Theory of Computation

Adapted By Manik Hosen

Transition Diagram

- Define transition diagram and transition table.
- Transition Diagram: Transition diagram is a special kind of flowchart for language analysis. In transition diagram the boxes of flowchart are drawn as circle and called as states. States are connected by arrows called as edges. The label or weight on edge indicates the input character that can appear after that state.
- Transition Table: The transition table is basically a tabular representation of the transition function. It takes two arguments: a state with input symbol and the next state.

Finite Automata

- Define & Classify Finite Automata (Finite State Machine).
- Finite Automata:
  - Finite automata are used to recognize patterns.
  - It takes the string of symbol as input and changes its state accordingly. When the desired symbol is found, then the transition occurs.
  - At the time of transition, the automata can either move to the next state or stay in the same state.
  - Finite automata have two states, Accept state or Reject state. When the input string is processed successfully, and the automata reached its final state, then it will accept.
- There are two types of finite automata:
  - DFA(deterministic finite automata)
  - NFA(non-deterministic finite automata)

- 1(a) Define alphabet, string and language. Discuss the basic operations of languages.
- Alphabets are defined as a finite set of symbols. Examples: $\sum = \{0, 1\}$ is an alphabet of binary digits.
- A string is a finite sequence of symbols selected from some alphabet. It is generally denoted as w. For example for alphabet $\sum = \{0, 1\}$ w = 010101 is a string.
- A language is a set of string all of which are chosen from some $\sum^*$, where $\sum$ is a particular alphabet. An example is English language, where the collection of legal English words is a set of strings over the alphabet that consists of all the letters. Another example is the C programming language where the alphabet is a subset of the ASCII characters and programs are subset of strings that can be formed from this alphabet.
- On languages we can define the usual set operations that is union, intersection and complement of languages.
  Moreover we have the following:
- Concatenation: If L1 and L2 are two languages, the concatenation of L1 and L2, denoted by L1L2, as the set of all words given by the concatenation of any word in L1 with any word in L2.L1={x|x=uv and u belongs to L1 and v belongs to L2}
- Kleen Closure: If S is a set of words then by Kleen Closure S* we mean the set of all finite strings formed by concatenating words from S, where any word may be used as often we like, and where the null string is also included. For example for $\sum = \{a\}$
  $\sum^* = \{\epsilon, a, aa, aaa, \ldots.\}$

Basic of TOC

Finite Automata

- 1(c) What are the ways by which a finite automata can be represented? Explain with example.
- Finite automata can be represented by input tape and finite control.
- Input tape: It is a linear tape having some number of cells. Each input symbol is placed in each cell.
- Finite control: The finite control decides the next state on receiving particular input from input tape. The tape reader reads the cells one by one from left to right, and at a time only one input symbol is read.
- Example:

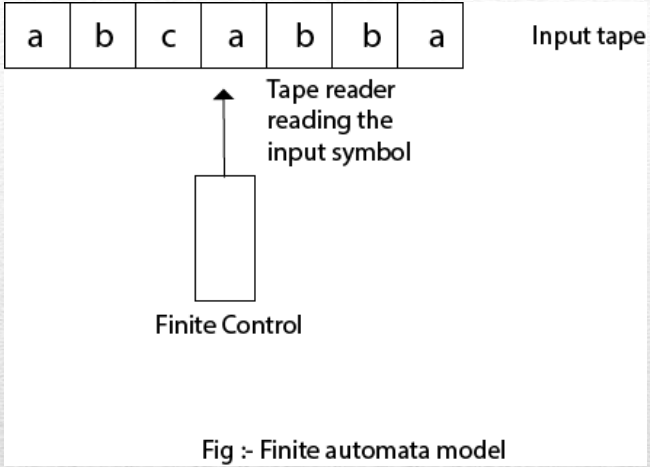


Fig :- Finite automata model

DFA

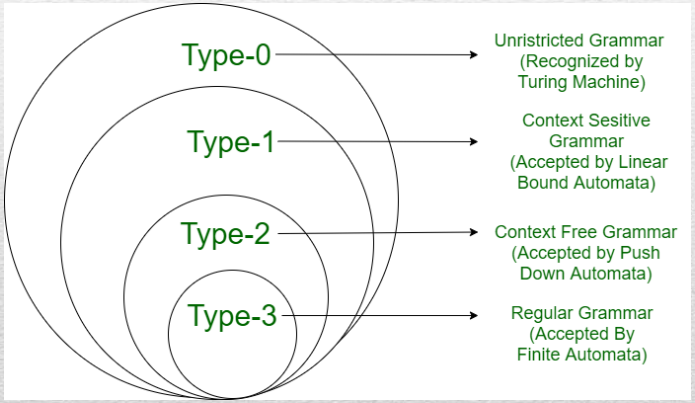
- Write down formal definition of a DFA.
- A DFA can be represented by a 5-tuple $(Q, \sum, \delta, q_0, F)$ where −
  - **Q** is a finite set of states.
  - $\sum$ is a finite set of symbols called the alphabet.
  - **δ** is the transition function where $\delta: Q \times \sum \rightarrow Q$
  - $\mathbf{q_0}$ is the initial state from where any input is processed ($q_0 \in Q$).
  - **F** is a set of final state/states of Q ($F \subseteq Q$).

Formal Language

- What do you mean by formal language?
- Formal language is a set of words ,i.e. finite strings of letters, symbols or tokens. In computer science they are used for the precise definition of data formats and the syntax of programming languages.

Formal Language

- Discuss the hierarchies of Chomsky's formal language.
- According to Chomsky hierarchy, grammars are divided of 4 types:
  - Type 0 known as unrestricted grammar.
  - Type 1 known as context sensitive grammar.
  - Type 2 known as context free grammar.
  - Type 3 Regular Grammar.



Formal Language

- Give the mathematical definition of formal grammar.
- A grammar G can be formally written as a 4-tuple (N, T, S, P) where –
  - N or V_N is a set of variables or non-terminal symbols.
  - T or $\sum$ is a set of Terminal symbols.
  - S is a special variable called the Start symbol, $S \in N$
  - P is Production rules for Terminals and Non-terminals. A production rule has the form $\alpha \rightarrow \beta$, where α and β are strings on $V_N \cup \sum$ and least one symbol of α belongs to V_N.

Formal Language

Explain Chomsky's classification of grammar in detail.

Type - 3 Grammar: Type-3 grammars generate regular languages. Type-3 grammars must have a single non-terminal on the left-hand side and a right-hand side consisting of a single terminal or single terminal followed by a single non-terminal.

The productions must be in the form $X \rightarrow a$ or $X \rightarrow aY$

where $X, Y \in N$ (Non terminal)

and $a \in T$ (Terminal)

The rule $S \rightarrow \varepsilon$ is allowed if S does not appear on the right side of any rule.

Example

$X \rightarrow \varepsilon$ $X \rightarrow a \mid aY$ $Y \rightarrow b$

Type - 2 Grammar: Type-2 grammars generate context-free languages.

The productions must be in the form $A \rightarrow \gamma$

where $A \in N$ (Non terminal)

and $\gamma \in (T \cup N)^*$ (String of terminals and non-terminals).

These languages generated by these grammars are be recognized by a non-deterministic pushdown automaton.

Example

$S \rightarrow X\ a$ $X \rightarrow a\ X \rightarrow aX\ X \rightarrow abc\ X \rightarrow \varepsilon$

Type - 1 Grammar: Type-1 grammars generate context-sensitive languages. The productions must be in the form

$\alpha\ A\ \beta \rightarrow \alpha\ \gamma\ \beta$

where $A \in N$ (Non-terminal)

and $\alpha, \beta, \gamma \in (T \cup N)^*$ (Strings of terminals and non-terminals)

The strings α and β may be empty, but γ must be non-empty.

The rule $S \rightarrow \varepsilon$ is allowed if S does not appear on the right side of any rule. The languages generated by these grammars are recognized by a linear bounded automaton.

Example

$AB \rightarrow AbBc\ A \rightarrow bcA\ B \rightarrow b$

Type - 0 Grammar: Type-0 grammars generate recursively enumerable languages. The productions have no restrictions. They are any phase structure grammar including all formal grammars.

They generate the languages that are recognized by a Turing machine.

The productions can be in the form of $\alpha \rightarrow \beta$ where α is a string of terminals and nonterminals with at least one non-terminal and α cannot be null. β is a string of terminals and non-terminals.

Example

$S \rightarrow ACaB\ Bc \rightarrow acB\ CB \rightarrow DB\ aD \rightarrow Db$

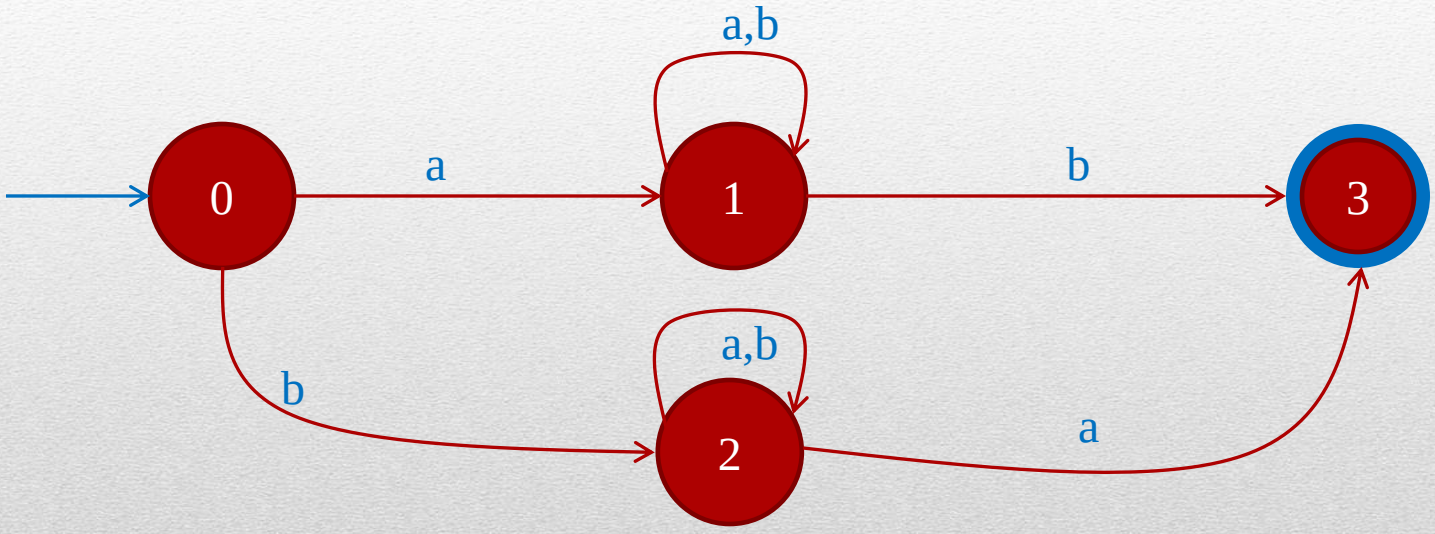
Grammar

- What is the function of grammar?
- Grammatical function is the syntactic role played by a word or phrase in the context of a particular sentence.
- How to get a string or word from a grammar?
- If there is a grammar

  $G: N = \{S, A, B\}\ T = \{a, b\}\ P = \{S \rightarrow AB, A \rightarrow a, B \rightarrow b\}$
- Here S produces AB, and we can replace A by a, and B by b. Here, the only accepted string is ab, i.e.,

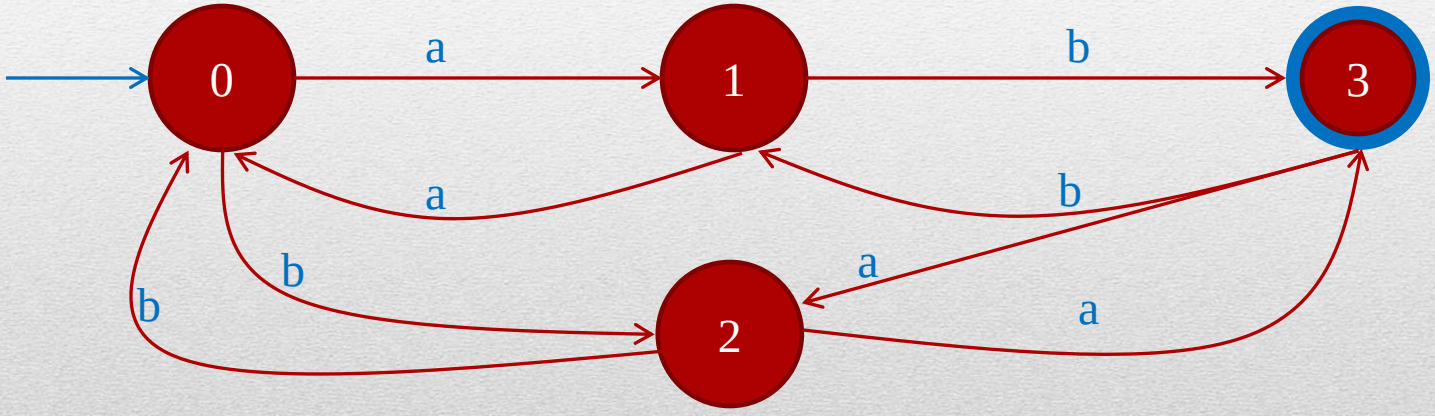
  $L(G) = \{ab\}$

- m-DFA accepting w∈{a, b}*|w starts and ends with different symbol:



D Finite Automata

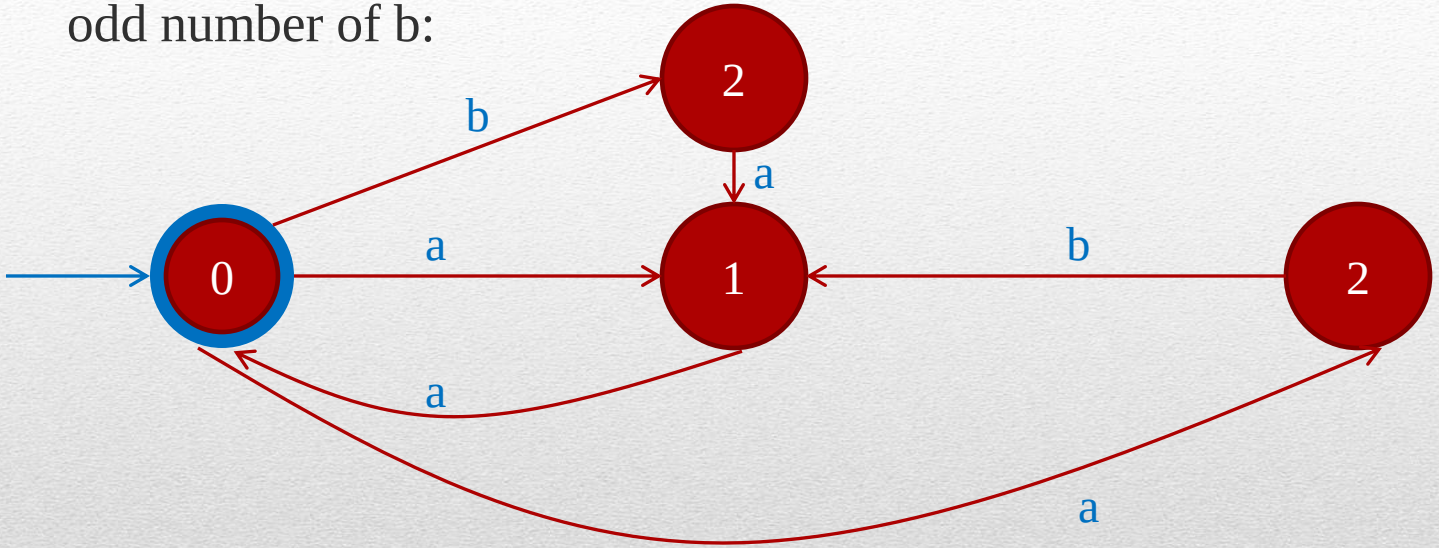
- m-DFA accepting w∈{a, b}*|w has odd number of a and odd number of b:



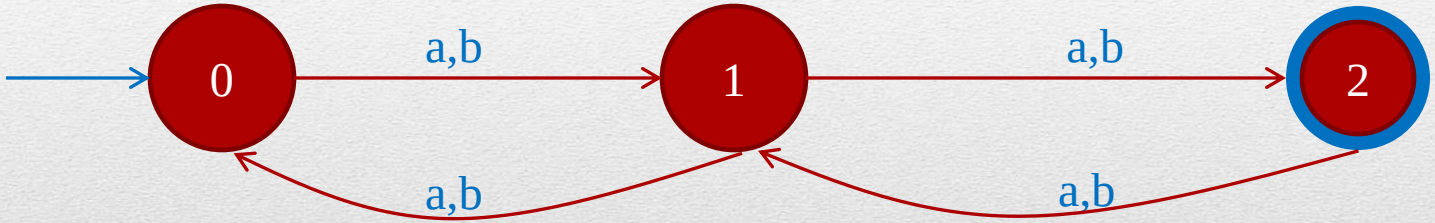
D Finite Automata

D Finite Automata

- 2019.1(c) m-DFA accepting w∈{a, b}*|w has even number of a and odd number of b:



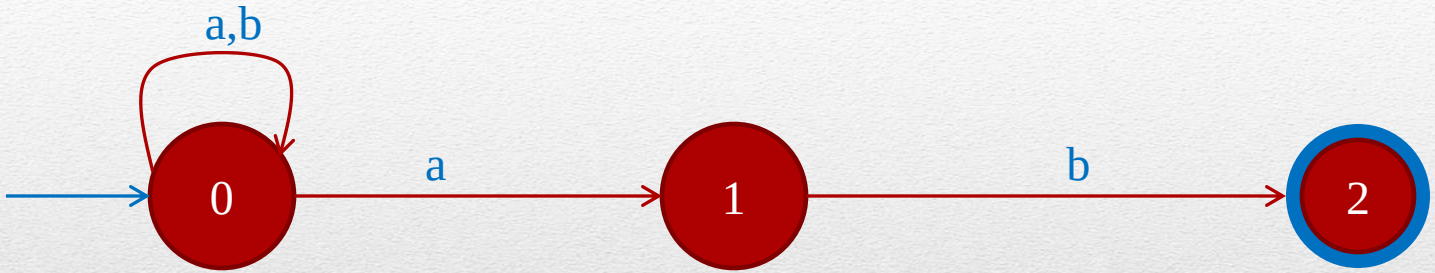
- 2019.1(c) m-DFA accepting w∈{a, b}*|w is a string of even length:



D Finite Automata

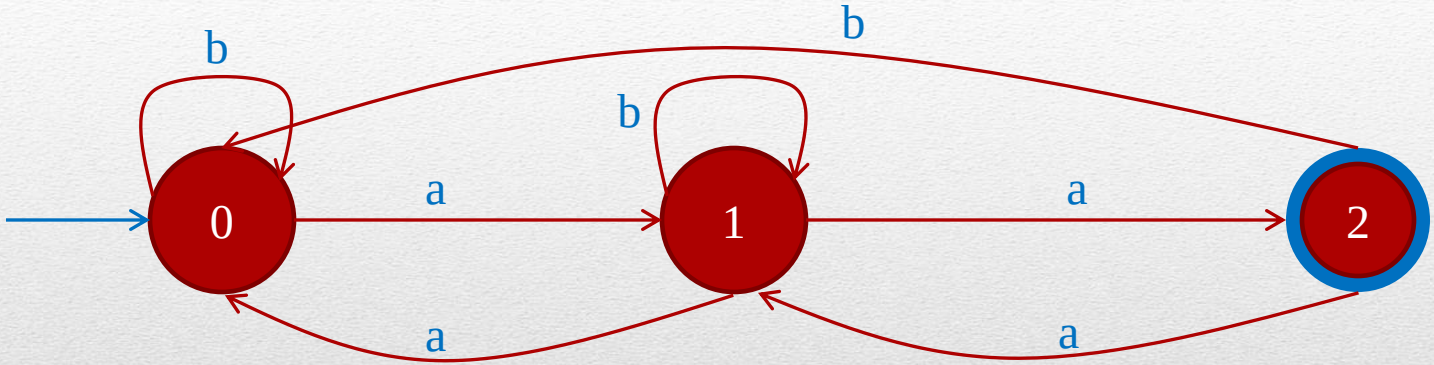
D Finite Automata

- 2019.1(c) m-DFA accepting w∈{a, b}*|w ends with ab:



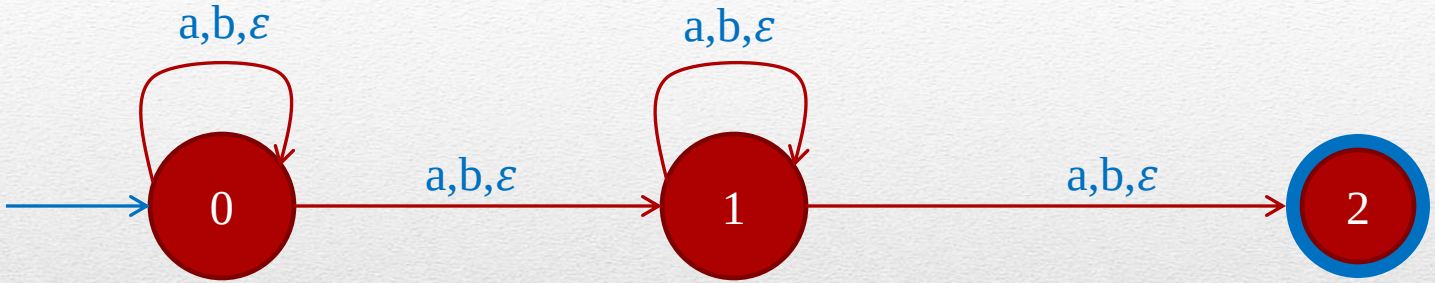
D Finite Automata

- 2018.2(a) m-DFA accepting w∈{a, b}*|w has even number of a:



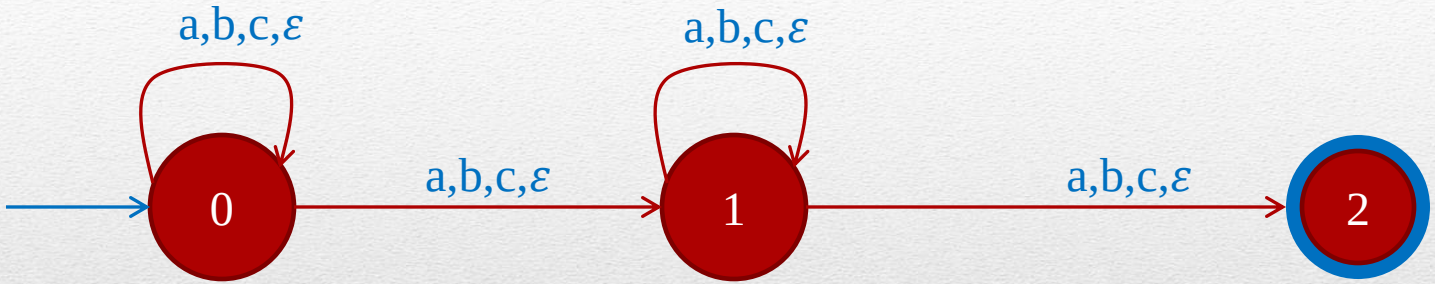
D Finite Automata

- 2018.2(a) m-DFA accepting $\{a^n b^m \mid m,n \geq 0\}$:



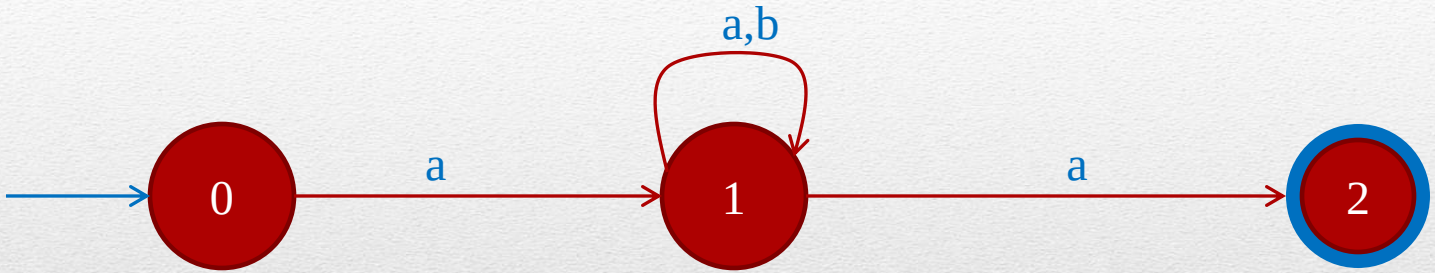
D Finite Automata

- 2017.2(b) m-DFA accepting $\{a^n b^m c^l | m,n,l \geq 0\}$:



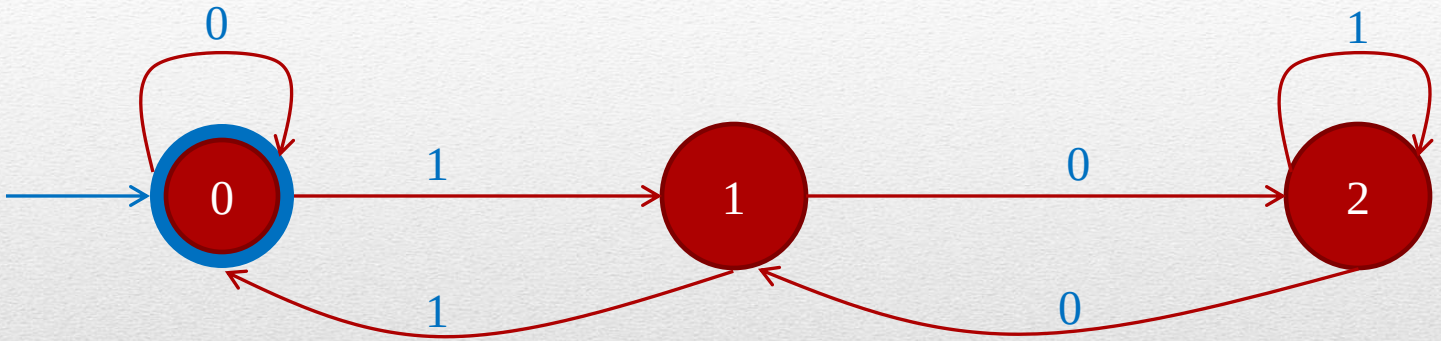
D Finite Automata

- 2017.2(b) m-DFA accepting w∈{a, b}*|w starts and ends with a:



D Finite Automata

- 2017.2(b) m-DFA accepting w∈{a, b}*|w is a binary number divisible by 3:

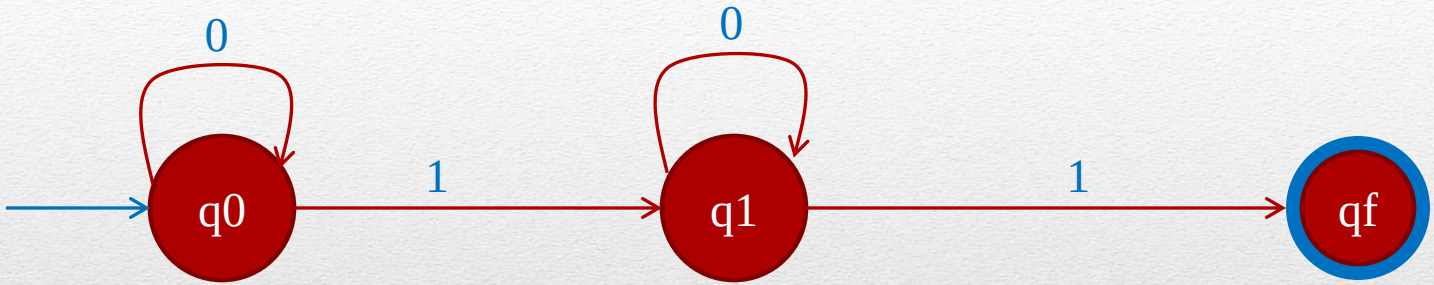


- Define Regular Expression.
- A regular expression is a sequence of characters that specifies a search pattern. Usually such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation.

Regular Expression

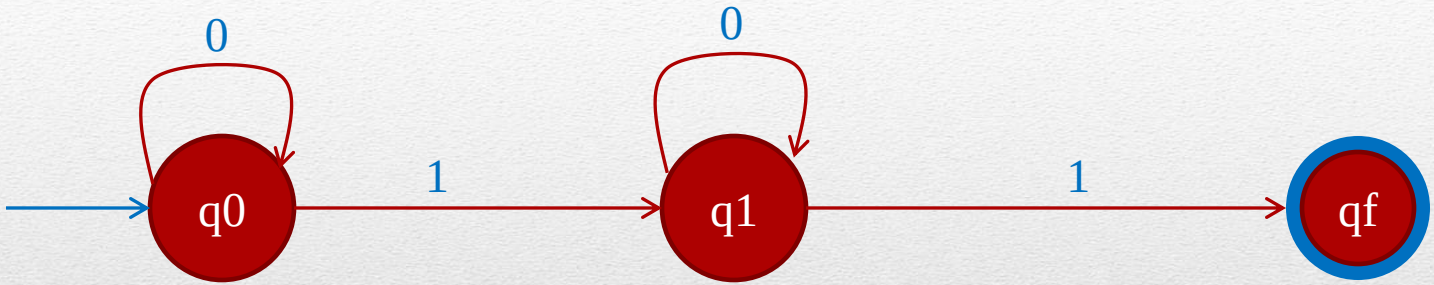
D Finite Automata

- 2019.2(a) DFA from regular expression 0*10*1

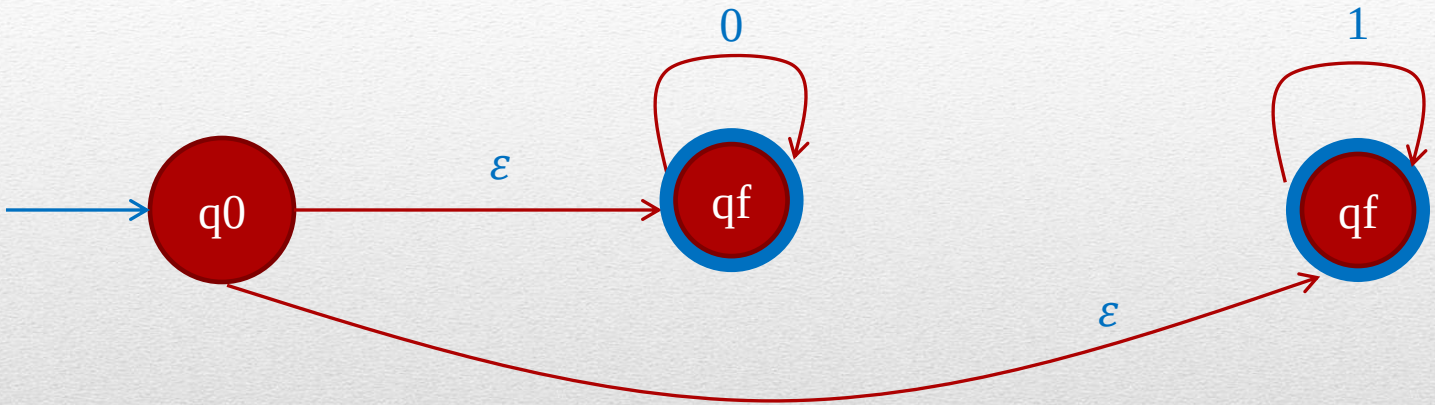


D Finite Automata

- 2019.2(a) DFA from regular expression 0*10*1



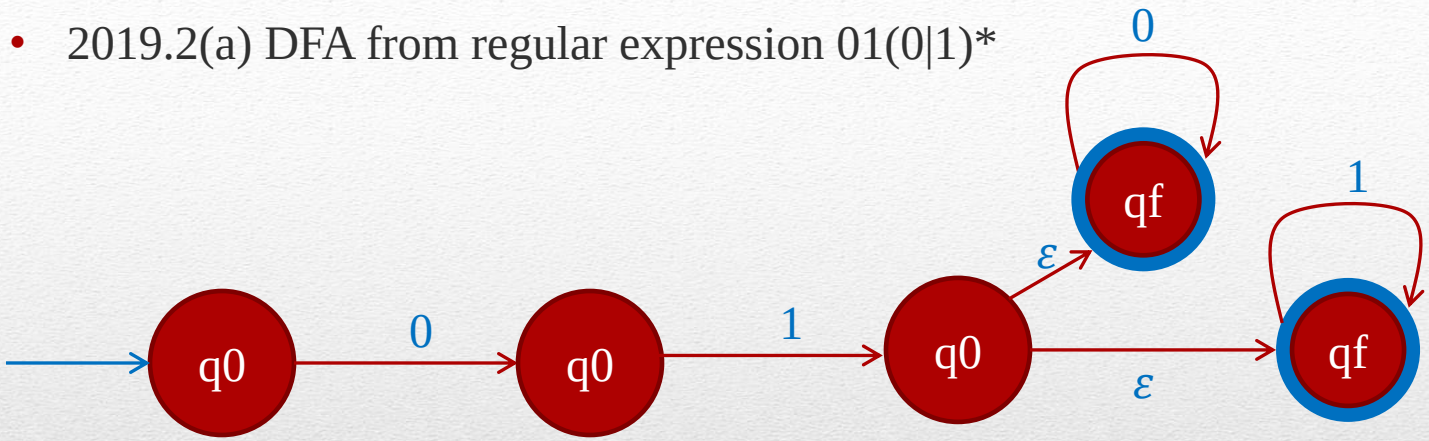
- 2019.2(a) DFA from regular expression (0|1)*



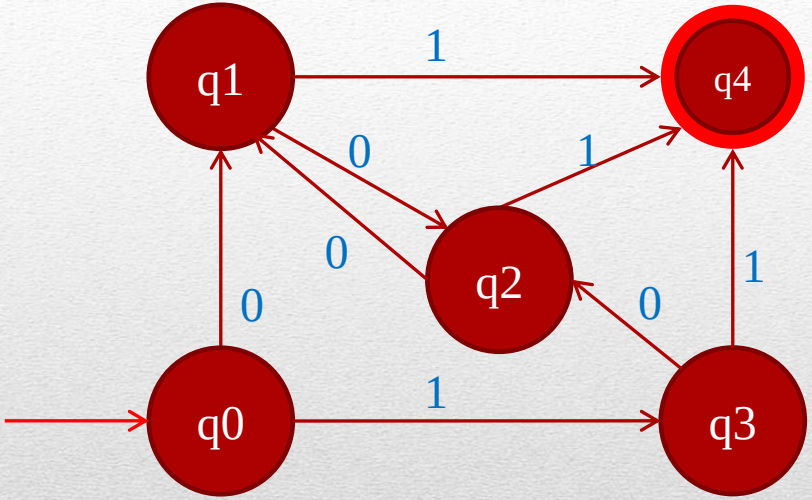
D Finite Automata

D Finite Automata

- 2019.2(a) DFA from regular expression 01(0|1)*



- 2019.2(b) Given,



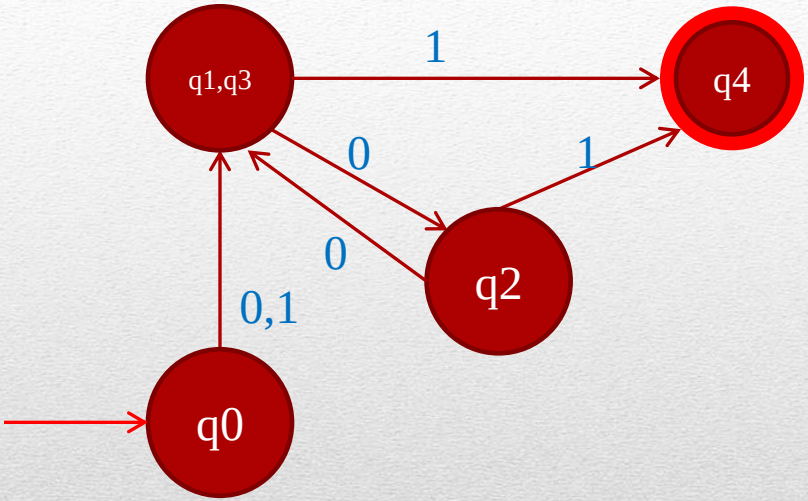
D Finite Automata

- Table:

	q0	q1	q2	q3	q4
q0					
q1	0		0		
q2		0		0	
q3	1				
q4		1	1	1	

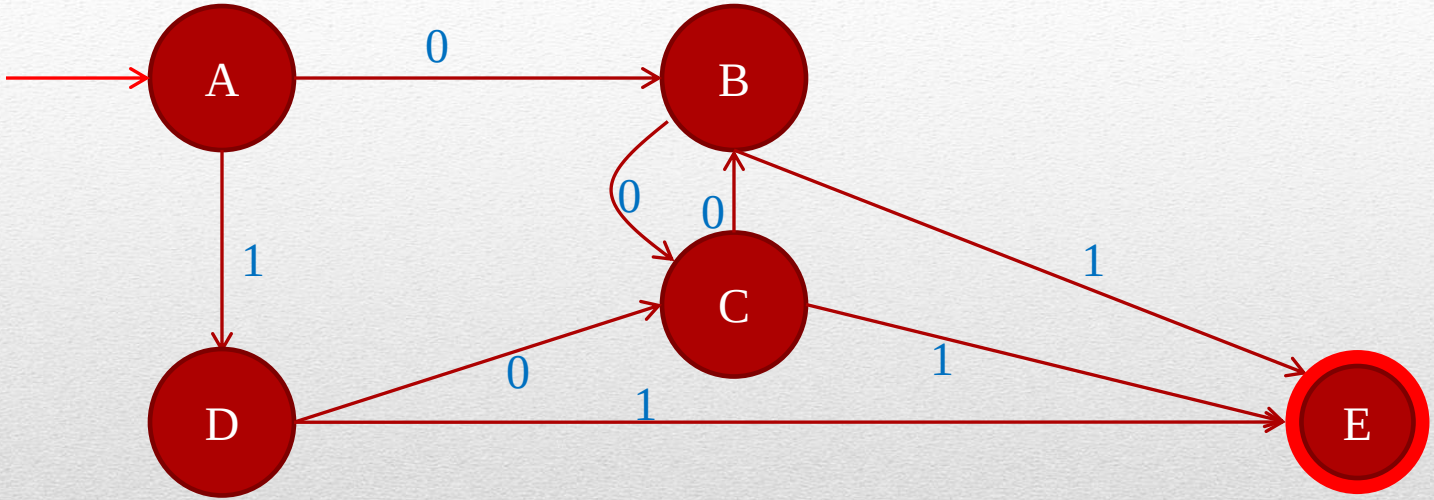
D Finite Automata

- Minimized,



D Finite Automata

- 2017.4(a) Given,



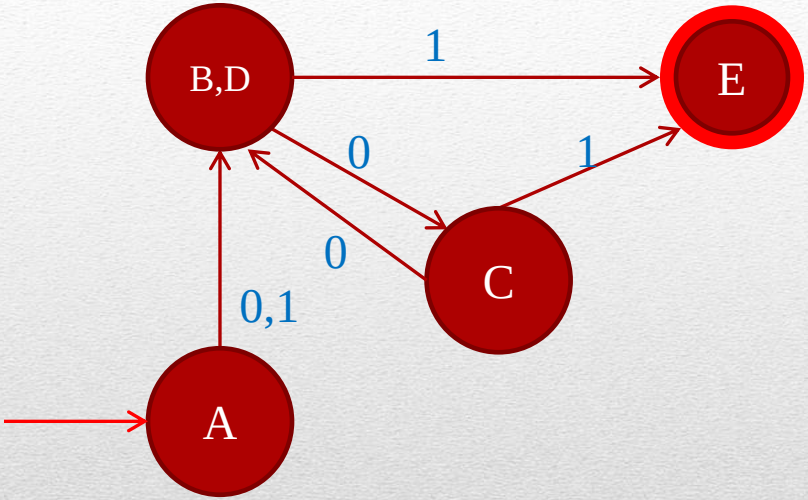
D Finite Automata

- Table:

	A	B	C	D	E
A					
B	0		0		
C		0		0	
D	1				
E		1	1	1	

D Finite Automata

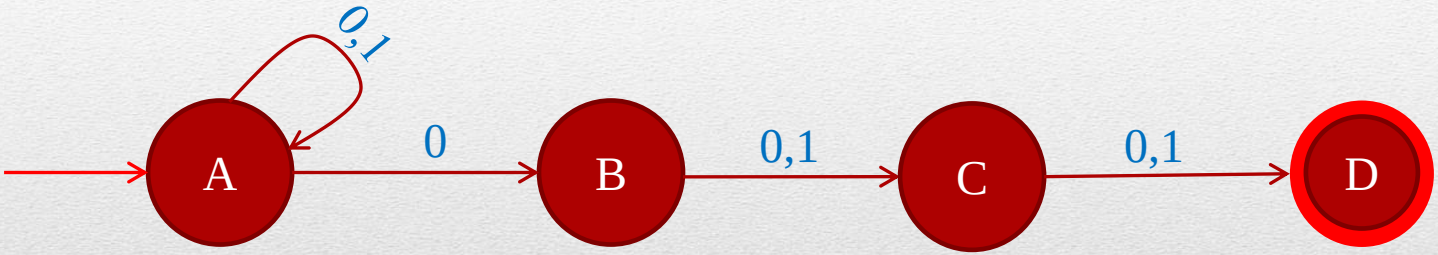
- Minimized,



D Finite Automata

Finite Automata

- 2019.3(a) Convert an NFA to DFA for the language containing ‘all strings in (0,1)* in which the 3rd symbol from the right is 0’.
- NFA:



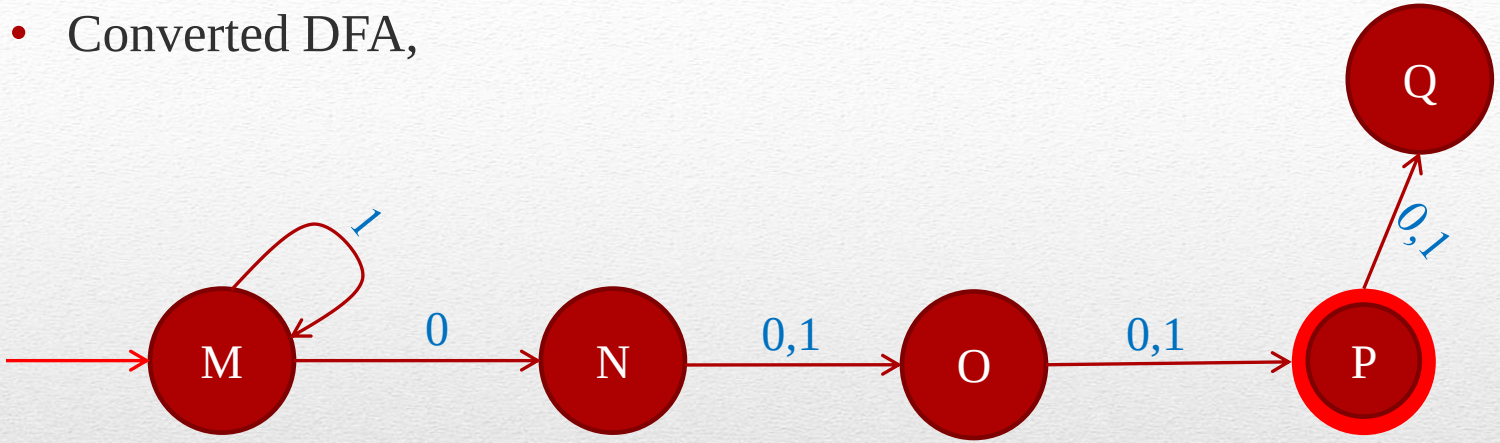
Finite Automata

- Table:

NFA State	DFA State	0	1
{A}	M	N	M
{A, B}	N	O	O
{A, C}	O	P	P
{A, D}	P	Q	Q

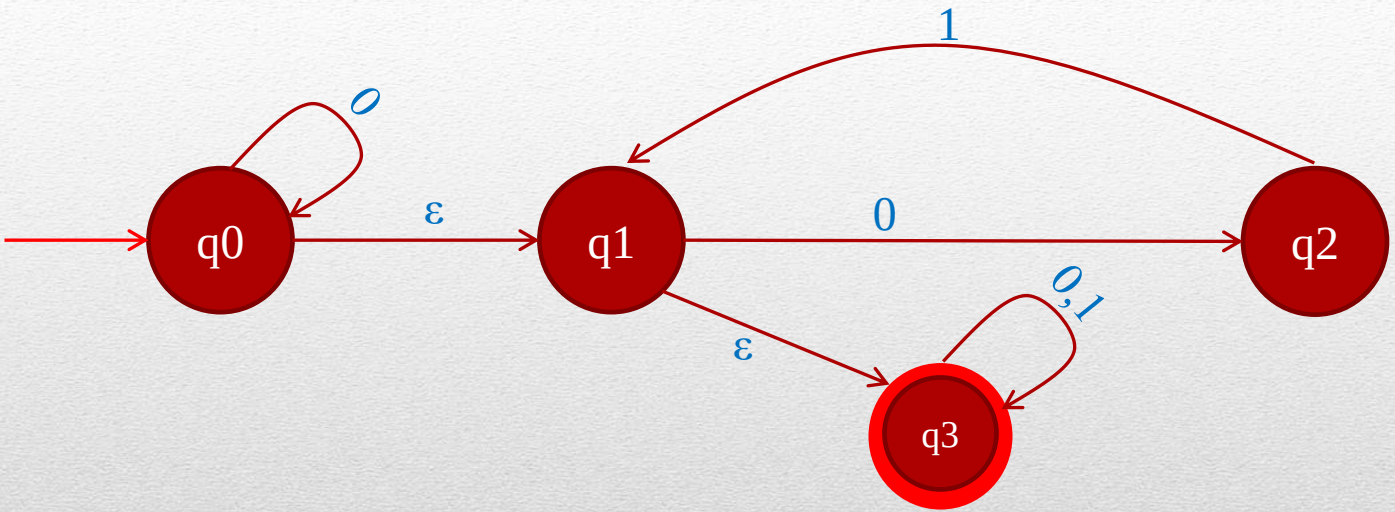
Finite Automata

- Converted DFA,

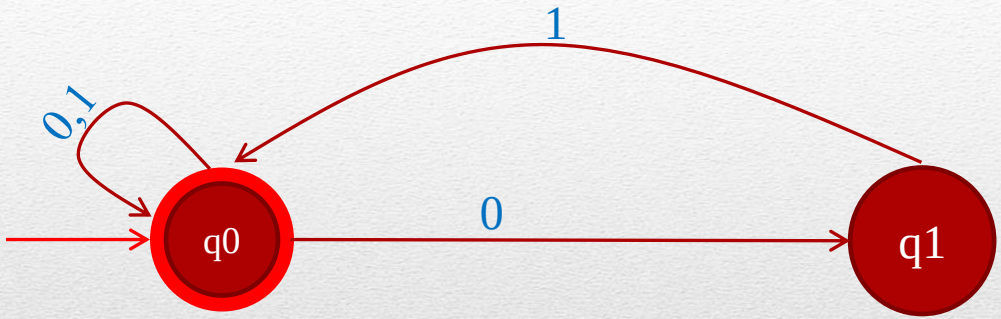


N Finite Automata

- 2019.3(b) Given ε-NFA,

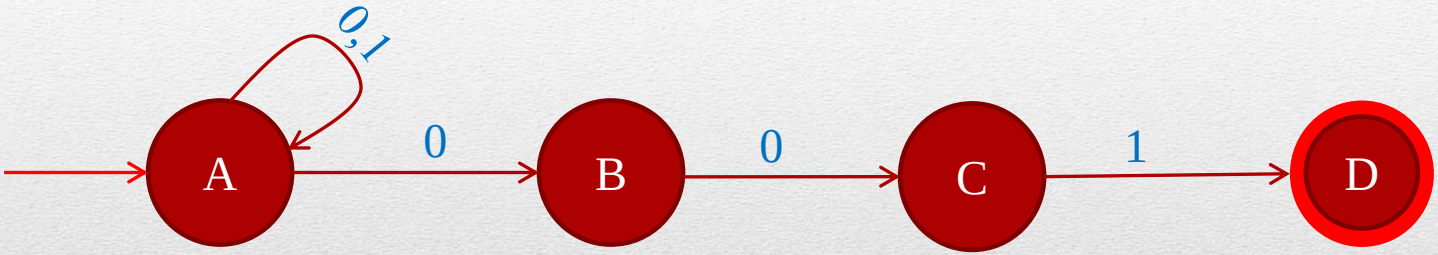


- Converted NFA,



N Finite Automata

- 2018.2(b) Construct an NFA that accepts a binary language ending with 001. Convert it to a DFA using subset construction method.



Finite Automata

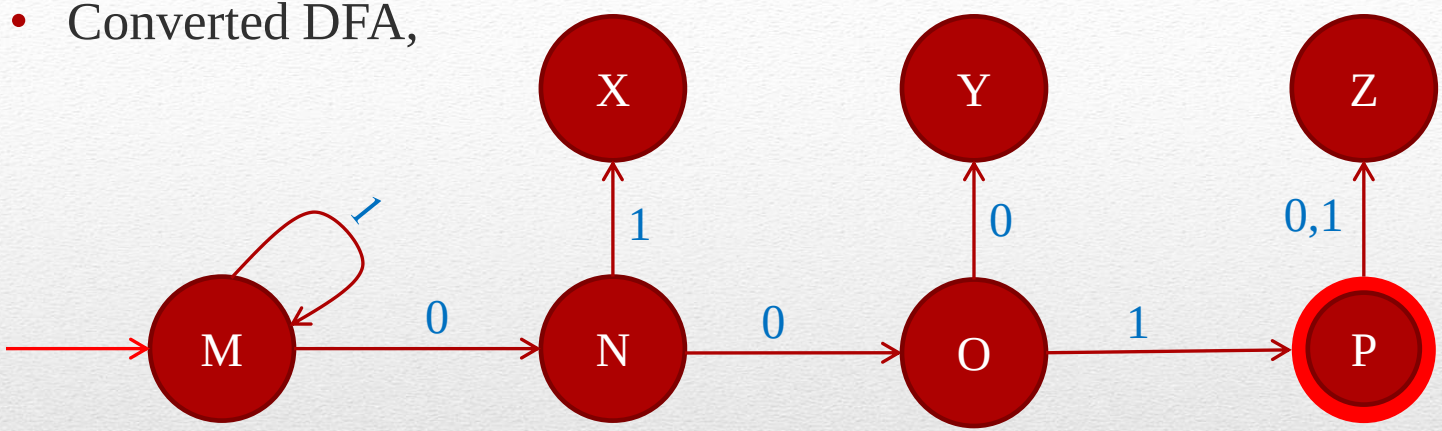
Finite Automata

- Table:

NFA State	DFA State	0	1
{A}	M	N	M
{A, B}	N	O	X
{A, C}	O	Y	P
{A, D}	P	Z	Z

Finite Automata

- Converted DFA,



- Can an FA accept any formal language? Explain why?
- Yes.
- You can define a formal language as a set of strings. a formal language can be expressed using a regular expression. So, any formal language is accepted by some FA.

Finite Automata

Language

- Why the language $L=\{a^n b^n | n \geq 1\}$ is not regular.
- A regular language must have a subset which is empty {}. But in this case the number of a or b cannot be 0. Hence there is not possible any empty subset. So, the language L is not regular.

Regular Expression

- Define Regular Expression.
- Regular expression is an important notation for specifying patterns. Each pattern matches a set of strings, so regular expressions serve as names for a set of strings.
- Write a regular expression for a language containing the strings starting and ending with same symbol over alphabet {a, b}.
- RE - a(a|b)*a|b(a|b)*b

Regular Expression

- 2018.3(b). RE from FA:
- (i) RE – b*ab*(ba*a)*
- (ii) RE – 0*(10*10*1)*0*

- 2018.3(c). What is the regular language represented by the following left-linear grammar?
- $A \rightarrow Aa|Ab|Ba, B \rightarrow Cb, C \rightarrow \varepsilon$
- $A \rightarrow Aa|Ab|Ba, B \rightarrow \varepsilon b$
- $A \rightarrow Aa|Ab|Ba, B \rightarrow b$
- $A \rightarrow Aa|Ab|ba$
- $A \rightarrow A(a|b)|ba$
- $A \rightarrow ba(a|b)*$
- $L(A) = \left(L\left(ba(a|b)\right)\right)^*$

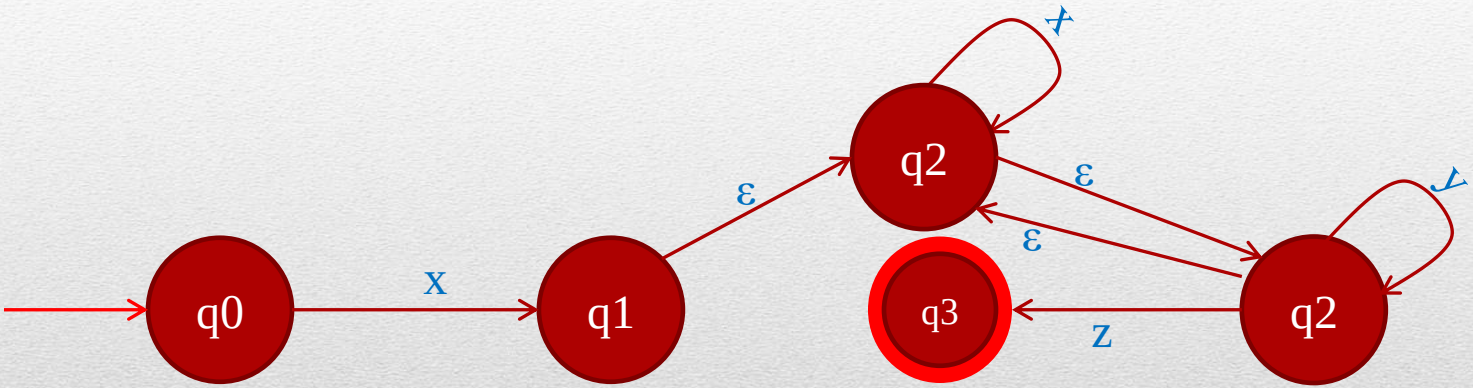
Regular Expression

Regular Expression

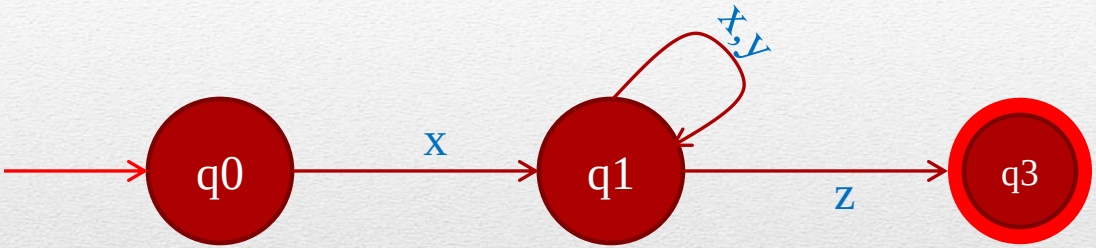
- Write a regular expression for a language containing:
- 17.3(b).(i) The strings has exactly two a’s over alphabet {a, b}.
- RE – b*ab*ab*
- 17.3(b).(ii) The strings starts and ends with same symbol over alphabet {a, b}.
- RE - a(a|b)*a|b(a|b)*b

Regular Expression

- Given a regular expression x(x|y)*z. Convert it to a DFA via ε-NFA.
- ε-NFA:



- DFA:



Regular Expression

State Machine

- Define Moore Machine and Mealy Machine.
- In the theory of computation, a Moore machine is a finite-state machine whose output values are determined only by its current state.
- In the theory of computation, a Moore machine is a finite-state machine whose output values are determined both by its current state and by the values of its inputs.

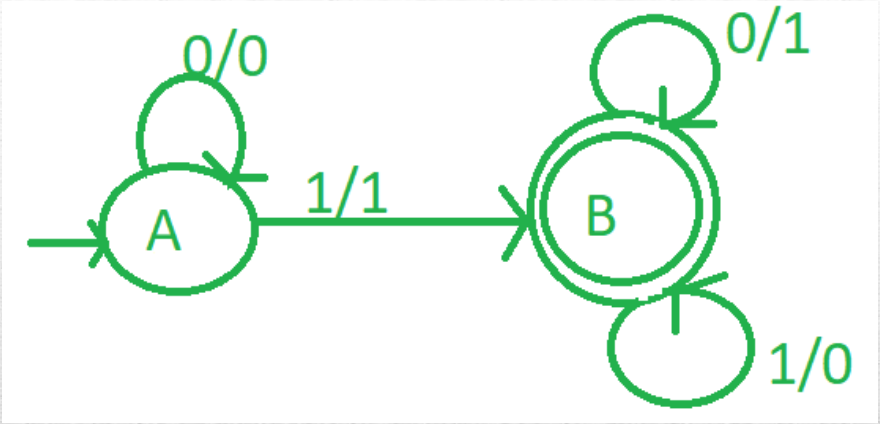
State Machines

- Contrast between Moore Machine and Mealy Machine.
- **Moore Machine –**
- Output depends only upon present state.
- If input changes, output does change.
- More number of states are required.
- There is less hardware requirement for circuit implementation.
- They react slower to inputs(One clock cycle later).
- Synchronous output and state generation.
- Output is placed on states.
- Easy to design.

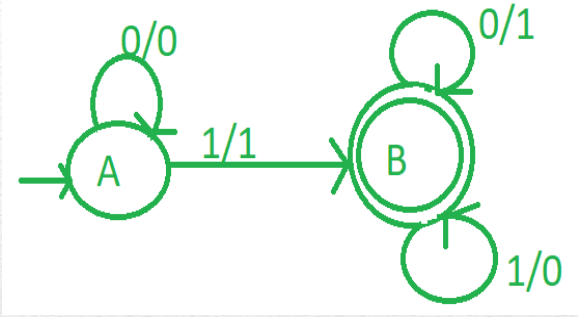
- **Mealy Machine –**
- Output depends on present state as well as present input.
- If input changes, output also changes.
- Less number of states are required.
- There is more hardware requirement for circuit implementation.
- They react faster to inputs.
- Asynchronous output generation.
- Output is placed on transitions.
- It is difficult to design.

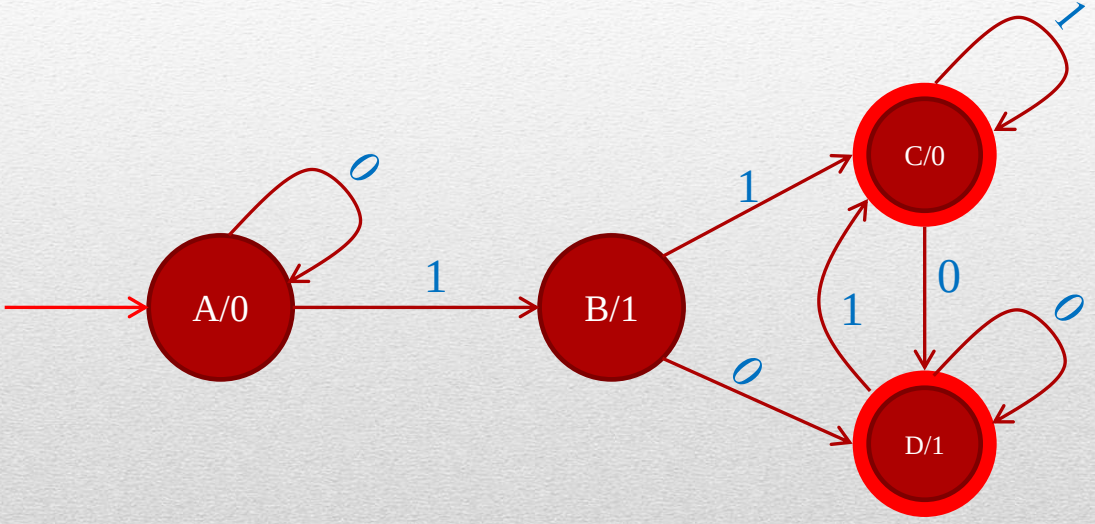
State Machines

- Construct a Mealy Machine that takes binary numbers as input and produces 2's complement of that numbers as output. Assume that the string is read LSB to MSB and end carry is discarded.
- Mealy Machine Design:
  - Take initial state A.
  - If there are n number of zeros at initial state, it will remain at initial state.
  - Whenever first input 1 is found then it gives output 1 and go to state B.
  - In state B, if input is zero, output will be 1.And if input is 1 then output will be 0.
  - And then set state B as final state.



- Convert the above Mealy Machine into Moore Machine.
- Moore Machine.

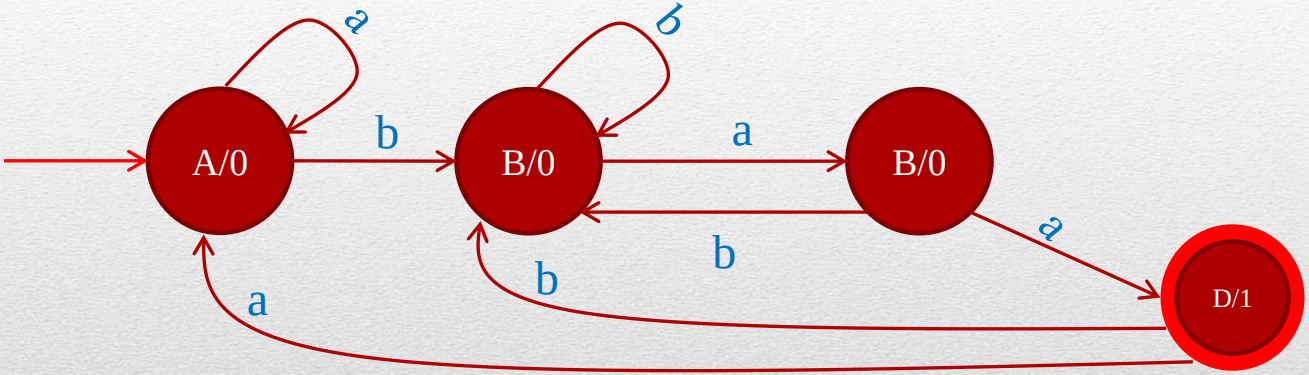




State Machines

State Machines

- Construct a Moore Machine that takes set of all strings over {a,b} as input and prints 1 as output for every occurrence of ‘baa’ as substring, otherwise print ‘0’. Using the machine produce an output string for the input string ‘abaaababaa’.



- Output String (Read from left to right): 0001000001

- 2017.4(c) See Question Paper.
- Ans: Yes.
- Because,
- $A \cup (N - A), where\ A\ is\ a\ subset\ of\ N$
- $= N$
- P is the set of two partition of N.
- T is also subset of N and T belongs to P.
- So, f is a bisection from T to P.

State Machines

- Write down the classification of problems. What are the tractable and intractable problems? Explain with examples.
- The type of computational problem: The most commonly used problems are decision problems, function problems, counting problems, optimization problems, promise problems etc.
- A problem that can be solved in theory (e.g. given large but finite resources, especially time), but for which in practice any solution takes too many resources to be useful, is known as an intractable problem.
- Conversely, a problem that can be solved in practice is called a tractable problem.

Context Free Grammar

- What do you mean by derivation? Define Sentence and Sentential form.
- A derivation of a string for a grammar is a sequence of grammar rule applications that transform the start symbol into the string.
- A sentence is a sentential form consisting only of terminals such as a + a * a.
- A sentential form is any string derivable from the start symbol. Thus, in the derivation of a + a * a , E + T * F and E + F * a and F + a * a are all sentential forms as are E and a + a * a themselves.

Context Free Grammar

Context Free Grammar

- 2019.5(a) Discuss two types of normal form.
- Chomsky's Normal Form (CNF)

CNF stands for Chomsky normal form. A CFG(context free grammar) is in CNF(Chomsky normal form) if all production rules satisfy one of the following conditions:

Start symbol generating ε. For example, A → ε.

A non-terminal generating two non-terminals. For example, S → AB.

A non-terminal generating a terminal. For example, S → a.

For example:

G1 = {S → AB, S → c, A → a, B → b}

G2 = {S → aA, A → a, B → c}

- Greibach Normal Form (GNF)

GNF stands for Greibach normal form. A CFG(context free grammar) is in GNF(Greibach normal form) if all the production rules satisfy one of the following conditions:

A start symbol generating ε. For example, S → ε.

A non-terminal generating a terminal. For example, A → a.

A non-terminal generating a terminal which is followed by any number of non-terminals. For example, S → aASB.

For example:

G1 = {S → aAB | aB, A → aA| a, B → bB | b}

G2 = {S → aAB | aB, A → aA | ε, B → bB | ε}

Context Free Grammar

- 2019.5(b) Consider the following grammar $S \to AB, A \to a, B \to C|b, C \to D, D \to E, E \to a$. Eliminate until productions and get an equivalent grammar.
- $S \to AB, A \to a, B \to C|b, C \to D, D \to E, E \to a$
- $S \to AB, A \to a, B \to C|b, C \to D, D \to a$
- $S \to AB, A \to a, B \to C|b, C \to a$
- $S \to AB, A \to a, B \to a|b$
- $S \to a(a|b)$
- $S \to aa|ab$

- {w∈{a,b}*|w starts and ends with same symbol.
- CFG: $S \rightarrow aS'a|bS'b, S' \rightarrow S'aS'|S'bS'|\varepsilon$
- {w∈{a,b}*|w starts and ends with same symbol.
- CFG: $S \rightarrow aS'b|bS'a, S' \rightarrow S'aS'|S'bS'|\varepsilon$
- {w∈{a,b}*|w has even length.
- CFG: $S \rightarrow aSb|bSa|aSa|bSb|aa|bb|ab|ba$
- {w∈{a,b}*|w is palindrome.
- CFG: $S \rightarrow aSa|bSb|bS'b|aS'a, S' \rightarrow a|b|\varepsilon$
- {w∈{$a^{2n}b^n$}, n≥ 1}
- CFG: $S \rightarrow aaSb|aab$
- {w∈{$a^nb^mc^n$}, m,n≥ 1}
- CFG: $S \rightarrow aSc|aS'c|abc, S' \rightarrow S'b|b$

Context Free Grammar

Context Free Grammar

- CFG to CNF:
- $S \rightarrow XaX|bX|Y, X \rightarrow XaX|XbX|\varepsilon, Y \rightarrow ab$
- $S \rightarrow a|b|XaX|bX|Y, X \rightarrow XaX|XbX, Y \rightarrow ab$
- $S \rightarrow XPX|QX|PQ|a|b, X \rightarrow XPX|XQX, Y \rightarrow PQ, P \rightarrow a, Q \rightarrow b$
- $S \rightarrow RX|QX|PQ|a|b, X \rightarrow RX|XT, Y \rightarrow PQ, P \rightarrow a, Q \rightarrow b, R \rightarrow XP, T \rightarrow QX$
- CFG to CNF:
- $S \rightarrow aX|Yb, X \rightarrow S|\varepsilon, Y \rightarrow bY|b$
- $S \rightarrow aS|a|Yb, Y \rightarrow bY|b$
- $S \rightarrow MS|YN|a, Y \rightarrow NY|b, M \rightarrow a, N \rightarrow b$

Context Free Grammar

- CFG to CNF:
- $S \rightarrow aX|bS|ab|X, X \rightarrow aX|a|\varepsilon$
- $S \rightarrow a|aX|bS|ab|X, X \rightarrow aX|a$
- $S \rightarrow a|aX|bS|ab, X \rightarrow aX|a$
- $S \rightarrow a|PX|QS|PQ, X \rightarrow PX|a, P \rightarrow a, Q \rightarrow b$

The End