UNIT-IV

Pushdown Automata

A Pushdown Automata (PDA) can be defined as :

- Q is the set of states
- ∑is the set of input symbols
- Γ is the set of pushdown symbols (which can be pushed and popped from stack)
- q0 is the initial state
- Z is the initial pushdown symbol (which is initially present in stack)
- F is the set of final states
- δ is a transition function which maps Q x {Σ ∪ ∈} x Γ into Q x Γ*. In a given state, PDA will read input symbol and stack symbol (top of the stack) and move to a new state and change the symbol of stack.

Instantaneous Description (ID)

Instantaneous Description (ID) is an informal notation of how a PDA "computes" a input string and make a decision that string is accepted or rejected.

A ID is a triple (q, w, α), where:
1. q is the current state.
2. w is the remaining input.
3.α is the stack contents, top at the left.

Turnstile notation

⊢ sign is called a "turnstile notation" and represents one move.
⊢* sign represents a sequence of moves.
Eg- (p, b, T) ⊢ (q, w, α)
This implies that while taking a transition from state p to state q, the input symbol 'b' is consumed, and the top of the stack 'T' is replaced by a new string 'α'

Example : Define the pushdown automata for language $\{a^n b^n \mid n > 0\}$
Solution : M = where Q = { q0, q1 } and Σ = { a, b } and Γ = { A, Z } and δ is given by :

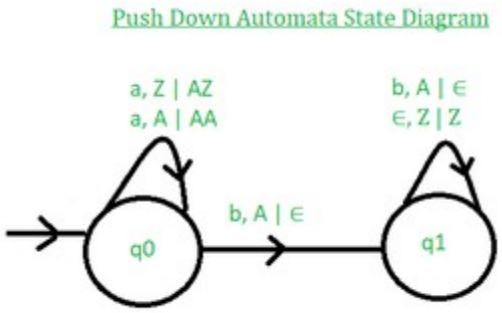
δ(q0, a, Z) = { (q0, AZ) }
δ(q0, a, A) = { (q0, AA) }
δ(q0, b, A) = { (q1, ∈) }
δ(q1, b, A) = { (q1, ∈) }
δ(q1, ∈, Z) = { (q1, ∈) }

Let us see how this automata works for aaabbb.

Row	State	Input	δ(transition function used)	Stack(Leftmost symbol represents top of stack)	State after move
1	q0	aaabbb		Z	q0
2	q0	**a**aabbb	δ(q0,a,Z)={(q0,AZ)}	AZ	q0
3	q0	a**a**abbb	δ(q0,a,A)={(q0,AA)}	AAZ	q0
4	q0	aa**a**bbb	δ(q0,a,A)={(q0,AA)}	AAAZ	q0
5	q0	aaa**b**bb	δ(q0,b,A)={(q1,∈)}	AAZ	q1
6	q1	aaab**b**b	δ(q1,b,A)= {(q1,∈)}	AZ	q1
7	q1	aaabb**b**	δ(q1,b,A)= {(q1,∈)}	Z	q1
8	q1	∈	δ(q1, ∈,Z)= {(q1,∈)}	∈	q1

Explanation : Initially, the state of automata is q0 and symbol on stack is Z and the input is aaabbb as shown in row 1. On reading ‘a’ (shown in bold in row 2), the state will remain q0 and it will push symbol A on stack. On next ‘a’ (shown in row 3), it will push another symbol A on stack. After reading 3 a’s, the stack will be AAAZ with A on the top. After reading ‘b’ (as shown in row 5), it will pop A and move to state q1 and stack will be AAZ. When all b’s are read, the state will be q1 and stack will be Z. In row 8, on input symbol ‘∈’ and Z on stack, it will pop Z and stack will be empty. This type of acceptance is known as **acceptance by empty stack.**

Push Down Automata State Diagram:



Note :

- The above pushdown automaton is deterministic in nature because there is only one move from a state on an input symbol and stack symbol.
- The non-deterministic pushdown automata can have more than one move from a state on an input symbol and stack symbol.
- It is not always possible to convert non-deterministic pushdown automata to deterministic pushdown automata.
- The expressive power of non-deterministic PDA is more as compared to expressive deterministic PDA as some languages are accepted by NPDA but not by deterministic PDA which will be discussed in the next article.
- The pushdown automata can either be implemented using acceptance by empty stack or acceptance by final state and one can be converted to another.

We have discussed Pushdown Automata (PDA) and its acceptance by empty stack article. Now, in this article, we will discuss how PDA can accept a CFL based on the final state. Given a PDA P as:

P = (Q, Σ, Γ, δ, q0, Z, F)

The language accepted by P is the set of all strings consuming which PDA can move from initial state to final state irrespective of any symbol left on the stack which can be depicted as:

L(P) = {w |(q0, w, Z) =>(qf, ε, s)}

Here, from start state q0 and stack symbol Z, the final state qf ε F is reached when input w is consumed. The stack can contain a string s which is irrelevant as the final state is reached and w will be accepted.

Example: Define the pushdown automata for language {a^nb^n | n > 0} using final state.

Solution: M = where Q = {q0, q1, q2, q3} and ∑ = {a, b} and Γ = { A, Z } and F={q3} and δ is given by:

δ(q0, a, Z) = { (q1, AZ) }

δ(q1, a, A) = { (q1, AA) }

δ(q1, b, A) = { (q2, ε) }

δ(q2, b, A) = { (q2, ε) }

δ(q2, ε, Z) = { (q3, Z) }

Let us see how this automaton works for aaabbb:

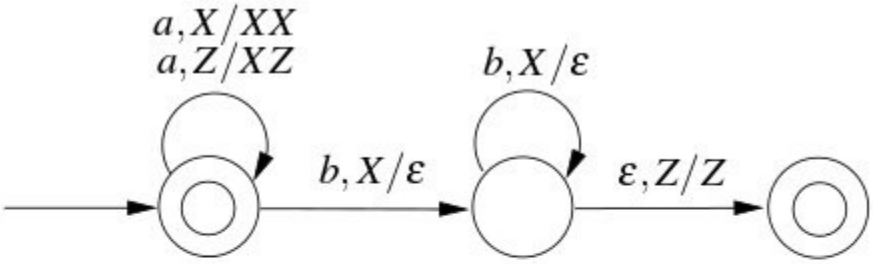
Row	State	Input	δ (transition function) used	Stack (Leftmost symbol represents top of stack)	State after move
0	q0	aaabbb		Z	
1	q0	**a**aabbb	δ(q0,a,Z)={(q1,AZ)}	AZ	q1
2	q1	a**a**abbb	δ(q1,a,A)={(q1,AA)}	AAZ	q1
3	q1	aa**a**bbb	δ(q1,a,A)={(q1,AA)}	AAAZ	q1
4	q1	aaa**b**bb	δ(q1,b,A)={(q2, ε)}	AAZ	q2
5	q2	aaab**b**b	δ(q2,b,A)={(q2,ε)}	AZ	q2
6	q2	aaabb**b**	δ(q2,b,A)={(q2,ε)}	Z	q2
7	q2	ε	δ(q2,ε,Z)={(q3,ε)}	Z	q3

Explanation: Initially, the state of automata is q0 and symbol on the stack is Z and the input is aaabbb as shown in row 0. On reading a (shown in bold in row 1), the state will be changed to q1 and it will push symbol A on the stack. On next a (shown in row 2), it will push another symbol A on the stack and remain in state q1. After reading 3 a's, the stack will be AAAZ with A on the top. After reading b (as shown in row 4), it will pop A and move to state q2 and the stack will be AAZ. When all b's are read, the state will be q2 and the stack will be Z. In row 7, on input symbol ε and Z on the stack, it will move to q3. As the final state q3 has been reached after processing input, the string will be accepted. This type of acceptance is known as *acceptance by the final state*. Next, we will see how this automata works for aab:

Row	State	Input	δ (transition function) used	Stack (Leftmost symbol represents top of stack)	State after move
0	q0	aab		Z	
1	q0	**a**ab	δ(q0,a,Z)={(q1,AZ)}	AZ	q1
2	q1	a**a**b	δ(q1,a,A)={(q1,AA)}	AAZ	q1
3	q1	aa**b**	δ(q1,b,A)={(q2,ε)}	AZ	q2
4	q2	ε		AZ	

As we can see in row 4, the input has been processed and PDA is in state q2 which is a non-final state, the string aab will not be accepted. Let us discuss the question based on this:

Que-1. Consider the transition diagram of a PDA given below with input alphabet ∑ = {a, b}and stack alphabet Γ = {X, Z}. Z is the initial stack symbol. Let L denote the language accepted by the PDA. (GATE-CS-2016)



Which one of the following is **TRUE**?

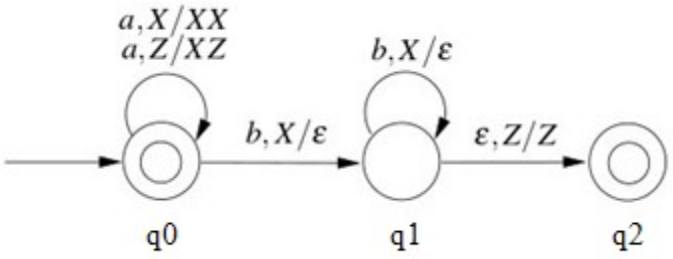
(A) $L = \{a^n b^n | n \geq 0\}$ and is not accepted by any finite automata

(B) $L = \{a^n | n \geq 0\} \cup \{a^n b^n | n \geq 0\}$ and is not accepted by any deterministic PD

(C) L is not accepted by any Turing machine that halts on every input

(D) $L = \{a^n | n \geq 0\} \cup \{a^n b^n | n \geq 0\}$ and is deterministic context-free

Solution: We first label the state of the given PDA as:



Next, the given PDA P can be written as:

Q = {q0, q1, q2} and ∑ = {a, b}

And Γ = {X, Z} and F={q0,q2} and δ is given by :

δ(q0, a, Z) = {(q0, XZ)}

δ(q0, a, X) = {(q0, XX)}

δ(q0, b, X) = {(q1, ε)}

δ(q1, b, X) = {(q1, ε)}

δ(q1, ε, Z) = {(q2, Z)}

As we can see, q0 is the initial as well as the final state, ε will be accepted. For every a, X is pushed onto the stack and PDA remains in the final state. Therefore, any number of a's can be accepted by PDA. If the input contains b, X is popped from the stack for every b. Then PDA is moved to the final state if the stack becomes empty after processing input (δ(q1, ε, Z) = {(q2, Z)}). Therefore, a number of b must be equal to the number of an if they exist. As there is only one move for a given state and input, the PDA is deterministic. So, the correct option is (D).

Construct Pushdown Automata for given languages

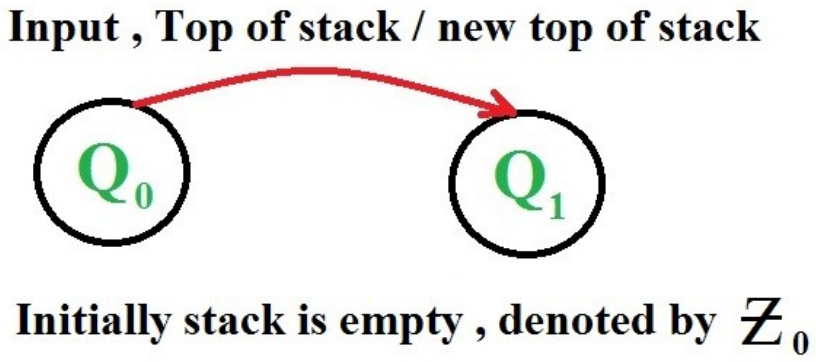
A push down automata is similar to deterministic finite automata except that it has a few more properties than a DFA.The data structure used for implementing a PDA is stack. A PDA has an output associated with every input. All the inputs are either pushed into a stack or just ignored. User can perform the basic push and pop operations on the stack which is use for PDA. One of the problems associated with DFAs was that could not make a count of number of characters which were given input to the machine. This problem is avoided by PDA as it uses a stack which provides us this facility also.

A Pushdown Automata (PDA) can be defined as –

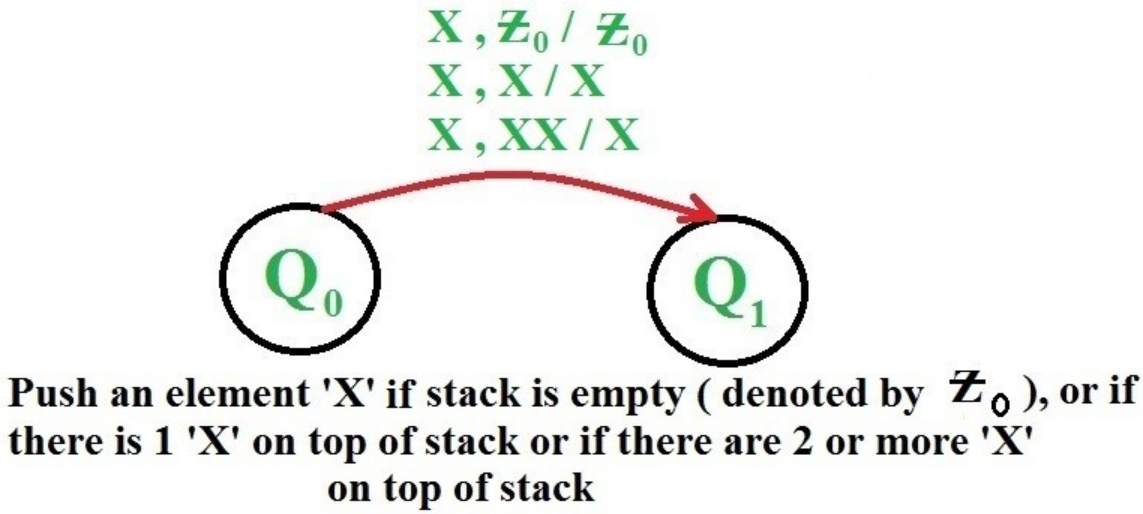
M = (Q, Σ, Γ, δ, q0, Z, F) where

- Q is a finite set of states
- Σ is a finite set which is called the input alphabet
- Γ is a finite set which is called the stack alphabet
- δ is a finite subset of Q X (Σ ∪ {ε} X Γ X Q X Γ^*) the transition relation.
- $q_0 \in Q$ is the start state
- Z ∈ Γ is the initial stack symbol
- F ⊆ Q is the set of accepting states

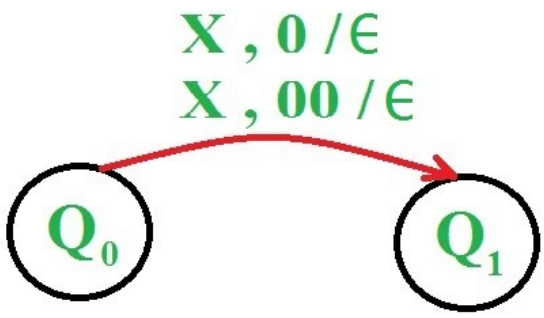
Representation of State Transition –



Representation of Push in a PDA –



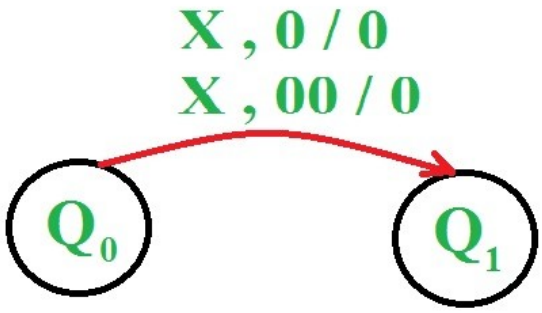
Representation of Pop in a PDA –



Pop an element 'X' if we have 1 or more 0's on top of stack

∈ shows deletion or pop

Representation of Ignore in a PDA –



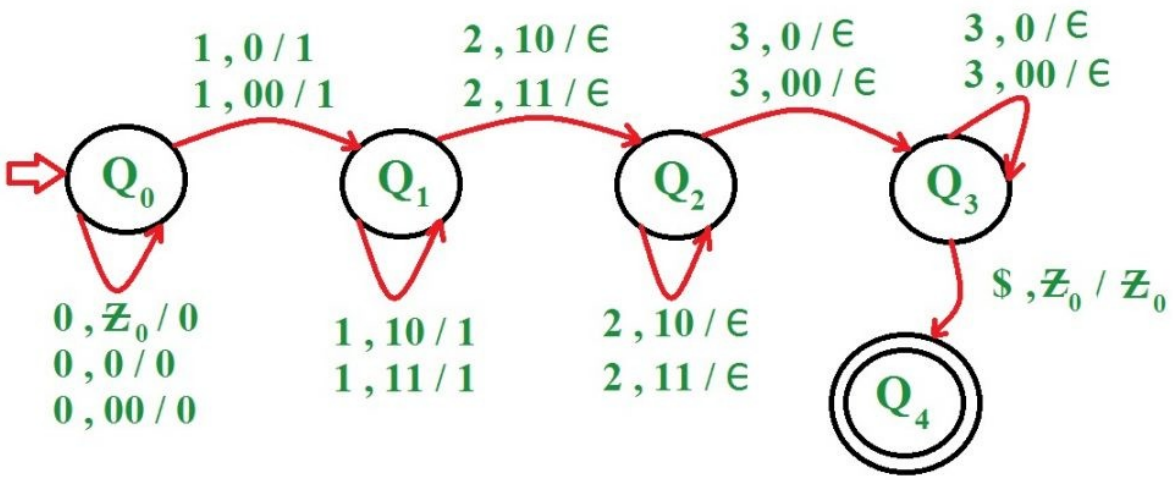
Ignore an element 'X' if we have 1 or more 0's on top of stack

Q) Construct a PDA for language L = {$0^n1^m2^m3^n$ | n>=1, m>=1}

Approach used in this PDA –
First 0's are pushed into stack. Then 1's are pushed into stack.
Then for every 2 as input a 1 is popped out of stack. If some 2's are still left and top of stack is a 0 then string is not accepted by the PDA. Thereafter if 2's are finished and top of stack is a 0 then for every 3 as input equal number of 0's are popped out of stack. If string is finished and stack is empty then string is accepted by the PDA otherwise not accepted.

- **Step-1:** On receiving 0 push it onto stack. On receiving 1, push it onto stack and goto next state
- **Step-2:** On receiving 1 push it onto stack. On receiving 2, pop 1 from stack and goto next state

- **Step-3:** On receiving 2 pop 1 from stack. If all the 1's have been popped out of stack and now receive 3 then pop a 0 from stack and goto next state
- **Step-4:** On receiving 3 pop 0 from stack. If input is finished and stack is empty then goto last state and string is accepted



Examples:

Input : 0 0 1 1 1 2 2 2 3 3

Result : ACCEPTED

Input : 0 0 0 1 1 2 2 2 3 3

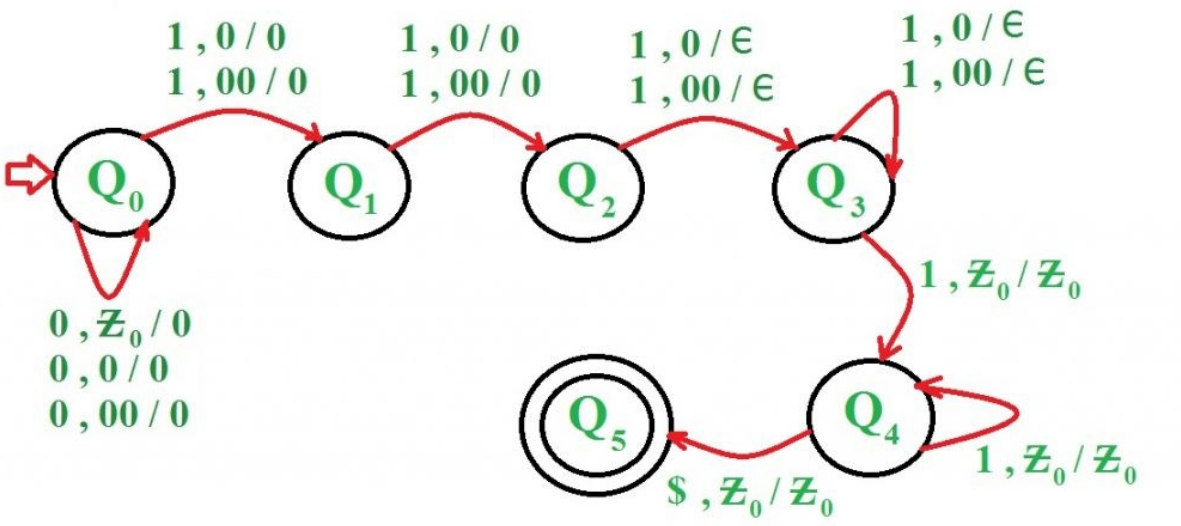
Result : NOT ACCEPTED

Construct a PDA for language L = {0^n1^m | n >= 1, m >= 1, m > n+2}

Approach used in this PDA –
First 0's are pushed into stack.When 0's are finished, two 1's are ignored. Thereafter for every 1 as input a 0 is popped out of stack. When stack is empty and still some 1's are left then all of them are ignored.

- **Step-1:** On receiving 0 push it onto stack. On receiving 1, ignore it and goto next state
- **Step-2:** On receiving 1, ignore it and goto next state
- **Step-3:** On receiving 1, pop a 0 from top of stack and go to next state

- **Step-4:** On receiving 1, pop a 0 from top of stack. If stack is empty, on receiving 1 ignore it and goto next state
- **Step-5:** On receiving 1 ignore it. If input is finished then goto last state



Examples:

Input : 0 0 0 1 1 1 1 1 1

Result : ACCEPTED

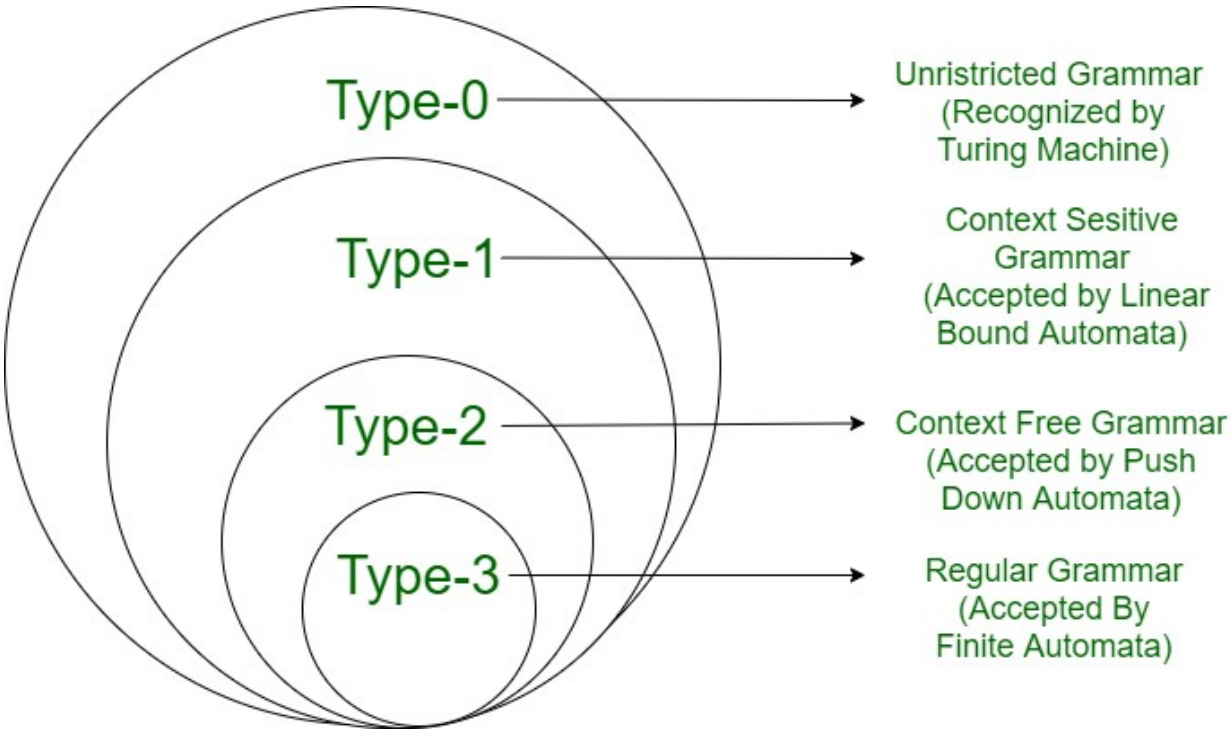
Input : 0 0 0 0 1 1 1 1

Result : NOT ACCEPTED

Chomsky Hierarchy in Theory of Computation

According to Chomsky hierarchy, grammar is divided into 4 types as follows:

1. Type 0 is known as unrestricted grammar.
2. Type 1 is known as context-sensitive grammar.
3. Type 2 is known as a context-free grammar.
4. Type 3 Regular Grammar.



Type 0: Unrestricted Grammar:

Type-0 grammars include all formal grammar. Type 0 grammar languages are recognized by turing machine. These languages are also known as the Recursively Enumerable languages.

Grammar Production in the form

of where

\alpha is (V + T)* V (V + T)*

V : Variables

T : Terminals.

is (V + T)*.

In type 0 there must be at least one variable on the Left side of production.

For example:

Sab --> ba

A --> S

Here, Variables are S, A, and Terminals a, b.

Type 1: Context-Sensitive Grammar

Type-1 grammars generate context-sensitive languages. The language generated by the grammar is recognized by the Linear Bound Automata

In Type 1

- First of all Type 1 grammar should be Type 0.
- Grammar Production in the form of

|\alpha |<=|\beta |

That is the count of symbol in is less than or equal to

Also $\beta \in (V + T)^+$

i.e. β can not be ε

For Example:

S --> AB

AB --> abc

B --> b

Type 2: Context-Free Grammar: Type-2 grammars generate context-free languages. The language generated by the grammar is recognized by a Pushdown automata. In Type 2:

- First of all, it should be Type 1.
- The left-hand side of production can have only one variable and there is no restriction on |\alpha | = 1.

For example:

S --> AB

A --> a

B --> b

Type 3: Regular Grammar: Type-3 grammars generate regular languages. These languages are exactly all languages that can be accepted by a finite-state automaton. Type 3 is the most restricted form of grammar.

Type 3 should be in the given form only :

V --> VT / T (left-regular grammar)

(or)

V --> TV /T (right-regular grammar)

For example:

S --> a

The above form is called strictly regular grammar.

There is another form of regular grammar called extended regular grammar. In this form:

V --> VT* / T*. (extended left-regular grammar)

(or)

V --> T*V /T* (extended right-regular grammar)

For example :

S --> ab.

Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Undecidability

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

Examples

- **Equivalence of two regular languages:** Given two regular languages, there is an algorithm and Turing machine to decide whether two regular languages are equal or not.
- **Finiteness of regular language:** Given a regular language, there is an algorithm and Turing machine to decide whether regular language is finite or not.
- **Emptiness of context free language:** Given a context free language, there is an algorithm whether CFL is empty or not.

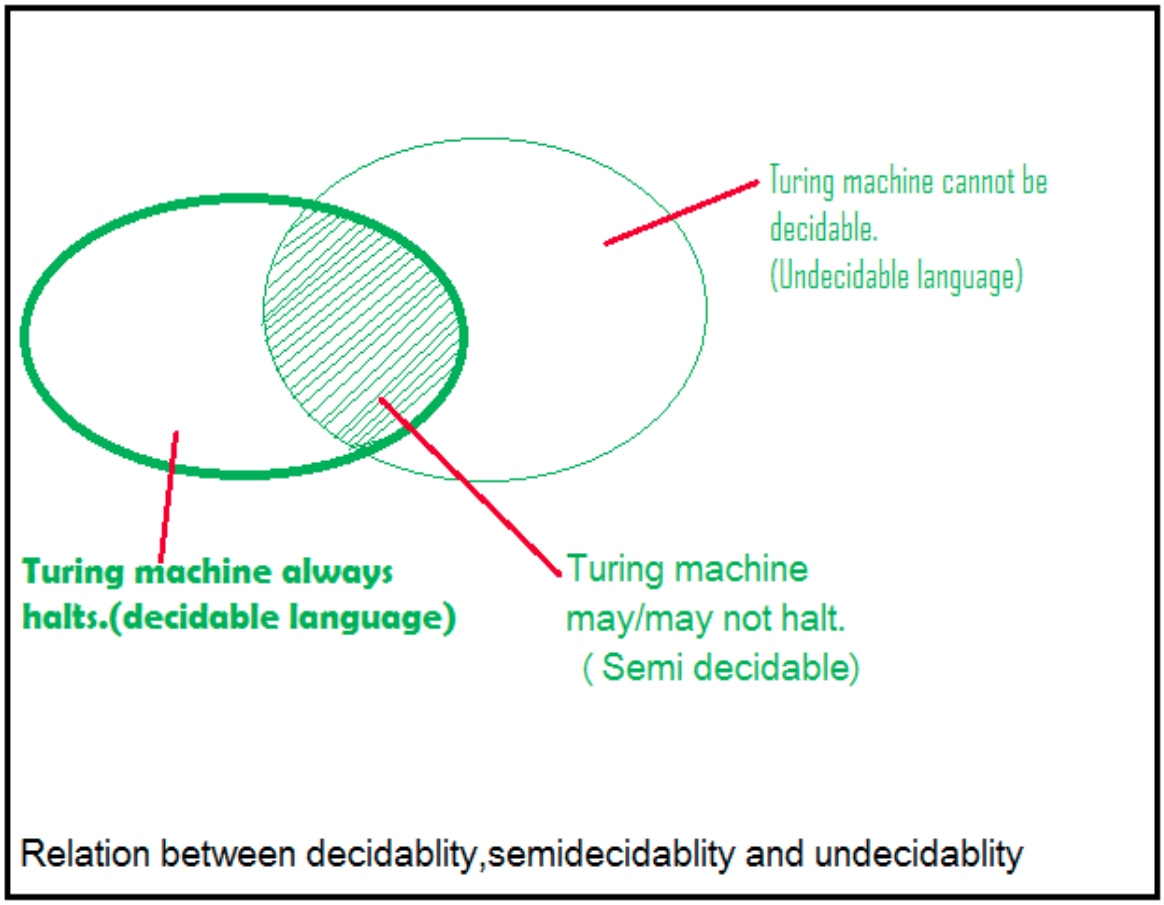
Undecidable Problems
A problem is undecidable if there is no Turing machine which will always halt in finite amount of time to give answer as ‘yes’ or ‘no’. An undecidable problem has no algorithm to determine the answer for a given input.

Examples

- **Ambiguity of context-free languages:** Given a context-free language, there is no Turing machine which will always halt in finite amount of time and give answer whether language is ambiguous or not.

- **Equivalence of two context-free languages:** Given two context-free languages, there is no Turing machine which will always halt in finite amount of time and give answer whether two context free languages are equal or not.
- **Everything or completeness of CFG:** Given a CFG and input alphabet, whether CFG will generate all possible strings of input alphabet (Σ^*)is undecidable.
- **Regularity of CFL, CSL, REC and REC:** Given a CFL, CSL, REC or REC, determining whether this language is regular is undecidable.

Note: Two popular undecidable problems are halting problem of TM and PCP (Post Correspondence Problem). Semi-decidable Problems
A semi-decidable problem is subset of undecidable problems for which Turing machine will always halt in finite amount of time for answer as 'yes' and may or may not halt for answer as 'no'.
Relationship between semi-decidable and decidable problem has been shown in Figure 1 as:



Relation between decidablity,semidecidablity and undecidablity

Rice's Theorem
Every non-trivial (answer is not known) problem on Recursive Enumerable languages is undecidable.e.g.; If a language is Recursive Enumerable, its complement will be recursive enumerable or not is undecidable.

Reducibility and Undecidability

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

$$w \varepsilon A <=> f(w) \varepsilon B$$

Theorem 1: If A≤B and B is decidable then A is also decidable.
Theorem 2: If A≤B and A is undecidable then B is also undecidable.

Question: Which of the following is/are undecidable?

1. G is a CFG. Is $L(G)=\phi$?
2. G is a CFG. Is $L(G)=\sum^*$?
3. M is a Turing machine. Is L(M) regular?
4. A is a DFA and N is an NFA. Is L(A)=L(N)?

A. 3 only
B. 3 and 4 only
C. 1, 2 and 3 only
D. 2 and 3 only

Explanation:

- Option 1 is whether a CFG is empty or not, this problem is decidable.
- Option 2 is whether a CFG will generate all possible strings (everything or completeness of CFG), this problem is undecidable.
- Option 3 is whether language generated by TM is regular is undecidable.
- Option 4 is whether language generated by DFA and NFA are same is decidable. So option D is correct.

Question: Which of the following problems are decidable?

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

A. 1,2,3,4
B. 1,2
C. 2,3,4
D. 3,4

Explanation:

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

- Context free languages are not closed under complementation, option 2 is undecidable.
- Option 1 is also undecidable as there is no TM to determine whether a given program will produce an output. **So, option D is correct.**

Question: Consider three decision problems P1, P2 and P3. It is known that P1 is decidable and P2 is undecidable. Which one of the following is TRUE?

A. P3 is undecidable if P2 is reducible to P3
B. P3 is decidable if P3 is reducible to P2's complement
C. P3 is undecidable if P3 is reducible to P2
D. P3 is decidable if P1 is reducible to P3

Explanation:

- Option A says P2≤P3. According to theorem 2 discussed, if P2 is undecidable then P3 is undecidable. It is given that P2 is undecidable, so P3 will also be undecidable. So option **(A) is correct**.
- Option C says P3≤P2. According to theorem 2 discussed, if P3 is undecidable then P2 is undecidable. But it is not given in question about undecidability of P3. So option **(C) is not correct.**
- Option D says P1≤P3. According to theorem 1 discussed, if P3 is decidable then P1 is also decidable. But it is not given in question about decidability of P3. So option **(D) is not correct**.
- Option (B) says P3≤P2'. According to theorem 2 discussed, if P3 is undecidable then P2' is undecidable. But it is not given in question about undecidability of P3. So option **(B) is not correct**.

Quiz on Undecidability

This article is contributed by **Sonal Tuteja**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above