Discrete mathematics course
Chapter 5: Modeling Computation

Anh Tuan GIANG

ICTLab, ICT Department
University of Science and Technology of Hanoi-USTH

May 5, 2017

Formal language

Formal language is:

- A set of natural language (i.e English, Vietnamese).

Formal language

Formal language is:

- A set of natural language (i.e English, Vietnamese).
- Specified by a well-defined set of rules of syntax.

Formal language

Formal language is:

- A set of natural language (i.e English, Vietnamese).
- Specified by a well-defined set of rules of syntax.
- Sentences can be described by using grammar

Formal language

Formal language is:

- A set of natural language (i.e English, Vietnamese).
- Specified by a well-defined set of rules of syntax.
- Sentences can be described by using grammar

Example: The rabbit eats quickly.

Phrase-structure grammars

Definition 1.1.

A vocabulary (or alphabet) V is a finite, nonempty set of elements called symbols. A word (or sentence) over V is a string of finite length of elements of V. The empty string or null string, denoted by λ, is the string containing no symbols. The set of all words over V is denoted by V^*. A language over V is a subset of V^*.

Phrase-structure grammars

Definition 1.2.

A phrase-structure grammar $G = (V, T, S, P)$ consists of a vocabulary V, a subset T of V consisting of terminal symbols, a start symbol S from V, and a finite set of productions P. The set $V - T$ is denoted by N. Elements of N are called nonterminal symbols. Every production in P must contain at least one nonterminal on its left side.

Phrase-structure grammars

Definition 1.2.

A phrase-structure grammar $G = (V, T, S, P)$ consists of a vocabulary V, a subset T of V consisting of terminal symbols, a start symbol S from V, and a finite set of productions P. The set $V - T$ is denoted by N. Elements of N are called nonterminal symbols. Every production in P must contain at least one nonterminal on its left side.

Example: let $G = (V, T, S, P)$, where $V = \{a, b, A, B, S\}$, $T = \{a, b\}$, S is the start symbol, and $P = \{S \to ABa, A \to BB, B \to ab, AB \to b\}$. G is an example of a phrase-structure grammar.

Phrase-structure grammars

Definition 1.3.

Let $G = (V, T, S, P)$ be a phrase-structure grammar. Let $w_0 = lz_0r$ (that is, the concatenation of l, z_0, and r) and $w_1 = lz_1r$ be strings over V. If $z_0 \to z_1$ is a production of G, we say that w_1 is directly derivable from w_0 and we write $w_0 \Rightarrow w_1$. If $w_0, w_1, ..., w_n$ are strings over V such that $w_0 \Rightarrow w_1, w_1 \Rightarrow w_2, ..., w_{n1} \Rightarrow w_n$,then we say that w_n is derivable from w_0, and we write $w_0 \overset{*}{\Rightarrow} w_n$. The sequence of steps used to obtain w_n from w_0 is called a derivation.

Phrase-structure grammars

Definition 1.3.

Let $G = (V, T, S, P)$ be a phrase-structure grammar. Let $w_0 = lz_0r$ (that is, the concatenation of l, z_0, and r) and $w_1 = lz_1r$ be strings over V. If $z_0 \to z_1$ is a production of G, we say that w_1 is directly derivable from w_0 and we write $w_0 \Rightarrow w_1$. If $w_0, w_1, ..., w_n$ are strings over V such that $w_0 \Rightarrow w_1, w_1 \Rightarrow w_2, ..., w_{n1} \Rightarrow w_n$,then we say that w_n is derivable from w_0, and we write $w_0 \overset{*}{\Rightarrow} w_n$. The sequence of steps used to obtain w_n from w_0 is called a derivation.

Example: the string $Aaba$ is directly derivable from ABa in the grammar in the previous example, because $B \to ab$ is a production in the grammar. The string $abababa$ is derivable from ABa because $ABa \Rightarrow Aaba \Rightarrow BBaba \Rightarrow Bababa \Rightarrow abababa$, using the productions $B \to ab, A \to BB, B \to ab, andB \to ab$ in succession.

Phrase-structure grammars

Definition 1.4.

Let $G = (V, T, S, P)$ be a phrase-structure grammar. The language generated by G (or the language of G), denoted by $L(G)$, is the set of all strings of terminals that are derivable from the starting state S. In other words,

$$L(G) = \{w \in T^* | S \overset{*}{\Rightarrow} w\}.$$

Phrase-structure grammars

Definition 1.4.

Let $G = (V, T, S, P)$ be a phrase-structure grammar. The language generated by G (or the language of G), denoted by $L(G)$, is the set of all strings of terminals that are derivable from the starting state S. In other words,

$$L(G) = \{w \in T^* | S \overset{*}{\Rightarrow} w\}.$$

Example: let G be the grammar with vocabulary $V = S, A, a, b$, set of terminals $T = \{a, b\}$, starting symbol S, and productions $P = \{S \to aA, S \to b, A \to aa\}$. What is $L(G)$?

Phrase-structure grammars

Definition 1.4.

Let $G = (V, T, S, P)$ be a phrase-structure grammar. The language generated by G (or the language of G), denoted by $L(G)$, is the set of all strings of terminals that are derivable from the starting state S. In other words,

$$L(G) = \{w \in T^* | S \overset{*}{\Rightarrow} w\}.$$

Example: let G be the grammar with vocabulary $V = S, A, a, b$, set of terminals $T = \{a, b\}$, starting symbol S, and productions $P = \{S \to aA, S \to b, A \to aa\}$. What is $L(G)$?

Example: let G be the grammar with vocabulary $V = \{S, 0, 1\}$, set of terminals $T = \{0, 1\}$, starting symbol S, and productions $P = \{S \to 11S, S \to 0\}$. What is $L(G)$?

Exercises

- Exercise: give a phrase-structure grammar that generates the set $\{0^n 1^n | n = 0, 1, 2, ...\}$.

Exercises

- Exercise: give a phrase-structure grammar that generates the set $\{0^n1^n|n = 0, 1, 2, ...\}$.
- Exercise: find a phrase-structure grammar to generate the set $\{0^m1^n|m \quad and \quad n \quad are \quad nonnegative \quad integers\}$.

Types of Phrase-structure grammars

- A **type 0** grammar has no restrictions on its productions

Types of Phrase-structure grammars

- A **type 0** grammar has no restrictions on its productions
- A **type 1** grammar can have productions of the form $w_1 \rightarrow w_2$, where $w_1 = lAr$ and $w_2 = lwr$, where A is a nonterminal symbol,l and r are strings of zero or more terminal or nonterminal symbols, and w is a nonempty string of terminal or nonterminal symbols.
(context-sensitive language)

Types of Phrase-structure grammars

- A **type 0** grammar has no restrictions on its productions
- A **type 1** grammar can have productions of the form $w_1 \rightarrow w_2$, where $w_1 = lAr$ and $w_2 = lwr$, where A is a nonterminal symbol,l and r are strings of zero or more terminal or nonterminal symbols, and w is a nonempty string of terminal or nonterminal symbols.
(context-sensitive language)
- A **type 2** grammar can have productions only of the form $w_1 \rightarrow w_2$, where w_1 is a single symbol that is not a terminal symbol.
(context-free grammars)

Types of Phrase-structure grammars

- A **type 0** grammar has no restrictions on its productions
- A **type 1** grammar can have productions of the form $w_1 \to w_2$, where $w_1 = lAr$ and $w_2 = lwr$, where A is a nonterminal symbol,l and r are strings of zero or more terminal or nonterminal symbols, and w is a nonempty string of terminal or nonterminal symbols. (context-sensitive language)
- A **type 2** grammar can have productions only of the form $w_1 \to w_2$, where w_1 is a single symbol that is not a terminal symbol. (context-free grammars)
- A **type 3** grammar can have productions only of the form $w_1 \to w_2$ with $w_1 = A$ and either $w_2 = aB$ or $w_2 = a$,where A and B are nonterminal symbols and a is a terminal symbol, or with $w_1 = S$ and $w_2 = \lambda$. (regular grammars)

Derivation trees

- A derivation in the language generated by a context-free grammar can be represented graphically using an ordered rooted tree, called a derivation, or parse tree.

Derivation trees

- A derivation in the language generated by a context-free grammar can be represented graphically using an ordered rooted tree, called a derivation, or parse tree.
- The root of this tree represents the starting symbol.

Derivation trees

- A derivation in the language generated by a context-free grammar can be represented graphically using an ordered rooted tree, called a derivation, or parse tree.
- The root of this tree represents the starting symbol.
- The internal vertices of the tree represent the nonterminal symbols that arise in the derivation.

Derivation trees

- A derivation in the language generated by a context-free grammar can be represented graphically using an ordered rooted tree, called a derivation, or parse tree.
- The root of this tree represents the starting symbol.
- The internal vertices of the tree represent the nonterminal symbols that arise in the derivation.
- The leaves of the tree represent the terminal symbols that arise.

Derivation trees

- Example: Construct a derivation tree for the derivation of *the hungry rabbit eats quickly*.

Derivation trees

- Example: Construct a derivation tree for the derivation of *the hungry rabbit eats quickly*.

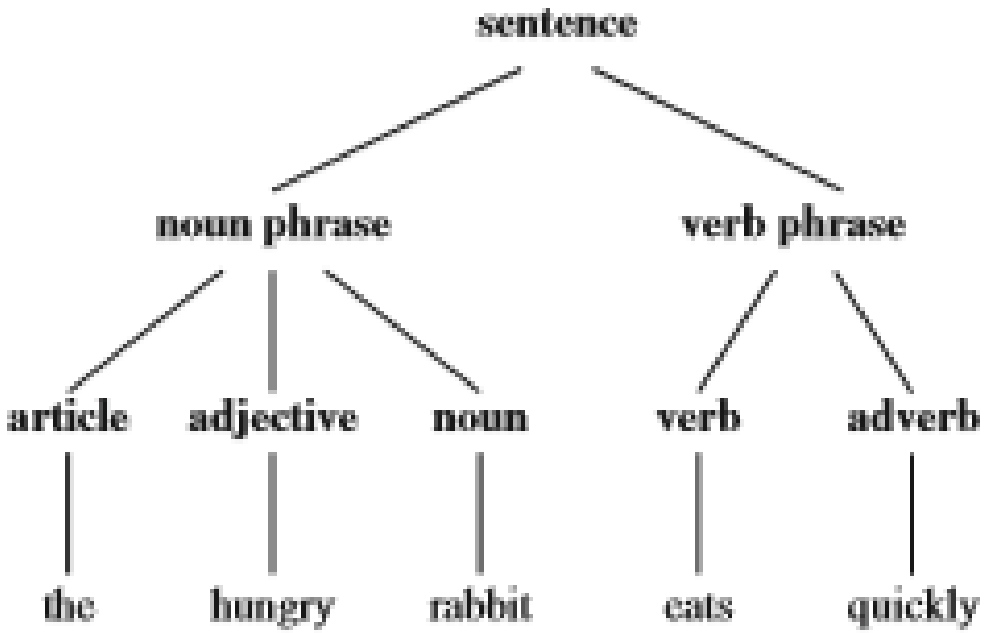


FIGURE 1 **A Derivation Tree.**

Backus-Naur form

- A type 2 grammar can be referred to as the BackusNaur form (BNF).

Backus-Naur form

- A type 2 grammar can be referred to as the BackusNaur form (BNF).
- Use in the specification of the programming language ALGOL.

Backus-Naur form

- A type 2 grammar can be referred to as the BackusNaur form (BNF).
- Use in the specification of the programming language ALGOL.
- The BackusNaur form is used to specify the syntactic rules of many computer languages, including Java.

Definitions

Definition 2.1.

A finite-state machine $M = (S, I, O, f, g, s_0)$ consists of a finite set S of states, a finite input alphabet I, a finite output alphabet O, a transition function f that assigns to each state and input pair a new state, an output function g that assigns to each state and input pair an output, and an initial state s_0.

Examples

- Construct the state diagram for the finite-state machine with the state table shown in Table on the left.

Examples

- Construct the state diagram for the finite-state machine with the state table shown in Table on the left.

TABLE 2

	f		g	
	Input		*Input*	
State	**0**	**1**	**0**	**1**
s_0	s_1	s_0	1	0
s_1	s_3	s_0	1	1
s_2	s_1	s_2	0	1
s_3	s_2	s_1	0	0

Examples

- Construct the state diagram for the finite-state machine with the state table shown in Table on the left.

TABLE 2

State	f Input 0	f Input 1	g Input 0	g Input 1
s_0	s_1	s_0	1	0
s_1	s_3	s_0	1	1
s_2	s_1	s_2	0	1
s_3	s_2	s_1	0	0

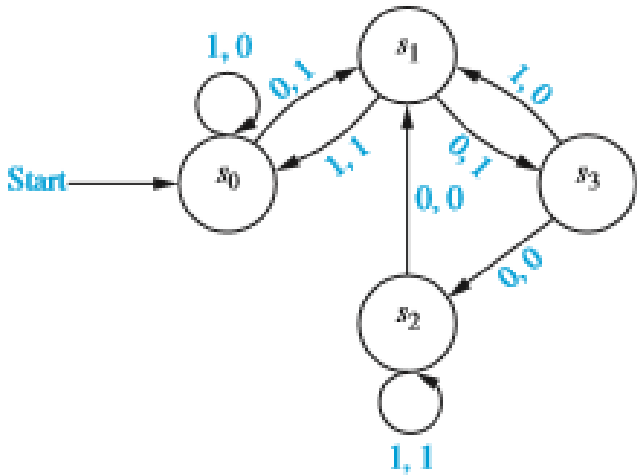


FIGURE 2 The State Diagram for the Finite-State Machine Shown in Table 2.

Examples

- Construct the state table for the finite-state machine with the state diagram shown in Figure on the left.

Examples

- Construct the state table for the finite-state machine with the state diagram shown in Figure on the left.

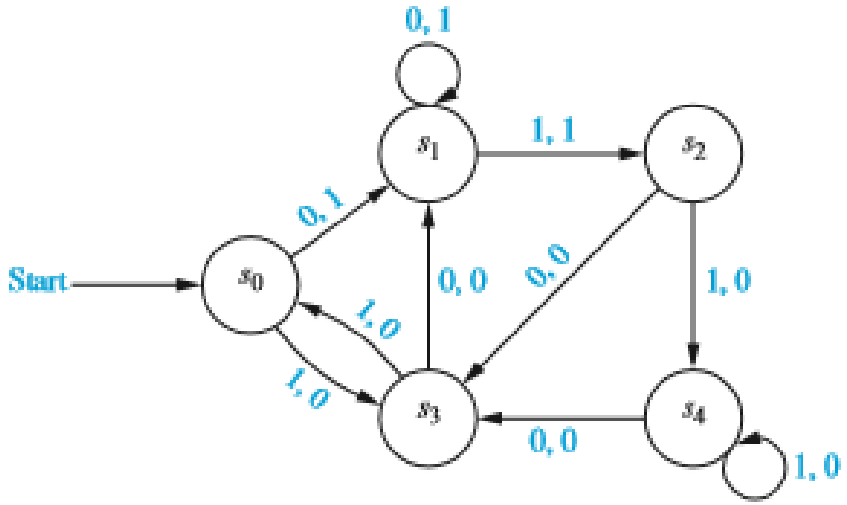


FIGURE 3 **A Finite-State Machine.**

Examples

- Construct the state table for the finite-state machine with the state diagram shown in Figure on the left.

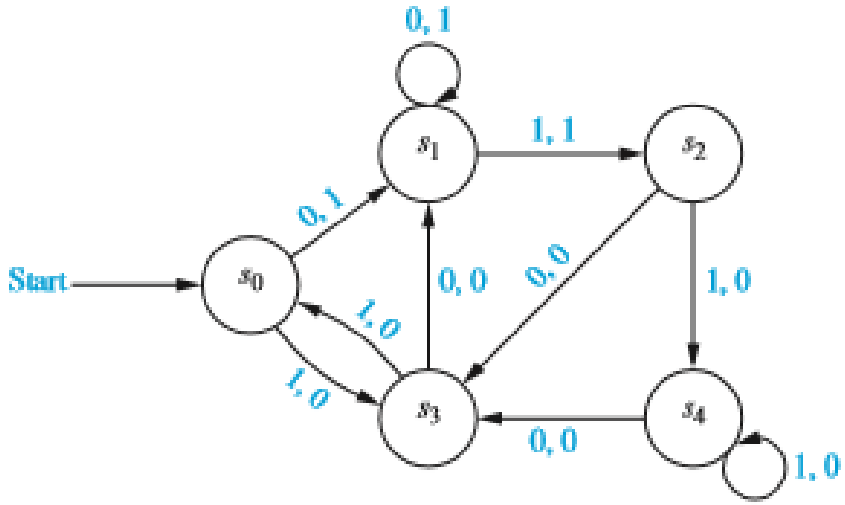


FIGURE 3 A Finite-State Machine.

TABLE 3

State	f Input 0	f Input 1	g Input 0	g Input 1
s_0	s_1	s_3	1	0
s_1	s_1	s_2	1	1
s_2	s_3	s_4	0	0
s_3	s_1	s_0	0	0
s_4	s_3	s_4	0	0

Examples

- Find the output string generated by the finite-state machine in previous Figure if the input string is 101011.

Examples

- Find the output string generated by the finite-state machine in previous Figure if the input string is 101011.

TABLE 4

Input	1	0	1	0	1	1	—
State	s_0	s_3	s_1	s_2	s_3	s_0	s_3
Output	0	0	1	0	0	0	—

Examples

- In a certain coding scheme, when three consecutive 1s appear in a message, the receiver of the message knows that there has been a transmission error. Construct a finite-state machine that gives a 1 as its current output bit if and only if the last three bits received are all 1s.

Examples

- In a certain coding scheme, when three consecutive 1s appear in a message, the receiver of the message knows that there has been a transmission error. Construct a finite-state machine that gives a 1 as its current output bit if and only if the last three bits received are all 1s.

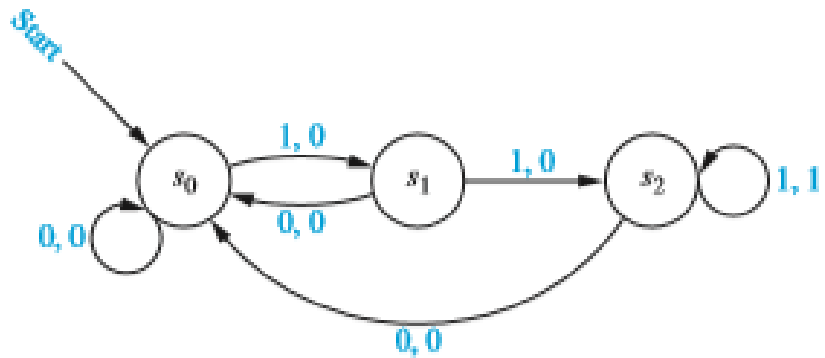


FIGURE 6 **A Finite-State Machine That Gives an Output of 1 If and Only If the Input String Read So Far Ends with 111.**

Definitions

Definition 2.2.

Let $M = (S, I, O, f, g, s_0)$ be a finite-state machine and $L \subseteq I^*$. We say that M recognizes (or accepts) L if an input string x belongs to L if and only if the last output bit produced by M when given x as input is a 1.

Set of Strings

Definition 3.1.

Suppose that A and B are subsets of V^*, where V is a vocabulary. The *concatenation* of A and B, denoted by AB, is the set of all strings of the form xy, where x is a string in A and y is a string in B.

Set of Strings

Definition 3.1.

Suppose that A and B are subsets of V^*, where V is a vocabulary. The *concatenation* of A and B, denoted by AB, is the set of all strings of the form xy, where x is a string in A and y is a string in B.

- Example: let $A = \{0, 11\}$ and $B = \{1, 10, 110\}$. Find AB and BA.

Set of Strings

Definition 3.1.

Suppose that A and B are subsets of V^*, where V is a vocabulary. The *concatenation* of A and B, denoted by AB, is the set of all strings of the form xy, where x is a string in A and y is a string in B.

- Example: let $A = \{0, 11\}$ and $B = \{1, 10, 110\}$. Find AB and BA.
- Example: let $A = \{1, 00\}$. Find A^n for n=0,1,2, and 3.

Set of Strings

Definition 3.2.

Suppose that A is a subset of V^*. Then the *Kleene closure* of A, denoted by A^*, is the set consisting of concatenations of arbitrarily many strings from A. That is, $A^* = \cup_{k=0}^{\infty} A^k$

Set of Strings

Definition 3.2.

Suppose that A is a subset of V^*. Then the *Kleene closure* of A, denoted by A^*, is the set consisting of concatenations of arbitrarily many strings from A. That is, $A^* = \cup_{k=0}^{\infty} A^k$

- Example: what are the Kleene closures of the sets $A = \{0\}, B = \{0, 1\}$, and $C = \{11\}$?

Finite-State Automata

Definition 3.3.

A finite-state automaton $M = (S, I, f, s_0, F)$ consists of a finite set S of states, a finite input alphabet I, a transition function f that assigns a next state to every pair of state and input (so that $f : S \times I \to S$), an initial or start state s_0, and a subset F of S consisting of final (or accepting states).

Finite-State Automata

- Construct the state diagram for the finite-state automaton $M = (S, I, f, s_0, F)$, where $S = \{s_0, s_1, s_2, s_3\}$, $I = \{0, 1\}$, $F = \{s_0, s_3\}$, and the transition function f is given in the Table below.

Finite-State Automata

- Construct the state diagram for the finite-state automaton $M = (S, I, f, s_0, F)$, where $S = \{s_0, s_1, s_2, s_3\}$, $I = \{0, 1\}$, $F = \{s_0, s_3\}$, and the transition function f is given in the Table below.

TABLE 1

	f	
	Input	
State	**0**	**1**
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_0	s_0
s_3	s_2	s_1

Finite-State Automata

- Construct the state diagram for the finite-state automaton $M = (S, I, f, s_0, F)$, where $S = \{s_0, s_1, s_2, s_3\}$, $I = \{0, 1\}$, $F = \{s_0, s_3\}$, and the transition function f is given in the Table below.

TABLE 1

	f	
	Input	
State	**0**	**1**
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_0	s_0
s_3	s_2	s_1

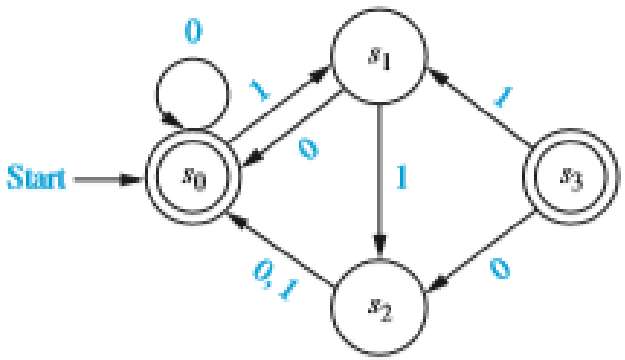


FIGURE 1 The State Diagram for a Finite-State Automaton.

Language Recognition by Finite-State Machines

Definition 3.4.

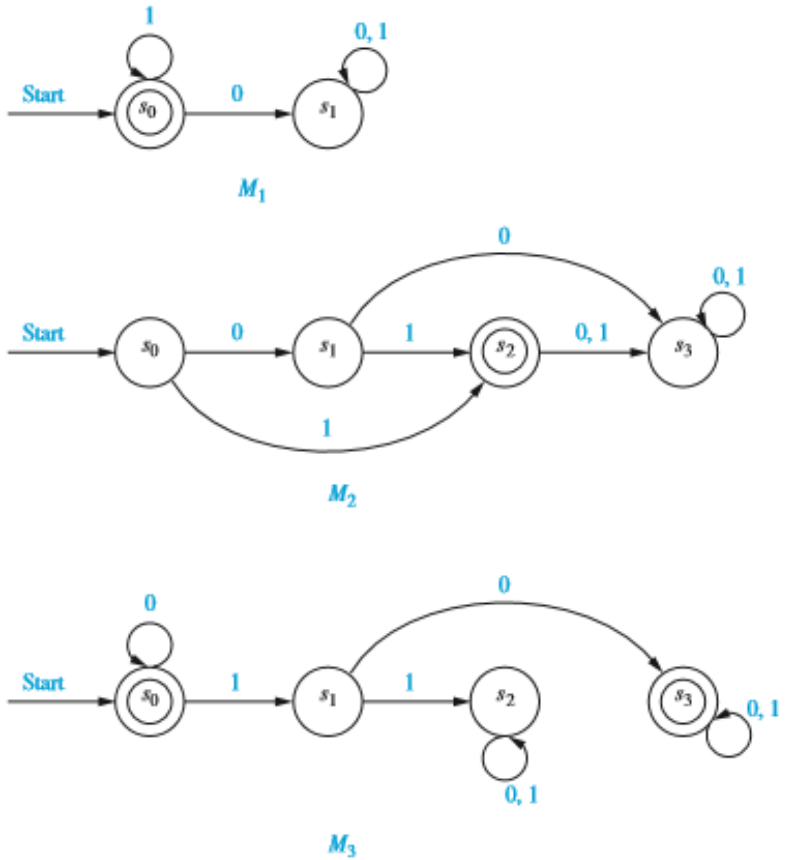
A string x is said to be recognized or accepted by the machine $M = (S, I, f, s_0, F)$ if it takes the initial state s_0 to a final state, that is, $f(s_0, x)$ is a state in F. The language recognized or accepted by the machine M, denoted by $L(M)$, is the set of all strings that are recognized by M. Two finite-state automata are called equivalent if they recognize the same language.

Language Recognition by Finite-State Machines

- Determine the languages recognized by the finite-state automata M_1, M_2, and M_3 in Figure below.

Language Recognition by Finite-State Machines

- Determine the languages recognized by the finite-state automata M_1, M_2, and M_3 in Figure below.



Non-deterministic Finite-State Automata

Definition 3.5.

A non-deterministic finite-state automaton $M = (S, I, f, s_0, F)$ consists of a set S of states, an input alphabet I, a transition function f that assigns a set of states to each pair of state and input (so that $f : S \times I \to P(S)$), a starting state s_0, and a subset F of S consisting of the final states.

Nondeterministic Finite-State Automata

- Find the state table for the non-deterministic finite-state automaton with the state diagram shown in Figure below.

Nondeterministic Finite-State Automata

- Find the state table for the non-deterministic finite-state automaton with the state diagram shown in Figure below.

TABLE 3

	f	
	Input	
State	**0**	**1**
s_0	s_0, s_2	s_1
s_1	s_3	s_4
s_2		s_4
s_3	s_3	
s_4	s_3	s_3

Nondeterministic Finite-State Automata

- Find the state table for the non-deterministic finite-state automaton with the state diagram shown in Figure below.

TABLE 3

State	f Input 0	f Input 1
s_0	s_0, s_2	s_1
s_1	s_3	s_4
s_2		s_4
s_3	s_3	
s_4	s_3	s_3

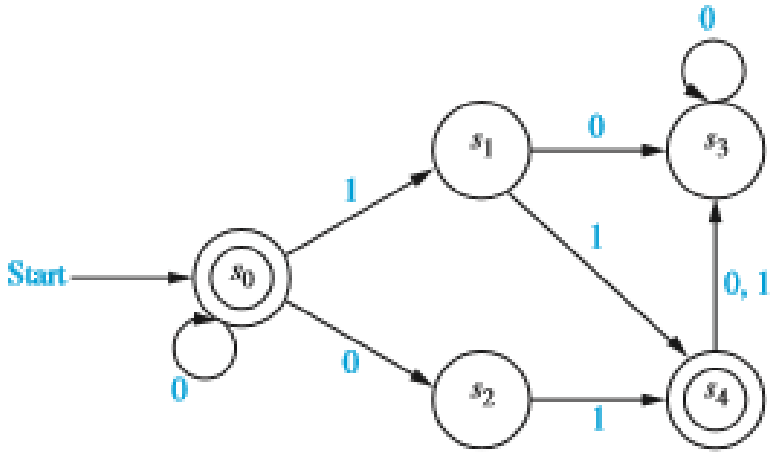


FIGURE 7 **A Nondeterministic Finite-State Automaton.**

Nondeterministic Finite-State Automata

Theorem 3.6.

If the language L is recognized by a non-deterministic finite-state automaton M_0, then L is also recognized by a deterministic finite-state automaton M_1.

Regular expression

Definition 4.1.

The regular expressions over a set I are defined recursively by:

Regular expression

Definition 4.1.

The regular expressions over a set I are defined recursively by:

- the symbol $\oslash$ is a regular expression;

Regular expression

Definition 4.1.

The regular expressions over a set I are defined recursively by:

- the symbol $\oslash$ is a regular expression;
- symbol λ is a regular expression;

Regular expression

Definition 4.1.

The regular expressions over a set I are defined recursively by:

- the symbol $\oslash$ is a regular expression;
- symbol λ is a regular expression;
- the symbol x is a regular expression whenever $x \in I$;

Regular expression

Definition 4.1.

The regular expressions over a set I are defined recursively by:

- the symbol $\oslash$ is a regular expression;
- symbol λ is a regular expression;
- the symbol x is a regular expression whenever $x \in I$;
- the symbols (AB),$(A \cup B)$, and A^* are regular expressions whenever A and B are regular expressions.

Regular expression

Definition 4.1.

The regular expressions over a set I are defined recursively by:

- the symbol $\oslash$ is a regular expression;
- symbol λ is a regular expression;
- the symbol x is a regular expression whenever $x \in I$;
- the symbols (AB),$(A \cup B)$, and A^* are regular expressions whenever A and B are regular expressions.

- Example: what are the strings in the regular sets specified by the regular expressions 10^*,$(10)^*$, $0 \cup 01$, $0(0 \cup 1)^*$, and $(0^*1)^*$?

Kleene Theorem

Theorem 4.2 (Kleene theorem).

A set is regular if and only if it is recognized by a finite-state automaton.

Regular Sets and Regular Grammars

Theorem 4.3.

A set is generated by a regular grammar if and only if it is a regular set.

Regular Sets and Regular Grammars

Theorem 4.3.

A set is generated by a regular grammar if and only if it is a regular set.

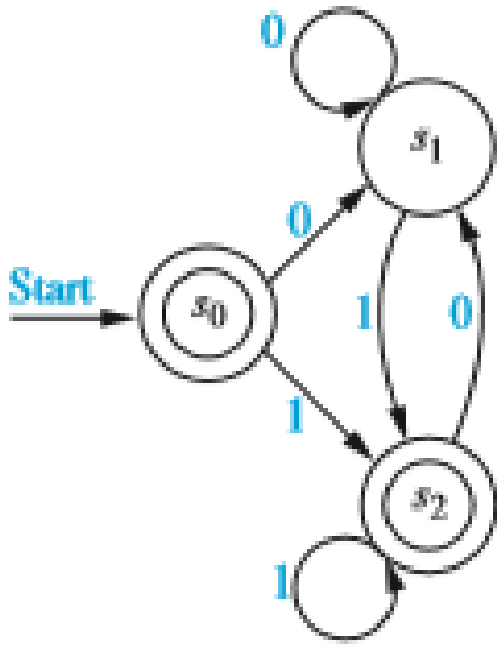
- Example: construct a non-deterministic finite-state automaton that recognizes the language generated by the regular grammar $G = (V, T, S, P)$, where $V = \{0, 1, A, S\}$, $T = \{0, 1\}$, and the productions in P are $S \to 1A$, $S \to 0$, $S \to \lambda$, $A \to 0A$, $A \to 1A$, and $A \to 1$.

Regular Sets and Regular Grammars

- Example: find a regular grammar that generates the regular set recognized by the finite-state automaton shown in the Figure below.

Regular Sets and Regular Grammars

- Example: find a regular grammar that generates the regular set recognized by the finite-state automaton shown in the Figure below.



A Set Not Recognized by a Finite-State Automaton

- Example: Show that the set $\{0^n 1^n | n = 0, 1, 2, ...\}$, made up of all strings consisting of a block of 0s followed by a block of an equal number of 1s, is not regular.

A Set Not Recognized by a Finite-State Automaton

- Example: Show that the set $\{0^n 1^n | n = 0, 1, 2, ...\}$, made up of all strings consisting of a block of 0s followed by a block of an equal number of 1s, is not regular.
- Finite-State Automaton are unable to carry out many computations due to the limitation of finite memory.

A Set Not Recognized by a Finite-State Automaton

- Example: Show that the set $\{0^n 1^n | n = 0, 1, 2, ...\}$, made up of all strings consisting of a block of 0s followed by a block of an equal number of 1s, is not regular.
- Finite-State Automaton are unable to carry out many computations due to the limitation of finite memory.
- Pushdown automaton

A Set Not Recognized by a Finite-State Automaton

- Example: Show that the set $\{0^n 1^n | n = 0, 1, 2, ...\}$, made up of all strings consisting of a block of 0s followed by a block of an equal number of 1s, is not regular.
- Finite-State Automaton are unable to carry out many computations due to the limitation of finite memory.
- Pushdown automaton
- Linear bounded automaton

A Set Not Recognized by a Finite-State Automaton

- Example: Show that the set $\{0^n 1^n | n = 0, 1, 2, ...\}$, made up of all strings consisting of a block of 0s followed by a block of an equal number of 1s, is not regular.
- Finite-State Automaton are unable to carry out many computations due to the limitation of finite memory.
- Pushdown automaton
- Linear bounded automaton
- Turing machine

Introduction

- The finite-state automata studied earlier in this chapter cannot be used as general models of computation.

Introduction

- The finite-state automata studied earlier in this chapter cannot be used as general models of computation.
- Finite-state automata are able to recognize regular sets, but are not able to recognize many easy-to-describe sets, including $\{0^n 1^n | n \geq 0\}$, which computers recognize using memory.

Introduction

- The finite-state automata studied earlier in this chapter cannot be used as general models of computation.
- Finite-state automata are able to recognize regular sets, but are not able to recognize many easy-to-describe sets, including $\{0^n 1^n | n \geq 0\}$, which computers recognize using memory.
- Basically, a Turing machine consists of a control unit, which at any step is in one of finitely many different states, together with a tape divided into cells, which is infinite in both directions.

Introduction

- The finite-state automata studied earlier in this chapter cannot be used as general models of computation.
- Finite-state automata are able to recognize regular sets, but are not able to recognize many easy-to-describe sets, including $\{0^n 1^n | n \geq 0\}$, which computers recognize using memory.
- Basically, a Turing machine consists of a control unit, which at any step is in one of finitely many different states, together with a tape divided into cells, which is infinite in both directions.
- Turing machines have read and write capabilities on the tape as the control unit moves back and forth along this tape, changing states depending on the tape symbol read.

Definition

Definition 5.1.

A Turing machine $T = (S, I, f, s_0)$ consists of a finite set S of states, an alphabet I containing the blank symbol B, a partial function f from $S \times I$ to $S \times I \times \{R, L\}$, and a starting state s_0.

Definition

Definition 5.1.

A Turing machine $T = (S, I, f, s_0)$ consists of a finite set S of states, an alphabet I containing the blank symbol B, a partial function f from $S \times I$ to $S \times I \times \{R, L\}$, and a starting state s_0.

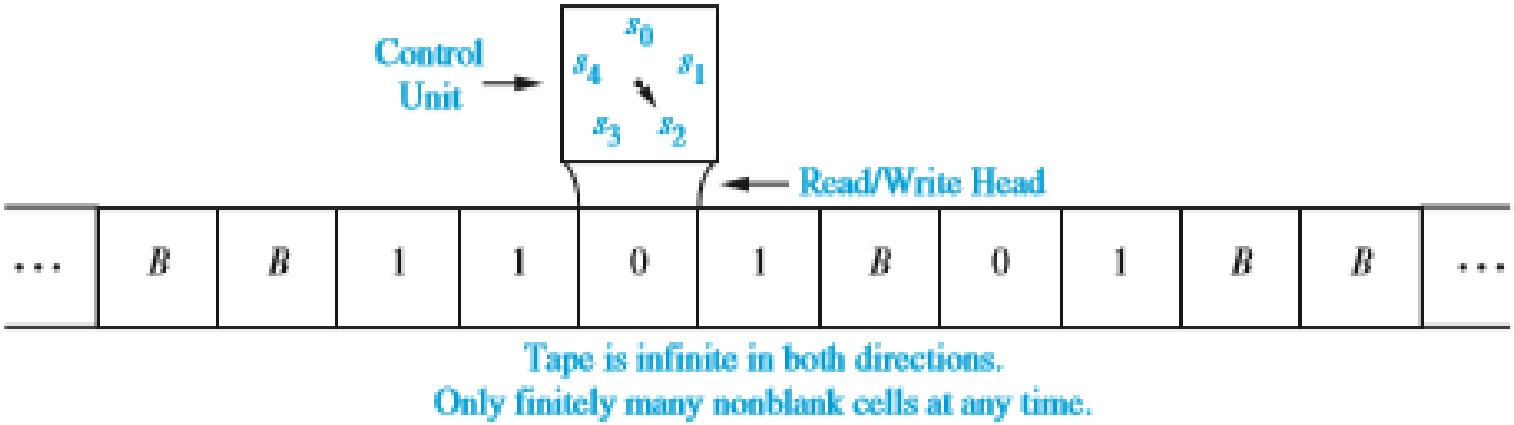


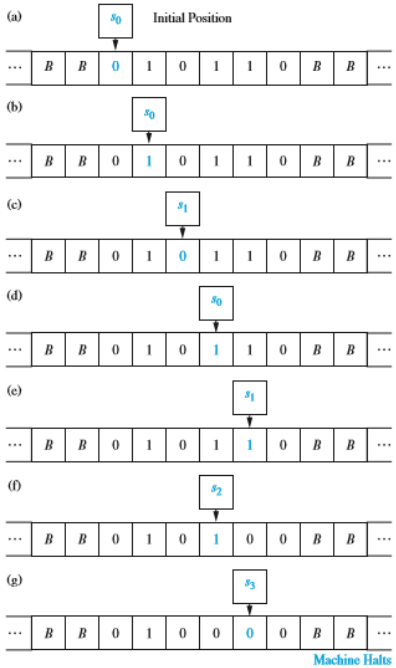
FIGURE 1 A Representation of a Turing Machine.

Example

- What is the final tape when the Turing machine T defined by the seven five-tuples $(s_0, 0, s_0, 0, R), (s_0, 1, s_1, 1, R), (s_0, B, s_3, B, R), (s_1, 0, s_0, 0, R), (s_1, 1, s_2, 0$ and $(s_2, 1, s_3, 0, R)$ is run on the tape shown in Figure (a) below?

Example

- What is the final tape when the Turing machine T defined by the seven five-tuples $(s_0, 0, s_0, 0, R), (s_0, 1, s_1, 1, R), (s_0, B, s_3, B, R), (s_1, 0, s_0, 0, R), (s_1, 1, s_2, 0$ and $(s_2, 1, s_3, 0, R)$ is run on the tape shown in Figure (a) below?



Turing Machines to Recognize Sets

Definition 5.2.

Let V be a subset of an alphabet I. A Turing machine $T = (S, I, f, s_0)$ recognizes a string x in V^* if and only if T, starting in the initial position when x is written on the tape, halts in a final state. T is said to recognize a subset A of V^* if x is recognized by T if and only if x belongs to A.

Turing Machines to Recognize Sets

Definition 5.2.

Let V be a subset of an alphabet I. A Turing machine $T = (S, I, f, s_0)$ recognizes a string x in V^* if and only if T, starting in the initial position when x is written on the tape, halts in a final state. T is said to recognize a subset A of V^* if x is recognized by T if and only if x belongs to A.

- Example: find a Turing machine that recognizes the set of bit strings that have a 1 as their second bit, that is, the regular set $(0 \cup 1)1(0 \cup 1)^*$.

Turing Machines to Recognize Sets

Definition 5.2.

Let V be a subset of an alphabet I. A Turing machine $T = (S, I, f, s_0)$ recognizes a string x in V^* if and only if T, starting in the initial position when x is written on the tape, halts in a final state. T is said to recognize a subset A of V^* if x is recognized by T if and only if x belongs to A.

- Example: find a Turing machine that recognizes the set of bit strings that have a 1 as their second bit, that is, the regular set $(0 \cup 1)1(0 \cup 1)^*$.
- Example: find a Turing machine that recognizes the set $\{0^n 1^n | n \geq 1\}$.

Some definitions

Definition 5.3.

A *decision problem* asks whether statements from a particular class of statements are true. Decision problems are also known as *yes-or-no problems*.

Some definitions

Definition 5.3.

A *decision problem* asks whether statements from a particular class of statements are true. Decision problems are also known as *yes-or-no problems*.

Definition 5.4.

The halting problem is the decision problem that asks whether a Turing machine T eventually halts when given an input string x.

Some definitions

Definition 5.3.

A *decision problem* asks whether statements from a particular class of statements are true. Decision problems are also known as *yes-or-no problems*.

Definition 5.4.

The halting problem is the decision problem that asks whether a Turing machine T eventually halts when given an input string x.

Theorem 5.5.

The halting problem is an unsolvable decision problem. That is, no Turing machine exists that, when given an encoding of a Turing machine T and its input string x as input, can determine whether T eventually halts when started with x written on its tape.

Some definitions

Definition 5.6.

A decision problem is in P, the *class of polynomial-time problems*, if it can be solved by a deterministic Turing machine in polynomial time in terms of the size of its input. That is, a decision problem is in P if there is a deterministic Turing machine T that solves the decision problem and a polynomial $p(n)$ such that for all integers n, T halts in a final state after no more than $p(n)$ transitions whenever the input to T is a string of length n. A decision problem is in NP, the *class of non-deterministic polynomial-time problems*, if it can be solved by a non-deterministic Turing machine in polynomial time in terms of the size of its input. That is, a decision problem is in NP if there is a non-deterministic Turing machine T that solves the problem and a polynomial $p(n)$ such that for all integers n, T halts for every choice of transitions after no more than $p(n)$ transitions whenever the input to T is a string of length n.