Theory of Automaton

These notes have been prepared mainly from the following book:-

Theory of Computer Science : Automata, Languages and Computation
by K.L.P. Mishra, N. Chandrasekaran

Kindly refer above books for more details.

www.ankurgupta.net

Finite Automaton

Finite Automaton :-

The finite automaton has complete absense of memory outside its fixed central processor. It delivers no output at all, except an indication of whether or not the input is considered acceptable.

A deterministic finite state transducer is a device much like a DFA except that its purpose is not to accept strings or languages but to transform input strings into output strings.

For every NDFA, there exists a DFA which simulates the behaviour of NDFA. NDFA is a more general machine without being more powerful.

If an NDFA has n states, the corresponding finite automaton has 2^n states. However, we need not construct δ for all these 2^n states, but only for those states reachable from $[q_0]$.

Two automata are considered to be equivalent if they accept the same language, even though they may use the different methods to do so.

The class of languages accepted by finite automata is the same as the class of regular languages - those that can be described by regular expressions.

A language is regular if and only if it is accepted by a finite state automaton.

For any language L :-

$$\Phi L = L \Phi = \Phi$$

The class of languages accepted by finite automata is closed under union, concatenation, Kleen Star, Complementation, intersection and transpose operation.

Identities for Regular Expressions :-

(1) $\Phi + R = R$

(2) $\Phi R = R\Phi = \Phi$

(3) $\wedge R = R \wedge = R$

(4) $\wedge^* = \wedge$ and $\Phi^* = \wedge$

(5) $R + R = R$

(6) $R^* R^* = R^*$

(7) $R R^* = R^* R$

(8) $(R^*)^* = R^*$

(9) $\wedge + RR^* = R^* = \wedge + R^* R$

(10) $(PQ)^* P = P(QP)^*$

(11) $(P + Q)^* = (P^* Q^*)^* = (P^* + Q^*)^*$

(12) $(P + Q)R = PR + QR$

and $R(P + Q) = RP + RQ$

Arden's Theorem :-

Let P and Q be two regular expressions over Σ. If P does not contain $\wedge$, then the following equation in R :-

$$R = Q + RP$$

has a unique solution given by :-

$$R = QP^*$$

Transition System Containing $\wedge$-moves :-

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

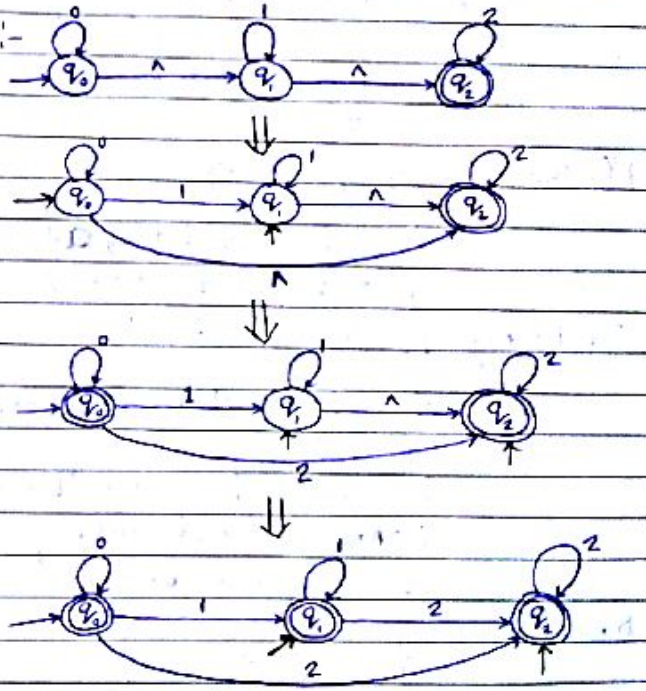
Step-1 :- Find all edges starting from v_2.

Step-2 :- Duplicate all these edges starting from v_1, without changing the edge labels.

Step-3 :- If v_1 is an initial state, make v_2 also an initial state.

Step-4 :- If v_2 is a final state, make v_1 as the final state.

Example :-

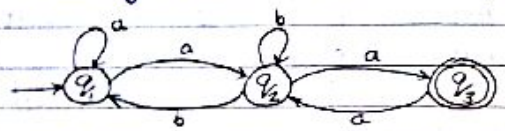


Finding the Regular Expression recognized by a transition system :-

The following assumptions are made regarding the transition system:-

(i) The transition graph does not have Λ-moves.
(ii) It has only one initial state, say v_1

Example :-



$$q_1 = q_1 a + q_2 b + \Lambda \quad \text{— (i)}$$

If will come only with initial state.

$$q_2 = q_1 a + q_2 b + q_3 a \quad \text{— (ii)}$$

$$q_3 = q_2 a \quad \text{— (iii)}$$

Placing value of q_3 in eq -(ii) :-

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

$$\Rightarrow \quad q_2 = q_1 a + q_2 (b + aa)$$

$$\Rightarrow \quad q_2 = q_1 a (b + aa)^* \quad \text{— (iv)}$$

Substituting q_2 from equation (iv) in eq (i) :-

$$q_1 = q_1 a + q_1 a (b+aa)^* b + \Lambda$$

$$\Rightarrow \quad q_1 = \Lambda + q_1 (a + a (b+aa)^* b)$$

$$\Rightarrow \quad q_1 = (a + a (b+aa)^* b)^*$$

Now :-

$$q_3 = q_2 a$$
$$= q_1 a (b+aa)^* a$$
$$= (a + a(b+aa)^* b)^* a (b+aa)^* a$$

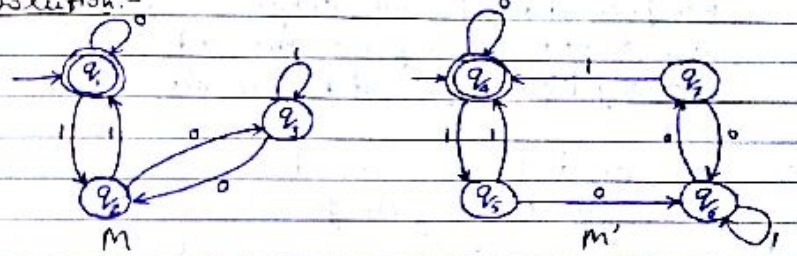
$\Rightarrow R_3$

Equivalence of two finite automata :-

Two finite automata over Σ are equivalent if they accept the same set of strings over Σ.

Example :- Consider the following two DFAs M and M' over {0,1}. Find out whether M and M' are equivalent.

Solution :-



(q_1, q_4')	(q_2, q_5')	(q_1, q_1')
(q_1, q_4)	(q_1, q_4)	(q_2, q_5)
(q_2, q_5)	(q_3, q_6)	(q_1, q_4)
(q_3, q_6)	(q_2, q_7)	(q_3, q_1)
(q_2, q_7)	(q_3, q_6)	(q_1, q_4)

The first element of the first column must be (q_1, q_4), i.e. initial state of M and M'.

As we do not get a pair (q, q'), where q is a final state and q' is a non-final state (or vice versa) at every row, we proceed until all the elements in the second and third column are also in first column.

Therefore M and M' are equivalent.

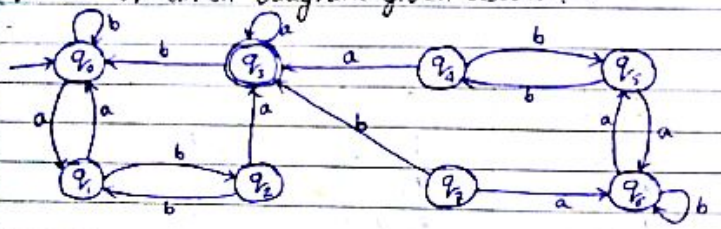
Minimization of Finite Automata :-

Definition-1 :- Two states q_1 and q_2 are equivalent if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states, or both of them are non final states for all $x \in \Sigma^*$

Definition-2 :- 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 strings x of length k or less. In particular, any two final states are 0-equivalent and any two non final states are also 0-equivalent.

Example :- Construct the minimum state automaton equivalent to the transition diagram given below :-

Solution :-



State/Σ	a	b
→ q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_5	q_1
(q_3)	q_3	q_0
q_4	q_3	q_5
q_5	q_4	q_4
q_6	q_5	q_6
q_7	q_6	q_3

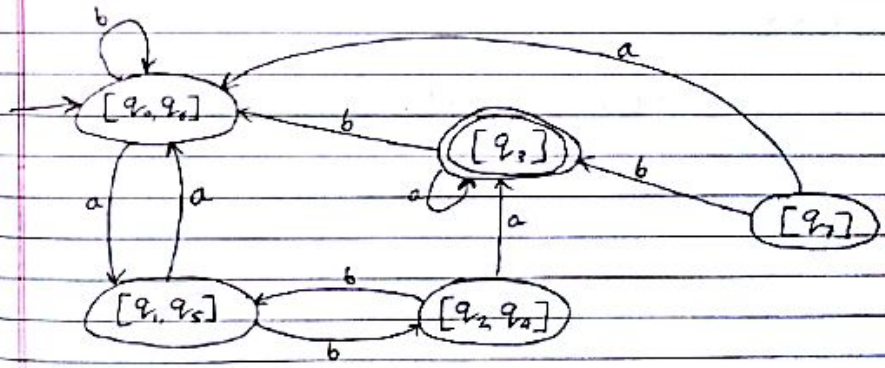
$\pi_0 = \{ [q_3], [q_0, q_1, q_2, q_4, q_5, q_6, q_7] \}$

$\pi_1 = \{ [q_3], [q_0, q_1, q_5, q_6], [q_2, q_4], [q_7] \}$

$\pi_2 = \{ [q_3], [q_0, q_6], [q_1, q_5], [q_2, q_4], [q_7] \}$

$\pi_3 = \{ [q_3], [q_0, q_6], [q_1, q_5], [q_2, q_4], [q_7] \}$

State/Σ	a	b
$[q_0, q_6]$	$[q_1, q_5]$	$[q_0, q_6]$
$[q_1, q_5]$	$[q_0, q_6]$	$[q_2, q_4]$
$[q_2, q_4]$	$[q_3]$	$[q_1, q_5]$
$[q_3]$	$[q_3]$	$[q_0, q_6]$
$[q_7]$	$[q_0, q_6]$	$[q_5]$



Two-way finite automaton :-

It is like a deterministic finite automaton except that the reading head can go backwards as well as forwards on the input tape.

It is a quintuple $(Q, \Sigma, \delta, q_0, F)$, where δ is a function from $Q \times \Sigma$ to $Q \times \{\leftarrow, \rightarrow\}$; $\leftarrow$ or $\rightarrow$ indicates the direction of head movement.

A configuration (p, u, v) indicates that the machine is in state p with the head on the first symbol of v and with u to the left of the head.

If $v = \Lambda$, configuration (p, u, Λ) means that it has completed its operation on u and ended up in state p.

Formal Languages

Language :- A language generated by a grammer G (denoted by $L(G)$) is defined as :-

$$\{w \in \Sigma^* \mid S \xrightarrow{*}_{G} w\}.$$

If w contains non-terminals, it is called a sentential form of G.

If w does not contain non-terminals, it is called as a sentence of G.

OR

$L(G)$ is the set of all terminal strings derived from the start symbol S.

Two grammers are equivalent if they produce the same language.

Chomsky Classification of Languages :-

Type-0 :- A type 0 grammer is any phrase structured grammer without any restriction.

A production without any restriction is called a type 0 production.

Type-1 :-

A production of the form $\phi A \psi \to \phi \alpha \psi$ is called a type-1 production if $\alpha \neq \Lambda$

In type-1 productions erasing of A is not permitted.

Example :-

(i) $A\underline{B} \to A\underline{b}\underline{B}c$ is a type-1 production. The left context is A, right context is Λ.

(ii) $A \to abA$ is a type-1 production. Here both the left and right context are Λ.

A grammer is called type-1 or context sensitive if all its productions are type-1 productions. The production $S \to \Lambda$ is also allowed in a type-1 grammer, but in this case S does not appear on the right hand side of any production.

The language generated by a type-1 grammer is called a type-1 or context-sensitive language.

In a context-sensitive grammer G, we allow $S \to \Lambda$ for including Λ in $L(G)$. Apart from $S \to \Lambda$, all the other productions do not decrease the length of the working string (if $\alpha \to \beta$, then $|\alpha| \leq |\beta|$).

$BC \to CB$ is not context sensitive.

Type-2 :-

A type-2 production is a production of the form $A \to \alpha$, where $A \in V_N$ and $\alpha \in (V_N \cup \Sigma)^*$. In other words, the L.H.S. has no left context and right context.

Example :- $S \to Aa$, $A \to a$, $B \to abc$, $A \to \Lambda$

A grammer is called a type-2 grammer if it contains only type-2 productions. It is also called a context-free grammer.

A language generated by a context-free grammer is called a type-2 language or context-~~sensitive~~ free language.

Type-3 :-

A production of the form $A \to a$ or $A \to aB$, where $A, B \in V_N$ and $a \in \Sigma$, is called a type-3 production.

A grammer is called a type-3 or regular grammer if all the productions are type-3 productions. A production $S \to \Lambda$ is allowed in type-3 grammer, but in this case S does not appear on the right hand side of any production.

A language is regular if it can be described by a regular expression.

Relation between languages :-

$L_{RL} \subseteq L_{C.F.L.} \subseteq L_{C.S.L.} \subseteq L_O$

Two grammers of different types may generate the same language

	$L_{R.L.}$	$L_{C.F.L.}$	$L_{C.S.L.}$	L_O
Union	✓	✓	✓	✓
Concatenation	✓	✓	✓	✓
Transpose	✓	✓	✓	✓
Intersection	✓	×	✓	✓
Complement	✓	×	✓	✓
Kleen Star	✓	✓		✓

$L_{C.F.L.}$ is not closed under intersection, but the intersection of a context-free language and regular language is context-free.

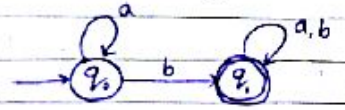
#

Construction of a Regular Grammer from a given DFA :-

Example :- Construct a regular grammer G generating the regular set represented by $P = a^*b(a+b)^*$

Solution :-

DFA corresponding to P is :-



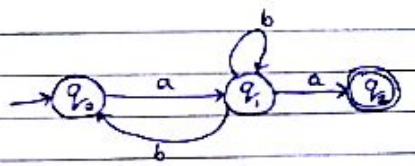
Regular grammer corresponding to P is :-

$A_0 \to aA_0,\ A_0 \to bA_1,\ A_0 \to b,$
$A_1 \to aA_1,\ A_1 \to bA_1,\ A_1 \to a,\ A_1 \to b$

Construction of a transition system accepting a regular grammer G :-

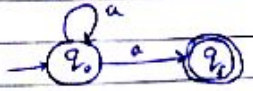
Example :- $G = (\{A_0, A_1\}, \{a, b\}, P, A_0)$
$P = \{A_0 \to aA_1,\ A_1 \to bA_1,\ A_1 \to a,\ A_1 \to bA_0\}$

Solution :-



Example :- $S \to aS \mid a$

Solution :-



Property of languages that are regular:-

As a string is scanned left to right, the amount of memory that is required in order to determine at the end whether or not the string is in the language must be bounded, fixed in advance and dependent on the language, not the particular input string.

Regular languages with an infinite number of strings are represented by automata with cycles and regular expressions involving the Kleen star. It means they must have some repetive structure.

Example:- $L(G) = \{a^n : n \geq 1 \text{ is a prime}\}$ is not regular.

$L = \{w \in \{a,b\}^* : w \text{ has an equal no of } a\text{'s and } b\text{'s}\}$ is not regular.

$L = a^*b^*$ is regular.

Every finite subset of a non-regular set is regular.

Context-Free Languages

Ambiguity in C.F.G.:-

A terminal string $w \in L(G)$ is ambiguous if there exists two or more derivation trees for w.

A context-free grammer G is ambiguous if there exists some $w \in L(G)$, which is ambiguous.

Construction of Reduced Grammers:-

For every CFG G there exists a reduced grammer G' which is equivalent to G.

We construct the reduced grammer in two steps:-

Step-1:-

We construct a grammer G_1 equivalent to the given grammer G so that every variable in G_1 derives some terminal string.

Step 2:-

We construct a grammer G' equivalent to G_1 so that every symbol in G' appears in some sentential form of G'.

G' is the required reduced grammer.

Example:- Find a reduced grammer equivalent to the grammer G, whose productions are:-

$S \to AB \mid CA,\ B \to BC \mid AB,\ A \to a,\ C \to aB \mid b$

Solution:-

Step-1:-

$W_1 = \{A, C\}$ as $A \to a$ and $C \to b$ are productions with a terminal string on R.H.S.

$W_2 = \{A, C\} \cup \{A_i \mid A_i \to \alpha \text{ with } \alpha \in (\Sigma \cup \{A, C\})^*\}$

$= \{A, C\} \cup \{S\}$

$= \{A, C, S\}$

$W_3 = \{A, C, S\} \cup \{A_i \mid A_i \to \alpha \text{ with } \alpha \in (\Sigma \cup \{A, C, S\})^*\}$

$= \{A, C, S\} \cup \phi$

$= \{A, C, S\} = W_2$

$V_N' = W_2 = \{S, A, C\}$

$P' = \{A_i \to \alpha \mid A_i, \alpha \in (V_N' \cup \Sigma)^*\}$

$= \{S \to CA,\ A \to a,\ C \to b\}$

Thus, $G_1 = (\{S, A, C\}, \{a, b\}, \{S \to CA, A \to a, C \to b\}, S)$

Step-2:-

$W_1 = \{S\}$

$W_2 = \{S\} \cup \{A, C\} = \{S, A, C\}$

$W_3 = \{S, A, C\} \cup \{a, b\}$

$= \{S, A, C, a, b\}$

$V_N'' = V_N' \cap W_3 = \{S, A, C\}$

$\Sigma'' = \Sigma' \cap W_3 = \{a, b\}$

$P'' = \{A_i \to \alpha \mid A_i \in W_3\} = P'$

Therefore:

$G' = (\{S, A, C\}, \{a, b\}, \{S \to CA, A \to a, C \to b\}, S)$

Elimination of Null Productions:-

A production of the form $A \to \Lambda$, where A is a variable, is called a null production.

A variable A in a context-free grammer is nullable if $A \xrightarrow{*} \Lambda$

If $G = (V_N, \Sigma, P, S)$ is a context-free grammer, then we can find a context-free grammer G_1 having no null productions such that:-

$L(G_1) = L(G) - \{\Lambda\}$.

Example:- Consider the grammer G whose productions are $S \to aS \mid AB,\ A \to \Lambda,\ B \to \Lambda,\ D \to b$. Construct a grammer G_1 without null productions generating $L(G) - \{\Lambda\}$.

Solution:-

Step-1:- Construction of the set W of all nullable variables:-

$W_1 = \{A_i \in V_N \mid A_i \to \Lambda \text{ is a production in } G\}$

$= \{A, B\}$

$W_2 = \{A, B\} \cup \{S\}$ as $S \to AB$ is a production with $AB \in W_1^*$

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

$W_3 = W_2 \cup \phi = W_2$

Step-2:- Construction of P':-

(i) $D \to b$ is included in P' as $b \notin W_2$

(ii) $S \to aS$ gives rise to $S \to aS$ and $S \to a$.

(iii) $S \to AB$ gives rise to $S \to AB$, $S \to A$ and $S \to B$

(Note:- We can not erase both the nullable variables A and B in $S \to AB$ as we will get $S \to \Lambda$ in that case).

The required grammer without null productions is:-

$G_1 = (\{S, A, B, D\}, \{a, b\}, P', S)$

where P' consists of:-

$D \to b,\ S \to aS,\ S \to AB,\ S \to a,\ S \to A,\ S \to B$

Elimination of Unit Productions :-

A unit production in a context-free grammer G is a production of the form $A \to B$, where A and B are variables in G.

If G is a context-free grammer, we can find a context-free grammer G_1 which has no null or unit productions such that $L(G_1) = L(G)$

Let A be any variable in V_N, then :-

Step-1 :- Construction of the set of variables derivable from A.

Define $W_i(A)$ recursively as follows :-

$$W_0(A) = \{A\}$$

$$W_{i+1}(A) = W_i(A) \cup \{B \in V_N \mid C \to B \text{ is in } P \text{ with } C \in W_i(A)\}$$

Step-2 :- Construction of A-productions in G_1.

The A-productions in G_1 are either :-

(a) The non-unit productions in G

or

(b) $A \to \alpha$ whenever $B \to \alpha$ is in G with $B \in W(A)$ and $\alpha \notin V_N$.

Now we define $G_1 = (V_N, \Sigma, P_1, S)$, where P_1 is constructed using step-2 for every $A \in V_N$.

Example :- Let G be $S \to AB$, $A \to a$, $B \to C \mid b$, $C \to D$, $D \to E$ and $E \to a$. Eliminate unit productions and get an equivalent grammer.

Solution :-

Step-1 :-

$W_0(S) = \{S\}$, $W_1(S) = W_0(S) \cup \phi = W_0(S)$.

Hence $W(S) = \{S\}$.

Similiarly,

$W(A) = \{A\}$, $W(E) = \{E\}$

$W_0(B) = \{B\}$, $W_1(B) = \{B\} \cup \{C\} = \{B, C\}$,

$W_2(B) = \{B, C\} \cup \{D\} = \{B, C, D\}$

$W_3(B) = \{B, C, D\} \cup \{E\} = \{B, C, D, E\}$

Therefore $W(B) = \{B, C, D, E\}$

Similiarly,

$W(C) = \{C, D, E\}$

$W(D) = \{D, E\}$

Step-2 :- The productions in G_1 are :-

$S \to AB$, $A \to a$, $E \to a$,

$B \to a \mid b$, $C \to a$, $D \to a$

Corollary :- If G is a context-free grammer, we can construct an equivalent grammer G' which is reduced and has no null productions or unit productions.

Proof :- We construct G' in the following way :-

Step-1 :- Eliminate null productions to get G_1.

Step-2 :- Eliminate unit productions in G_1 to get G_2.

Step-3 :- Construct a reduced grammer G' equivalent to G_2.

Chomsky Normal Form for Context-Free Grammers :-

A context-free grammer G is in Chomsky normal form if every production is of the form :-

$A \to a$, or $A \to BC$,

and $S \to \Lambda$ is in G if $\Lambda \in L(G)$.

When Λ is in $L(G)$, we assume that S does not appear on the R.H.S. of any production.

Reduction to Chomsky Normal Form :-

Step-1 :- Elimination of null productions and unit productions.

Step-2 :- Elimination of terminals on R.H.S :-

We define $G_1 = (V_N', \Sigma, P_1, S)$, where P_1 and V_N' are constructed as follows :-

(i) All the productions in P of the form $A \to a$ or $A \to BC$ are included in P_1. All the variables in V_N are included in V_N'.

(ii) Consider $A \to X_1 X_2 \dots X_n$ with some terminals on R.H.S. If X_j is a terminal, say a_i, add a new variable C_{a_i} to V_N' and $C_{a_i} \to a_i$ to P_1. In production $A \to X_1 X_2 \dots X_n$, every terminal on the R.H.S. is replaced by the corresponding new variable and the variables on the R.H.S. are retained. The resulting production is added to P_1. Thus we get :-

$G_1 = (V_N', \Sigma, P_1, S)$

Step-3 :- Restricting the number of variables on R.H.S.

We define $G_2 = (V_N'', \Sigma, P_2, S)$ as follows :-

(i) All productions in P_1 are added to P_2 if they are in the required form. All the variables in V_N' are added to V_N''.

(ii) Consider $A \to A_1 A_2 \dots A_m$, where $m \geq 3$. We introduce new productions $A \to A_1 C_1$, $C_1 \to A_2 C_2$, $\dots$ $C_{m-2} \to A_{m-1} A_m$, and new variables $C_1, C_2, \dots, C_{m-1}$. These are added to P' and V_N'', respectively.

Example :- Reduce the following grammer G to CNF.

G is $S \to aAD$, $A \to aB \mid bAB$, $B \to b$, $D \to d$.

Solution :-

Step-1 :- There are no null or unit productions.

Step-2 :- Let $G_1 = (V_N', \{a, b, d\}, P_1, S)$, where P_1 and V_N' are constructed as follows :-

(i) $B \to b$, $D \to d$ are included in P_1.

(ii) $S \to aAD$ gives rise to $S \to C_a AD$ and $C_a \to a$

$A \to aB$ gives rise to $A \to C_a B$.

$A \to bAB$ gives rise to $A \to C_b AB$ and $C_b \to b$

$V_N' = \{S, A, B, D, C_a, C_b\}$

Step-3 :- $P_1 = \{S \to C_a AD, A \to C_a B \mid C_b AB, B \to b, D \to d, C_a \to a, C_b \to b\}$

$A \to C_a B$, $B \to b$, $D \to d$, $C_a \to a$, $C_b \to b$ are added to P_2.

$S \to C_a AD$ is replaced by $S \to C_a C_1$ and $C_1 \to AD$.

$A \to C_b AB$ is replaced by $A \to C_b C_2$ and $C_2 \to AB$.

Now, $G_2 = (\{S, A, B, D, C_a, C_b, C_1, C_2\}, \{a, b, d\}, P_2, S)$,

where $P_2 = \{S \to C_a C_1, A \to C_a B \mid C_b C_2, C_1 \to AD, C_2 \to AB, B \to b, D \to d, C_a \to a, C_b \to b\}$

Greibach Normal Form :-

A context free grammer generating the set accepted by a pushdown automaton is in Greibach Normal Form.

A context-free grammer is in Greibach Normal Form if every production is of the form $A \to a\alpha$, where $\alpha \in V_N^*$ and $a \in \Sigma$ (α may be Λ), and $S \to \Lambda$ is in G if $\Lambda \in L(G)$. When $\Lambda \in L(G)$, we assume that S does not appear on the R.H.S. of any products

Lemma-1 :- Deleting a variable B appearing as the first symbol on the R.H.S. of some A-production provided no B-production has B as the first symbol on the R.H.S.

If $P = \{A \to B\gamma, B \to \beta_1 | \beta_2 | \dots | \beta_n\}$, then

$P_1 = \{A \to \beta_1\gamma | \beta_2\gamma | \dots | \beta_n\gamma\}$

Lemma-2 :- Eliminating Left Recursion.

Let $P = \{A \to A\alpha_1 | A\alpha_2 | \dots | A\alpha_r | \beta_1 | \beta_2 | \dots | \beta_s\}$, then

$P_1 = \{A \to \beta_1 Z | \beta_2 Z | \dots | \beta_s Z, Z \to \alpha_1 Z | \alpha_2 Z | \dots | \alpha_r Z\}$
$A \to \beta_1 | \beta_2 | \dots | \beta_s.\}$

Example :- Remove left-recursion from the following grammer :-
$A \to aBD | bDB | c, \quad A \to AB | AD$

Solution :-
$A \to aBD | bDB | c, \quad A \to aBDZ | bDBZ | cZ,$
$Z \to BZ | DZ | B | D.$

Example :- Remove Λ-productions from the following grammer :-
$S \to ASB | \Lambda, \quad A \to aAS | AB | a | \Lambda,$
$B \to SbS | Aa | b$

Solution :-

Step-1 :- Construction of set W of all nullable variables :-

$W_0 = \{S, \Lambda\}, \quad W_1 = \{S, A\} = W_0$

So $W = \{S, A\}$

Step-2 :- Construction of P :-

$S \to ASB | SB | AB | B$

~~$A \to aAS | aS | a | B | a$~~

$A \to aAS | aS | aA | AB | B | a$

$B \to SbS | Sb | bS | b | Aa | a$

#L = $\{a^n b^n c^n | n \geq 1\}$ is not context-free but context sensitive.

#L = $\{a^p | p$ is a prime$\}$ is not context-free.

Pushdown Automata :-

Each context-free language is accepted by some pushdown automaton.

If a language is accepted by a pushdown automaton, it is a context-free language.

Pushdown Automaton for $\{a^n b^n \mid n \geq 1\}$:-

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

(Z_0 : Initial Pushdown Symbol; Γ : Set of pushdown symbols.)

$$\delta \Rightarrow Q \times (\Sigma \cup \{\Lambda\}) \times \Gamma \rightarrow Q \times \Gamma^*$$

where :-

$Q = \{q_0, q_1, q_f\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, Z_0\}$,

$F = \{q_f\}$, and

δ is given by :-

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, a) = (q_1, \Lambda)$$
$$\delta(q_1, b, a) = (q_1, \Lambda)$$
$$\delta(q_1, \Lambda, Z_0) = (q_f, \Lambda)$$

Infinite context-free languages display periodicity of a somewhat subtler form than do regular languages.

→ If A is a pda accepting L by empty store, we can find a pda B, which accepts L by final state.

→ If A is a pda accepting L by final state, we can find a pda B, which accepts L by empty store.

Determinism and Parsing :-

A pushdown automaton is not of immediate use in parsing, because it's a nondeterministic device.

A pushdown automaton M is deterministic if for each configuration, there is at most one configuration that can succed it in a computation by M.

Deterministic Context-Free Languages are essentially those that are accepted by deterministic pushdown automaton.

There are some context-free languages that can not be accepted by deterministic pushdown automata.

The machine to accept the language $\{wcw^R : w \in \{a,b\}^*\}$ is deterministic, while the machine to accept $\{ww^R : w \in \{a,b\}^*\}$ is nondeterministic.

The class of deterministic context-free languages is closed under complement.

Nondeterminism is more powerful than determinism in the context of pushdown automata.

www.ankurgupta.net

Linear Bounded Automaton:-

A linear bounded automaton is a nondeterministic 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.

Turing Machines and Recursive & Recursive Enumerable Languages

Turing Machine:-

A turing machine consists of a finite control, a tape and a head that can be used for reading or writing on that tape.

The turing machines form a stable and maximal class of computational devices, in terms of the computations they can perform.

A turing machine M does not accept w if the machine either halts in a non-accepting state or does not halt.

A general purpose turing machine is usually called universal turing machine which is powerful enough to simulate the behaviour of any computer including any turing machine itself.

If we increase the no. of heads or make the tape two or three dimensional, the turing machine will atmost speedup the operation of the machine, but will not increase the computing power of doing something more than basic model.

The set of languages accepted by nondeterministic turing machine is same as the set of languages accepted by deterministic turing machine.

In linear bounded automaton, the length of the tape is restricted.

If L is a context-sensitive language, then L is accepted by a Linear Bounded Automaton (LBA). The converse is also true.

Recursive and Recursive Enumerable Languages:-

A turing machine M decides a language L if when started with input w, it always halts and does so in a halt state. If it accepts w, it halts in an accepting state and it does not accept w, it halts in non-accepting state).

A turing machine M semi-decides a language L, if when started with input w, it halts if and only if it accepts w, otherwise it does not halt.

A procedure for solving a problem is a finite sequence of instructions which can be mechanically carried out given any input.

An algorithm is a procedure that terminates after a finite number of steps for any input.

Recursive Language:-

A language L is recursive if there is a turing machine that decides it.

OR

A set X is recursive if we have an algorithm to determine whether a given element belongs to X or not.

Recursive Enumerable Language:-

A language L is recursive enumerable if and only if there is a turing machine M that semidecides it.

OR

A recursive enumerable set is a set X for which we have a procedure to determine whether a given element belongs to X or not.

Every recursive language is recursive enumerable, but the two classes are not the same.

Recursive Language $\subset$ Recursive Enumerable Lang.

The class of recursive languages is a proper subset of the class of recursively enumerable languages.

If L is a recursive language, then its complement $\bar{L}$ is also recursive.

A language is recursive if and only if both it and its complement are recursively enumerable.

The class of recursively enumerable languages is not closed under complement.

A language is generated by a grammer if and only if it is recursively enumerable.

Recursive Enumerable Language = Type-0 Language.

If L_1 is not recursive and there is a reduction from L_1 to L_2, then L_2 is also not recursive.

A context-sensitive language is recursive.

There exists a recursive set which is not a context-sensitive language.

Closure Properties of Recursive and Recursive Enumerable Languages:-

Recursive Languages are closed under union, concatenation, closure, intersection, complementation and set difference.

If L, L_1, and L_2 are recursive languages, then so are $L_1 \cup L_2$, $L_1 L_2$, L^*, $L_1 \cap L_2$, $L_1 - L_2$ and $\Sigma^* - L$.

Recursive Enumerable languages are closed under union concatenation, closure and intersection.

If L, L_1, and L_2 are recursive enumerable, then so are $L_1 \cup L_2$, $L_1 L_2$, L^* and $L_1 \cap L_2$.

If L_R is a recursive language and L_{RE} is a recursively enumerable language, then:-

(i) $L_R \cup L_{RE}$, $L_R \cap L_{RE}$, $L_R L_{RE}$, $L_{RE} L_R$, are recursive enumerable.

(ii) $L_{RE} - L_R$ is recursive enumerable.

(iii) $L_R - L_{RE}$ is not recursive enumerable.

Undecidablity

Halting Problem:-

Halting Machine takes as input an encoding of a turing machine e(M) and an encoding of an input string e(w), and returns 'yes' if M halts on w, and "no" if M does not halt on w.

The halting problem is undecidable:-

→ There exists no turing machine that decides it.

→ There is no turing machine that halts on all inputs, and always say "yes" if M halt on w, and always say "no" if M does not halt on w.

The following problems about Turing machines are undecidable:-

(i) Given a turing machine M, does M halt on the empty tape?

(ii) Given two turing machines M_1 and M_2, do they halt on same input strings?

(iii) Given a turing machine M, is the language that M semidecides regular? is it context free? is it recursive?

Reduction:-

To solve a problem B is undecidable:-

(i) Start with an instance of a known undecidable problem.

(ii) Create an instance of problem B, such that the answer to the instance of problem B gives the answer to the undecidable problem.

(iii) If we could solve problem B, we could solve halting problem.

Thus, problem B must be undecidable.

If we can reduce problem A to problem B, and problem A is undecidable, then:-
problem B must also be undecidable.

If we can reduce problem B to problem A, and problem A is undecidable, then:-
It tells nothing about problem B.

Unsolvable Problems about Grammers:-
Each of the following problems is undecidable:-

(1) Given a grammer G and a string w,
Is $w \in L(G)$?
(2) Given a grammer G,
Is $\wedge \in L(G)$?
(3) Given grammers G_1 and G_2,
Is $L(G_1) = L(G_2)$?
(4) Given grammer G,
Is $L(G) = \phi$

The following questions about Context-Free Grammers are decidable:-
(1) Given a C.F.G. G and string w,
Is $w \in L(G)$
(2) Given a CFG G,
Is $\wedge \in L(G)$

However there are some problems about CFGs that are undecidable:-
(1) Given any CFG G, is $L(G) = \Sigma^*$?

(2) Given any two CFGs G_1 and G_2, is $L(G_1) = L(G_2)$?

(3) Given two PDA M_1 and M_2, is $L(M_1) = L(M_2)$?

(4) Given a PDA M, find an equivalent PDA with the smallest possible number of states.

Tiling problem is undecidable.

Ambiguity problem for CFGs is undecidable.

Some Important Results

(1) Regular Expression for all strings over {0,1} that:-

(i) have an even no. of 1s = $0^*(10^*10^*)^*$

(ii) have no occurrences of 00 = $1^*(011^+)^*(0+1^*)$

(iii) have exactly one occurance of 00
= $1^*(011)^*00(11^*0)^*1^*$

(iv) do not contain 01 = 1^*0^*

(2) $L = \{a^n b^n c^n \mid n \geq 1\}$ is not context-free, but context sensitive.

(3) $L = \{a^p \mid p \text{ is a prime}\}$ is not a context-free language.

(4) $L = \{0^n 1^n \mid n \geq 1\}$ is not regular, but context-free.

(5) $L = \{ww \mid w \in \{a,b\}^*\}$ is not context-free.

(6) $L = \{a^{n^2} \mid n \geq 0\}$ is not context-free.

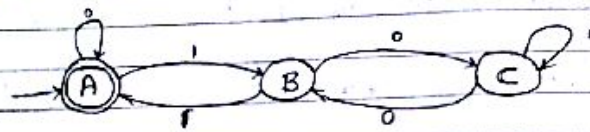
(7) $L = \{a^{2n} \mid n \geq 1\}$ is regular.

(8) $L = \{babaabaaa \ldots ba^{n-1}ba^nb \mid n \geq 1\}$ is not context-free

(9) $L = \{wxw^T \mid x, w \in \{a,b\}^+\}$ is regular.

(10) $L = \{ww^T \mid w \in \{a,b\}^*\}$ is context-free.

(11) DFA that accepts all binary strings whose decimal value is divisible by 3.



NP Completeness

Tractable Problems :-

→ The set of all the problems that can be solved within polynomial amount of time using deterministic machine.

→ It is of type $O(n^k)$.

Intractable Problems :-

→ The set of all problems that can't be solved within polynomial amount of time using deterministic machine.

→ It is of type $O(k^n)$.

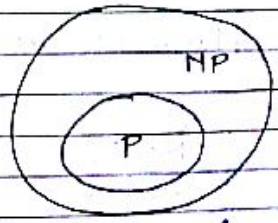
P-Class Problems :-

The set of problems, that can be solved in polynomial amount of time by deterministic machines, is called the P-class or polynomial problems.

NP-Class Problems :-

The set of problems that can be solved in polynomial time using non-deterministic machines.

NP = Non-deterministic Polynomial Time.
NP ≠ Non-polynomial.



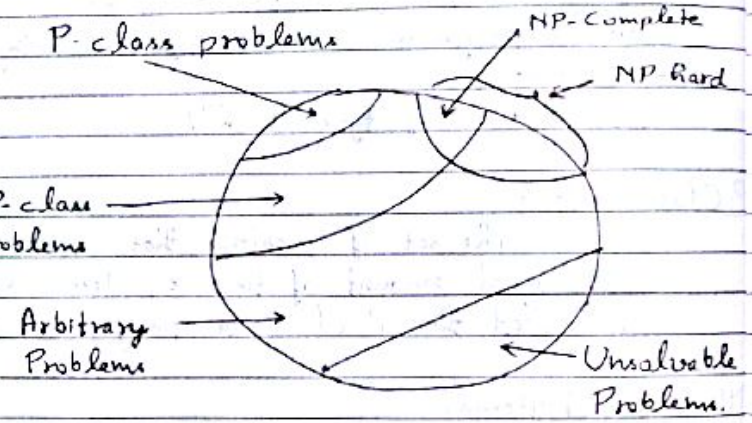
$P \subset NP$

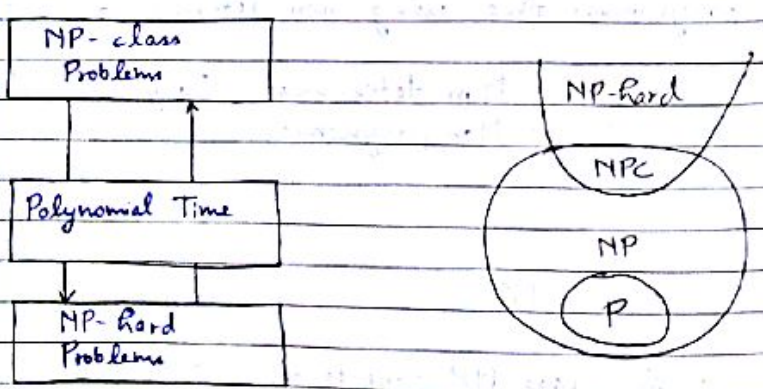
→ The class NP consists of those problems that are verifiable in polynomial time.

NP-Hard Problems:-

→ These problems are at least as hard as the hardest problems in NP, but not necessarily in NP.

→ The problems, to which all NP-class problems are reducible in polynomial time, are known as NP-hard class of problems.





NP-Complete Problems:-

→ NP-Complete problems are those NP-hard problems which are within NP.

→ These are set of the hardest problems in NP, i.e. in real sense these are the ones that most likely not to be in P.

→ NP-Complete problems are solvable using heuristic.

→ It is not known whether NP problems are tractable or intractable.

→ If a problem in a class of NP-Complete can be solved in polynomial time, then all the problems in that class can be solved in a polynomial time using the same algorithm.

→ If a deterministic polynomial time algorithm is found for an NP-Complete problem, then every problem in NP-class can be solved in deterministic polynomial time.

The class of all polynomially decidable languages, denoted by P is closed under union, intersection, concatenation, complement and Kleene Star.

Examples of NP-Complete Problems:-

(1) Bounded Tiling

(2) 3-Satisfiability / Satisfiability (SAT)

(3) MAX-SAT

(4) Hamiltonian Cycle

(5) Traveling Salesman Problem

(6.) Exact Cover and Node Cover

(7) Independent Set

(8) 0-1 Knapsack

(9) Bin Packing

(10) Printed Circuit Board (PCB)

(11) CLIQUE