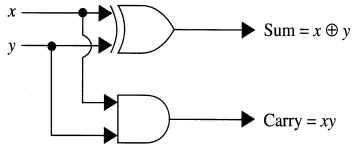
13.

x	y	$x \odot y$	$x \oplus y$	$\overline{(x \oplus y)}$
1	1	1	0	1
1	0	0	1	0
0	1	0	1	0
0	0	1	0	1

15. Yes, as a truth table shows **17. a)** 6 **b)** 5 **c)** 5 **d)** 6

19.



21. $x_3 + x_2\overline{x_1}$ **23.** Suppose it were with weights a and b. Then there would be a real number T such that $xa + yb \geq T$ for (1,0) and (0,1), but with $xa + yb < T$ for (0,0) and (1,1). Hence, $a \geq T$, $b \geq T$, $0 < T$, and $a + b < T$. Thus, a and b are positive, which implies that $a + b > a \geq T$, a contradiction.

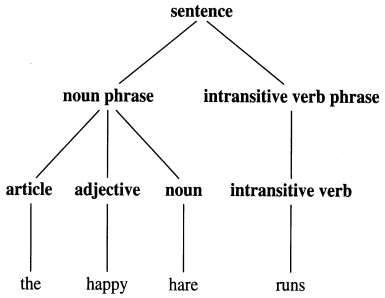
CHAPTER 12

Section 12.1

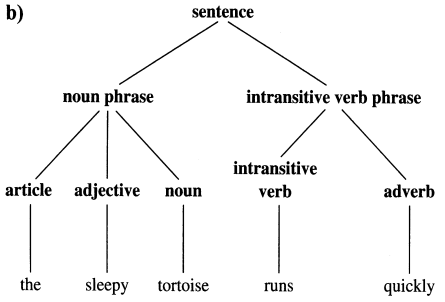
1. a) sentence ⇒ noun phrase intransitive verb phrase ⇒ article adjective noun intransitive verb phrase ⇒ article adjective noun intransitive verb ⇒ . . . (after 3 steps) . . . ⇒ *the happy hare runs.*
b) sentence ⇒ noun phrase intransitive verb phrase ⇒ article adjective noun intransitive verb phrase ⇒ article adjective noun intransitive verb adverb. . . (after 4 steps). . . ⇒ *the sleepy tortoise runs quickly*
c) sentence ⇒ noun phrase transitive verb phrase noun phrase ⇒ article noun transitive verb phrase noun phrase ⇒ article noun transitive verb noun phrase ⇒ article noun transitive verb article noun ⇒ . . . (after 4 steps). . . ⇒ *the tortoise passes the hare*
d) sentence ⇒ noun phrase transitive verb phrase noun phrase ⇒ article adjective noun transitive verb phrase noun phrase ⇒ article adjective noun transitive verb noun phrase ⇒ article adjective noun transitive verb article adjective noun ⇒ . . . (after 6 steps). . . ⇒ *the sleepy hare passes the happy tortoise*

3. The only way to get a noun, such as *tortoise*, at the end is to have a noun phrase at the end, which can be achieved only via the production **sentence → noun phrase transitive verb phrase noun phrase**. However, **transitive verb phrase → transitive verb** → *passes*, and this sentence does not contain *passes*.
5. a) $S \Rightarrow 1A \Rightarrow 10B \Rightarrow 101A \Rightarrow 1010B \Rightarrow 10101$ **b)** Because of the productions in this grammar, every 1 must be followed by a 0 unless it occurs at the end of the string. **c)** All strings consisting of a 0 or a 1 followed by one or more repetitions of 01
7. $S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 000111$
9. a) $S \Rightarrow 0S \Rightarrow 00S \Rightarrow 00S1 \Rightarrow 00S11 \Rightarrow 00S111 \Rightarrow 00S1111 \Rightarrow 001111$ **b)** $S \Rightarrow 0S \Rightarrow 00S \Rightarrow 001A \Rightarrow 0011A \Rightarrow 00111A \Rightarrow 001111$ **11.** $S \Rightarrow 0SAB \Rightarrow 00SABAB \Rightarrow 00ABAB \Rightarrow 00AABB \Rightarrow 001ABB \Rightarrow 0011BB \Rightarrow 00112B \Rightarrow 001122$
13. a) $S \to 0, S \to 1, S \to 11$ **b)** $S \to 1S, S \to \lambda$ **c)** $S \to 0A1, A \to 1A, A \to 0A, A \to \lambda$ **d)** $S \to 0A, A \to 11A, A \to \lambda$ **15. a)** $S \to 00S, S \to \lambda$ **b)** $S \to 10A, A \to 00A, A \to \lambda$ **c)** $S \to AAS, S \to BBS, AB \to BA, BA \to AB, S \to \lambda, A \to 0, B \to 1$ **d)** $S \to 0000000000A, A \to 0A, A \to \lambda$ **e)** $S \to AS, S \to ABS, S \to A, AB \to BA, BA \to AB, A \to 0, B \to 1$ **f)** $S \to ABS, S \to \lambda, AB \to BA, BA \to AB, A \to 0, B \to 1$ **g)** $S \to ABS, S \to T, S \to U, T \to AT, T \to A, U \to BU, U \to B, AB \to BA, BA \to AB, A \to 0, B \to 1$ **17. a)** $S \to 0S, S \to \lambda$ **b)** $S \to A0, A \to 1A, A \to \lambda$ **c)** $S \to 000S, S \to \lambda$ **19. a)** Type 2, not type 3 **b)** Type 3 **c)** Type 0, not type 1 **d)** Type 2, not type 3 **e)** Type 2, not type 3 **f)** Type 0, not type 1 **g)** Type 3 **h)** Type 0, not type 1 **i)** Type 2, not type 3 **j)** Type 2, not type 3 **21.** Let S_1 and S_2 be the start symbols of G_1 and G_2, respectively. Let S be a new start symbol. **a)** Add S and productions $S \to S_1$ and $S \to S_2$. **b)** Add S and production $S \to S_1S_2$. **c)** Add S and production $S \to \lambda$ and $S \to S_1S$.

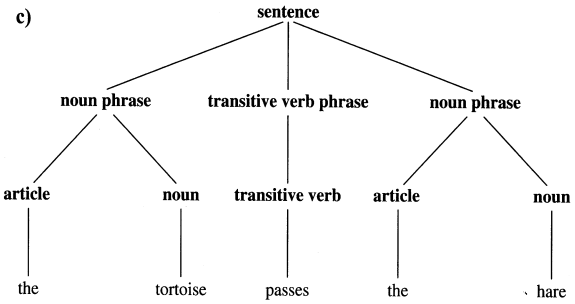
23. a)



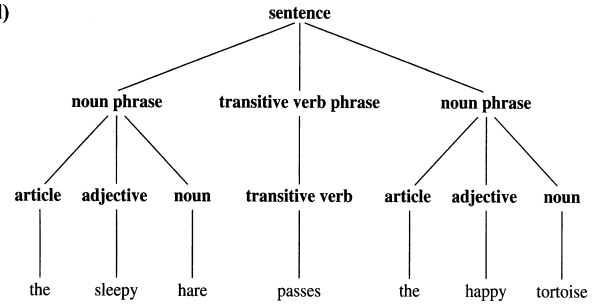
b)



c)

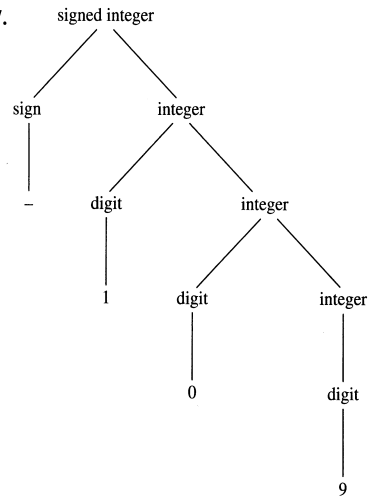


d)



25. a) Yes **b)** No **c)** Yes **d)** No

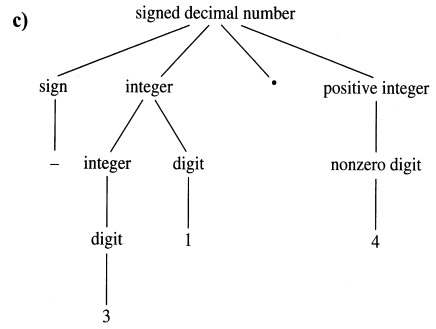
27.



29. a) $S \rightarrow \langle sign \rangle \langle integer \rangle$
$S \rightarrow \langle sign \rangle \langle integer \rangle . \langle positive\ integer \rangle$
$\langle sign \rangle \rightarrow +$
$\langle sign \rangle \rightarrow -$
$\langle integer \rangle \rightarrow \langle digit \rangle$
$\langle integer \rangle \rightarrow \langle integer \rangle \langle digit \rangle$
$\langle digit \rangle \rightarrow i, i = 1, 2, 3, 4, 5, 6, 7, 8, 9, 0$
$\langle positive\ integer \rangle \rightarrow \langle integer \rangle \langle nonzero\ digit \rangle$
$\langle positive\ integer \rangle \rightarrow \langle nonzero\ digit \rangle \langle integer \rangle$
$\langle positive\ integer \rangle \rightarrow \langle integer \rangle \langle nonzero\ digit \rangle \langle integer \rangle$
$\langle positive\ integer \rangle \rightarrow \langle nonzero\ digit \rangle$
$\langle nonzero\ digit \rangle \rightarrow i, i = 1, 2, 3, 4, 5, 6, 7, 8, 9$

b) $\langle signed\ decimal\ number \rangle ::= \langle sign \rangle \langle integer \rangle \mid \langle sign \rangle \langle integer \rangle . \langle positive\ integer \rangle$
$\langle sign \rangle ::= + \mid -$
$\langle integer \rangle ::= \langle digit \rangle \mid \langle integer \rangle \langle digit \rangle$
$\langle digit \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
$\langle nonzero\ digit \rangle ::= 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
$\langle positive\ integer \rangle ::= \langle integer \rangle \langle nonzero\ digit \rangle \mid \langle nonzero\ digit \rangle \langle integer \rangle \mid \langle integer \rangle \langle nonzero\ integer \rangle \langle integer \rangle \mid \langle nonzero\ digit \rangle$

c)



31. a) ⟨*identifier*⟩ ::= ⟨*lcletter*⟩ | ⟨*identifier*⟩⟨*lcletter*⟩
⟨*lcletter*⟩ ::= *a* | *b* | *c* | · · · | *z*

b) ⟨*identifier*⟩ ::= ⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩ | ⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩ |
⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩ |
⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩⟨*lcletter*⟩
⟨*lcletter*⟩ ::= *a* | *b* | *c* | · · · | *z*

c) ⟨*identifier*⟩ ::= ⟨*ucletter*⟩ | ⟨*ucletter*⟩⟨*letter*⟩ | ⟨*ucletter*⟩⟨*letter*⟩⟨*letter*⟩ |
⟨*ucletter*⟩⟨*letter*⟩⟨*letter*⟩⟨*letter*⟩ | ⟨*ucletter*⟩⟨*letter*⟩⟨*letter*⟩⟨*letter*⟩⟨*letter*⟩ |
⟨*ucletter*⟩⟨*letter*⟩⟨*letter*⟩⟨*letter*⟩⟨*letter*⟩⟨*letter*⟩
⟨*letter*⟩ ::= ⟨*lcletter*⟩ | ⟨*ucletter*⟩
⟨*lcletter*⟩ ::= *a* | *b* | *c* | · · · | *z*
⟨*ucletter*⟩ ::= *A* | *B* | *C* | · · · | *Z*

d) ⟨*identifier*⟩ ::= ⟨*lcletter*⟩⟨*digitorus*⟩⟨*alphanumeric*⟩⟨*alphanumeric*⟩⟨*alphanumeric*⟩ |
⟨*lcletter*⟩⟨*digitorus*⟩⟨*alphanumeric*⟩⟨*alphanumeric*⟩⟨*alphanumeric*⟩⟨*alphanumeric*⟩
⟨*digitorus*⟩ ::= ⟨*digit*⟩ | _
⟨*alphanumeric*⟩ ::= ⟨*letter*⟩ | ⟨*digit*⟩
⟨*letter*⟩ ::= ⟨*lcletter*⟩ | ⟨*ucletter*⟩
⟨*lcletter*⟩ ::= *a* | *b* | *c* | · · · | *z*
⟨*ucletter*⟩ ::= *A* | *B* | *C* | · · · | *Z*
⟨*digit*⟩ ::= 0 | 1 | 2 | · · · | 9

33. ⟨*identifier*⟩ ::= ⟨*letterorus*⟩ | ⟨*identifier*⟩⟨*symbol*⟩
⟨*letterorus*⟩ ::= ⟨*letter*⟩ | _
⟨*symbol*⟩ ::= ⟨*letterorus*⟩ | ⟨*digit*⟩
⟨*letter*⟩ ::= ⟨*lcletter*⟩ | ⟨*ucletter*⟩
⟨*lcletter*⟩ ::= *a* | *b* | *c* | · · · | *z*
⟨*ucletter*⟩ ::= *A* | *B* | *C* | · · · | *Z*
⟨*digit*⟩ ::= 0 | 1 | 2 | · · · | 9

35. *numeral* ::= *sign*? *nonzerodigit digit*∗ *decimal*? | *sign*? 0 *decimal*?
sign ::= + | −
nonzerodigit ::= 1 | 2 | · · · | 9
digit ::= 0 | *nonzerodigit*
decimal ::= .*digit*∗

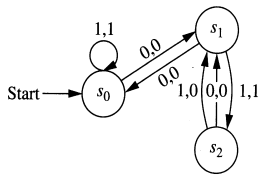
37. *identifier* ::= *letterorus symbol*∗
letterorus ::= *letter* | _
symbol ::= *letterorus* | *digit*
letter ::= *lcletter* | *ucletter*
lcletter ::= *a* | *b* | *c* | · · · | *z*
ucletter ::= *A* | *B* | *C* | · · · | *Z*
digit ::= 0 | 1 | 2 | · · · | 9

39. a) ⟨*expression*⟩
⟨*term*⟩ ⟨*term*⟩ ⟨*addOperator*⟩
⟨*factor*⟩ ⟨*factor*⟩ ⟨*factor*⟩ ⟨*mulOperator*⟩ ⟨*addOperator*⟩
⟨*identifier*⟩ ⟨*identifier*⟩ ⟨*identifier*⟩ ⟨*mulOperator*⟩ ⟨*addOperator*⟩
a b c ∗ +
b) Not generated
c) ⟨*expression*⟩
⟨*term*⟩
⟨*factor*⟩ ⟨*factor*⟩ ⟨*mulOperator*⟩
⟨*expression*⟩ ⟨*factor*⟩ ⟨*mulOperator*⟩
⟨*term*⟩ ⟨*term*⟩ ⟨*addOperator*⟩ ⟨*factor*⟩ ⟨*mulOperator*⟩
⟨*factor*⟩ ⟨*factor*⟩ ⟨*addOperator*⟩ ⟨*factor*⟩ ⟨*mulOperator*⟩
⟨*identifier*⟩ ⟨*identifier*⟩ ⟨*addOperator*⟩ ⟨*identifier*⟩ ⟨*mulOperator*⟩
x y − *z* ∗
d) ⟨*expression*⟩
⟨*term*⟩
⟨*factor*⟩ ⟨*factor*⟩ ⟨*mulOperator*⟩
⟨*factor*⟩ ⟨*expression*⟩ ⟨*mulOperator*⟩
⟨*factor*⟩ ⟨*term*⟩ ⟨*mulOperator*⟩
⟨*factor*⟩ ⟨*factor*⟩ ⟨*factor*⟩ ⟨*mulOperator*⟩ ⟨*mulOperator*⟩
⟨*factor*⟩ ⟨*factor*⟩ ⟨*expression*⟩ ⟨*mulOperator*⟩ ⟨*mulOperator*⟩
⟨*factor*⟩ ⟨*factor*⟩ ⟨*term*⟩ ⟨*term*⟩ ⟨*addOperator*⟩ ⟨*mulOperator*⟩ ⟨*mulOperator*⟩
⟨*factor*⟩ ⟨*factor*⟩ ⟨*factor*⟩ ⟨*factor*⟩ ⟨*addOperator*⟩ ⟨*mulOperator*⟩ ⟨*mulOperator*⟩
⟨*identifier*⟩ ⟨*identifier*⟩ ⟨*identifier*⟩ ⟨*identifier*⟩ ⟨*addOperator*⟩ ⟨*mulOperator*⟩ ⟨*mulOperator*⟩
w x y z − ∗ /
e) ⟨*expression*⟩
⟨*term*⟩
⟨*factor*⟩ ⟨*factor*⟩ ⟨*mulOperator*⟩
⟨*factor*⟩ ⟨*expression*⟩ ⟨*mulOperator*⟩
⟨*factor*⟩ ⟨*term*⟩ ⟨*term*⟩ ⟨*addOperator*⟩ ⟨*mulOperator*⟩
⟨*factor*⟩ ⟨*factor*⟩ ⟨*factor*⟩ ⟨*addOperator*⟩ ⟨*mulOperator*⟩
⟨*identifier*⟩ ⟨*identifier*⟩ ⟨*identifier*⟩ ⟨*addOperator*⟩ ⟨*mulOperator*⟩
a d e − ∗
41. a) Not generated
b) ⟨*expression*⟩
⟨*term*⟩ ⟨*addOperator*⟩ ⟨*term*⟩
⟨*factor*⟩ ⟨*mulOperator*⟩ ⟨*factor*⟩ ⟨*addOperator*⟩ ⟨*factor*⟩ ⟨*mulOperator*⟩ ⟨*factor*⟩
⟨*identifier*⟩ ⟨*mulOperator*⟩ ⟨*identifier*⟩ ⟨*addOperator*⟩ ⟨*identifier*⟩ ⟨*mulOperator*⟩ ⟨*identifier*⟩
a/*b* + *c*/*d*
c) ⟨*expression*⟩
⟨*term*⟩
⟨*factor*⟩ ⟨*mulOperator*⟩ ⟨*factor*⟩
⟨*factor*⟩ ⟨*mulOperator*⟩ (⟨*expression*⟩)
⟨*factor*⟩ ⟨*mulOperator*⟩ (⟨*term*⟩ ⟨*addOperator*⟩ ⟨*term*⟩)
⟨*factor*⟩ ⟨*mulOperator*⟩ (⟨*factor*⟩ ⟨*addOperator*⟩ ⟨*factor*⟩)
⟨*identifier*⟩ ⟨*mulOperator*⟩ (⟨*identifier*⟩ ⟨*addOperator*⟩ ⟨*identifier*⟩)
m ∗ (*n* + *p*)

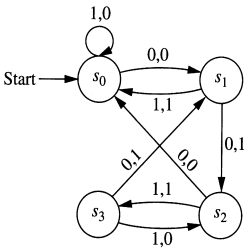
d) Not generated
e) ⟨*expression*⟩
⟨*term*⟩
⟨*factor*⟩⟨*mulOperator*⟩⟨*factor*⟩
(⟨*expression*⟩)⟨*mulOperator*⟩(⟨*expression*⟩)
(⟨*term*⟩⟨*addOperator*⟩⟨*term*⟩)⟨*mulOperator*⟩(⟨*term*⟩⟨*addOperator*⟩⟨*term*⟩)
(⟨*factor*⟩⟨*addOperator*⟩⟨*factor*⟩)⟨*mulOperator*⟩(⟨*factor*⟩⟨*addOperator*⟩⟨*factor*⟩)
(⟨*identifier*⟩⟨*addOperator*⟩⟨*identifier*⟩)⟨*mulOperator*⟩(⟨*identifier*⟩⟨*addOperator*⟩⟨*identifier*⟩)
$(m + n) * (p - q)$

Section 12.2

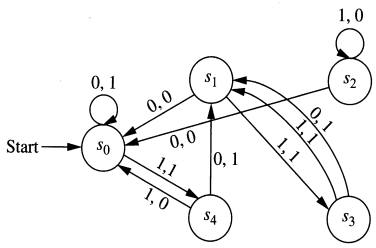
1. a)



b)

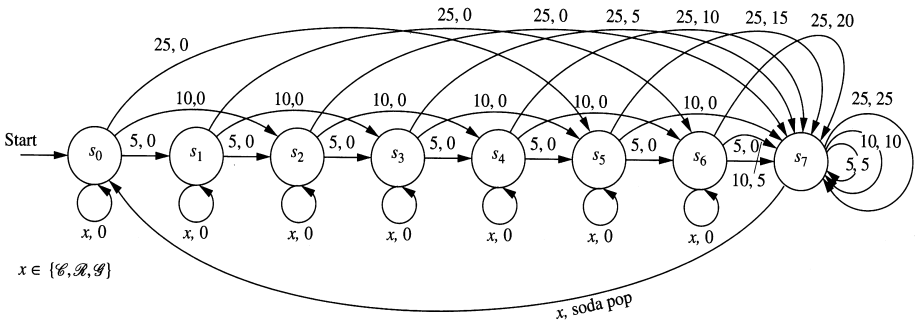


c)



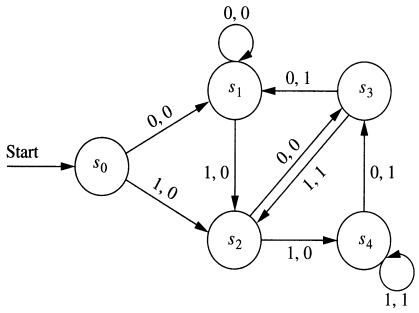
3. a) 01010 **b)** 01000 **c)** 11011 **5. a)** 1100 **b)** 00110110 **c)** 11111111111

7.

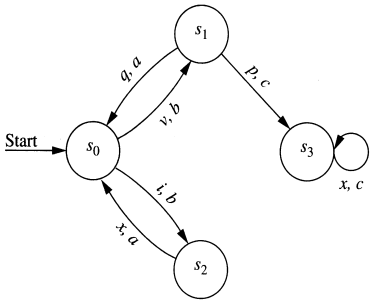


$\mathscr{C}$ = cola
$\mathscr{R}$ = root beer
$\mathscr{G}$ = ginger ale

9.

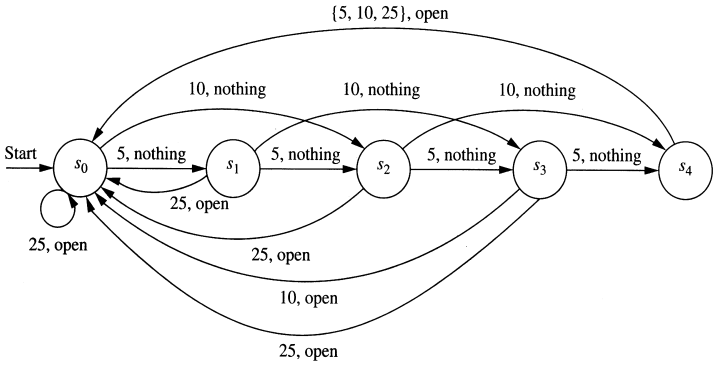


11.



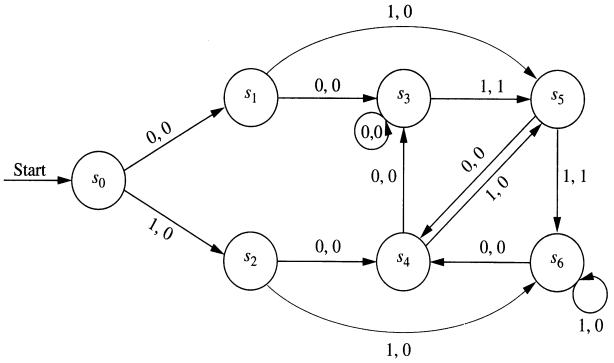
v = Valid ID
i = Invalid ID
p = Valid password
q = Invalid password
a = "Enter user ID"
b = "Enter password"
c = Prompt
x = Any input

13.

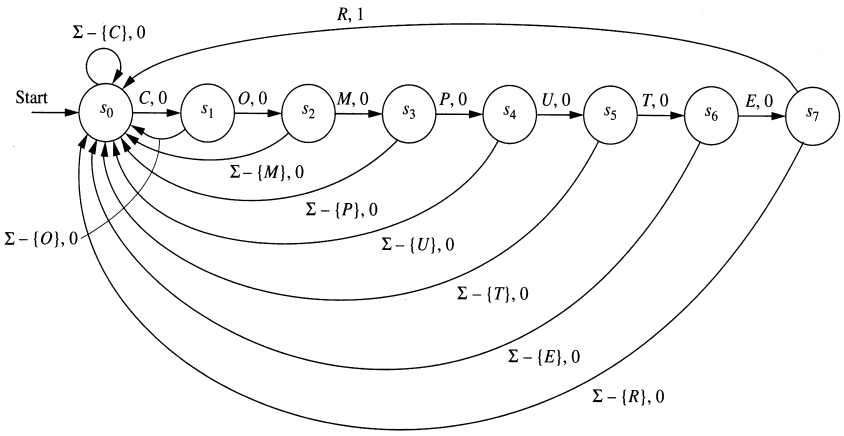


15. Let s_0 be the start state and let s_1 be the state representing a successful call. From s_0, inputs of 2, 3, 4, 5, 6, 7, or 8 send the machine back to s_0 with output of an error message for the user. From s_0 an input of 0 sends the machine to state s_1, with the output being that the 0 is sent to the network. From s_0 an input of 9 sends the machine to state s_2 with no output; from there an input of 1 sends the machine to state s_3 with no output; from there an input of 1 sends the machine to state s_1 with the output being that the 911 is sent to the network. All other inputs while in states s_2 or s_3 send the machine back to s_0 with output of an error message for the user. From s_0 an input of 1 sends the machine to state s_4 with no output; from s_4 an input of 2 sends the machine to state s_5 with no output; and this path continues in a similar manner to the 911 path, looking next for 1, then 2, then any seven digits, at which point the machine goes to state s_1 with the output being that the ten-digit input is sent to the network. Any "incorrect" input while in states s_5 or s_6 (that is, anything except a 1 while in s_5 or a 2 while in s_6) sends the machine back to s_0 with output of an error message for the user. Similarly, from s_4 an input of 8 followed by appropriate successors drives us eventually to s_1, but inappropriate outputs drive us back to s_0 with an error message. Also, inputs while in state s_4 other than 2 or 8 send the machine back to state s_0 with output of an error message for the user.

17.



19.

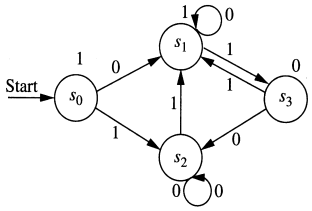


21.

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

23. a) 11111
b) 1000000
c) 100011001100

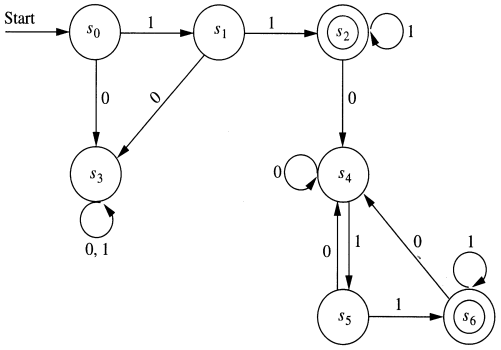
25.



Section 12.3

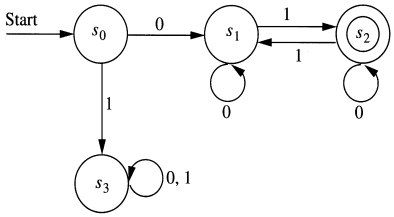
1. a) {000, 001, 1100, 1101} **b)** {000, 0011, 010, 0111} **c)** {00, 011, 110, 1111} **d)** {000000, 000001, 000100, 000101, 010000, 010001, 010100, 010101} **3.** $A = \{1, 101\}$, $B = \{0, 11, 000\}$; $A = \{10, 111, 1010, 1000, 10111, 101000\}$, $B = \{\lambda\}$; $A = \{\lambda, 10\}$, $B = \{10, 111, 1000\}$ or $A = \{\lambda\}$, $B = \{10, 111, 1010, 1000, 10111, 101000\}$ **5. a)** The set of strings consisting of zero or more consecutive bit pairs 10 **b)** The set of strings consisting of all 1s such that the number of 1s is divisible by 3, including the null string **c)** The set of strings in which every 1 is immediately preceded by a 0 **d)** The set of strings that begin and end with a 1 and have at least two 1s between every pair of 0s **7.** A string is in A^* if and only if it is a concatenation of an arbitrary number of strings in A. Because each string in A is also in B, it follows that a string in A^* is also a concatenation of strings in B. Hence, $A^* \subseteq B^*$. **9. a)** Yes **b)** Yes **c)** No **d)** No **e)** Yes **f)** Yes **11. a)** Yes **b)** No **c)** Yes **d)** No **13. a)** Yes **b)** Yes **c)** No **d)** No **e)** No **f)** No **15.** We use structural induction on the input string y. The basis step considers $y = \lambda$, and for the inductive step we write $y = wa$, where $w \in I^*$ and $a \in I$. For the basis step, we have $xy = x$, so we must show that $f(s, x) = f(f(s, x), \lambda)$. But part (*i*) of the definition of the extended transition function says that this is true. We then assume the inductive hypothesis that the equation holds for w and prove that $f(s, xwa) = f(f(s, x), wa)$. By part (*ii*) of the definition, the left-hand side of this equation equals $f(f(s, xw), a)$. By the inductive hypothesis, $f(s, xw) = f(f(s, x), w)$, so $f(f(s, xw), a) = f(f(f(s, x), w), a)$. The right-hand side of our desired equality is, by part (*ii*) of the definition, also equal to $f(f(f(s, x), w), a)$, as desired. **17.** {0, 10, 11}{0, 1}* **19.** $\{0^m 1^n \mid m \geq 0 \text{ and } n \geq 1\}$ **21.** $\{\lambda\} \cup \{0\}\{1\}^*\{0\} \cup \{10, 11\}\{0, 1\}^* \cup \{0\}\{1\}^*\{01\}\{0, 1\}^* \cup \{0\}\{1\}^*\{00\}\{0\}^*\{1\}\{0, 1\}^*$ **23.** Let s_2 be the only final state, and put transitions from s_2 to itself on either input. Put a transition from the start state s_0 to s_1 on input 0, and a transition from s_1 to s_2 on input 1. Create state s_3, and have the other transitions from s_0 and s_1 (as well as both transitions from s_3) lead to s_3. **25.** Start state s_0, only final state s_3; transitions from s_0 to s_0 on 0, from s_0 to s_1 on 1, from s_1 to s_2 on 0, from s_1 to s_1 on 1, from s_2 to s_0 on 0, from s_2 to s_3 on 1, from s_3 to s_3 on 0, from s_3 to s_3 on 1 **27.** Have five states, with only s_3 final. For $i = 0, 1, 2, 3$, transition from s_i to itself on input 1 and to s_{i+1} on input 0. Both transitions from s_4 are to itself. **29.** Have four states, with only s_3 final. For $i = 0, 1, 2$, transition from s_i to s_{i+1} on input 1 but back to s_0 on input 0. Both transitions from s_3 are to itself.

31.



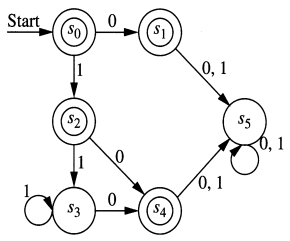
33. Start state s_0, only final state s_1; transitions from s_0 to s_0 on 1, from s_0 to s_1 on 0, from s_1 to s_1 on 1; from s_1 to s_0 on 0

35.



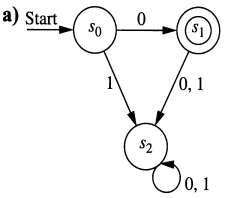
37. Suppose that such a machine exists, with start state s_0 and other state s_1. Because the empty string is not in the language but some strings are accepted, we must have s_1 as the only final state, with at least one transition from s_0 to s_1. Because the string 0 is not in the language, the transition from s_0 on input 0 must be to itself, so the transition from s_0 on input 1 must be to s_1. But this contradicts the fact that 1 is not in the language. **39.** Change each final state to a nonfinal state and vice versa. **41.** Same machine as in Exercise 25, but with s_0, s_1, and s_2 as the final states **43.** {0,01,11} **45.** $\{\lambda, 0\} \cup \{0^m 1^n \mid m \geq 1, n \geq 1\}$ **47.** $\{10^n \mid n \geq 0\} \cup \{10^n 10^m \mid n, m \geq 0\}$ **49.** The union of the set of all strings that start with a 0 and the set of all strings that have no 0s

51.

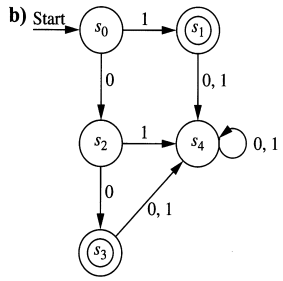


53. Add a nonfinal state s_3 with transitions to s_3 from s_0 on input 0, from s_1 on input 1, and from s_3 on input 0 or 1.

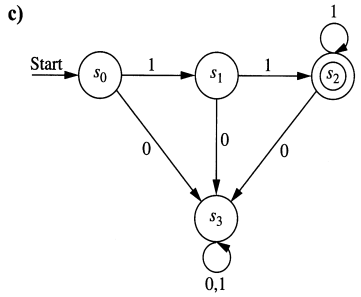
55. a)



b)



c)

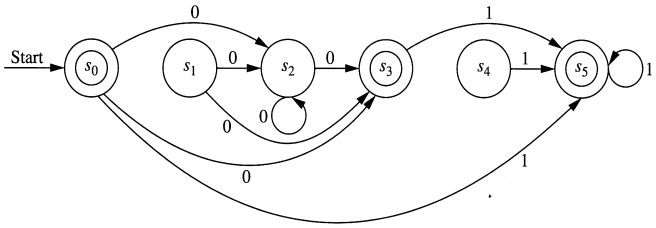


57. Suppose that M is a finite-state automaton that accepts the set of bit strings containing an equal number of 0s and 1s. Suppose M has n states. Consider the string $0^{n+1}1^{n+1}$. By the pigeonhole principle, as M processes this string, it must encounter the same state more than once as it reads the first $n + 1$ 0s; so let s be a state it hits at least twice. Then k 0s in the input takes M from state s back to itself for some positive integer k. But then M ends up exactly at the same place after reading $0^{n+1+k}1^{n+1}$ as it will after reading $0^{n+1}1^{n+1}$. Therefore, because M accepts $0^{n+1}1^{n+1}$ it also accepts $0^{n+k+1}1^{n+1}$, which is a contradiction. **59.** We know from Exercise 58d that the equivalence classes of R_k are a refinement of the equivalence classes of R_{k-1} for each positive integer k. The equivalence classes are finite sets, and finite sets cannot be refined indefinitely (the most refined they can be is for each equivalence class to contain just one state). Therefore this sequence of refinements must remain unchanged from some point onward. It remains to show that as soon as we have $R_n = R_{n+1}$, then $R_n = R_m$ for all $m > n$, from which it follows that $R_n = R_*$, and so the equivalence classes for these two relations will be the same. By induction, it suffices to show that if $R_n = R_{n+1}$, then $R_{n+1} = R_{n+2}$. Suppose that $R_{n+1} \neq R_{n+2}$. This means that there are states s and t that are $(n + 1)$-equivalent but not $(n + 2)$-equivalent. Thus there is a string x of length $n + 2$ such that, say, $f(s, x)$ is final but $f(t, x)$ is nonfinal. Write $x = aw$, where $a \in I$. Then $f(s, a)$ and $f(t, a)$ are not $(n + 1)$-equivalent, because w drives the first to a final state and the second to a nonfinal state. But $f(s, a)$ and $f(t, a)$ are n-equivalent, because s and t are $(n + 1)$-equivalent. This contradicts the fact that $R_n = R_{n+1}$. **61. a)** By the way the machine $\overline{M}$ was constructed, a string will drive M from the start state to a final state if and only if that string drives $\overline{M}$ from the start state to a final state. **b)** For a proof of this theorem, see a source such as *Introduction to Automata Theory, Languages, and Computation* (2nd Edition) by John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman (Addison-Wesley, 2000).

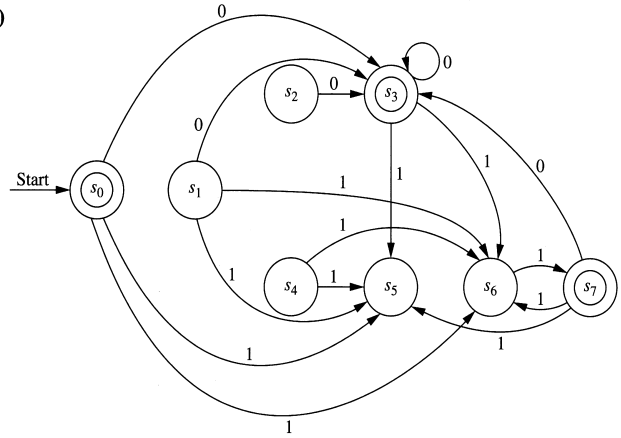
Section 12.4

1. a) Any number of 1s followed by a 0 **b)** Any number of 1s followed by one or more 0s **c)** 111 or 001 **d)** A string of any number of 1s or 00s or some of each in a row **e)** λ or a string that ends with a 1 and has one or more 0s before each 1 **f)** A string of length at least 3 that ends with 00 **3. a)** No **b)** No **c)** Yes **d)** Yes **e)** Yes **f)** No **g)** No **h)** Yes **5. a)** $\mathbf{0 \cup 11 \cup 010}$ **b)** $\mathbf{000000^*}$ **c)** $\mathbf{(0 \cup 1)((0 \cup 1)(0 \cup 1))^*}$ **d)** $\mathbf{0^*10^*}$ **e)** $\mathbf{(1 \cup 01 \cup 001)^*}$ **7. a)** $\mathbf{00^*1}$ **b)** $\mathbf{(0 \cup 1)(0 \cup 1)(0 \cup 1)^*0000^*}$ **c)** $\mathbf{0^*1^* \cup 1^*0^*}$ **d)** $\mathbf{11(111)^*(00)^*}$ **9. a)** Have the start state s_0, nonfinal, with no transitions. **b)** Have the start state s_0, final, with no transitions. **c)** Have the nonfinal start state s_0 and a final state s_1 and the transition from s_0 to s_1 on input a. **11.** Use an inductive proof. If the regular expression for A is $\emptyset$, λ, or x, the result is trivial. Otherwise, suppose the regular expression for A is $\mathbf{BC}$. Then $A = BC$ where B is the set generated by $\mathbf{B}$ and C is the set generated by $\mathbf{C}$. By the inductive hypothesis there are regular expressions $\mathbf{B}'$ and $\mathbf{C}'$ that generate B^R and C^R, respectively. Because $A^R = (BC)^R = C^R B^R$, $\mathbf{C}'\mathbf{B}'$ is a regular expression for A^R. If the regular expression for A is $\mathbf{B} \cup \mathbf{C}$, then the regular expression for A is $\mathbf{B}' \cup \mathbf{C}'$ because $(B \cup C)^R = (B^R) \cup (C^R)$. Finally, if the regular expression for A is $\mathbf{B}^*$, then it is easy to see that $(\mathbf{B}')^*$ is a regular expression for A^R.

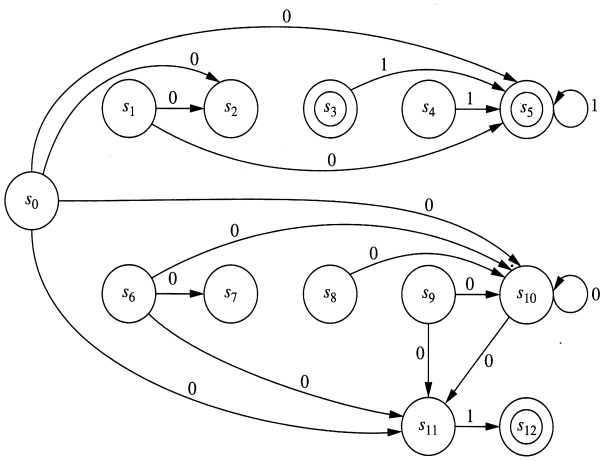
13. a)


Start
s_0
s_1
s_2
s_3
s_4
s_5
0
1

b)


Start
s_0
s_1
s_2
s_3
s_4
s_5
s_6
s_7
0
1

c)


s_0
s_1
s_2
s_3
s_4
s_5
s_6
s_7
s_8
s_9
s_{10}
s_{11}
s_{12}
0
1

15. $S \to 0A$, $S \to 1B$, $S \to 0$, $A \to 0B$, $A \to 1B$, $B \to 0B$, $B \to 1B$ **17.** $S \to 0C$, $S \to 1A$, $S \to 1$, $A \to 1A$, $A \to 0C$, $A \to 1$, $B \to 0B$, $B \to 1B$, $B \to 0$, $B \to 1$, $C \to 0C$, $C \to 1B$, $C \to 1$. **19.** This follows because input that leads to a final state in the automaton corresponds uniquely to a derivation in the grammar. **21.** The "only if" part is clear because I is finite. For the "if" part let the states be $s_{i_0}, s_{i_1}, s_{i_2}, \ldots, s_{i_n}$, where $n = l(x)$. Because $n \geq |S|$, some state is repeated by the pigeonhole principle. Let y be the part of x that causes the loop, so that $x = uyv$ and y sends s_j to s_j, for some j. Then $uy^kv \in L(M)$ for all k. Hence, $L(M)$ is infinite. **23.** Suppose that $L = \{0^{2n}1^n\}$ were regular. Let S be the set of states of a finite-state machine recognizing this set. Let $z = 0^{2n}1^n$ where $3n \geq |S|$. Then by the pumping lemma, $z = 0^{2n}1^n = uvw$, $l(v) \geq 1$, and $uv^iw \in \{0^{2n}1^n \mid n \geq 0\}$. Obviously v cannot contain both 0 and 1, because v^2 would then contain 10. So v is all 0s or all 1s, and hence, uv^2w contains too many 0s or too many 1s, so it is not in L. This contradiction shows that L is not regular. **25.** Suppose that the set of palindromes over $\{0, 1\}$ were regular. Let S be the set of states of a finite-state machine recognizing this set. Let $z = 0^n10^n$, where $n > |S|$. Apply the pumping lemma to get $uv^iw \in L$ for all nonnegative integers i where $l(v) \geq 1$, and $l(uv) \leq |S|$, and $z = 0^n10^n = uvw$. Then v must be a string of 0s (because $|n| > |S|$), so uv^2w is not a palindrome. Hence, the set of palindromes is not regular. **27.** Let $z = 1$; then $111 \notin L$ but $101 \in L$, so 11 and 10 are distinguishable. For the second question, the only way for $1z$ to be in L is for z to end with 01, and that is also the only way for $11z$ to be in L, so 1 and 11 are indistinguishable. **29.** This follows immediately from Exercise 28, because the n distinguishable strings must drive the machine from the start state to n different states. **31.** Any two distinct strings of the same length are distinguishable with respect to the language P of all palindromes, because if x and y are distinct strings of length n, then $xx^R \in P$ but $yx^R \notin P$. Because there are 2^n different strings of length n, Exercise 29 tells us that any deterministic finite-state automaton for recognizing palindromes must have at least 2^n states. Because n is arbitrary, this is impossible.

Section 12.5

1. a) The nonblank portion of the tape contains the string 1111 when the machine halts. **b)** The nonblank portion of the tape contains the string 011 when the machine halts. **c)** The nonblank portion of the tape contains the string 00001 when the machine halts. **d)** The nonblank portion of the tape contains the string 00 when the machine halts. **3. a)** The machine halts (and accepts) at the blank following the input, having changed the tape from 11 to 01. **b)** The machine changes every other occurrence of a 1, if any, starting with the first, to a 0, and otherwise leaves the string unchanged; it halts (and accepts) when it comes to the end of the string. **5. a)** Halts with 01 on the tape, and does not accept **b)** The first 1 (if any) is changed to a 0 and the others are left alone. The input is not accepted.

7. $(s_0, 0, s_1, 1, R)$, $(s_0, 1, s_0, 1, R)$ **9.** $(s_0, 0, s_0, 0, R)$,$(s_0, 1, s_1, 1, R)$,$(s_1, 0, s_1, 0, R)$,$(s_1, 1, s_1, 0, R)$ **11.** $(s_0, 0, s_1, 0, R)$, $(s_0, 1, s_0, 0, R)$, $(s_1, 0, s_1, 0, R)$, $(s_1, 1, s_0, 0, R)$, (s_1, B, s_2, B, R) **13.** $(s_0, 0, s_0, 0, R)$, $(s_0, 1, s_1, 1, R)$, $(s_1, 0, s_1, 0, R)$, $(s_1, 1, s_0, 1, R)$, (s_0, B, s_2, B, R) **15.** If the input string is blank or starts with a 1 the machine halts in nonfinal state s_0. Otherwise, the initial 0 is changed to an M and the machine skips past all the intervening 0s and 1s until it either comes to the end of the input string or else comes to an M. At this point, it backs up one square and enters state s_2. Because the acceptable strings must have a 1 at the right for each 0 at the left, there must be a 1 here if the string is acceptable. Therefore, the only transition out of s_2 occurs when this square contains a 1. If it does, the machine replaces it with an M and makes its way back to the left; if it does not, the machine halts in nonfinal state s_2. On its way back, it stays in s_3 as long as it sees 1s, then stays in s_4 as long as it sees 0s. Eventually either it encouters a 1 while in s_4 at which point it halts without accepting or else it reaches the rightmost M that had been written over a 0 at the start of the string. If it is in s_3 when this happens, then there are no more 0s in the string, so it had better be the case that there are no more 1s either; this is accomplished by the transitions (s_3, M, s_5, M, R) and (s_5, M, s_6, M, R), and s_6 is a final state. Otherwise the machine halts in nonfinal state s_5. If it is in s_4 when this M is encountered, things start all over again, except now the string will have had its leftmost remaining 0 and its rightmost remaining 1 replaced by Ms. So the machine moves, staying in state s_4, to the leftmost remaining 0 and goes back into state s_0 to repeat the process.

17. (s_0, B, s_9, B, L), $(s_0, 0, s_1, 0, L)$, (s_1, B, s_2, E, R), (s_2, M, s_2, M, R), $(s_2, 0, s_3, M, R)$, $(s_3, 0, s_3, 0, R)$, (s_3, M, s_3, M, R), $(s_3, 1, s_4, M, R)$, $(s_4, 1, s_4, 1, R)$, (s_4, M, s_4, M, R), $(s_4, 2, s_5, M, R)$, $(s_5, 2, s_5, 2, R)$, (s_5, B, s_6, B, L), (s_6, M, s_8, M, L), $(s_6, 2, s_7, 2, L)$, $(s_7, 0, s_7, 0, L)$, $(s_7, 1, s_7, 1, L)$, $(s_7, 2, s_7, 2, L)$, (s_7, M, s_7, M, L), (s_7, E, s_2, E, R), (s_8, M, s_8, M, L), (s_8, E, s_9, E, L) where M and E are markers, with E marking the left end of the input

19. $(s_0, 1, s_1, B, R)$, $(s_1, 1, s_2, B, R)$, $(s_2, 1, s_3, B, R)$, $(s_3, 1, s_4, 1, R)$, $(s_1, B, s_4, 1, R)$, $(s_2, B, s_4, 1, R)$, $(s_3, B, s_4, 1, R)$

21. $(s_0, 1, s_1, B, R)$, $(s_1, 1, s_2, B, R)$, (s_1, B, s_6, B, R), $(s_2, 1, s_3, B, R)$, (s_2, B, s_6, B, R), $(s_3, 1, s_4, B, R)$, (s_3, B, s_6, B, R), $(s_4, 1, s_5, B, R)$, (s_4, B, s_6, B, R), $(s_6, B, s_{10}, 1, R)$, $(s_5, 1, s_5, B, R)$, $(s_5, B, s_7, 1, R)$, $(s_7, B, s_8, 1, R)$, $(s_8, B, s_9, 1, R)$, $(s_9, B, s_{10}, 1, R)$

23. $(s_0, 1, s_0, 1, R)$, (s_0, B, s_1, B, L), $(s_1, 1, s_2, 0, L)$, $(s_2, 0, s_2, 0, L)$, $(s_2, 1, s_3, 0, R)$, (s_2, B, s_6, B, R), $(s_3, 0, s_3, 0, R)$, $(s_3, 1, s_3, 1, R)$, $(s_3, B, s_4, 1, R)$, $(s_4, B, s_5, 1, L)$, $(s_5, 1, s_5, 1, L)$, $(s_5, 0, s_2, 0, L)$, $(s_6, 0, s_6, 1, R)$, $(s_6, 1, s_7, 1, R)$, (s_6, B, s_7, B, R)

25. $(s_0, 0, s_0, 0, R)$, $(s_0, *, s_5, B, R)$, $(s_3, *, s_3, *, L)$, $(s_3, 0, s_3, 0, L)$, $(s_3, 1, s_3, 1, L)$, (s_3, B, s_0, B, R),

$(s_5, 1, s_5, B, R)$, $(s_5, 0, s_5, B, R)$, (s_5, B, s_6, B, L), (s_6, B, s_6, B, L), $(s_6, 0, s_7, 1, L)$, $(s_7, 0, s_7, 1, L)$, $(s_0, 1, s_1, 0, R)$, $(s_1, 1, s_1, 1, R)$, $(s_1, *, s_2, *, R)$, $(s_2, 0, s_2, 0, R)$, $(s_2, 1, s_3, 0, L)$, (s_2, B, s_4, B, L), $(s_4, 0, s_4, 1, L)$, $(s_4, *, s_8, B, L)$, $(s_8, 0, s_8, B, L)$, $(s_8, 1, s_8, B, L)$

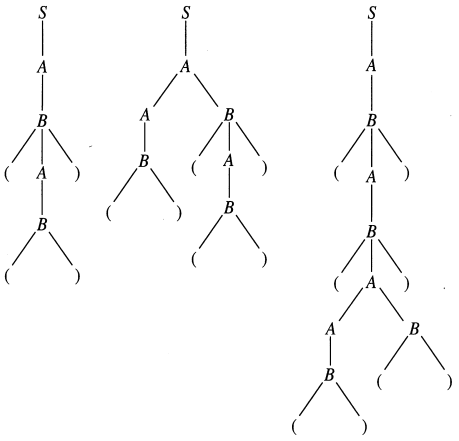
27. Suppose that s_m is the only halt state for the Turing machine in Exercise 22, where m is the largest state number, and suppose that we have designed that machine so that when the machine halts the tape head is reading the leftmost 1 of the answer. Renumber each state in the machine for Exercise 18 by adding m to each subscript, and take the union of the two sets of five-tuples.

29. a) No **b)** Yes **c)** Yes **d)** Yes **31.** $(s_0, B, s_1, 1, L)$, $(s_0, 1, s_1, 1, R)$, $(s_1, B, s_0, 1, R)$

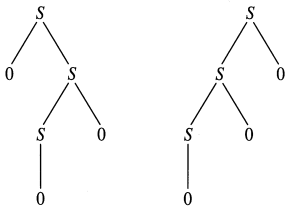
Supplementary Exercises

1. a) $S \to 00S111, S \to \lambda$ **b)** $S \to AABS, AB \to BA, BA \to AB, A \to 0, B \to 1, S \to \lambda$ **c)** $S \to ET, T \to 0TA, T \to 1TB, T \to \lambda, 0A \to A0, 1A \to A1, 0B \to B0, 1B \to B1, EA \to E0, EB \to E1, E \to \lambda$

3.

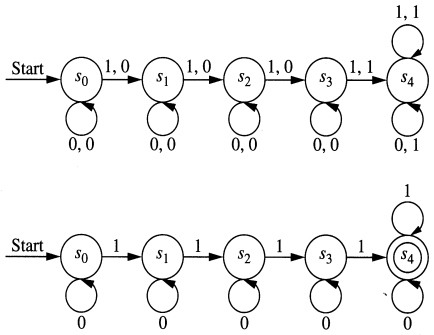


5.

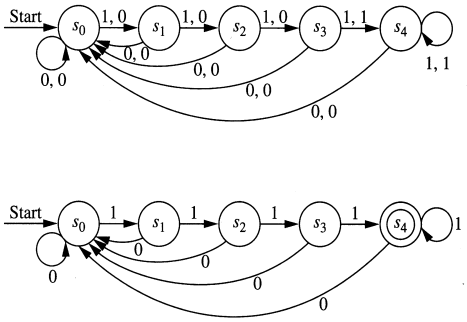


7. No, take $A = \{1, 10\}$ and $B = \{0, 00\}$. **9.** No, take $A = \{00, 000, 00000\}$ and $B = \{00, 000\}$. **11. a)** 1 **b)** 1 **c)** 2 **d)** 3 **e)** 2 **f)** 4

13.

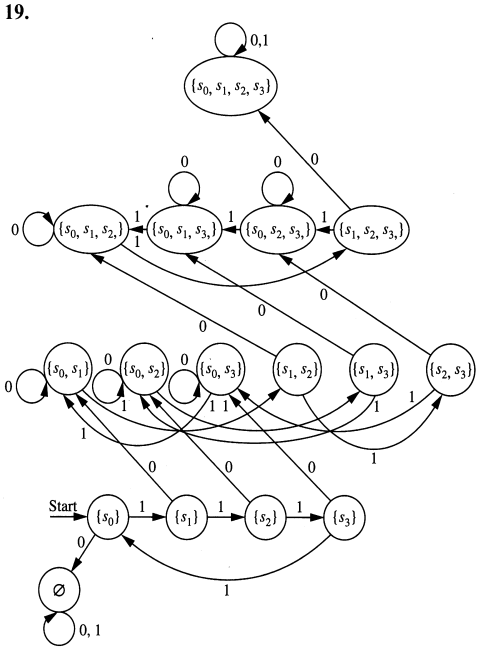


15.

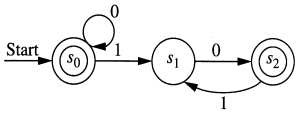


17. a) $n^{nk+1}m^{nk}$ **b)** $n^{nk+1}m^{n}$

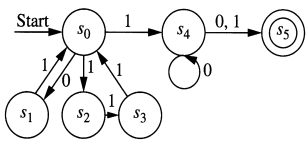
19.



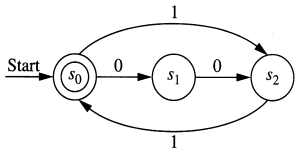
21. a)



b)

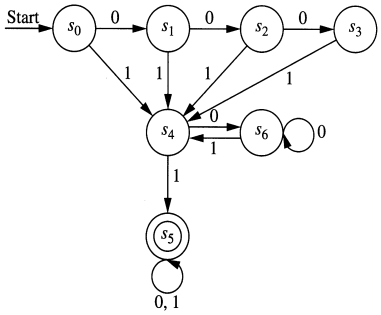


c)

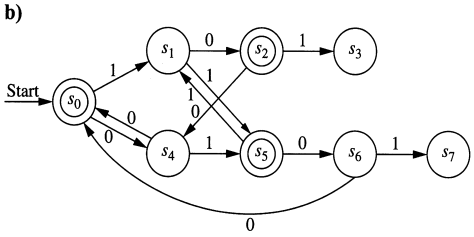


23. Construct the deterministic finite automaton for A with states S and final states F. For $\overline{A}$ use the same automaton but with final states $S - F$.

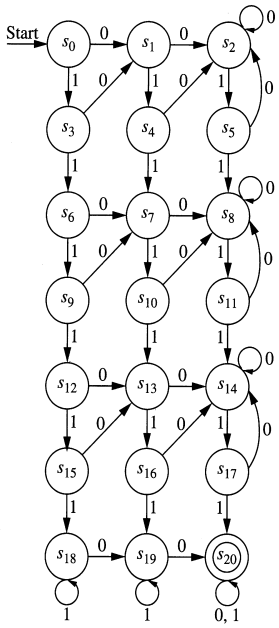
25. a)



b)



c)



27. Suppose that $L = \{1^p \mid p \text{ is prime}\}$ is regular, and let S be the set of states in a finite-state automaton recognizing L. Let $z = 1^p$ where p is a prime with $p > |S|$ (such a prime exists because there are infinitely many primes). By the pumping lemma it must be possible to write $z = uvw$ with $l(uv) \leq |S|$, $l(v) \geq 1$, and for all nonnegative integers i, $uv^iw \in L$. Because z is a string of all 1s, $u = 1^a$, $v = 1^b$, and $w = 1^c$, where $a + b + c = p$, $a + b \leq n$, and $b \geq 1$. This means that $uv^iw = 1^a 1^{bi} 1^c = 1^{(a+b+c)+b(i-1)} = 1^{p+b(i-1)}$. Now take $i = p + 1$. Then $uv^iw = 1^{p(1+b)}$. Because $p(1 + b)$ is not prime, $uv^iw \notin L$, which is a contradiction. **29.** $(s_0, *, s_5, B, L)$, $(s_0, 0, s_0, 0, R)$, $(s_0, 1, s_1, 0, R)$, $(s_1, *, s_2, *, R)$, $(s_1, 1, s_1, 1, R)$, $(s_2, 0, s_2, 0, R)$, $(s_2, 1, s_3, 0, L)$, (s_2, B, s_4, B, L), $(s_3, *, s_3, *, L)$, $(s_3, 0, s_3, 0, L)$, $(s_3, 1, s_3, 1, L)$, (s_3, B, s_0, B, R), $(s_4, *, s_8, B, L)$, $(s_4, 0, s_4, B, L)$, $(s_5, 0, s_5, B, L)$, (s_5, B, s_6, B, R), $(s_6, 0, s_7, 1, R)$, (s_6, B, s_6, B, R), $(s_7, 0, s_7, 1, R)$, $(s_7, 1, s_7, 1, R)$, $(s_8, 0, s_8, 1, L)$, $(s_8, 1, s_8, 1, L)$

APPENDIXES

Appendix 1

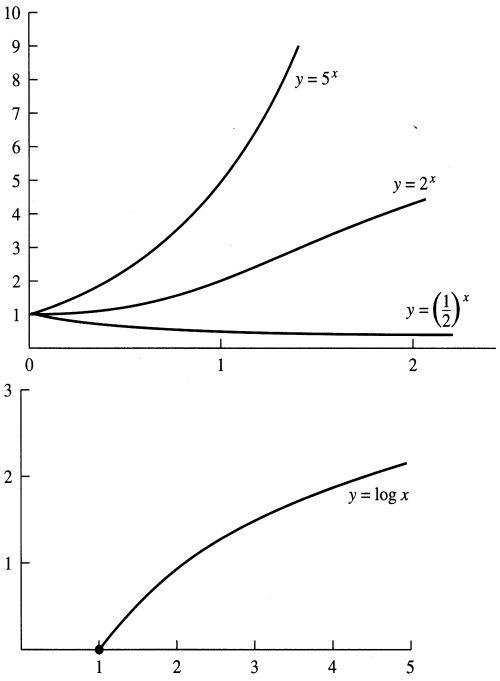
1. Suppose that $1'$ is also a multiplicative identity for the real numbers. Then, by definition, we have both $1 \cdot 1' = 1$ and $1 \cdot 1' = 1'$, so $1' = 1$. **3.** For the first part, it suffices to show that $[(-x) \cdot y] + (x \cdot y) = 0$, because Theorem 2 guarantees

that additive inverses are unique. Thus $[(-x)\cdot y]+(x\cdot y)=(-x+x)\cdot y$ (by the distributive law) $=0\cdot y$ (by the inverse law) $=y\cdot 0$ (by the commutative law) $=0$ (by Theorem 5). The second part is almost identical. **5.** It suffices to show that $[(-x)\cdot(-y)]+[-(x\cdot y)]=0$, because Theorem 2 guarantees that additive inverses are unique: $[(-x)\cdot(-y)]+[-(x\cdot y)]=[(-x)\cdot(-y)]+[(-x)\cdot y]$ (by Exercise 3) $=(-x)\cdot[(-y)+y]$ (by the distributive law) $=(-x)\cdot 0$ (by the inverse law) $=0$ (by Theorem 5). **7.** By definition, $-(-x)$ is the additive inverse of $-x$. But $-x$ is the additive inverse of x, so x is the additive inverse of $-x$. Therefore $-(-x)=x$ by Theorem 2. **9.** It suffices to show that $(-x-y)+(x+y)=0$, because Theorem 2 guarantees that additive inverses are unique: $(-x-y)+(x+y)=[(-x)+(-y)]+(x+y)$ (by definition of subtraction) $=[(-y)+(-x)]+(x+y)$ (by the commutative law) $=(-y)+[(-x)+(x+y)]$ (by the associative law) $=(-y)+[(-x+x)+y]$ (by the associative law) $=(-y)+(0+y)$ (by the inverse law) $=(-y)+y$ (by the identity law) $=0$ (by the inverse law). **11.** By definition of division and uniqueness of multiplicative inverses (Theorem 4) it suffices to prove that $[(w/x)+(y/z)]\cdot(x\cdot z)=w\cdot z+x\cdot y$. But this follows after several steps, using the distributive law, the associative and commutative laws for multiplication, and the definition that division is the same as multiplication by the inverse. **13.** We must show that if $x>0$ and $y>0$, then $x\cdot y>0$. By the multiplicative compatibility law, the commutative law, and Theorem 5, we have $x\cdot y>0\cdot y=0$. **15.** First note that if $z<0$, then $-z>0$ (add $-z$ to both sides of the hypothesis). Now given $x>y$ and $-z>0$, we have $x\cdot(-z)>y\cdot(-z)$ by the multiplicative compatibility law. But by Exercise 3 this is equivalent to $-(x\cdot z)>-(y\cdot z)$. Then add $x\cdot z$ and $y\cdot z$ to both sides and apply the various laws in the obvious ways to yield $x\cdot z<y\cdot z$. **17.** The additive compatibility law tells us that $w+y<x+y$ and (together with the commutative law) that $x+y<x+z$. By the transitivity law, this gives the desired conclusion. **19.** By Theorem 8, applied to $1/x$ in place of x, there is an integer n (necessarily positive, because $1/x$ is positive) such that $n>1/x$. By the multiplicative compatibility law, this means that $n\cdot x>1$. **21.** We must show that if $(w,x)\sim(w',x')$ and $(y,z)\sim(y',z')$, then $(w+y,x+z)\sim(w'+y',x'+z')$ and that $(w\cdot y+x\cdot z,x\cdot y+w\cdot z)\sim(w'\cdot y'+x'\cdot z',x'\cdot y'+w'\cdot z')$. Thus we are given that $w+x'=x+w'$ and that $y+z'=z+y'$, and we want to show that $w+y+x'+z'=x+z+w'+y'$ and that $w\cdot y+x\cdot z+x'\cdot y'+w'\cdot z'=x\cdot y+w\cdot z+w'\cdot y'+x'\cdot z'$. For the first of the desired conclusions, add the two given equations. For the second, rewrite the given equations as $w-x=w'-x'$ and $y-z=y'-z'$, multiply them, and do the algebra.

Appendix 2

1. a) 2^3 **b)** 2^6 **c)** 2^4 **3. a)** $2y$ **b)** $2y/3$ **c)** $y/2$

5.



Appendix 3

1. After the first block is executed, a has been assigned the original value of b and b has been assigned the original value of c, whereas after the second block is executed, b is assigned the original value of c and a the original value of c as well. **3.** The following **while** construction does the same thing.

i := *initial value*
while *i* ≤ *final value*
begin
  statement
  i := *i* + 1
end