GeeksforGeeks

GATE फर्स्ट

CSE

TOC

SHORT NOTES

ENROLL NOW

TO EXCEL IN GATE
AND ACHIEVE YOUR DREAM IIT OR PSU!

ENROLL NOW

1. Basics of TOC

1.1. Symbol

- The smallest indivisible entity; cannot be broken into smaller parts.
- Every unique atom of information.

1.2. Alphabet (Σ)

- A finite set of symbols.
- Examples:
  - English letters: {a, b, …, z}
  - Binary: {0, 1}

1.3. String

- A finite sequence of symbols over Σ.
- Denoted w ∈ Σ*; empty string is ε (|ε| = 0).
- Examples over Σ = {a,b}: ε, a, b, ab, ba, …

1.4. Operations on Strings

- Length: |w| = number of symbols in w.
- Reversal: w^R = symbols of w in reverse order.
- Concatenation: $w_1.w_2$ = sequence w_1 followed by w_2.
- Prefix/suffix: any leading/trailing substring of w (including ε and w).
- Substring: any contiguous sequence within w.

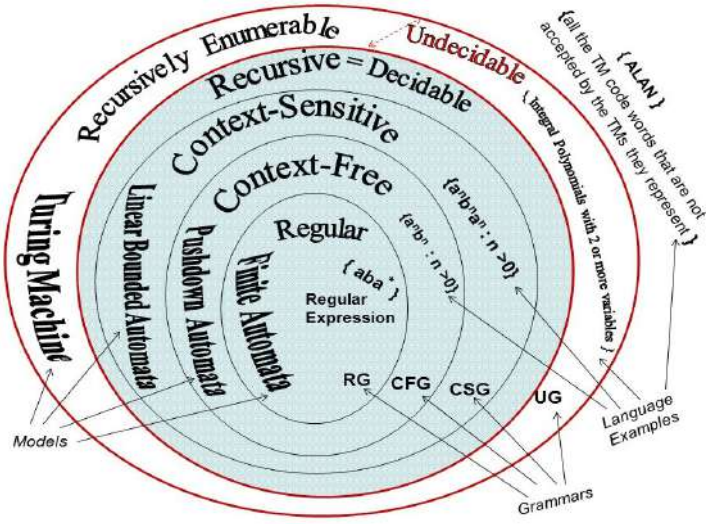
1.5. Language

- A set of strings over Σ (i.e. L ⊆ Σ*).
- Examples:
  - Universal: Σ* = all strings over Σ.
  - L = ab* = {a, ab, abb, …}.

Note:

1. Kleene star, $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$
2. Positive closure, $L^+ = L^1 \cup L^2 \cup \ldots$

1.6. Chomsky Hierarchy



Chomsky hierarchy	Grammars	Languages	Minimal automaton
Type-0	(unrestricted)	Recursively enumerable	Turing machine
	(unrestricted)	Recursive	Decider
Type-1	Context-sensitive	Context-sensitive	Linear-bounded
Type-2	Context-free	Context-free	Pushdown
Type-3	Regular	Regular	Finite

1.7. Grammar

- Defined as G = (V,T,S,P) where,
  V: variables/nonterminals; T: terminals; S ∈ V: start symbol; P: production rules.
- Language generated L(G) = all terminal strings derivable from S.

1.7.1. Types of grammars

1.7.1.1. Type 0 (Unrestricted Grammar)

Form of productions: α → β, where α ∈ $(V \cup T)^+$ & β ∈ $(V \cup T)^*$ (no restrictions on length or symbols except α ≠ ε).

1.7.1.2. Type 1 (Context-Sensitive Grammar)

Form of productions: αAβ → αγβ, where A ∈ V, (α, β) ∈ $(V \cup T)^*$, γ ∈ $(V \cup T)^+$, and |αγβ| ≥ |αAβ| (i.e. productions do not shrink the string).

Note: ε is only allowed if S → ε and S does not appear on the right side.

1.7.1.3. Type 2 (Context-Free Grammar)

Form of productions: A → γ, where A ∈ V and γ ∈ (V ∪ T)*.

Normal form: Chomsky Normal Form (CNF), Greibach Normal Form (GNF).

1.7.1.4. Type 3 (Regular Grammar)

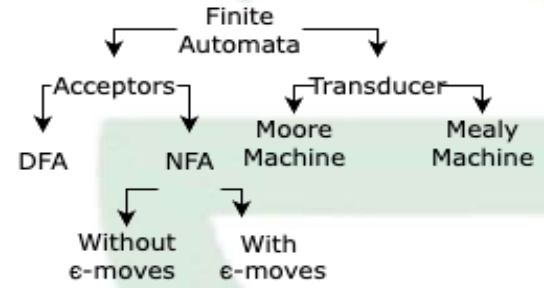
Form of productions

(Right-linear): A → aB or A → a, where A,B ∈ V and a ∈ T*.

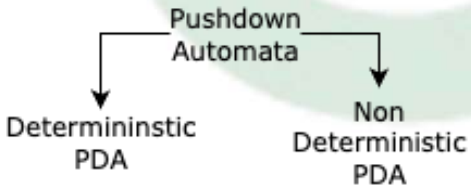
(Left-linear): A → Ba or A → a, where A,B ∈ V and a ∈ T*.

1.8. Types of Automata

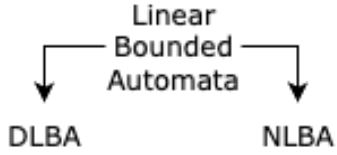
1.8.1.



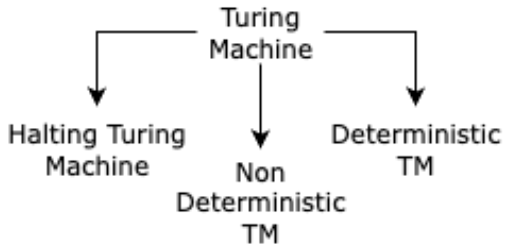
1.8.2.



1.8.3.



1.8.4.

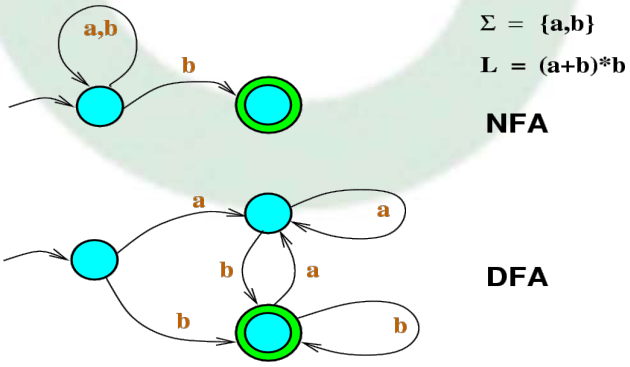


1.8.5. Deterministic Automata

- At any point, given the current state and input (or tape symbol/stack symbol), there is at most one possible move.
- Computation is a single, linear sequence of configurations.
- Easier to simulate directly on hardware.

1.8.6. Non Deterministic Automata

- At some points, the machine may have multiple choices (or none) for the next move.
- Conceptually "branches" into many computation paths in parallel.
- Accepts if any branch reaches an accepting configuration.



Note:

1. An automaton whose output response is limited to a simple "yes" or "no" is called an accepter.
2. Automaton, capable of producing strings of symbols as output, is called a transducer.

Properties:

1. $(L^*)' \neq (L')^*$
2. $(L_1L_2)^R = L_2^R L_1^R$ ∀ languages L_1 & L_2

3. $(L^*)^* = L^*$
4. $(L_1 \cup L_2)^R = L_1^R \cup L_2^R \forall$ languages L_1 & L_2
5. $(L^R)^* = (L^*)^R \forall$ languages L
6. Let L be any language on non-empty alphabet, both L and L′ can't be finite as $L \cup L' = \Sigma^*$

7. Relation b/w automata on basis of power (i.e. can represent more no. of languages) is FA < DPDA < NPDA < LBA < HTM < TM

2. Finite Automata & Regular Languages

2.1. DFA

Defined by M = (Q, Σ, δ, q0, F), where **Q** is a finite set of states,
Σ is a finite set of symbols called the input alphabet,
δ :Q × Σ → Q is called transition function,
$\mathbf{q_0} \in Q$ is the initial state,
F ⊆ Q is a set of final states.

Note:

1. If a DFA accepts any string of length (n-1), where n is no. of states (including dead state), then L(DFA) will be infinite as a loop will surely exist. (refer Fig. 1)

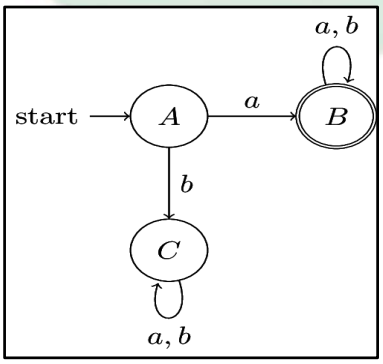


Fig. 1

∴ For finite L(DFA), max (n-2) length string is possible for n states DFA (including dead state). (refer Fig. 2)

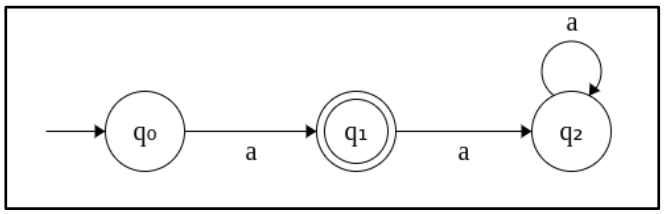


Fig. 2

2. L(M′) = (L(M))′ i.e. a complimentary machine accepts complimentary language (in case of DFA).
3. We get same power as TM when,
   - FA with queue
   - FA with read/write head & 2 way head movement
   - FA + 2 stack of infinite memory
   - FA + 2 counter

& power of FA is same when,

- FA + read/write head
- FA + 2 way head movement

2.2. NFA

Defined by M = (Q, Σ, δ, q0, F), where
Q, Σ, q0, F are same as for DFA, but
$\delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow 2^Q$ for NFA

Note:

1. In NFA, the range of δ is 2^Q, so that its value is a subset of Q. This subset defines the set of all possible states that can be reached by the transition like $\delta(q_1,a) = (q_0, q_2)$
   - Either q0 or q2 could be the next state of the NFA.
   - NFA can make a transition without consuming an input symbol by using null moves.
2. If we complement states in NFA, the new NFA may or may not recognize L′ (i.e. complement of language L).
3. For a NFA with n states, there is an equivalent minimum DFA with at most 2^n states.

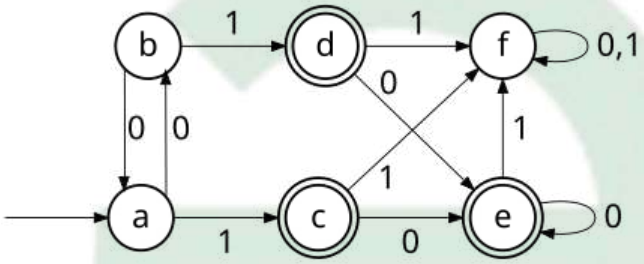
2.3. Minimization of DFA

For a given language, there are many DFA that accept it.

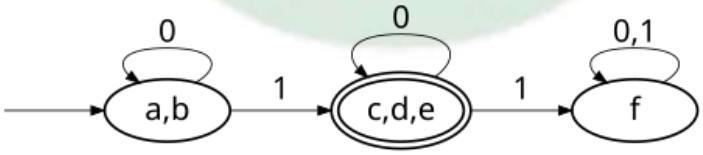
Algorithm:

1. First remove unreachable states from the initial state (Q_0).
2. Then, merge equivalent states.
3. If there are more than one final states then check for final states also if they are equivalent or not.

Note: Two states p, q of a DFA are equivalent if for every input string w ∈ Σ*, the runs from p and q on w either both end in accepting states or both end in non-accepting states.



Example DFA (If in state c, it exhibits the same behavior for every input string as in state d, or in state e. Similarly, states a and b are nondistinguishable. The DFA has no unreachable states.)



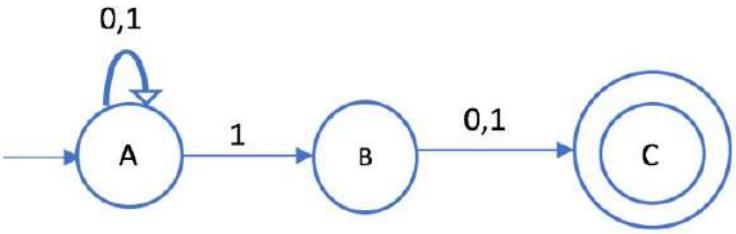
Equivalent minimal DFA (Nondistinguishable states have been merged into a single one.)

2.4. NFA to DFA conversion

NFA

State Transition Diagram for NFA

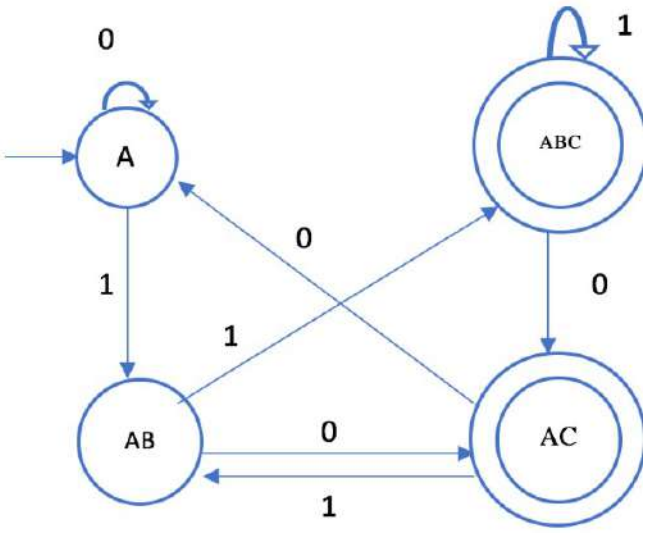
	0	1
→ A	{A}	{A,B}
B	{C}	{C}
* C	∅	∅



DFA

State Transition Diagram for DFA

	0	1
→ A	A	AB
AB	AC	ABC
* AC	A	AB
*ABC	AC	ABC



2.5. Moore & Mealy Machines

GeeksforGeeks

GATE CSE BATCH

KEY HIGHLIGHTS:

- 300+ HOURS OF RECORDED CONTENT
- 900+ HOURS OF LIVE CONTENT
- SKILL ASSESSMENT CONTESTS
- 6 MONTHS OF 24/7 ONE-ON-ONE AI DOUBT ASSISTANCE
- SUPPORTING NOTES/DOCUMENTATION AND DPPS FOR EVERY LECTURE

COURSE COVERAGE:

- ENGINEERING MATHEMATICS
- GENERAL APTITUDE
- DISCRETE MATHEMATICS
- DIGITAL LOGIC
- COMPUTER ORGANIZATION AND ARCHITECTURE
- C PROGRAMMING
- DATA STRUCTURES
- ALGORITHMS
- THEORY OF COMPUTATION
- COMPILER DESIGN
- OPERATING SYSTEM
- DATABASE MANAGEMENT SYSTEM
- COMPUTER NETWORKS

LEARNING BENEFIT:

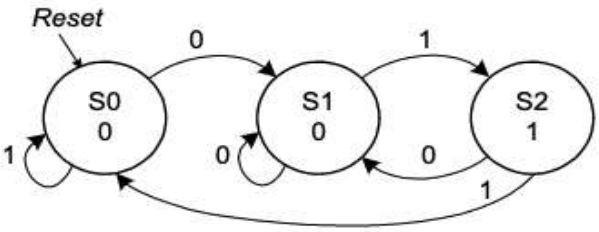
- GUIDANCE FROM EXPERT MENTORS
- COMPREHENSIVE GATE SYLLABUS COVERAGE
- EXCLUSIVE ACCESS TO E-STUDY MATERIALS
- ONLINE DOUBT-SOLVING WITH AI
- QUIZZES, DPPS AND PREVIOUS YEAR QUESTIONS SOLUTIONS

ENROLL NOW

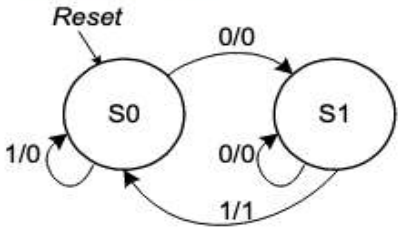
TO EXCEL IN GATE
AND ACHIEVE YOUR DREAM IIT OR PSU!

ENROLL NOW

Moore FSM

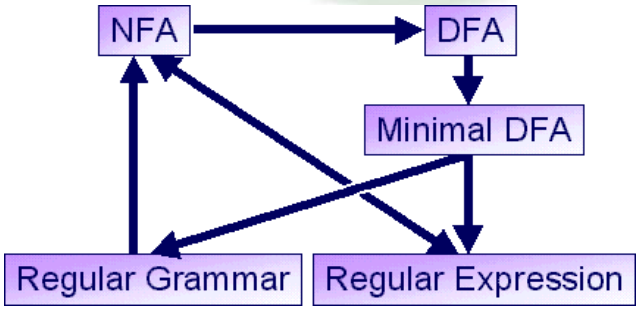


Mealy FSM



Feature	Mealy Machine	Moore Machine
Definition	6-tuple $(Q, \Sigma, \Lambda, \delta, \omega, q_0)$	6-tuple $(Q, \Sigma, \Lambda, \delta, \lambda, q_0)$
Output function	$\omega: Q \times \Sigma \to \Lambda$	$\lambda: Q \to \Lambda$
Output depends on	Current state and input	Current state only
Output timing	Immediate on transition	After entering state
State count	Often fewer	May require more
Latency	Lower (no delay)	One-step delay
Transition function	$\delta: Q \times \Sigma \to Q$	$\delta: Q \times \Sigma \to Q$
Equivalence	Convertible to Moore	Convertible to Mealy

Note:



2.6. Regular expressions

Let Σ be a given alphabet. Then,

1. Ø, λ and a ∈ Σ are all regular expressions.
2. If r1 and r2 are regular expressions, so are r1+ r2, r1.r2, r1*, and (r1).
3. Ø is a regular expression denoting the empty set { }
4. λ is a regular expression denoting {λ}
5. For every a ∈ Σ, **a** is regular expression denoting {a}.

Regular Expressions	Regular Set
(0 + 10*)	L = { 0, 1, 10, 100, 1000, 10000, ... }
(0*10*)	L = {1, 01, 10, 010, 0010, ...}
(0 + ε)(1 + ε)	L = {ε, 0, 1, 01}
(a+b)*	Set of strings of a's and b's of any length including the null string. So L = { ε, a, b, aa , ab , bb , ba, aaa.....}
(a+b)*abb	Set of strings of a's and b's ending with the string abb. So L = {abb, aabb, babb, aaabb, ababb, ...}

Theorem 1.1.1 : For any regular expressions α, β and γ,

(i) $\alpha + \beta \equiv \beta + \alpha$,
(ii) $(\alpha + \beta) + \gamma \equiv \alpha + (\beta + \gamma)$,
(iii) $\emptyset + \alpha \equiv \alpha + \emptyset \equiv \alpha$,
(iv) $(\alpha\,\beta)\,\gamma \equiv \alpha\,(\beta\,\gamma)$,
(v) $\alpha(\beta + \gamma) \equiv \alpha\beta + \alpha\gamma$,
(vi) $(\alpha + \beta)\,\gamma \equiv \alpha\gamma + \beta\gamma$,
(vii) $\varepsilon\,\alpha \equiv \alpha\,\varepsilon \equiv \alpha$,
(viii) $\emptyset\,\alpha \equiv \alpha\,\emptyset \equiv \emptyset$,
(ix) $\emptyset^* \equiv \varepsilon$,
(x) $(\alpha + \varepsilon)^* \equiv \alpha^*$,
(xi) $\alpha(\beta\,\alpha)^* \equiv (\alpha\,\beta)^*\,\alpha$,
(xii) $(\alpha^*)^* \equiv \alpha^*$,
(xiii) $(\alpha^*\,\beta^*)^* \equiv (\alpha + \beta)^*$,
(xiv) $(\alpha\,\beta^*)^* \equiv \varepsilon + \alpha(\alpha + \beta)^*$.

2.7. Identify Regular Languages

- If the set of strings in L is finite, L is regular since all finite languages are regular.
- If the set of strings in L is infinite, check if we can draw an NFA for recognizing L. If so, L is regular

Some twisted examples of Regular Grammar:

1. ∞ language can be regular too like
   $L = \{a^{\lfloor\sqrt{n}\rfloor}, n \geq 0\}$, $L = \{a^{\lceil\sqrt{n}\rceil}, n \geq 0\}$
2. $L = \{a^n b^m \mid n \geq m \text{ or } n \leq m\} = \{a^n b^m \mid n,m \geq 0\} = a^* b^*$
3. Binary encoding of 2^k (k>0) as it is same as 10^k or 10^+

4. L = {$a^n b^m$ | n*m **β** constant}, where β is = or ≤ or ≥ or < or > or ≠
5. L = {$a^n b^m$ | n+m **β** constant}, where β is = or ≤ or ≥ or < or > or ≠

2.8. Reversal of DFA

L = Language starts with a.

L^r = Language ends with a.

In the given DFA,

1. Make the final states as initial state.
2. Make the initial state as final state.
3. Reverse all the transition from q0 → q1 to q1→q0 for any two states in DFA.
4. Self-loops are unchanged.
5. Reversal of a DFA may result in a DFA or an NFA.
6. If there are multiple initial states in the resulting DFA (or NFA) , take an initial state and add all initial states with epsilon transitions.

2.9. Complement of DFA

In a given DFA,

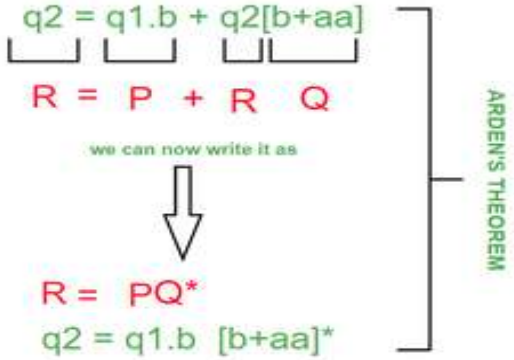
1. Convert final states into non-final states, and
2. Convert non-final states into final states.
3. Don't change initial state

This DFA will accept complement of the language accepted by the original DFA.

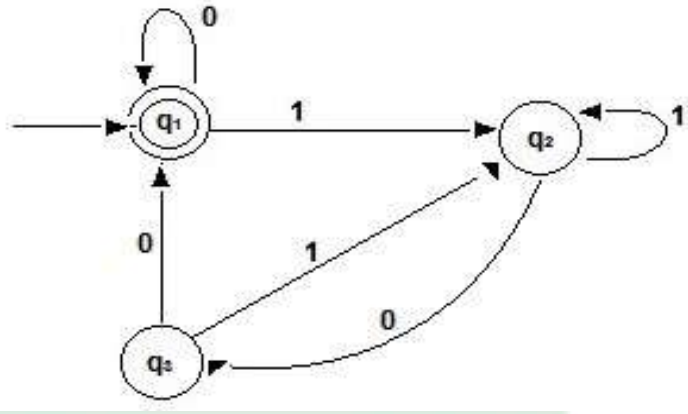
2.10. Arden's Theorem

Let P and Q be two regular expressions over alphabet Σ. If P does not contain null string, then R = Q + RP

has a unique solution that is R = QP*



Eg.



Let us form the equations

q1 = q10 + q30 + ϵ

q2 = q11 + q21 + q31

q3 = q20

Solving the equations,

q2 = q11 + q21 + (q20)1 = q11 + q2(1 + 01)

q2 = q11 (1 + 01)* (by Arden's theorem)

So, q1 = q10 + q30 + ϵ

q1= q10 + q11(1 + 01*)00 + ϵ

= q1(0 + 1(1 + 01)*00) + ϵ

= ϵ (0 + 1(1 + 01)*00)*

q1=(0 + 1(1 + 01)*00)*

So the regular expression for the given automata is

= (0 + 1(1 + 01)*00)*

2.11. Pumping Lemma for Regular Languages

Let L be a regular language. Then there exists an integer p≥1 depending only on L such that every

string w in L of length at least p (p is called the "pumping length") can be written as
w =xyz (i.e., w can be divided into three substrings), satisfying the following conditions:

1. $|y| \geq 1$
2. $|xy| \leq p$
3. For every $i \in N$, $xy^iz \in L$

Eg. The language $L = \{a^nb^n, n \geq 0\}$ over the alphabet $\Sigma = \{a,b\}$ can be shown to be non-regular as follows:

1. **Assume** L is regular. Then by the pumping lemma, $\exists$ p such that every $s \in L$ with $|s| \geq p$ can be pumped.
2. **Choose** $s = a^p b^p$ (so $|s| = 2p \geq p$).
3. **Decompose** $s = xyz$ with $|xy| \leq p$ and $|y| > 0$.
   - Since $|xy| \leq p$, both x and y consist only of a's.
   - Thus $y = a^k$ for some k where $1 \leq k \leq p$.
4. **Pump Down (i = 0)**:
   - $x y^0 z = x z = a^{(p-k)} b^p$
5. **Contradiction**:
   - xz has (p–k) a's followed by p b's, so #a's ≠ #b's.
   - Therefore $xz \notin L$, contradicting the lemma's requirement that $x y^0 z \in L$.
6. **Conclusion**
   - Our assumption that L is regular must be false.
   - Hence, $L = \{ a^n b^n \}$ is **not** a regular language.

Note: Pumping lemma gives a necessary but not sufficient condition for a language to be regular i.e. if pumping lemma satisfies, language may or may not be regular.

2.12. Myhill Nerode Theorem

Note: It is necessary & sufficient condition for a language to be regular.

It states that L is regular iff $\equiv_L$ (Equivalence Relation) has finite no. of equivalence classes.
OR L has finite no. of equivalence classes.
No. of equivalence classes = No. of States in Minimal DFA

For better understanding:

What is Equivalence Relation?: Let x, y be strings and L be a language. We say that x and y are indistinguishable by L if there for every z the following holds: $xz \in L$ iff $yz \in L$. We write $x \equiv_L y$.

Eg. $L = \{ w \in \Sigma^* \mid w \text{ ends with 'a'} \}$,
$\Sigma = \{a, b\}$
Equivalence Classes:

1. Class [ε] (strings not ending in 'a'):
   a. Examples: ε, b, bb, ab, ba, …
   b. For any z, xz ends in 'a' exactly when z ends in 'a'.
2. Class [a] (strings ending in 'a'):
   a. Examples: a, aa, ba, aba, …
   b. For any z, xz ends in 'a' exactly when the last symbol of z is 'a'.

No other distinctions exist, so there are exactly 2 equivalence classes. (Minimal DFA has two states corresponding to [ε] (non-accepting) and [a] (accepting))

3. PDA & CFLs

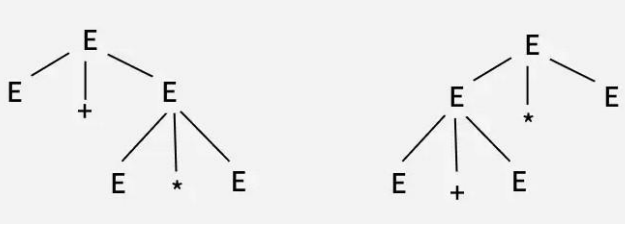
3.1. CFL

A language L over alphabet Σ is context-free if ∃ a context-free grammar (CFG) G such that L = L(G).

3.2. Parse Trees & Ambiguity

- Parse Tree: tree representation of a derivation.
- Ambiguous Grammar: ∃ w ∈ L(G) with >1 distinct parse trees.

Eg. For E → E+E | E*E | id



Note: Inherently ambiguous
If every grammar that generates L is ambiguous, then the language is called inherently ambiguous.

3.3. Normal Forms

3.3.1. Chomsky Normal Form (CNF) Productions $A \rightarrow BC$ or $A \rightarrow a$ ($S \rightarrow \varepsilon$ if $\varepsilon \in L$ is exception).

3.3.2. Greibach Normal Form (GNF) Productions $A \rightarrow a\alpha$, where $a \in T$ and $\alpha \in V^*$.

3.4. CFG to CNF conversion

1. If the start symbol S occurs on some right side, we create a new start variable S' and add a new production $S' \rightarrow S$.
2. Then, we eliminate all λ-productions of the form $A \rightarrow \lambda$
3. We also eliminate all unit rules of the form $A \rightarrow B$.
4. If there is production $S \rightarrow S$ remove it as its trivial production.

Note:

- We need (2n-1) productions to generate n length string.
- Any CFG with $\lambda \notin L(G)$ has an equivalent grammar in CNF & GNF.

3.5. Membership Algorithm for CFG

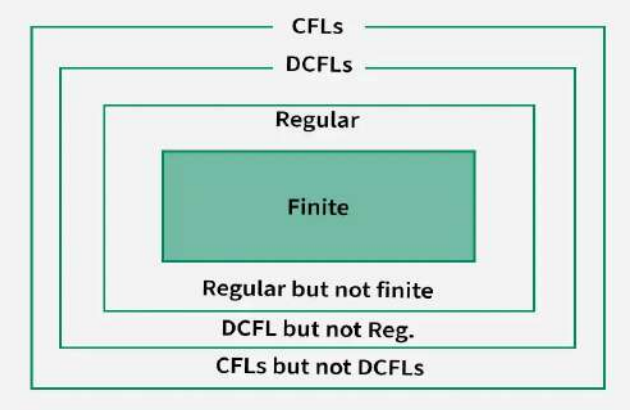
- CYK Membership and parsing algorithms for CFG exist that require approximately $|w|^3$ steps to parse a string w.
- CYK algorithm works only if the grammar is in CNF and succeeds by breaking one problem into a sequence of smaller ones.

Note:

1. Some CFLs
   a. $L = \{a^n b^m \mid n\text{-}m \ \beta \text{ constant}\}$, where β is = or ≤ or ≥ or < or > or ≠
   b. $L = \{a^n b^m \mid n/m \ \beta \text{ constant}\}$, where β is = or ≤ or ≥ or < or > or ≠
   c. $L = \{a^m b^n c^p d^q \mid m+n = p+q\}$
   d. $L = \{a^m b^n c^p d^q \mid m+p = n+q\}$
   e. $L = \{a^m b^n c^p d^q \mid m+q = n+p\}$

   A twisted example: Consider,
   $L = \{w_1 w_1^R w_2 w_2^R \mid w_1, w_2 \in (a,b)^*\}$
   Let $x = w_1 w_1^R$ & $y = w_2 w_2^R$
   x, y are DCFL & DCFL ∪ DCFL = CFL

2. Comparisons between 'a' & 'b' (ordered, unordered) → CFL
3. Palindromes → CFL
4. If we can recognize L using a PDA then L is CFL.
5. If the moves of PDA are all deterministic, then L is a DCFL.



GeeksforGeeks

GATE CSE BATCH

KEY HIGHLIGHTS:

- 300+ HOURS OF RECORDED CONTENT
- 900+ HOURS OF LIVE CONTENT
- SKILL ASSESSMENT CONTESTS
- 6 MONTHS OF 24/7 ONE-ON-ONE AI DOUBT ASSISTANCE
- SUPPORTING NOTES/DOCUMENTATION AND DPPS FOR EVERY LECTURE

COURSE COVERAGE:

- ENGINEERING MATHEMATICS
- GENERAL APTITUDE
- DISCRETE MATHEMATICS
- DIGITAL LOGIC
- COMPUTER ORGANIZATION AND ARCHITECTURE
- C PROGRAMMING
- DATA STRUCTURES
- ALGORITHMS
- THEORY OF COMPUTATION
- COMPILER DESIGN
- OPERATING SYSTEM
- DATABASE MANAGEMENT SYSTEM
- COMPUTER NETWORKS

LEARNING BENEFIT:

- GUIDANCE FROM EXPERT MENTORS
- COMPREHENSIVE GATE SYLLABUS COVERAGE
- EXCLUSIVE ACCESS TO E-STUDY MATERIALS
- ONLINE DOUBT-SOLVING WITH AI
- QUIZZES, DPPS AND PREVIOUS YEAR QUESTIONS SOLUTIONS

ENROLL NOW

TO EXCEL IN GATE
AND ACHIEVE YOUR DREAM IIT OR PSU!

ENROLL NOW

3.6. Pushdown Automata

= FA + stack of ∞ memory

Defined by the septuple,

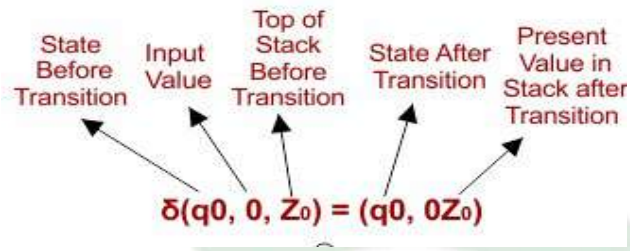
M = (Q, Σ, Γ, δ, q_0, z, F) where,

Q is a finite set of internal states of the control unit,

Σ is the input alphabet,

Γ is a finite set of symbols called the stack alphabet

δ: Q × (Σ ∪ {λ}) × Γ ⟶ set of finite subsets of Q × Γ* is the transition function,



q_0 ∈ Q is the initial state of the control unit,

z ∈ Γ is the stack start symbol,

F ⊆ Q is the set of final states.

Note: A language L is said to be a deterministic CFL if and only if there exists a DPDA M, such that L = L (M).

3.6.1. Acceptance criteria of PDA

1. **By final state:** reach (p, ε, α) with p ∈ F
2. **By empty stack:** reach (p, ε, ε) regardless of p

Note:

1. A DPDA with acceptance by EMPTY STACK is a proper subset of the languages accepted by a DPDA with final state.
2. For each DCFL which satisfies prefix property, can be accepted by a DPDA with empty stack.
3. $NPDA_{empty\ stack}$ = $NPDA_{final\ state}$ (in terms of power)

3.7. Pumping Lemma for CFL

If A is a context-free language, then there is a number p (the pumping length) where, if s is any string in A of the length at least p, then s may be divided into five pieces s = uvxyz the conditions:

1. for each i ≥ 0, uv^ixy^iz ∈ A
2. |vy| > 0, and
3. |vyx| ≤ p

Note:

- When S is being divided into uvxyz, condition 2 says that either v or y is not an empty string.
- The pieces v, x and y together have length at most p.

3.8. Pumping Lemma Length (p)

For every regular language L, ∃ positive integer 'p' such that all possible strings (w) whose length is ≥ p, if they ∈ L, then they must have a substring (s.t. 1 ≤ |substring| ≤ p) which can be pumped any no. of times & all generated strings must ∈ L.

Eg. L = 10*1, p =?

1. Let p = 1,

Strings possible with length 1 are 0,1 & they ∉ L. (∴ check for next, p ≠ 1)

2. Let p = 2,

Strings possible with length 2 are 00,01,10,11 & as 11 ∈ L, we need to check for substrings of 11 i.e. 1 & 11

   a. If we pump 1, 0 times, we get 1 ∉ L, no need to check further.
   b. If we pump 11 0 times, we get null ∉ L, no need to check further.

Conclusion, p ≠ 2.

3. Let p = 3,

Now, for |w| = 3, 101 ∈ L, we need to check for substrings of 101 i.e. 1, 0, 10, 01, 101

   a. If we pump substring '0', we find that every generated string i.e. 11 (when 0 is pumped null times), 101 (pumped once), 1001, 10001, 100001,... ∈ L.

∴ It satisfies when |w| = 3

We can also check for |w| = 4 as |w| ≥ p, 1001 ∈ L & its substrings 0 or 00 when pumped will generate strings which ∈ L.
Hence, p = 3.
Similarly, p = 4, p = 5,... will also satisfy.
We can say that p = 3 is **minimum** pumping length (M.P.L).

Note: Properties of M.P.L

1. M.P.L ≥ 1
2. M.P.L ≤ n-1, where n is no. of states in DFA (including dead state)
3. M.P.L is unique
4. M.P.L > w_{min}, where w_{min} is the minimum length string in L
5. M.P.L for finite language = $|w_{max}|+1$
6. M.P.L for Φ = 1
7. M.P.L for {∈} i.e. language which has only 1 string i.e null = $|w_{max}|+1 = 0 + 1 = 1$
8. P.L ≥ M.P.L
9. If $L = L_1 \cup L_2$, then M.P.L(L) = max(M.P.L(L_1), M.P.L(L_2))

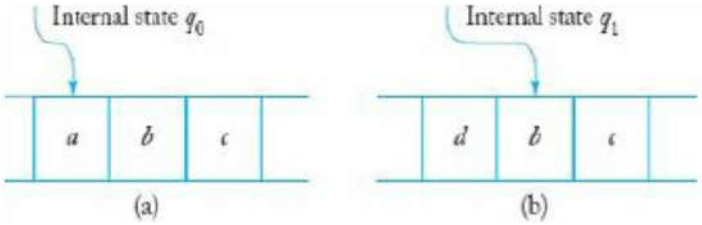
4. Turing Machine & Decidability

4.1. Turing Machine

A Turing machine M is defined by,
M = (Q, Σ, Γ, δ, q0, B, F) where,
Q: finite set of states,
Σ: input alphabet (does not include blank),
Γ: tape alphabet (Σ ∪ {**B**}, where B = blank symbol),
δ: transition function
δ:Q×Γ→Q×Γ×{L,R} (reads a symbol, writes a symbol, moves Left or Right)
$\mathbf{q_0}$ ∈ Q: start state,
F ⊆ Q: set of accepting (final) states

Eg. shows the situation before and after the move

δ (q0, a) = (q1, d, R).



Acceptance in TM: halts in some q ∈ F

Note:

1. TM >> LBA(FA + 2 counter) > NPDA(NFA + 1 counter) > DPDA (DFA + 1 counter) > NFA = DFA
2. Any TM with m symbols and n states can be simulated using **4mn + n** states by other TM.

4.2. Context Sensitive Language (CSL)

A language L ⊆ Σ* is context sensitive if it can be generated by a context-sensitive grammar (CSG).

4.2.1. Identify CSL

1. If * or / happening in power then CSL. **Eg.** L = $\{a^{n!}b^2 \mid n \geq 0\}$, L = $\{a^m b^n c^p d^q \mid m*n = p*q\}$
2. If >1 linear comparison of the same variable ('**n**' in given eg.) then CSL. Eg. L = $\{a^n b^m c^o \mid$ **n < m** & **n < o**, n,m,o ≥ 0$\}$
3. ∞ non-linear power is CSL. **Eg.** L = $\{a^{n^2} \mid n \geq 0\}$

4.2.2. Some twisted examples

1. L = $\{a^p$, p is prime$\}$ is CSL
2. L = {ww | w ∈ (a,b)*}
3. L = {ww | w ∈ $(a,b)^+$}
4. L={xww | w,x ∈ (a+b)*} might seem like CSL but **it's not CSL**. Proof: w can be ϵ and x ∈ (a+b)∗, making L=Σ* i.e. the set of strings generated by L is {ϵ,a,b,aa,ab,ba,...} = Σ* making L regular.

5. L={xww | w,x ∈ $(a+b)^+$} is CSL as Here, w can't be ϵ and hence to accept the string we do need the power of an LBA making L a CSL.

4.3. RE & REC Languages

4.3.1. Recursively Enumerable (RE)

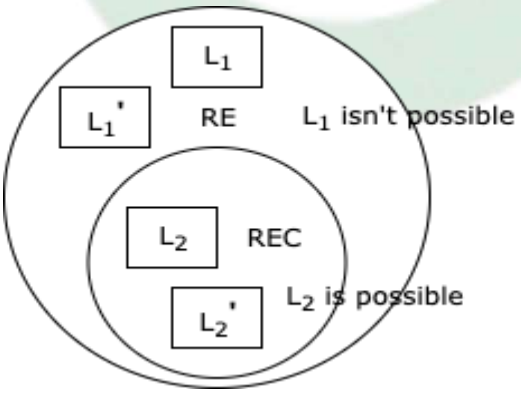
A language L is RE if there exists a Turing machine that accepts every
w ∈ L (halts and enters an accepting state), but on w ∉ L may either reject or loop forever.

4.3.2. Recursive (REC) (Decidable)

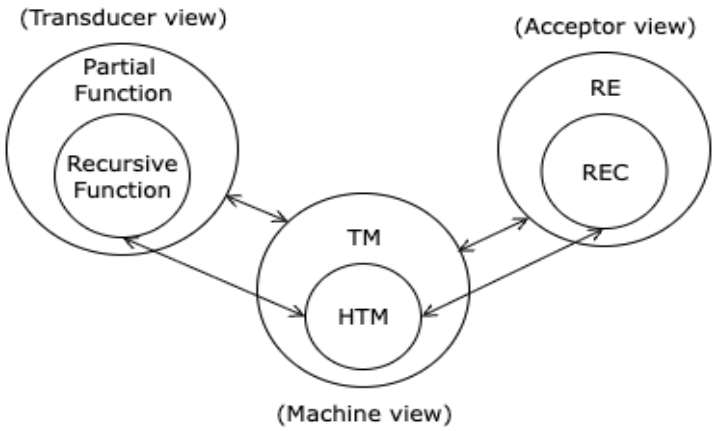
A language L is recursive if there exists a Turing machine that halts on every input, accepting exactly those in L & rejecting all others.

Note:

1. {Regular} ⊂ {CFL} ⊂ {Context-sensitive} ⊂ {REC} ⊂ {RE}
2. If L & L' are both RE then L is REC, hence also RE because every recursive language is RE but not all RE languages are recursive. (refer fig. below)



3.



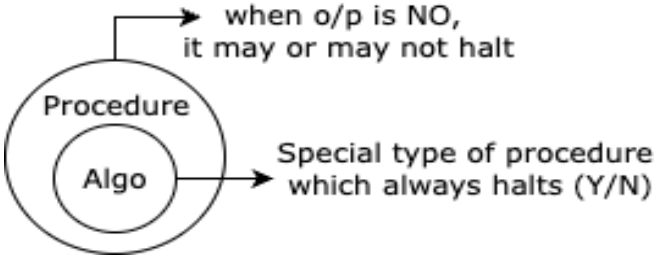
4. (RE but not REC)' = Non RE but
   RE' ≠ Non RE (i.e. it may or may not be Non RE)
5. L is REC iff L can be enumerable in lexicographic ordering.

4.4. TM variations

TM ≅ Single tape TM
TM ≅ One-way infinite tape TM
TM ≅ Two-way infinite tape TM
TM ≅ Multi tape and multi head TM
TM ≅ Universal TM
TM ≅ Multi stack PDA
TM ≅ FA with two stacks
TM ≅ FA + R/W tape + Bidirectional head

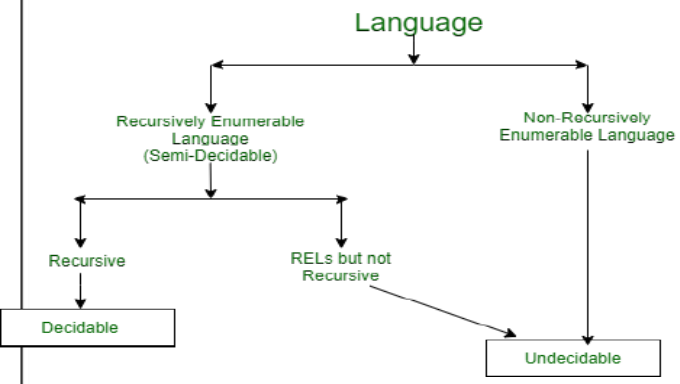
4.5. Church Turing Thesis

"What could naturally be called as effective procedure can be realised by turing machine"



4.6. Decidability

4.6.1. Decidable & Undecidable Language



4.6.2. Closure Properties

Operations	REG	DCFL	CFL	CSL	REC	RE
Union	✓	X	✓	✓	✓	✓
Intersection	✓	X	X	✓	✓	✓
Complement	✓	✓	X	✓	✓	X
Concatenation	✓	X	✓	✓	✓	✓
Kleene star	✓	X	✓	✓	✓	✓
Homomorphism	✓	X	✓	X	X	✓
Inverse Homomorphism	✓	✓	✓	✓	✓	✓
Reverse	✓	X	✓	✓	✓	✓
Substitution (∈ - free)	✓	X	✓	✓	X	✓

1. All types of languages are closed under all the operations with regular languages such as L∪R, L∩R, L-R.
2. CFLs are not closed under difference operation as L1-L2 = L1∩L2′, and CFLs are not closed under complement operation.
3. No languages are closed under subset ⊆ and Infinite union.
4. Regular languages are not closed under infinite UNION and infinite INTERSECTION.
5. If L is DCFL then so are MIN(L) and MAX(L).
6. Complement of non-regular is always non-regular
7. Let L be a DCFL and R is a regular language then L/R is DCFL.
8. DCFL U CFL = CFL
9. If something is closed under UNION and COMPLEMENT then it will be surely closed under

Operations	REG	CFL
INIT	✓	✓
L/a	✓	✓
CYCLE	✓	✓
MIN	✓	X
MAX	✓	X
HALF	✓	X
ALT	✓	X

INTERSECTION.

Let L be a language,

1. **HALF(L)** = {x | for some y such that |x| = |y| and xy ∈ L }
2. **MIN(L)** = { w | w is in L and no proper prefix of w is in L}
3. **MAX(L)** = {w | w is in L and for no x other than epsilon wx is in L}
4. **INIT(L)** = {w | for some x, wx is in L}
5. **CYCLE(L)** = { w | we can write w as w=xy such that yx is in L}
6. **ALT(L, M)** is regular provided that L and M are regular languages.
7. **SHUFFLE(L, L')** is a CFL if L is CFL and L' is regular.
8. **SUFFIX(L)** = { y | xy ∈ L for some string x }, CFL is closed under SUFFIX operation.
9. **NOPREFIX(L)** = {w ∈ A | and no prefix of w is member of A }
10. **NOEXTEND(L)** = { w ∈ A | w is not proper prefix of any string in A }
11. **DROP-OUT(L)** let A be any language, define DROP-OUT(L) to be the language containing all strings that can be obtained by removing one symbol from a string in L.
12. Regular languages are closed under NOPREFIX, NOEXTEND, and DROP-OUT operations.

4.6.3. Decidability Table

Operations	REG	DCFL	CFL	CSL	REC	RE	Comments
w ∈ L(G)	✓	✓	✓	✓	✓	X	Membership property.
L(G)=Φ	✓	✓	✓	X	X	X	Emptiness Property
L(G)=Σ*	✓	✓	X	X	X	X	Language accepts everything?
L(G1)⊆L(G2)	✓	X	X	X	X	X	Is L(G1) subset of L(G2)?
L(G1)=L(G2)	✓	✓	X	X	X	X	Are both languages equal?
L(G1)∩L(G2)=Φ	✓	X	X	X	X	X	Disjointness Property.
L(G) is regular?	✓	✓	X	X	X	X	G generates Regular language.
L(G) is finite?	✓	✓	✓	X	X	X	G generates finite language?
Ambiguity	✓	✓	X	X	X	X	Is the given grammar ambiguous?

4.6.4. Some Decidable & Undecidable Problems

1. Give a Turing machine A,
   a. A has at least 481 states (decidable)
   b. A takes more than 481 steps on epsilon (decidable)
   c. A takes more than 481 steps on some input (decidable)
   d. A takes more than 481 steps on all inputs (decidable)
2. A = {<G> | G is a CFG that generates ϵ } is decidable
3. A_{CFG} ={G | G is a CFG and L(G) = Φ} is decidable language
4. ∑* over alphabet ∑ = {a, b} is Decidable
5. {M M is DFA, M accepts ab} is Decidable
6. {<M, w> | M is a TM and M halts on input w} – Undecidable, RE
7. {<M> | M is a TM and L(M) = Φ} – Undecidable, NOT RE
8. {<M> | M is a TM and L(M)!= Φ} – Undecidable, RE
9. {<M> | M is a TM and L(M) is a regular language} – Undecidable, NOT RE
10. {<M> | M is a TM and L(M) is a REC} - Undecidable, NOT RE
11. {<M> | M is a TM and L(M) is a NOT REC}, Undecidable, NOT RE
12. {<M1, M2> | M1 and M2 are TMs and L(M1) = L(M2)} – Undecidable, NOT RE
13. {<M, w> | M is an LBA that accepts string w} – Decidable
14. {<M> | M is an LBA where L(M) = Φ} – Undecidable, NOT RE
15. {<G>|G is a context free grammar and L(G) = Σ*} Undecidable, NOT RE
16. T = {<M > | M is a TM that accepts w^r whenever it accepts w} undecidable
17. A TM ever writes a blank symbol over a non-blank symbol during the course of its computation. - Undecidable, NOT RE
18. {< G> | G is ambiguous} RE, while L3' is NOT RE.
19. L(M) has at least 10 strings – RE
20. L(M) has at most 10 strings – NOT RE
21. L = {M | M is a TM that accepts a string of length 2014} – RE; There are a finite number of strings of length 2014, if we can execute multiple instances of TM in parallel, if any string is accepted we can stop.
22. L(M) is recognized by a TM having an even number of states. Decidable (trivial property)
23. L={⟨M,w⟩ | M does not modify the tape on input w} - Decidable
24. an arbitrary TM ever prints a specific letter – Undecidable
25. Post correspondence problem (PCP) is Undecidable (RE but not REC)
26. Modified PCP is undecidable (semi-decidable)

4.7. Countable & Uncountable sets

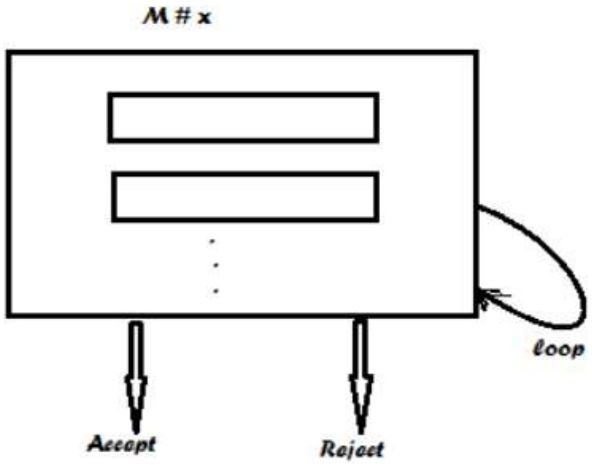
Let ∑ = {a,b}

1. ∑* is countably infinite (CI)
2. 2^{Σ^*} is uncountably infinite (UCI)
3. Set of all languages over ∑* is UCI
4. Set of all TM is CI

Hence, no. of languages over ∑* >>> No. of TM, so non RE languages exist.

5. Set of all non RE languages is UCI
6. But each non RE languages is CI

4.8. Universal/Total Turing Machine (UTM)



1. m#x is an input to UTM where n is binary coding of a valid TM.
2. # is a separator.
3. X is an input string to Turing machine M, x is binary representation.
4. We will simulate TM M on x by UTM, if M accepts x, then UTM will accept m#x.
5. If m is not a valid representation of a TM, reject it without doing any simulation. UTM will reject it.

Note:
Decidable – If the language/property has a total TM.
Semi-decidable – If the language has just a TM.
Undecidable – No TM exists.

4.9. Rice's Theorem

"Any non-trivial property of the language recognizable by a Turing machine (i.e. RE language) is Undecidable i.e. Not REC"

How to apply? A property of RE languages is non-trivial if there are two RE languages: one that has the property and one that doesn't. Equivalently, there must exist two Turing machines, T_{yes} whose language satisfies the property and T_{no} whose language does not.

Note: Any "trivial" property is always decidable.

Eg.

1. **L(M) has at least 10 strings**

We can have T_{yes} which accepts at least 10 strings and T_{no} which doesn't accept at least 10 strings. Hence, L={M | L(M) has at least 10 strings} - Undecidable (Not REC)

2. **L(M) has at most 10 strings**

We can have T_{yes} & T_{no}. Hence, L={M | L(M) has at most 10 strings} - Undecidable (Not REC)

3. **L(M) is recognized by a TM having even number of states**

This is a trivial property because for any RE language we have a TM and even if that TM is having an odd number of states we can make an equivalent TM having even number of states by adding one extra state. Thus this set equals the set of RE languages and hence decidable.

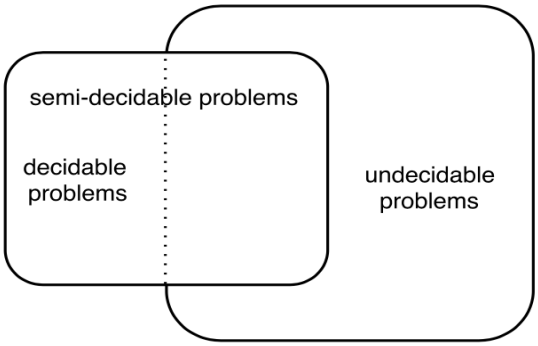
4. **L(M) is a subset of Σ***

This is a trivial property. All languages are subset of Σ* and hence this set contains all languages including all RE languages.

4.9.1. Advanced Rice's Theorem

Any non-monotonic property of the language recognizable by a Turing machine (RE language) is unrecognizable i.e. Non RE.

Need? To further identify the Not REC language because a language which is Not REC can either be RE but Not REC (Semi-Decidable) or Non RE (Undecidable).



Non-monotonic property? A property of RE languages is non-monotonic if you can find two RE languages, one (with TM T_{yes}) that has the property & another (with TM T_{no}) that doesn't; such that the first language is a proper subset of the second.

Eg. L(M)={0}

We can have T_{yes} for {0} and T_{no} for Σ^* ($\{0\} \subset \Sigma^*$) ∴ L={M | L(M)={0}} is Non RE

GeeksforGeeks

GATE CSE BATCH

KEY HIGHLIGHTS:

- 300+ HOURS OF RECORDED CONTENT
- 900+ HOURS OF LIVE CONTENT
- SKILL ASSESSMENT CONTESTS
- 6 MONTHS OF 24/7 ONE-ON-ONE AI DOUBT ASSISTANCE
- SUPPORTING NOTES/DOCUMENTATION AND DPPS FOR EVERY LECTURE

COURSE COVERAGE:

- ENGINEERING MATHEMATICS
- GENERAL APTITUDE
- DISCRETE MATHEMATICS
- DIGITAL LOGIC
- COMPUTER ORGANIZATION AND ARCHITECTURE
- C PROGRAMMING
- DATA STRUCTURES
- ALGORITHMS
- THEORY OF COMPUTATION
- COMPILER DESIGN
- OPERATING SYSTEM
- DATABASE MANAGEMENT SYSTEM
- COMPUTER NETWORKS

LEARNING BENEFIT:

- GUIDANCE FROM EXPERT MENTORS
- COMPREHENSIVE GATE SYLLABUS COVERAGE
- EXCLUSIVE ACCESS TO E-STUDY MATERIALS
- ONLINE DOUBT-SOLVING WITH AI
- QUIZZES, DPPS AND PREVIOUS YEAR QUESTIONS SOLUTIONS

ENROLL NOW

TO EXCEL IN GATE
AND ACHIEVE YOUR DREAM IIT OR PSU!

ENROLL NOW