Scheme of Teaching and Examionation

B.E. V Semester Computer Science & Engineering

S. No	Board of Study	Subject Code	Subject Name	Periods per week			Scheme of exam			Total Marks	Credit L+(T+P) / 2
				L	T	P	Theory / Practical				
							ESE	CT	TA		
4	Comp Science & Engg	322514(22)	Theory of Computation	3	1	-	80	20	20	120	4

Syllabus

CHHATTISGARH SWAMI VIVEKANAND TECHNICAL UNIVERSITY, BHILAI (C.G.)

Semester – B.E. V
Subject: **Theory of Computation**
Total theory periods-**40**
Total marks in end semester exam **– 80**
Minimum number of class tests to be conducted **– 02**

Branch-**Computer Science & Engineering.**
Code **–322514 (22)**
Total Tutorial Periods: **12**

UNIT-1. THE THEORY OF AUTOMATA :

Introduction to automata theory, Examples of automata machine, Finite automata as a language acceptor and translator. Deterministic finite automata. Non deterministic finite automata, finite automata with output (Mealy Machine. Moore machine). Finite automata with ? moves, Conversion of NFA to DFA by Arden's method, Minimizing number of states of a DFA. My hill Nerode theorem, Properties and limitation of FSM. Two way finite automata. Application of finite automata.

UNIT-2. REGULAR EXPRESSIONS :

Regular expression, Properties of Regular Expression. Finite automata and Regular expressions. Regular Expression to DFA conversion & vice versa. Pumping lemma for regular sets. Application of pumping lemma, Regular sets and Regular grammar. Closure properties of regular sets. Decision algorithm for regular sets and regular grammar.

Syllabus

UNIT-3. GRAMMARS.

Definition and types of grammar. Chomsky hierarchy of grammar. Relation between types of grammars. Role and application areas of grammars. Context free grammar. Left most linear & right most derivation trees. Ambiguity in grammar. Simplification of context free grammar. Chomsky normal from. Greibach normal form, properties of context free language. Pumping lemma from context free language. Decision algorithm for context tree language.

UNIT-4. PUSH DOWN AUTOMATA AND TURING MACHINE.

Basic definitions. Deterministic push down automata and non deterministic push down automata. Acceptance of push down automata. Push down automata and context free language. Turing machine model. Representation of Turing Machine Construction of Turing Machine for simple problem's. Universal Turing machine and other modifications. Church's Hypothesis. Post correspondence problem. Halting problem of Turing Machine

UNIT-5 COMPUTABILITY

Introduction and Basic concepts. Recursive function. Partial recursive function. Partial recursive function. Initial functions, computability, A Turing model for computation. Turing computable functions, Construction of Turing machine for computation. Space and time complexity. Recursive enumerable language and sets.

Text Books

(1) Theory of Computer Science (Automata Language & Computation), K.L.P. Mishra and N. Chandrasekran, PHI.

(2) Introduction to Automata theory. Language and Computation, John E. Hopcropt & Jeffery D. Ullman, Narosa Publishing House.

Reference Books

(1) Theory of Automata and Formal Language, R.B. Patel & P. Nath, Umesh Publication.
(2) An Indtroduction and finite automata theory, Adesh K. Pandey, TMH.
(3) Theory of Computation, AM Natrajan. Tamilarasi, Bilasubramani, New Age International Publishers.

Unit-I
Theory of Automata

Introduction to automata theory, Examples of automata machine, Finite automata as a language acceptor and translator. Deterministic finite automata. Non deterministic finite automata, finite automata with output (Mealy Machine. Moore machine). Finite automata with ? moves, Conversion of NFA to DFA by Arden's method, Minimizing number of states of a DFA. My hill Nerode theorem, Properties and limitation of FSM. Two way finite automata. Application of finite automata.

Introduction to Automata Theory

1 DEFINITION OF AN AUTOMATON

'e shall give the most general definition of an automaton and later modify it to omputer applications. An automaton is defined as a system where energy, materials id information are transformed, transmitted and used for performing some functions ithout direct participation of man. Examples are automatic machine tools, automatic acking machines, and automatic photo printing machines.

In Computer Science the term 'automaton' means "discrete automaton" and is efined in a more abstract way as shown in Fig. 2.1.

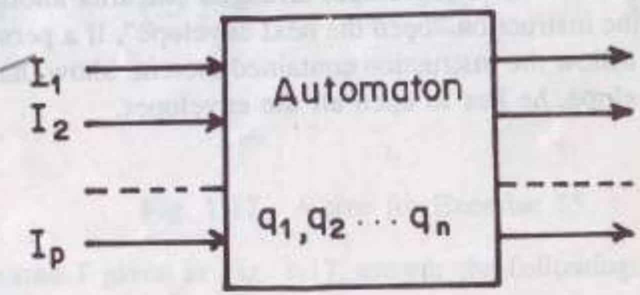


Fig. 2.1 Model of a discrete automaton.

Introduction to Automata Theory

Its characteristics are now described.

(i) *Input.* At each of the discrete instants of time $t_1, t_2, \ldots,$ input values $I_1, I_2, \ldots,$ each of which can take a finite number of fixed values from the input alphabet Σ, are applied to the input side of model shown in Fig. 2.1.

(ii) *Output.* $O_1, O_2, \ldots, O_q$ are the outputs of the model, each of which can take finite numbers of fixed values from an output O.

(iii) *States.* At any instant of time the automaton can be in one of the states $q_1, q_2, \ldots, q_n$.

(iv) *State relation.* The next state of an automaton at any instant of time is determined by the present state and the present input.

(v) *Output relation.* Output is related to either state only or to both the input and the state. It should be noted that at any instant of time the automaton is in some state. On 'reading' an input symbol, the automaton moves to a next state which is given by the state relation.

NOTE: An automaton in which the output depends only on the input is called an automaton without a memory. An automaton in which the output depends on the states also is called automaton with a finite memory. An automaton in which the output depends only on the states of the machine is called a *Moore machine*. An automaton in which the output depends on the state and the input at any instant of time is called a *Mealy machine*.

Introduction to Automata Theory

efinition 2.1 Analytically, a finite automaton can be represented by a 5-tuple , Σ, δ, q_0, F), where

(i) Q is a finite nonempty set of states;

(ii) Σ is a finite nonempty set of inputs called input alphabet;

(iii) δ is a function which maps $Q \times \Sigma$ into Q and is usually called direct nsition function. This is the function which describes the change of states ring the transition. This mapping is usually represented by a transition table a transition diagram.

(iv) $q_0 \in Q$ is the initial state; and

(v) $F \subseteq Q$ is the set of final states. It is assumed here that there may be ore than one final state.

TE: The transition function which maps $Q \times \Sigma^*$ into Q (i.e. maps a state and tring of input symbols including the empty string into a state) is called indirect nsition function. We shall use the same symbol δ to represent both types of nsition functions and the difference can be easily identified by nature of mapping ymbol or a string), i.e. by the argument. δ is also called the next state function. e above model can be represented graphically by Fig. 2.4.

Introduction to Automata Theory

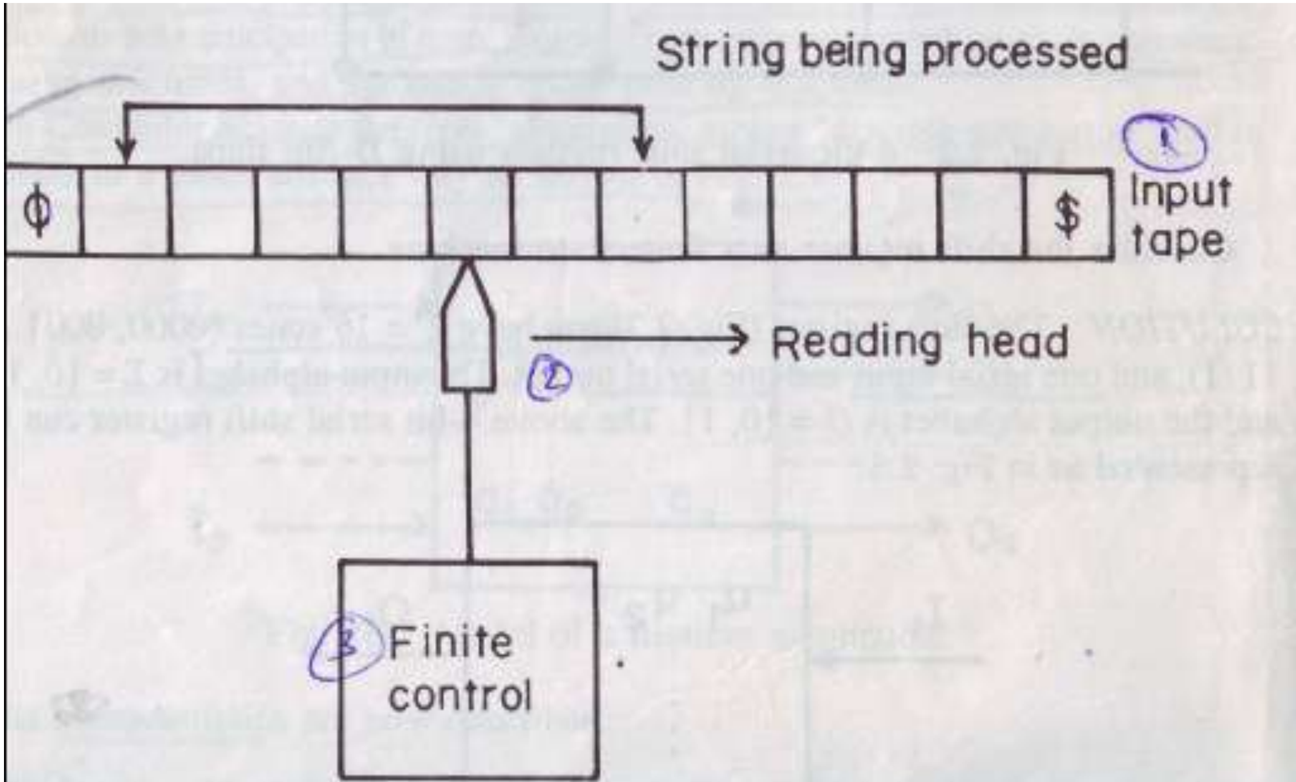


Fig. 2.4 Block diagram of a finite automaton.

Figure 2.4 is the block diagram for a finite automaton. The various components
e explained as follows:

Introduction to Automata Theory

(i) *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 end-markers ₵ at the left end and $ at the right end. Absence of end-markers indicates that the tape is of infinite length. The left-to-right sequence of symbols between the end-markers is the input string to be processed.

(ii) *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 R-head only to the right side.

(iii) *Finite control.* The input to the finite control will be usually: symbol under the R-head, say a, or the present state of the machine, say q, to give the following outputs: (a) A motion of R-head along the tape to the next square (In some a null move, i.e. R-head remaining to the same square is permitted); (b) the next state of the finite state machine given by $\delta(q, a)$.

Transition System

A transition graph or a transition system is a finite directed labelled graph in which each vertex (or node) represents a state and the directed edges indicate the transition of a state and the edges are labelled with input/output.

A typical transition system is shown in Fig. 2.5. In the figure, the initial state is represented by a circle with an arrow pointing towards it, the final state by two

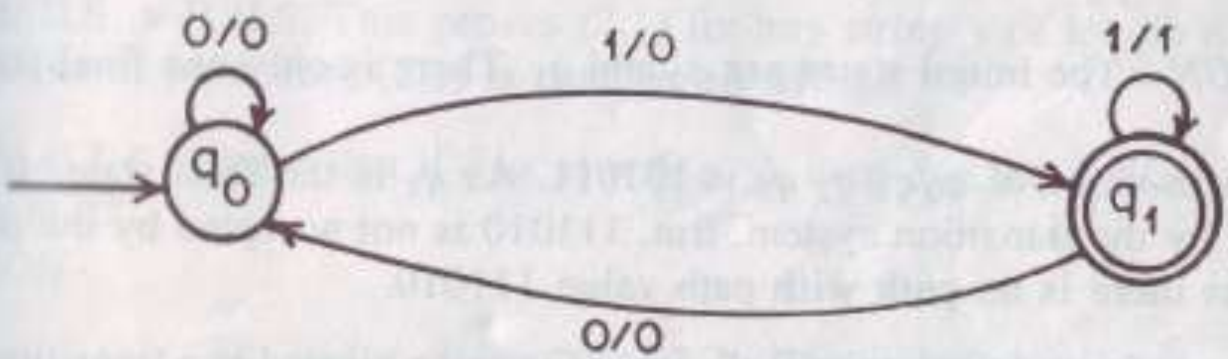


Fig. 2.5 A transition system.

concentric circles, and the other states are represented by just a circle. The edges are labelled by input/output (e.g. by 1/0 or 1/1). For example, if the system is in state q_0 and the input 1 is applied, the system moves to state q_1 as there is a directed edge from q_0 to q_1 with label 1/0. It outputs 0.

Property of Transition Function

perty 1 $\delta(q, \Lambda) = q$ in a finite automaton. This means the state of the tem can be changed only by an input symbol.

perty 2 For all strings w and input symbols a,

$$\delta(q, aw) = \delta(\delta(q, a), w)$$

$$\delta(q, wa) = \delta(\delta(q, w), a)$$

s property gives the state after the automaton consumes or reads the first ıbol of a string aw and the state after the automaton consumes a prefix of the ng wa.

Acceptability of String by FA

Definition 2.4 A string x is accepted by a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ if $\delta(q_0, x) = q$ for some $q \in F$. This is basically the acceptability of a string by the final state.

Example

States	Inputs 0	Inputs 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

OLUTION

$$\delta(q_0, 110101) = \delta(q_1, 10101)$$
$$= \delta(q_0, 0101)$$
$$= \delta(q_2, 101)$$
$$= \delta(q_3, 01)$$
$$= \delta(q_1, 1)$$
$$= \delta(q_0, \Lambda) = q_0$$

ence,

$$q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_0 \xrightarrow{0} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_1 \xrightarrow{1} q_0$$

ie symbol ↓ indicates the current input symbol being processed by the machine.

Types of Automata

Two Types

1.Automata without output

I. DFA (Deterministic Finite Automata)

II. NFA(Nondeterministic Finite Automata)

a. NFA without ϵ(or Λ)

b. NFA with ϵ(or Λ)

2. Automata with output

I. Mealy Machine

II. Moore Machine

DFA

finition 2.1 Analytically, a finite automaton can be represented by a 5-tuple , Σ, δ, q_0, F), where

(i) Q is a finite nonempty set of states;

(ii) Σ is a finite nonempty set of inputs called input alphabet;

(iii) δ is a function which maps $Q \times \Sigma$ into Q and is usually called direct nsition function. This is the function which describes the change of states ring the transition. This mapping is usually represented by a transition table a transition diagram.

(iv) $q_0 \in Q$ is the initial state; and

(v) $F \subseteq Q$ is the set of final states. It is assumed here that there may be ore than one final state.

NFA(NFA without ϵ)

Definition 2.5 A nondeterministic finite automaton (NDFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

(i) Q is a finite nonempty set of states;
(ii) Σ is a finite nonempty set of inputs;
(iii) δ is the transition function mapping from $Q \times \Sigma$ into 2^Q which is the power set of Q, the set of all subsets of Q;
(iv) $q_0 \in Q$ is the initial state; and
(v) $F \subseteq Q$ is the set of final states.

NFA(NFA without ϵ)
Example

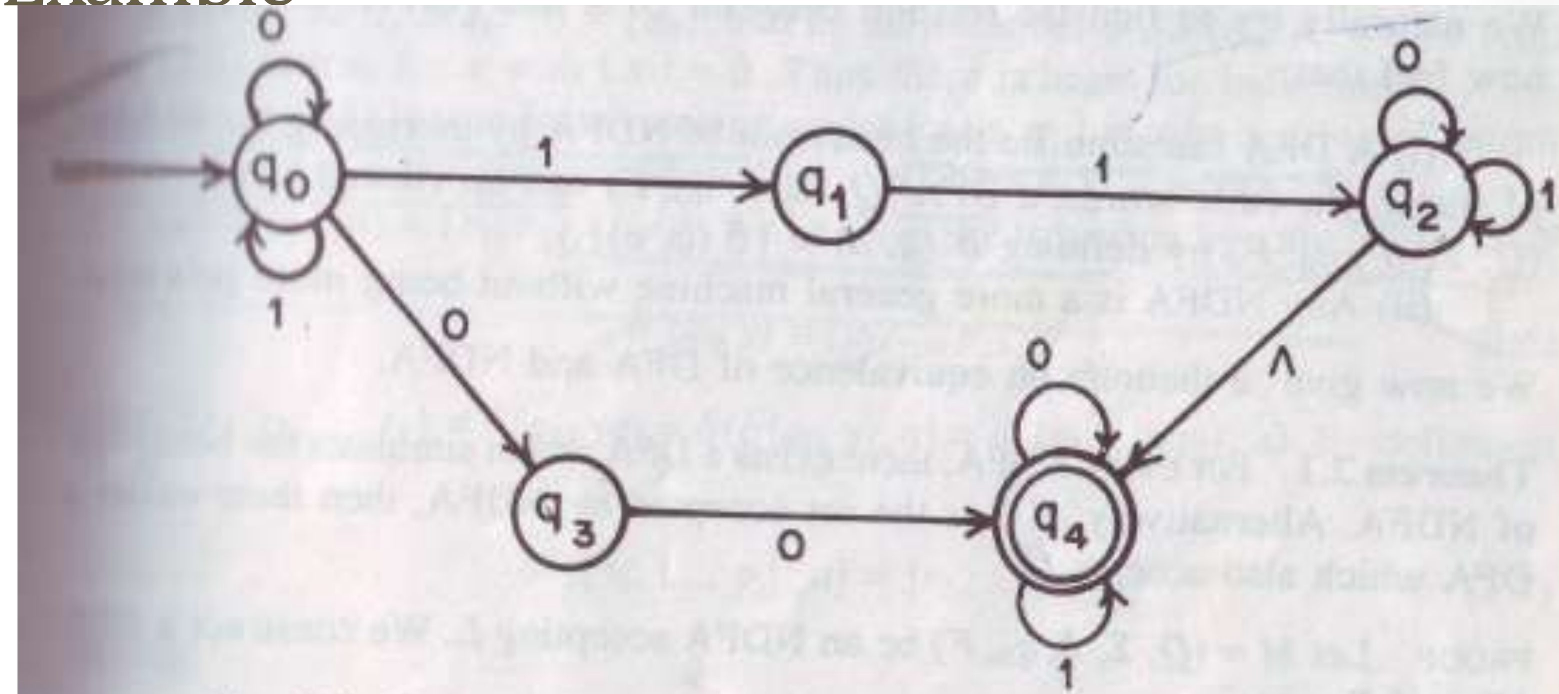


Fig. 2.8 Transition system for a nondeterministic automaton.

The sequence of states for the input string 0100 is given in Fig. 2.9. Hence,

$$\delta(q_0, 0100) = \{q_0, q_3, q_4\}$$

Since q_4 is an accepting state, the input string 0100 will be accepted by the nondeterministic automaton.

Acceptability in NFA

Definition 2.6 A string $w \in \Sigma^*$ is accepted by NDFA M if $\delta(q_0, w)$ contains some final state.

Acceptability in NFA

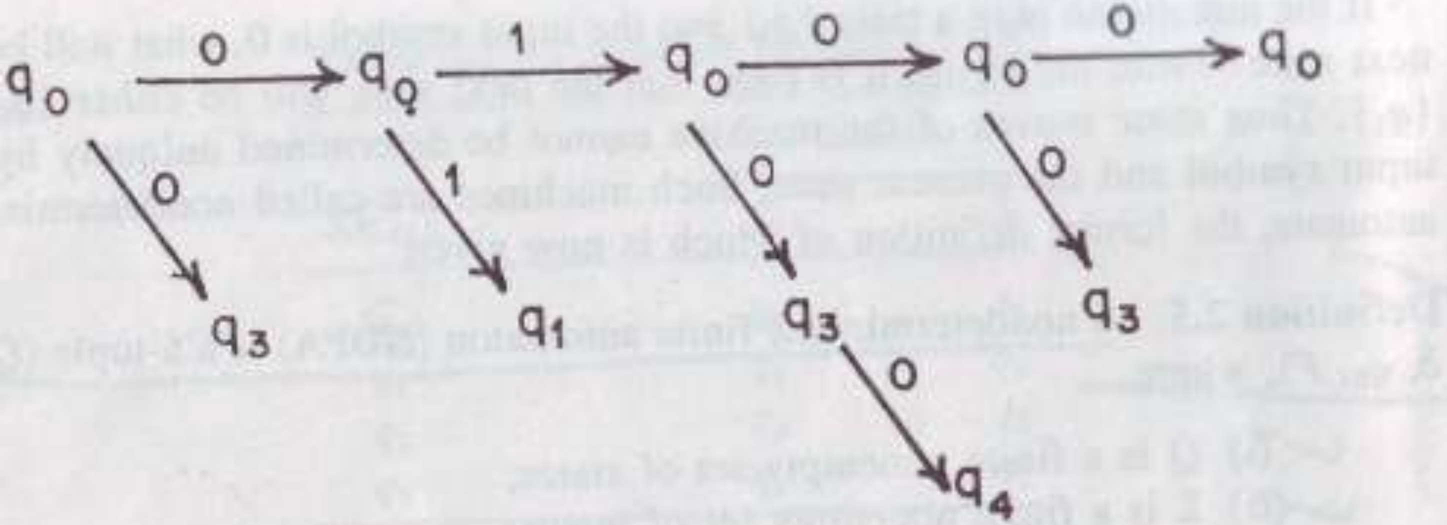


Fig. 2.9 States reached while processing 0100.

ccepted by M if a final state is *one* among the possible states M can reach application of w.

finition 2.7 The set accepted by an automaton M (deterministic or nondeterministic) is the set of all input strings accepted by M. It is denoted by $T(M)$.

Equivalence of DFA and NFA

THE EQUIVALENCE OF DFA AND NDFA

e naturally try to find the relation between DFA and NDFA. Intuitively we w feel that:

(i) A DFA can simulate the behaviour of NDFA by increasing the number states. (In other words, a DFA $(Q, \Sigma, \delta, q_0, F)$ can be viewed as an NDFA $), \Sigma, \delta', q_0, F)$ by defining $\delta'(q, a) = \{\delta(q, a)\}$.)

(ii) Any NDFA is a more general machine without being more powerful.

'e now give a theorem on equivalence of DFA and NDFA.

heorem 2.1 For every NDFA, there exists a DFA which simulates the behaviour NDFA. Alternatively, if L is the set accepted by NDFA, then there exists a FA which also accepts L.

ROOF Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NDFA accepting L. We construct a DFA I' as follows:

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

where

(i) $Q' = 2^Q$ (any state in Q' is denoted by $[q_1, q_2 \cdots q_j]$, where $q_1, q_2 \cdots q_j \in Q$);

(ii) $q'_0 = [q_0]$;

(iii) F' is the set of all subsets of Q containing an element of F.

Equivalence of DFA and NFA

(iv) $\delta'([q_1, q_2, \ldots, q_i], a) = \delta(q_1, a) \cup \delta(q_2, a) \cup \ldots \cup \delta(q_i, a)$.

Equivalently, $\delta'([q_1, q_2 \ldots q_i], a) = [p_1 \ldots p_j]$ if and only if

$$\delta(\{q_1, \ldots, q_i\}, a) = \{p_1, p_2, \ldots, p_j\}$$

Example

Table 2.2 State Table for Example 2.6

State/Σ	0	1
$\rightarrow (q_0)$	q_0	q_1
q_1	q_1	q_0, q_1

XAMPLE 2.6 Construct a deterministic automaton equivalent to $M = (\{q_0, q_1\},$ $), 1\}, \delta, q_0, \{q_0\})$. δ is given by its state table (Table 2.2).

OLUTION For the deterministic automaton M_1,

(i) the states are subsets of $\{q_0, q_1\}$, i.e. Ø, $[q_0]$, $[q_0, q_1]$, $[q_1]$;
(ii) $[q_0]$ is the initial state;
(iii) $[q_0]$ and $[q_0, q_1]$ are the final states as these are the only states containing q_0; and
(iv) δ is defined by the state table given by Table 2.3.

Table 2.3 State Table of M_1

States/Σ	0	1
Ø	Ø	Ø
$[q_0]$	$[q_0]$	$[q_1]$
$[q_1]$	$[q_1]$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$

q_0 and q_1 appear in the rows corresponding to q_0 and q_1 and the column corresponding o 0. So, $\delta([q_0, q_1], 0) = [q_0, q_1]$.

Example

EXAMPLE 2.7 Find a deterministic acceptor equivalent to

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$$

δ is given in Table 2.4.

Table 2.4 State Table for Example 2.7

States/Σ	a	b
$\to q_0$	q_0, q_1	q_2
q_1	q_0	q_1
(q_2)		q_0, q_1

SOLUTION The deterministic automaton M_1 equivalent to M is defined as follows:

$$M_1 = (2^Q, \{a, b\}, \delta, [q_0], F')$$

where

$$F' = \{[q_2], [q_0, q_2], [q_1, q_2], [q_0, q_1, q_2]\}$$

We start the construction by considering $[q_0]$ first. We get $[q_2]$ and $[q_0, q_1]$. Then we construct δ for $[q_2]$ and $[q_0, q_1]$. $[q_1, q_2]$ is a new state appearing under input columns. After constructing δ for $[q_1, q_2]$, we do not get any new states and so we terminate the construction of δ. The state table is given in Table 2.5.

Table 2.5 State Table of M_1

States/Σ	a	b
$[q_0]$	$[q_0, q_1]$	$[q_2]$
$[q_2]$	$\emptyset$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_1, q_2]$
$[q_1, q_2]$	$[q_0]$	$[q_0, q_1]$

Example

EXAMPLE 2.8 Construct a deterministic finite automaton equivalent to $M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_3\})$. δ is given in Table 2.6.

Table 2.6 State Table for Example 2.8

States/Σ	a	b
$\rightarrow q_0$	q_0, q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_3
(q_3)		q_2

SOLUTION Let $Q = \{q_0, q_1, q_2, q_3\}$. Then the deterministic automaton M_1 equivalent to M is given by $M_1 = (2^Q, \{a, b\}, \delta, [q_0], F)$, where F consists

Solution

$q_3]$, $[q_0, q_3]$, $[q_1, q_3]$, $[q_2, q_3]$, $[q_0, q_1, q_3]$, $[q_0, q_2, q_3]$, $[q_1, q_2, q_3]$ and
$q_1, q_2, q_3]$. δ is given in Table 2.7.

Table 2.7 State Table of M_1

States/Σ	a	b
$[q_0]$	$[q_0, q_1]$	$[q_0]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_1]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_3]$
$[q_0, q_1, q_3]$	$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$
$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$

Finite Automata with Output

e finite automata which we considered in the earlier sections have binary output, . they accept the string or do not accept the string. This acceptability was cided on the basis of reachability of the final state by the initial state. Now, we move this restriction and consider the model where the outputs can be chosen om some other alphabet. The value of the output function $Z(t)$ in the most general se is a function of the present state $q(t)$ and the present input $x(t)$, i.e.

$$Z(t) = \lambda(q(t), x(t))$$

here λ is called the output function. This generalised model is usually called *ealy machine*. If the output function $Z(t)$ depends only on the present state and s independent of the current input, the output function may be written as

$$Z(t) = \lambda(q(t))$$

his restricted model is called *Moore machine*. It is more convenient to use Moore nachine in automata theory. We now give the most general definitions of these nachines.

Moore Machine

Definition 2.8 The Moore machine is a six-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where

(i) Q is a finite set of states;
(ii) Σ is the input alphabet;
(iii) Δ is the output alphabet;
(iv) δ is the transition function, $\Sigma \times Q$ into Q;
(v) λ is the output function mapping Q into Δ; and
(vi) q_0 is the initial state.

Mealy Machine

Definition 2.9 A Mealy machine is a six-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where all the symbols except λ have the same meaning as in the Moore machine. λ is the output function mapping $\Sigma \times Q$ into Δ.

Example
Mealy Machine

Table 2.10 Mealy Machine of Example 2.9

Present state	Next state: input $a = 0$ state	Next state: input $a = 0$ output	Next state: input $a = 1$ state	Next state: input $a = 1$ output
$\rightarrow q_1$	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

Example
Moore Machine

Table 2.13 Moore Machine of Example 2.9

Present state	Next state		Output
	$a = 0$	$a = 1$	
$\rightarrow q_0$	q_3	q_{20}	0
q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

Procedure for Transforming Moore machine to Mealy machine

We modify the acceptability of input string by a Moore machine by neglecting the response of the Moore machine to input Λ. We thus define that Mealy Machine M and Moore Machine M' are equivalent if for all input strings w, $bZ_M(w) = Z_{M'}(w)$, where b is the output of Moore machine for its initial state. We give the following result: Let $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ be a Moore machine. Then the following procedure may be adopted to construct an equivalent Mealy machine M_2.

Construction

(a) We have to define the output function λ' for Mealy machine as a function of present state and input symbol. We define λ' by

$$\lambda'(q, a) = \lambda(\delta(q, a)) \quad \text{for all states } q \text{ and input symbols } a.$$

(b) the transition function is the same as that of the given Moore machine.

Example

Table 2.14 Moore Machine of Example 2.10

Present state	Next state		Output
	$a = 0$	$a = 1$	
$\rightarrow q_0$	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

Mealy Machine

Table 2.15 Mealy Machine of Example 2.10

Present state	Next state			
	$a = 0$		$a = 1$	
	state	output	state	output
$\rightarrow q_0$	q_3	0	q_1	1
q_1	q_1	1	q_2	0
q_2	q_2	0	q_3	0
q_3	q_3	0	q_0	0

Moore machine to Mealy Machine

Table 2.16 Moore Machine of Example 2.11

Present state	Next state		Output
	$a = 0$	$a = 1$	
$\rightarrow q_1$	q_1	q_2	0
q_2	q_1	q_3	0
q_3	q_1	q_3	1

Mealy Machine

Table 2.17 Transition Table of Example 2.11

Present state	Next state $a = 0$ state	$a = 0$ output	$a = 1$ state	$a = 1$ output
$\rightarrow q_1$	q_1	0	q_2	0
q_2	q_1	0	q_3	1
q_3	q_1	0	q_3	1

Table 2.18 Mealy Machine of Example 2.11

Present state	Next state $a = 0$ state	$a = 0$ output	$a = 1$ state	$a = 1$ output
$\rightarrow q_1$	q_1	0	q_2	0
q_2	q_1	0	q_2	1

Moore to Mealy machine conversion-Example

EXAMPLE 2.11 Consider the Moore machine described by the transition table given in Table 2.16. Construct the corresponding Mealy machine.

Table 2.16 Moore Machine of Example 2.11

Present state	Next state		Output
	$a = 0$	$a = 1$	
$\rightarrow q_1$	q_1	q_2	0
q_2	q_1	q_3	0
q_3	q_1	q_3	1

SOLUTION We construct the transition table in Table 2.17 by associating the output with the transitions.

In Table 2.17 the rows corresponding to q_2 and q_3 are identical. So, we can delete one of the two states, i.e., q_2 or q_3. We delete q_3. Table 2.18 gives the reconstructed table.

Moore to Mealy machine conversion-Example

Table 2.17 Transition Table of Example 2.11

Present state	Next state			
	$a = 0$		$a = 1$	
	state	output	state	output
$\rightarrow q_1$	q_1	0	q_2	0
q_2	q_1	0	q_3	1
q_3	q_1	0	q_3	1

Table 2.18 Mealy Machine of Example 2.11

Present state	Next state			
	$a = 0$		$a = 1$	
	state	output	state	output
$\rightarrow q_1$	q_1	0	q_2	0
q_2	q_1	0	q_2	1

In Table 2.17, we have deleted q_3-row and replaced q_3 by q_2 in the other rows.

Mealy to Moore Example

EXAMPLE 2.12 Consider a Mealy machine represented by Fig. 2.10. Construct a Moore machine equivalent to this Mealy machine.

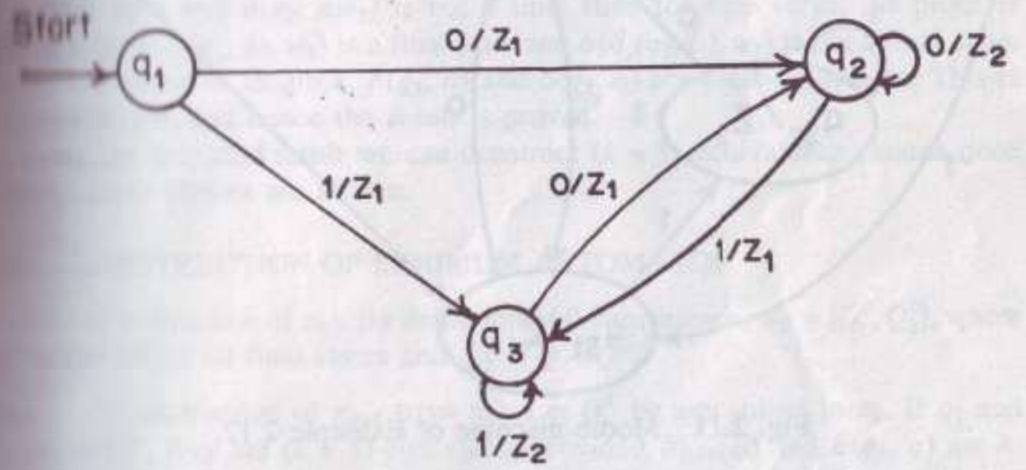


Fig. 2.10 Mealy machine of Example 2.12.

SOLUTION Let us convert the transition diagram into the transition Table 2.19. For the given problem: q_1 is not associated with any output. q_2 is associated with two different outputs Z_1 and Z_2; q_3 is associated with two different outputs Z_1 and Z_2. Thus we must split q_2 into q_{21} and q_{22} with outputs Z_1 and Z_2, respectively and q_3 into q_{31} and q_{32} with outputs Z_1 and Z_2, respectively. Table 2.19 may be reconstructed as Table 2.20.

Mealy to Moore Example

Table 2.19 Transition Table for Example 2.12

Present state	Next state $a = 0$: state	Next state $a = 0$: output	Next state $a = 1$: state	Next state $a = 1$: output
$\rightarrow q_1$	q_2	Z_1	q_3	Z_1
q_2	q_2	Z_2	q_3	Z_1
q_3	q_2	Z_1	q_3	Z_2

Table 2.20 Transition Table of Moore Machine

Present state	Next state $a = 0$	Next state $a = 1$	Output
$\rightarrow q_1$	q_{21}	q_{31}	
q_{21}	q_{22}	q_{31}	Z_1
q_{22}	q_{22}	q_{31}	Z_2
q_{31}	q_{21}	q_{32}	Z_1
q_{32}	q_{21}	q_{32}	Z_2

Figure 2.11 gives the transition diagram of the required Moore machine.

Mealy to Moore Example

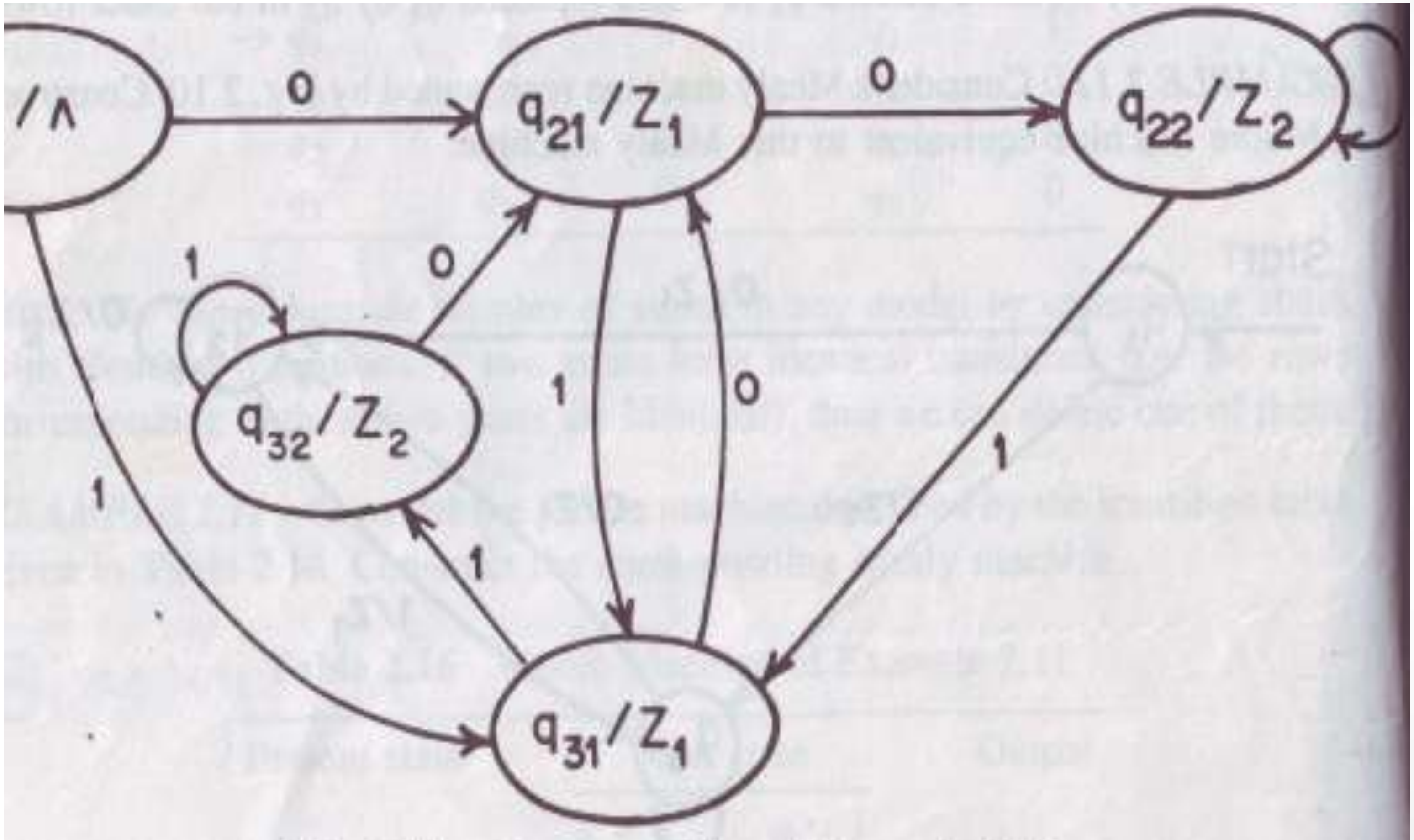


Fig. 2.11 Moore machine of Example 2.12.

Minimization of Automata

3.9.1 CONSTRUCTION OF MINIMUM AUTOMATON

Step 1 (Construction of π_0). By definition of 0-equivalence, $\pi_0 = \{Q_1^0, Q_2^0\}$, where Q_1^0 is the set of all final states and $Q_2^0 = Q - Q_1^0$.

Step 2 (Construction of π_{k+1} from π_k). Let Q_i^k be any subset in π_k. If q_1 and q_2 are in Q_i^k, they are $(k + 1)$-equivalent provided $\delta(q_1, a)$ and $\delta(q_2, a)$ are k-equivalent. Find out whether $\delta(q_1, a)$ and $\delta(q_2, a)$ are in the same equivalence class in π_k for every $a \in \Sigma$. If so, q_1 and q_2 are $(k + 1)$-equivalent. In this way, Q_i^k is further divided into $(k + 1)$-equivalence classes. Repeat this for every Q_i^k in π_k to get all the elements of π_{k+1}.

Step 3 Construct π_n for $n = 1, 2, \ldots$ until $\pi_n = \pi_{n+1}$.

Step 4 (Construction of minimum automaton). For the required minimum state automaton, the states are the equivalence classes obtained in step 3, i.e. the elements of π_n. The state table is obtained by replacing a state q by the corresponding equivalence class $[q]$.

Example – FA minimization

EXAMPLE 2.13 Construct a minimum state automaton equivalent to the finite automaton given in Fig. 2.12.

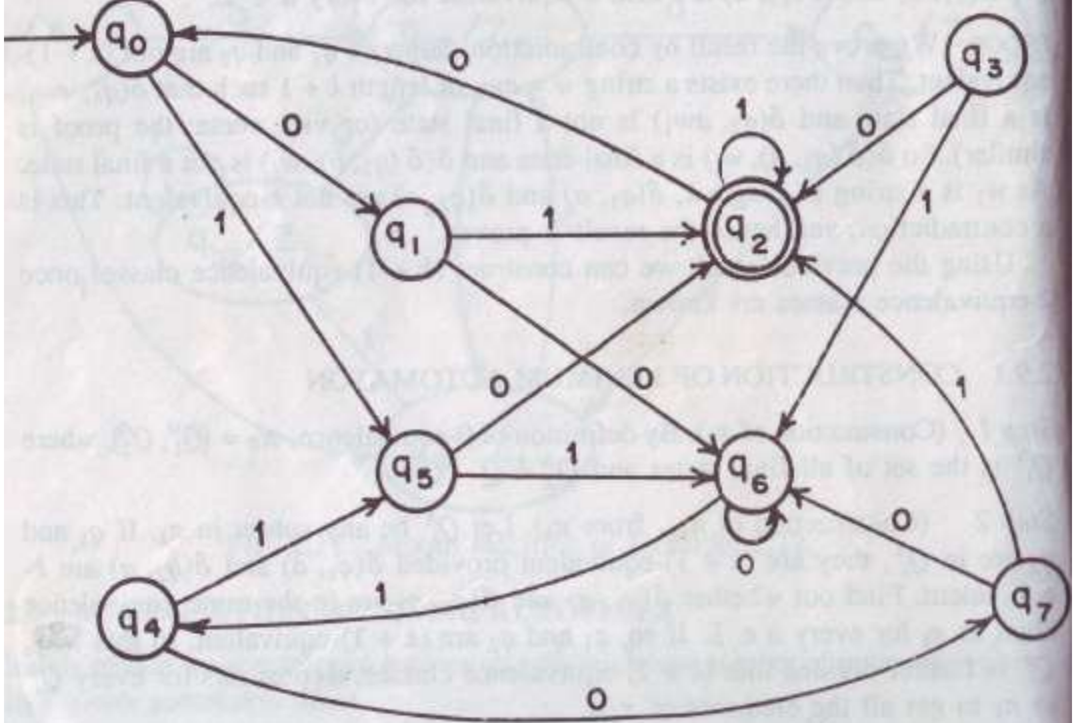


Fig. 2.12 FA of Example 2.13.

Example – FA minimization

Table 2.21 Transition Table for Example 2.13

State/Σ	0	1
$\rightarrow q_0$	q_1	q_5
q_1	q_6	q_2
(q_2)	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

By applying step 1, we get

$$Q_1^0 = F = \{q_2\}, \quad Q_2^0 = Q - Q_1^0$$

So,

$$\pi_0 = \{\{q_2\}, \{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}\}$$

Example – FA minimization

$\{q_2\}$ in π_0 cannot be further partitioned. So, $Q'_1 = \{q_2\}$. Consider q_0 and $q_1 \in Q_2^0$. The entries under 0-column corresponding to q_0 and q_1 are q_1 and q_6; they lie in Q_2^0. The entries under 1-column are q_5 and q_2. $q_2 \in Q_1^0$ and $q_5 \in Q_2^0$. Therefore, q_0 and q_1 are not 1-equivalent. Similarly, q_0 is not 1-equivalent to q_3, q_5 and q_7.

Now, consider q_0 and q_4. The entries under 0-column are q_1 and q_7. Both are in Q_2^0. The entries under 1-column are q_5, q_5. So q_4 and q_0 are 1-equivalent. Similarly, q_0 is 1-equivalent to q_6. $\{q_0, q_4, q_6\}$ is a subset in π_1. So, $Q'_2 = \{q_0, q_4, q_6\}$.

Repeat the construction by considering q_1 and any one of the states q_3, q_5, q_7. q_1 is not 1-equivalent to q_3 or q_5 but 1-equivalent to q_7. Hence, $Q'_3 = \{q_1, q_7\}$. The elements left over in Q_2^0 are q_3 and q_5. By considering the entries under 0-column and 1-column, we see that q_3 and q_5 are 1-equivalent. So $Q'_4 = \{q_3, q_5\}$. Therefore,

$$\pi_1 = \{\{q_2\}, \{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$$

$\{q_2\}$ is also in π_2 as it cannot be partitioned further. Now the entries under 0-column corresponding to q_0 and q_4 are q_1 and q_7, and these lie in the same equivalence class in π_1. The entries under 1-column are q_5, q_5. So q_0 and q_4 are 2-equivalent. But q_0 and q_6 are not 2-equivalent. Hence, $\{q_0, q_4, q_6\}$ is partitioned into $\{q_0, q_4\}$ and $\{q_6\}$. q_1 and q_7 are 2-equivalent. q_3 and q_5 are also 2-equivalent. Thus, $\pi_2 = \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$ q_0 and q_4 are 3-equivalent. q_1 and q_7 are 3-equivalent. Also, q_3 and q_5 are 3-equivalent. Therefore,

$$\pi_3 = \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$$

As $\pi_2 = \pi_3$, π_2 gives the equivalence classes, the minimum state automaton is

$$M' = (Q', \{0, 1\}, \delta', q'_0, F')$$

Example – FA minimization

re

$$Q' = \{[q_2], [q_0, q_4], [q_6], [q_1, q_7], [q_3, q_5]\}$$

$$q_0' = [q_0, q_4], \qquad F' = [q_2]$$

δ' is given by Table 2.22.

Table 2.22 Transition Table of Minimum State Automaton

State/Σ	0	1
$[q_0, q_4]$	$[q_1, q_7]$	$[q_3, q_5]$
$[q_1, q_7]$	$[q_6]$	$[q_2]$
$[q_2]$	$[q_0, q_4]$	$[q_2]$
$[q_3, q_5]$	$[q_2]$	$[q_6]$
$[q_6]$	$[q_6]$	$[q_0, q_4]$

TE: The transition diagram for the minimum state automaton is given in . 2.13. The states q_0 and q_4 are identified and treated as one state. (So also q_1, q_7 and q_3, q_5.) But the transitions in both the diagrams (i.e. Figs. 2.12 and

Example – FA minimization

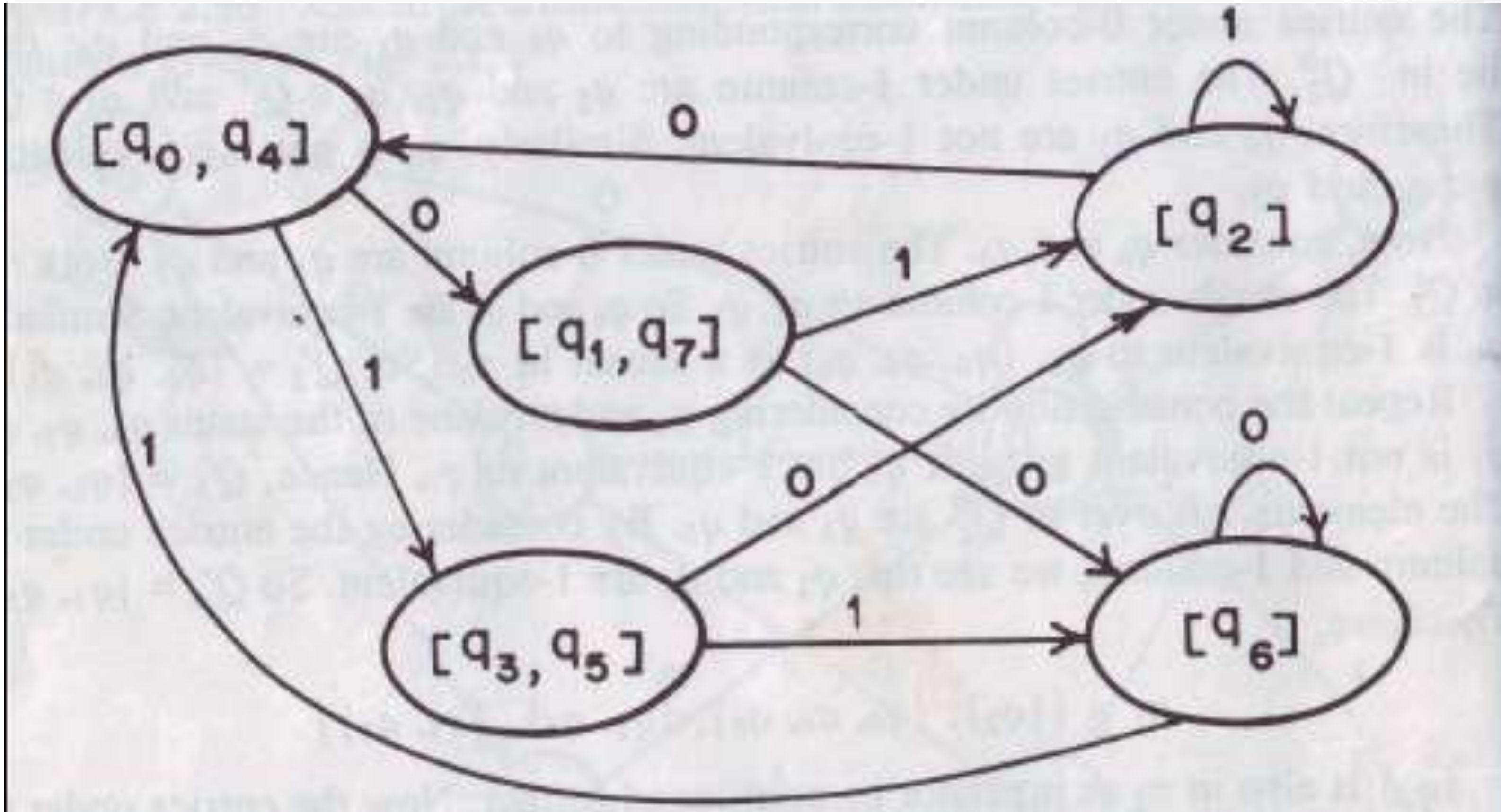
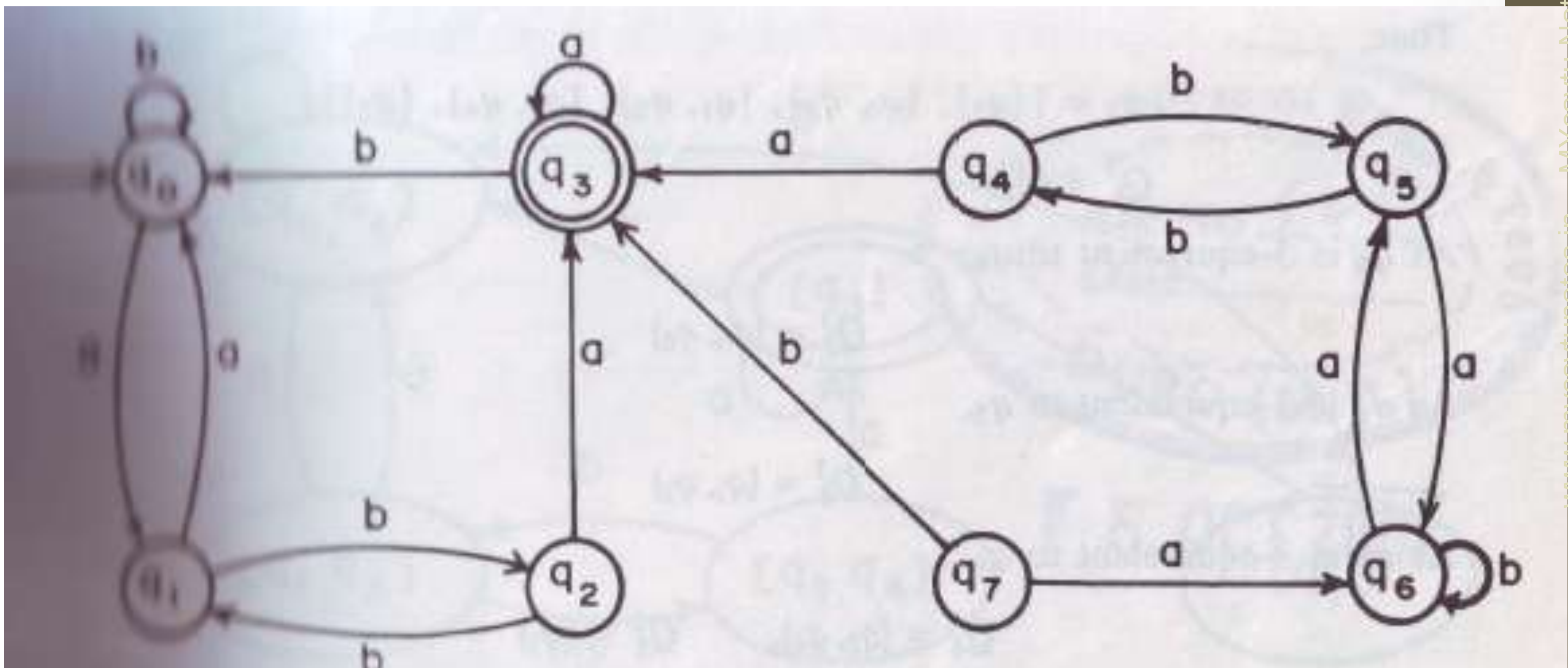


Fig. 2.13 Minimum state automaton of Example 2.13.

13) are the same. If there is an arrow from q_i to q_j with label a, then there is arrow from $[q_i]$ to $[q_j]$ with the same label in the diagram for minimum state tomaton. Symbolically, if $\delta(q_i, a) = q_j$, then $\delta'([q_i], a) = [q_j]$.

Question



Assignment

7. The transition table of a nondeterministic finite automaton M is given in Table 2.25. Construct a deterministic finite automaton equivalent to M.

Table 2.25 Transition Table for Exercise 2.7

State	0	1	2
$\rightarrow q_0$	q_1q_4	q_4	q_2q_3
q_1		q_4	
q_2			q_2q_3
(q_3)		q_4	
q_4			

8. Construct a DFA equivalent to the NDFA given in Fig. 2.8.

9. $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \delta, q_1, \{q_3\})$ is a nondeterministic finite automaton, where δ is given by

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

$$\delta(q_2, 0) = \{q_1, q_2\} \quad \delta(q_2, 1) = \emptyset$$

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

Construct an equivalent DFA.

Assignment

Construct a Mealy machine which is equivalent to the Moore machine given Table 2.26.

Table 2.26 Moore Machine of Exercise 2.11

Present state	Next state $a = 0$	Next state $a = 1$	Output
$\rightarrow q_0$	q_1	q_2	1
q_1	q_3	q_2	0
q_2	q_2	q_1	1
q_3	q_0	q_3	1

2. Construct a Moore machine equivalent to the Mealy machine M given in Table 2.27.

Table 2.27 Mealy Machine of Exercise 2.12

Present state	Next state			
	$a = 0$		$a = 1$	
	state	output	state	output
$\rightarrow q_1$	q_1	1	q_2	0
q_2	q_4	1	q_4	1
q_3	q_2	1	q_3	1
q_4	q_3	0	q_1	1

13. Construct a Mealy machine which can output EVEN, ODD according as the total number of 1's encountered is even or odd. The input symbols are 0 and 1.

14. Construct a minimum state automaton equivalent to a given automaton M whose transition table is given in Table 2.28.

Table 2.28 FA of Exercise 2.14

States	Input a	Input b
$\rightarrow q_0$	q_0	q_3
q_1	q_2	q_5
q_2	q_3	q_4
q_3	q_0	q_5
q_4	q_0	q_6
q_5	q_1	q_4
(q_6)	q_1	q_3

Regular Set and Regular Grammer

REGULAR EXPRESSIONS

Regular expressions are useful for representing certain sets of strings in an algebraic fashion. Actually these describe the languages accepted by finite state automata.

We give a formal recursive definition of regular expressions over Σ as follows:

1. Any terminal symbol (i.e. an element of Σ), Λ and $\emptyset$ are regular expressions. When we view a in Σ as a regular expression, we denote it by **a**.
2. The union of two regular expressions $\mathbf{R}_1$ and $\mathbf{R}_2$, written as $\mathbf{R}_1 + \mathbf{R}_2$, is also a regular expression.
3. The concatenation of two regular expressions $\mathbf{R}_1$ and $\mathbf{R}_2$, written as $\mathbf{R}_1\mathbf{R}_2$, is also a regular expression.
4. The iteration (or closure) of a regular expression **R**, written as $\mathbf{R}^*$, is also a regular expression.
5. If **R** is a regular expression, then (**R**) is also a regular expression.
6. The regular expressions over Σ are precisely those obtained recursively by the application of the rules 1–5 once or several times.

Regular Set

Definition 4.1 Any set represented by a regular expression is called a regular set.

If, for example, $a, b \in \Sigma$, then (a) **a** denotes the set $\{a\}$, (b) **a + b** denotes $\{a, b\}$, (c) **ab** denotes $\{ab\}$, (d) **a*** denotes the set $\{\Lambda, a, aa, aaa, \ldots\}$ and (e) **(a + b)*** denotes $\{a, b\}^*$.

Now we shall explain the evaluation procedure for the three basic operations. Let $\mathbf{R}_1$ and $\mathbf{R}_2$ denote any two regular expressions. Then (a) a string in $\mathbf{R}_1 + \mathbf{R}_2$ is a string from $\mathbf{R}_1$ or a string from $\mathbf{R}_2$; (b) a string in $\mathbf{R}_1\mathbf{R}_2$ is a string from $\mathbf{R}_1$ followed by a string from $\mathbf{R}_2$, and (c) a string in $\mathbf{R}^*$ is a string obtained by concatenating n elements for some $n \geq 0$. Consequently, (a) the set represented by $\mathbf{R}_1 + \mathbf{R}_2$ is the union of the sets represented by $\mathbf{R}_1$ and $\mathbf{R}_2$, (b) the set represented by $\mathbf{R}_1\mathbf{R}_2$ is the concatenation of the sets represented by $\mathbf{R}_1$ and $\mathbf{R}_2$ (Recall that the concatenation AB of sets A and B of strings over Σ is given by $AB = \{w_1w_2 | w_1 \in A, w_2 \in B\}$, and (c) the set represented by $\mathbf{R}^*$ is $\{w_1w_2 \ldots w_n | w_i$ is in the set represented by $\mathbf{R}$ and $n \geq 0\}$.

Reg. Set to Regular Expression

EXAMPLE 4.1 Describe the following sets by regular expressions: (a) {101}, (b) {*abba*}, (c) {01, 10}, (d) {Λ, *ab*}, (e) {*abb*, *a*, *b*, *bba*}, (f) {Λ, 0, 00, 000, ...}, and (g) {1, 11, 111, ...}.

SOLUTION (a) Now, {1}, {0} are represented by **1** and **0**, respectively. 101 is obtained by concatenating 1, 0 and 1. So, {101} is represented by **101**.

(b) **abba** represents {*abba*}.

(c) As {01, 10} is the union of {01} and {10}, {01, 10} is represented by **01 + 10**.

(d) The set {Λ, *ab*} is represented by Λ + **ab**.

(e) The set {*abb*, *a*, *b*, *bba*} is represented by **abb + a + b + bba**.

(f) As {Λ, 0, 00, 000, ...} is simply {0}*, it is represented by **0***.

(g) Any element in {1,11, 111, ...} can be obtained by concatenating 1 and any element of {1}*. Hence 1(1)* represents {1, 11, 111, ...}.

Reg. Set to Regular Expression

EXAMPLE 4.2 Describe the following sets by regular expressions:

(a) L_1 = the set of all strings of 0's and 1's ending in 00.

(b) L_2 = the set of all strings of 0's and 1's beginning with 0 and ending with 1.

(c) $L_3 = \{\Lambda, 11, 1111, 111111, \ldots\}$.

SOLUTION (a) Any string in L_1 is obtained by concatenating any string over {0, 1} and the string 00. {0, 1} is represented by **0 + 1**. Hence L_1 is represented by **(0 + 1)* 00**.

(b) As any element of L_2 is obtained by concatenating 0, any string over {0, 1} and 1, L_2 can be represented by **0(0 + 1)* 1**.

(c) Any element of L_3 is either Λ or a string of even number of 1's, i.e. a string of the form $(11)^n$, $n \geq 0$. So L_3 can be represented by **(11)***.

Identities of RE

IDENTITIES FOR REGULAR EXPRESSIONS

Two regular expressions **P** and **Q** are equivalent (we write **P** = **Q**) if **P** and **Q** represent the same set of strings.

We now give the identities for regular expressions; these are useful for simplifying regular expressions.

I_1 $\emptyset + R = R$

I_2 $\emptyset R = R\emptyset = \emptyset$

I_3 $\Lambda R = R\Lambda = R$

I_4 $\Lambda^* = \Lambda$ and $\emptyset^* = \Lambda$

I_5 $R + R = R$

I_6 $R^*R^* = R^*$

I_7 $RR^* = R^*R$

I_8 $(R^*)^* = R^*$

I_9 $\Lambda + RR^* = R^* = \Lambda + R^*R$

I_{10} $(PQ)^*P = P(QP)^*$

I_{11} $(P + Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$

I_{12} $(P + Q)R = PR + QR$ and $R(P + Q) = RP + RQ$

Arden's Theorem

Theorem 4.1 (Arden's theorem) Let **P** and **Q** be two regular expressions over Σ. If **P** does not contain Λ, then the following equation in **R**, viz.

$$\mathbf{R} = \mathbf{Q} + \mathbf{RP} \qquad (4.1)$$

has a unique solution (i.e. one and only one solution) given by $\mathbf{R} = \mathbf{QP}^*$.

Proof

$$\mathbf{Q} + (\mathbf{QP}^*)\,\mathbf{P} = \mathbf{Q}(\Lambda + \mathbf{P}^*\mathbf{P}) = \mathbf{QP}^* \text{ by } I_9$$

Hence (4.1) is satisfied when $\mathbf{R} = \mathbf{QP}^*$. This means $\mathbf{R} = \mathbf{QP}^*$ is a solution of (4.1).

To prove uniqueness, consider (4.1). Here, replacing **R** by **Q + RP** on the R.H.S., we get the equation

$$\mathbf{Q} + \mathbf{RP} = \mathbf{Q} + (\mathbf{Q} + \mathbf{RP})\mathbf{P}$$

Arden's Theorem

$$= \mathbf{Q + QP + RPP}$$

$$= \mathbf{Q + QP + RP^2}$$

$$= \mathbf{Q + QP + QP^2 + \ldots + QP^i + RP^{i+1}}$$

$$= \mathbf{Q(\Lambda + P + P^2 + \ldots + P^i) + RP^{i+1}}$$

From (4.1),

$$\mathbf{R = Q(\Lambda + P + P^2 + \ldots + P^i) + RP^{i+1}} \quad \text{for } i \geq 0 \qquad (4.2)$$

We now show that any solution of (4.1) is equivalent to $\mathbf{QP^*}$. Suppose $\mathbf{R}$ satisfies (4.1), then it satisfies (4.2). Let w be a string of length i in the set $\mathbf{R}$. Then w belongs to the set $\mathbf{Q(\Lambda + P + P^2 + \ldots + P^i) + RP^{i+1}}$. As $\mathbf{P}$ does not contain Λ, $\mathbf{RP^{i+1}}$ has no string of length less than $i + 1$ and so w is not in the set $\mathbf{RP^{i+1}}$. This means w belongs to the set $Q(\Lambda + \mathbf{P + P^2} + \ldots + \mathbf{P^i})$, and hence to $\mathbf{QP^*}$.

Consider a string w in the set $\mathbf{QP^*}$. Then w is in the set $\mathbf{QP^k}$ for some $k \geq 0$, and hence in $Q(\Lambda + \mathbf{P + P^2} + \ldots + \mathbf{P^k})$. So w is on the R.H.S. of (4.2). Therefore, w is in $\mathbf{R}$ (L.H.S. of (4.2)). Thus $\mathbf{R}$ and $\mathbf{QP^*}$ represent the same set. This proves the uniqueness of the solution of (4.1). ∎

RE

EXAMPLE 4.3 (a) Give an r.e. for representing the set L of strings in whic
every 0 is immediately followed by at least two 1's.

(b) Prove that the regular expression $\mathbf{R} = \Lambda + \mathbf{1^*(011)^*(1^*\ (011)^*)^*}$ als
describes the same set of strings.

SOLUTION (a) If w is in L, then either (i) w does not contain any 0, or (ii)
contains a 0 preceded by 1 and followed by 11. So w can be written as $w_1 w_2$.
w_n, where each w_i is either 1 or 011. So L is represented by the r.e. $(1 + 011)$

(b) $\mathbf{R} = \Lambda + \mathbf{P_1 P_1^*}$, where $\mathbf{P_1 = 1^*\ (011)^*}$

$= P_1^*$ using I_9

$= (1^*\ (011)^*)^*$

$= (\mathbf{P_2^* P_3^*})^*$ letting $\mathbf{P_2 = 1, P_3 = 011}$

$= (\mathbf{P_2 + P_3})^*$ using I_{11}

$= (1 + 011)^*$

FA to RE Conversion

EXAMPLE 4.8 Consider the transition system given in Fig. 4.10. Prove th[e] the strings recognised are **(a + a(b + aa)*b)* a(b + aa)* a.**

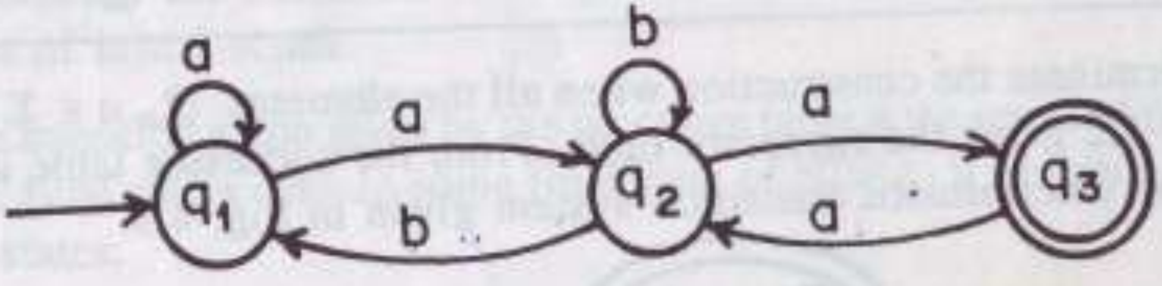


Fig. 4.10 Transition system of Example 4.8.

SOLUTION We can directly apply the above method since the graph does n[ot] contain any Λ-move and there is only one initial state.

The three equations for q_1, q_2 and q_3 can be written as

$$q_1 = q_1a + q_2b + \Lambda, \qquad q_2 = q_1a + q_2b + q_3a, \qquad q_3 = q_2a$$

It is necessary to reduce the number of unknowns by repeated substitution. [B]y substituting q_3 in q_2-equation, we get

$$q_2 = q_1a + q_2b + q_2aa$$

FA to RE Conversion

$$= q_1 a + q_2(b + aa)$$

$$= q_1 a\ (b + aa)^*$$

By applying Theorem 4.1. Substituting q_2 in q_1, we get

$$q_1 = q_1 a + q_1 a(b + aa)^* b + \Lambda$$

$$= q_1(a + a(b + aa)^* b) + \Lambda$$

Hence

$$q_1 = \Lambda(a + a(b + aa)^* b)^*$$

$$q_2 = (a + a(b + aa)^* b)^*\ a(b + aa)^*$$

$$q_3 = (a + a(b + aa)^* b)^*\ a(b + aa)^* a$$

Since q_3 is a final state, the set of strings recognised by the graph is given by

$$(a + a(b + aa)^* b)a(b + aa)^* a$$

FA to RE conversion

EXAMPLE 4.9 Prove that the FA whose transition diagram is given in Fig. 4.11 accepts the set of all strings over the alphabet {a, b} with an equal number of

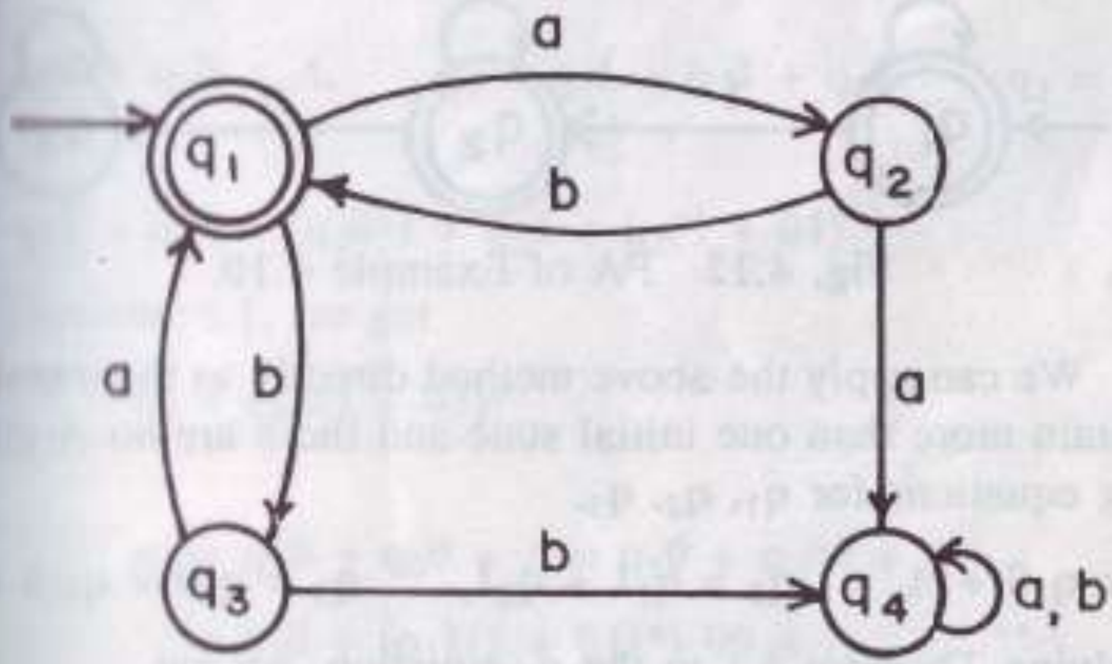


Fig. 4.11 FA of Example 4.9.

a's and b's, such that each prefix has atmost one more a than b's and atmost one more b than a's.

FA to RE conversion

SOLUTION We can apply the above method directly since the graph does not contain Λ-move and there is only one initial state. We get the following equations for q_1, q_2, q_3, q_4:

$$q_1 = q_2b + q_3a + \Lambda$$

$$q_2 = q_1a,$$

$$q_3 = q_1b$$

$$q_4 = q_2a + q_3b + q_4a + q_4b$$

As q_1 is the only final state and the q_1-equation involves only q_2 and q_3, we use

FA to RE conversion

only q_2- and q_3-equations (the q_4-equation is redundant for our purposes). Substituting for q_2 and q_3, we get

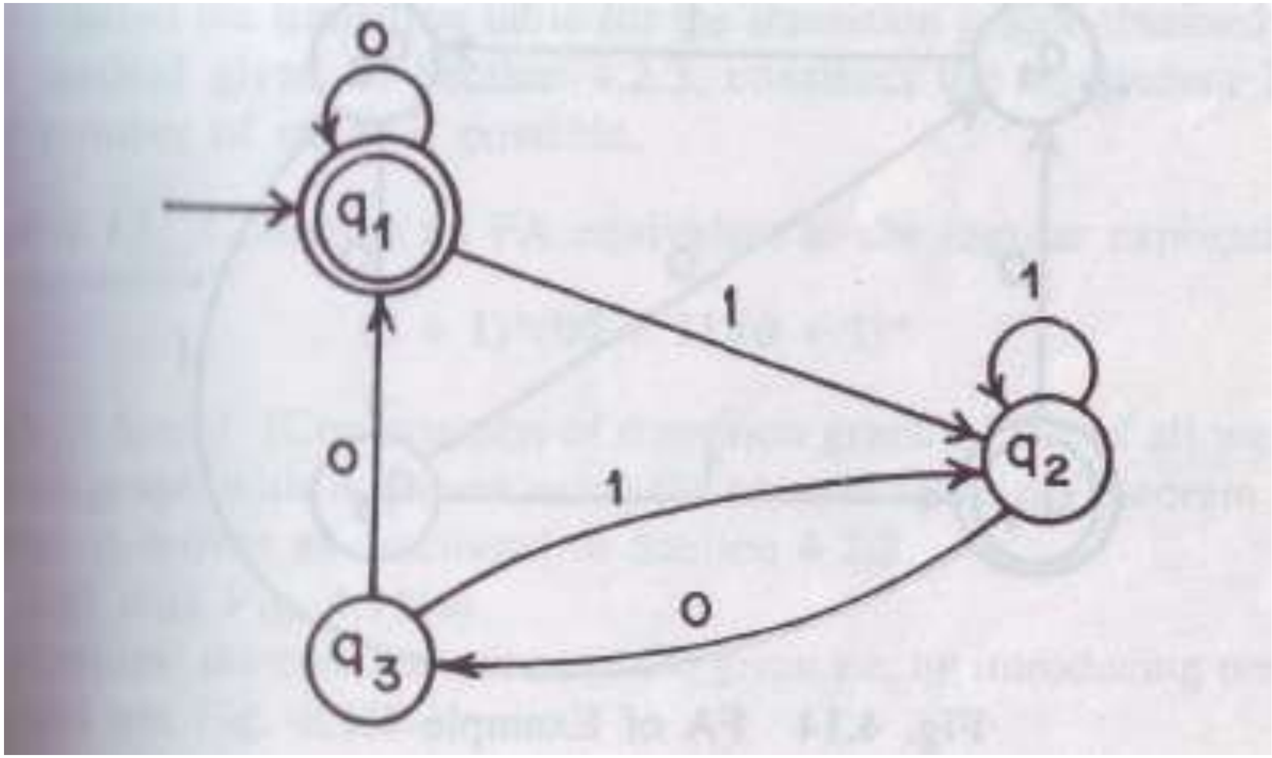
$$q_1 = q_1 ab + q_1 ba + \Lambda = q_1(ab + ba) + \Lambda$$

By applying Theorem 4.1, we get

$$q_1 = \Lambda(ab + ba)^* = (ab + ba)^*$$

As q_1 is the only final state, the strings accepted by the given FA are strings given by $(ab + ba)^*$. As any such string is a string of *ab*'s and *ba*'s we get equal number of *a*'s and *b*'s. If a prefix *x* of a sentence accepted by the FA has even number of symbols, then it should have equal number of *a*'s and *b*'s since *x* is a substring formed by *ab*'s and *ba*'s. If the prefix *x* has odd number of symbols, then we can write *x* as *ya* or *yb*. As *y* has even number of symbols, *y* has equal number of *a*'s and *b*'s. Thus *x* has one more *a* than *b* or vice versa.

FA to RE conversion



FA to RE conversion

EXAMPLE 4.10 Describe in English the set accepted by FA whose transition diagram is given in Fig. 4.12.

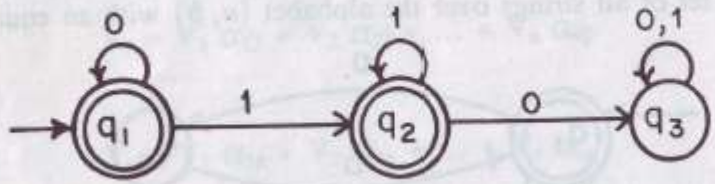


Fig. 4.12 FA of Example 4.10.

RE to DFA

EXAMPLE 4.13 Construct an FA equivalent to the regular expression.

$$(0 + 1)^*(00 + 11)(0 + 1)^*$$

SOLUTION *Step 1* (Construction of transition graph). First of all we construct the transition graph with Λ-moves using the constructions of Theorem 4.2. Then we eliminate Λ-moves as discussed in Section 4.2.2.

We start with Fig. 4.15(a).

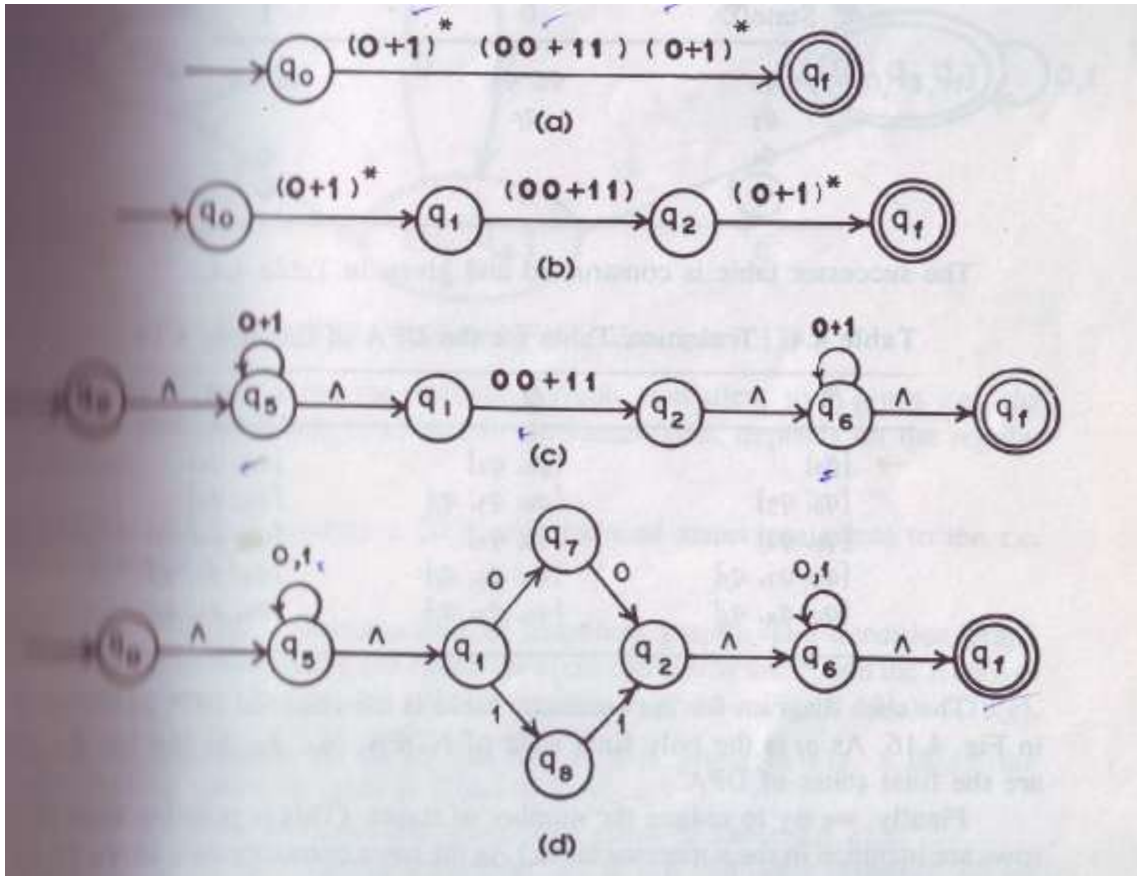
We eliminate the concatenations in the given r.e. by introducing new vertices q_1 and q_2 and get Fig. 4.15(b).

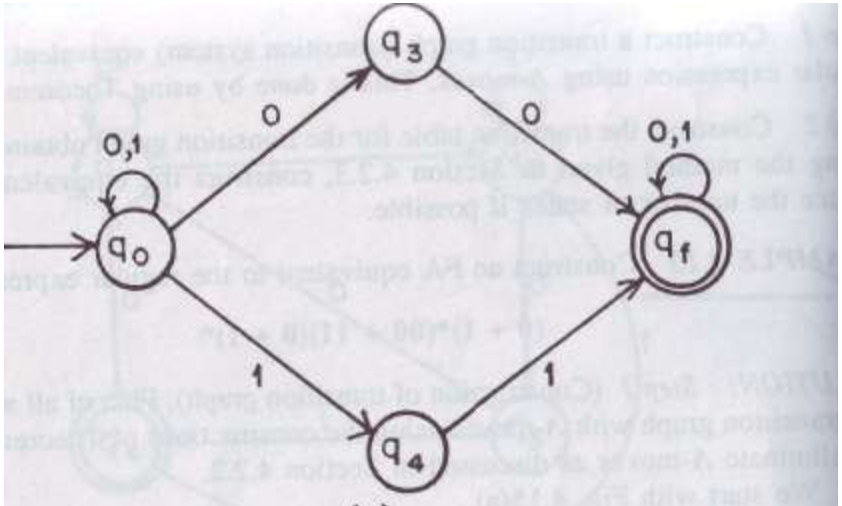
We eliminate * operations in Fig. 4.15(b) by introducing two new vertices q_5 and q_6 and Λ-moves as shown in Fig. 4.15(c).

We eliminate concatenations and + in Fig. 4.15(c) and get Fig. 4.15(d).

We eliminate Λ-moves in Fig. 4.15(d) and get Fig. 4.15(e) which gives the DFA equivalent to the given r.e.

RE to DFA


(0+1)* (00 + 11) (0+1)*
q0
qf
(a)
(0+1)*
(00+11)
(0+1)*
q0
q1
q2
qf
(b)
0+1
00+11
0+1
Λ
q5
q1
q2
q6
qf
(c)
0,1
0
0
1
1
0,1
Λ
q5
q1
q7
q8
q2
q6
qf
(d)



(e)

Formal Language

$S \rightarrow$ ⟨noun⟩ ⟨verb⟩ ⟨adverb⟩
$S \rightarrow$ ⟨noun⟩ ⟨verb⟩
⟨noun⟩ $\rightarrow$ Sam
⟨noun⟩ $\rightarrow$ Ram
⟨noun⟩ $\rightarrow$ Gita
⟨verb⟩ $\rightarrow$ ran
⟨verb⟩ $\rightarrow$ ate
⟨verb⟩ $\rightarrow$ walked
⟨adverb⟩ $\rightarrow$ slowly
⟨adverb⟩ $\rightarrow$ quickly

ach arrow represents a rule meaning that the word on the right side of the row can replace the word on the left side of the arrow.) Let us denote the llection of the rules given above by P.

If our vocabulary is thus restricted to 'Ram', 'Sam', 'Gita', 'ate', 'ran', alked', 'quickly' and 'slowly', and our sentences are of the form ⟨noun⟩ erb⟩ ⟨adverb⟩ and ⟨noun⟩ ⟨verb⟩, we can describe the grammar by a 4-tuple $_N$, Σ, P, S), where

V_N = {⟨noun⟩, ⟨verb⟩, ⟨adverb⟩}

Σ = {Ram, Sam, Gita, ate, ran, walked, quickly, slowly}

P is the collection of rules described above (the rules may be called oductions).

S is the special symbol denoting a sentence.

Grammar

4.1.1 DEFINITION OF A GRAMMAR

Definition 4.1 A phrase-structure grammar (or simply a grammar) is (V_N, Σ, P, S), where

(i) V_N is a finite nonempty set whose elements are called variables,
(ii) Σ is a finite nonempty set whose elements are called terminals,
(iii) $V_N \cap \Sigma = \emptyset$,
(iv) S is a special variable (i.e. an element of V_N) called the start symbol, and
(v) P is a finite set whose elements are $\alpha \rightarrow \beta$, where α and β are strings on $V_N \cup \Sigma$. α has at least one symbol from V_N. The elements of P are called productions or production rules or rewriting rules.

Note: The set of productions is the kernel of grammars and language specification. We observe the following regarding the production rules.

(i) Reverse substitution is not permitted. For example, if $S \rightarrow AB$ is a production, then we can replace S by AB, but we cannot replace AB by S.
(ii) No inversion operation is permitted. For example, if $S \rightarrow AB$ is a production, it is not necessary that $AB \rightarrow S$ is a production.

Grammar

$G = (V_N, \Sigma, P, S)$ is a grammar

where

$V_N = \{\langle\text{sentence}\rangle, \langle\text{noun}\rangle, \langle\text{verb}\rangle, \langle\text{adverb}\rangle\}$

$\Sigma = \{\text{Ram, Sam, ate, sang, well}\}$

$S = \langle\text{sentence}\rangle$

P consists of the following productions:

$\langle\text{sentence}\rangle \rightarrow \langle\text{noun}\rangle\ \langle\text{verb}\rangle$

$\langle\text{sentence}\rangle \rightarrow \langle\text{noun}\rangle\ \langle\text{verb}\rangle\ \langle\text{adverb}\rangle$

$\langle\text{noun}\rangle \rightarrow \text{Ram}$

$\langle\text{noun}\rangle \rightarrow \text{Sam}$

$\langle\text{verb}\rangle \rightarrow \text{ate}$

$\langle\text{verb}\rangle \rightarrow \text{sang}$

$\langle\text{adverb}\rangle \rightarrow \text{well}$

Grammar

Chomsky Hierarchy

1. Phrase Structure Grammar (Unrestricted Grammar) or Type-0

2.Context Sensitive Grammar or Type-1

3.Context Free Grammar or Type-2

4.Regular Grammar or Type-3

Phrase Structure Grammar

Definition 4.1 A phrase-structure grammar (or simply a grammar) is (V_N, Σ, P, S), where

(i) V_N is a finite nonempty set whose elements are called variables,
(ii) Σ is a finite nonempty set whose elements are called terminals,
(iii) $V_N \cap \Sigma = \emptyset$,
(iv) S is a special variable (i.e. an element of V_N) called the start symbol, and
(v) P is a finite set whose elements are $\alpha \to \beta$, where α and β are strings on $V_N \cup \Sigma$. α has at least one symbol from V_N. The elements of P are called productions or production rules or rewriting rules.

Derivation and Language Generated by Grammar

efinition 4.4 The language generated by a grammar G (denoted by $L(G)$)
efined as $\{w \in \Sigma^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w\}$. The elements of $L(G)$ are called *senten*

Stated in another way, $L(G)$ is the set of all terminal strings derived
e start symbol S.

efinition 4.5 If $S \overset{*}{\underset{G}{\Rightarrow}} \alpha$, then α is called a *sentential form*. We can
at the elements of $L(G)$ are sentential forms but not vice versa.

Derivation and Language Generated by Grammar

If G = ({S}, {0, 1}, {$S \rightarrow 0S1$, $S \rightarrow \Lambda$}, S), find $L(G)$.

Solution

As $S \rightarrow \Lambda$ is a production, $S \underset{G}{\Rightarrow} \Lambda$. So Λ is in $L(G)$. Also, for all $n \geq 1$,

$$S \underset{G}{\Rightarrow} 0S1 \underset{G}{\Rightarrow} 0^2S1^2 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} 0^nS1^n \underset{G}{\Rightarrow} 0^n1^n$$

Therefore,

$$0^n1^n \in L(G) \text{ for } n \geq 0$$

Note that in the above derivation, $S \rightarrow 0S1$ is applied at every step except in the last step. In the last step, we apply $S \rightarrow \Lambda$). Hence, $\{0^n1^n \mid n \geq 0\} \subseteq L(G)$.

To show that $L(G) \subseteq \{0^n1^n \mid n \geq 0\}$, we start with w in $L(G)$. The derivation of w starts with S. If $S \rightarrow \Lambda$ is applied first, we get Λ. In this case $w = \Lambda$. Otherwise the first production to be applied is $S \rightarrow 0S1$. At any stage if we apply $S \rightarrow \Lambda$, we get a terminal string. Also, the terminal string is obtained only by applying $S \rightarrow \Lambda$. Thus the derivation of w is of the form

$$S \overset{*}{\underset{G}{\Rightarrow}} 0^nS1^n \underset{G}{\Rightarrow} 0^n1^n \quad \text{for some } n \geq 1$$

$$L(G) \subseteq \{0^n1^n \mid n \geq 0\}$$

EXAMPLE 4.3

If $G = (\{S\}, \{a\}, \{S \rightarrow SS\}, S)$, find the language generated by G.

Solution

$L(G) = \emptyset$, since the only production $S \rightarrow SS$ in G has no terminal on the right-hand side.

EXAMPLE 4.4

Let $G = (\{S, C\}, \{a, b\}, P, S)$, where P consists of $S \rightarrow aCa$, $C \rightarrow aCa \mid b$. Find $L(G)$.

Solution

$$S \Rightarrow aCa \Rightarrow aba. \text{ So } aba \in L(G)$$

$$S \Rightarrow aCa \quad \text{(by application of } S \rightarrow aCa)$$

$$\stackrel{*}{\Rightarrow} a^nCa^n \quad \text{(by application of } C \rightarrow aCa \ (n-1) \text{ times)}$$

$$\Rightarrow a^nba^n \quad \text{(by application of } C \rightarrow b)$$

Hence, $a^nba^n \in L(G)$, where $n \geq 1$. Therefore,

$$\{a^nba^n \mid n \geq 1\} \subseteq L(G)$$

As the only S-production is $S \rightarrow aCa$, this is the first production we have to apply in the derivation of any terminal string. If we apply $C \rightarrow b$, we get aba. Otherwise we have to apply only $C \rightarrow aCa$, either once or several times. So we get a^nCa^n with a single variable C. To get a terminal string we have to replace C by b, by applying $C \rightarrow b$. So any derivation is of the form

$$S \stackrel{*}{\Rightarrow} a^nba^n \text{ with } n \geq 1$$

Therefore,

$$L(G) \subseteq \{a^nba^n \mid n \geq 1\}$$

Thus,

$$L(G) = \{a^nba^n \mid n \geq 1\}$$

EXERCISE Construct a grammar G so that $L(G) = \{a^nba^m \mid n, m \geq 1\}$

Derivation and Language Generated by Grammar

EXAMPLE 4.9 If G is $S \rightarrow aS \mid bS \mid a \mid b$, find $L(G)$.

Solution

We show that $L(G) = \{a, b\}^+$. As we have only two terminals a, b, $L(G) \subseteq \{a, b\}^*$. All productions are S-productions, and so Λ can be in $L(G)$ only when $S \rightarrow \Lambda$ is a production in the grammar G. Thus,

$$L(G) \subseteq \{a, b\}^* - \{\Lambda\} = \{a, b\}^+$$

To show $\{a, b\}^+ \subseteq L(G)$, consider any string $a_1 a_2 \ldots a_n$, where each a_i is a or b. The first production in the derivation of $a_1 a_2 \ldots a_n$ is $S \rightarrow aS$ or $S \rightarrow bS$ according as $a_1 = a$ or $a_1 = b$. The subsequent productions are obtained in a similar way. The last production is $S \rightarrow a$ or $S \rightarrow b$ according as $a_n = a$ or $a_n = b$. So $a_1 a_2 \ldots a_n \in L(G)$. Thus, we have $L(G) = \{a, b\}^+$.

EXAMPLE 4.10 If G is $S \rightarrow aS \mid a$, then show that $L(G) = \{a\}^+$.

Some of the following examples illustrate the method of constructing a grammar G generating a given subset of strings over Σ. The difficult part is the construction of productions. We try to define the given set by recursion and then develop the productions generating the strings in the given subset of Σ^*.

Derivation and Language Generated by Grammar

EXAMPLE 4.6

Let L be the set of all palindromes over $\{a, b\}$. Construct a grammar G generating L.

Solution

(i) For constructing a grammar G generating the set of all palindromes, we use the recursive definition (given in Section 2.4) to observe the following:

(i) Λ is a palindrome.

(ii) a, b are palindromes.

(iii) If x is a palindrome axa, then bxb are palindromes.

(b) We define P as the set consisting of:

(i) $S \to \Lambda$

(ii) $S \to a$ and $S \to b$

(iii) $S \to aSa$ and $S \to bSb$

(c) $G = (\{S\}, \{a, b\}, P, S)$. Then

$$S \Rightarrow \Lambda, \qquad S \Rightarrow a, \qquad S \Rightarrow b$$

Therefore,

$$\Lambda, a, b \in L(G)$$

If x is a palindrome of even length, then $x = a_1a_2 \ldots a_m a_m \ldots a_1$, where each a_i is either a or b. Then $S \overset{*}{\Rightarrow} a_1a_2 \ldots a_m a_m a_{m-1} \cdots a_1$ by applying $S \to aSa$ or $S \to bSb$. Thus, $x \in L(G)$.

Derivation and Language Generated by Grammar

AMPLE 4.7

struct a grammar generating $L = \{wcw^T \mid w \in \{a, b\}^*\}$.

ution

$G = (\{S\}, \{a, b, c\}, P, S)$, where P is defined as $S \rightarrow aSa \mid bSb \mid c$. It
asy to see the idea behind the construction. Any string in L is generated
recursion as follows: (i) $c \in L$; (ii) if $x \in L$, then $wxw^T \in L$. So, as in
earlier example, we have the productions $S \rightarrow aSa \mid bSb \mid c$.

Derivation and Language Generated by Grammar

XAMPLE 4.8

d a grammar generating $L = \{a^n b^n c^i \mid n \geq 1,\ i \geq 0\}$.

lution

$$L = L_1 \cup L_2$$

$$L_1 = \{a^n b^n \mid n \geq 1\}$$

$$L_2 = \{a^n b^n c^i \mid n \geq 1,\ i \geq 1\}$$

We construct L_1 by recursion and L_2 by concatenating the elements of L_1 d c^i, $i \geq 1$. We define P as the set of the following productions:

$$S \to A, \qquad A \to ab, \qquad A \to aAb, \qquad S \to Sc$$

Let $G = (\{S, A\}, \{a, b, c\}, P, S)$. For $n \geq 1$, $i \to 0$, we have

$$S \overset{*}{\Rightarrow} Sc^i \Rightarrow Ac^i \overset{*}{\Rightarrow} a^{n-1}Ab^{n-1}c^i \Rightarrow a^{n-1}abb^{n-1}c^i = a^n b^n c^i$$

hus,

$$\{a^n b^n c^i \mid n \geq 1,\ i \geq 0\} \subseteq L(G)$$

To prove the reverse inclusion, we note that the only S-productions re $S \to Sc$ and $S \to A$. If we start with $S \to A$, we have to apply

$$A \Rightarrow a^{n-1}Ab^{n-1} \overset{*}{\Rightarrow} a^n b^n, \text{ and so } a^n b^n c^0 \in L(G)$$

If we start with $S \to Sc$, we have to apply $S \to Sc$ repeatedly to get Sc^i. But o get a terminal string, we have to apply $S \to A$. As $A \overset{*}{\Rightarrow} a^n b^n$, the resulting terminal string is $a^n b^n c^i$. Thus, we have shown that

$$L(G) \subseteq \{a^n b^n c^i \mid n \geq 1,\ i \geq 0\}$$

Therefore,

$$L(G) = \{a^n b^n c^i \mid n \geq 1,\ i \geq 0\}$$

Closure Properties of families of languages

LANGUAGES AND THEIR RELATION

In this section we discuss the relation between the classes of languages that we have defined under the Chomsky classification.

Let $\mathcal{L}_0$, $\mathcal{L}_{csl}$, $\mathcal{L}_{cfl}$ and $\mathcal{L}_{rl}$ denote the family of type 0 languages, context-sensitive languages, context-free languages and regular languages, respectively.

Property 1 From the definition, it follows that $\mathcal{L}_{rl} \subseteq \mathcal{L}_{cfl}$, $\mathcal{L}_{csl} \subseteq \mathcal{L}_0$, $\mathcal{L}_{cfl} \subseteq \mathcal{L}_0$.

Property 2 $\mathcal{L}_{cfl} \subseteq \mathcal{L}_{csl}$. The inclusion relation is not immediate as we allow $A \to \Lambda$ in context-free grammars even when $A \neq S$, but not in context-sensitive grammars (we allow only $S \to \Lambda$ in context-sensitive grammars). In Chapter 6 we prove that a context-free grammar G with productions of the form $A \to \Lambda$ is equivalent to a context-free grammar G_1 which has no productions of the form $A \to \Lambda$ (except $S \to \Lambda$). Also, when G_1 has $S \to \Lambda$, S does not appear on the right-hand side of any production. So G_1 is context-sensitive. This proves $\mathcal{L}_{cfl} \subseteq \mathcal{L}_{csl}$.

Closure Properties of families of languages

Property 3 $\mathcal{L}_{rl} \subseteq \mathcal{L}_{cfl} \subseteq \mathcal{L}_{csl} \subseteq \mathcal{L}_0$. This follows from properties 1 and 2.

Property 4 $\mathcal{L}_{rl} \subsetneq \mathcal{L}_{cfl} \subsetneq \mathcal{L}_{csl} \subsetneq \mathcal{L}_0$.

Closure Properties of families of languages

OPERATIONS ON LANGUAGES

consider the effect of applying set operations on $\mathcal{L}_0$, $\mathcal{L}_{cs1}$, $\mathcal{L}_{cf1}$, $\mathcal{L}_{r1}$. Let
nd B be any two sets of strings. The concatenation AB of A and B is
ined by $AB = \{uv \mid u \in A, v \in B\}$. (Here, uv is the concatenation of the
ngs u and v.)

We define A^1 as A and A^{n+1} as A^nA for all $n \geq 1$.

The transpose set A^T of A is defined by

$$A^T = \{u^T \mid u \in A\}$$

Closure Properties of families of languages

eorem 4.5 Each of the classes $\mathcal{L}_0$, $\mathcal{L}_{csl}$, $\mathcal{L}_{cfl}$, $\mathcal{L}_{rl}$ is closed under union.

oof Let L_1 and L_2 be two languages of the same type i. We can apply eorem 4.1 to get grammars

$$G_1 = (V'_N, \Sigma_1, P_1, S_1) \quad \text{and} \quad G_2 = (V''_N, \Sigma_2, P_2, S_2)$$

type i generating L_1 and L_2, respectively. So any production in G_1 or G_2 either $\alpha \rightarrow \beta$, where α, β contain only variables or $A \rightarrow a$, where $A \in V_N$ $\in \Sigma$.

We can further assume that $V'_N \cap V''_N = \emptyset$. (This is achieved by renaming e variables of V''_N if they occur in V'_N.)

Define a new grammar G_u as follows:

$$G_u = (V'_N \cup V''_N \cup \{S\}, \Sigma_1 \cup \Sigma_2, P_u, S)$$

here S is a new symbol, i.e. $S \notin V'_N \cup V''_N$

$$P_u = P_1 \cup P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}$$

Closure Properties of families of languages

We prove $L(G_u) = L_1 \cup L_2$ as follows: If $w \in L_1 \cup L_2$, then $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} w$ or $S_2 \overset{*}{\underset{G_2}{\Rightarrow}} w$. Therefore,

$$S \underset{G_u}{\Rightarrow} S_1 \overset{*}{\underset{G_u}{\Rightarrow}} w \quad \text{or} \quad S \underset{G_u}{\Rightarrow} S_2 \overset{*}{\underset{G_u}{\Rightarrow}} w, \text{ i.e. } w \in L(G_u)$$

Thus, $L_1 \cup L_2 \subseteq L(G_u)$.

To prove that $L(G_u) \subseteq L_1 \cup L_2$, consider a derivation of w. The first step should be $S \Rightarrow S_1$ or $S \Rightarrow S_2$. If $S \Rightarrow S_1$ is the first step, in the subsequent steps S_1 is changed. As $V'_N \cap V''_N \neq \emptyset$, these steps should involve only the variables of V'_N and the productions we apply are in P_1. So $S \overset{*}{\underset{G_1}{\Rightarrow}} w$. Similarly, if the first step is $S \Rightarrow S_2$, then $S \underset{G_2}{\Rightarrow} S_2 \overset{*}{\underset{G_2}{\Rightarrow}} w$. Thus, $L(G_u) = L_1 \cup L_2$. Also, $L(G_u)$ is of type 0 or type 2 according as L_1 and L_2 are of type 0 or type 2. If Λ is not in $L_1 \cup L_2$, then $L(G_u)$ is of type 3 or type 1 according as L_1 and L_2 are of type 3 or type 1.

Suppose $\Lambda \in L_1$. In this case, define

$$G_u = (V'_N \cup V''_N \cup \{S, S'\}, \Sigma_1 \cup \Sigma_2, P_u, S')$$

where (i) S' is a new symbol, i.e. $S' \notin V'_N \cup V''_N \cup \{S\}$, and (ii) $P_u = P_1 \cup P_2 \cup \{S' \rightarrow S, S \rightarrow S_1, S \rightarrow S_2\}$. So, $L(G_u)$ is of type 1 or type 3 according as L_1 and L_2 are of type 1 or type 3. When $\Lambda \in L_2$, the proof is similar. ∎

Closure Properties of families of languages

Theorem 4.6 Each of the classes $\mathcal{L}_0$, $\mathcal{L}_{csl}$, $\mathcal{L}_{cfl}$, $\mathcal{L}_{rl}$ is closed under concatenation.

Proof Let L_1 and L_2 be two languages of type i. Then, as in Theorem 4.5, we get $G_1 = (V'_N, \Sigma_1, P_1, S_1)$ and $G_2 = (V''_N, \Sigma_2, P_2, S_2)$ of the same type i. We have to prove that L_1L_2 is of type i.

Construct a new grammar G_{con} as follows:

$$G_{con} = (V'_N \cup V''_N \cup \{S\}, \Sigma_1 \cup \Sigma_2, P_{con}, S)$$

where $S \notin V'_N \cup V''_N$.

$$P_{con} = P_1 \cup P_2 \cup \{S \rightarrow S_1S_2\}$$

We prove $L_1L_2 = L(G_{con})$. If $w = w_1w_2 \in L_1L_2$ then

$$S_1 \underset{G_1}{\overset{*}{\Rightarrow}} w_1, \qquad S_2 \underset{G_2}{\overset{*}{\Rightarrow}} w_2$$

So,

$$S \underset{G_{con}}{\Rightarrow} S_1S_2 \underset{G_{con}}{\overset{*}{\Rightarrow}} w_1w_2$$

Therefore,

$$L_1L_2 \subseteq L(G_{con})$$

Pumping Lemma

4.3 PUMPING LEMMA FOR REGULAR SETS

In this section we give a necessary condition for an input string to belong to a regular set. The result is called *pumping lemma* as it gives a method of pumping (generating) many input strings from a given string. As pumping lemma gives a necessary condition, it can be used to show that certain sets are not regular.

Theorem 4.5 (Pumping Lemma) Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton with n states. Let L be the regular set accepted by M. Let $w \in L$ and $|w| \geq m$. If $m \geq n$, then there exists x, y, z such that $w = xyz$, $y \neq \Lambda$ and $xy^iz \in L$ for each $i \geq 0$.

PROOF Let

$$w = a_1a_2 \ldots a_m, \quad m \geq n$$

$$\delta(q_0, a_1a_2 \ldots a_i) = q_i \quad \text{for } i = 1, 2, \ldots, m; \; Q_1 = \{q_0, q_1, \ldots, q_m\}$$

That is, Q_1 is the sequence of states in the path with path value $w = a_1a_2 \ldots a_m$. As there are only n distinct states, at least two states in Q_1 must coincide. Among various pairs of repeated states, we take the first pair. Let us take them as q_j and q_k $(q_j = q_k)$. Then j and k satisfy the condition $0 \leq j < k \leq n$.

The string w can be decomposed into three substrings $a_1a_2 \ldots a_j$, $a_{j+1} \ldots a_k$ and $a_{k+1} \ldots a_m$. Let x, y, z denote these strings $a_1a_2 \ldots a_j$, $a_{j+1} \ldots a_k$, $a_{k+1} \ldots a_m$, respectively. As $k \leq n$, $|xy| \leq n$ and $w = xyz$. The path with path value w in the transition diagram of M is shown in Fig. 4.24.

The automaton M starts from the initial state q_0. On applying the string x, it reaches $q_j (= q_k)$. On applying the string y, it comes back to $q_j (= q_k)$. So after application of y^i for each $i \geq 0$, the automaton is in the same state q_j. On applying z, it reaches q_m, a final state. Hence $xy^iz \in L$. As every state in Q_1 is obtained by applying an input symbol, $y \neq \Lambda$. ∎

Pumping Lemma

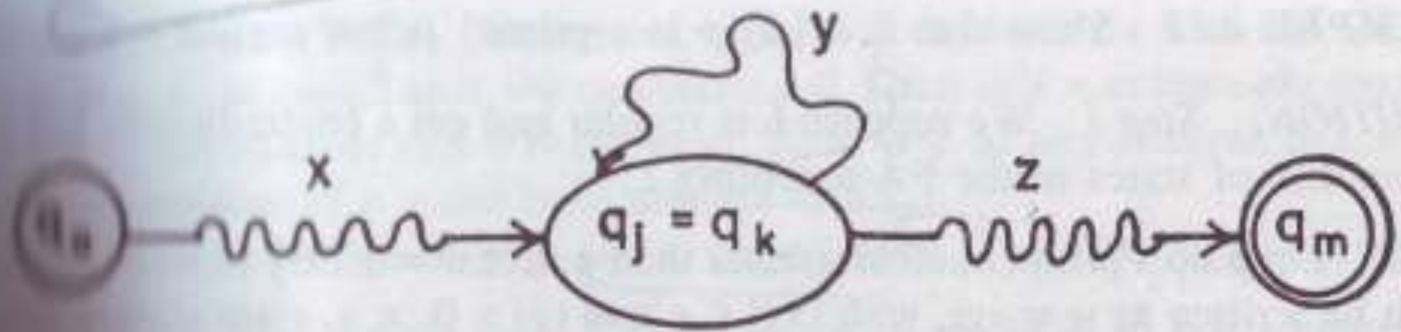


Fig. 4.24 String accepted by *M*.

The decomposition is valid only for strings of length greater than or equal to the number of states. For such a string $w = xyz$, we can 'iterate' the substring y in xyz as many times as we like and get strings of the form xy^iz which are longer than xyz and are in L. By considering the path from q_0 to q_k and then the path from q_k to q_m (without going through the loop), we get a path ending in a final state with path value xz. (This corresponds to the case when $i = 0$.)

Pumping Lemma

APPLICATION OF PUMPING LEMMA

This theorem can be used to prove that certain sets are not regular. We now give the steps needed for proving that a given set is not regular.

Step 1 Assume L is regular. Let n be the number of states in the corresponding FA.

Step 2 Choose a string w such that $|w| \geq n$. Use pumping lemma to write $w = xyz$, with $|xy| \leq n$ and $|y| > 0$.

Step 3 Find a suitable integer i such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.

Note: The crucial part of the procedure is to find i such that $xy^iz \notin L$. In some cases we prove $xy^iz \notin L$ by considering $|xy^iz|$. In some cases we may have to use the 'structure' of strings in L.

Unit-III
CFG and PDA

UNIT-3 Context free grammar and their properties, derivation tree, simplifying CFG, unambigufying CFG, CNF and GNF of CFG, push down automata, Two way PDA, relation of PDA with CFG, Determinism and Non determinism in PDA and related theorems.

CFG

6.1 CONTEXT-FREE LANGUAGES AND DERIVATION TREES

Context-free languages are applied in parser design. They are also useful for describing block structures in programming languages. It is easy to visualize derivations in context-free languages as we can represent derivations using tree structures.

Let us recall the definition of a context-free grammar (CFG). G is context-free if every production is of the form $A \to \alpha$, where $A \in V_N$ and $\alpha \in (V_N \cup \Sigma)^*$.

Derivation Tree

6.1.1 DERIVATION TREES

The derivations in a CFG can be represented using trees. Such trees representing derivations are called derivation trees. We give below a rigorous definition of a derivation tree.

Definition 6.1 A derivation tree (also called a parse tree) for a CFG $G = (V_N, \Sigma, P, S)$ is a tree satisfying the following conditions:

(i) Every vertex has a label which is a variable or terminal or Λ.
(ii) The root has label S.
(iii) The label of an internal vertex is a variable.
(iv) If the vertices $n_1, n_2, \ldots, n_k$ written with labels $X_1, X_2, \ldots, X_k$ are the sons of vertex n with label A, then $A \to X_1X_2 \ldots X_k$ is a production in P.
(v) A vertex n is a leaf if its label is $a \in \Sigma$ or Λ; n is the only son of its father if its label is Λ.

For example, let $G = (\{S, A\}, \{a, b\}, P, S)$, where P consists of $S \to aAS \mid a \mid SS$, $A \to SbA \mid ba$. Figure 6.1 is an example of a derivation tree.

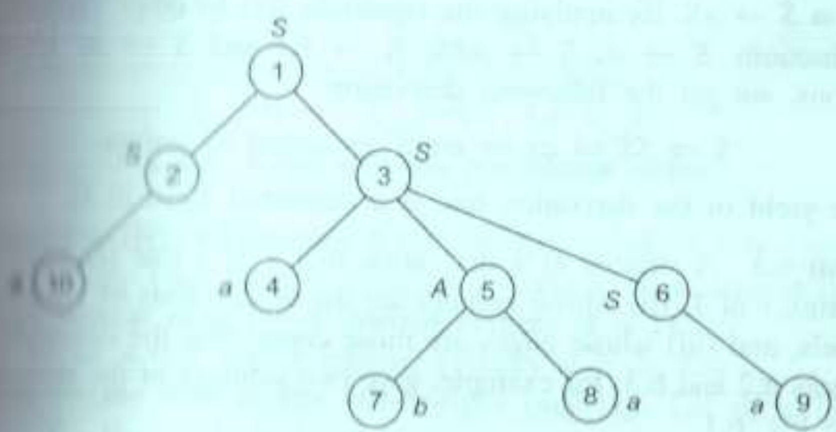


Fig. 6.1 An example of a derivation tree.

Ambiguity in Grammar

.2 AMBIGUITY IN CONTEXT-FREE GRAMMARS

ometimes we come across ambiguous sentences in the language we are using. onsider the following sentence in English: "In books selected information is ven." The word 'selected' may refer to books or information. So the sentence ay be parsed in two different ways. The same situation may arise in context ee languages. The same terminal string may be the yield of two derivation ees. So there may be two different leftmost derivations of w by Theorem 6.2 his leads to the definition of ambiguous sentences in a context-free language

efinition 6.6 A terminal string $w \in L(G)$ is ambiguous if there exist two r more derivation trees for w (or there exist two or more leftmost derivations f w).

Consider, for example, $G = (\{S\}, \{a, b, +, *\}, P, S)$, where P consists f $S \rightarrow S + S \mid S * S \mid a \mid b$. We have two derivation trees for $a + a * b$ given n Fig. 6.10.

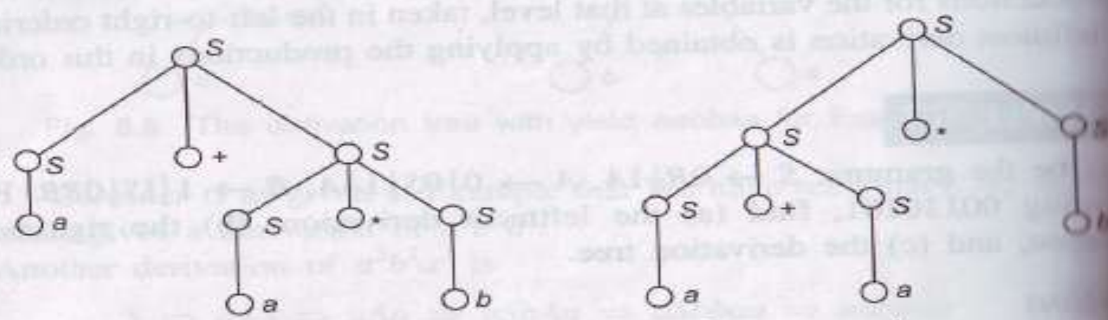


Fig. 6.10 Two derivation trees for $a + a * b$.

The leftmost derivations of $a + a * b$ induced by the two derivation trees are

$$S \Rightarrow S + S \Rightarrow a + S \Rightarrow a + S * S \Rightarrow a + a * S \Rightarrow a + a * b$$

$$S \Rightarrow S * S \Rightarrow S + S * S \Rightarrow a + S * S \Rightarrow a + a * S \Rightarrow a + a * b$$

Therefore, $a + a * b$ is ambiguous.

Ambiguity in Grammar

EXAMPLE 6.4

If G is the grammar $S \rightarrow SbS \mid a$, show that G is ambiguous.

Solution

To prove that G is ambiguous, we have to find a $w \in L(G)$, which is ambiguous. Consider $w = ababa \in L(G)$. Then we get two derivation trees for w (see Fig. 6.11). Thus, G is ambiguous.

Ambiguity in Grammar

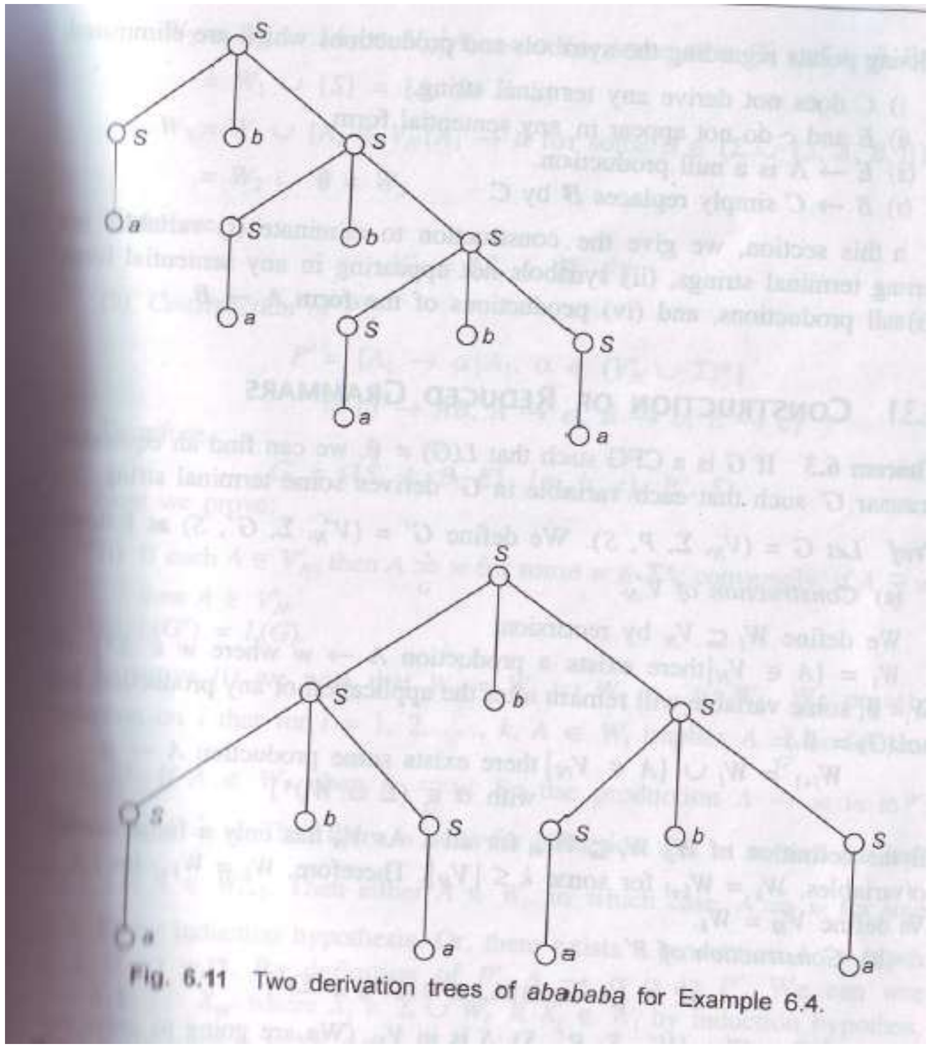


Fig. 6.11 Two derivation trees of *abababa* for Example 6.4.

Useless Symbol

Useless symbols

We now undertake the task of eliminating useless symbols from a grammar. Le $G = (V, T, P, S)$ be a grammar. A symbol X is *useful* if there is a derivatio $S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w$ for some α, β, and w, where w is in T^* (recall our conventio regarding names of symbols and strings). Otherwise X is *useless*. There are tw aspects to usefulness. First some terminal string must be derivable from X an second, X must occur in some string derivable from S. These two conditions ar not, however, sufficient to guarantee that X is useful, since X may occur only i sentential forms that contain a variable from which no terminal string can b derived.

Simplification of Context Free Grammar

.3.1 CONSTRUCTION OF REDUCED GRAMMARS

heorem 6.3 If G is a CFG such that $L(G) \neq \emptyset$, we can find an equivalent rammar G' such that each variable in G' derives some terminal string.

roof Let $G = (V_N, \Sigma, P, S)$. We define $G' = (V'_N, \Sigma, G', S)$ as follows:

(a) *Construction of* V'_N:

We define $W_i \subseteq V_N$ by recursion:

$W_1 = \{A \in V_N \mid$ there exists a production $A \to w$ where $w \in \Sigma^*\}$. (If $W_1 = \emptyset$, some variable will remain after the application of any production, and so $L(G) = \emptyset$.)

$$W_{i+1} = W_i \cup \{A \in V_N \mid \text{there exists some production } A \to \alpha \text{ with } \alpha \in (\Sigma \cup W_i)^*\}$$

By the definition of W_i, $W_i \subseteq W_{i+1}$ for all i. As V_N has only a finite number of variables, $W_k = W_{k+1}$ for some $k \leq |V_N|$. Therefore, $W_k = W_{k+j}$ for $j \geq 1$. We define $V'_N = W_k$.

(b) *Construction of* P':

$$P' = \{A \to \alpha \mid A, \alpha \in (V'_N \cup \Sigma)^*\}$$

We can define $G' = (V'_N, \Sigma, P', S)$. S is in V'_N. (We are going to prove that every variable in V'_N derives some terminal string. So if $S \notin V'_N$, $L(G) = \emptyset$. But $L(G) \neq \emptyset$.)

Before proving that G' is the required grammar, we apply the construction to an example.

EXAMPLE 6.5

Let $G = (V_N, \Sigma, P, S)$ be given by the productions $S \to AB$, $A \to a$, $B \to b$, $B \to C$, $E \to c$. Find G' such that every variable in G' derives some terminal string.

Solution

(a) *Construction of* V'_N:

$W_1 = \{A, B, E\}$ since $A \to a$, $B \to b$, $E \to c$ are productions with a terminal string on the R.H.S.

Simplification of Context Free Grammar

$$W_2 = W_1 \cup \{A_1 \in V_N \mid A_1 \rightarrow \alpha \text{ for some } \alpha \in (\Sigma \cup \{A, B, E\})^*\}$$
$$= W_1 \cup \{S\} = \{A, B, E, S\}$$
$$W_3 = W_2 \cup \{A_1 \in V_N \mid A_1 \rightarrow \alpha \text{ for some } \alpha \in (\Sigma \cup \{S, A, B, E\})^*\}$$
$$= W_2 \cup \emptyset = W_2$$

Therefore,

$$V'_N = \{S, A, B, F\}$$

(b) *Construction of P':*

$$P' = \{A_1 \rightarrow \alpha \mid A_1, \alpha \in (V'_N \cup \Sigma)^*\}$$
$$= \{S \rightarrow AB, A \rightarrow a, B \rightarrow b, E \rightarrow c\}$$

Therefore,

$$G' = (\{S, A, B, E\}, \{a, b, c\}, P', S)$$

Simplification of Context Free Grammar

EXAMPLE 6.7

Find a reduced grammar equivalent to the grammar G whose productions are

$$S \to AB \mid CA, \quad B \to BC \mid AB, \quad A \to a, \quad C \to aB \mid b$$

Solution

Step 1 $W_1 = \{A, C\}$ as $A \to a$ and $C \to b$ are productions with a terminal string on R.H.S.

$$W_2 = \{A, C\} \cup \{A_1 \mid A_1 \to \alpha \text{ with } \alpha \in (\Sigma \cup \{A, C\})^*\}$$

$$= \{A, C\} \cup \{S\} \text{ as we have } S \to CA$$

$$W_3 = \{A, C, S\} \cup \{A_1 \mid A_1 \to \alpha \text{ with } \alpha \in (\Sigma \cup \{S, A, C\})^*\}$$

$$= \{A, C, S\} \cup \emptyset$$

As $W_3 = W_2$,

$$V'_N = W_2 = \{S, A, C\}$$

$$P' = \{A_1 \to \alpha \mid A_1, \alpha \in (V'_N \cup \Sigma)^*\}$$

$$= \{S \to CA, A \to a, C \to b\}$$

Thus,

$$G_1 = (\{S, A, C\}, \{a, b\}, \{S \to CA, A \to a, C \to b\}, S)$$

Simplification of Context Free Grammar

Step 2 We have to apply Theorem 6.4 to G_1. Thus,

$$W_1 = \{S\}$$

As we have production $S \rightarrow CA$ and $S \in W_1$, $W_2 = \{S\} \cup \{A, C\}$
As $A \rightarrow a$ and $C \rightarrow b$ are productions with $A, C \in W_2$, $W_3 = \{S, A, C, a, b\}$

$$\text{As } W_3 = V'_N \cup \Sigma,\ P'' = \{S \rightarrow a \mid A_1 \in W_3\} = P'$$

Therefore,

$$G' = (\{S, A, C\}, \{a, b\}, \{S \rightarrow CA, A \rightarrow a, C \rightarrow b\}, S)$$

is the reduced grammar.

Simplification of Context Free Grammar

EXAMPLE 6.8

Construct a reduced grammar equivalent to the grammar

$$S \to aAa, \quad A \to Sb \mid bCC \mid DaA, \quad C \to abb \mid DD,$$
$$E \to aC, \quad D \to aDA$$

Solution

Step 1 $W_1 = \{C\}$ as $C \to abb$ is the only production with a terminal string in the R.H.S.

$$W_2 = \{C\} \cup \{E, A\}$$

as $E \to aC$ and $A \to bCC$ are productions with R.H.S. in $(\Sigma \cup \{C\})^*$

$$W_3 = \{C, E, A\} \cup \{S\}$$

as $S \to aAa$ and aAa is in $(\Sigma \cup W_2)^*$

$$W_4 = W_3 \cup \emptyset$$

Hence,

$$V'_N = W_3 = \{S, A, C, E\}$$
$$P' = \{A_1 \to \alpha \mid \alpha \in (V_N \cup \Sigma)^*\}$$
$$= \{S \to aAa, A \to Sb \mid bCC, C \to abb, E \to aC\}$$
$$G_1 = (V'_N, \{a, b\}, P', S)$$

Step 2 We have to apply Theorem 6.4 to G_1. We start with

$$W_1 = \{S\}$$

As we have $S \to aAa$,

$$W_2 = \{S\} \cup \{A, a\}$$

As $A \to Sb \mid bCC$,

$$W_3 = \{S, A, a\} \cup \{S, b, C\} = \{S, A, C, a, b\}$$

As we have $C \to abb$,

Simplification of Context Free Grammar

ce,

$$P'' = \{A_1 \rightarrow \alpha \mid A_1 \in W_3\}$$

$$= \{S \rightarrow aAa,\ A \rightarrow Sb \mid bCC,\ C \rightarrow abb\}$$

refore,

$$G' = (\{S, A, C\}, \{a, b\}, P'', S)$$

e reduced grammar.

Elimination of Null Production

inition 6.9 A variable A in a context-free grammar is nullable if A [illegible]

orem 6.6 If $G = (V_N, \Sigma, P, S)$ is a context-free grammar, then we [illegible] a context-free grammar G_1 having no null prodctions such that $L(G_1)$ [illegible] $) - \{\Lambda\}$.

of We construct $G_1 = (V_N, \Sigma, P', S)$ as follows:

p 1 *Construction of the set of nullable variables:*

find the nullable variables recursively:

(i) $W_1 = \{A \in V_N \mid A \rightarrow \Lambda \text{ is in } P\}$

(ii) $W_{i+1} = W_i \cup \{A \in V_N \mid$ there exists a production $A \rightarrow \alpha$ with $\alpha \in W_i^*\}$

definition of W_i, $W_i \subseteq W_{i+1}$ for all i. As V_N is finite, $W_{k+1} = W_k$ for [illegible] $|V_N|$. So, $W_{k+j} = W_k$ for all j. Let $W = W_k$. W is the set of all nullable [illegible] iables.

p 2 (i) *Construction of P':*

production whose R.H.S. does not have any nullable variable is included [illegible] P'.

(ii) If $A \rightarrow X_1 X_2 \ldots X_k$ is in P, the productions of the form $A \rightarrow$ [illegible] α_k are included in P', where $\alpha_j = X_i$ if $X_i \notin W$, $\alpha_i = X_i$ or Λ if $X_i \in$ [illegible] and $\alpha_1\alpha_2 \ldots \alpha_k \neq \Lambda$. Actually, (ii) gives several productions in P'. The [illegible]

Elimination of Null Production

R.H.S. of $A \to X_1X_2 \ldots X_k$ or by erasing some or all nullable variables provided some symbol appears on the R.H.S. after erasing.

Let $G_1 = (V_N, \Sigma, P', S)$. G_1 has no null productions.

Before proving that G_1 is the required grammar, we apply the construction to an example.

EXAMPLE 6.9

Consider the grammar G whose productions are $S \to aS \mid AB$, $A \to \Lambda$, $B \to \Lambda$, $D \to b$. Construct a grammar G_1 without null productions generating $L(G) - \{\Lambda\}$.

Solution

Step 1 *Construction of the set W of all nullable variables:*

$$W_1 = \{A_1 \in V_N \mid A_1 \to \Lambda \text{ is a production in } G\}$$
$$= \{A, B\}$$

$$W_2 = \{A, B\} \cup \{S\} \text{ as } S \to AB \text{ is a production with } AB \in W_1^*$$
$$= \{S, A, B\}$$

$$W_3 = W_2 \cup \emptyset = W_2$$

Thus,

$$W = W_2 = \{S, A, B\}$$

Step 2 *Construction of P':*

(i) $D \to b$ is included in P'.
(ii) $S \to aS$ gives rise to $S \to aS$ and $S \to a$.
(iii) $S \to AB$ gives rise to $S \to AB$, $S \to A$ and $S \to B$.

(Note: We cannot erase both the nullable variables A and B in $S \to AB$ as we will get $S \to \Lambda$ in that case.)

Hence the required grammar without null productions is

$$G_1 = (\{S, A, B, D\}, \{a, b\}, P, S)$$

where P' consists of

$$D \to b,\ S \to aS,\ S \to AB,\ S \to a,\ S \to A,\ S \to B$$

Elimination of Unit Production

6.1.3 ELIMINATION OF UNIT PRODUCTIONS

A context-free grammar may have productions of the form $A \rightarrow B$, $A, B \in V_N$.

Consider, for example, G as the grammar $S \rightarrow A$, $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow a$. It is easy to see that $L(G) = \{a\}$. The productions $S \rightarrow A$, $A \rightarrow B$, $B \rightarrow C$ are useful just to replace S by C. To get a terminal string, we need $C \rightarrow a$. If G_1 is $S \rightarrow a$, then $L(G_1) = L(G)$.

The next construction eliminates productions of the form $A \rightarrow B$.

Definition 6.10 A unit production (or a chain rule) in a context-free grammar G is a production of the form $A \rightarrow B$, where A and B are variables in G.

Theorem 6.7 If G is a context-free grammar, we can find a context-free grammar G_1 which has no null productions or unit productions such that $L(G_1) = L(G)$.

Elimination of Unit Production

EXAMPLE 6.10

Let G be $S \to AB$, $A \to a$, $B \to C \mid b$, $C \to D$, $D \to E$ and $E \to a$. Eliminate unit productions and get an equivalent grammar.

Solution

Step 1 $W_0(S) = \{S\}$, $W_1(S) = W_0(S) \cup \emptyset$

Hence $W(S) = \{S\}$. Similarly,

$W(A) = \{A\}$, $W(E) = \{E\}$

$W_0(B) = \{B\}$, $W_1(B) = \{B\} \cup \{C\} = \{B, C\}$

$W_2(B) = \{B, C\} \cup \{D\}$, $W_3(B) = \{B, C, D\} \cup \{E\}$, $W_4(B) = W_3(B)$

Therefore,

$$W(B) = \{B, C, D, E\}$$

Similarly,

$W_0(C) = \{C\}$, $W_1(C) = \{C, D\}$, $W_2\{C\} = \{C, D, E\} = W_3(C)$

Therefore,

$W(C) = \{C, D, E\}$, $W_0(D) = \{D\}$

Hence,

$$W_1(D) = \{D, E\} = W_2(D)$$

Thus,

$$W(D) = \{D, E\}$$

Step 2 The productions in G_1 are

$S \to AB$, $A \to a$, $E \to a$

$B \to b \mid a$, $C \to a$, $D \to a$

By construction, G_1 has no unit productions.

To complete the proof we have to show that $L(G') = L(G_1)$.

Normal Form of Grammar
Chomsky Normal Form

6.4 NORMAL FORMS FOR CONTEXT-FREE GRAMMARS

In a context-free grammar, the R.H.S. of a production can be any string of variables and terminals. When the productions in G satisfy certain restrictions, then G is said to be in a 'normal form'. Among several 'normal forms' we study two of them in this section—the Chomsky normal form (CNF) and the Greibach normal form.

6.4.1 CHOMSKY NORMAL FORM

In the Chomsky normal form (CNF), we have restrictions on the length of R.H.S. and the nature of symbols in the R.H.S. of productions.

Definition 6.11 A context-free grammar G is in Chomsky normal form if every production is of the form $A \to a$, or $A \to BC$, and $S \to \Lambda$ is in G if

CNF

L(G). When Λ is in $L(G)$, we assume that S does not appear on the
. of any production.
or example, consider G whose productions are $S \to AB \mid \Lambda$, $A \to a$,
b. Then G is in Chomsky normal form.

rk For a grammar in CNF, the derivation tree has the following
rty: Every node has atmost two descendants—either two internal vertices
single leaf.
When a grammar is in CNF, some of the proofs and constructions are
er.

uction to Chomsky Normal Form

we develop a method of constructing a grammar in CNF equivalent to a
n context-free grammar. Let us first consider an example. Let G be $S \to$
$\mid aC$, $A \to a$, $B \to b$, $C \to c$. Except $S \to aC \mid ABC$, all the other
luctions are in the form required for CNF. The terminal a in $S \to aC$ can
eplaced by a new variable D. By adding a new production $D \to a$, the effect
pplying $S \to aC$ can be achieved by $S \to DC$ and $D \to a$. $S \to ABC$ is not
ne required form, and hence this production can be replaced by $S \to AE$ and
$\to BC$. Thus, an equivalent grammar is $S \to AE \mid DC$, $E \to BC$, $A \to a$,
$\to b$, $C \to c$, $D \to a$.
The techniques applied in this example are used in the following theorem.

eorem 6.8 (Reduction to Chomsky normal form). For every context-free
mmar, there is an equivalent grammar G_2 in Chomsky normal form.

oof (Construction of a grammar in CNF)

ep 1 *Elimination of null productions and unit productions:*
e apply Theorem 6.6 to eliminate null productions. We then apply
eorem 6.7 to the resulting grammar to eliminate chain productions. Let the
ammar thus obtained be $G = (V_N, \Sigma, P, S)$.

ep 2 *Elimination of terminals on R.H.S.:*
e define $G_1 = (V'_N, \Sigma, P_1, S')$, where P_1 and V'_N are constructed as follows:

(i) All the productions in P of the form $A \to a$ or $A \to BC$ are included in P_1. All the variables in V_N are included in V'_N.
(ii) Consider $A \to X_1X_2 \ldots X_n$ with some terminal on R.H.S. If X_i is a terminal, say a_i, add a new variable C_{a_i} to V'_N and $C_{a_i} \to a_i$ to P_1. In production $A \to X_1X_2 \ldots X_n$, every terminal on R.H.S. is replaced by the corresponding new variable and the variables on the R.H.S. are retained. The resulting production is added to P_1. Thus, we get $G_1 = (V'_N, \Sigma, P_1, S)$.

Step 3 *Restricting the number of variables on R.H.S.:*
For any production in P_1, the R.H.S. consists of either a single terminal or

CNF

A in $S \to A$) or two or more variables. We define $G_2 = (V_N'', \Sigma, P_2, S)$ as follows:

(ii) All productions in P_1 are added to P_2 if they are in the required form. All the variables in V_N' are added to V_N''.

(iii) Consider $A \to A_1A_2 \ldots A_m$, where $m \geq 3$. We introduce new productions $A \to A_1C_1$, $C_1 \to A_2C_2$, $\ldots$, $C_{m-2} \to A_{m-1}A_m$ and new variables $C_1, C_2, \ldots, C_{m-2}$. These are added to P'' and V_N'', respectively.

Thus we get G_2 in Chomsky normal form.

Before proving that G_2 is the required equivalent grammar, we apply the construction to the context-free grammar given in Example 6.11.

CNF

EXAMPLE 6.11

Reduce the following grammar G to CNF. G is $S \rightarrow aAD$, $A \rightarrow aB \mid bAB$, $B \rightarrow b$, $D \rightarrow d$.

Solution

As there are no null productions or unit productions, we can proceed to step 2.

Step 2 Let $G_1 = (V'_N, \{a, b, d\}, P_1, S)$, where P_1 and V'_N are constructed as follows:

(i) $B \rightarrow b$, $D \rightarrow d$ are included in P_1.

(ii) $S \rightarrow aAD$ gives rise to $S \rightarrow C_aAD$ and $C_a \rightarrow a$.

$A \rightarrow aB$ gives rise to $A \rightarrow C_aB$.

$A \rightarrow bAB$ gives rise to $A \rightarrow C_bAB$ and $C_b \rightarrow b$.

$V'_N = \{S, A, B, D, C_a, C_b\}$.

Step 3 P_1 consists of $S \rightarrow C_aAD$, $A \rightarrow C_aB \mid C_bAB$, $B \rightarrow b$, $D \rightarrow d$, $C_a \rightarrow a$, $C_b \rightarrow b$.

$A \rightarrow C_aB$, $B \rightarrow b$, $D \rightarrow d$, $C_a \rightarrow a$, $C_b \rightarrow b$ are added to P_2

$S \rightarrow C_aAD$ is replaced by $S \rightarrow C_aC_1$ and $C_1 \rightarrow AD$.

$A \rightarrow C_bAB$ is replaced by $A \rightarrow C_bC_2$ and $C_2 \rightarrow AB$.

$G_2 = (\{S, A, B, D, C_a, C_b, C_1, C_2\}, \{a, b, d\}, P_2, S)$

P_2 consists of $S \rightarrow C_aC_1$, $A \rightarrow C_aB \mid C_bC_2$, $C_1 \rightarrow AD$, $C_2 \rightarrow AB$, $B \rightarrow b$, $D \rightarrow d$, $C_a \rightarrow a$, $C_b \rightarrow b$. G_2 is in CNF and equivalent to G.

CNF

EXAMPLE 6.13

Find a grammar in CNF equivalent to the grammar

$$S \rightarrow \sim S \mid [S \supset S] \mid p \mid q \qquad (S \text{ being the only variable})$$

Solution

As the given grammar has no unit or null productions, we omit step 1 and proceed to step 2.

Step 2 Let $G_1 = (V'_N, \Sigma, P_1, S)$, where P_1 and V'_N are constructed as follows:

(i) $S \rightarrow p \mid q$ are added to P_1.

(ii) $S \rightarrow \sim S$ induces $S \rightarrow AS$ and $A \rightarrow \sim$.

(iii) $S \rightarrow [S \supset S]$ induces $S \rightarrow BSCSD$, $B \rightarrow [$, $C \rightarrow \supset$, $D \rightarrow]$

$$V'_N = \{S, A, B, C, D\}$$

Step 3 P_1 consists of $S \rightarrow p \mid q$, $S \rightarrow AS$, $A \rightarrow \sim$, $B \rightarrow [$, $C \rightarrow \supset$, $D \rightarrow]$, $S \rightarrow BSCSD$.

$S \rightarrow BSCSD$ is replaced by $S \rightarrow BC_1$, $C_1 \rightarrow SC_2$, $C_2 \rightarrow CC_3$, $C_3 \rightarrow SD$.

Let

$$G_2 = (\{S, A, B, C, D, C_1, C_2, C_3\}, \Sigma, P_2, S)$$

Then P_2 consists of $S \rightarrow p \mid q \mid AS \mid BC_1$, $A \rightarrow \sim$, $B \rightarrow [$, $C \rightarrow \supset$, $D \rightarrow]$, $C_1 \rightarrow SC_2$, $C_2 \rightarrow CC_3$, $C_3 \rightarrow SD$. G_2 is in CNF and equivalent to the given grammar.

Greibach Normal Form(GNF)

4.2 GREIBACH NORMAL FORM

eibach normal form (GNF) is another normal form quite useful in some oofs and constructions. A context-free grammar generating the set accepted a pushdown automaton is in Greibach normal form as will be seen in eorem 7.4.

finition 6.12 A context-free grammar is in Greibach normal form if every oduction is of the form $A \rightarrow a\alpha$, where $\alpha \in V_N^*$ and $a \in \Sigma$(α may be Λ), d $S \rightarrow \Lambda$ is in G if $\Lambda \in L(G)$. When $\Lambda \in L(G)$, we assume that S es not appear on the R.H.S. of any production. For example, G given by $\rightarrow aAB \mid \Lambda$, $A \rightarrow bC$, $B \rightarrow b$, $C \rightarrow c$ is in GNF.

Greibach Normal Form(GNF)

The lemma is useful to eliminate A from the R.H.S. of $A \to A\alpha$

Lemma 6.2 Let $G = (V_N, \Sigma, P, S)$ be a context-free grammar. Let the set of A-productions be $A \to A\alpha_1 | \ldots A\alpha_r | \beta_1 | \ldots | \beta_s$ (β_i's do not start with A).

Let Z be a new variable. Let $G_1 = (V_N \cup \{Z\}, \Sigma, P_1, S)$, where P_1 is defined as follows:

(i) The set of A-productions in P_1 are $A \to \beta_1 | \beta_2 | \ldots | \beta_s$

$$A \to \beta_1 Z | \beta_2 Z | \ldots | \beta_s Z$$

(ii) The set of Z-productions in P_1 are $Z \to \alpha_1 | \alpha_2 | \ldots | \alpha_r$

$$Z \to \alpha_1 Z | \alpha_2 Z \ldots | \alpha_r Z$$

(iii) The productions for the other variables are as in P. Then G_1 is a CFG and equivalent to G.

GNF

EXAMPLE 6.15

Construct a grammar in Greibach normal form equivalent to the grammar $S \rightarrow AA \mid a$, $A \rightarrow SS \mid b$.

Solution

The given grammar is in CNF. S and A are renamed as A_1 and A_2, respectively. So the productions are $A_1 \rightarrow A_1A_2 \mid a$ and $A_2 \rightarrow A_1A_1 \mid b$. As the

grammar has no null productions and is in CNF we need not carry out
1. So we proceed to step 2.

2 (i) A_1-productions are in the required form. They are $A_1 \rightarrow A_1A_2 \mid a$
(ii) $A_2 \rightarrow b$ is in the required form. Apply Lemma 6.1 to $A_2 \rightarrow A_1A_1$. Thus the resulting productions are $A_2 \rightarrow A_2A_2A_1$, $A_2 \rightarrow aA_1$. Thus the productions are

$$A_2 \rightarrow A_2A_2A_1, \quad A_2 \rightarrow aA_1, \quad A_2 \rightarrow b$$

p 3 We have to apply Lemma 6.2 to A_2-productions as we have $A_2 \rightarrow A_2A_2A_1$. Let Z_2 be the new variable. The resulting productions are

$$A_2 \rightarrow aA_1, \quad A_2 \rightarrow b$$
$$A_2 \rightarrow aA_1Z_2, \quad A_2 \rightarrow bZ_2$$
$$Z_2 \rightarrow A_2A_1, \quad Z_2 \rightarrow A_2A_1Z_2.$$

GNF

ep 4 (i) The A_2-productions are $A_2 \to aA_1 | b | aA_1Z_2 | bZ_2$.

(ii) Among the A_1-productions we retain $A_1 \to a$ and elimin[ate] ... $A_1 \to A_2A_2$ using Lemma 6.1. The resulting productions are $A_1 \to aA_1A_2 | bA_2$ ... $\to aA_1Z_2A_2 | bZ_2A_2$. The set of all (modified) A_1-productions is

$$A_1 \to a | aA_1A_2 | bA_2 | aA_1Z_2A_2 | bZ_2A_2$$

tep 5 The Z_2-productions to be modified are $Z_2 \to A_2A_1$, $Z_2 \to A_2A_1Z_2$.
We apply Lemma 6.1 and get

$$Z_2 \to aA_1A_1 | bA_1 | aA_1Z_2A_1 | bZ_2A_1$$

$$Z_2 \to aA_1A_1Z_2 | bA_1Z_2 | aA_1Z_2A_1Z_2 | bZ_2A_1Z_2$$

Hence the equivalent grammar is

$$G' = (\{A_1, A_2, Z_2\}, \{a, b\}, P_1, A_1)$$

where P_1 consists of

$$A_1 \to a | aA_1A_2 | bA_2 | aA_1Z_2A_1 | bZ_2A_2$$

$$A_2 \to aA_1 | b | aA_1Z_2 | bZ_2$$

$$Z_2 \to aA_1A_1 | bA_1 | aA_1Z_2A_1 | bZ_2A_1$$

$$Z_2 \to aA_1A_1Z_2 | bA_1Z_2 | aA_1Z_2A_1Z_2 | bZ_2A_1Z_2$$

GNF

EXAMPLE 6.16

Convert the grammar $S \to AB$, $A \to BS \mid b$, $B \to SA \mid a$ into GNF.

Solution

As the given grammar is in CNF, we can omit step 1 and proceed to step 2 after renaming S, A, B as A_1, A_2, A_3, respectively. The productions are $A_1 \to A_2A_3$, $A_2 \to A_3A_1 \mid b$, $A_3 \to A_1A_2 \mid a$.

Step 2 (i) The A_1-production $A_1 \to A_2A_3$ is in the required form.
(ii) The A_2-productions $A_2 \to A_3A_1 \mid b$ are in the required form.
(iii) $A_3 \to a$ is in the required form.

Apply Lemma 6.1 to $A_3 \to A_1A_2$. The resulting productions are $A_3 \to A_2A_3A_2$. Applying the lemma once again to $A_3 \to A_2A_3A_2$, we get

$$A_3 \to A_3A_1A_3A_2 \mid bA_3A_2.$$

Step 3 The A_3-productions are $A_3 \to a \mid bA_3A_2$ and $A_3 \to A_3A_1A_3A_2$. As we have $A_3 \to A_3A_1A_3A_2$, we have to apply Lemma 6.2 to A_3-productions. Let Z_3 be the new variable. The resulting productions are

$$A_3 \to a \mid bA_3A_2, \quad A_3 \to aZ_3 \mid bA_3A_2Z_3$$

$$Z_3 \to A_1A_3A_2, \quad Z_3 \to A_1A_3A_2Z_3$$

GNF

Step 4 (i) The A_3-productions are

$$A_3 \rightarrow a \mid bA_3A_2 \mid aZ_3 \mid bA_3A_2Z_3 \quad (6.9)$$

(ii) Among the A_2-productions, we retain $A_2 \rightarrow b$ and eliminate $A_2 \rightarrow A_1A_1$ using Lemma 6.1. The resulting productions are

$$A_2 \rightarrow aA_1 \mid bA_3A_2A_1 \mid aZ_3A \mid bA_3A_2Z_3A_1$$

The modified A_2-productions are

$$A_2 \rightarrow b \mid aA_1 \mid bA_3A_2A_1 \mid aZ_3A_1 \mid bA_3A_2Z_3A_1 \quad (6.10)$$

(iii) We apply Lemma 6.1 to $A_1 \rightarrow A_2A_3$ to get

$$A_1 \rightarrow bA_3 \mid aA_1A_3 \mid bA_3A_2A_1A_3 \mid aZ_3A_1A_3 \mid bA_3A_2Z_3A_1A_3 \quad (6.11)$$

Step 5 The Z_3-productions to be modified are

$$Z_3 \rightarrow A_1A_3A_2 \mid A_1A_3A_2Z_3$$

We apply Lemma 6.1 and get

$$Z_3 \rightarrow bA_3A_3A_2 \mid bA_3A_2Z_3$$
$$Z_3 \rightarrow aA_1A_3A_3A_2 \mid aA_1A_3A_3A_2Z_3$$
$$Z_3 \rightarrow bA_3A_2A_1A_3A_3A_2 \mid bA_3A_2A_1A_3A_3A_2Z_3 \quad (6.12)$$
$$Z_3 \rightarrow aZ_3A_1A_3A_3A_2 \mid aZ_3A_1A_3A_3A_2Z_3$$
$$Z_3 \rightarrow bA_3A_2Z_3A_1A_3A_3A_2 \mid bA_3A_2Z_3A_1A_3A_3A_2Z_3$$

GNF

EXAMPLE 6.17

Find a grammar in GNF equivalent to the grammar

$$E \to E + T \mid T,\ T \to T * F \mid F,\ F \to (E) \mid a$$

Solution

Step 1 We first eliminate unit productions. Hence

$$W_0(E) = \{E\}, \quad W_1(E) = \{E\} \cup \{T\} = \{E, T\}$$

$$W_2(E) = \{E, T\} \cup \{F\} = \{E, T, F\}$$

So,

$$W(E) = \{E, T, F\}$$

So,

$$W_0(T) = \{T\}, \quad W_1(T) = \{T\} \cup \{F\} = \{T, F\}$$

Thus,

$$W(T) = \{T, F\}$$

$$W_0(F) = \{F\}, \quad W_1(F) = \{F\} = W(F)$$

The equivalent grammar without unit productions is, therefore, $G_1 = (V_N,$ $\Sigma, P_1, S)$, where P_1 consists of

(i) $E \to E + T \mid T * F \mid (E) \mid a$
(ii) $T \to T * F \mid (E) \mid a$, and
(iii) $F \to (E) \mid a$.

GNF

We apply step 2 of reduction to CNF. We introduce new variables A, B, C corresponding to $+$, $*$,). The modified productions are

(i) $E \rightarrow EAT \mid TBF \mid (EC \mid a$
(ii) $T \rightarrow TBF \mid (EC \mid a$
(iii) $F \rightarrow (EC \mid a$
(iv) $A \rightarrow +$, $B \rightarrow *$, $C \rightarrow$)

The variables A, B, C, F, T and E are renamed as A_1, A_2, A_3, A_4, A_5, A_6. Then the productions become

$$A_1 \rightarrow +, \quad A_2 \rightarrow *, \quad A_3 \rightarrow), \quad A_4 \rightarrow (A_6A_3 \mid a$$

$$A_5 \rightarrow A_5A_2A_4 \mid (A_6A_3 \mid a$$

$$A_6 \rightarrow A_6A_1A_5 \mid A_5A_2A_4 \mid (A_6A_3 \mid a$$

Step 2 We have to modify only the A_5- and A_6-productions. $A_5 \rightarrow A_5A_2A_4$ can be modified by using Lemma 6.2. The resulting productions are

$$A_5 \rightarrow (A_6A_3 \mid a, \quad A_5 \rightarrow (A_6A_3Z_5 \mid aZ_5$$

$$Z_5 \rightarrow A_2A_4 \mid A_2A_4Z_5$$

$A_6 \rightarrow A_5A_2A_4$ can be modified by using Lemma 6.1. The resulting productions are

$$A_6 \rightarrow (A_6A_3A_2A_4 \mid aA_2A_4 \mid (A_6A_3Z_5A_2A_4 \mid aZ_5A_2A_4$$

$A_6 \rightarrow (A_6A_3 \mid a$ are in the proper form.

GNF

Step 3 $A_6 \rightarrow A_6A_1A_5$ can be modified by using Lemma 6.2. The resulting productions give all the A_6-productions:

$$A_6 \rightarrow (A_6A_3A_2A_4 \mid aA_2A_4 \mid (A_6A_3Z_5A_2A_4$$

$$A_6 \rightarrow aZ_5A_2A_4 \mid (A_6A_3 \mid a \qquad (6.15)$$

$$A_6 \rightarrow (A_6A_3A_2A_4Z_6 \mid aA_2A_4Z_6 \mid (A_6A_3Z_5A_2A_4Z_6$$

$$A_6 \rightarrow aZ_5A_2A_4Z_6 \mid (A_6A_3Z_6 \mid aZ_6 \qquad (6.16)$$

$$A_6 \rightarrow A_1A_5 \mid A_1A_5Z_6$$

Step 4 The step is not necessary as A_i-productions for i = 5, 4, 3, 2, 1 are in the required form.

Step 5 The Z_5-productions are $Z_5 \rightarrow A_2A_4 \mid A_2A_4Z_5$. These can be modified as

$$Z_5 \rightarrow \ * A_4 \mid * A_4Z_5 \qquad (6.17)$$

The Z_6-productions are $Z_6 \rightarrow A_1A_5 \mid A_1A_5Z_6$. These can be modified as

$$Z_6 \rightarrow + A_5 \mid + A_5Z_6 \qquad (6.18)$$

The required grammar in GNF is given by (6.13)–(6.18).

Exercise

EXERCISES

6.1 Find a derivation tree of $a * b + a * b$ given that $a * b + a * b$ [illegible] $L(G)$, where G is given by $S \to S + S | S * S$, $S \to a | b$.

6.2 A context-free grammar G has the following productions:

$$S \to 0S0 | 1S1 | A, \quad A \to 2B3, \quad B \to 2B3 | 3$$

Describe the language generated by the parameters.

6.3 A derivation tree of a sentential form of a grammar G is given [illegible] Fig. 6.15.

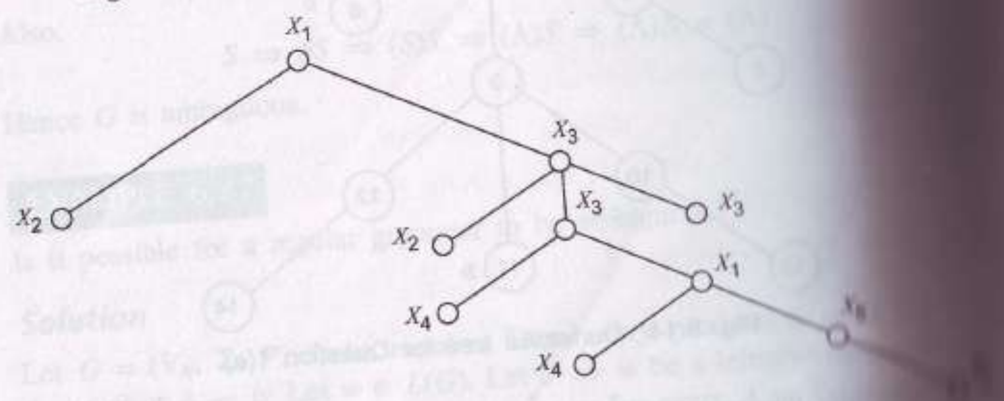


Fig. 6.15 Derivation tree for Exercise 6.3.

(a) What symbols are necessarily in V_N?
(b) What symbols are likely to be in Σ?
(c) Determine if the following strings are sentential forms: (i) [illegible] (ii) $X_2X_2X_3X_2X_3X_3$, and (iii) $X_2X_4X_4X_2$.

6.4 Find (i) a leftmost derivation, (ii) a rightmost derivation, and (iii) [illegible] derivation which is neither leftmost nor rightmost of [illegible] given that *abababa* is in $L(G)$, where G is the grammar [illegible]

Exercise

6.5 Consider the following productions:

$$S \to aB \mid bA$$

$$A \to aS \mid bAA \mid a$$

$$B \to bS \mid aBB \mid b$$

For the string *aaabbabbba*, find

(a) the leftmost derivation,

(b) the rightmost derivation, and

(c) the parse tree.

6.6 Show that the grammar $S \to a \mid abSb \mid aAb$, $A \to bS \mid aAAb$ is ambiguous.

6.7 Show that the grammar $S \to aB \mid ab$, $A \to aAB \mid a$, $B \to ABb \mid b$ is ambiguous.

6.8 Show that if we apply Theorem 6.4 first and then Theorem 6.3 to a grammar G, we may not get a reduced grammar.

6.9 Find a reduced grammar equivalent to the grammar $S \to aAa$, $A \to bBB$, $B \to ab$, $C \to aB$.

6.10 Given the grammar $S \to AB$, $A \to a$, $B \to C \mid b$, $C \to D$, $D \to E$, $E \to a$, find an equivalent grammar which is reduced and has no unit productions.

6.11 Show that for getting an equivalent grammar in the most simplified form, we have to eliminate unit productions first and then the redundant symbols.

Exercise

Reduce the following grammars to Chomsky normal form:

(a) $S \to 1A \mid 0B, \quad A \to 1AA \mid 0S \mid 0, \quad B \to 0BB \mid 1S \mid 1$

(b) $G = (\{S\}, \{a, b, c\}, \{S \to a \mid b \mid cSS\}, S)$

(c) $S \to abSb \mid a \mid aAb, \quad A \to bS \mid aAAb.$

Reduce the grammars given in Exercises 6.1, 6.2, 6.6, 6.7, 6.9, 6.10 to Chomsky normal form.

Reduce the following grammars to Greibach normal form:

(a) $S \to SS, \quad S \to 0S1 \mid 01$

(b) $S \to AB, \quad A \to BSB, \quad A \to BB, \quad B \to aAb, \quad B \to a, \quad A \to b$

(c) $S \to A0, \quad A \to 0B, \quad B \to A0, \quad B \to 1$

Reduce the grammars given in Exercises 6.1, 6.2, 6.6, 6.7, 6.9, 6.10 to Greibach normal form.

Construct the grammars in Chomsky normal form generating the following:

(a) $\{wcw^T \mid w \in 0\ \{a, b\}^*\}$,

(b) the set of all strings over $\{a, b\}$ consisting of equal number of a's and b's.

Exercise

(c) $\{a^m b^n \mid m \neq n,\ m, n \geq 1\}$, and
(d) $\{a^n b^m c^n \mid m, n \geq 1\}$.

Construct grammars in Greibach normal form generating
in Exercise 6.16.

If $w \in L(G)$ and $|w| = k$, where G is in (i) Chomsky
(ii) Greibach normal form, what can you say about the
in the derivation of w?

9 Show that the language $\{a^{n^2} \mid n \geq 1\}$ is not context-free.

0 Show that the following are not context-free languages
(a) The set of all strings over $\{a, b, c\}$ in which the
occurrences of a, b, c is the same.
(b) $\{a^m b^m c^n \mid m \leq n \leq 2m\}$.
(c) $\{a^m b^n \mid n = m^2\}$.

right-linear

Relationship between Languages

LANGUAGES AND THEIR RELATION

In this section we discuss the relation between the classes of languages that we have defined under the Chomsky classification.

Let $\mathcal{L}_0$, $\mathcal{L}_{csl}$, $\mathcal{L}_{cfl}$ and $\mathcal{L}_{rl}$ denote the family of type 0 languages, context-sensitive languages, context-free languages and regular languages, respectively.

Property 1 From the definition, it follows that $\mathcal{L}_{rl} \subseteq \mathcal{L}_{cfl}$, $\mathcal{L}_{csl} \subseteq \mathcal{L}_0$, $\mathcal{L}_{cfl} \subseteq \mathcal{L}_0$.

Property 2 $\mathcal{L}_{cfl} \subseteq \mathcal{L}_{csl}$. The inclusion relation is not immediate as we allow $A \to \Lambda$ in context-free grammars even when $A \neq S$, but not in context-sensitive grammars (we allow only $S \to \Lambda$ in context-sensitive grammars). In Chapter 6 we prove that a context-free grammar G with productions of the form $A \to \Lambda$ is equivalent to a context-free grammar G_1 which has no productions of the form $A \to \Lambda$ (except $S \to \Lambda$). Also, when G_1 has $S \to \Lambda$, S does not appear on the right-hand side of any production. So G_1 is context-sensitive. This proves $\mathcal{L}_{cfl} \subseteq \mathcal{L}_{csl}$.

Property 3 $\mathcal{L}_{rl} \subseteq \mathcal{L}_{cfl} \subseteq \mathcal{L}_{csl} \subseteq \mathcal{L}_0$. This follows from properties 1 and 2.

Property 4 $\mathcal{L}_{rl} \subsetneq \mathcal{L}_{cfl} \subsetneq \mathcal{L}_{csl} \subsetneq \mathcal{L}_0$.

Properties of Language(Operation on language)

consider the effect of applying set operations on $\mathcal{L}_0$, $\mathcal{L}_{csl}$, $\mathcal{L}_{cfl}$, $\mathcal{L}_{rl}$. Let
nd B be any two sets of strings. The concatenation AB of A and B is
ined by $AB = \{uv \mid u \in A, v \in B\}$. (Here, uv is the concatenation of the
ngs u and v.)

We define A^1 as A and A^{n+1} as A^nA for all $n \geq 1$.

The transpose set A^T of A is defined by

$$A^T = \{u^T \mid u \in A\}$$

eorem 4.5 Each of the classes $\mathcal{L}_0$, $\mathcal{L}_{csl}$, $\mathcal{L}_{cfl}$, $\mathcal{L}_{rl}$ is closed under union.

oof Let L_1 and L_2 be two languages of the same type i. We can apply
eorem 4.1 to get grammars

$$G_1 = (V'_N, \Sigma_1, P_1, S_1) \quad \text{and} \quad G_2 = (V''_N, \Sigma_2, P_2, S_2)$$

type i generating L_1 and L_2, respectively. So any production in G_1 or G_2
either $\alpha \to \beta$, where α, β contain only variables or $A \to a$, where $A \in V_N$
$\in \Sigma$.

We can further assume that $V'_N \cap V''_N = \emptyset$. (This is achieved by renaming
e variables of V''_N if they occur in V'_N.)

Define a new grammar G_u as follows:

$$G_u = (V'_N \cup V''_N \cup \{S\}, \Sigma_1 \cup \Sigma_2, P_u, S)$$

here S is a new symbol, i.e. $S \notin V'_N \cup V''_N$

$$P_u = P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}$$

Properties of Language(Operation on language)

We prove $L(G_u) = L_1 \cup L_2$ as follows: If $w \in L_1 \cup L_2$, then $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} w$ or $S_2 \overset{*}{\underset{G_2}{\Rightarrow}} w$. Therefore,

$$S \underset{G_u}{\Rightarrow} S_1 \overset{*}{\underset{G_u}{\Rightarrow}} w \quad \text{or} \quad S \underset{G_u}{\Rightarrow} S_2 \overset{*}{\underset{G_u}{\Rightarrow}} w, \text{ i.e. } w \in L(G_u)$$

Thus, $L_1 \cup L_2 \subseteq L(G_u)$.

To prove that $L(G_u) \subseteq L_1 \cup L_2$, consider a derivation of w. The first step should be $S \Rightarrow S_1$ or $S \Rightarrow S_2$. If $S \Rightarrow S_1$ is the first step, in the subsequent steps S_1 is changed. As $V'_N \cap V''_N = \emptyset$, these steps should involve only the variables of V'_N and the productions we apply are in P_1. So $S \overset{*}{\underset{G_1}{\Rightarrow}} w$. Similarly, if the first step is $S \Rightarrow S_2$, then $S \underset{G_2}{\Rightarrow} S_2 \overset{*}{\underset{G_2}{\Rightarrow}} w$. Thus, $L(G_u) = L_1 \cup L_2$. Also, $L(G_u)$ is of type 0 or type 2 according as L_1 and L_2 are of type 0 or type 2. If Λ is not in $L_1 \cup L_2$, then $L(G_u)$ is of type 3 or type 1 according as L_1 and L_2 are of type 3 or type 1.

Suppose $\Lambda \in L_1$. In this case, define

$$G_u = (V'_N \cup V''_N \cup \{S, S'\}, \Sigma_1 \cup \Sigma_2, P_u, S')$$

where (i) S' is a new symbol, i.e. $S' \notin V'_N \cup V''_N \cup \{S\}$, and (ii) $P_u = P_1 \cup P_2 \cup \{S' \to S, S \to S_1, S \to S_2\}$. So, $L(G_u)$ is of type 1 or type 3 according as L_1 and L_2 are of type 1 or type 3. When $\Lambda \in L_2$, the proof is similar. ∎

Properties of Language(Operation on language)

Theorem 4.6 Each of the classes $\mathcal{L}_0$, $\mathcal{L}_{csl}$, $\mathcal{L}_{cfl}$, $\mathcal{L}_{rl}$ is closed under concatenation.

Proof Let L_1 and L_2 be two languages of type i. Then, as in Theorem 4.5, we get $G_1 = (V'_N, \Sigma_1, P_1, S_1)$ and $G_2 = (V''_N, \Sigma_2, P_2, S_2)$ of the same type i. We have to prove that L_1L_2 is of type i.

Construct a new grammar G_{con} as follows:

$$G_{con} = (V'_N \cup V''_N \cup \{S\}, \Sigma_1 \cup \Sigma_2, P_{con}, S)$$

where $S \notin V'_N \cup V''_N$.

$$P_{con} = P_1 \cup P_2 \cup \{S \rightarrow S_1S_2\}$$

We prove $L_1L_2 = L(G_{con})$. If $w = w_1w_2 \in L_1L_2$, then

$$S_1 \overset{*}{\underset{G_1}{\Rightarrow}} w_1, \qquad S_2 \overset{*}{\underset{G_2}{\Rightarrow}} w_2$$

So,

$$S \underset{G_{con}}{\Rightarrow} S_1S_2 \overset{*}{\underset{G_{con}}{\Rightarrow}} w_1w_2$$

Therefore,

$$L_1L_2 \subseteq L(G_{con})$$

Pumping Lemma for Context free Language

Theorem 6.10 (Pumping lemma for context-free languages). Let L be a context-free language. Then we can find a natural number n such that:

(i) Every $z \in L$ with $|z| \geq n$ can be written as $uvwxy$ for some strings u, v, w, x, y.

(ii) $|vx| \geq 1$.

(iii) $|vwx| \leq n$.

(iv) $uv^k wx^k y \in L$ for all $k \geq 0$.

Pumping Lemma for Context free Language

free language. ...
lemma we get a contradiction.

The procedure can be carried out by using the following steps:

Step 1 Assume L is context-free. Let n be the natural number obtained by using the pumping lemma.

Step 2 Choose $z \in L$ so that $|z| \geq n$. Write $z = uvwxy$ using the pumping lemma.

Step 3 Find a suitable k so that $uv^kwx^ky \notin L$. This is a contradiction, and so L is not context-free.

EXAMPLE 6.18

Show that $L = \{a^nb^nc^n \mid n \geq 1\}$ is not context-free but context-sensitive.

Decision Algorithm for Context Free Language

In this section we give some decision algorithms for context-free languages and regular sets.

(i) *Algorithm for deciding whether a context-free language L is empty.* We can apply the construction given in Theorem 6.3 for getting $V_N' = W_k$. L is nonempty if and only if $S \in W_k$.

(ii) *Algorithm for deciding whether a context-free language L is finite.* Construct a non-redundant context-free grammar G in CNF generating $L - \{\Lambda\}$. We draw a directed graph whose vertices are variables in G. If $A \rightarrow BC$ is a production, there are directed edges from A to B and A to C. L is finite if and only if the directed graph has no cycles.

Decision Algorithm for Context Free Language

(iii) *Algorithm for deciding whether a regular language L is empty.* Construct a deterministic finite automaton M accepting L. We construct the set of all states reachable from the initial state q_0. We find the states which are reachable from q_0 by applying a single input symbol. These states are arranged as a row under columns corresponding to every input symbol. The construction is repeated for every state appearing in an earlier row. The construction terminates in a finite number of steps. If a final state appears in this tabular column, then L is nonempty. (Actually, we can terminate the construction as soon as some final state is obtained in the tabular column.) Otherwise, L is empty.

(iv) *Algorithm for deciding whether a regular language L is infinite.* Construct a deterministic finite automaton M accepting L. L is infinite if and only if M has a cycle.

PDA

We now give a formal definition of a pushdown automaton.

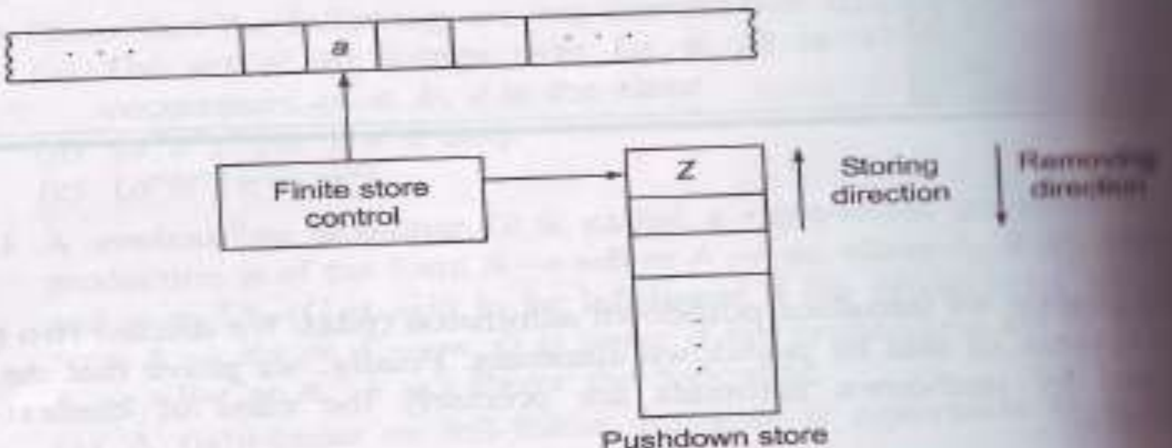


Fig. 7.1 Model of a pushdown automaton.

Definition 7.1 A pushdown automaton consists of

(i) a finite nonempty set of states denoted by Q,
(ii) a finite nonempty set of input symbols denoted by Σ,
(iii) a finite nonempty set of pushdown symbols denoted by Γ,
(iv) a special state called the initial state denoted by q_0,
(v) a special pushdown symbol called the *initial symbol* on the pushdown store denoted by Z_0,
(vi) a set of final states, a subset of Q denoted by F, and
(vii) a transition function δ from $Q \times (\Sigma \cup \{\Lambda\}) \times \Gamma$ to the set of finite subsets of $Q \times \Gamma^*$.

Symbolically, a pda is a 7-tuple, namely $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$.

Note: When $\delta(q, a, Z) = \emptyset$ for $(q, a, Z) \in Q \times (\Sigma \cup \{\Lambda\}) \times \Gamma$, we do not mention it.

PDA

EXAMPLE 7.1

Let

$$A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where

$$Q = \{q_0, q_1, q_f\}, \quad \Sigma = \{a, b\}, \quad \Gamma = \{a, Z_0\}, \quad F = \{q_f\}$$

PDA

and δ is given by

$$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}, \quad \delta(q_1, b, a) = \{(q_1, \Lambda)\}$$

$$\delta(q_0, a, a) = \{(q_0, aa)\}, \quad \delta(q_1, \Lambda, Z_0) = \{(q_1, \Lambda)\}$$

$$\delta(q_0, b, a) = \{(q_1, \Lambda)\}$$

Remarks 1. $\delta(q, a, Z)$ is a finite subset of $Q \times \Gamma^*$. The elements of $\delta(q, a, Z)$ are of the form (q', α), where $q' \in Q$, $\alpha \in \Gamma^*$. $\delta(q, a, Z)$ may be the empty set.

2. At any time the pda is in some state q and the PDS has some symbols from Γ. The pda reads an input symbol a and the topmost symbol Z in PDS. Using the transition function δ, the pda makes a transition to a state q' and writes a string α after removing Z. The elements in PDS which were below Z initially are not disturbed. Here (q', α) is one of the elements of the finite set $\delta(q, a, Z)$. When $\alpha = \Lambda$, the topmost symbol, Z, is erased.

3. The behaviour of a pda is nondeterministic as the transition is given by any element of $\delta(q, a, Z)$.

4. As δ is defined on $Q \times (\Sigma \cup \{\Lambda\}) \times \Gamma$, the pda may make transition without reading any input symbol (when $\delta(q, \Lambda, Z)$ is defined as a nonempty set for $q \in Q$ and $Z \in \Gamma$). Such transitions are called Λ-moves.

5. The pda cannot take a transition when PDS is empty (We can apply δ only when the pda reads an input symbol and the topmost pushdown symbol in PDS). In this case the pda halts.

6. When we write $\alpha = Z_1Z_2 \ldots Z_m$ in PDS, Z_1 is the topmost element, Z_2 is below Z_1, etc. and Z_m is below Z_{m-1}.

PDA

Definition 7.2 Let $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a pda. An instantaneous description (ID) is (q, x, α), where $q \in Q$, $x \in \Sigma^*$ and $\alpha \in \Gamma^*$.

For example, $(q, a_1a_2 \ldots a_n, Z_1Z_2 \ldots Z_m)$ is an ID. This describes the pda when the current state is q, the input string to be processed is $a_1a_2 \ldots a_n$. The pda will process $a_1a_2 \ldots a_n$ in that order. The PDS has $Z_1, Z_2, \ldots, Z_m$ with Z_1 at the top, Z_2 is the second element from the top, etc. and Z_m is the lowest element in PDS.

Definition 7.3 An initial ID is (q_0, x, Z_0). This means that initially the pda is in the initial state q_0, the input string to be processed is x, and the PDS has only one symbol, namely Z_0.

PDA

ACCEPTANCE BY pda

A pda has final states like a nondeterministic finite automaton and has also the additional structure, namely PDS. So we can define acceptance of input strings by pda in terms of final states or in terms of PDS.

Definition 7.6 Let $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a pda. The set accepted by pda by final state is defined by

$$T(A) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \overset{*}{\vdash} (q_f, \Lambda, \alpha) \text{ for some } q_f \in F \text{ and } \alpha \in \Gamma^*\}$$

The next definition describes the second type of acceptance.

Definition 7.7 Let $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a pda. The set $N(A)$ accepted by null store (or empty store) is defined by

$$N(A) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \overset{*}{\vdash} (q, \Lambda, \Lambda) \text{ for some } q \in Q\}$$

In other words, w is in $N(A)$ if A is in initial ID (q_0, w, Z_0) and empties the PDS after processing all the symbols of w. So in defining $N(A)$, we consider the change brought about on PDS by application of w, and not the transition of states.

PDA and CFG

.3 PUSHDOWN AUTOMATA AND CONTEXT-FREE LANGUAGES

n this section we prove that the sets accepted by pda (by null store or final tate) are precisely the context-free languages.

Theorem 7.3 If L is a context-free language, then we can construct a pda A accepting L by empty store, i.e. $L = N(A)$.

PDA and CFG

Proof We construct A by making use of productions in G.

Step 1 (Construction of A) Let $L = L(G)$, where $G = (V_N, \Sigma, P, S)$ is a context-free grammar. We construct a pda A as

$$A = (\{q\}, \Sigma, V_N \cup \Sigma, \delta, q, S, \emptyset)$$

where δ is defined by the following rules:

R_1: $\delta(q, \Lambda, A) = \{(q, \alpha) \mid A \to \alpha \text{ is in } P\}$

R_2: $\delta(q, a, a) = \{(q, \Lambda)\}$ for every a in Σ

Example

EXAMPLE 7.7

Construct a pda A equivalent to the following context-free grammar: $S \rightarrow 0BB$, $B \rightarrow 0S \mid 1S \mid 0$. Test whether 010^4 is in $N(A)$.

Solution

Define pda A as follows:

$$A = (\{q\}, \{0, 1\}, \{S, B, 0, 1\}, \delta, q, S, \emptyset)$$

δ is defined by the following rules:

R_1: $\delta(q, \Lambda, S) = \{(q, 0BB)\}$

R_2: $\delta(q, \Lambda, B) = \{(q, 0S), (q, 0S), (q, 0)\}$

R_3: $\delta(q, 0, 0) = \{(q, \Lambda)\}$

R_4: $\delta(q, 1, 1) = \{(q, \Lambda)\}$

Cont.

$(q, 010^4, S)$

$\vdash (q, 010^4, 0BB)$ by Rule R_1

$\vdash (q, 10^4, BB)$ by Rule R_3

$\vdash (q, 10^4, 1SB)$ by Rule R_2 since $(q, 1S) \in \alpha(q, \Lambda, B)$

$\vdash (q, 0^4, SB)$ by Rule R_4

$\vdash (q, 0^4, 0BBB)$ by Rule R_1

$\vdash (q, 0^3, BBB)$ by Rule R_3

$\overset{*}{\vdash} (q, 0^3, 000)$ by Rule R_2 since $(q, 0) \in \alpha(q, \Lambda, B)$

$\overset{*}{\vdash} (q, \Lambda, \Lambda)$ by Rule R_3

Thus, $010^4 \subseteq N(A)$

PDA to CFG

Theorem 7.4 If $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a pda, then there exists a context-free grammar G such that $L(G) = N(A)$.

Proof We first give the construction of G and then prove that $N(A) = L(G)$.

Step 1 (Construction of G). We define $G = (V_N, \Sigma, P, S)$, where

$$V_N = \{S\} \cup \{[q, Z, q'] \mid q, q' \in Q, Z \in \Gamma\}$$

i.e. any element of V_N is either the new symbol S acting as the start symbol for G or an ordered triple whose first and third elements are states and the second element is a pushdown symbol.

The productions in P are induced by moves of pda as follows:

R_1: S-productions are given by $S \rightarrow [q_0, Z_0, q]$ for every q in Q.

R_2: Each move erasing a pushdown symbol given by $(q', \Lambda) \in \delta(q, a, Z)$ induces the production $[q, Z, q'] \rightarrow a$.

R_3: Each move not erasing a pushdown symbol given by $(q_1, Z_1Z_2 \ldots Z_m) \in \delta(q, a, Z)$ induces many productions of the form

$$[q, Z, q'] \rightarrow a[q_1, Z_1, q_2][q_2, Z_2, q_3] \ldots [q_m, Z_m, q']$$

where each of the states $q', q_2, \ldots, q_m$ can be any state in Q. Each move yields many productions because of R_3. We apply this construction to an example before proving that $L(G) = N(A)$.

Example

EXAMPLE 7.8

Construct a context-free grammar G which accepts $N(A)$, where

$$A = (\{q_0, q_1\}, \{a, b\}, \{Z_0, Z\}, \delta, q_0, Z_0, \emptyset)$$

and δ is given by

$$\delta(q_0, b, Z_0) = \{(q_0, ZZ_0)\}$$
$$\delta(q_0, \Lambda, Z_0) = \{(q_0, \Lambda)\}$$
$$\delta(q_0, b, Z) = \{(q_0, ZZ)\}$$
$$\delta(q_0, a, Z) = \{(q_1, Z)\}$$
$$\delta(q_1, b, Z) = \{(q_1, \Lambda)\}$$
$$\delta(q_1, a, Z_0) = \{(q_0, Z_0)\}$$

Solution

Let

$$G = (V_N, \{a, b\}, P, S)$$

Cont.

Here V_N consists of S, $[q_0, Z_0, q_0]$, $[q_0, Z_0, q_1]$, $[q_0, Z, q_0]$, $[q_0, Z, q_1]$, $[q_1, Z_0, q_0]$, $[q_1, Z_0, q_1]$, $[q_1, Z, q_0]$, $[q_1, Z, q_1]$.

The productions are

$$P_1: S \rightarrow [q_0, Z_0, q_0]$$

$$P_2: S \rightarrow [q_0, Z_0, q_1]$$

$\delta(q_0, b, Z_0) = \{(q_0, ZZ_0)\}$ yields

$$P_3: [q_0, Z_0, q_0] \rightarrow b[q_0, Z, q_0][q_0, Z_0, q_0]$$

$$P_4: [q_0, Z_0, q_0] \rightarrow b[q_0, Z, q_1][q_1, Z_0, q_0]$$

$$P_5: [q_0, Z_0, q_1] \rightarrow b[q_0, Z, q_0][q_0, Z_0, q_1]$$

$$P_6: [q_0, Z_0, q_1] \rightarrow b[q_0, Z, q_1][q_1, Z_0, q_1]$$

$\delta(q_0, \Lambda, Z_0) = \{(q_0, \Lambda)\}$ gives

$$P_7: [q_0, Z_0, q_0] \rightarrow \Lambda$$

Cont.

$\delta(q_0, b, Z) = \{(q_0, ZZ)\}$ gives

$$P_8: [q_0, Z, q_0] \rightarrow b[q_0, Z, q_0][q_0, Z, q_0]$$

$$P_9: [q_0, Z, q_0] \rightarrow b[q_0, Z, q_1][q_1, Z, q_0]$$

$$P_{10}: [q_0, Z, q_1] \rightarrow b[q_0, Z, q_0][q_0, Z, q_1]$$

$$P_{11}: [q_0, Z, q_1] \rightarrow b[q_0, Z, q_1][q_1, Z, q_1]$$

$\delta(q_0, a, Z) = \{(q_1, Z)\}$ yields

$$P_{12}: [q_0, Z, q_0] \rightarrow a[q_1, Z, q_0]$$

$$P_{13}: [q_0, Z, q_1] \rightarrow a[q_1, Z, q_1]$$

$\delta(q_1, b, Z) = \{(q_1, \Lambda)\}$ gives

$$P_{14}: [q_1, Z, q_1] \rightarrow b$$

$\delta(q_1, a, Z_0) = \{(q_0, Z_0)\}$ gives

$$P_{15}: [q_1, Z_0, q_0] \rightarrow a[q_0, Z_0, q_0]$$

$$P_{16}: [q_1, Z_0, q_1] \rightarrow a[q_0, Z_0, q_1]$$

P_1–P_{16} give the productions in P.

Example

EXAMPLE 7.9

Construct a pda accepting $\{a^n b^m a^n \mid m, n \geq 1\}$ by null store. Construct the corresponding context-free grammar accepting the same set.

Solution

The pda A accepting $\{a^n b^m a^n \mid m, n \geq 1\}$ is defined as follows:

$$A = (\{q_0, q_1\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \emptyset)$$

where δ is defined by

R_1: $\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$

R_2: $\delta(q_0, a, a) = \{(q_0, aa)\}$

R_3: $\delta(q_0, b, a) = \{(q_1, a)\}$

R_4: $\delta(q_1, b, a) = \{(q_1, a)\}$

R_5: $\delta(q_1, a, a) = \{(q_1, \Lambda)\}$

R_6: $\delta(q_1, \Lambda, Z_0) = \{(q_1, \Lambda)\}$

This is a modification of δ given in Example 7.2.

We start storing a's until a b occurs (Rules R_1 and R_2). When the current input symbol is b, the state changes, but no change in PDS occurs (Rule R_3). Once all the b's in the input string are exhausted (using Rule R_4), the remaining a's are erased (Rule R_5). Using R_6, Z_0 is erased. So,

$$(q_0, a^n b^m a^n, Z_0) \overset{*}{\vdash} (q_1, \Lambda, Z_0) \vdash (q_1, \Lambda, \Lambda)$$

This means that $a^n b^m a^n \in N(A)$. We can show that

$$N(A) = \{a^n b^m a^n \mid m, n \geq 1\}$$

Cont.

Define $G = (V_N, \{a, b\}, P, S)$, where V_N consists of

$$[q_0, Z_0, q_0], [q_1, Z_0, q_0], [q_0, a, q_0], [q_1, a, q_0]$$

$$[q_0, Z_0, q_1], [q_1, Z_0, q_1], [q_0, a, q_1], [q_1, a, q_1]$$

The productions in P are constructed as follows:

The S-productions are

$$P_1: S \rightarrow [q_0, Z_0, q_0], \qquad P_2: S \rightarrow [q_0, Z_0, q_1]$$

$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$ induces

$$P_3: [q_0, Z_0, q_0] \rightarrow a[q_0, a, q_0][q_0, Z_0, q_0]$$

$$P_4: [q_0, Z_0, q_0] \rightarrow a[q_0, a, q_1][q_1, Z_0, q_0]$$

$$P_5: [q_0, Z_0, q_1] \rightarrow a[q_0, a, q_0][q_0, Z_0, q_1]$$

$$P_6: [q_0, Z_0, q_1] \rightarrow a[q_0, a, q_1][q_1, Z_0, q_1]$$

Cont.

$\delta(q_0, a, a) = \{(q_0, aa)\}$ yields

$$P_7: [q_0, a, q_0] \rightarrow a[q_0, a, q_0][q_0, a, q_0]$$

$$P_8: [q_0, a, q_0] \rightarrow a[q_0, a, q_1][q_1, a, q_0]$$

$$P_9: [q_0, a, q_1] \rightarrow a[q_0, a, q_0][q_0, a, q_1]$$

$$P_{10}: [q_0, a, q_1] \rightarrow a[q_0, a, q_1][q_1, a, q_1]$$

$\delta(q_0, b, a) = \{(q_1, a)\}$ gives

$$P_{11}: [q_0, a, q_0] \rightarrow b[q_1, a, q_0]$$

$$P_{12}: [q_0, a, q_1] \rightarrow b[q_1, a, q_1]$$

$\delta(q_1, b, a) = \{(q_1, a)\}$ yields

$$P_{13}: [q_1, a, q_0] \rightarrow b[q_1, a, q_0]$$

$$P_{14}: [q_1, a, q_1] \rightarrow b[q_1, a, q_1]$$

$\delta(q_1, a, a) = \{(q_1, \Lambda)\}$ gives

$$P_{15}: [q_1, a, q_1] \rightarrow a$$

$\delta(q_1, \Lambda, Z_0) = \{(q_1, \Lambda)\}$ yields

$$P_{16}: [q_1, Z_0, q_1] \rightarrow \Lambda$$

Exercise

SELF-TEST

Choose the correct answer to Questions 1–6.

1. If $\delta(q, a_1, Z_1)$ contains (q', β), then
(a) $(q, a_1a_2, Z_1Z_2) \vdash (q', a_2, \beta Z_2)$
(b) $(q, a_2a_2, Z_1Z_2) \vdash (q', a_1a_2, \beta Z_2)$
(c) $(q, a_1a_2, Z_2) \vdash (q', a_1, Z_1)$
(d) $(q, a_1a_2, Z_1Z_2) \vdash (q', a_2, Z_1Z_2)$

2. In a deterministic pda, $|\delta(q, a, Z)|$ is
(a) equal to 1
(b) less than or equal to 1
(c) greater than 1
(d) greater than or equal to 1

3. In a deterministic pda:
(a) $\delta(q, a, Z) = \emptyset \Rightarrow \delta(q, \Lambda, Z) \neq \emptyset$
(b) $\delta(q, a, Z) \neq \emptyset \Rightarrow \delta(q, \Lambda, Z) = \emptyset$
(c) $\delta(q, \Lambda, Z) \neq \emptyset \Rightarrow \delta(q, a, Z) \neq \emptyset$
(d) $\delta(q, \Lambda, Z) \neq \emptyset \Rightarrow \delta(q, a, Z) = \emptyset$

4. $\{a^nb^n \mid n \geq 1\}$ is accepted by a pda
(a) by null store and also by final state.
(b) by null store but not by final state.
(c) by final state but not by null store.
(d) by none of these.

5. $\{a^nb^{2n} \mid n \geq 1\}$ is accepted by
(a) a finite automaton
(b) a nondeterministic finite automaton
(c) a pda
(d) none of these.

Cont.

7.3 Construct a pda accepting by empty store each of the following languages.

(a) $\{a^n b^m a^n \mid m, n \geq 1\}$

(b) $\{a^n b^{2n} \mid n \geq 1\}$

(c) $\{a^n b^m c^n \mid m, n \geq 1\}$

(d) $\{a^m b^n \mid m > n \geq 1\}$

7.4 Construct a pda accepting by final state each of the languages given in Exercise 7.3.

7.5 Construct a context-free grammar generating each of the following languages, and hence a pda accepting each of them by empty store.

(a) $\{a^n b^n \mid n \geq 1\} \cup \{a^m b^{2m} \mid m \geq 1\}$

(b) $\{a^n b^m a^n \mid m, n \geq 1\} \cup \{a^n c^n \mid n \geq 1\}$

(c) $\{a^n b^m c^m d^n \mid m, n \geq 1\}$

7.6 Let $L = \{a^n b^m \mid n < m\}$. Construct (i) a context-free grammar accepting L, (ii) a pda accepting L by empty store, and (iii) a pda accepting L by final state.

Cont.

solving Exercise 7.4.

12 Show that $\{a^n b^n \mid n \geq 1\} \cup \{a^m b^{2m} \mid m \geq 1\}$ cannot be accepted by a deterministic pda.