FORMAL LANGUAGES AND AUTOMATA THEORY

Module 1

Rijin IK

Assistant Professor
Department of Computer Science and Engineering
Vimal Jyothi Engineering College
Chemperi

May 12, 2024

Outline

Course Outcomes

After the completion of the course the student will be able to

1. Classify a given formal language into Regular, Context-Free, Context Sensitive, Recursive or Recursively Enumerable. [Cognitive knowledge level: Understand]
2. Explain a formal representation of a given regular language as a finite state automaton, regular grammar, regular expression and Myhill-Nerode relation. [Cognitive knowledge level: Understand]
3. Design a Pushdown Automaton and a Context-Free Grammar for a given context-free language. [Cognitive knowledge level : Apply]
4. Design Turing machines as language acceptors or transducers. [Cognitive knowledge level: Apply]
5. Explain the notion of decidability. [Cognitive knowledge level: Understand]

Symbols

- Symbols are indivisible objects or entity that cannot be defined
- Symbols are the attoms of the world of language
- A symbol is any single objects such as Δ,a,0,1,# etc

Alphabets

An alphabet is a finite, nonempty set of symbols. It is denoted by Σ

Example:

- $\Sigma = \{0, 1\}$ is binary alphabet consisting of the symbols 0 and 1.
- $\Sigma = \{a, b, c...z\}$ is lowercase English alphabet.

String

A string (or word) is a finite sequence of symbols chosen from some alphabet.

Example:

- If $\Sigma = \{a, b, c\}$ then **abcbb** is a string formed from that alphabet.

Length of a string

- The length of a string w, denoted $|w|$, is the number of symbols composing the string.
- **Example:**
  - The string abcb has length 4.

Empty string (ϵ):

- The empty string denoted by ϵ, is the string consisting of zero symbols. Thus $|\epsilon|$ =0.

Operations on Strings

1. Concatenation of strings
2. String Reversal
3. Substring

Concatenation of strings

- The concatenation of two strings is the string formed by writing the first, followed by the second, with no intervening space.
- Concatenation of strings is denoted by .(dot)
- That is, if w and x are strings, then wx is the concatenation of these two strings.
- **Example:**
  - The concatenation of dog and house is doghouse.
  - Let x=0100101 and y= 1111 then x . y=01001011111

String Reversal

- Reversing a string means writing the string backwards.
- It is denoted by w^R
- **Example:**
  - Reverse of the string **abcd** is **dcba**.

Note: If $w = w^R$, then that string is called palindrome.

Substring

- A substring is a part of a string.
- **Example:**
  - If abcd is string then possible substrings are ϵ,a,b,c,d,ab,bc,cd,abc,bcd are proper substrings for the given string
- A **prefix** of a string is any number of leading symbols of that string.
- A **suffix** of a string is any number of trailing symbols.
- **Example:**
  - String abc has prefixes ϵ, a, ab, and abc;
  - its suffixes are ϵ, c, bc, and abc.
- A prefix or suffix of a string, other than the string itself, is called a proper prefix or suffix.

Language

- A (formal) language is a set of strings of symbols from someone alphabet.
- It is denoted by L. We denote this language by Σ^*.
  - The empty set, ϕ, and the set consisting of the empty string $\{\epsilon\}$ are languages.

Example:

- If $\Sigma = \{a\}$, then $\Sigma^* = \{\epsilon, a, aa, aaa, ...\}$.
- If $\Sigma = \{0, 1\}$, then $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, ...\}$.

Operations on languages

1. Union
2. Intersection
3. Complementation
4. Concatenation
5. Reversal
6. Kleene Closure
7. Positive Closure

Operations on languages

Union

- If L1 and L2 are two languages over an alphabet Σ.Then the union of L1 and L2 is denoted by $L1 \cup L2$
- **Example**
  - L1={0,01,011} and L2={001}, then $L1 \cup L2$={0,01,011,001}

Intersection

- If L1 and L2 are two languages over an alphabet Σ.Then the intersection of L1 and L2 is denoted by by $L1 \cap L2$
- **Example**
  - L1={0,01,011} and L2={01}, then $L1 \cap L2$={01}

Operations on languages

Complementation

- If L is a language over an alphabet Σ.then the complement of L denoted by L' is the language consisting of those strings that are not in L over the alphabet.
- **Example**
  - If Σ= {a,b}, and L = {a,b,aa}. then
  - $L' = \Sigma^* - L = \{\epsilon, , a, b, aa, bb, ab...\} - \{a, b, aa\} = \{\epsilon, bb, ab, ba....\}$.

Concatenation

- Concatenation of two languages L1 and L2 is the language L1 . L2 ,each element of which is a string formed by combining one string of L1 with another string of L2.
- **Example**
  - $L1 = \{bc, bcc, cc\}$ *and* $L2 = \{cc, ccc\}$, then
  - $L1.L2 = \{bccc, bcccc, bccccccc, cccc, ccccc\}$

Operations on languages

Reversal

- If L is language, then L^R is obtained by reversing the corresponding string in L. This operation is similar to the reversal of a string.

$$L^R = \{w^R | w \in L\}$$

- **Example**
  - *If* $L = \{0, 011, 0111\}$, *then* $L^R = \{0, 110, 1110\}$

Operations on languages

Kleene Closure

- The Kleene closure (or just closure) of L, denoted L*, is the set

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

$L^* = L^0 \cup L^1 \cup L^2 \cup ....$

- **Example**
  - Let $L1 = \{10, 1\}$
  - $L^* = L^0 \cup L^1 \cup L^2 ..... = \{\epsilon, 1, 10, 11, 111, 1111, ...\}$

Operations on languages

Positive Closure

- The positive closure of L, denoted L^+, is the set

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

$L^+ = L^1 \cup L^2 \cup L^3 \cup ....$

- **Example**
  - Let $L1 = \{10, 1\}$
  - $L^+ = L^1 \cup L^2 \cup L^3..... = \{1, 10, 11, 111, 1111, ...\}$

Finite Automata

Finite Automata or Finite Automaton

- Mathematical model of finite state system

Finite state system

- A system containing only finite number of states and transition among them is called finite state system(FSS)

State

- State of a system is an instantaneous description of that system.
- Which gives all relevant information necessary to determine how the system can evolve from that point on

Transition

- Change of states that occur spontaneously or in response to inputs to the states

Finite Automata Model

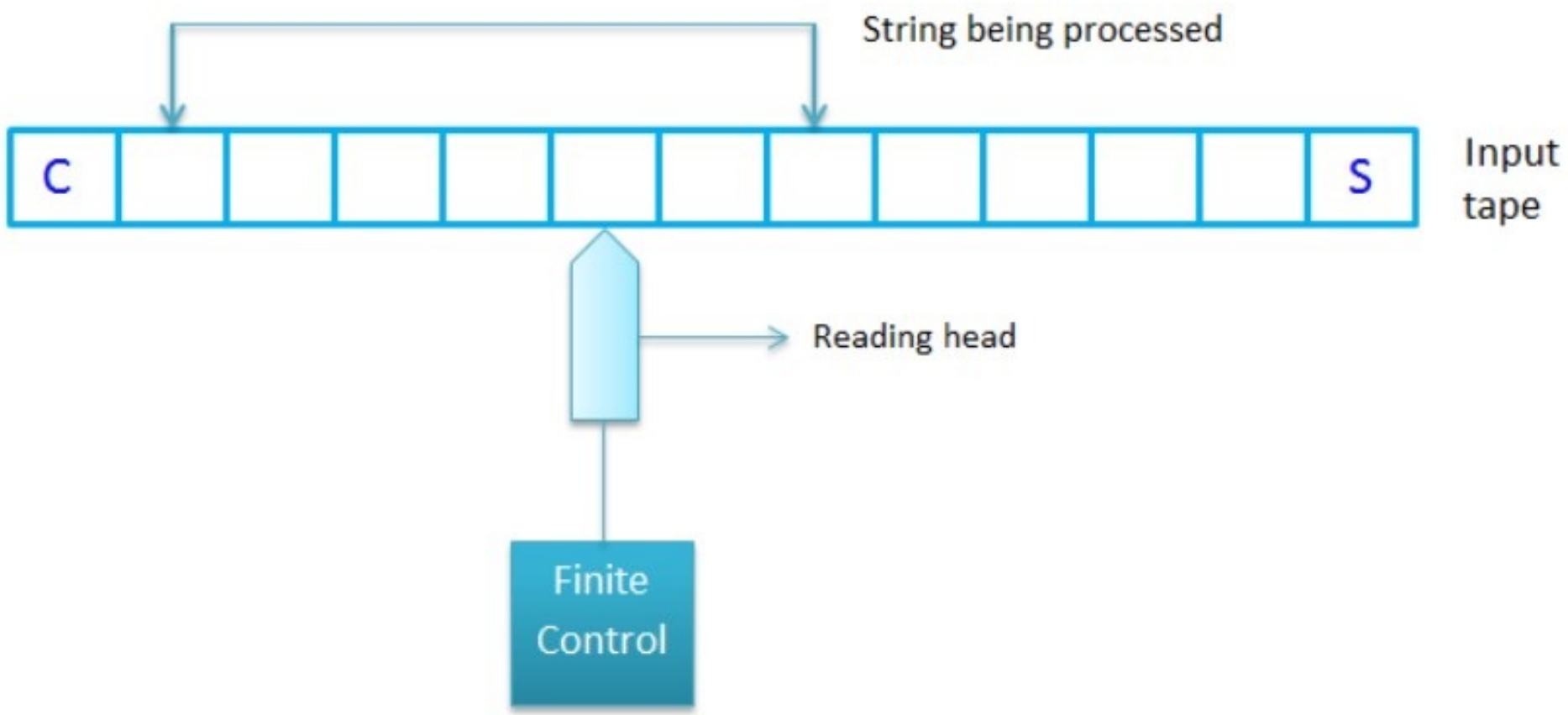


Figure: Block diagram of a finite automaton

Finite Automata

Input tape:

- The input tape is divided into squares, each square containing a single symbol from the input alphabet Σ
- The end squares of the tape contain the endmarker
- The left-to-right sequence of symbols between the two endmarkers is the input string to be processed.

Reading head:

- The head examines only one square at a time and can move one square either to the left or to the right.
- For further analysis, we restrict the movement of the R-head only to the right side.
- Initially the head is placed at the leftmost square of the tape.

Finite control:

- The input to the finite control will usually be the symbol under the R-head, say a, and the present state of the machine, say q, to give the following outputs
  - A motion of the R-head along the tape to the next square
  - The next state of the finite state machine given by $\delta(q, a)$.

Acceptance of String by a Finite Automaton

- The FA accepts a string x if the sequence of transitions corresponding to the symbols of x leads from the start state to an accepting state(final state) and the entire string has to be consumed

Finite State Automata-Formal definition

- A finite automaton (FA) is a 5-tuple or quintuple

$$M = (Q, \Sigma, \delta, q_0, F)$$

- Q is a finite set of states
- Σ is a finite input alphabet
- δ is the transition function
- $q_0 \in Q$ is the initial state
- F $\subseteq$ Q is the set of final states

Transition Diagram:

- A transition diagram is a directed graph associated with an FA in which the vertices of the graph correspond to the states of the FA.
- If there is a transition from state q to state p on input a, then there is an arc labelled a from state q to state p in the transition diagram.
  - State is denoted by ○
  - Transition is denoted by →
  - Initial state is denoted by →○
  - Final state is denoted by ◎

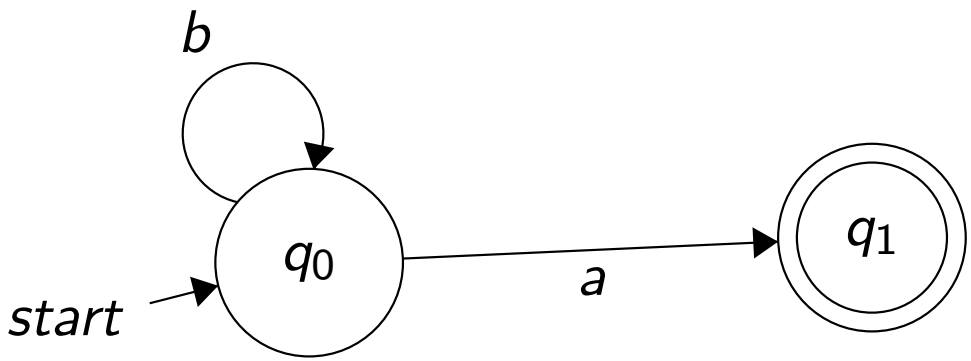


Figure: Transition digram

Transition Table:

- A tabular representation in which rows correspond to states, columns correspond to inputs and entries correspond to next states.
- The start state is denoted by an arrow with no source.
- The accept state is denoted by a star.

State\input	a	b
→q0	q1	q0
*q1	-	-

Table: Transition Table

Finite Automata

There are two types of finite automata

1. Deterministic Finite Automata (DFA)
2. Non-Deterministic Finite Automata (NFA)

Deterministic Finite Automata (DFA)

- It is a finite state machine that reads input symbols from an alphabet one by one and transitions between states based on a fixed set of rules.
- The transitions are deterministic, meaning that for a given input symbol and current state, there is only one next state.
- DFAs are capable of recognizing and accepting certain types of languages, known as regular languages.

Deterministic Finite Automata (DFA)

- Formally, a deterministic finite automaton can be represented by a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, F)$$

  - Q is a finite set of states
  - Σ is a finite input alphabet
  - δ is the transition function mapping $Q \times \Sigma$ *to* Q i.e.,$\delta(q, a)$ is a state for each state q and input symbol a.
  - $q_0 \in Q$ is the initial state
  - F $\subseteq$ Q is the set of final states.It is assumed here that there may be more than one final state.

Deterministic Finite Automata (DFA)

Steps to design a DFA

1. Understand the language for which the DFA has to be designed and write the language for the set of strings starting with minimum string that are accepted by FA.
2. Draw transition diagram for the minimum length string.
3. Obtain the possible transitions to be made for each state on each input symbol.
4. Draw the transition table.
5. Test DFA with few strings that are accepted and few strings that are rejected by the given language.
6. Represent DFA with tuples.

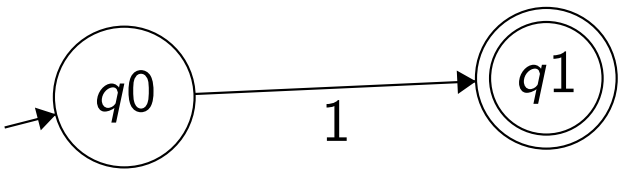
Deterministic Finite Automata (DFA)

Q:Design DFA that accepts all strings which starts with '1' over the alphabet {0,1}

1. Understand the language for which the DFA has to be designed and write the language for the set of strings starting with minimum string that are accepted by FA.

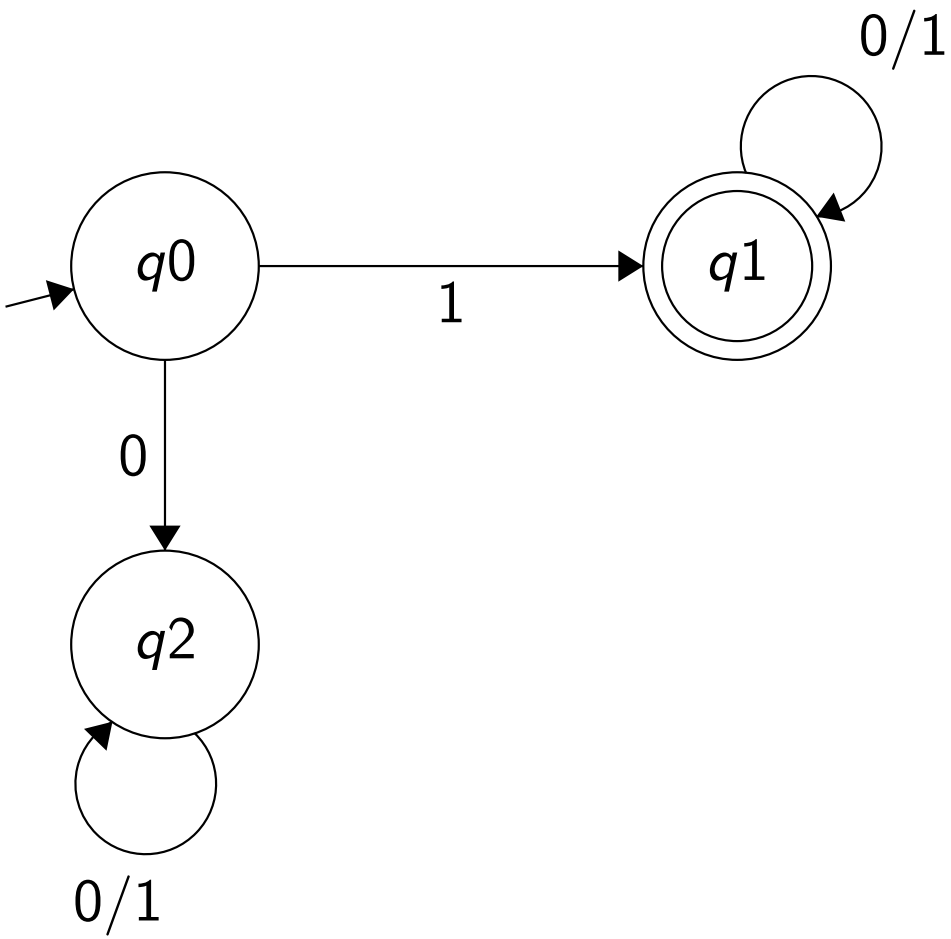
$$L = \{1, 10, 11, 100, 110, 101, 111, ...\}$$

2. Draw transition diagram for the minimum length string.



Deterministic Finite Automata (DFA)

3 Obtain the possible transitions to be made for each state on each input symbol.



Deterministic Finite Automata (DFA)

4 Draw the transition table.

	0	1
→q0	q2	q1
*q1	q1	q1
q2	q2	q2

- Test DFA with few strings that are accepted and few strings that are rejected by the given language.
- Represent DFA with tuples.

Deterministic Finite Automata (DFA)

5 Test DFA with few strings that are accepted and few strings that are rejected by the given language.

- **Case i)** Let w=1010 $\in$ L

$$\begin{aligned} \delta(\text{q0,1010}) &= \delta(\text{q1,010}) \\ &= \delta(\text{q1,10}) \\ &= \delta(\text{q1,0}) \\ &= \text{q1} \end{aligned}$$

q1 is final state and the entire string has been consumed i.e., given string is accepted by DFA.

- **Case ii)** Let w=0001 $\notin$ L

$$\begin{aligned} \delta(\text{q0,0001}) &= \delta(\text{q2,001}) \\ &= \delta(\text{q2,01}) \\ &= \delta(\text{q2,1}) \\ &= \text{q2} \end{aligned}$$

q2 is not final state and the entire string has been consumed i.e., given string is rejected by DFA

- Represent DFA with tuples.

Deterministic Finite Automata (DFA)

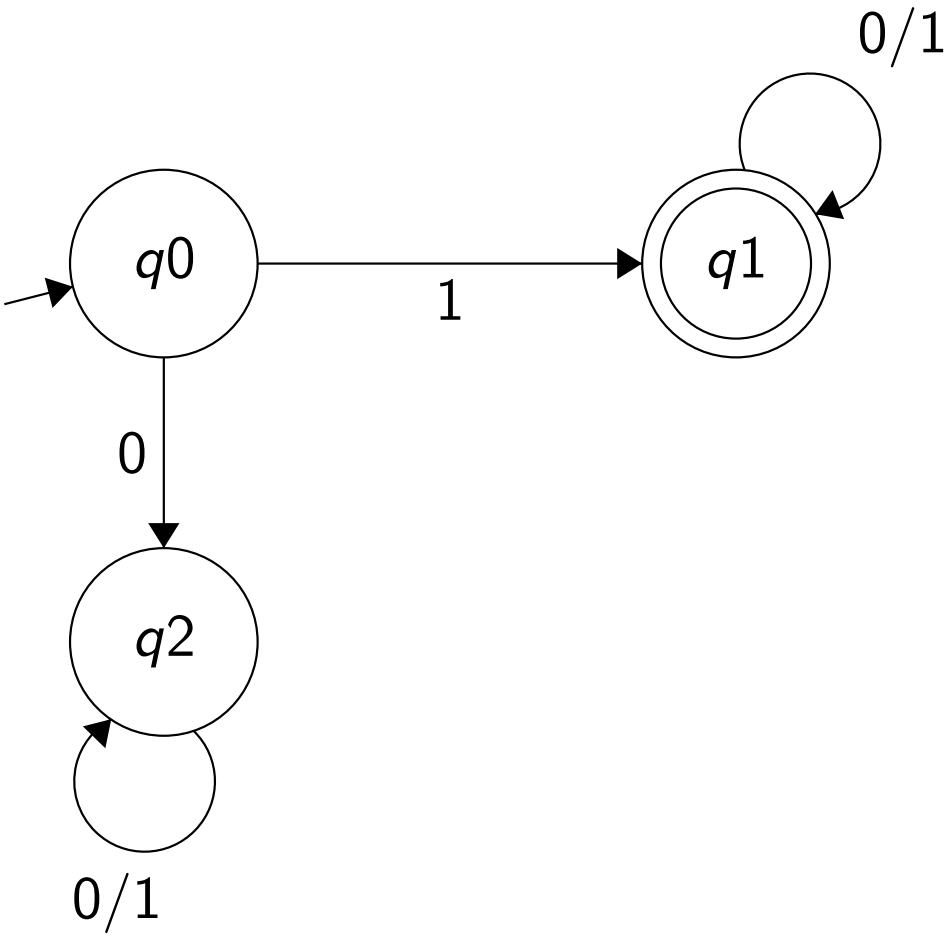
6 Represent DFA with tuples.

$$DFA, M = (Q, \Sigma, \delta, q_0, F)$$

$$\begin{aligned}
where\ Q &= \{q0, q1, q2\} \\
\Sigma &= \{0, 1\} \\
\delta : \delta(q0, 0) &= q2 \\
\delta(q0, 1) &= q1 \\
\delta(q1, 0) &= q1 \\
\delta(q1, 1) &= q1 \\
\delta(q2, 0) &= q2 \\
\delta(q2, 1) &= q2 \\
q0\text{–}initial\ state &= q0 \\
F\text{–}finalstate &= \{q1\}
\end{aligned}$$

Deterministic Finite Automata (DFA)

Dead State: A rejecting state that is essentially a dead end. Once the machine enters a dead state, there is no way for it to reach an accepting state, so we already know that the string is going to be rejected.



state q_2 is a dead state

Deterministic Finite Automata (DFA)

Language acceptd by a DFA

- The language accepted by a DFA M is the set of all string accepted by M .
- It is denoted by $L(M)$.

$$L(M) = \{w \in \Sigma^* / M \text{ accepts } w\}$$

Deterministic Finite Automata (DFA)

Extending the transition function to string (Extended transition function - $\hat{\delta}$)

- It is a function that takes a state q and a string w and return a state p.
- **Example:**
  - consider a string w =101
  - generally we write the transition function as

$$\delta(\delta(\delta(q0,1),0),1)$$

  - Instead of that we can write.

$$\hat{\delta}(q0,101)$$

  - generally $\hat{\delta}(q0,w) = p$
  - if p∈f then w is accepted

Deterministic Finite Automata (DFA)

Properties of $\hat{\delta}$

1. $\hat{\delta}(qo, \epsilon) = q0$
2. $\hat{\delta}(qo, wa) = \delta(\hat{\delta}(q0, w), a)$

$$\delta : Q \times \Sigma \rightarrow Q$$

$$\hat{\delta} : Q \times \Sigma^* \rightarrow Q$$

3. $\hat{\delta} = \delta$

$$\begin{aligned} \hat{\delta}(q, a) &= \hat{\delta}(q, \epsilon a) \\ &= \delta(\hat{\delta}(q, \epsilon), a) \\ &= \delta(q, a) \\ \hat{\delta} &= \delta \end{aligned}$$

Deterministic Finite Automata (DFA)

Language acceptd by a DFA

- Set of all string accepted by the finite automata.
- It is denoted by $L(M)$.

$$L(M) = \{w / \hat{\delta}(q_0, w) = p \text{ and } p \in f\}$$

Nondeterministic finite automaton (NFA)

Nondeterminisim means a choice of moves for an automaton. Rather than prescribing a unique move in each situation
The transition function that a state & input symbol as arguments , but returns a set of zero , one or more states.

Nondeterministic finite automaton (NFA)

- Formally, a Nondeterministic finite automaton can be represented by a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, f)$$

  - Q is a finite set of states
  - Σ is a finite input alphabet
  - δ is the transition function mapping $Q \times \Sigma$ *into* 2^Q, (which is the power set of Q, the set of all subsets of Q;)
  - $q_0 \in Q$ is the initial state
  - F $\subseteq$ Q is the set of final states.(It is assumed here that there may be more than one final state.)

Nondeterministic finite automaton (NFA)

Example:

NFA that accepts strings which contains either two consecutive 0's or two consecutive 1's.

$$L = \{00, 11, 100, 001, 110, 011, 111, 000, 0100, 1011, ...\}$$

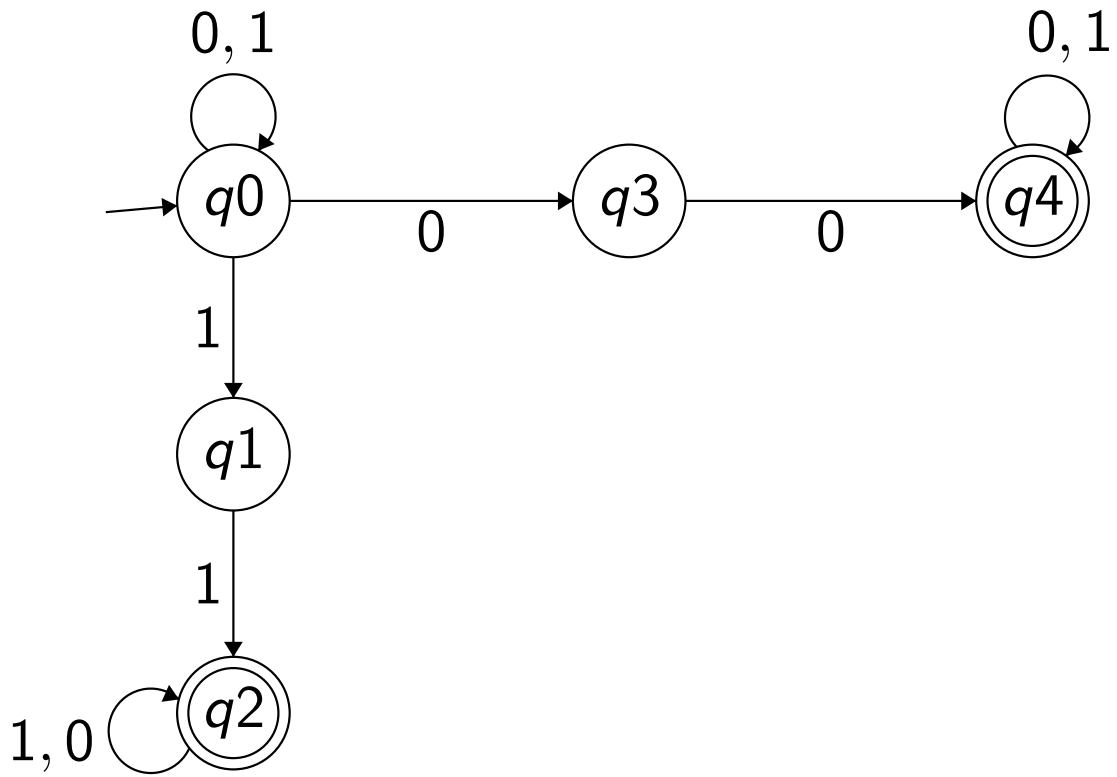


Figure: Transition Diagram

Nondeterministic finite automaton (NFA)

Example Cont..

State\input	0	1
$\rightarrow$q0	{q0,q3}	{q0,q1}
q1	ϕ	{q2}
*q2	{q2}	{q2}
q3	{q4}	ϕ
*q4	{q4}	{q4}

Table: Transition Table

Nondeterministic finite automaton (NFA)

Example Cont..

Let the input, $w = 01001 \in L$

$$
\begin{aligned}
\delta(q0, 0) &= \{q0, q3\} \\
\hat{\delta}(q0, 01) &= \delta(\delta(q0, 0), 1) \\
&= \delta(\{q0, q3\}, 1) \\
&= \delta(q0, 1) \cup \delta(q3, 1) \\
&= \{q0, q1\} \cup \phi \\
&= \{q0, q1\} \\
\hat{\delta}(q0, 010) &= \delta(\hat{\delta}(q0, 01), 0) \\
&= \delta(\{q0, q1\}, 0) \\
&= \delta(qo, 0) \cup \delta(q1, 0) \\
&= \{qo, q3\} \cup \phi \\
&= \{q0, q3\}
\end{aligned}
$$

Nondeterministic finite automaton (NFA)

Example Cont..

$$
\begin{aligned}
\hat{\delta}(q0, 0100) &= \delta(\hat{\delta}(q0, 010), 0) \\
&= \delta(\{q0, q3\}, 0) \\
&= \delta(q0, 0) \cup \delta(q3, 0) \\
&= \{q0, q3\} \cup \{q4\} \\
&= \{q0, q3, q4\} \\
\hat{\delta}(q0, 01001) &= \delta(\hat{\delta}(q0, 0100), 1) \\
&= \delta(\{q0, q3, q4\}, 1) \\
&= \delta(qo, 1) \cup \delta(q3, 1) \cup \delta(q4, 1) \\
&= \{q0, q1\} \cup \phi \cup \{q4\} \\
&= \{q0, q1, q4\}
\end{aligned}
$$

After the entire string is consumed, the FA is in the state q4. As q4 is the final state, the string is a accepted by FA

Nondeterministic finite automaton (NFA)

Example Cont..

$M = (Q, \Sigma, \delta, q_0, F)$

$$
\begin{aligned}
\textit{where } Q &= \{q0, q1, q2, q3, q4\} \\
\Sigma &= \{0, 1\} \\
\delta : \delta(q0, 0) &= \{q0, q3\} \\
\delta(q0, 1) &= \{q0, q1\} \\
\delta(q1, 0) &= \phi \\
\delta(q1, 1) &= \{q2\} \\
\delta(q2, 0) &= \{q2\} \\
\delta(q2, 1) &= \{q2\} \\
\delta(q3, 0) &= \{q4\} \\
\delta(q3, 1) &= \phi \\
\delta(q4, 0) &= \{q4\} \\
\delta(q4, 1) &= \{q4\} \\
q_0 &= q0 \\
F &= \{q2, q4\}
\end{aligned}
$$

Nondeterministic finite automaton (NFA)

The extended transition function

1. $\hat{\delta}$(q,ϵ)=q
   - NFA cannot change its state without accepting an input symbol
2. $\hat{\delta}(q, wa) = \{p/$ *for some state r in* $\hat{\delta}(q, w)$, *for some state p in* $\delta(r, a)\}$
   - $\hat{\delta}(q, w) = \{r1, r2, r3, r4, ...\}$
   - $\delta(\hat{\delta}(q, w), a) = \delta(\{r1, r2, r3, r4, ...\}, a) = \{p1, p2, p3, p4, ...\}$
3. $\hat{\delta}(q, a)$=$\delta(q, a)$
   - $\hat{\delta}(q, a) = \hat{\delta}(q, \epsilon a) = \delta(\hat{\delta}(q, \epsilon), a) = \delta(q, a)$
   - $\hat{\delta}$=δ
4. $\delta(p, w) = \bigcup_{q \ in \ p} \delta(q, w)$
   - Suppose p={q1,q2,q3,...,qn}

$$\begin{aligned} \delta(\{q1, q2, q3, ..., qn\}, w) &= \delta(q1, w) \cup \delta(q2, w) \cup ....\delta(qn, w) \\ &= \bigcup_{i=1}^{n} \delta(q_i, w) \end{aligned}$$

Nondeterministic finite automaton (NFA)

The Language accepted by an NFA

- Set of all string accepted by an NFA is called language accepted by an NFA

$$\begin{aligned} L(M) &= \{x/\hat{\delta}(q,x) \text{ is in } f\} \\ &\text{or} \\ L(M) &= \{w/\hat{\delta}(q0,w) \cap f \neq \phi\} \end{aligned}$$

$\hat{\delta}(q,w)$ contain a final state then it is accepted

All DFA's are NFA's

Differences between NFA and DFA

NFA	DFA
A Nondeterministic finite automaton is a 5-tuple $M = (Q, \Sigma, \delta, q_0, f)$ where $\delta : Q \times \Sigma$ *into* 2^Q	A Deterministic finite automaton is a 5-tuple $M = (Q, \Sigma, \delta, q_0, f)$ where $\delta : Q \times \Sigma$ *into* Q
NFA is the one in which there exists many paths for a specific input from current state to next state.	DFA is a FA in which there is only one path for a specific input from current state to next state.
Next move cannot be determind by the present state and current input symbol.	Next move can be determind by the present state and current input symbol.
NFA is easier to construct.	DFA is more difficult to construct.
NFA requires less space.	dfa requires more space.
Time required for executing an input string is more.	Time required for executing an input string is less

Table: Differences between NFA and DFA

Applications of FA:

- Used in Lexical analysis phase of a compiler to recognize tokens.
- Used in text editors for string matching.
- Software for designing and checking the behavior of digital circuits.

Limitations of FA:

- FA's will have finite amount of memory.
- The class of languages recognized by FA s is strictly the regular set. There are certain languages which are non regular i.e. cannot be recognized by any FA.

NFA to DFA Conversion

Let, $M = (Q, \Sigma, \delta, q0, F)$ is an NFA which accepts the language L(M). There should be equivalent DFA denoted by $M' = (Q', \Sigma', q0', \delta', F')$ such that$L(M) = L(M')$.

1. Initially Q' = ϕ
2. Add q0 of NFA to Q'. Then find the transitions from this start state.
3. In Q', find the possible set of states for each input symbol. If this set of states is not in Q', then add it to Q'.
4. In DFA, the final state will be all the states which contain F(final states of NFA)

Q:Convert the given NFA to DFA.

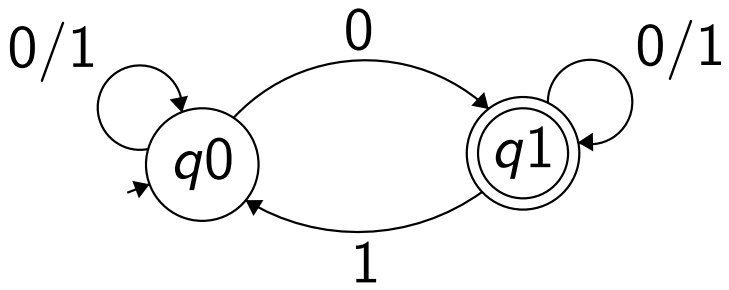


Figure: NFA

Solution: For the given transition diagram we will first construct the transition table.

δ	0	1
→q0	{q0, q1}	{q0}
*q1	{q1}	{q0, q1}

Convert the given NFA to DFA

Now we will obtain δ' transition for state q0.

$$\begin{aligned}\delta'([q0], 0) &= \{q0, q1\} \\ &= [q0, q1] \quad \textbf{(new state generated)} \\ \delta'([q0], 1) &= \{q1\} = [q1]\end{aligned}$$

The δ' transition for state q1 is obtained as:

$$\begin{aligned}\delta'([q1], 0) &= \{q1\} \\ \delta'([q1], 1) &= \{q0, q1\}\end{aligned}$$

Now we will obtain δ' transition on [q0, q1].

$$\begin{aligned}\delta'([q0, q1], 0) &= \delta(q0, 0) \cup \delta(q1, 0) \\ &= \{q0, q1\} \cup \{q1\} \\ &= \{q0, q1\} \\ &= [q0, q1]\end{aligned}$$

Convert the given NFA to DFA

Similarly

$$\begin{aligned}\delta'([q0,q1],1) &= \delta(q0,1)\cup\delta(q1,1)\\ &= \{q0\}\cup\{q0,q1\}\\ &= \{q0,q1\}\\ &= [q0,q1]\end{aligned}$$

As in the given NFA, q1 is a final state, then in DFA wherever, q1 exists that state becomes a final state. Hence in the DFA, final states are [q1] and [q0, q1].

The transition table for the constructed DFA will be:

δ	0	1
$\rightarrow$[q0]	[q0, q1]	[q0]
*[q1]	[q1]	[q0, q1]
*[q0, q1]	[q0, q1]	[q0, q1]

Table: Transition table

Convert the given NFA to DFA

The Transition diagram will be:

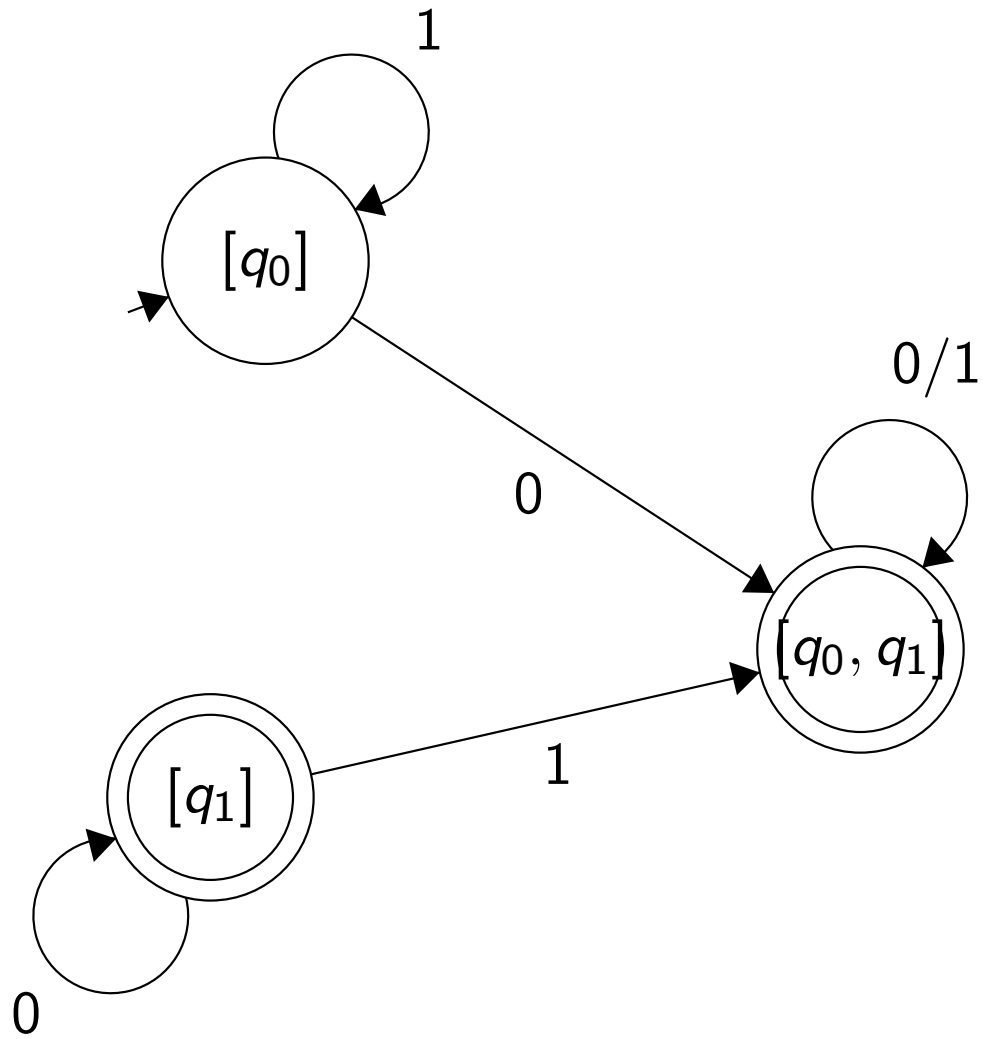


Figure: Transition Digram - DFA

Convert the given NFA to DFA

Even we can change the name of the states of DFA.

$$\begin{aligned} A &= [q0] \\ B &= [q1] \\ C &= [q0, q1] \end{aligned}$$

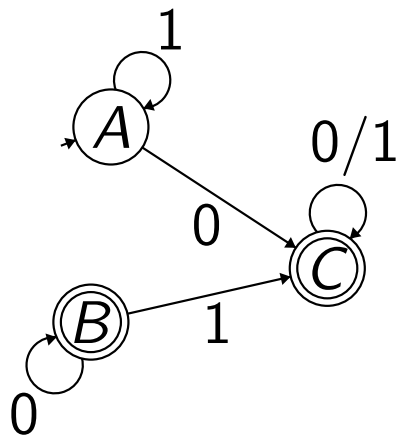


Figure: Transition diagram -DFA

Upon careful inspection, it appears that state B is unreachable from the initial state A. Thus, for the purpose of minimizing the DFA, it is advisable to remove state B.

Equivalence of NFA and DFA

Equivalence of NFA and DFA

- If L is a Language accepted by an NFA then there is a DFA that accept L
  - i.e for every NFA there exist an equivalent DFA
  - The transition of NFA to DFA is normally called a subset construction

Equivalence of NFA and DFA

Theorem 1.0

Let language $L \subseteq \Sigma^*$, and suppose L is accepted by NFA $N = (\Sigma, Q, q0, F, \delta)$. There exists a DFA $D = (\Sigma, Q', q0', F', \delta')$ that also accepts L. $(L(N) = L(D))$.

By allowing each state in the DFA D to represent a set of states in the NFA N, we are able to prove through induction that D is equivalent to N. Before we begin the proof, let's define the parameters of D:

- Q' is equal to the powerset of Q, $Q' = 2^Q$
- $q_0' = \{q_0\}$
- F' is the set of states in Q' that contain any element of F, $F' = \{q \in Q' / q \cap F \neq \phi\}$
- δ' is the transition function for D. $\delta'(q, a) = \bigcup_{p \in q} \delta(p, a)$ *for* $q \in Q'$ *and* $a \in \Sigma$.

Equivalence of NFA and DFA

Now we will prove that $\hat{\delta}'(q_0, x) = \hat{\delta}(q_0, x)$ for every x. ie, L(D) = L(N)

Basis Step:

- Let x be the empty string ϵ.
- By the definition of extended transition function, NFA & DFA cannot change its state without accepting an input symbol

$$\begin{aligned} \hat{\delta}'(q_0, \epsilon) &= \{q_0\} \\ \hat{\delta}(q_0, \epsilon) &= \{q_0\} \end{aligned}$$

Therefor $\delta'(q_0, \epsilon) = \delta(q_0, \epsilon)$ is true for $x = \epsilon$.

Inductive Step:

- Assume that for any y with $|y| \geq 0, \hat{\delta}'(q_0, y) = \hat{\delta}(q_0, y)$.
  - Let both state be $p = \{p_1, p_2, p_3 ...\}$
- Let $n = |y|$, then we need to prove that for a string z with $|z| = n + 1, \hat{\delta}'(q_0, z) = \hat{\delta}(q_0, z)$.

Equivalence of NFA and DFA

- We can represent the string z as a concatenation of string y ($|y| = n$) and symbol a from the alphabet Σ.($a \in \Sigma$). So, $z = ya$.

$$
\begin{aligned}
\hat{\delta}'(q_0, z) &= \hat{\delta}'(q_0, ya) \\
&= \delta'(\hat{\delta}'(q_0, y), a) \\
&= \delta'(\hat{\delta}(q_0, y), a) \quad (\textit{by assumption}) \\
&= \bigcup_{p \in \hat{\delta}(q_0, y)} \delta'(p, a) \quad (\textit{by definition of } \hat{\delta}) \\
&= \bigcup_{p \in \hat{\delta}(q_0, y)} \delta(p, a) \quad (\textit{by assumption}) \\
&= \hat{\delta}(q_0, ya) \\
&= \hat{\delta}(q_0, z)
\end{aligned}
$$

- DFA D accepts a string x iff $\hat{\delta}'(q_0, x) \in F'$. From the above it follows that D accepts x iff $\hat{\delta}(q_0, x) \cap F \neq \phi$.
- So a string is accepted by DFA D if, and only if, it is accepted by NFA N.

NFA with ϵ transitions(ϵ-NFA)

In the case of ϵ-NFA without applying an input symbol the state will change

Formal Definition of NFA with ϵ transitions

$$M = (Q, \Sigma, \delta, q_0, F)$$

where

- Q is a set of states,
- Σ is the alphabet,
- δ is the transition function $\delta : Q \times \{\Sigma \cup \epsilon\}$ into 2^Q,
- q0 is the initial state,
- $F \subset Q$ is the set of final (or accepting) states.

$\delta(q, a)$ will consists of all states p such that there is a transition labelled a from q to p. Where a is either ϵ or a symbol in Σ.

NFA with ϵ transitions(ϵ-NFA)

Example:

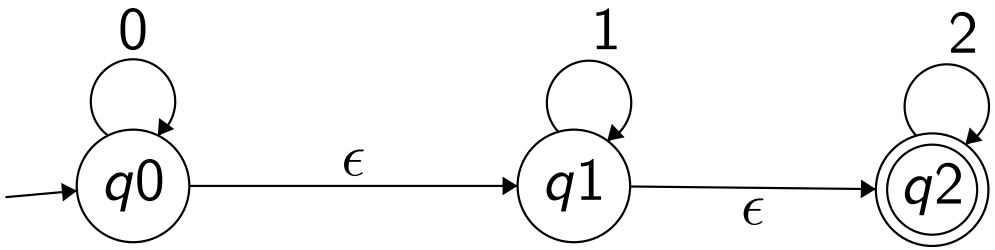


Figure: Transition diagram

δ	0	1	2	ϵ
$\rightarrow$q0	{q0}	ϕ	ϕ	{q1}
q1	ϕ	{q1}	ϕ	{q2}
*q2	ϕ	ϕ	{q2}	ϕ

Table: Transition table

ϵ-closure

ϵ-closure(q)

- It is a set of states that can be reached from q along the path labelled by ϵ only

$$\epsilon - closure(p_i) = \{q_i / q_j \text{ is reachable from } p_i \text{ using zero or more } \epsilon - transition\}$$

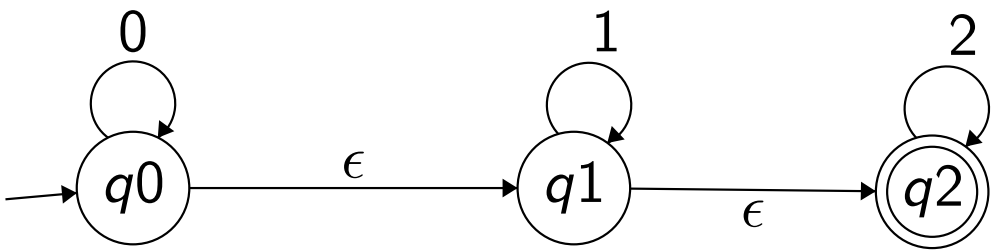


Figure: Transition diagram

$$\epsilon - closure(q_0) = \{q_0, q_1, q_2\}$$
$$\epsilon - closure(q_1) = \{q_1, q_2\}$$
$$\epsilon - closure(q_2) = \{q_2\}$$

NFA with ϵ transitions(ϵ-NFA)

Extended transition function($\hat{\delta}$) in ϵ-NFA

1. $\hat{\delta}(q, \epsilon) = \epsilon - closure(q)$
2. for w in Σ^* and a in Σ

$$\begin{aligned} \hat{\delta}(q, wa) &= \epsilon - closure(p) \\ &= \{p / for\ some\ r\ in\ \hat{\delta}(q, w), p\ is\ in\ \delta(r, a)\} \end{aligned}$$

3. $\delta(R, a) = \bigcup_{q\ in\ R} \delta(q, a)$
4. $\hat{\delta}(R, w) = \bigcup_{q\ in\ R} \hat{\delta}(q, w)$
5. $\hat{\delta}(q, a) = \epsilon - closure(\delta(\hat{\delta}(q, \epsilon), a))$

Equivalence of NFA with and without epsilon transitions

Conversion of Epsilon-NFA to NFA

1. Find out all the ϵ-transitions from each state from Q. That will be called as $\epsilon - closure(q_i)$ where, $qi \in Q$.
2. Then, δ_1 transitions can be obtained. The δ_1 transitions means an ϵ-closure on δ moves.
3. Step 2 is repeated for each input symbol and for each state of given NFA.
4. By using the resultant status, the transition table for equivalent NFA without ϵ can be built.

Equivalence of NFA with and without epsilon transitions

Conversion of Epsilon-NFA to NFA

1. Find out all the ϵ-transitions from each state from Q. That will be called as $\epsilon - closure(q_i)$ where, $q_i \in$Q.
2. Then, δ_1 transitions can be obtained. The δ_1 transitions means an ϵ-closure on δ moves.
3. Step 2 is repeated for each input symbol and for each state of given NFA.
4. By using the resultant status, the transition table for equivalent NFA without ϵ can be built.

Equivalence of NFA with and without epsilon transitions

Example: Convert the given NFA with epsilon to NFA without epsilon.

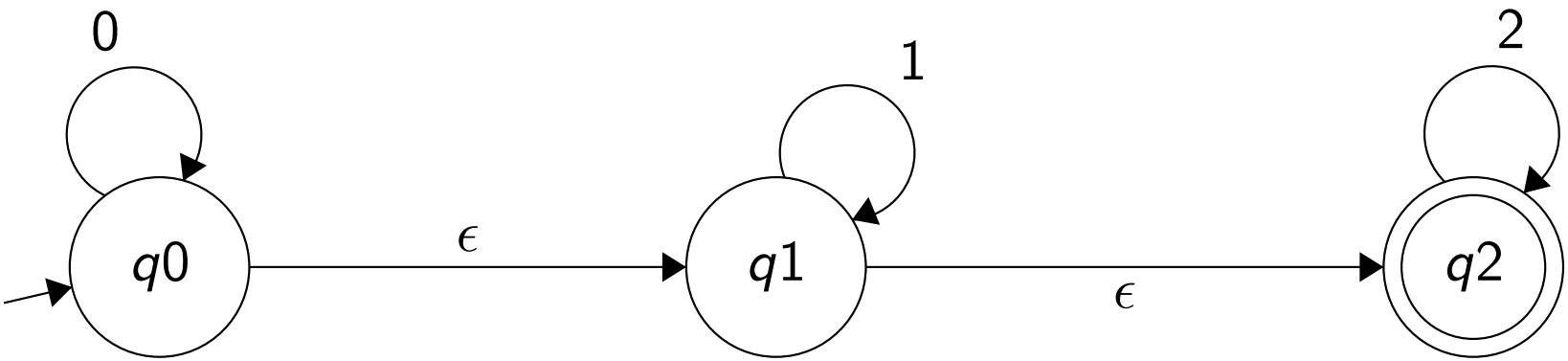


Figure: NFA-With ϵ transition

Equivalence of NFA with and without epsilon transitions

Solution: We will first obtain $\epsilon - closure$ of each state

$$\begin{aligned}
\epsilon - closure(q0) &= \{q0, q1, q2\} \\
\epsilon - closure(q1) &= \{q1, q2\} \\
\epsilon - closure(q2) &= \{q2\}
\end{aligned}$$

Now we will obtain $\delta 1$ transitions for each state on each input symbol

$$\begin{aligned}
\delta'(q0, 0) &= \epsilon - closure(\delta(\hat{\delta}(q0, \epsilon), 0)) \\
&= \epsilon - closure(\delta(\epsilon - closure(q0), 0)) \\
&= \epsilon - closure(\delta(q0, q1, q2), 0)) \\
&= \epsilon - closure(\delta(q0, 0) \cup \delta(q1, 0) \cup \delta(q2, 0)) \\
&= \epsilon - closure(q0 \cup \phi \cup \phi) \\
&= \epsilon - closure(q0) \\
&= \{q0, q1, q2\}
\end{aligned}$$

Equivalence of NFA with and without epsilon transitions

Solution cont..

$$
\begin{aligned}
\delta'(q0,1) &= \epsilon - closure(\delta(\hat{\delta}(q0,\epsilon),1)) \\
&= \epsilon - closure(\delta(q0,q1,q2),1)) \\
&= \epsilon - closure(\delta(q0,1) \cup \delta(q1,1) \cup \delta(q2,1)) \\
&= \epsilon - closure(\phi \cup q1 \cup \phi) \\
&= \epsilon - closure(q1) \\
&= \{q1,q2\} \\
\delta'(q0,2) &= \epsilon - closure(\delta(\hat{\delta}(q0,\epsilon),2)) \\
&= \epsilon - closure(\delta(q0,q1,q2),2)) \\
&= \epsilon - closure(\delta(q0,2) \cup \delta(q1,2) \cup \delta(q2,2)) \\
&= \epsilon - closure(\phi \cup \phi \cup q2) \\
&= \epsilon - closure(q2) \\
&= \{q2\}
\end{aligned}
$$

Equivalence of NFA with and without epsilon transitions

Solution cont..

$$
\begin{aligned}
\delta'(q1,0) &= \epsilon - closure(\delta(\hat{\delta}(q1,\epsilon),0)) \\
&= \epsilon - closure(\delta(q1,q2),0)) \\
&= \epsilon - closure(\delta(q1,0) \cup \delta(q2,0)) \\
&= \epsilon - closure(\phi \cup \phi) \\
&= \epsilon - closure(\phi) \\
&= \phi \\
\delta'(q1,1) &= \epsilon - closure(\delta(\hat{\delta}(q1,\epsilon),1)) \\
&= \epsilon - closure(\delta(q1,q2),1)) \\
&= \epsilon - closure(\delta(q1,1) \cup \delta(q2,1)) \\
&= \epsilon - closure(q1 \cup \phi) \\
&= \epsilon - closure(q1) \\
&= \{q1,q2\}
\end{aligned}
$$

Equivalence of NFA with and without epsilon transitions

Solution cont..

$$
\begin{aligned}
\delta'(q1,2) &= \epsilon - closure(\delta(\hat{\delta}(q1,\epsilon),2)) \\
&= \epsilon - closure(\delta(q1,q2),2)) \\
&= \epsilon - closure(\delta(q1,2) \cup \delta(q2,2)) \\
&= \epsilon - closure(\phi \cup q2) \\
&= \epsilon - closure(q2) \\
&= \{q2\} \\
\delta'(q2,0) &= \epsilon - closure(\delta(\hat{\delta}(q2,\epsilon),0)) \\
&= \epsilon - closure(\delta(q2),0)) \\
&= \epsilon - closure(\delta(q2,0)) \\
&= \epsilon - closure(\phi) \\
&= \phi
\end{aligned}
$$

Equivalence of NFA with and without epsilon transitions

Solution cont..

$$
\begin{aligned}
\delta'(q2,1) &= \epsilon - closure(\delta(\hat{\delta}(q2,\epsilon),1)) \\
&= \epsilon - closure(\delta(q2),1) \\
&= \epsilon - closure(\delta(q2,1)) \\
&= \epsilon - closure(\phi) \\
&= \phi \\
\delta'(q2,2) &= \epsilon - closure(\delta(\hat{\delta}(q2,\epsilon),)) \\
&= \epsilon - closure(\delta(q2),2)) \\
&= \epsilon - closure(\delta(q2,2)) \\
&= \epsilon - closure(q2) \\
&= \{q2\}
\end{aligned}
$$

Equivalence of NFA with and without epsilon transitions

Solution cont..

Now, we will summarize all the computed δ' transitions as given below

$$
\begin{aligned}
\delta'(q0,0) &= \{q0,q1,q2\} \\
\delta'(q0,1) &= \{q1,q2\} \\
\delta'(q0,2) &= \{q2\} \\
\delta'(q1,0) &= \{\phi\} \\
\delta'(q1,1) &= \{q1,q2\} \\
\delta'(q1,2) &= \{q2\} \\
\delta'(q2,0) &= \{\phi\} \\
\delta'(q2,1) &= \{\phi\} \\
\delta'(q2,2) &= \{q2\}
\end{aligned}
$$

Equivalence of NFA with and without epsilon transitions

Solution cont.. The transition table is given below

δ	0	1	2
$\rightarrow$*q0	{q0,q1,q2}	{q1,q2}	{q2}
q1	ϕ	{q1,q2}	{q2}
*q2	ϕ	ϕ	{q2}

Table: Transition table NFA without ϵ transition

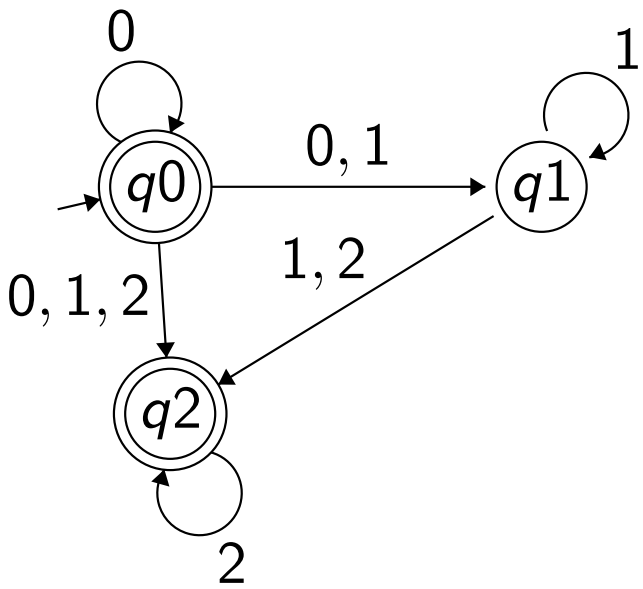


Figure: Transition digram NFA without ϵ transition

Conversion NFA with epsilon to DFA

The method for converting the NFA with ϵ to DFA is explained below

- Consider $M = (Q, \Sigma, \delta, q0, F)$ is NFA with ϵ.
- We have to convert this NFA with ϵ to equivalent DFA denoted by $M_D = (Q_D, \Sigma, \delta_D, q0, F_D)$
- Then obtain, $\epsilon - closure(q0) = \{p1, p2, p3, \ldots\ldots pn\}$
- then $[p1, p2, p2, \ldots . pn]$ becomes a start state of DFA
- now$[p1, p2, p3, \ldots . pn] \in Q_D$
- We will obtain δ transition on [p1,p2,p3,. . . pn] for each input.
  - $\delta_D([p1, p2, p3, ..pn], a) = \epsilon - closure(\delta(p1, a)$ $\cup\ \delta(p2, a2) \cup \ldots\ldots\ldots\ldots\ldots\ldots \delta(pn, a))$
  - $= \bigcup_{i=1}^{n} \epsilon - closured(pi, a)$
    Where a is input $\in \Sigma$
- The state obtained $[p1, p2, p3, \ldots pn] \in Q_D$.
- The states containing final state in pi is a final state in DFA

Conversion NFA with epsilon to DFA

Example: Convert the following NFA with epsilon to equivalent DFA

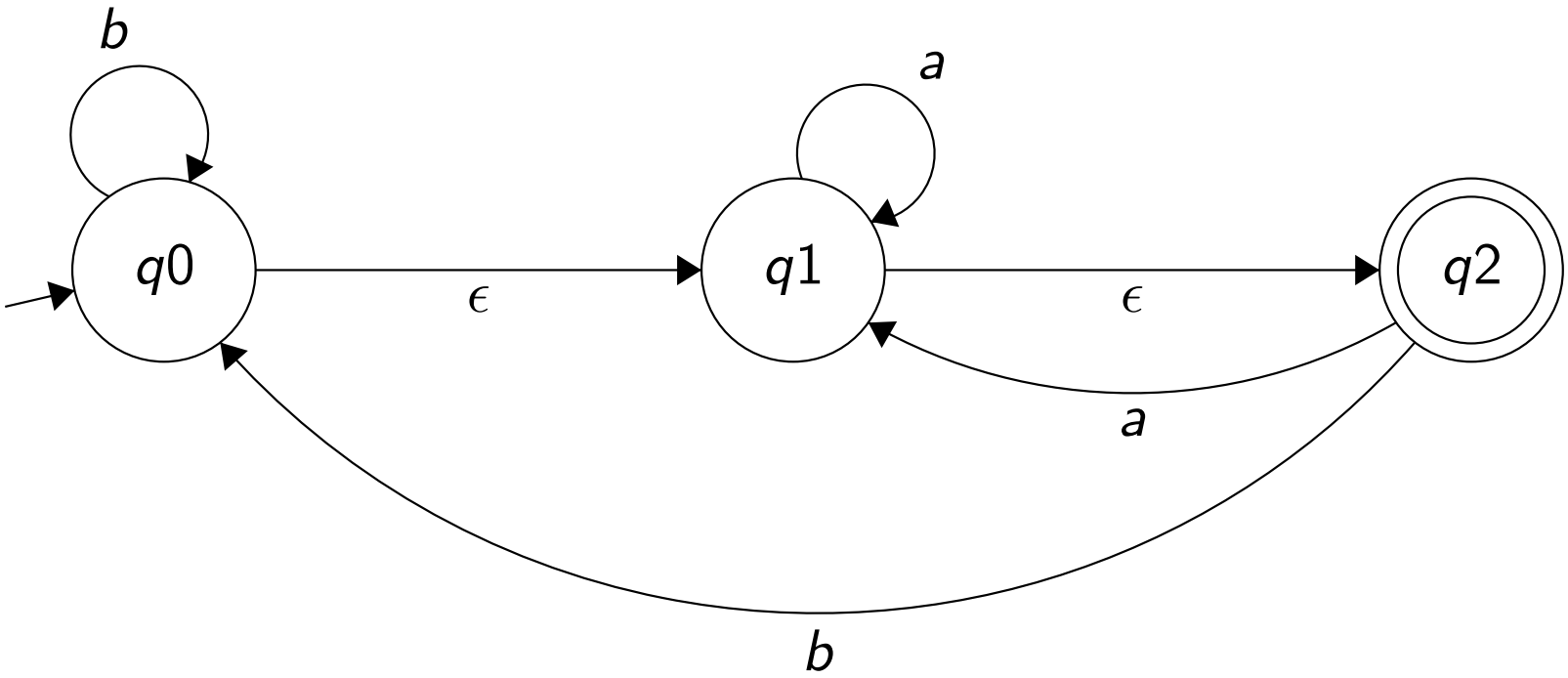


Figure: NFA with ϵ transition

Conversion NFA with epsilon to DFA

Solution:

To convert this NFA with epsilon, we will first find the ϵ -closures,

$$\begin{aligned} \epsilon - closure(q0) &= \{q0, q1, q2\} \\ \epsilon - closure(q1) &= \{q1, q2\} \\ \epsilon - closure(q2) &= \{q2\} \end{aligned}$$

When, $\epsilon - closure(q0) = \{q0, q1, q2\}$, we will call this state as A.
Now, let us find transition on A with every input symbol

$$\begin{aligned} \delta'(A, a) &= \epsilon - closure(\delta(A, a)) \\ &= \epsilon - closure(\delta(q0, q1, q2), a)) \\ &= \epsilon - closure(\delta(q0, a) \cup \delta(q1, a) \cup \delta(q2, a)) \\ &= \epsilon - closure(\phi \cup q1 \cup q2) \\ &= \epsilon - closure(q1 \cup q2) \\ &= \{q1, q2\} \textit{ let us call it as state } B \end{aligned}$$

Conversion NFA with epsilon to DFA

Solution cont..

$$
\begin{aligned}
\delta'(A,b) &= \epsilon - closure(\delta(A,b)) \\
&= \epsilon - closure(\delta(q0,q1,q2),b)) \\
&= \epsilon - closure(\delta(q0,b) \cup \delta(q1,b) \cup \delta(q2,b)) \\
&= \epsilon - closure(q0 \cup \phi \cup q0) \\
&= \epsilon - closure(q0) \\
&= \{q0,q1,q2\} \ its\ nothing\ but\ state\ A \\
\delta'(B,a) &= \epsilon - closure(\delta(B,a)) \\
&= \epsilon - closure(\delta(q1,q2),a)) \\
&= \epsilon - closure(\delta(q1,a) \cup \delta(q2,a)) \\
&= \epsilon - closure(q1 \cup q2) \\
&= \epsilon - closure(q1) \\
&= \{q1,q2\} \ its\ nothing\ but\ state\ B
\end{aligned}
$$

Conversion NFA with epsilon to DFA

Solution cont..

$$
\begin{aligned}
\delta'(B, b) &= \epsilon - closure(\delta(B, b)) \\
&= \epsilon - closure(\delta(q1, q2), b)) \\
&= \epsilon - closure(\delta(q1, b) \cup \delta(q2, b)) \\
&= \epsilon - closure(\phi \cup q0) \\
&= \epsilon - closure(q0) \\
&= \{q0, q1, q2\} \ its \ nothing \ but \ state \ A
\end{aligned}
$$

Conversion NFA with epsilon to DFA

Solution cont.. Hence, the transition table for the generated DFA is as follows

δ	a	b
$\rightarrow$*A	B	A
*B	B	A

Table: DFA without ϵ

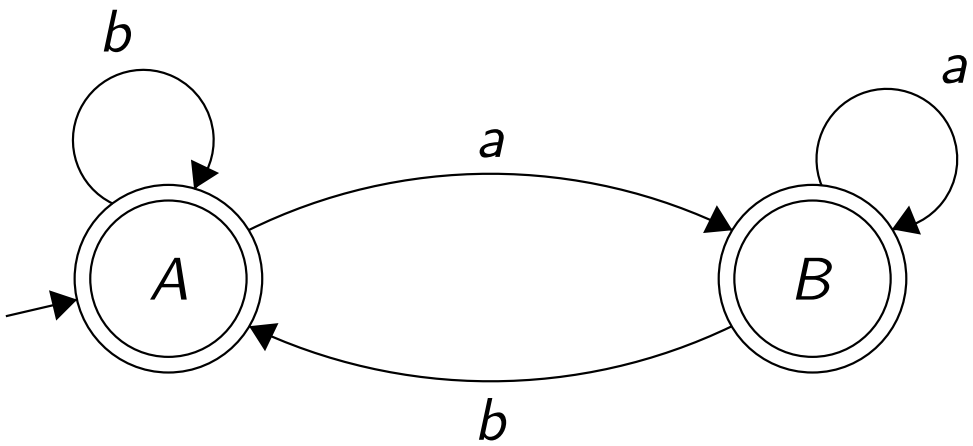


Figure: DFA without ϵ diagram

Equivalence of NFA with and without epsilon transitions

Theorem 1.1

If L is a language accepted by NFA with ϵ- transition, Then L is accepted by NFA without ϵ-transition.

Proof:

- Let $M = (Q, \Sigma, \delta, q0, F)$ be a NFA with ϵ- transition, then we need to construct an NFA, $M' = (Q, \Sigma, \delta', q_0, F')$
- Where

$$F' = \begin{cases} F \cup \{q_0\} & \text{if } \epsilon - \text{closure}(q_0) \text{ contains a state from } F, \\ F & \text{Otherwise} \end{cases}$$

and

$$\delta'(q, a) = \delta(q, a) \text{for } q \in Q \text{ and } a \in \Sigma$$

Equivalence of NFA with and without epsilon transitions

Proof Cont..

Basis

- $|x| = 1$ then x is a symbol a and

$$\delta'(q0, a) = \delta(q0, a) \text{ by definition of } \delta'$$

Induction

- Assume the hypothysis is true for the string of length m or less

$$i.e \quad \hat{\delta}'(q0, x) = \hat{\delta}(q0, x)$$

To prove this theorem we must show that

$$\hat{\delta}'(q0, xa) = \hat{\delta}(q0, xa) \quad \text{where } |xa| = m + 1$$

Equivalence of NFA with and without epsilon transitions

$$\begin{aligned}\hat{\delta}(q0, xa) &= \delta'(\hat{\delta}'(q0, x), a) \\ &= \delta'(\hat{\delta}(q0, x), a) \quad \textit{by hypothysis}\end{aligned}$$

let $\hat{\delta}(q0, x) = P = \{q0, q1, q2, q3, ...q_n\}$

$$\begin{aligned}\textit{therefor } \hat{\delta}'(q0, xa) &= \delta'(P, a) \\ &= \bigcup_{q \in P} \delta'(q, a) \\ &= \bigcup_{q \in P} \delta(q, a) \\ &= \delta(P, a) \\ &= \delta(\hat{\delta}(q0, x), a) \\ &= \hat{\delta}(q0, xa)\end{aligned}$$

$$\textit{Thus } \delta'(q0, xa) = \hat{\delta}(q0, xa)$$

Chomsky Hierarchy in Theory of Computation

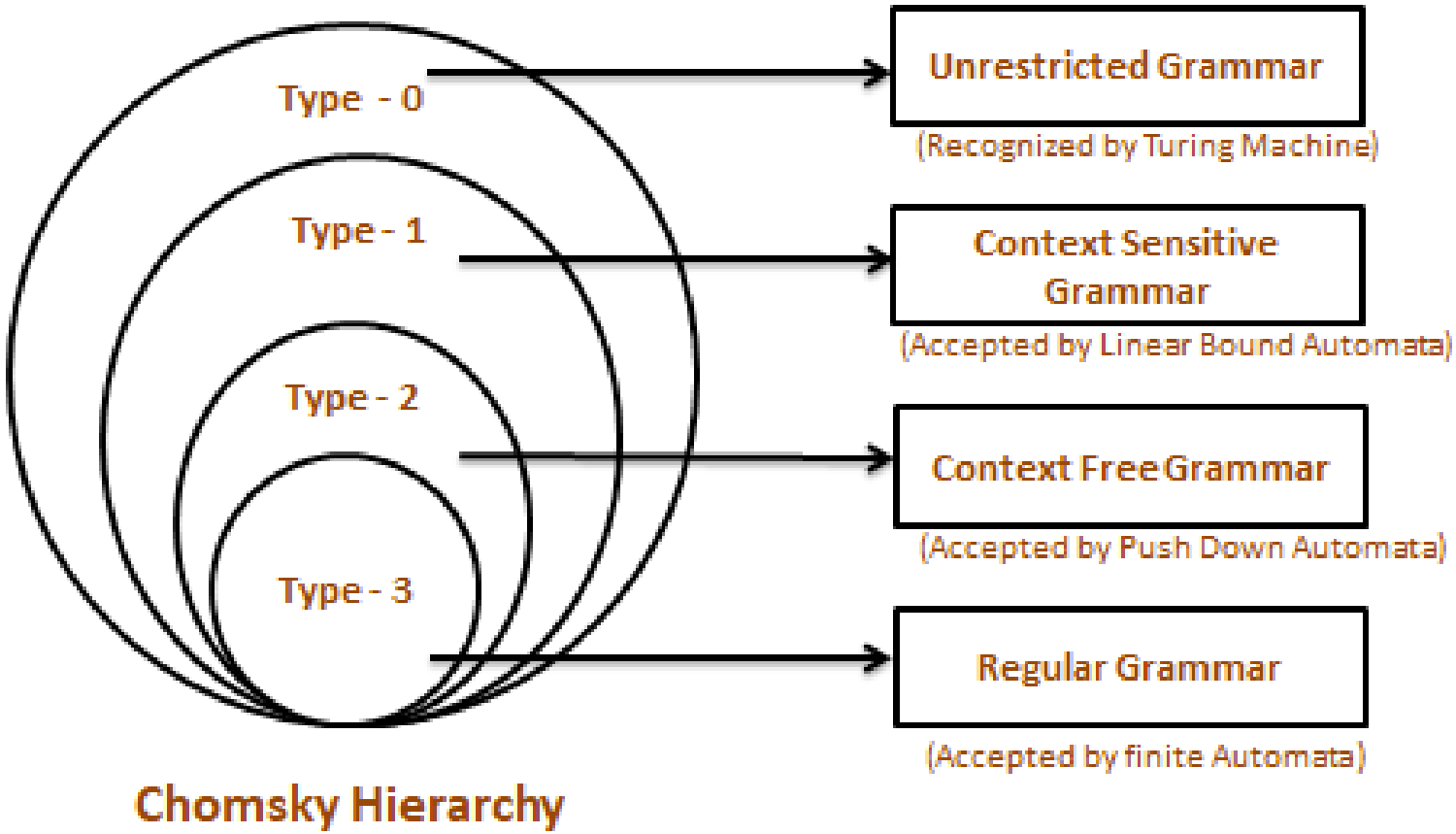


Figure: Chomsky Hierarchy in Theory of Computation

Regular Grammar

Regular Grammar(RG)-Type 3

- Regular grammar generates regular language.
- Regular Languages are recognized by using finite state automata.

Regular Grammar

Formal definition of Regular Grammar(RG)-Type 3

- A Regular Grammar 'G' can be formally described using four tuples as $G = (V, T, P, S)$
- where,
  - V = Set of Variables or Non-Terminal Symbols(Capital letters)
  - T = Set of Terminal Symbols(Small letters, Numbers etc)
  - S = Start Symbol $S \in V$
  - P = Production Rules for Terminals and Non-Terminals

All productions are of the form

$$A \rightarrow xB$$

$$A \rightarrow Cx$$

$$A \rightarrow x$$

where $A, B, C \in V, x \in T^*$

Regular Grammar

Types of regular grammar:

1. Right linear grammar(RLG)
2. Left Linear grammar(LLG)

Regular Grammar

Right-linear grammar

- A grammar is said to be Right-linear if all productions are of the form:

$$A \to xB$$
$$A \to xC$$
$$A \to y$$
$$\text{where } A, B, C \in V, x, y \in T^*$$

Example

$$\begin{aligned} G &= (\{S, B\}, \{a, b\}, P, S), \\ P &= \\ &\quad S \to aB|bS|\epsilon \\ &\quad B \to aS|bB \end{aligned}$$

Regular Grammar

Left Linear grammar

- A grammar is said to be Left Linear if all productions are of the form:

$$A \to Bx$$
$$A \to Cy$$
$$A \to x$$
$$\text{where } A, B, C \in V, x, y \in T^*$$

Example

$$\begin{aligned} G &= (\{S, B\}, \{a, b\}, P, S), \\ P &= \\ &\quad S \to Ba|Sb|\epsilon \\ &\quad B \to Sa|Bb \end{aligned}$$

Regular Grammar

Derivation of sentences
Grammar

$$\begin{aligned} A &\rightarrow aB \\ B &\rightarrow aB \\ B &\rightarrow bB \\ B &\rightarrow \epsilon \end{aligned}$$

Derivation from the above grammar

$$\begin{aligned} A &\Longrightarrow aB \Longrightarrow a \\ A &\Longrightarrow aB \Longrightarrow aaB \Longrightarrow aa \\ A &\Longrightarrow aB \Longrightarrow abB \Longrightarrow ab \\ A &\Longrightarrow aB \Longrightarrow aaB \Longrightarrow aaaB \Longrightarrow aaa \end{aligned}$$

Language generated by the grammar is $L(G) = \{ax | x \in \{a, b\}^*\}$

Regular Grammar

- Language generated by the grammar is $L(G) = \{ax | x \in \{a, b\}^*\}$
- i.e $a(a + b)^*$(Called Regular Expression)

Regular Grammar

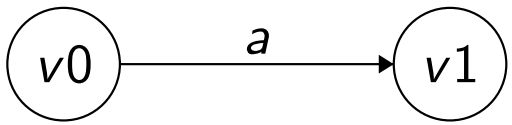
Language generated by Regular Grammar

- The class of languages generated by the family of regular grammar is precisely the set of Regular Language
  - The language genereated by a Regular grammar is Regular
  - Any Regular Language can be generated by a Regular Grammar

Converting Regular Grammar to DFA

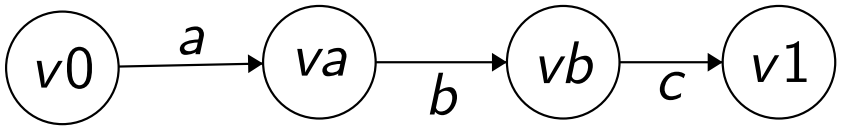
Assume that a regular grammar is given in its right-linear form, this grammar may be easily converted to a DFA. A right-linear grammar, defined by $G = (V, T, S, P)$, may be converted to a DFA, defined by $M = (Q, \Sigma, \delta, q0, F)$ by:

1. Create a state for each variable.
2. Convert each production rule into a transition.
   a. If the production rule is of the form $V_i \rightarrow aV_j$, where $a \in T$, add the transition $\delta(V_i, a) = V_j$ to M. For example, $V_0 \rightarrow aV_1$ becomes:

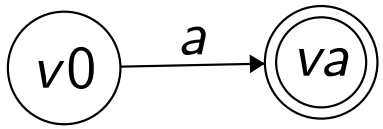


Regular Grammar

b If the production rule is of the form $V_i \rightarrow wV_j$, where $w \in T^*$, create a series of states which derive w and end in V_j. Add the states in between to Q. For example, $V_0 \rightarrow abcV_1$ becomes



c If the production rule is of the form $V_i \rightarrow w$, where $w \in T^*$, create a series of states which derive w and end in a final state. For example, $V_0 \rightarrow a$ becomes:



Regular Grammar

Theorem

Let G=(V,T,P,S) be a right linear grammar. then L(G) is regular language

Proof
We assume that $V = \{V_0, V_1, ...\}$, *that* $S = V_0$, and that we have productions of the form $V_0 \to v_1 V_i, V_i \to v_2 V_j, ...$ *or* $V_n \to v_l, ...$ If w is a string in L(G), then because of the form of the productions

$$\begin{aligned} V_0 &\Longrightarrow v_1 V_i \\ &\Longrightarrow v_1 v_2 V_j \\ &\vdots \\ &\Longrightarrow v_1 v_2 \dots v_k V_n \\ &\Longrightarrow v_1 v_2 \dots v_k v_l = w. \end{aligned}$$

Regular Grammar

The automaton to be constructed will reproduce the derivation by consuming each of these v's in turn. The initial state of the automaton will be labeled V_0 , and for each variable V_i there will be a nonfinal state labeled V_i . For each production

$$V_i \implies a_1 a_2 \ldots\ldots a_m V_j,$$

the automaton will have transitions to connect V_i and V_j that is, δ will be defined so that

$$\delta^*(V_i, a_1 a_2 \ldots . a_m) = V_j$$

For each production

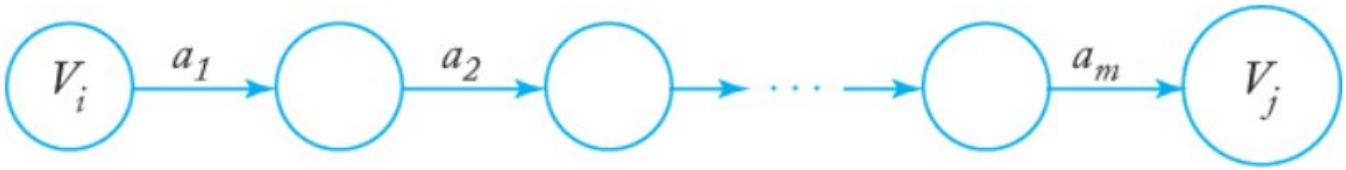
$$V_i \implies a_1 a_2 \ldots . a_m,$$

the corresponding transition of the automaton will be

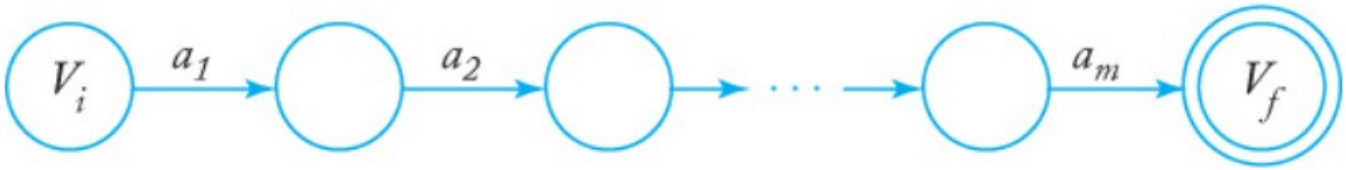
$$\delta^*(V_i, a_1 a_2 \ldots . a_m) = V_f,$$

Regular Grammar

where V_f is a final state. The intermediate states that are needed to do this are of no concern and can be given arbitrary labels.



Represents $V_i \rightarrow a_1 a_2 \ldots a_m V_j$



Represents $V_i \rightarrow a_1 a_2 \ldots a_m$

Regular Grammar

Suppose now that $w \in L(G)$ so that the above equation is satisfied. In the nfa there is, by construction, a path from V_0 to V_i labeled v1 , a path from V_i to V_j labeled v_2 , and so on, so that clearly

$$V_f \in \delta^*(V_0, w),$$

and w is accepted by M.
Conversely, assume that w is accepted by M. Because of the way in which M was constructed, to accept w the automaton has to pass through a sequence of states V_0 , V_i , ...to V_f , using paths labeled v_1 , v_2 , ...
Therefore, w must have the form

$$w = v_1 v_2 v_k v_l$$

and the derivation

$$V_o \implies v_1 V_i \implies v_1 v_2 V_j \stackrel{*}{\Rightarrow} v_1 v_2 \dots v_k V_k \implies v_1 v_2 \dots v_k v_1$$

is possible. Hence w is in L (G), and the theorem is proved.

Regular Grammar

Theorem

If L is a regular language on the alphabet Σ, then there exists a right-linear grammar $G = (V, \Sigma, S, P)$ such that $L = L(G)$.

Proof

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a dfa that accepts L. We assume that$Q = \{q_0, q_1, ..., q_n\} and \Sigma = \{a_1, a_2, ..., a_m\}$. Construct the right-linear grammar $G = (V, \Sigma, S, P)$ with $V = \{q_0, q_1, ..., q_n\}$ and $S = q_0$. For each transition $\delta(q_i, a_j) = q_k$ of M, we put in P the production

$$q_i \rightarrow a_j q_k.$$

In addition, if q_k is in F, we add to P the production

$$q_k \rightarrow \lambda.$$

Regular Grammar

Proofcont..
We first show that G defined in this way can generate every string in L. Consider $w \in L$ of the form $w = a_i a_j ... a_k a_l$ For M to accept this string it must make moves via

$$\begin{aligned} \delta(q_0, a_i) &= q_p, \\ \delta(q_p, a_j) &= q_r, \\ &\vdots \\ \delta(q_s, a_k) &= q_t, \\ \delta_(q_t, a_l) &= q_f \in F. \end{aligned}$$

Regular Grammar

Proofcont..
By construction, the grammar will have one production for each of these δ's. Therefore, we can make the derivation

$$\begin{aligned} q_0 &\Longrightarrow a_i q_p \Longrightarrow a_i a_j q_r \overset{*}{\Rightarrow} a_i a_j \dots a_k q_t \\ &\Longrightarrow a_i a_j \dots a_k a_l q_f \Longrightarrow a_i a_j \dots a_k a_l, \end{aligned}$$

with the grammar G, and $w \in L(G)$.
Conversely, if$w \in L(G)$, then its derivation must have the form the above equation. But this implies that $\delta^*(q_0, a_i a_j \dots a_k a_l) = q_f$, completing the proof.