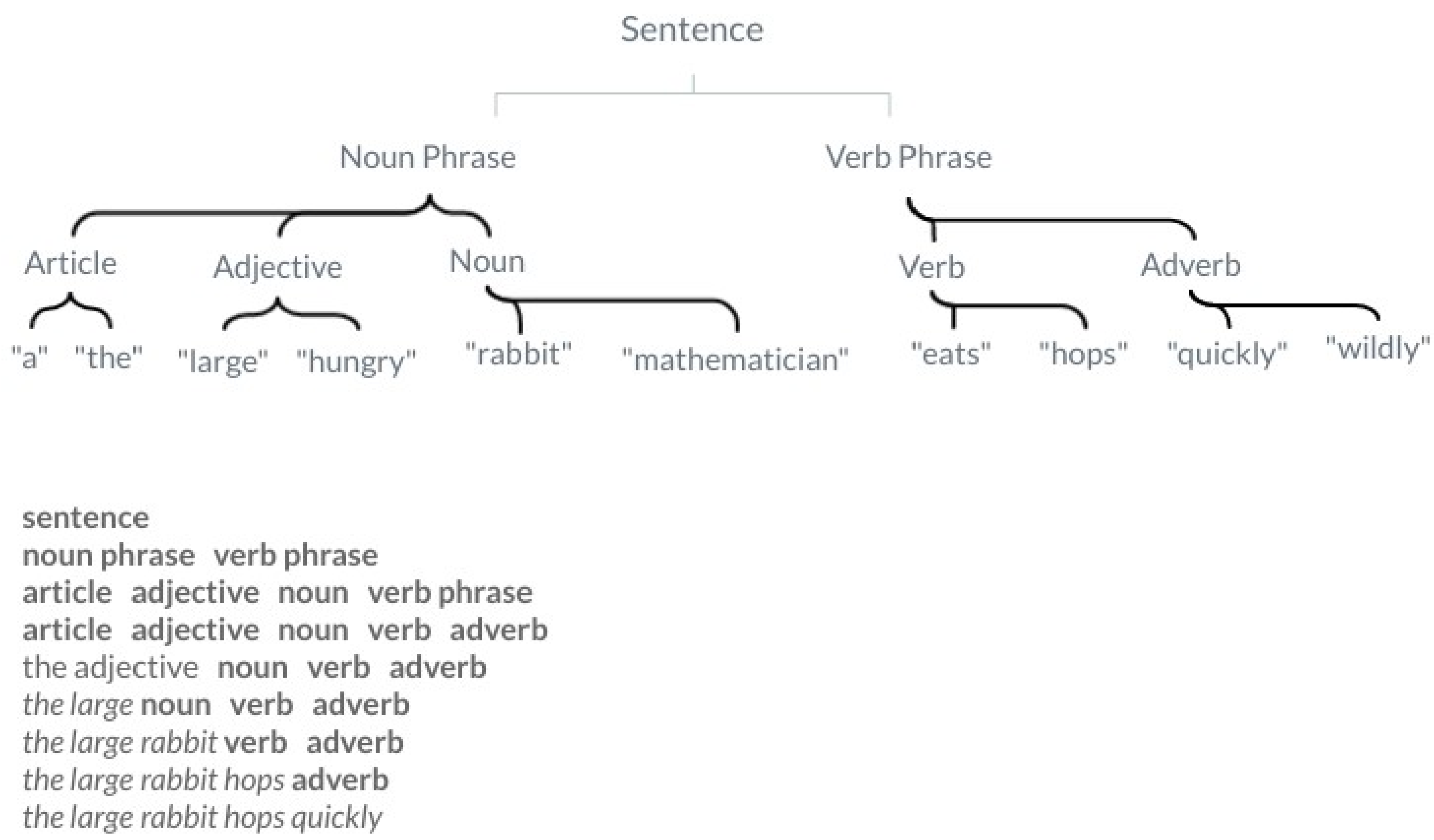
Ch 13 Modeling Computation (R)

Structures used in models of computation: **grammar, finite-state machines,** and **Turing Machines**.

Ch 13.1 Languages and Grammars

- **Syntax** is the form, **semantics** is the meaning.



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 (and sometimes by ϵ), 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^*.

- λ, the empty string, is different from $\emptyset$, the empty set. $\{\lambda\}$ is the set containing exactly one string.
- V is the set of symbols used to derive members of the language
- **Terminal** are elements of the vocabulary that cannot be replaced by other sumbols, T.
- **Nonterminal** are elements otherwise, N.

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 N. Elements of N are called *nonterminal symbols*. Every production in P must contain at least one nonterminal on its left side.

Let $G = (V, T, S, P)$ be a phrase-structure grammar. Let $w_0 = lz_0r$ (concatenation of l, z_0, 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 $w_0, w_1, \ldots, w_n$ are strings over V such that $w_0 \Rightarrow w_1, w_1 \Rightarrow w_2, \ldots, w_{n-1} \Rightarrow w_n$, 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*.

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

TABLE 1 Types of Grammars.

Type	*Restrictions on Productions* $w_1 \to w_2$
0	No restrictions
1	$w_1 = lAr$ and $w_2 = lwr$, where $A \in N$, $l, r, w \in (N \cup T)^*$ and $w \neq \lambda$; or $w_1 = S$ and $w_2 = \lambda$ as long as S is not on the right-hand side of another production
2	$w_1 = A$, where A is a nonterminal symbol
3	$w_1 = A$ and $w_2 = aB$ or $w_2 = a$, where $A \in N$, $B \in N$, and $a \in T$; or $w_1 = S$ and $w_2 = \lambda$

- **type 1** grammar are called **context-sensitive**
- **type 2** grammars are called **context-free grammars**
- **type 3** grammars are called **regular grammars**

▼ Example 12 **Top-down parsing & Bottom-up parsing**

Determine whether the word *cbab* belongs to the language generated by the grammar $G = (V, T, S, P)$, where $V = \{a, b, c, A, B, C, S\}, T = \{a, b, c\}$, S is the starting symbol, and the productions are

$S \to AB$

$A \to Ca$

$B \to Ba$

$B \to Cb$

$B \to b$

$C \to cb$

$C \to b$

Solution: (Top-down approach)

$S \Rightarrow AB \Rightarrow CaB \Rightarrow cbaB \Rightarrow cbab$

Solution: (Bottom-up approach)

$cbab \Leftarrow Cab \Leftarrow Ab \Leftarrow AB \Leftarrow S$

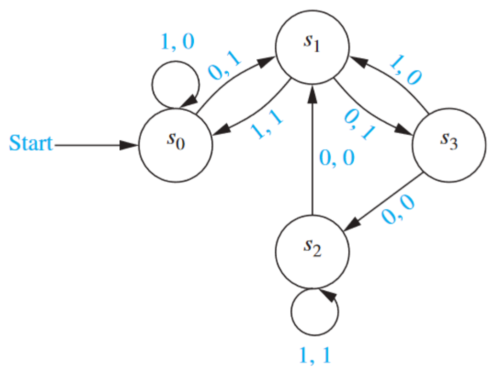
Ch 13.2 Finite-State Machines with Output

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.

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

Example of **state table**



Example of **state diagram**

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 *iff* the last output bit produced by M when given x as input is a 1.

Ch 13.3 Finite-State Machines with No Output

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 1

Let $A = \{0, 11\}$ and $B = \{1, 10, 110\}$. Find AB and BA.

Solution: The set AB contains every concatenation of a string in A and a string in B. Hence, $AB = \{01, 010, 0110, 111, 1110, 11110\}$. The set BA contains every concatenation of a string in B and a string in A. Hence, $BA = \{10, 111, 100, 1011, 1100, 11011\}$.

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

▼ Example 3

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

Solution:

$A^* = \{0^n \mid n = 0, 1, 2, \ldots\}$.

$B^* = V^*$

$C^* = \{1^{2n} \mid n = 0, 1, 2, \ldots\}$

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*).

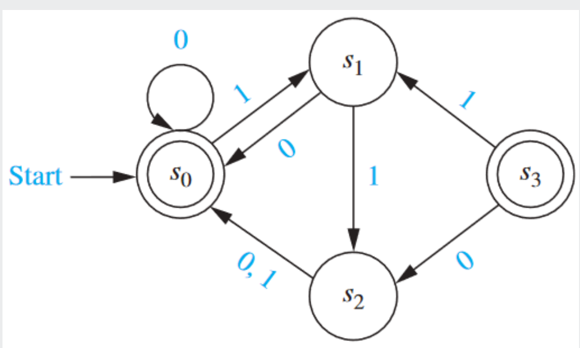
▼ Example 4

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.

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

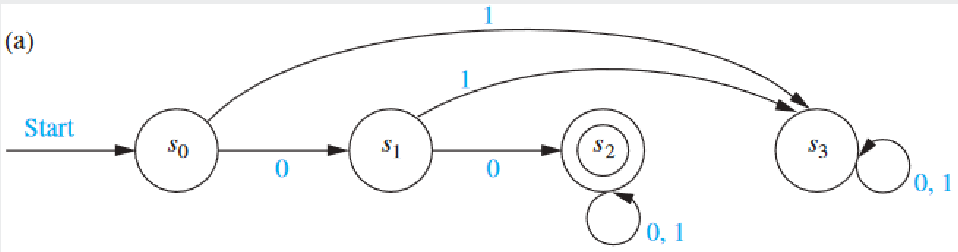
Solution:



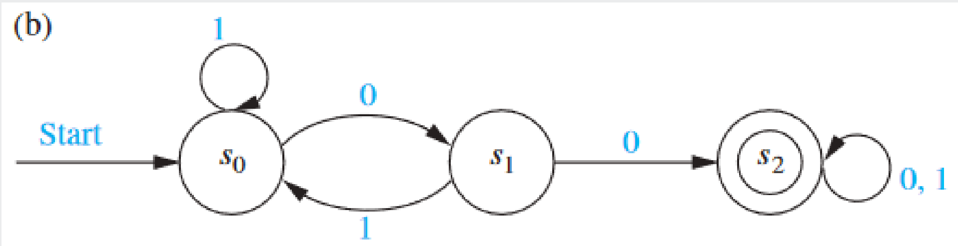
A string x is said to be *recognized* or *accepted* by machine $M = (S, I, f, s_0, F)$ if it takes the initial state s_0 to a final state, that is, $f(s_0, x)$ is a state in F. The *language recognized* or *accepted* by the machine M, denoted by $L(M)$, is the set of all strings that are recognized by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

▼ Example 6

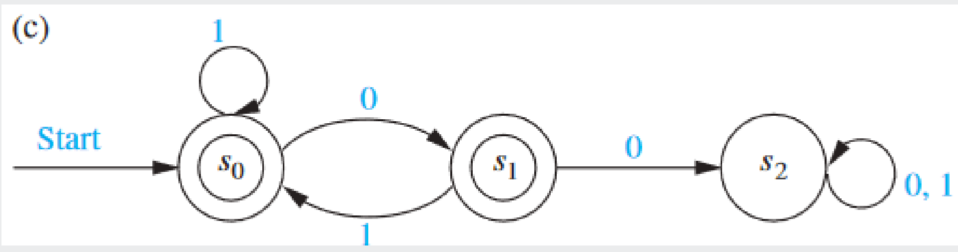
1. The set of bit strings that begin with two 0s



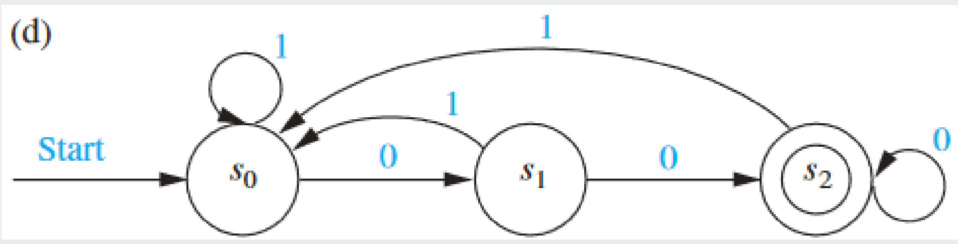
2. The set of bit strings that contain two consecutive 0s



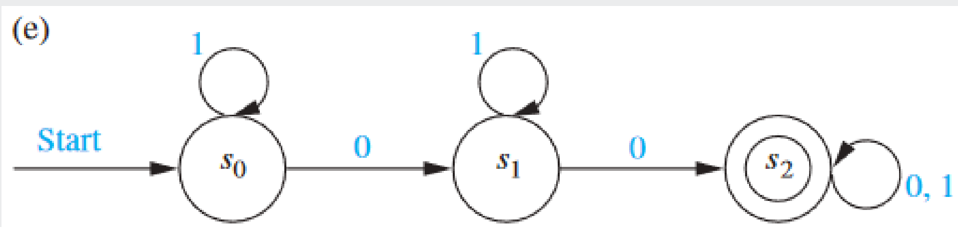
3. The set of bit strings that do not contain two consecutive 0s



4. The set of bit strings that end with two 0s



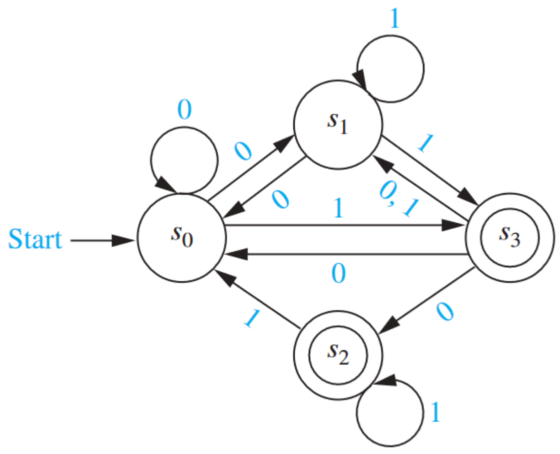
5. The set of bit strings that contain at least two 0s



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

TABLE 2

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



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

Ch 13.4 Language Recognition

> The *regular expressions* over a set I are defined recursively by:
> the symbol $\emptyset$ is a regular expression;
> the symbol λ is a regular expression;
> the symbol x is a regular expression whenever $x \in I$;
> the symbols (AB), $(A \cup B)$, A^* are regular expressions whenever A and B are regular expressions.

> **KLEENE'S THEOREM** A set is regular *iff* it is recognized by a finite-state automaton.

> A set is generated by a regular grammar *iff* it is a regular set.