value is denoted by $\mu_y(g(x_1, x_2, \ldots, x_n, y) = 0)$.
  (b) $f(x_1, x_2, \ldots, x_n)$ is undefined if there is no y such that $g(x_1, x_2, \ldots, x_n, y) = 0$

Recursive functions

- In general, f is partial. But, if g is regular then f is total.
- A function is recursive if it can be obtained from the initial functions by a finite number of applications of composition, recursion and minimization over regular functions.
- A function is partial recursive if it can be obtained from the initial functions by a finite number of applications of composition, recursion and minimization.
- Example: Show that $f(x) = x/2$ is a partial recursive function over N.
- Solution: Let $g(x, y) = |2y - x|$ where $2y - x = 0$ for some y only when x is even. Let $f_1(x) = \mu_y(|2y - x| = 0)$. Then $f_1(x)$ is defined only for even values of x and is equal to $x/2$. When x is odd, $f_1(x)$ is not defined $f_1(x)$ is partial recursive. As $f(x) = x/2 = f_1(x)$ is a partial recursive function.
- Exercise: Show that $f(x, y) = x^2y^4 + 7xy^3 + 4y^5$ is primitive recursive.

References

1. Mishra, K. L. P., and N. Chandrasekaran. "Theory of computer science: automata, languages and computation". PHI Learning Pvt. Ltd., 2006.
2. Hopcroft, John E., and Jefferey D. Ullman. "Introduction to Automata Theory, Languages, and Computation. Adison." (1979).
3. Cohen, Daniel IA. Introduction to computer theory. John Wiley & Sons, 1996.
4. Martin, John C. Introduction to Languages and the Theory of Computation. McGraw-Hill Higher Education, 2011.
5. Sipser, Michael. "Introduction to the theory of computation" Computer Science Series. Thomson Course Technology (2006).
6. Linz, Peter, and Susan H. Rodger. An introduction to formal languages and automata. Jones & Bartlett Learning, 2022.