BAPATLA ENGINEERING COLLEGE :: BAPATLA
(AUTONOMOUS)

Automata & Compiler Design (18IT502)

Unit-I Material

Department of Information Technology
BAPATLA ENGINEERING COLLEGE::BAPATLA
(AUTONOMOUS)
Affiliated to Acharya Nagarjuna University
Bapatla-522102, Guntur (District), AP.

AUTOMATA & COMPILER DESIGN
18IT502
B.Tech.,(Semester- V)

Lectures	**:**	**3** Periods/Week **,** Tutorial: 1	**Continuous Assessment**	**:**	50
Final Exam	**:**	**3** Hours	**Final Exam Marks**	**:**	50

Course Objectives: The student will understand:

COB 1: The concepts of finite automata and regular languages and their properties.

COB 2: The concepts of Context free grammars and push down automata

COB 3: The phases of a compiler, lexical analysis and parsing techniques.

COB 4: Different intermediate code forms and code generation algorithm for target machine.

Course Outcomes: Upon successful completion of the course, the student will be able to:

CO 1: Design finite state machines for acceptance of strings and understand the concepts of regular languages and their properties.

CO 2: Design context free grammars for formal languages and develop pushdown automata for accepting strings

CO 3: Understand the phases of a compiler and construct lexical analysis, top-down and bottom-up parsers

CO 4: Apply intermediate, code generation techniques and runtime allocation strategies.

UNIT - I

Finite Automata: Introduction to Automata, Deterministic finite automata (DFA), Problems on DFA, Non deterministic finite automata (NFA), Equivalence of DFA and NFA, Finite Automata with € transitions**,** Equivalence and minimization of automata.

Regular Expressions and Languages: Regular expressions, Algebraic laws of regular expressions, Pumping lemma for regular languages, Applications of the pumping lemma, Closure Properties of Regular Languages.

UNIT – II

Context Free Grammars: Context Free Grammars, Parse Trees, Constructing parse trees, derivations and parse trees, ambiguous grammars.

Pushdown Automata: Definition of the Pushdown automata, the languages of PDA, Equivalences of PDA's and CFG's.

Context free languages: Normal forms for context- Free grammars, the pumping lemma for context free languages.

UNIT-III

Introduction to compiling: Compilers, The Phases of a compiler.

Lexical Analysis: The role of the lexical analyzer, input buffering, simplification of tokens, Recognition of tokens, implementing transition diagrams, a language for specifying lexical analyzers.

Syntax analysis: Top down parsing - Recursive descent parsing, Predictive parsers. Bottom up parsing - Shift Reduce parsing, LR Parsers – Construction of SLR, Canonical LR and LALR parsing techniques, Parser generators – YACC Tool.

UNIT-IV

Intermediate code Generation: Intermediate languages, Declarations, Assignment statements, Boolean expressions, back patching.

Runtime Environment: Source language issues, Storage organization, Storage-allocation strategies.

Code Generation- Issues in the design of code generator, the target machines, Basic blocks and flow graphs, Next use information, a simple code generator.

TEXT BOOKS:

John E. Hopcroft et al., Introduction to Automata Theory, Languages and Computation, 3rd Ed., Pearson, 2007.

A.V. Aho et al., “Compilers: Priniciples, Techniques, Tools”, 2nd Edition, Pearson, 2006.

REFERENCES:

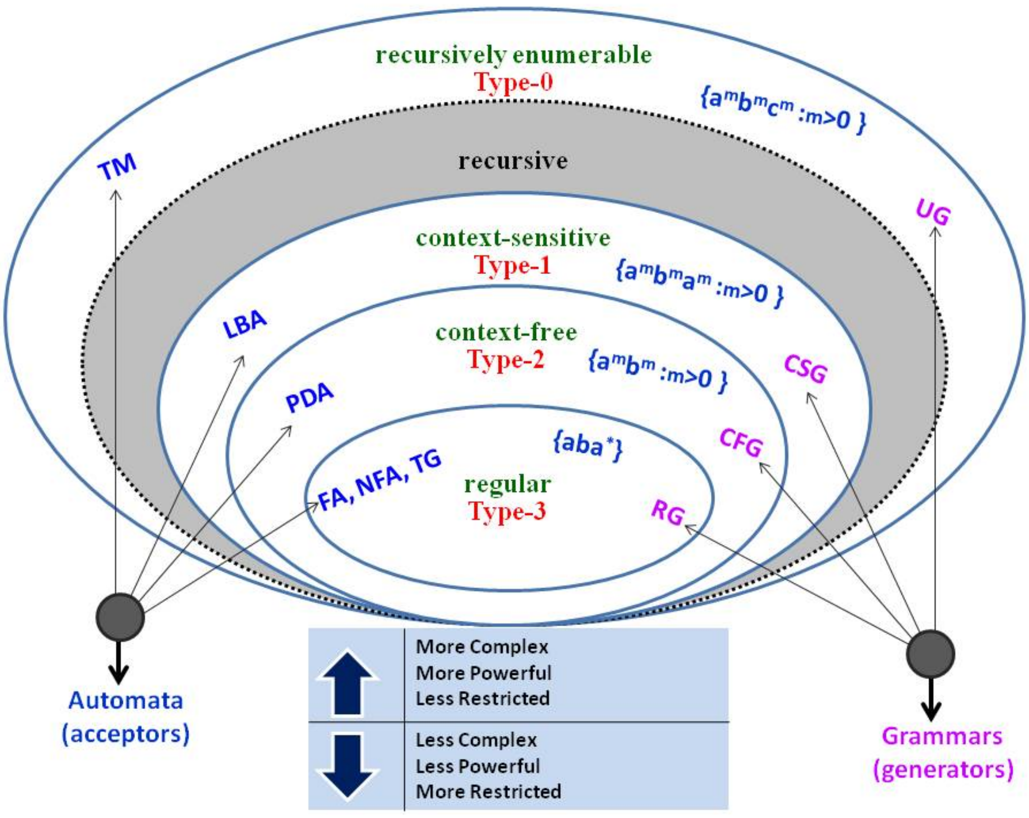
1.John E Hopcroft & Jeffery D Ullman, “Introduction to Automata Theory & Languages and Computation”, Narosa Publishing House.

2. Alfred V.Aho, Jeffrey D. Ullman, “Principles of Compiler Design”, Narosa publishing.

UNIT-I

Finite Automata

Chomsky Hierarchy:



Language	Grammar	Machine	Example
Regular language	Regular grammar —Right-linear grammar —Left-linear grammar	Deterministic or Nondeterministic finite-state acceptor	a^*
Context-free language	Context-free grammar	Nondeterministic pushdown automaton	$a^n b^n$
Context-sensitive language	Context sensitive grammar	Linear-bounded automaton	$a^n b^n c^n$
Recursively enumerable language	Unrestricted grammar	Turing machine	Any computable function

An automaton (Automata in plural) is an abstract self-propelled computing device which follows a predetermined sequence of operations automatically.

An automaton with a finite number of states is called a Finite Automaton (FA) or Finite State Machine (FSM).

Formal definition of a Finite Automaton

An automaton can be represented by a 5-tuple (Q, Σ, δ, q0, F), where:

- **Q** is a finite set of states.
- **Σ** is a finite set of symbols, called the **alphabet** of the automaton.
- **δ** is the transition function.
- **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 ($F \subseteq Q$).

Central Concepts of Automata Theory

Alphabet:

Definition: An **alphabet** is any finite set of symbols. It is denoted by ∑

Example: Σ = {a, b, c, d} is an **alphabet set** where 'a', 'b', 'c', and'd' are **symbols**.

Example:

1. If Σ is an alphabet containing all the 26 characters used in English language, then Σ is finite and nonempty set, and $\Sigma = \{a, b, c, \dots, z\}$.
2. $X = \{0,1\}$ is an alphabet.
3. $Y = \{1, 2, 3, \dots\}$ is not an alphabet because it is infinite.
4. $Z = \{\ \}$ is not an alphabet because it is empty.

String

Definition: A **string** is a finite sequence of symbols taken from Σ.

Example: 'cabcad' is a valid string on the alphabet set Σ = {a, b, c, d}

Length of a String

Definition: It is the number of symbols present in a string. (Denoted by **|S|**).

Examples

If S='cabcad', |S|= 6

If |S|= 0, it is called an **empty string** (Denoted by **λ** or **ε**)

Concatenation of strings

If w_1 and w_2 are two strings then concatenation of w_2 with w_1 is a string and it is denoted by w_1w_2. In other words, we can say that w_1 is followed by w_2 and $|w_1w_2| = |w_1| + |w_2|$.

Prefix of a string

A string obtained by removing zero or more trailing symbols is called prefix. For example, if a string $w = abc$, then a , ab , abc are prefixes of w.

Suffix of a string

A string obtained by removing zero or more leading symbols is called suffix. For example, if a string $w = abc$, then c, bc , abc are suffixes of w.
A string a is a proper prefix or suffix of a string w if and only if $a \neq w$.

Substrings of a string

A string obtained by removing a prefix and a suffix from string w is called substring of w . For example, if a string $w = abc$, then b is a substring of w. Every prefix and suffix of string w is a substring of w, but not every substring of w is a prefix or suffix of w. For every string w, both w and $\in$ are prefixes, suffixes, and substrings of w.
Substring of w = w – (one prefix) – (one suffix).

Kleene Closure / Star

Definition: The Kleene star, Σ*, is a unary operator on a set of symbols or strings, Σ, that gives the infinite set of all possible strings of all possible lengths over Σ including €.

Representation: Σ* = Σ0 U Σ1 U Σ2 U……. where Σp is the set of all possible strings of length p.

Example: If Σ = {a, b}, Σ*= {λ, a, b, aa, ab, ba, bb,………..}

$L^* = \{\textit{Set of all words over } \Sigma\}$
= {word of length zero, words of length one, words of length two, ...}

$$= \bigcup_{K=0}^{\infty} (\Sigma^K) = L^0 \cup L^1 \cup L^2 \cup \ldots$$

Example:

1. $\Sigma = \{a, b\}$ and a language L over Σ. Then
   $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$
   $L^0 = \{\in\}$
   $L^1 = \{a, b\}$,
   $L^2 = \{aa, ab, ba, bb\}$ and so on.
   So, $L^* = \{\in, a, b, aa, ab, ba, bb \ldots\}$
2. $S = \{0\}$, then $S^* = \{\in, 0, 00, 000, 0000, 00000, \ldots\}$

Positive Closure / Plus

Definition: The set Σ+ is the infinite set of all possible strings of all possible lengths over Σ excluding €.

Representation: Σ+ = Σ1 U Σ2 U Σ3 U…….

Σ+= Σ* − { € }

Example: If Σ = { a, b } , Σ+ ={ a, b, aa, ab, ba, bb,………..}

Example :

$$\text{if } \Sigma = \{0\}, \text{ then } \Sigma^{+} = \{0, 00, 000, 0000, 00000, ...\}$$

Language

Definition: A language is a subset of Σ* for some alphabet Σ. It can be finite or infinite.

Example: If the language takes all possible strings of length 2 over Σ = {a, b}, then L = { ab, bb, ba, bb}

1. $L_1 = \{01, 0011, 000111\}$ is a language over alphabet $\{0,1\}$
2. $L_2 = \{\in, 0, 00, 000, ...\}$ is a language over alphabet $\{0\}$
3. $L_3 = \{0^n 1^n 2^n ; n \geq 1\}$ is a language.

Deterministic Finite Automata (DFA)

Finite Automaton can be classified into two types:

- Deterministic Finite Automaton (DFA)
- Non-deterministic Finite Automaton (NDFA / NFA)

Deterministic Finite Automaton (DFA)

In DFA, for each input symbol, one can determine the state to which the machine will move. Hence, it is called **Deterministic Automaton**. As it has a finite number of states, the machine is called **Deterministic Finite Machine or Deterministic Finite Automaton.**

Formal Definition of a DFA:

A DFA can be represented by a 5-tuple (Q, Σ, δ, q_0, F) where:

- **Q** is a finite set of states.
- **Σ** is a finite set of symbols called the alphabet.
- **δ** is the transition function where δ: Q × Σ → Q
- **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 ($F \subseteq Q$).

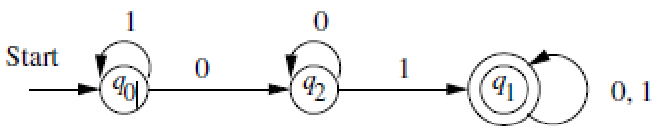
Simpler Notations for DFA:

There are two preferred notations for describing automata:

1. A Transition Diagram – This is a graph

A DFA is represented by digraphs called **state diagram**.

- The vertices represent the states.
- The arcs labelled with an input alphabet show the transitions.
- The initial state is denoted by an arrow into the start state q_0 , labelled ***start***
- The final state is indicated by double circles.



2. A Transition Table – This is a tabular listing of the δ function.

A transition table is a conventional, tabular representation of a function like δ that takes two arguments and returns a value.

- The rows of the table correspond to the **states** and the columns correspond to the **input**.

The entry for the row corresponding to state q and column corresponding to input a is **the state δ (q, a)**

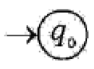
Example:

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

Notations used for representing FA

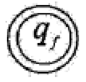
We represent a FA by describing all the five - terms $(Q, \Sigma, \delta, q_0, F)$. By using diagram to represent FA make things much clearer and readable. We use following notations for representing the FA :

1. The initial state is represented by a state within a circle and an arrow entering into circle as shown below :



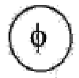
(Initial state q_0)

2. Final state is represented by final state within double circles :

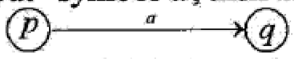


(Final state q_f)

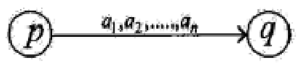
3. The hang state is represented by the symbol 'ϕ' within a circle as follows :



4. Other states are represented by the state name within a circle.
5. A directed edge with label shows the transition (or move). Suppose p is the present state and q is the next state on input - symbol 'a', then this is represented by



6. A directed edge with more than one label shows the transitions (or moves). Suppose p is the present state and q is the next state on input - symbols 'a_1' or 'a_2' or ... or 'a_n' then this is represented by



Extending the transition function to strings:

If δ is our transition function, then the extended transition function constructed from δ will be called δ^1 or

$\hat{\delta}$.

We define δ^1 by induction on the length of the input string, as follows

BASIS: $\hat{\delta}(q, \epsilon) = q$. That is, if we are in state q and read no inputs, then we are still in state q.

INDUCTION: Suppose w is a string of the form xa; that is, a is the last symbol of w, and x is the string consisting of all but the last symbol.[3] For example, $w = 1101$ is broken into $x = 110$ and $a = 1$. Then

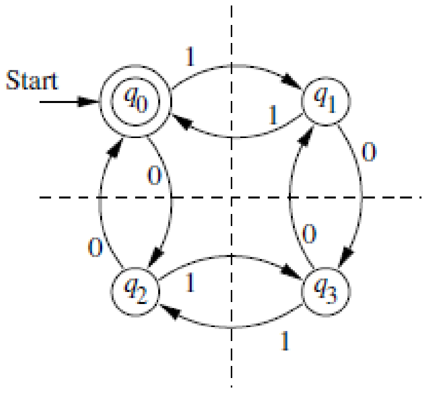
$$\hat{\delta}(q, w) = \delta\big(\hat{\delta}(q, x), a\big)$$

Problems on DFA:

1. Design a DFA to accept the language L = {w | w has both an even number of 0's and an even number of 1's}

Sol:

State q_0 is both the start state and the lone accepting state. It is the start state, because before reading any inputs, the numbers of 0's and 1's seen so far are both zero, and zero is even. It is the only accepting state, because it describes exactly the condition for a sequence of 0's and 1's to be in language L.



The DFA for language L is

$$A = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_0\})$$

Where the transition function δ is described by the transition diagram

	0	1
$* \rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

The check involves computing $\hat{\delta}(q_0, w)$ for each prefix w of 110101, starting at ϵ and going in increasing size. The summary of this calculation is:

- $\hat{\delta}(q_0, \epsilon) = q_0$.
- $\hat{\delta}(q_0, 1) = \delta\big(\hat{\delta}(q_0, \epsilon), 1\big) = \delta(q_0, 1) = q_1$.
- $\hat{\delta}(q_0, 11) = \delta\big(\hat{\delta}(q_0, 1), 1\big) = \delta(q_1, 1) = q_0$.
- $\hat{\delta}(q_0, 110) = \delta\big(\hat{\delta}(q_0, 11), 0\big) = \delta(q_0, 0) = q_2$.
- $\hat{\delta}(q_0, 1101) = \delta\big(\hat{\delta}(q_0, 110), 1\big) = \delta(q_2, 1) = q_3$.
- $\hat{\delta}(q_0, 11010) = \delta\big(\hat{\delta}(q_0, 1101), 0\big) = \delta(q_3, 0) = q_1$.
- $\hat{\delta}(q_0, 110101) = \delta\big(\hat{\delta}(q_0, 11010), 1\big) = \delta(q_1, 1) = q_0$.

The string 110101 is accepted by the DFA.

Case 1: If Final State is q0

L={ w | w has both an even number of 0's and an even number of 1's }

Case 2: If Final State is q1

L={ w | w has both an odd number of 0's and an even number of 1's }

Case 3: If Final state is q2

L={ w | w has both an even number of 0's and an odd number of 1's }

Case 4: If Final state is q3

L={ w | w has both an odd number of 0's and an odd number of 1's }

2. Give DFA's accepting the following languages over the alphabet {0, 1}

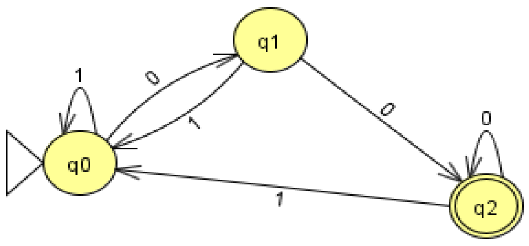
a) The set of all strings ending in 00

b) The set of all strings with three consecutive 0's (not necessarily at the end)

c) The set of strings with 011 as a substring.

a) The Set of all strings ending 00

Possible strings: L={00, 000,100,0100,1000,1100,0000,100100…… }



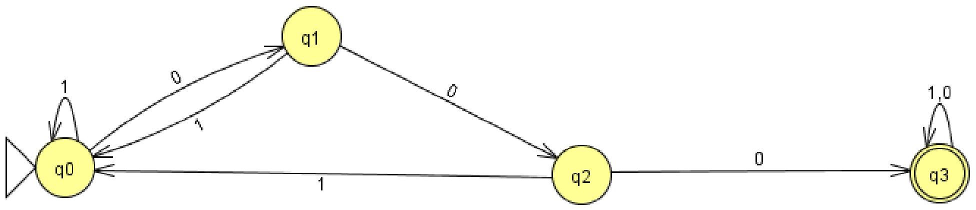
DFA A= ({q0,q1,q2}, {0,1}, δ,q0,{q2})

δ Transition Function:

	0	1
→q0	q1	q0
q1	q2	q0
*q2	q2	q0

b) The set of all strings with three consecutive 0's. (Not necessarily at the end)

L= { 000,0001,00010,10001,00011,00010,000101,010001,0100010,0010001……..}



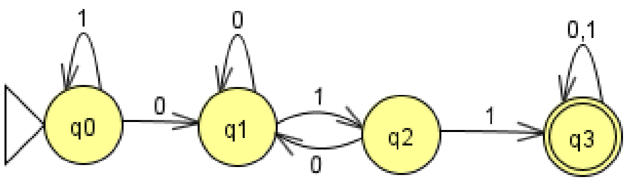
DFA A= ({q0,q1,q2,q3}, {0,1}, δ,q0,{q3})

δ Transition Function:

	0	1
→q0	q1	q0
q1	q2	q0
q2	q3	q0
*q3	q3	q3

c) The set of strings with 011 as substring.

L= { 011,0011,1011,0110,01101,01110,10110,10111,…….}

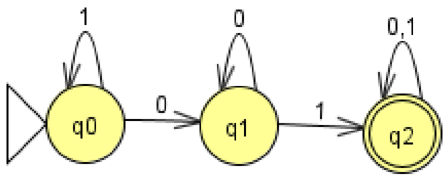


DFA A= ({q0,q1,q2,q3}, {0,1}, δ,q0,{q3}) δ Transition Function:

	0	1
→q0	q1	q0
q1	q1	q2
q2	q1	q3
*q3	q3	q3

3) The DFA accepting all strings with a substring 01.

L= { 01,101,001,011,010,0100,0110,0101,0111,1001,0001,……….}



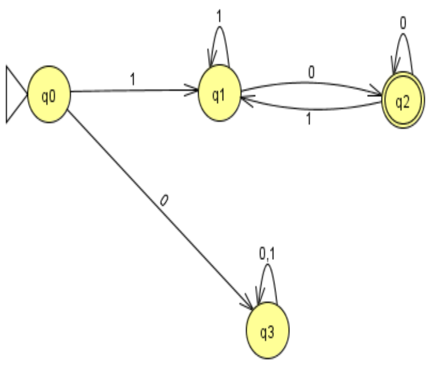
DFA A= ({q0,q1,q2}, {0,1}, δ,q0,{q2})

δ Transition Function:

	0	1
→q0	q1	q0
q1	q1	q2
*q2	q2	q2

4) Design a DFA which accepts the strings start with 1 and ends with 0

L= { 10,110,100,1000,1100,1010,1110,10000,11000,11100,11110,10110,……..}



DFA A= ({q0,q1,q2,q3}, {0,1}, δ,q0,{q2})

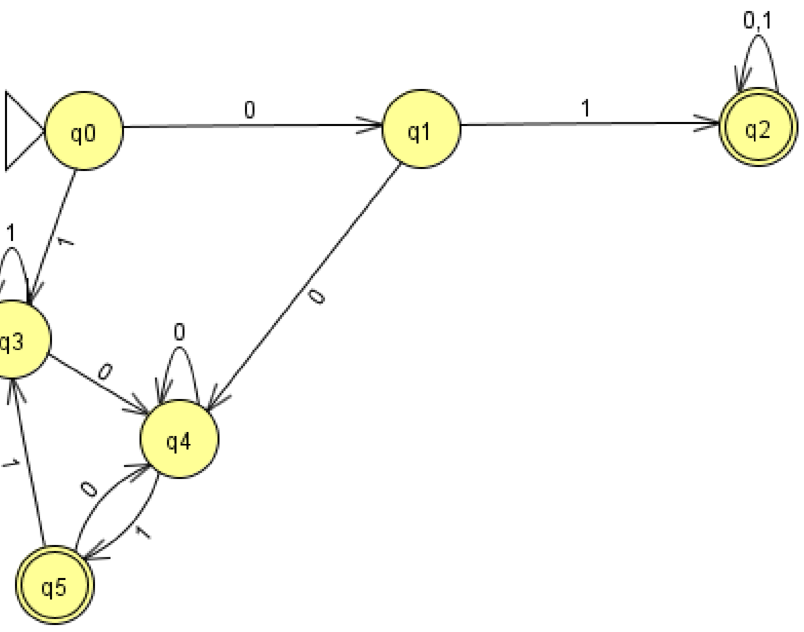
δ Transition Function:

	0	1
→q0	q3	q1
q1	q2	q1
*q2	q2	q1
q3 (Dead State)	q3	q3

5) The set of strings that either begin or end with 01

Begin with 0: L1={ 01w } → 01 followed by any string which consists of 0's and 1's.

End with 0: L2= { w01 } → Any string which consists of 0's and 1's followed by 01.



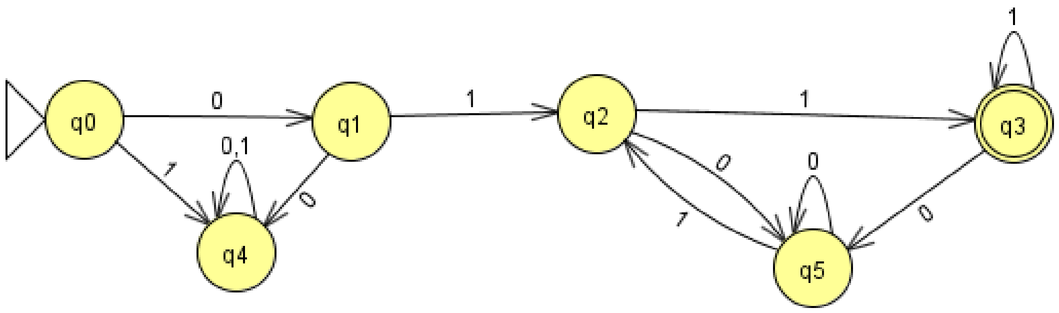
DFA A= ({q0,q1,q2,q3,q4,q5}, {0,1}, δ,q0,{q2,q5})

δ Transition Function:

	0	1
→q0	q1	q3
q1	q4	q2
q2	q2	q2
q3	q4	q1
q4	q4	q5
*q5	q4	q3

6) Design a DFA which accepts the string starts with 01 and ends with 11.

L={011,0111,01011,01111,010011,011111,011011,010111,…….}



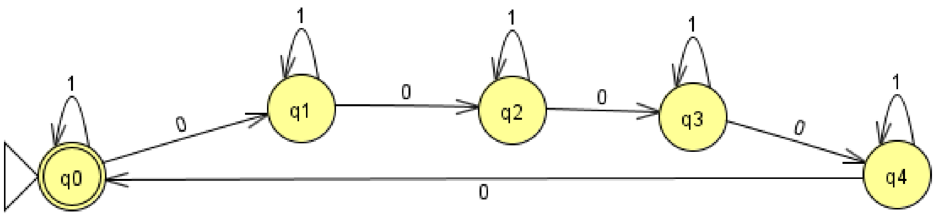
DFA A= ({q0,q1,q2,q3,q4,q5}, {0,1}, δ,q0,{q3})

δ Transition Function:

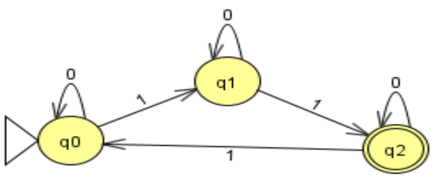
	0	1
→q0	q1	q4
q1	q4	q2
q2	q5	q3
*q3	q5	q3
q4(Dead State)	q4	q4
q5	q5	q2

7) The set of strings such that the number of 0's is divisible by 5 and the number of 1's is divisible by 3.

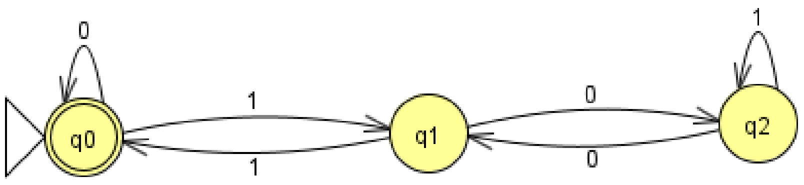
i) The number of 0's is divisible by 5



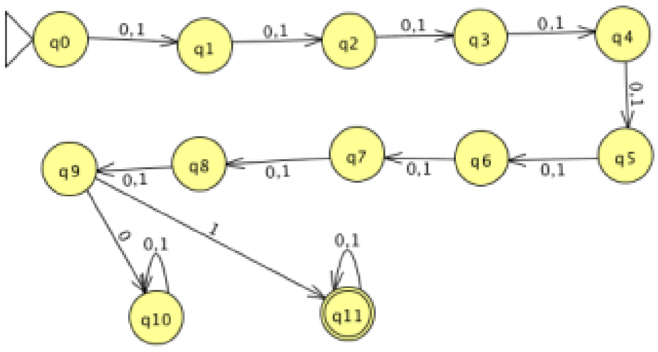
ii) The number of 1's is divisible by 3.



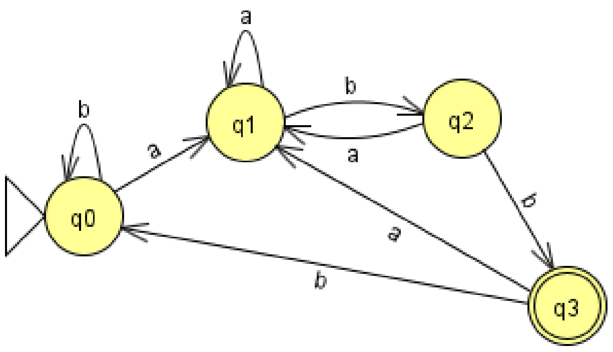
8) Design a DFA which checks whether a given binary number is divisible by 3.



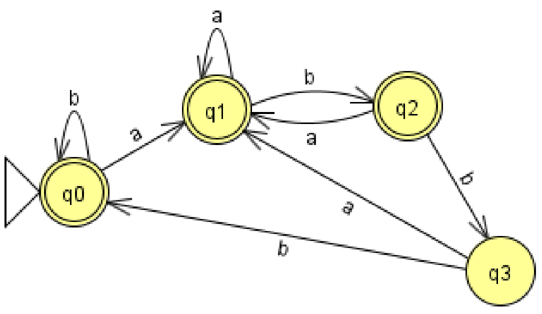
9) The set of all strings whose tenth symbol from the last end is a 1.



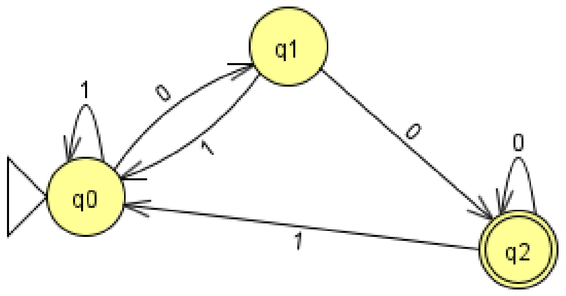
10) Design a DFA to accept string of a's and b's ending with 'abb' over ∑ = { a,b }



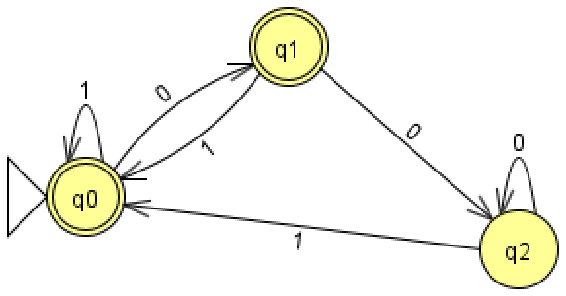
DFA to accept string of a's and b's not ending with 'abb' over ∑ = { a,b }



11) Give DFA's accepting the set of all strings ending in 00



The set of all strings not ending in 00



Non Deterministic Finite Automata

A nondeterministic finite automaton NFA has the power to be in several states at once.

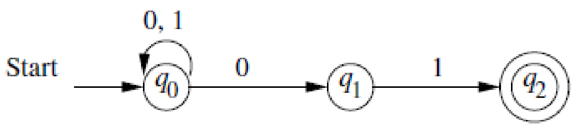
They are often more succinct and easier to design than DFA's.

An NFA is represented essentially like a DFA A = $(Q, \Sigma, \delta, q_0, F)$

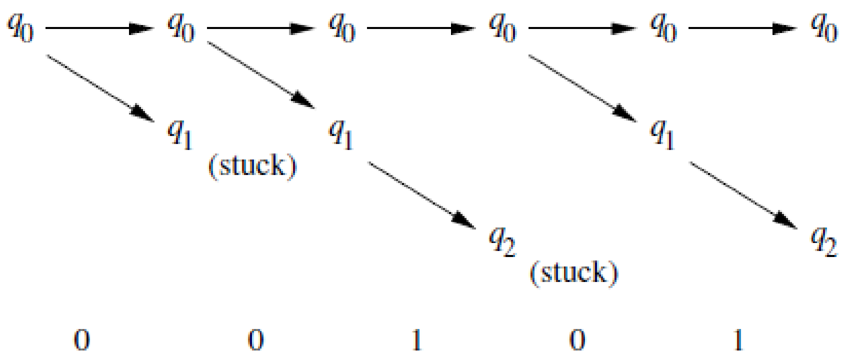
Where

- Q is a finite set of states
- Σ is a finite set of input symbols
- q_0 a member of Q, is the start state
- F is a subset of Q, is the set of final or accepting states
- δ, the transition function is a function that takes a state in Q and an input symbol in Σ as arguments and returns a subset of Q. Notice that the only difference between an NFA and a DFA is in the type of value that δ returns a set of states in the case of an NFA and a single state in the case of a DFA.

Example: An NFA accepting all strings that end in 01.



The states an NFA is in during the processing of input sequence 00101



The NFA can be specified formally as A=({q0,q1,q2},{0,1}, δ,q0,{q2})

Where the transition function is δ given by the transition table

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
$*q_2$	$\emptyset$	$\emptyset$

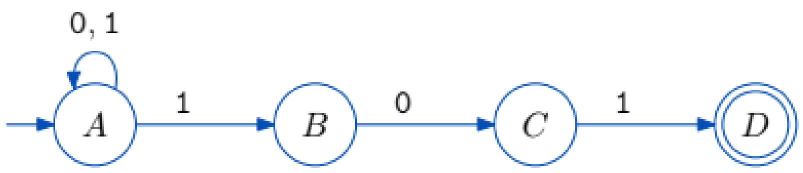
Let us use $\hat{\delta}$ to describe the processing of input 00101 by the NFA

A summary of the steps is

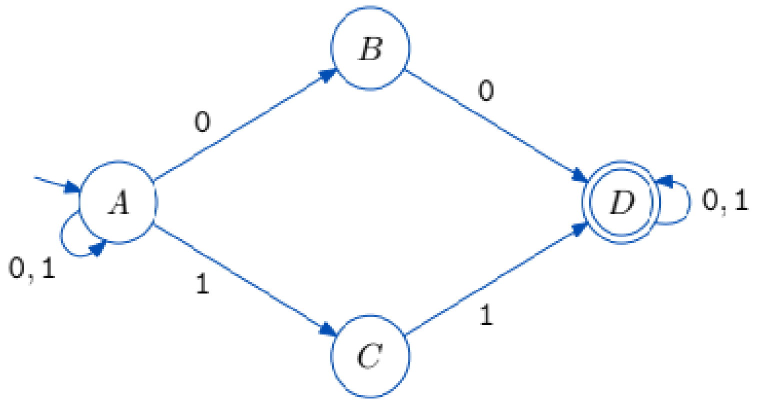
1. $\hat{\delta}(q_0, \epsilon) = \{q_0\}$.
2. $\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$.
3. $\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$.
4. $\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$.
5. $\hat{\delta}(q_0, 0010) = \delta(q_0, 0) \cup \delta(q_2, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$.
6. $\hat{\delta}(q_0, 00101) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$.

Examples:

1. Design an NFA that accepts all binary strings end with 101.

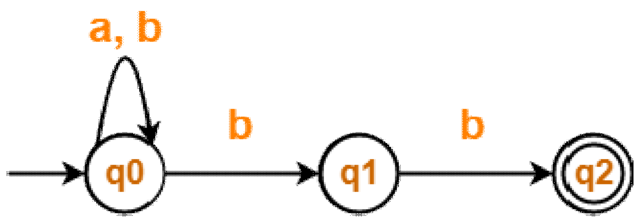


2. Design an NFA that accepts any binary string that contains 00 or 11 as a substring.



Equivalence of Deterministic and Nondeterministic Finite Automata

Example 1: Convert the following Non-Deterministic Finite Automata (NFA) to Deterministic Finite Automata (DFA).



Solution:

Transition table for the given Non-Deterministic Finite Automata (NFA) is

State / Alphabet	a	b
→q0	q0	q0, q1
q1	–	*q2
*q2	–	–

Let Q_D be a new set of states of the Deterministic Finite Automata (DFA).

Let T_D be a new transition table of the DFA.

Add transitions of start state q0 to the transition table T_D.

State / Alphabet	a	b
→q0	{q0}	{q0, q1}

New state present in state Q_D is {q0, q1}.

Add transitions for set of states {q0, q1} to the transition table T_D.

δ_D ({q0,q1}, a) = δ_N (q0,a) U δ_N (q1,a) = {q0} U Φ = **{q0}**

δ_D ({q0,q1}, b) = δ_N (q0,b) U δ_N (q1,b) = {q0,q1} U {q2} = **{q0,q1,q2}→New State**

New state present in state Q_D is {q0, q1, q2}.

Add transitions for set of states {q0, q1, q2} to the transition table T_D.

δ_D ({q0,q1,q2}, a) = δ_N (q0,a) U δ_N (q1,a) U δ_N (q2,a) = {q0} U Φ U Φ = **{q0}**

δ_D ({q0,q1,q2}, b) = δ_N (q0,b) U δ_N (q1,b) U δ_N (q2,b) = {q0,q1} U {q2} U Φ = **{q0,q1,q2}**

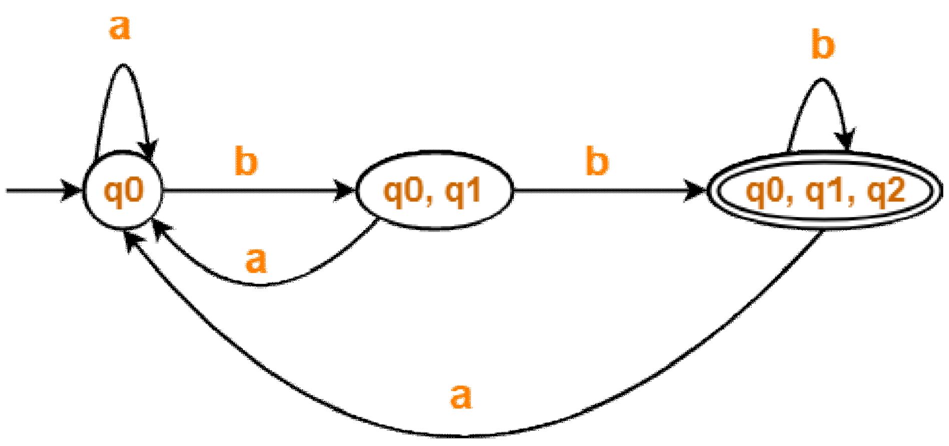
Since no new states are left to be added in the transition table T_D, so we stop.

States containing q2 as its component are treated as final states of the DFA.

Finally, Transition table for Deterministic Finite Automata (DFA) is-

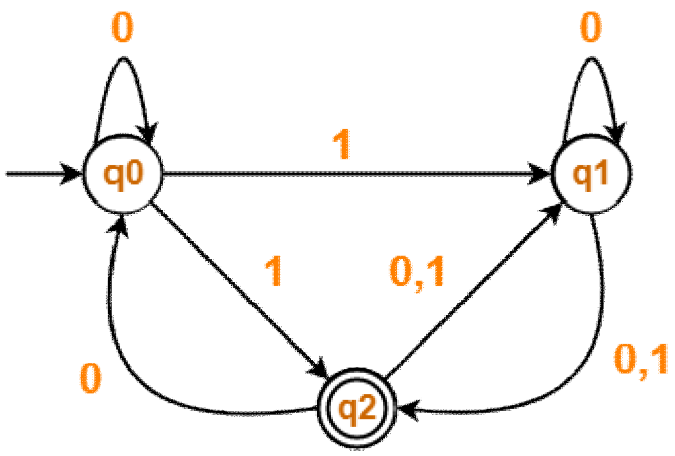
State / Alphabet	a	b
→q0	q0	{q0, q1}
{q0, q1}	q0	*{q0, q1, q2}
*{q0, q1, q2}	q0	*{q0, q1, q2}

Now, Deterministic Finite Automata (DFA) may be drawn as-



Deterministic Finite Automata (DFA)

2. Convert the following Non-Deterministic Finite Automata (NFA) to Deterministic Finite Automata (DFA).



Solution:

Transition table for the given Non-Deterministic Finite Automata (NFA) is-

State / Alphabet	0	1
→q0	q0	q1, *q2
q1	q1, *q2	*q2
*q2	q0, q1	q1

Let Q_D be a new set of states of the Deterministic Finite Automata (DFA).

Let T_D be a new transition table of the DFA.

Add transitions of start state q0 to the transition table T_D.

State / Alphabet	0	1
→q0	q0	{q1, q2}

New state present in state Q_D is {q1, q2}.

Add transitions for set of states {q1, q2} to the transition table T_D.

******Do the remaining steps like in the previous problem******

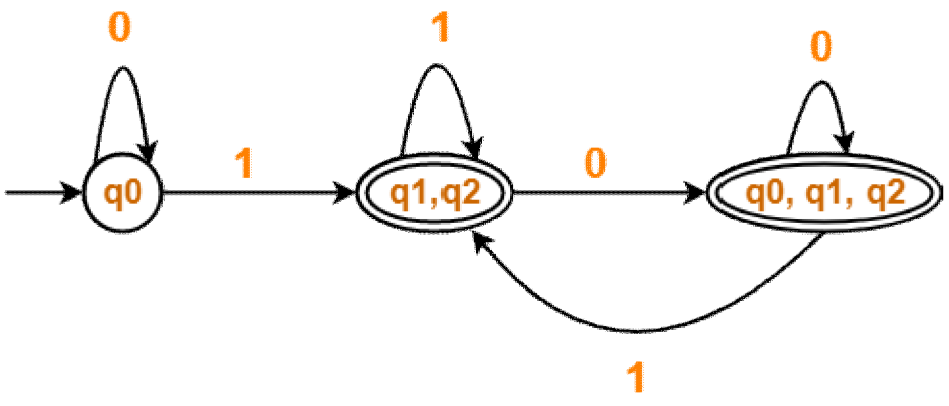
Since no new states are left to be added in the transition table T_D, so we stop.

States containing q2 as its component are treated as final states of the DFA.

Finally, Transition table for Deterministic Finite Automata (DFA) is-

State / Alphabet	0	1
→q0	q0	*{q1, q2}
*{q1, q2}	*{q0, q1, q2}	*{q1, q2}
*{q0, q1, q2}	*{q0, q1, q2}	*{q1, q2}

Now, Deterministic Finite Automata (DFA) may be drawn as-



Deterministic Finite Automata (DFA)

* **Exercise 2.3.1 :** Convert to a DFA the following NFA:

	0	1
$\rightarrow p$	$\{p,q\}$	$\{p\}$
q	$\{r\}$	$\{r\}$
r	$\{s\}$	$\emptyset$
$*s$	$\{s\}$	$\{s\}$

Exercise 2.3.2 : Convert to a DFA the following NFA:

	0	1
$\rightarrow p$	$\{q,s\}$	$\{q\}$
$*q$	$\{r\}$	$\{q,r\}$
r	$\{s\}$	$\{p\}$
$*s$	$\emptyset$	$\{p\}$

-

! Exercise 2.3.3: Convert the following NFA to a DFA and informally describe the language it accepts.

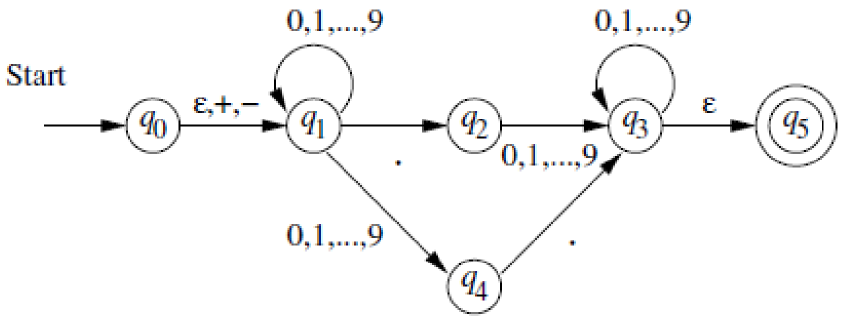
	0	1
$\rightarrow p$	$\{p, q\}$	$\{p\}$
q	$\{r, s\}$	$\{t\}$
r	$\{p, r\}$	$\{t\}$
$*s$	$\emptyset$	$\emptyset$
$*t$	$\emptyset$	$\emptyset$

Finite Automata with €- transitions (or) €- NFA

We shall now introduce another extension of the finite automaton. The new "feature" is that we allow a transition on €, the empty string.

An NFA is allowed to make a transition spontaneously, without receiving an input symbol.

We shall begin with an informal treatment of €-NFA's, using transition diagrams with € allowed as a label.

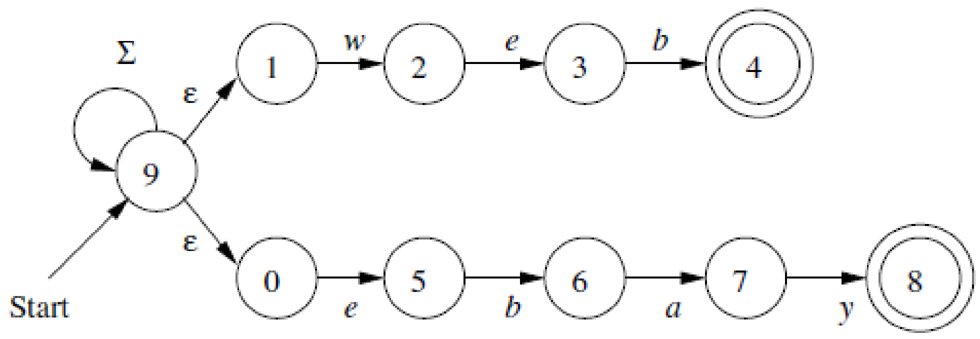


€-NFA that accepts decimal numbers consisting of

1. An optional + or − sign,
2. A string of digits,
3. A decimal point, and
4. Another string of digits. Either this string of digits, or the string (2) can be empty, but at least one of the two strings of digits must be nonempty.

For instance, the NFA recognizing the keywords web and eBay can also be implemented with €-transitions

Using €-transitions to help recognize keywords



The Formal Notation for an €-NFA

Formally, we represent an €-NFA A by A=(Q,∑,δ, q_0, F) where all components have their same interpretation as for an NFA, except that δ is now a function that takes as arguments:

$$\delta : Q \times \Sigma \cup \{ € \} \rightarrow 2^Q$$

Note: €, the symbol for the empty string, cannot be a member of the alphabet ∑, so no confusion results.

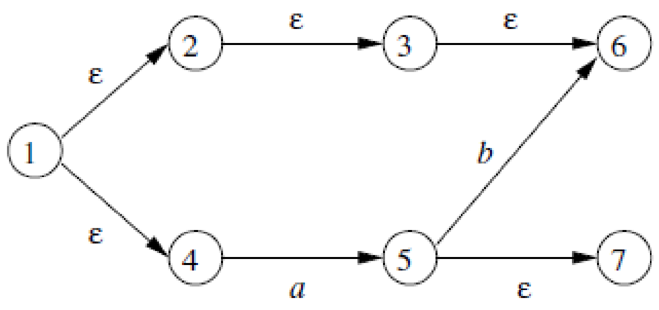
Epsilon-Closures

Epsilon (∈) – **closure** : **Epsilon closure** for a given state X is a set of states which can be reached from the states X with only (null) or ε moves including the state X itself.

BASIS: State q is in ECLOSE(q).

INDUCTION: If state p is in ECLOSE(q), and there is a transition from state p to state r labeled ϵ, then r is in ECLOSE(q). More precisely, if δ is the transition function of the ϵ-NFA involved, and p is in ECLOSE(q), then ECLOSE(q) also contains all the states in $\delta(p, \epsilon)$.

Example:



∈-closure(1) or ECLOSE(1) = { 1, 2, 3, 4, 6 } ∈-closure(2) or ECLOSE(2) = { 2, 3, 6 }

∈-closure(3) or ECLOSE(3) = { 3, 6 } ∈-closure(4) or ECLOSE(4) = { 4 }

∈-closure(5) or ECLOSE(5) = { 5, 7 } ∈-closure(6) or ECLOSE(6) = { 6 }

Conversion from €-NFA to DFA

Steps to Convert NFA with ε-move to DFA:

Step 1 : Take ∈ closure for the beginning state of NFA as beginning state of DFA.

Step 2 : Find the states that can be traversed from the present for each input symbol (union of transition value and their closures for each states of NFA present in current state of DFA).

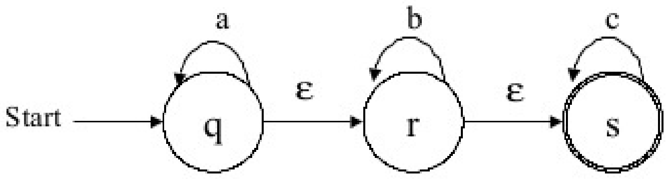
Step 3 : If any new state is found take it as current state and repeat step 2.

Step 4 : Do repeat Step 2 and Step 3 until no new state present in DFA transition table.

Step 5 : Mark the states of DFA which contains final state of NFA as final states of DFA.

Example:

Convert the following NFA with ε-move to a DFA



Step 1: Find ∈-closures

∈-closure (q) = { q, r, s }

∈-closure (r) = { r, s }

∈-closure (s) = { s }

Step 2: Find δ' for all new states

Now, we will obtain δ' transition for {q, r, s}

δ'({q,r,s},a) = ∈-closure(δ({q,r,s},a))

= ∈-closure(δ(q,a) U δ(r,a) U δ(s,a))

= ∈-closure(q U Φ U Φ)

= ∈-closure(q)

= { q, r, s }

δ'({q,r,s},b) = ∈-closure(δ({q,r,s},b))

= ∈-closure(δ(q,b) U δ(r,b) U δ(s,b))

= ∈-closure(Φ U r U Φ)

= ∈-closure(r)

= { r, s }

δ'({q,r,s},c) = ∈-closure(δ({q,r,s},c))

= ∈-closure(δ(q,c) U δ(r,c) U δ(s,c))

$= \in\text{-closure}(\Phi \cup \Phi \cup s)$

$= \in\text{-closure}(s)$

$= \{ s \}$

Now, we will obtain δ' transition for { r, s}

$\delta'(\{r,s\},a) = \in\text{-closure}(\delta(\{r,s\},a))$

$= \in\text{-closure}(\delta(r,a) \cup \delta(s,a))$

$= \in\text{-closure}(\Phi \cup \Phi)$

$= \in\text{-closure}(\Phi)$

$= \Phi$

$\delta'(\{r,s\},b) = \in\text{-closure}(\delta(\{r,s\},b))$

$= \in\text{-closure}(\delta(r,b) \cup \delta(s,b))$

$= \in\text{-closure}(r \cup \Phi)$

$= \in\text{-closure}(r)$

$= \{ r, s \}$

$\delta'(\{r,s\},c) = \in\text{-closure}(\delta(\{r,s\},c))$

$= \in\text{-closure}(\delta(r,c) \cup \delta(s,c))$

$= \in\text{-closure}(\Phi \cup s)$

$= \in\text{-closure}(s)$

$= \{ s \}$

Now, we will obtain δ' transition for {s}

$\delta'(\{s\},a) = \in\text{-closure}(\delta(s,a))$

$= \in\text{-closure}(\Phi)$

$= \Phi$

$\delta'(\{s\},b) = \in\text{-closure}(\delta(s,b))$

$= \in\text{-closure}(\Phi)$

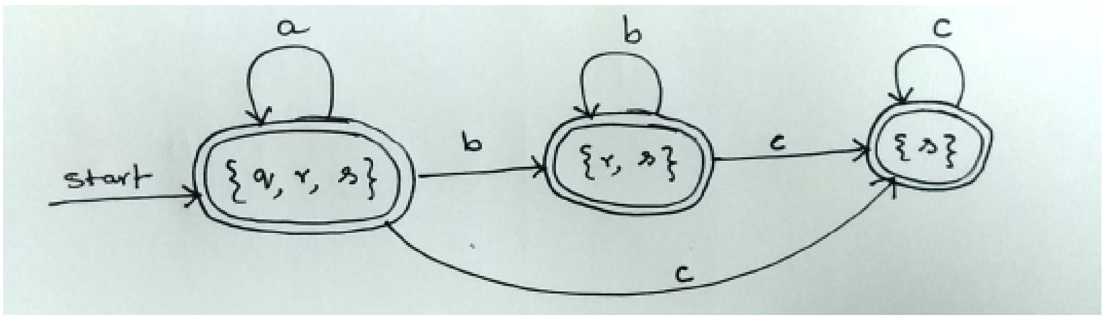
$= \Phi$

$\delta'(\{s\},c) = \in\text{-closure}(\delta(s,c))$

$= \in\text{-closure}(s)$

$= \{ s \}$

Hence, the equivalent DFA is:



* **Exercise 2.5.1 :** Consider the following ϵ-NFA.

	ϵ	a	b	c
$\rightarrow p$	$\emptyset$	$\{p\}$	$\{q\}$	$\{r\}$
q	$\{p\}$	$\{q\}$	$\{r\}$	$\emptyset$
$*r$	$\{q\}$	$\{r\}$	$\emptyset$	$\{p\}$

a) Compute the ϵ-closure of each state.

b) Give all the strings of length three or less accepted by the automaton.

c) Convert the automaton to a DFA.

Exercise 2.5.2 : Repeat Exercise 2.5.1 for the following ϵ-NFA:

	ϵ	a	b	c
$\rightarrow p$	$\{q, r\}$	$\emptyset$	$\{q\}$	$\{r\}$
q	$\emptyset$	$\{p\}$	$\{r\}$	$\{p, q\}$
$*r$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Equivalence and minimization of automata

DFA Minimization using Myhill-Nerode Theorem

Algorithm

Input − DFA

Output − Minimized DFA

Step 1 − Draw a table for all pairs of states (Q_i, Q_j) not necessarily connected directly [All are unmarked initially]

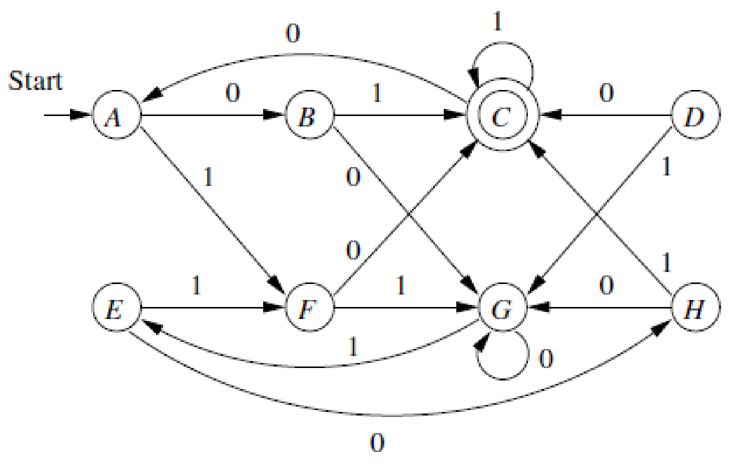
Step 2 − Consider every state pair (Q_i, Q_j) in the DFA where $Q_i \in F$ and $Q_j \notin F$ or vice versa and mark them. [Here F is the set of final states]

Step 3 − Repeat this step until we cannot mark anymore states −

If there is an unmarked pair (Q_i, Q_j), mark it if the pair {δ (Q_i, A), δ (Q_i, A)} is marked for some input alphabet.

Step 4 − Combine all the unmarked pair (Q_i, Q_j) and make them a single state in the reduced DFA.

Example: Construct the minimum-state equivalent DFA for the following

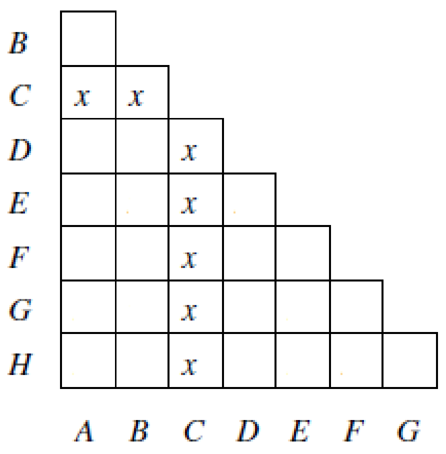


To find states that are equivalent, we make our best efforts to and pairs of states that are distinguishable. The algorithm, which we refer to as the **table-filling algorithm**, is a recursive discovery of distinguishable pairs in a DFA A =(Q,∑,δ,q_0, F).

BASIS: If p is an accepting state and q is nonaccepting, then the pair $\{p, q\}$ is distinguishable.

Let us execute the table-filling algorithm on the DFA, The final table is shown below where an **x** indicates pairs of distinguishable states, and the **blank squares indicate those pairs that have been found equivalent**.

For the basis, since C is the only accepting state, we put **x** in each pair that involves C.



B							
C	x	x					
D			x				
E			x				
F			x				
G			x				
H			x				
	A	B	C	D	E	F	G

Now, we know some distinguishable pairs, we can discover others.

Consider pair (A,B): $\left.\begin{array}{l}\delta(A,0) \rightarrow B\\ \delta(B,0) \rightarrow G\end{array}\right\}$ X $\left.\begin{array}{l}\delta(A,1) \rightarrow F\\ \delta(B,1) \rightarrow C\end{array}\right\}$ → Put X in (A,B)

Next Pair (A,D): $\left.\begin{array}{l}\delta(A,0) \rightarrow B\\ \delta(D,0) \rightarrow C\end{array}\right\}$ → Put X in (A,D) Pair

Next pair (A,E): $\left.\begin{array}{l}\delta(A,0) \rightarrow B\\ \delta(E,0) \rightarrow H\end{array}\right\}$ X $\left.\begin{array}{l}\delta(A,1) \rightarrow F\\ \delta(E,1) \rightarrow F\end{array}\right\}$ X

Next pair (A,F): $\left.\begin{array}{l}\delta(A,0) \rightarrow B\\ \delta(F,0) \rightarrow C\end{array}\right\}$ → Put X in (A,F) Pair

Next Pair (A,G): $\left.\begin{array}{l}\delta(A,0) \rightarrow B\\ \delta(G,0) \rightarrow G\end{array}\right\}$ X $\left.\begin{array}{l}\delta(A,1) \rightarrow F\\ \delta(G,1) \rightarrow E\end{array}\right\}$ X

Next Pair (AH): $\left.\begin{array}{l}\delta(A,0) \rightarrow B\\ \delta(H,0) \rightarrow G\end{array}\right\}$ X $\left.\begin{array}{l}\delta(A,1) \rightarrow F\\ \delta(H,1) \rightarrow C\end{array}\right\}$ →Put X in (A,H)

Continue like this for remaining pairs also....

i.e (B,D), (B,E), (B,F), (B,G), (B,H), (D,E), (D,F), (D,G), (D,H), (E,F), (E,G), (E,H), (F,G), (F,H), and (G,H).

Observe the above table (A,E), (A,G), (B,H), (D,F), and (E,G) pairs are empty.

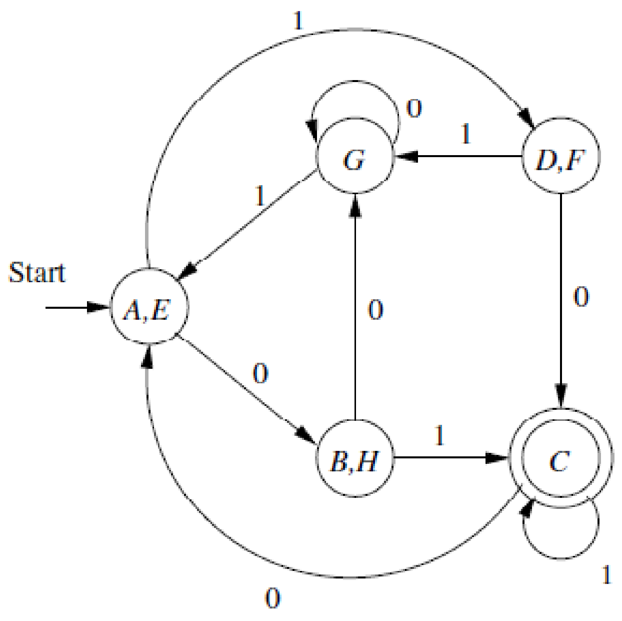
consider pair (A,G) $\left.\begin{array}{l}\delta(A,0) \rightarrow B\\ \delta(G,0) \rightarrow G\end{array}\right\}$ →(B,G) distinguishable pair or filled with 'X', So, pair (A,G) also distinguishable pair.

consider pair (E,G) $\left.\begin{array}{l}\delta(E,0)\rightarrow H\\ \delta(G,0)\rightarrow G\end{array}\right\}$ →(H,G) distinguishable pair or filled with 'X', So, pair (E,G) also distinguishable pair.

B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x
	A	B	C	D	E	F	G

Hence, (A,G), (B,H), and (D,F) are equivalent states.

So, Minimum State DFA is:



Exercise Problems:

	0	1
$\rightarrow A$	B	A
B	A	C
C	D	B
$*D$	D	A
E	D	F
F	G	E
G	F	G
H	G	D

Figure 4.14: A DFA to be minimized

4.4.5 Exercises for Section 4.4

* **Exercise 4.4.1:** In Fig. 4.14 is the transition table of a DFA.

a) Draw the table of distinguishabilities for this automaton.

b) Construct the minimum-state equivalent DFA.

Exercise 4.4.2: Repeat Exercise 4.4.1 for the DFA of Fig 4.15.

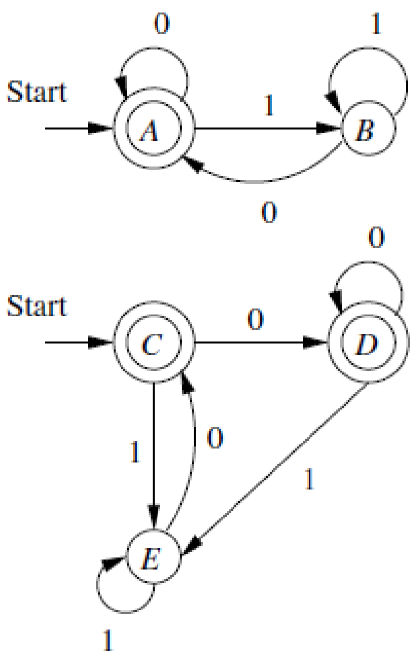
	0	1
$\rightarrow A$	B	E
B	C	F
$*C$	D	H
D	E	H
E	F	I
$*F$	G	B
G	H	B
H	I	C
$*I$	A	E

Figure 4.15: Another DFA to minimize

Testing Equivalence of Regular Languages (or) DFA’s

The table filling algorithm gives us an easy way to test if two regular languages are the same.

Example 1:



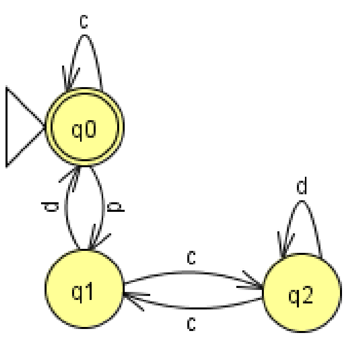
Now, test if the start states of the two original DFA’s are equivalent.

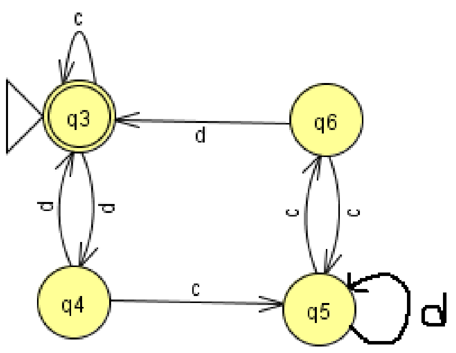
	0	1
{A,C}	{A,D}	{B,E}
{A,D}	{A,D}	{B,E}
{B,E}	{A,C}	{B,E}

Observe the pairs; both states are final states or non final states.

Hence, the given two DFA’s are equivalent.

Example: Test whether the given two DFA’s are equivalent or not





Regular Expressions and Languages

An algebraic description: the regular expression.

We shall find that regular expressions can define exactly the same languages that the various forms of automata describe the regular languages.

The Operators of Regular Expressions

Regular expressions denote languages.

For a simple example the regular expression $\mathbf{01^*+10^*}$ denotes the language consisting of all strings that are either a single 0 followed by any number of 1's or a single 1 followed by any number of 0's.

Before describing the regular expression notation, we need to learn the three operations on languages that the operators of regular expressions represent. These operations are:

1.The **union** of two languages L and M, denoted LUM, is the set of strings that are in either L or M, or both. For example, if L = {001, 10,111} and M= {€, 001} then LUM = {€, 10,001,111}.

2.The **concatenation** of languages L and M is the set of strings that can be formed by taking any string in L and concatenating it with any string in M. The concatenation of a pair of strings one string is followed by the other to form the result of the concatenation.We denote concatenation of languages either with a dot or with no operator at all, although the concatenation operator is frequently called "dot".

For example, if L = {001, 10,111} and M= {€, 001} then L.M or just LM= {001, 10, 111, 001001, 10001, 111001}.

3. The **closure** (or Star or Kleene closure) of a language L is denoted $\mathbf{L^*}$ and represents the set of those strings that can be formed by taking any number of strings from L, possibly with repetitions (i.e the same string may be selected more than once) and concatenating all of them.

$$L^* = L^0 \text{ U } L^1 \text{ U } L^2 \text{ U } L^3 \text{ U } L^4 \text{ U } L^5 \ldots\ldots$$

If L= {0, 1} then L^*= {€, 0, 1, 00, 01, 10, 11, 000, 001, 100, 010, 110, 011, 111,........}

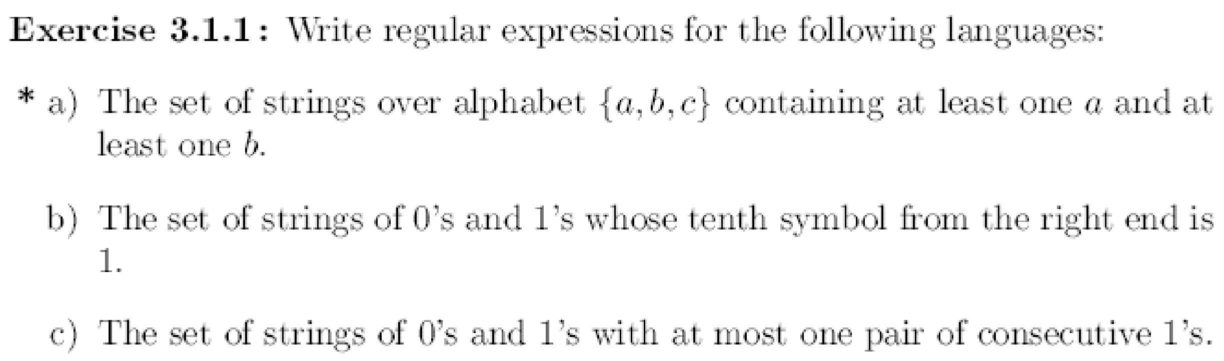
Precedence of Regular Expression Operators

For regular expressions, the following is the order of precedence for the operators:

1. The star operator is of highest precedence. That is, it applies only to the smallest sequence of symbols to its left that is a well formed regular expression.

2.Next in precedence comes the concatenation or "dot" operator.After grouping all stars to their operands, we group concatenation operators to their operands. That is, all expressions that are juxtaposed are grouped together.

3.Finally, all unions (+ operators) are grouped with their operands.Since union is also associative, it again matters little in which order consecutive unions are grouped, but we shall assume grouping from the left.



Exercise 3.1.1 : Write regular expressions for the following languages:

* a) The set of strings over alphabet $\{a, b, c\}$ containing at least one a and at least one b.

b) The set of strings of 0's and 1's whose tenth symbol from the right end is 1.

c) The set of strings of 0's and 1's with at most one pair of consecutive 1's.

Ans a) $(a+b+c)^*$ a $(a+b+c)^*$ b $(a+b+c)^*$

Ans b) $(0+1)^*$ 1 $(0+1)^+$ $(0+1)^+$ $(0+1)^+$ $(0+1)^+$ $(0+1)^+$ $(0+1)^+$ $(0+1)^+$ $(0+1)^+$ $(0+1)^+$

Ans c) $(0+1)^*$ 11 $(0+1)^*$

Algebraic Laws for Regular Expressions

1.Associativity and Commutativity

Commutativity is the property of an operator that says we can switch the order of its operands and get the same result.

Associativity is the property of an operator that allows us to regroup the operands when the operator is applied twice.

Here are three laws of these types that hold for regular expressions:

- L +M =M +L This law, the commutative law for union, says that we may take the union of two languages in either order.
- (L + M) + N = L + (M + N) This law, the associative law for union, says that we may take the union of three languages either by taking the union of the first two initially or taking the union of the last two initially.
- (LM)N = L(MN) This law, the associative law for concatenation, says that we can concatenate three languages by concatenating either the first two or the last two initially.

2.Identities and Annihilators

An identity for an operator is a value such that when the operator is applied to the identity and some other value, the result is the other value.

An annihilator for an operator is a value such that when the operator is applied to the annihilator and some other value, the result is the annihilator.

There are three laws for regular expressions involving these concepts we list them below.

- Φ +L = L + Φ = L, This law asserts that Φ is the identity for union.
- €L = L€ = L, This law asserts that € is the identity for concatenation.

- Φ L = L Φ = Φ. This law asserts that Φ is the annihilator for concatenation.

These laws are powerful tools in simplifications.

3.Distributive Laws

A distributive law involves two operators, and asserts that one operator can be pushed down to be applied to each argument of the other operator individually.

These laws are:

- $L(M + N) = LM + LN$. This law, is the *left distributive law of concatenation over union.*
- $(M + N)L = ML + NL$. This law, is the *right distributive law of concatenation over union.*

4.The Idempotent Law

An operator is said to be idempotent if the result of applying it to two of the same values as arguments is that value.

- $L + L = L$. This law, the *idempotence law for union*, states that if we take the union of two identical expressions, we can replace them by one copy of the expression.

5. Laws Involving Closures

There are a number of laws involving the closure operators

- $(L^*)^* = L^*$.
- $\emptyset^* = \epsilon$.
- $\epsilon^* = \epsilon$.
- $L^+ = LL^* = L^*L$.
- $L^* = L^+ + \epsilon$.
- $L? = \epsilon + L$.

Let ***r*, *r*1, *r*2**, and ***r*3** be any regular expressions

1. $r\varepsilon \approx \varepsilon r \approx r$.
2. $r_1 r_2 \not\approx r_2 r_1$, in general.
3. $r_1(r_2 r_3) \approx (r_1 r_2) r_3$.
4. $r\varnothing \approx \varnothing r \approx \varnothing$.
5. $\varnothing^* \approx \varepsilon$.
6. $\varepsilon^* \approx \varepsilon$.
7. If $\varepsilon \in L(r)$, then $r^* \approx r^+$.
8. $rr^* \approx r^* r \approx r^+$.
9. $(r_1 + r_2) r_3 \approx r_1 r_3 + r_2 r_3$.
10. $r_1(r_2 + r_3) \approx r_1 r_2 + r_1 r_3$.
11. $(r^*)^* \approx r^*$.
12. $(r_1 r_2)^* r_1 \approx r_1 (r_2 r_1)^*$.
13. $(r_1 + r_2)^* \approx (r_1^* r_2^*)^*$.

The Pumping Lemma for Regular Languages

Theorem 4.1: (The *pumping lemma for regular languages*) Let L be a regular language. Then there exists a constant n (which depends on L) such that for every string w in L such that $|w| \geq n$, we can break w into three strings, $w = xyz$, such that:

1. $y \neq \epsilon$.
2. $|xy| \leq n$.
3. For all $k \geq 0$, the string $xy^k z$ is also in L.

That is, we can always find a nonempty string y not too far from the beginning of w that can be "pumped"; that is, repeating y any number of times, or deleting it (the case $k = 0$), keeps the resulting string in the language L.

PROOF: Suppose L is regular. Then $L = L(A)$ for some DFA A. Suppose A has n states. Now, consider any string w of length n or more, say $w = a_1 a_2 \cdots a_m$, where $m \geq n$ and each a_i is an input symbol. For $i = 0, 1, \ldots, n$ define state p_i to be $\hat{\delta}(q_0, a_1 a_2 \cdots a_i)$, where δ is the transition function of A, and q_0 is the start state of A. That is, p_i is the state A is in after reading the first i symbols of w. Note that $p_0 = q_0$.

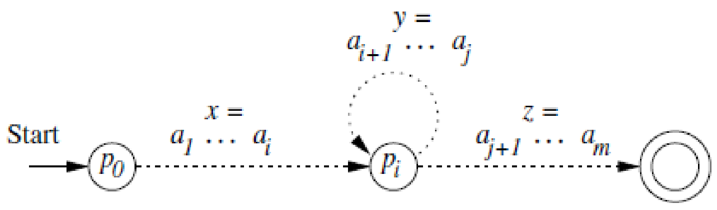
By the pigeonhole principle, it is not possible for the $n+1$ different p_i's for $i = 0, 1, \ldots, n$ to be distinct, since there are only n different states. Thus, we can find two different integers i and j, with $0 \leq i < j \leq n$, such that $p_i = p_j$. Now, we can break $w = xyz$ as follows:

1. $x = a_1 a_2 \cdots a_i$.

2. $y = a_{i+1} a_{i+2} \cdots a_j$.

3. $z = a_{j+1} a_{j+2} \cdots a_m$.

That is, x takes us to p_i once; y takes us from p_i back to p_i (since p_i is also p_j), and z is the balance of w. The relationships among the strings and states are suggested by Fig. 4.1. Note that x may be empty, in the case that $i = 0$. Also, z may be empty if $j = n = m$. However, y can not be empty, since i is strictly less than j.



Now, consider what happens if the automaton A receives the input xy^kz for any $k \geq 0$. If $k = 0$, then the automaton goes from the start state q_0 (which is also p_0) to p_i on input x. Since p_i is also p_j, it must be that A goes from p_i to the accepting state shown in Fig. 4.1 on input z. Thus, A accepts xz.

If $k > 0$, then A goes from q_0 to p_i on input x, circles from p_i to p_i k times on input y^k, and then goes to the accepting state on input z. Thus, for any $k \geq 0$, xy^kz is also accepted by A; that is, xy^kz is in L. □

Applications of the Pumping Lemma

- We shall propose a language and use the pumping lemma to prove that the language is not regular.
- The Pumping Lemma as an Adversarial Game

Example Problems:

1. Prove that the L= { 0^i 1^i / i>=1 } is not regular.

Solution:

The given language generates L= { 01, 0011, 000111, 00001111, 0000011111, 000000111111,.........}

Let w= 0^n 1^n such that |w| = 2n.

By pumping lemma we can write w=xyz such that |xy| <=n and |y| != 0

Now if xy^iz € L then the language L is said to be regular.

There are many cases:

i) Y has only 0’s

ii) Y has only 1’s

CASE i : If Y has only 0's then the string

Consider the string w= 0011

x=0 , y=0 and z=11

then $xy^iz = 0\ 0^i\ 11$

If i=2 then 00011 not belongs to L

CASE i i : If Y has only 1's then the string

Consider the string w= 0011

x=00 , y=1 and z=1

then $xy^iz = 0\ 0\ 1^i\ 1$

If i=2 then 00111 not belongs to L

Hence, form all these 2 cases it is clear that language L is not regular.

2. Prove that Language L = { 0^n10^n n>=1} is not regular.

The given language generates L={010,00100,0001000,000010000,.........}

By pumping lemma we can write w=xyz such that |xy| <=n and |y| != 0

Now if xy^iz € L then the language L is said to be regular.

Consider the string from L

w= 0001000= xyz

x=0; y=01; z=000

w= xy^iz= 00 $(01)^i$ 000

If i= 2 then 000101000 not belongs to L

Hence, the given language is not regular.

3. Prove that Language L = { 0^n1^{2n} n>=1} is not regular.

The language L={ 011,001111,000111111,000011111111,........}

By pumping lemma we can write w=xyz such that |xy| <=n and |y| != 0

Now if xy^iz € L then the language L is said to be regular.

Consider the string from L

w=001111=xyz

x=00 y=11 z=11

xy^iz= 00 $(11)^i$ 11

If i=2 then 00111111 not belongs to L

Hence ,the given language is not regular.

Exercise 4.1.1: Prove that the following are not regular languages.

a) $\{0^n1^n \mid n \geq 1\}$. This language, consisting of a string of 0's followed by an equal-length string of 1's, is the language L_{01} we considered informally at the beginning of the section. Here, you should apply the pumping lemma in the proof.

b) The set of strings of balanced parentheses. These are the strings of characters "(" and ")" that can appear in a well-formed arithmetic expression.

* c) $\{0^n10^n \mid n \geq 1\}$.

d) $\{0^n1^m2^n \mid n \text{ and } m \text{ are arbitrary integers}\}$.

e) $\{0^n1^m \mid n \leq m\}$.

f) $\{0^n1^{2n} \mid n \geq 1\}$.

! **Exercise 4.1.2:** Prove that the following are not regular languages.

* a) $\{0^n \mid n \text{ is a perfect square}\}$.

b) $\{0^n \mid n \text{ is a perfect cube}\}$.

c) $\{0^n \mid n \text{ is a power of } 2\}$.

d) The set of strings of 0's and 1's whose length is a perfect square.

e) The set of strings of 0's and 1's that are of the form ww, that is, some string repeated.

Closure Properties of Regular Languages

We shall prove several theorems of the form "if certain languages are regular, and a language L is formed from them by certain operations(e.g. L is the union of two regular languages), then L is also regular" These theorems are often called closure properties of the regular languages.

Here is a summary of the principal closure properties for regular languages:

- The union of two regular languages is regular
- The intersection of two regular languages is regular
- The complement of a regular language is regular
- The difference of two regular languages is regular
- The reversal of a regular language is regular

- The closure (star) of a regular language is regular
- The concatenation of regular languages is regular
- A homomorphism (substitution of strings for symbols) of a regular language is regular
- The inverse homomorphism of a regular language is regular

1. Closure under union

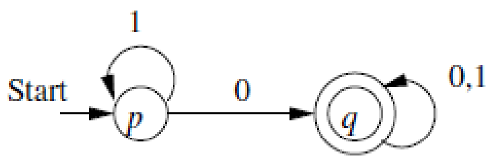
If L and M are regular languages, then so is LUM

Proof: Since L and M are regular, they have regular expressions say L= L(R) and M= L(S) Then LUM =L(R+ S) by the definition of the + operator for regular expressions.

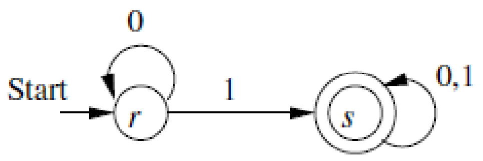
2. Closure under Intersection

Let L and M be the languages of regular expressions R and S, respectively then it a regular expression whose language is L∩M.

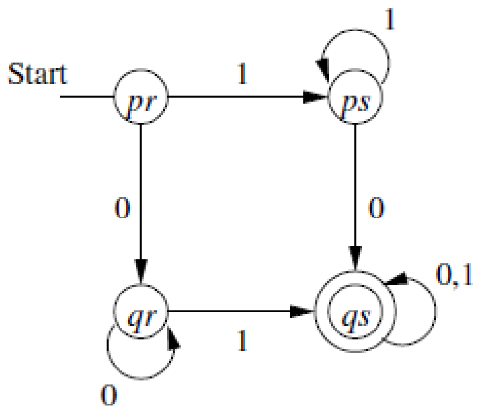
proof: Let A and B be DFA’s whose languages are L and M, respectively. Construct C, the product automaton of A and B make the final states of C are the pairs consisting of final states of both A and B.



(a)



(b)

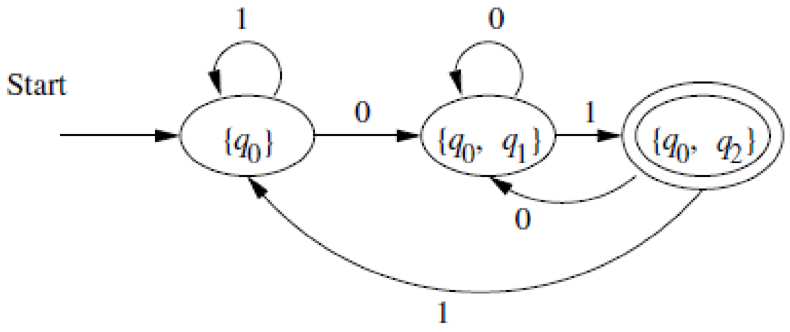


(c)

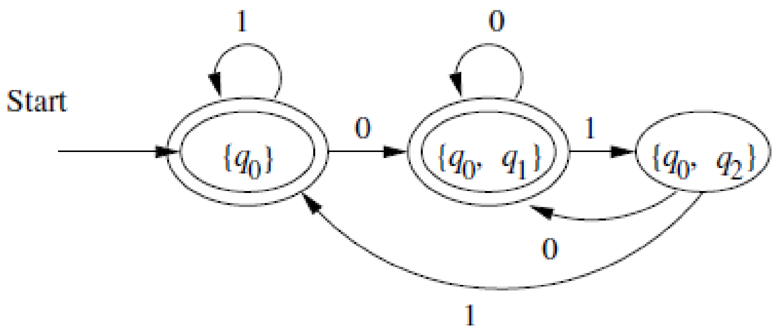
3. Closure under Complementation

The complement of a language L (with respect to an alphabet E such that E^* contains L) is E^* –L. Since E^* is surely regular, the complement of a regular language is always regular.

DFA:



Its Complementation:



4. Closure under Difference

If L and M are regular languages, then so is L – M = strings in L but not M.

Proof: Let A and B be DFA's whose languages are L and M, respectively. Construct C, the product automaton of A and B make the final states of C be the pairs, where A-state is final but B-state is not.

5. Closure under reversal

Given language L, L^R is the set of strings whose reversal is in L

Example: L = {0, 01, 100};

L^R ={0, 10, 001}.

Proof: Let E be a regular expression for L. We show how to reverse E, to provide a regular expression E^R for L^R.

6. Closure under Star (Closure)

Kleen Closure

RS is a regular expression whose language is L, M. R* is a regular expression whose language is L*.

Positive closure

RS is a regular expression whose language is L, M. R^+ is a regular expression whose language is L^+.

7. Closure under Concatenation

If r_1 and r_2 are regular expressions denoting L_1 and L_2 respectively, then L_1L_2 is denoted by the regular expression r_1r_2 and hence itself regular.

8. Closure under Homomorphism

A string homomorphism is a function on strings that works by substituting a particular string for each symbol.

Example 4.13: The function h defined by $h(0) = ab$ and $h(1) = \epsilon$ is a homomorphism. Given any string of 0's and 1's, it replaces all 0's by the string ab and replaces all 1's by the empty string. For example, h applied to the string 0011 is $abab$. □

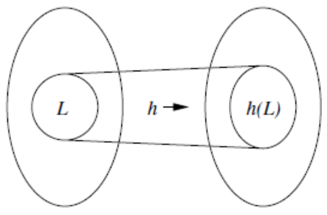
Formally, if h is a homomorphism on alphabet Σ, and $w = a_1a_2\cdots a_n$ is a string of symbols in Σ, then $h(w) = h(a_1)h(a_2)\cdots h(a_n)$. That is, we apply h to each symbol of w and concatenate the results, in order. For instance, if h is the homomorphism in Example 4.13, and $w = 0011$, then

$h(w) = h(0)h(0)h(1)h(1) = (ab)(ab)(\epsilon)(\epsilon) = abab$, as we claimed in that example.

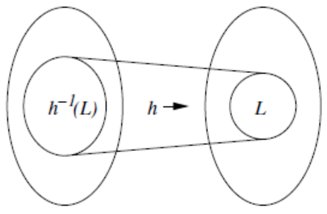
9. Closure under Inverse Homomorphism

Let h be a homomorphism and L a language whose alphabet is the output language of h.

(L) = {w | h(w) is in L}.



(a)



(b)