Theory of Computing

CSE-203

NDFA to DFA conversion, Language Decidability, PushDown Automata

PushDown Automata

A pushdown automaton is a way to implement a context-free grammar in a similar way we design DFA for a regular grammar. A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information.

Basically a pushdown automaton is −

"Finite state machine" + "a stack"

A pushdown automaton has three components −

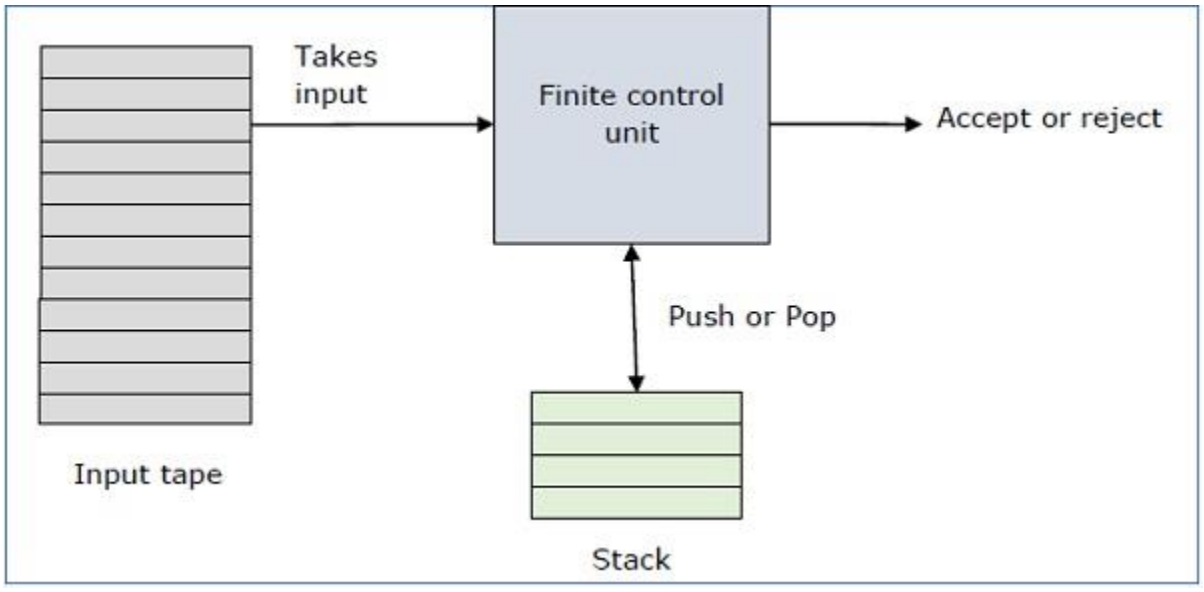
- an input tape,
- a control unit, and
- a stack with infinite size.

The stack head scans the top symbol of the stack.

A stack does two operations −

- **Push** − a new symbol is added at the top.
- **Pop** − the top symbol is read and removed.

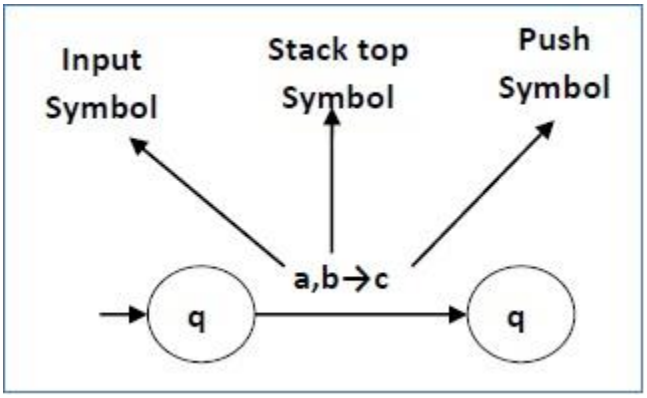
A PDA may or may not read an input symbol, but it has to read the top of the stack in every transition.



A PDA can be formally described as a 7-tuple $(Q, \sum, S, \delta, q_0, I, F)$ −

- **Q** is the finite number of states
- **∑** is input alphabet
- **S** is stack symbols
- **δ** is the transition function: $Q \times (\sum \cup \{\varepsilon\}) \times S \times Q \times S^*$
- **q_0** is the initial state ($q_0 \in Q$)
- **I** is the initial stack top symbol ($I \in S$)
- **F** is a set of accepting states ($F \in Q$)

The following diagram shows a transition in a PDA from a state q_1 to state q_2, labeled as a,b → c −



This means at state **q_1**, if we encounter an input string **'a'** and top symbol of the stack is **'b'**, then we pop **'b'**, push **'c'** on top of the stack and move to state **q_2**.

NDFA to DFA conversion

Problem

Let **$X = (Q_x, \sum, \delta_x, q_0, F_x)$** be an NDFA which accepts the language L(X). We have to design an equivalent DFA **$Y = (Q_y, \sum, \delta_y, q_0, F_y)$** such that **L(Y) = L(X)**. The following procedure converts the NDFA to its equivalent DFA −

Algorithm

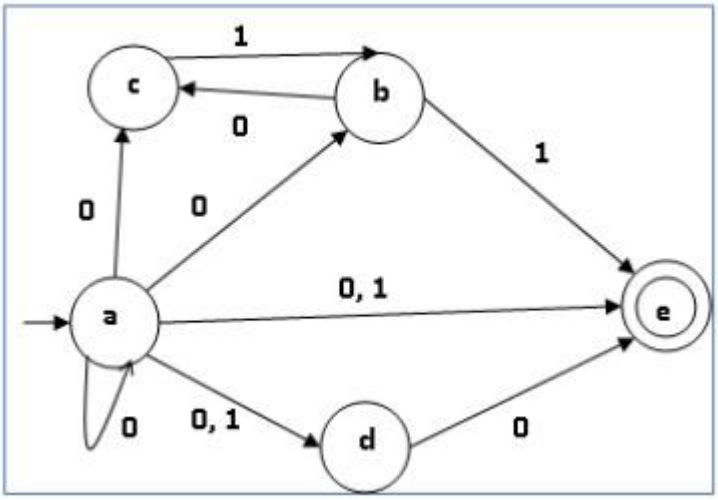
Input − An NDFA
Output − An equivalent DFA
Step 1 − Create state table from the given NDFA.
Step 2 − Create a blank state table under possible input alphabets for the equivalent DFA.
Step 3 − Mark the start state of the DFA by q0 (Same as the NDFA).
Step 4 − Find out the combination of States $\{Q_0, Q_1, \ldots, Q_n\}$ for each possible input alphabet.
Step 5 − Each time we generate a new DFA state under the input alphabet columns, we have to apply step 4 again, otherwise go to step 6.

Step 6 − The states which contain any of the final states of the NDFA are the final states of the equivalent DFA.

Example

Let us consider the NDFA shown in the figure below.

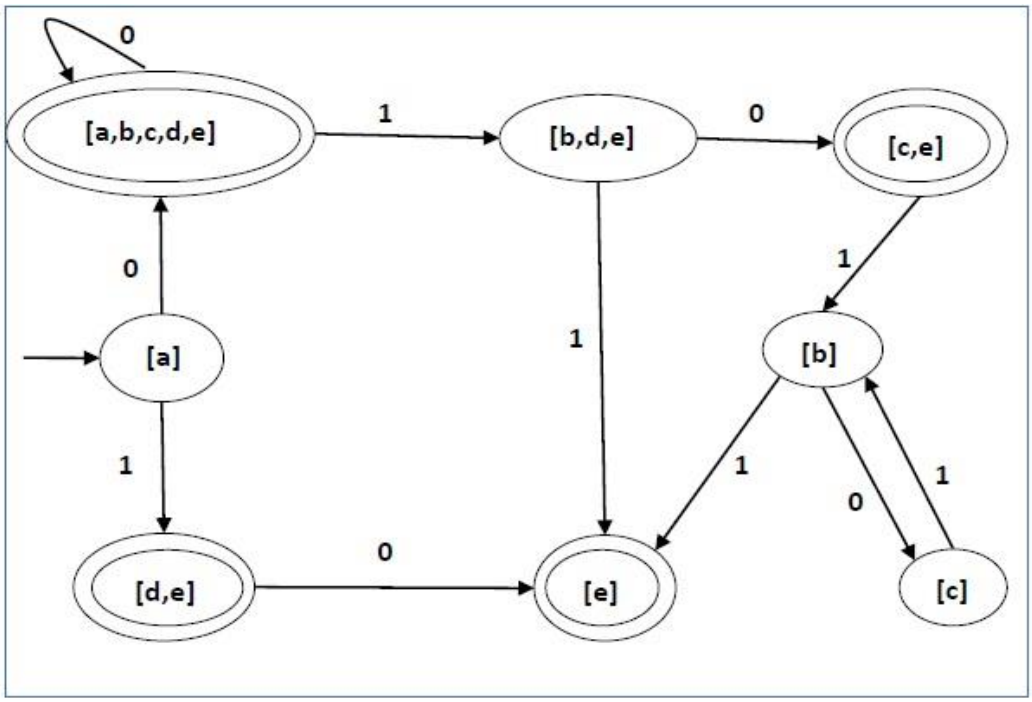
q	δ(q,0)	δ(q,1)
a	{a,b,c,d,e}	{d,e}
b	{c}	{e}
c	Ø	{b}
d	{e}	Ø
e	Ø	Ø



Using the above algorithm, we find its equivalent DFA. The state table of the DFA is shown in below.

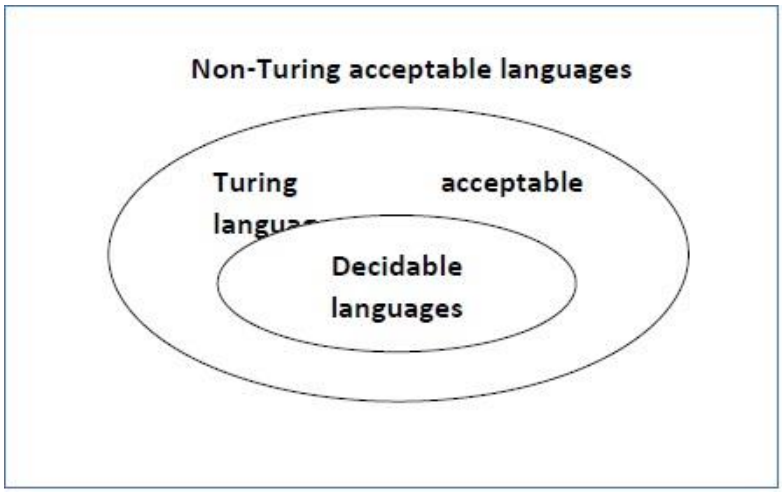
q	δ(q,0)	δ(q,1)
[a]	[a,b,c,d,e]	[d,e]
[a,b,c,d,e]	[a,b,c,d,e]	[b,d,e]
[d,e]	[e]	Ø
[b,d,e]	[c,e]	[e]
[e]	Ø	Ø
[c, e]	Ø	[b]
[b]	[c]	[e]
[c]	Ø	[b]

The state diagram of the DFA is as follows −

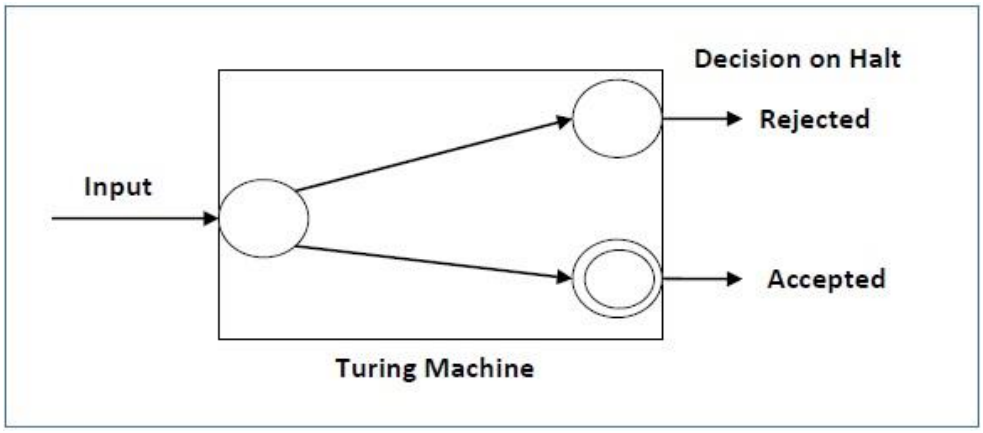


Language Decidability

A language is called **Decidable** or **Recursive** if there is a Turing machine which accepts and halts on every input string **w**. Every decidable language is Turing-Acceptable.



A decision problem **P** is decidable if the language **L** of all yes instances to **P** is decidable.
For a decidable language, for each input string, the TM halts either at the accept or the reject state as depicted in the following diagram –



Example 1

Find out whether the following problem is decidable or not –
Is a number 'm' prime?

Solution

Prime numbers = {2, 3, 5, 7, 11, 13, ……..}

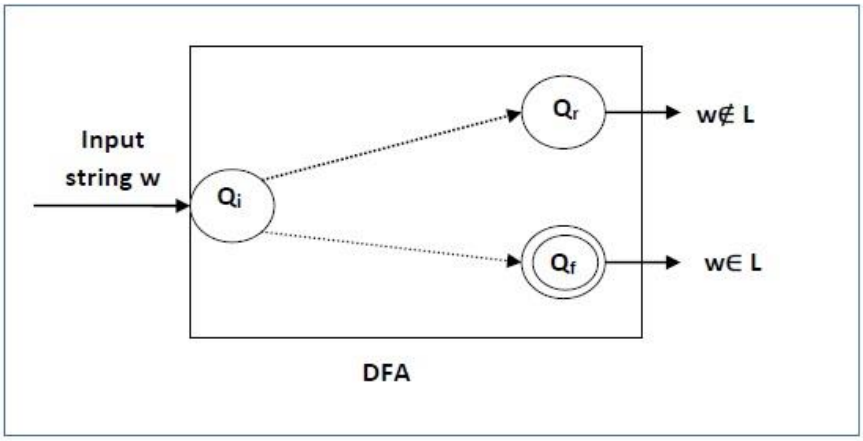
Divide the number **'m'** by all the numbers between '2' and '√m' starting from '2'.
If any of these numbers produce a remainder zero, then it goes to the "Rejected state", otherwise it goes to the "Accepted state". So, here the answer could be made by 'Yes' or 'No'.
Hence, it is a decidable problem.

Example 2

Given a regular language **L** and string **w**, how can we check if **w ∈ L**?

Solution

Take the DFA that accepts **L** and check if **w** is accepted



Some more decidable problems are −

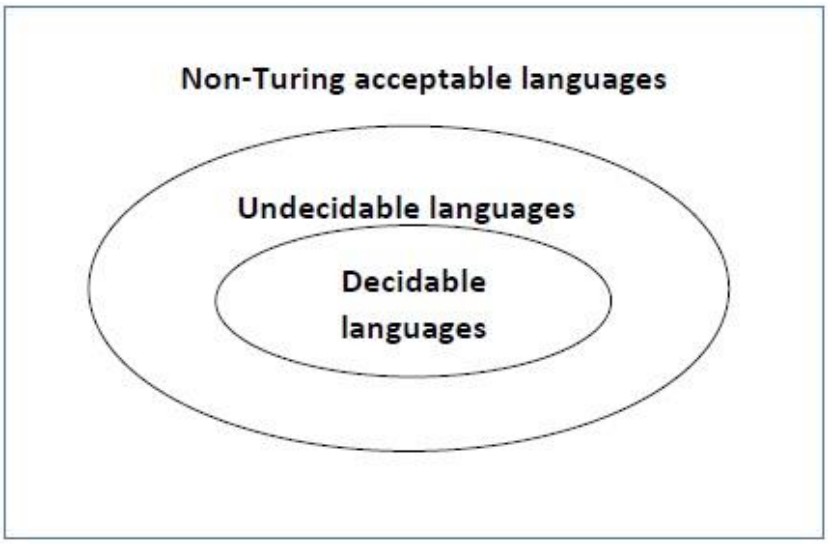
- Does DFA accept the empty language?
- Is $L_1 \cap L_2 = \emptyset$ for regular sets?

Note −

- If a language **L** is decidable, then its complement **L'** is also decidable
- If a language is decidable, then there is an enumerator for it.

Undecidable Languages

For an undecidable language, there is no Turing Machine which accepts the language and makes a decision for every input string **w** (TM can make decision for some input string though). A decision problem **P** is called "undecidable" if the language **L** of all yes instances to **P** is not decidable. Undecidable languages are not recursive languages, but sometimes, they may be recursively enumerable languages.



Example

- The halting problem of Turing machine
- The mortality problem
- The mortal matrix problem
- The Post correspondence problem, etc.

Homework

Construct deterministic pushdown automata to accept the following languages.

a. $\{10^n1^n \mid n>0\} \cup \{110^n1^{2n} \mid n>0\}$
b. Binary strings that contain an equal number of 1s and 0s
c. Binary strings with twice as many 1s as 0s
d. Binary strings that start and end with the same symbol and have the same number of 0s as 1s.
e. $L = \{a^nb^m : m \geq n + 2\}$