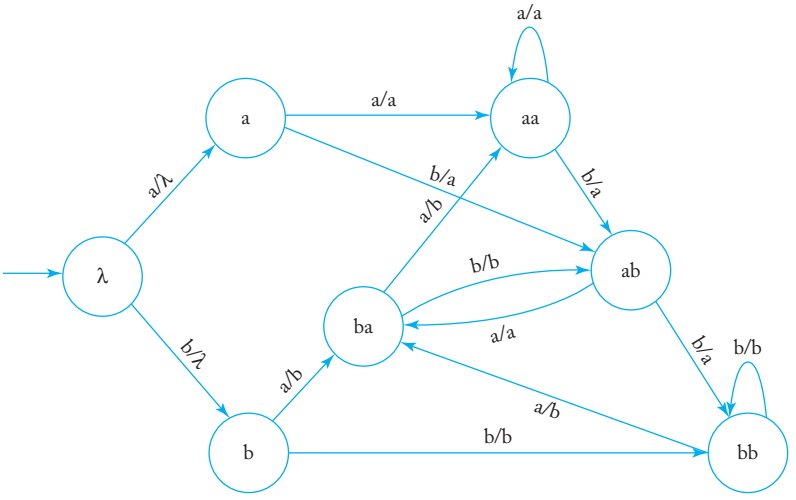


(b) For an n-unit delay transducer, we need to remember the input for n time periods. We label the states mnemonically, with $a_1a_2...a_n$, where $a_i \in \Sigma$. Since there are $|\Sigma|$ choices for each position, there must be at least $|\Sigma|^n$ states.

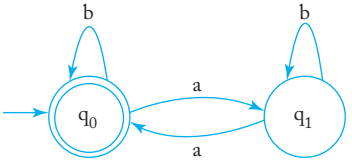
Chapter 2

Section 2.1

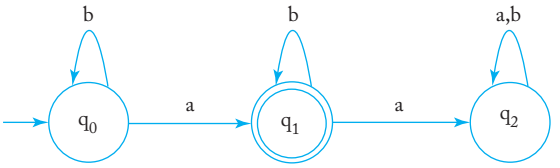
2.

$$\begin{aligned} \delta(\lambda, 1) &= \lambda, & \delta(\lambda, 0) &= 0, \\ \delta(0, 1) &= \lambda, & \delta(0, 0) &= 00, \\ \delta(00, 0) &= 00, & \delta(00, 1) &= 001, \\ \delta(001, 0) &= 001, & \delta(001, 1) &= 001. \end{aligned}$$

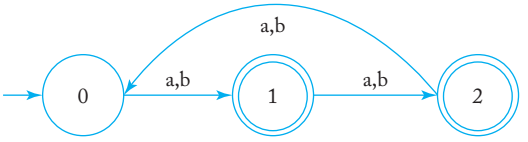
3. (c)



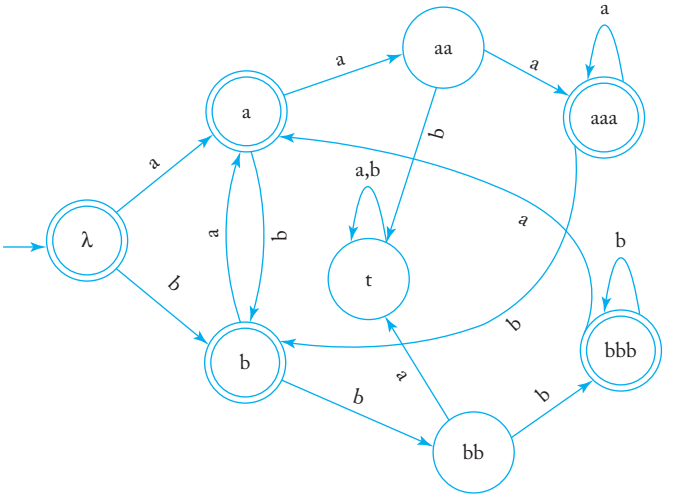
4. (a)



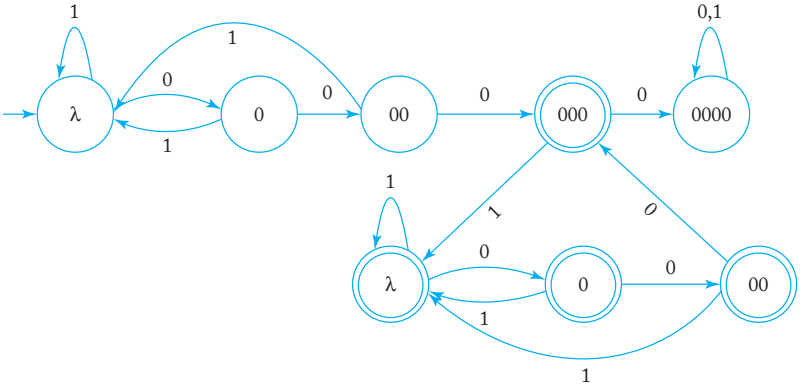
7. (a) Use states labeled with $|w|$ mod 3.



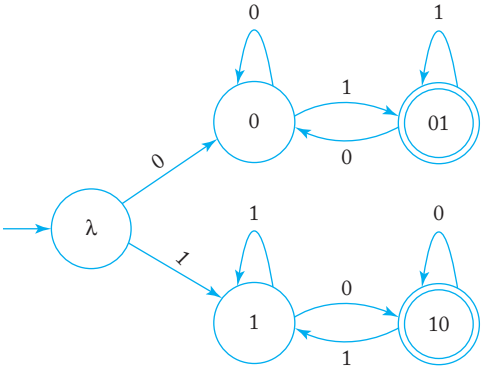
8. (a) Use appropriate a's and b's to label the states.



11. (b)



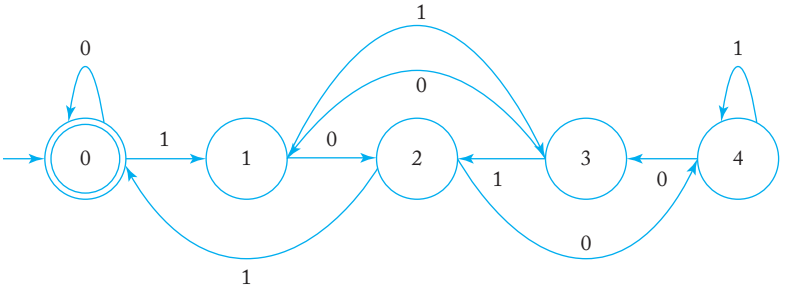
(c) We need to remember the leftmost symbol and put the dfa into a final state whenever a different symbol is encountered



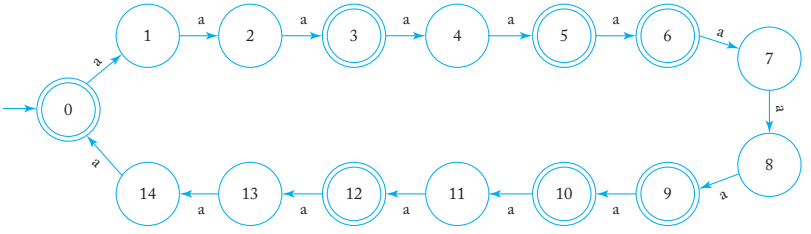
12. The trick is to label the states with the value (mod 5) of the partial bit string and take care of the next bit by

$$(2n+1) \bmod 5 = (2n \bmod 5 + 1) \bmod 5.$$

This leads to the solution shown below.



16. Since the least common multiplier of 3 and 5 is 15, we need 15 states to construct such a dfa.



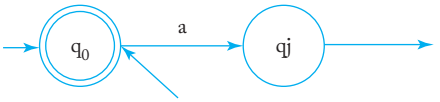
19. The catch is that if λ is in the language, the initial state must also be a final state. But we cannot just make it non-final as other input may reach this state. To get around this difficulty, we modify the original dfa by creating a new initial state p_0 and new transitions

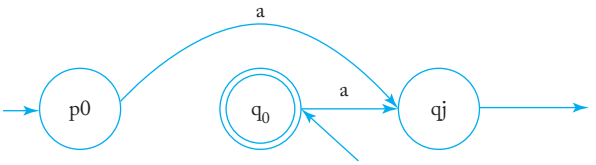
$$\delta(p_0, a) = q_j$$

for all original transitions

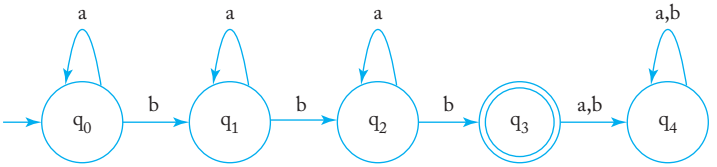
$$\delta(q_0, a) = q_j.$$

A graphical representation of the process is given below.



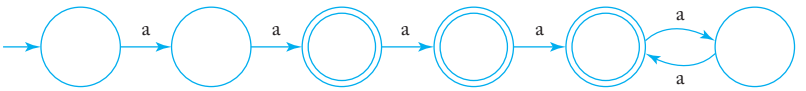


22. $L^3 = \{a^n ba^m ba^p b : n, m, p \geq 0\}$. A dfa that accepts L^3 is given by



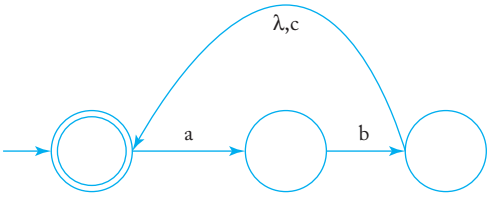
Section 2.2

3. The nfa in Figure 2.8 accepts the language $\{a^n : n = 3 \text{ or } n \text{ is even}\}$. A dfa for this language is



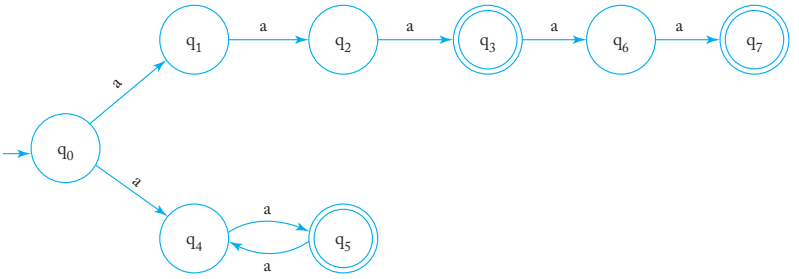
6. $\delta^*(q_0, a) = \{q_0, q_1, q_2\}$ and $\delta^*(q_1, \lambda) = \{q_0, q_1, q_2\}$.

9.

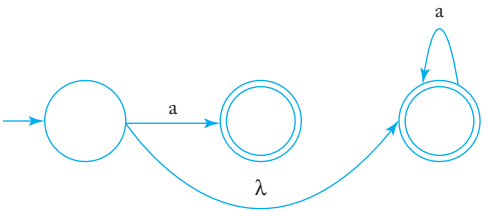


13. 01001 and 000 are the only two strings accepted.

15. Add two states after the nfa accepts a^3 with both new edges labeled a.



17. Remove the λ-edge from the graph below.

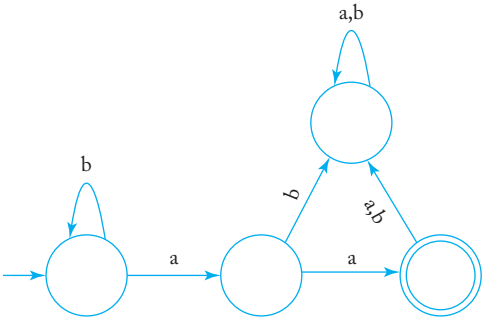


19. Introduce a new initial state p_0. Then add a transition

$$\delta(p_0, \lambda) = Q_0.$$

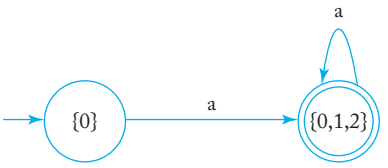
Next, remove starting state status from Q_0. It is straightforward to see that the new nfa is equivalent to the original one.

22. Introduce a non-accepting trap state and make all undefined transitions to this new state.



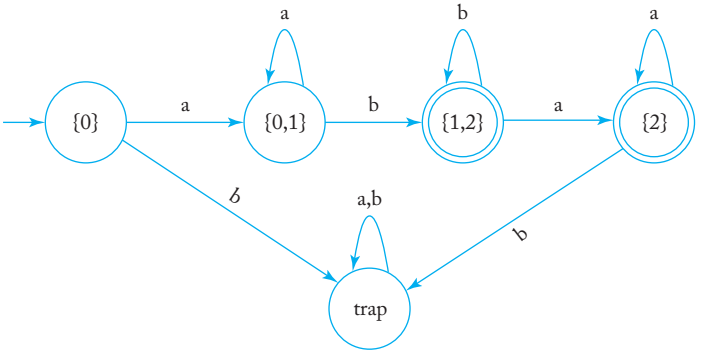
Section 2.3

1.



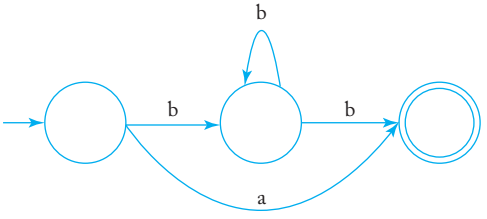
For a simple answer, note that the language is $\{a^n : n > 1\}$.

3.



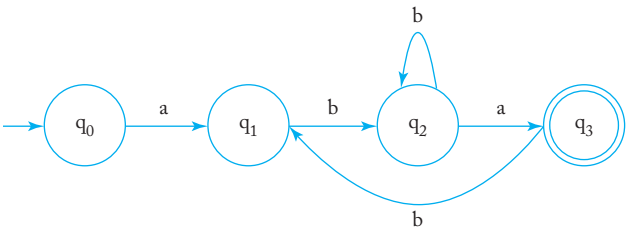
7. Yes, it is true. By definition, $w \in L$ if and only if $\delta^*(q_0, w) \cap F \neq \varnothing$. Consequently, if $\delta^*(q_0, w) \cap F = \varnothing$, then $w \in \bar{L}$.

10.

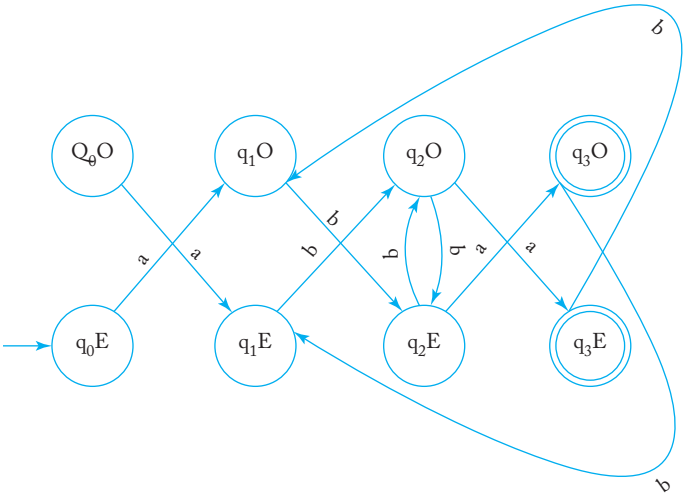


12. Introduce a single initial state, and connect it to previous initial states via λ-transitions. Then convert back to a dfa and note that the construction of Theorem 2.2 retains the single initial state.

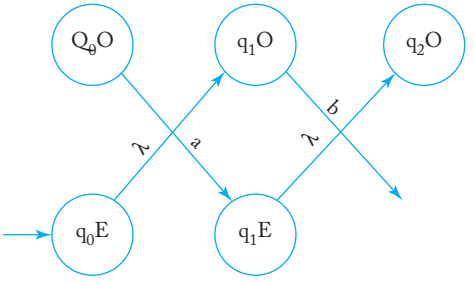
16. The trick is to use a dfa for L and modify it so that it remembers if it has read an even or an odd number of symbols. This can be done by doubling the number of states and adding O or E to the labels. For example, if part of the dfa is



its equivalent becomes



Now replace every transition from an E state to an O state with λ-transitions.



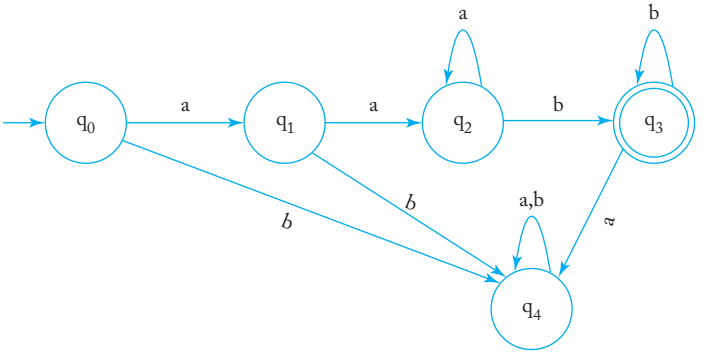
A few examples will convince your that if the original dfa accepts $a_1a_2a_3a_4$, the new automaton will accept $\lambda a_2 \lambda a_4 ...$, and therefore $even\,(L)$.

Section 2.4

1. Using procedure *mark*, we generate the equivalence classes $\{q_0, q_1\}$, $\{q_2\}, \{q_3\}$. Then the procedure *reduce* gives

$$\begin{aligned} \hat{\delta}(01, a) &= 2, & \hat{\delta}(01, b) &= 2, \\ \hat{\delta}(2, a) &= 3, & \hat{\delta}(2, b) &= 3, \\ \hat{\delta}(3, a) &= 3, & \hat{\delta}(3, b) &= 01. \end{aligned}$$

3. (a) The following graph is a five state dfa. It should be easy to see that the dfa accepts the language $L = \{a^n b^m : n \geq 2, m \geq 1\}$.



We claim it is a minimal dfa. Because $q_3 \in F$ and $q_4 \notin F$, so q_3 and q_4 are distinguishable. Next $\delta(q_4, b) = q_4 \notin F$ and $\delta(q_2, b) = q_3 \in F$, so q_2 and q_4 are distinguishable. Similarly, $\delta^*(q_0, ab) = q_4 \notin F$ and $\delta^*(q_1, ab) = q_3 \in F$, so q_0 and q_1 are distinguishable. Continuing this way, we see that all the five states are mutually distinguishable. Therefore,the dfa is minimal.

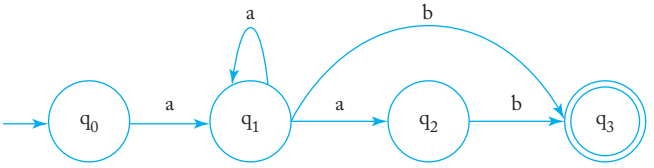
6. Yes, it is true. We argue by contradiction. Assume that $\widehat{M}$ is not minimal. Then we can construct a smaller dfa $\widetilde{M}$ that accepts $\overline{L}$. In $\widetilde{M}$, complement the final state set to give a dfa for L. But this dfa is smaller than M, contradicting the assumption that M is minimal.

9. Assume that q_b and q_c are indistinguishable. Since q_a and q_b are indistinguishable and indistinguishability is an equivalence relation as shown in Exercise 7, so q_a and q_c must be indistinguishable. This contradicts the assumption.

Chapter 3

Section 3.1

3.



5. Yes, because $\left((0+1)(0+1)^*\right)^*$ denotes any string of 0's and 1's.

7. A regular expression is $aaaa^*(bb)^*b$.

9. (a) Separate into cases $m = 0, 1, 2, 3, 4$. Generate 3 or more a's, followed by the requisite number of b's. A regular expression for L_1 is $aaaa^*\,(\lambda + b + bb + bbb + bbbb)$

16. Enumerate all cases with $|v| = 2$ to get

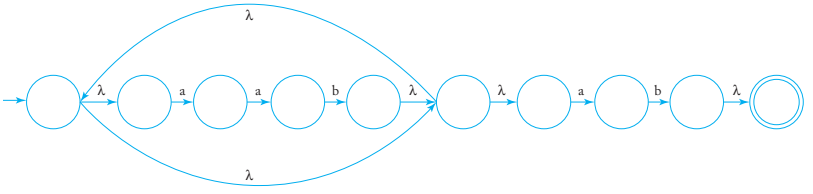
$$aa\,(a+b)^*\,aa + ab\,(a+b)^*\,ab + ba\,(a+b)^*\,ba + bb\,(a+b)^*\,bb.$$

19. (a) A regular expression is $(b+c)^*a(b+c)^*a(b+c)^*$.

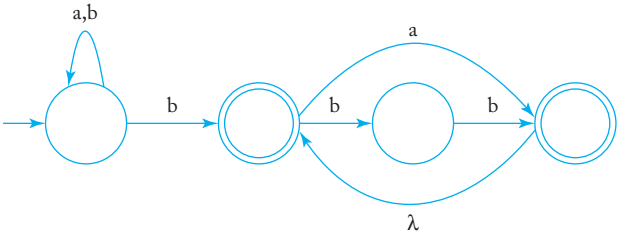
26. The graph for $(rd)^*$ is

Section 3.2

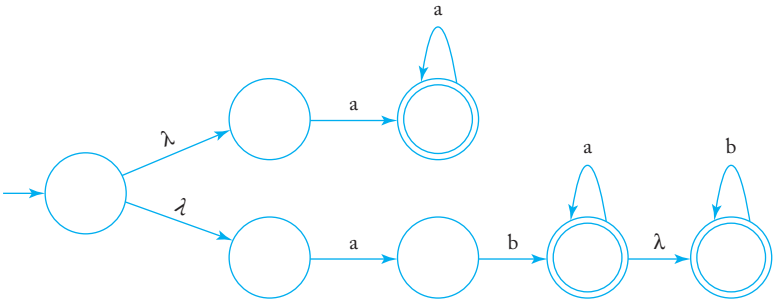
2.



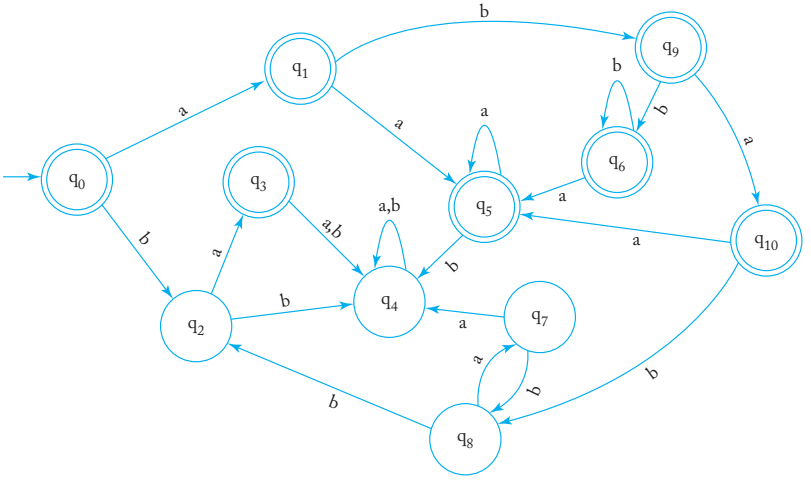
5. This can be solved from first principles without going through the regular expression to nfa construction. The latter will, of course, work but gives a more complicated answer.



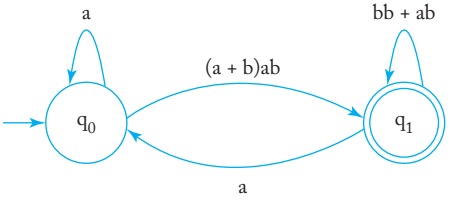
6. (a)



7. (a)



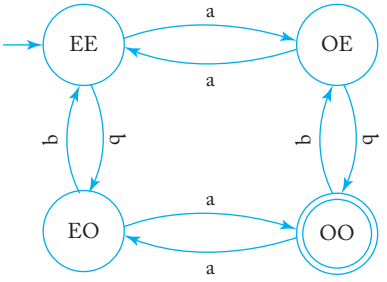
10. (a) Removing the middle vertex gives



(b) By Equation (3.1), the language accepted is $L(r)$, where

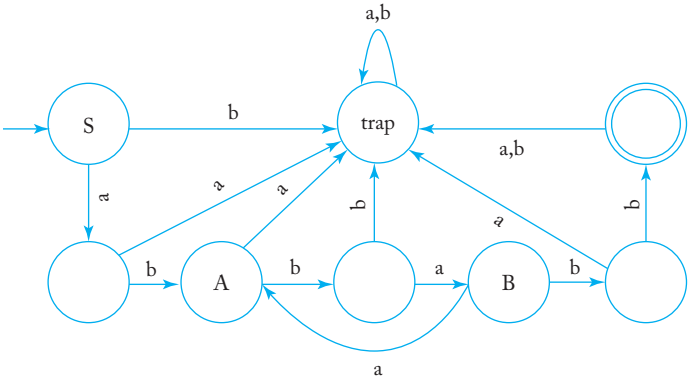
$$r = a^*(a+b)ab(bb+ab+aa^*(a+b)ab)^*.$$

15. (a) Label the vertices with EE to denote an even number of a's and an even number of b's, with OE to denote an even number of a's but an odd number of b's, and so on. Then we get the generalized transition graph



Section 3.3

1.



4. The language in Exercise 1 is $L = \{abba(aba)^*bb\}$. A left-linear grammar can be constructed by taking the regular expression in reverse order.

$$S \rightarrow Abb,$$
$$A \rightarrow Aaba|B,$$
$$B \rightarrow abba.$$

6. It is straightforward to get a grammar for $L(aaab^*ab)$. The closure of this language requires only a minor modification.

$$\begin{aligned} S &\rightarrow aaaA|\lambda, \\ A &\rightarrow bA|B, \\ B &\rightarrow ab|abS. \end{aligned}$$

8. We can show by induction that if w is a sentential form derived with G, then w^R can be derived in the same number of steps by $\widehat{G}$.

Because w is created with left linear derivations, it must have the form $w = Aw_1$, with $A \in V$ and $w_1 \in T^*$. By the inductive assumption $w^R = w_1^R A$ can be derived via $\widehat{G}$. If we now apply $a \rightarrow Bv$, then

$$w \Rightarrow Bvw_1.$$

But $\widehat{G}$ contains the rule $A \rightarrow v^R B$, so we can make the derivation

$$\begin{aligned} w^R &\rightarrow w_1^R v^R B \\ &= (Bvw_1)^R \end{aligned}$$

completing the inductive step.

12. Draw an nfa using mnemonic labels. Then use the construction in Theorem 3.4. An answer is

$$\begin{aligned} EE &\rightarrow aOE|bEO, \\ OE &\rightarrow aEE|bOO|\lambda, \\ OO &\rightarrow bEO, \\ EO &\rightarrow bOO|\lambda. \end{aligned}$$

14. (b) Using mnemonic labels to denote even-odd pairs of a and b we get a dfa for the problem. From the dfa, we get the grammar

$$\begin{aligned} EE &\rightarrow aOE|bEO|\lambda, \\ OE &\rightarrow aEE|bOO, \\ EO &\rightarrow aOO|bEE, \\ OO &\rightarrow aEO|bOE. \end{aligned}$$

15. For every rule in a right-linear grammar that has more than one terminal, replace terminals successively by variables and new rules. For example, for

$$A \rightarrow a_1 a_2 B,$$

introduce variable C, and new productions

$$\begin{aligned} A &\rightarrow a_1 C, \\ C &\rightarrow a_2 B. \end{aligned}$$

Chapter 4

Section 4.1

2. A regular expression for $(L_1 \bigcup L_2)^* L_2$ is

$$((ab^*aa) + (a^*bba^*))^*(a^*bba^*).$$

4. By DeMorgan's law, $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$. So if the language family is closed under complementation and union, $L_1 \cap L_2$ must be in the family.

9. By induction. From Theorem 4.1, we know that the union of two regular languages is a regular language. Now suppose it is true for any $k \leq n-1$,

$$\bigcup_{i=\{1,2,\ldots,k\}} L_i$$

is a regular language. Then

$$L_U = \bigcup_{i=\{1,2,\ldots,n\}} L_i = (\bigcup_{i=\{1,2,\ldots,n-1\}} L_i)) \cup L_n$$

is simply a union of two regular languages. Therefore L_U is a regular language.

A similar argument for intersection of regular languages shows L_I is a regular language. So regular languages are closed under finite intersection.

11. Notice that

$$nor\,(L_1, L_2) = \overline{L_1 \cup L_2}.$$

The result then follows from closure under intersection and complementation.

16. The answer is yes. It can be obtained by starting from the set identity

$$L_2 = \left((L_1 \cup L_2) \cap \overline{L_1}\right) \cup (L_1 \cap L_2)\,.$$

The key observation is that since L_1 is finite, $L_1 \cap L_2$ is finite and therefore regular for all L_2. The rest then follows easily from the known closures under union and complementation.

17. If r is a regular expression for L, and s is a regular expression for Σ, then rss is a regular expression for L_1 and L_1 is regular.

20. Use $L_1 = \Sigma^*$. Then, for any L_2, $L_1 \cup L_2 = \Sigma^*$, which is regular. The given statement would then imply that any L_2 is regular.

22. We can use the following construction. Find all states P such that there is a path from the initial vertex to some element of P, and from that element to a final state. Then make every element of P a final state.

27. Suppose $G_1 = (V_1, T, S_1, P_1)$ and $G_2 = (V_2, T, S_2, P_2)$. Without loss of generality, we can assume that V_1 and V_2 are disjoint. Combine the two grammars and

(a) Make S the new start symbol and add productions $S \to S_1|S_2$.
(b) In P_1, replace every production of the form $A \to x$, with $A \in V_1$ and $x \in T^*$, by $A \to xS_2$.
(c) In P_1, replace every production of the form $A \to x$, with $A \in V_1$, and $x \in T^*$, by $A \to xS_1, S_1 \to \lambda$.

Section 4.2

1. Use the dfa M for L, interchange initial and final states, and reverse edges to get an nfa $\widehat{M}$ for L^R. By Theorem 4.5, we can check for $w \in L^R$, or equivalently for $w^R \in L$.

4. Since $L_1 - L_2$ is regular, by Theorem 4.5, there exists such an algorithm.

7. Construct a dfa. Then $\lambda \in L$ if and only if $q_0 \in F$.

10. Since L^* and L are both regular, by Theorem 4.7, we have an algorithm testing for equality of regular languages.

13. If there are no even length strings in L, then

$$L((aa + ab + ba + bb)^*) \cap L = \oslash.$$

Therefore, by Theorem 4.6 the result follows

17. Construct a dfa M_1 for $L(G_1)$, and a dfa M_2 for $L(G_2)$. Then use the construction in Theorem 3.1 to get a nfa for $L = L(G_1) \cup L(G_2)$. Check equality for L and Σ^*

Section 4.3

1. Suppose L is regular and we are given m, then we pick $w = a^m bc^m$ in L. Now because $|xy|$ cannot be greater than m, it is clear that $y = a^k$ is the only possible choice for some $k \geq 1$. Then the pumped strings are

$$w_i = a^{m+(i-1)k} bc^m \in L,$$

for all $i = 0, 1, ...$ However, $w_2 = a^{m+k} bc^m \notin L$. This contradiction implies that L is not a regular language.

3. Use the closure under homomorphism for regular languages. Take

$$h(a) = a, \quad h(b) = a, \quad h(c) = c, \quad h(d) = c,$$

then

$$\begin{aligned} h(L) &= \{a^{n+k} c^{n+k} : n + k > 0\} \\ &= \{a^i c^i : i > 0\}. \end{aligned}$$

By the argument in Example 4.7, $h(L)$ is not regular. Therefore L is not regular.

5. (a) Suppose L is regular and we are given m, then we pick $w = a^m b^m c^{2m}$ in L. Now because $|xy|$ cannot be greater than m, it is clear that the $y = a^k$ is the only possible choice for some $k \geq 1$. Therefore by the pumping lemma

$$w_i = a^{m+(i-1)k} b^m c^{2m} \in L,$$

for all $i = 0, 1, ...$ However, $w_0 = a^{m-k} b^m c^{2m} \notin L$ because $m - k + m < 2m$. Therefore L is not regular.

(f) Suppose L is regular and m is given. We pick $w = a^m b a^m b$ in L. The string y must then be a^k and the pumped strings will be

$$w_i = a^{m+(i-1)k} b a^m b \in L,$$

for $i = 0, 1, ...$ However, $w_0 = a^{m-k} b a^m b \notin L$. Therefore L is not regular.

6. (a) Suppose L is regular and m is given. Let p be the smallest prime number such that $p \geq m$. Then we pick $w = a^p$ in L. The string y must then be a^k and the pumped strings will be

$$w_i = a^{p+(i-1)k} \in L,$$

for $i = 0, 1, ...$ However, $w_{p+1} = a^{p+pk} = a^{p(1+k)} \notin L$. Therefore L is not regular.

7. (a) Since

$$\begin{aligned} L &= \{a^n b^n : n \geq 1\} \cup \{a^n b^m : n \geq 1, m \geq 1\} \\ &= \{a^n b^m : n \geq 1, m \geq 1\} \end{aligned}$$

it is clearly regular.

12. Suppose L is regular and m is given. We pick $w = a^{m!}$ which is in L. The string y must then be a^k for some $1 \leq k \leq m$ and the pumped strings will be

$$w_i = a^{(m!)+(i-1)k} \in L,$$

for $i = 0, 1, ...$ However, $w_2 = a^{m!+k} \notin L$, because $m! + k \leq m! + m < (m+1)!$. Therefore L is not regular.

18. (a) The language is regular. This is most easily seen by splitting the problem into cases such as $l = 0, k = 0, n > 5$, for which one can easily construct regular expressions.

(e) This language is not regular. Suppose m is given, we pick $w = a^m b^{m+1}$. So, our opponent can only choose $y = a^k$ for some $1 \leq k \leq m$. The pumped strings are $w_i = a^{m+(i-1)k} b^{m+1}$. So, for example, $w_3 = a^{m+2k} b^{m+1}$ violates the required condition since $m + 1 < m + 2k$.

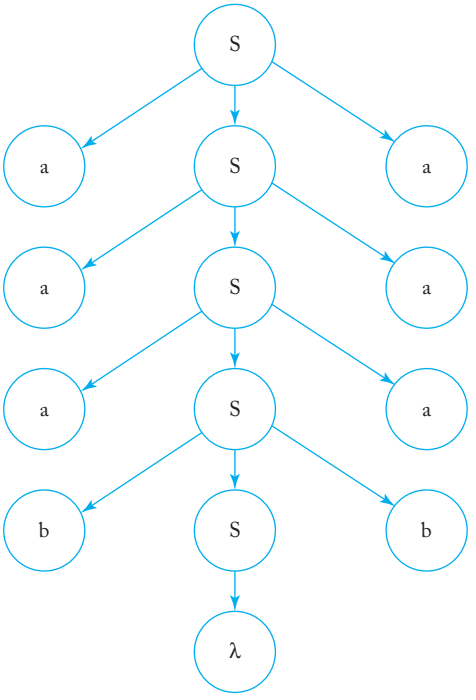
24. Take $L_i = \{a^i b^i\}, i = 0, 1, \ldots$. For each i, L_i is finite and therefore regular, but the union of all the languages is the nonregular language $L = \{a^n b^n : n \geq 0\}$.

Chapter 5

Section 5.1

1. (b) $S \to aAb, \quad A \to aaAbb|\lambda$.

2. A derivation tree is



9. (a) $S \to aSb|A|B, \quad A \to \lambda|a|aa|aaa, \quad B \to bB|b$.

(e) Split into two parts:(i) $n_a(w) > n_b(w)$ and ii) $n_b(w) > n_a(w)$. For part (i), generate an equal number of a's and b's then add more a's. Do the same thing for part (ii) by adding more b's.

$$
\begin{aligned}
S &\to S_1|S_2, \\
S_1 &\to S_1S_1|aS_1b|bS_1a|A, \\
A &\to aA|a, \\
S_2 &\to S_2S_2|aS_2b|bS_2a|B, \\
B &\to bB|b.
\end{aligned}
$$

10. The language generated by the grammar can be described recursively by

$$L = \{w : w = aub \text{ or } w = bua \text{ with } u \in L, \lambda \in L\}.$$

.

12. (a) We split the problem into two parts : (i) $L_1 = \{a^n b^m c^k : n = m\}$ and (ii) $L_2 = \{a^n b^m c^k : m \leq k\}$. Then use $S \to S_1|S_2$, where S_1 derives L_1 and S_2 derives L_2. A grammar is given below.

$$\begin{aligned} S &\to S_1C|AS_2, \\ S_1 &\to aS_1b|\lambda, \\ S_2 &\to bS_2c|C, \\ A &\to aA|\lambda, \\ C &\to cC|\lambda. \end{aligned}$$

(e)

$$S \to aSc|B, \quad B \to bBcc|\lambda.$$

15.

$$\begin{aligned} S &\to aSb|S_1, \\ S_1 &\to aS_1a|bS_1b|\lambda. \end{aligned}$$

16. (a) If S derives L, then $S_2 \to SS$ derives L^2.

26. The grammar has variables $\{S, V, R, T\}$, and terminals

$$T = \{a, b, A, B, C, \to\}$$

with productions

$$\begin{aligned} S &\to V \to R \\ R &\to TR|\lambda \\ V &\to A|B|C, \\ T &\to a|b|A|B|C|\lambda. \end{aligned}$$

Section 5.2

2. For any $w \in L(G)$ with G an s-grammar, it is obvious that there is a unique choice for the leftmost derivation. So every s-grammar is unambiguous.

4.

$$S \to aS_1, \quad S_1 \to aAB, \quad A \to aAB|b, \quad B \to b.$$

8. Here are two leftmost derivations for $w = aaab$.

$$\begin{aligned} S &\Rightarrow aaaB \Rightarrow aaab, \\ S &\Rightarrow AB \Rightarrow AaB \Rightarrow AaaB \Rightarrow aaaB \Rightarrow aaab. \end{aligned}$$

13. From the dfa for a regular language we can get a regular grammar by the method of Theorem 3.4. The grammar is an s-grammar except for $q_f \to \lambda$. But this rule does not create any ambiguity. Since the dfa never has a choice, there is never any choice in the production that can be applied.

15. Yes, for example, $S \to aS_1|ab.\ \ S_1 \to b$. The language $L(ab)$ is finite, so is regular. But $S \Rightarrow ab$ and $S \Rightarrow aS_1 \Rightarrow ab$ are two distinct derivations for ab.

17. The string $abab$ can be derived by either

$$S \Rightarrow aSbS \Rightarrow abS \Rightarrow abaSbS \Rightarrow ababS \Rightarrow abab,$$

or

$$S \Rightarrow aSbS \Rightarrow abSaSbS \Rightarrow abaSbS \Rightarrow ababS \Rightarrow abab.$$

Therefore, the grammar is ambiguous.

Chapter 6

Section 6.1

2. If we eliminate the variable B we get the second grammar.

6. First, we identify the set of variables that can lead to a terminal string. Because S is the only such variable, A and B can be removed and we get

$$S \to aS|\lambda$$

This grammar derives $L(a^*)$.

7. First we notice that every variable except C can lead to a terminal string. Therefore, the grammar becomes

$$\begin{aligned} S &\to a\,|aA|\,B, \\ A &\to aB|\lambda, \\ B &\to Aa, \\ D &\to ddd. \end{aligned}$$

Next we want to eliminate the variables that cannot be reached from the start variable. Clearly, D is useless and we remove it from the production list. The resulting grammar has no useless productions.

$$\begin{aligned} S &\to a\,|aA|\,B, \\ A &\to aB|\lambda, \\ B &\to Aa. \end{aligned}$$

10. The only nullable variable is A, so removing λ-productions gives

$$\begin{aligned} S &\rightarrow aA\,|a|\,aBB, \\ A &\rightarrow aaA|aa, \\ B &\rightarrow bC|bbC, \\ C &\rightarrow B. \end{aligned}$$

$C \rightarrow B$ is the only unit-production and removing it results in

$$\begin{aligned} S &\rightarrow aA\,|a|\,aBB, \\ A &\rightarrow aaA|aa, \\ B &\rightarrow bC|bbC, \\ C &\rightarrow bC|bbC. \end{aligned}$$

Finally, B and C are useless, so we get

$$\begin{aligned} S &\rightarrow aA|a, \\ A &\rightarrow aaA|aa. \end{aligned}$$

The language generated by this grammar is $L\left((aa)^* a\right)$.

17. An example is

$$\begin{aligned} S &\rightarrow aA, \\ A &\rightarrow BB, \\ B &\rightarrow aBb|\lambda. \end{aligned}$$

When we remove λ-productions we get

$$\begin{aligned} S &\rightarrow aA|a, \\ A &\rightarrow BB|B, \\ B &\rightarrow aBb|ab. \end{aligned}$$

21. This rather simple substitution can be justified by a straightforward argument. Show that every string derivable by the first grammar is also derivable by the second, and vice versa.

25. A difficult problem because the result is hard to see intuitively. The proof can be done by showing that crucial sentential forms generated by one grammar can also be generated by the other one. The important step in this is to show that if

$$A \stackrel{*}{\Rightarrow} Ax_k...x_jx_i \Rightarrow y_rx_k...x_jx_i,$$

then with the modified grammar, we can make the derivation

$$A \Rightarrow y_rZ \Rightarrow ... \stackrel{*}{\Rightarrow} y_rx_k...x_jx_i.$$

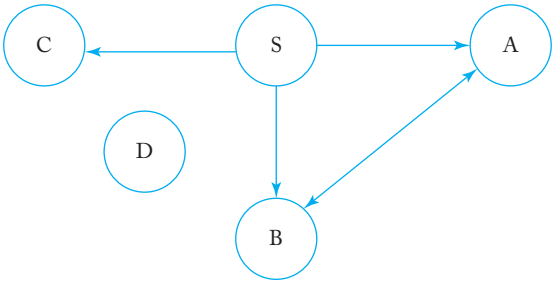
This is actually an important result, dealing with the removal of certain left-recursive productions from the grammar and is needed if one were to discuss the general algorithm for converting a grammar into Greibach normal form.

Section 6.2

3. Eliminate the unit-production first: $S \to aSaaA|abA|bb,\ \ A \to abA|bb$. Then apply Theorem 6.6.

$$\begin{aligned} S &\to V_aD_1|V_aE_1|V_bV_b, \\ D_1 &\to SD_2, \\ D_2 &\to V_aD_3, \\ D_3 &\to V_aA, \\ E_1 &\to V_bA, \\ A &\to V_aE_1|V_bV_b, \\ V_a &\to a, \\ V_b &\to b. \end{aligned}$$

7.



8. Introduce a new variable V_1 with the productions

$$V_1 \to a_2...a_nBb_1b_2...b_m$$

and

$$A \to a_1V_1.$$

Continue this process, introducing V_2 and

$$V_2 \to a_3...a_nBb_1b_2...b_m$$

and so on, until no terminals remain on the left. Then use a similar process to remove terminals on the right.

10. Introducing new variables $V_a \to a$ and $V_b \to b$, we get the Greibach normal form immediately.

$$\begin{aligned} S &\to aSV_b|bSV_a|a|b|aV_b, \\ V_a &\to a, \\ V_b &\to b. \end{aligned}$$

13. Removing the unit-production $A \to B$ in the grammar, we have new production rule $A \to aaA|bAb$. Next substitute the new productions into S, to get an equivalent grammar

$$\begin{aligned} S &\to aaABb|bAbBb|a|b, \\ A &\to aaA|bAb, \\ B &\to bAb. \end{aligned}$$

By introducing productions V_a, V_b, we can convert the grammar into Greibach normal form.

$$\begin{aligned} S &\to aV_aABV_b|bAV_bBV_b|a|b, \\ A &\to aV_aA|bAV_b, \\ B &\to bAV_b, \\ V_a &\to a, \\ V_b &\to b. \end{aligned}$$

Section 6.3

4. First convert the grammar to Chomsky normal form

$$\begin{aligned} S &\to AC|b, \\ C &\to SB, \\ B &\to b, \\ A &\to a. \end{aligned}$$

Then we can compute V_{ij}'s for the string *aaabbbb* from the converted grammar.

$$\begin{aligned} &V_{11} = V_{22} = V_{33} = \{A\}, \quad V_{44} = V_{55} = V_{66} = V_{77} = \{S, B\}, \\ &V_{12} = V_{23} = V_{34} = \varnothing, \quad V_{45} = V_{56} = V_{67} = \{C\}, \\ &V_{13} = V_{24} = V_{46} = V_{57} = \varnothing, \quad V_{35} = \{S\}, \\ &V_{14} = V_{25} = V_{47} = \varnothing, \quad V_{36} = \{C\}, \\ &V_{15} = V_{37} = \varnothing, \quad V_{26} = \{S\}, \\ &V_{16} = \varnothing, \quad V_{27} = \{C\}, \\ &V_{17} = \{S\}. \end{aligned}$$

Since $S \in V_{17}$, the string *aaabbbb* is in the language generated by the given grammar.

Chapter 7

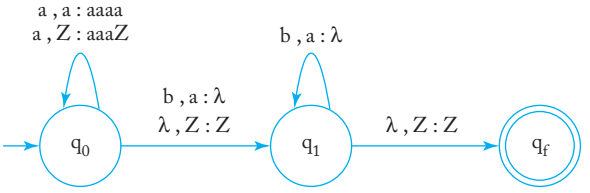
Section 7.1

3. (c) Let p_0 be the initial state of the new pda and q and q' denote the initial states of (a) and (b), respectively. Then we set

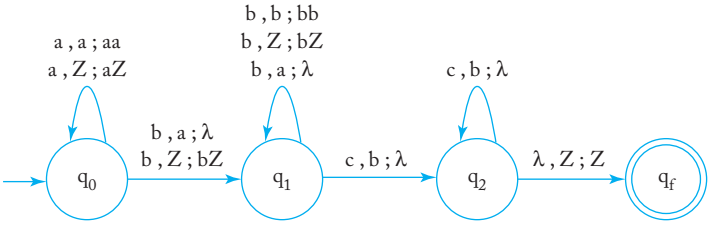
$$\delta(p_0, \lambda, z) = \{(q_0, z), (q_0', z)\}.$$

4. No need for state q_1. Substitute q_0 wherever q_1 occurs. The major difficulty is to argue convincingly that this works.

6. (a)



(d) Start with an initial z in the stack. Put a mark a on the stack when input symbol is an a, consume a mark a when input is a b. When all a's in the stack are consumed, put a mark b on the stack when input is a b. After input becomes c, eliminate a mark on the stack. The string will be accepted if the stack has only z left when the input is completed.



8. At first glance this looks like a simple problem: put w_1 in the stack, then go into a final trap state whenever a mismatch with w_2 is detected. But this is incomplete if the two substrings are of unequal length, for example, if $w_2 = w_1^R v$. A way to solve this is to nondeterministically split the problem into $w_1 = w_2$ and $w_1 \neq w_2$.

18. Here we use internal states to remember symbols to be put on the stack. For example,

$$\delta\,(q_i, a, b) = \{(q_j, cde)\}$$

is replaced by

$$\begin{aligned} \delta\,(q_i, a, b) &= \{(q_{jc}, de)\}, \\ \delta\,(q_{jc}, \lambda, d) &= \{(q_j, cd)\}\,. \end{aligned}$$

Since δ can have only a finite number of elements and each can only add a finite amount of information to the stack, this construction can be carried out for any pda.

Section 7.2

2. Convert the grammar into Greibach normal form.

$$\begin{aligned} S &\to aSSSA|\lambda, \\ A &\to aB, \\ B &\to b. \end{aligned}$$

Following the construction of Theorem 7.1, we get the solution

$$\begin{aligned} \delta(q_0, \lambda, z) &= \{(q_1, Sz)\}, \\ \delta(q_1, a, S) &= \{(q_1, SSSA)\}, \\ \delta(q_1, a, A) &= \{(q_1, B)\}, \\ \delta(q_1, \lambda, S) &= \{(q_1, \lambda)\}, \\ \delta(q_1, b, B) &= \{(q_1, \lambda)\}, \\ \delta(q_1, \lambda, z) &= \{(q_f, z)\}. \end{aligned}$$

with $F = \{q_f\}$

4. For the string $aabb$, the moves the npda makes are

$$\begin{aligned} (q_0, aabb, z) &\vdash (q_1, aabb, Sz) \vdash (q_1, abb, SAz) \vdash (q_1, bb, Az) \\ &\vdash (q_1, b, Bz) \vdash (q_1, \lambda, z) \vdash (q_2, \lambda, \lambda). \end{aligned}$$

Since q_2 is a final state, the pda accepts the string $aabb$. Similarly for $aaabbbb$, we find

$$\begin{aligned} (q_0, aaabbbb, z) &\vdash (q_1, aaabbbb, Sz) \vdash (q_1, aabbbb, SAz) \vdash (q_1, abbbb, SAAz) \\ &\vdash (q_1, bbbb, AAz) \vdash (q_1, bbb, BAz) \vdash (q_1, bb, Az) \\ &\vdash (q_1, b, Bz) \vdash (q_1, \lambda, z) \vdash (q_2, \lambda, \lambda). \end{aligned}$$

Therefore, $aaabbbb$ is also accepted by the pda.

6. First convert the grammar into Greibach normal form, giving $S \to aSSS$; $S \to bA$; $A \to a$. Then follow the construction of Theorem 7.1.

$$\begin{aligned} \delta(q_0, \lambda, z) &= \{(q_1, Sz)\}, \\ \delta(q_1, a, S) &= \{(q_1, SSS)\}, \\ \delta(q_1, b, S) &= \{(q_1, A)\}, \\ \delta(q_1, a, A) &= \{(q_1, \lambda)\}, \\ \delta(q_1, \lambda, z) &= \{(q_f, z)\}. \end{aligned}$$

9. From Theorem 7.2, given any npda, we can construct an equivalent context-free grammar. From that grammar we can then construct an equivalent three-state npda, using Theorem 7.1.

13. There must be at least one a to get started. After that, $\delta(q_0, a, A) = \{(q_0, A)\}$ simply reads a's without changing the stack. Finally, when the first b is encountered, the pda goes into state q_1, from which it can only make a λ-transition to the final state. Therefore, a string will be accepted if and only if it consists of one or more a's, followed by a single b.

14. From the grammar in Example 7.8, we find a derivation

$$\begin{aligned}
(q_0 z a_2) &\Rightarrow a(q0Aq_3)(q_3 z q_2) \\
&\Rightarrow aa(q_3 z a_2) \\
&\Rightarrow aa(q_0 A q_3)(q_3 z q_2) \\
&\Rightarrow aaa(q_0 A q_3)(q_3 z q_2) \\
&\overset{*}{\Rightarrow} aa^*(q_0 A q_3)(q_3 z q_2) \\
&\Rightarrow aa^*(q_3 z q_2) \\
&\Rightarrow aa^*(q_0 A q_1)(q_1 z q_2) \\
&\Rightarrow aa^*(q_1 z q_2) \\
&\Rightarrow aa^* b.
\end{aligned}$$

19. The key is that terminals can largely be treated as variables. For example, if

$$A \to abBBc$$

then the pda must have the transitions

$$(q_1, bBBc) \in \delta(q_1, a, A),$$

and

$$(q_1, BBC) \in \delta(q_1, b, b),$$

where $a, b, c \in T \cup \{\lambda\}$.

Section 7.3

2. A straightforward modification of Example 7.4. Put two tokens when input is an a and remove only one token when the input is a b. The solution is

$$\begin{aligned}
\delta(q_0, a, z) &= \{(q_1, AAz)\}, \\
\delta(q_1, a, A) &= \{(q_1, AAA)\}, \\
\delta(q_1, b, A) &= \{(q_2, \lambda)\}, \\
\delta(q_2, b, A) &= \{(q_2, \lambda)\}, \\
\delta(q_2, \lambda, z) &= \{(q_0, z)\},
\end{aligned}$$

where $F = \{q_0\}$.

4. This dpda goes into the final state after reading the prefix $a^n b^{n+3}$. If more b's are followed, it stays in this state and so accepts $a^n b^m$ for any $m > n+3$. A complete solution is given below.

$$\begin{aligned}
\delta(q_0, \lambda, z) &= \{(q_1, AAz)\}, \\
\delta(q_1, a, A) &= \{(q_1, AA)\}, \\
\delta(q_1, b, A) &= \{(q_2, \lambda)\}, \\
\delta(q_2, b, A) &= \{(q_2, \lambda)\}, \\
\delta(q_2, b, z) &= \{(q_3, z)\}, \\
\delta(q_3, b, z) &= \{(q_3, z)\},
\end{aligned}$$

where $F = \{q_3\}$.

6. At first glance, this may seem to be a nondeterministic language, since the prefix a calls for two different types of suffixes. Nevertheless, the language is deterministic, as we can construct a dpda. This dpda goes into a final state when the first input symbol is an a. If more symbols follow, it goes out of this state and then accepts $a^n b^n$. A complete solution is

$$\begin{aligned}
\delta(q_0, a, z) &= \{(q_3, Az)\}, \\
\delta(q_3, a, A) &= \{(q_1, AA)\}, \\
\delta(q_1, a, A) &= \{(q_1, AA)\}, \\
\delta(q_1, b, A) &= \{(q_2, \lambda)\}, \\
\delta(q_2, b, A) &= \{(q_2, \lambda)\}, \\
\delta(q_2, \lambda, z) &= \{(q_3, z)\}, \\
\delta(q_3, b, A) &= \{(q_2, \lambda)\},
\end{aligned}$$

where $F = \{q_3\}$.

9. Intuitively, we can check $n = m$ or $m = k$ if we know which case we want at the beginning of the string. But we have no way of deciding this deterministically.

11. The solution is straightforward. Put a's and b's on the stack. The c signals the switch from saving to matching, so everything can be done deterministically.

16. Let $L_1 = \{a^n b^n : n \geq 0\}$ and $L_2 = \{a^n b^{2n} c : n \geq 0\}$. Then clearly $L_1 \cup L_2$ is nondeterministic. But the reverse of this is $\{b^n a^n\} \cup \{cb^{2n} a^n\}$, which can be recognized deterministically by looking at the first symbol.

Section 7.4

1. (c) We can rewrite strings in L as follows:

$$a^n b^{n+2} c^m = a^n b^n b^2 c^2 c^{m-2} = a^n b^n b^2 c^2 c^k$$

where $n \geq 1,\ k \geq 1$. A grammar for L can be obtained as follows.

$$\begin{aligned} S &\rightarrow ABC, \\ A &\rightarrow aAb|ab, \\ B &\rightarrow bbcc, \\ C &\rightarrow cC|c. \end{aligned}$$

We claim that the grammar is $LL(2)$. First, the initial production must be $S \rightarrow ABC$. For the part $a^n b^n$, if the lookahead is aa, then we must use $A \rightarrow aAb$; if it is ab then we can only use $A \rightarrow \lambda$. For rest, there is never a choice of production, so the grammar is $LL(2)$.

3. Consider the strings $aabb$ and $aabbbbaa$. In the first case, the derivation must start with $S \Rightarrow aSb$, while in the second $S \Rightarrow SS$ is the necessary first step. But if we see only the first four symbols, we cannot decide which case applies. The grammar is therefore not in $LL(4)$. Since similar examples can be made for arbitrarily long strings, the grammar is not $LL(k)$ for any k.

Chapter 8

Section 8.1

2. Given m, we pick $w = a^m c^m b^m$ which is in L. The adversary now has several choices. If he chooses vxy to contain only b's, or a's or c's then the pumped string w_i will obviously not be in L for some $i \geq 1$. Therefore, the adversary chooses $v = a^k, y = c^l$. Then the pumped string is $w_i = a^{m+(i-1)k} c^{m+(i-1)l} b^m$. However, if $l \neq 0$, then $n_c(w_2) = m + l > m = n_b(w_0)$, so $w_2 \notin L$. On the other hand, if $l = 0$, then the pumped string $n_a(w_0) = m - k \neq m = n_b(w_0)$, so $w_0 \notin L$. Similarly, if the adversary chooses vxy to contain c's and b's, he cannot win either. Thus the pumping lemma fails and L is not context free.

6. Given m, we pick $w = a^m b^{2^m}$ which is in L. Obviously, the only choice for the adversary is $v = a^k,\ y = b^l$. The pumped string is $w_i = a^{m+(i-1)k} b^{2^m+(i-1)l}$. We claim that w_0 and w_2 cannot be both in L, therefore the pumping lemma fails and L is not context free. To show this, let us assume w_0 and w_2 are both in L, then we must have

$$2^{m-k} = 2^m - l,$$

and

$$2^{m+k} = 2^m + l.$$

Now multiply the above two equations side by side, we arrive at

$$2^{m-k} 2^{m+k} = (2^m - l)(2^m + l).$$

That is

$$2^{2m} = 2^{2m} - l^2.$$

Since $l = 0$ implies $k = 0$ and this in turn implies $|vy| = 0$.. We have a contradiction.

7. (c) Given m, we pick $w = a^m b^m c^{m^2}$ which is in L. It is easy to see that the only troublesome choice for the adversary is $v = b^k, y = c^l$ for $k \neq 0$, $l \neq 0$. Then the pumped string is $w_i = a^m b^{m+(i-1)k} c^{m^2+(i-1)l}$. However, for $w_0 = a^m b^{m-k} c^{m^2-l}$, we have

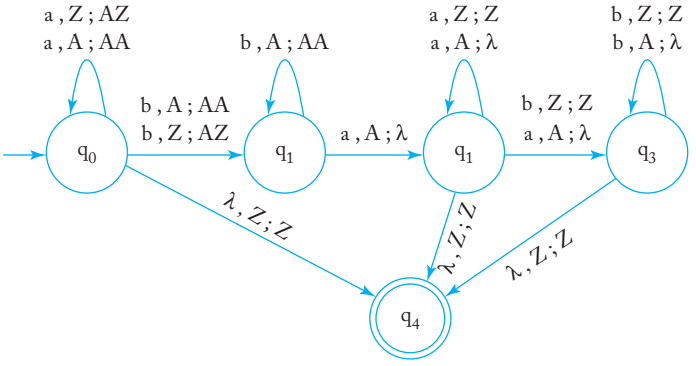
$$m(m-k) = m^2 - mk < m^2 - l$$

since $l \leq m$ and $k \geq 1$. So $w_0 \notin L$, so L is not context free.

8. (a) The language is context free. A grammar for it is

$$\begin{aligned} S &\rightarrow aSb|S_1, \\ S_1 &\rightarrow aS_1a|bS_1b|\lambda. \end{aligned}$$

(d) The language is context free and an npda is given below.



(f) The language is not context free. Use the pumping lemma with $w = a^m b^m c^m$ and examine various choices of v and y.

11. The language is context free with a grammar

$$\begin{aligned} S &\rightarrow S_1S_1, \\ S_1 &\rightarrow aS_1b|\lambda. \end{aligned}$$

We next see if the language is linear. Given m, we pick $w = a^m b^m a^m b^m$ which is in L. Obviously, whatever the decomposition is for vxy, it must be of the form $v = a^k$, $y = b^l$. Then the pumped string is $w_i = a^{m+(i-1)k} b^m a^m b^{m+(i-1)l}$ for some $1 \leq k + l \leq m$. For $w_0 = a^{m-k} b^m a^m b^{m-l} \notin L$. Therefore, L is context free but not linear.

Section 8.2

3. For each $a \in T$, substitute it by $h(a)$. The new set of productions gives a context-free grammar $\widehat{G}$. A simple induction shows that $L(\widehat{G}) = h(L)$.

7. The arguments are similar to those in Theorem 8.5. Consider $\Sigma^* - L = \overline{L}$. If the context-free languages were closed under difference, then $\overline{L}$ would always be context free, in contradiction to an established results.

To show that the closure result holds for L_2 regular, notice that regularity is closed under complement. L_2 is regular, so is $(\overline{L_2})$. Therefore, $L_1 - L_2 = L_1 \cap (\overline{L_2})$ is context free by Theorem 8.5.

9. Given two linear grammars $G_1 = (V_1, T, S_1, P_1)$ and $G_2 = (V_2, T, S_2, P_2)$ with $V_1 \cap V_2 = \varnothing$, form the combined grammar

$$\widehat{G} = (V_1 \cup V_2, T, S, \ P_1 \cup \ P_2 \cup S \to S_1 | S_2)$$

Then $\widehat{G}$ is linear and $L\left(\widehat{G}\right) = L(G_1) \cup L(G_2)$. To show that linear languages are not closed under concatenation, take the linear language $L = \{a^n b^n : n \geq 1\}$. The language L^2 is not linear, as can be shown by an application of the pumping lemma.

15. The languages $L_1 = \{a^n b^n c^m\}$ and $L_2 = \{a^n b^m c^m\}$ are both unambiguous. But their intersection $\{a^n b^n c^n\}$ is not context free.

21. $\lambda \in L(G)$ if and only if S is nullable.

23. Let L_1 be the given context-free language, and $L_2 = \{w \in \Sigma^* : |w| \text{ is even}\}$. Since L_2 is regular, by Theorem 8.5, $L = L_1 \cap L_2$ is context-free. Hence, by Theorem 8.6, there exists an algorithm for deciding whether or not L is empty. That means there is an algorithm to determine if L_1 contains any even length.

Chapter 9

Section 9.1

1. A Turing machine that accepts the language L is

$$\begin{aligned}
\delta(q_0, a) &= (q_1, a, R), \\
\delta(q_1, a) &= (q_2, a, R), \\
\delta(q_2, a) &= (q_3, a, R), \\
\delta(q_3, a) &= (q_3, a, L), \\
\delta(q_3, b) &= (q_4, b, R), \\
\delta(q_3, \sqcup) &= (q_5, \sqcup, L), \\
\delta(q_4, b) &= (q_4, b, R), \\
\delta(q_4, \sqcup) &= (q_5, \sqcup, L),
\end{aligned}$$

with $F = \{q_5\}$

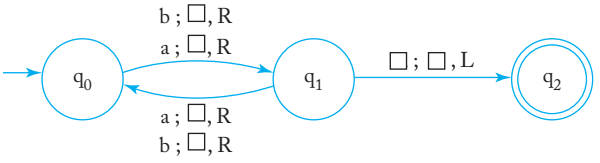
4. In both cases, the Turing machine will halt in the nonfinal state q_0. The instantaneous descriptions are

$$q_0aba \vdash xq_1ba \vdash xq_2ya \vdash q_2xya \vdash xq_0ya,$$

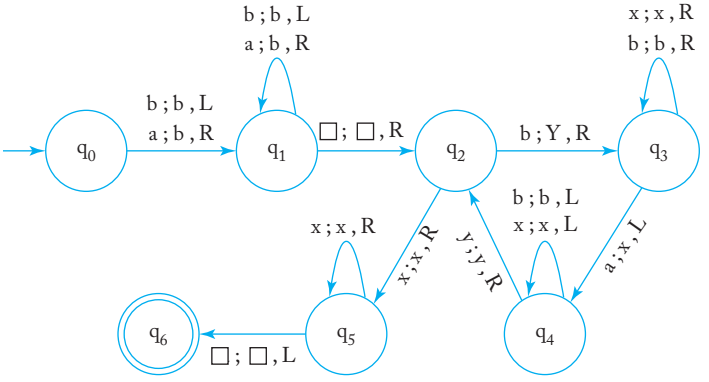
and

$$q_0aaabbbb \overset{*}{\vdash} xaaq_2ybbb \overset{*}{\vdash} xxxq_2yyyb \overset{*}{\vdash} xxxq_0yyyb.$$

8. (b) A transition graph with $F = \{q_2\}$ is



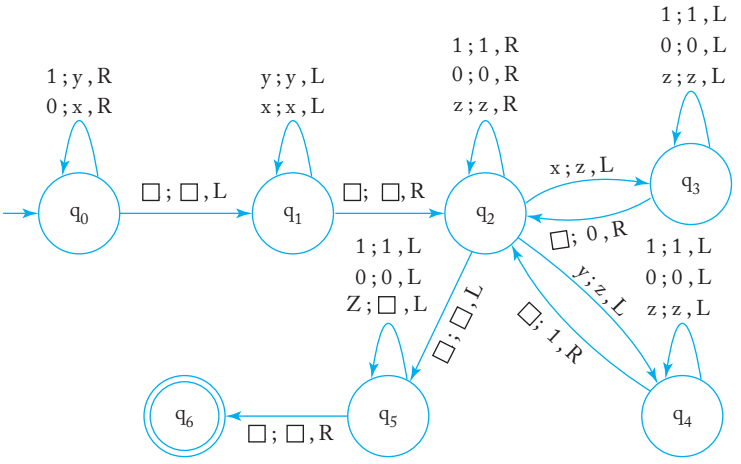
(f) A transition graph with $F = \{q_6\}$ is



10. To solve the problem, we implement the following process:

1. Replace every 0 by x and 1 by y.
2. Find the leftmost symbol of w which is not z, remember it by entering the appropriate internal state and then replace it by z.
3. Travel left to the first blank cell and create a 0 or 1 according to the internal state associated with the remembered symbol.
4. Repeat step 2 and step 3 until there are no more x's and y's.

The Turing machine for the solution is somewhat complicated.



12. Using unary representation for w, the Turing machine for the solution of this problem is.

$$\begin{aligned}
\delta(q_0, 1) &= (q_1, \sqcup, R),\\
\delta(q_0, \sqcup) &= (q_4, \sqcup, L),\\
\delta(q_2, 1) &= (q_3, 1, L),\\
\delta(q_2, \sqcup &= (q_4, \sqcup, L),\\
\delta(q_3, \sqcup) &= (q_4, \sqcup, R),
\end{aligned}$$

with $F = \{q_4\}$.

13. (a) Use the construction in Example 9.10 to duplicate x, then add a symbol 1 to the resulting string. The Turing machine has the following transition functions.

$$\begin{aligned}
\delta(q_0, 1) &= (q_0, x, R),\\
\delta(q_0, \sqcup) &= (q_1, \sqcup, L),\\
\delta(q_1, 1) &= (q_1, 1, L),\\
\delta(q_1, x) &= (q_2, 1, R),\\
\delta(q_2, 1) &= (q_2, 1, R),\\
\delta(q_2, \sqcup) &= (q_1, 1, L),\\
\delta(q_1, \sqcup) &= (q_3, 1, L),\\
\delta(q_3, \sqcup) &= (q_4, \sqcup, R),
\end{aligned}$$

with $F = \{q_4\}$.

Section 9.2

2. Suppose we use symbol s to denote the minus sign. For example, $s111 = -3$. Also suppose our input for the unary subtraction $x -$

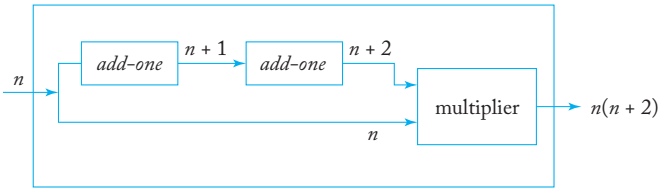
y is represented by $0w(x)sw(y)0$, then the following outline for the construction for *subtracter* will perform the computation

$$q_0 0w(x)sw(y)0 \overset{*}{\vdash} q_f(w(x) - w(y)) \quad \text{if } x \geq y$$
$$\overset{*}{\vdash} q_f s(w(y) - w(x)) \quad \text{if } x < y$$

with $F = \{q_f\}$. The subtraction can then be achieved by

1. Repeat the following steps until either x or y contains no more 1's. For each 1 in $w(x)$ find a corresponding 1 in $w(y)$, then replace both 1's by a token z, respectively.
2. If $w(y)$ contains no more 1's, remove s and all the z's to get the computed result, otherwise just remove all the z's to get the (negative) result.

3. (a) We can think of the machine as constituted of two main parts, an *add-one* machine that just adds one to the input, and a multiplier that multiplies two numbers. Schematically they are combined in a simple fashion.



5. (a) Define the macros:

$$3split: \text{ convert } w_1w_2w_3 \text{ into } w_1xw_2xw_3.$$
$$reverse - compare: \text{ compare } w \text{ against } w^R.$$

Then a Turing machine can be constructed by

$step\ 1$: $3split$ input.
$step\ 2$: $reverse - compare$ w against w^R followed by
$reverse - compare$ w^R against w.

Accept the input only when both steps are successful.

Chapter 10

Section 10.1

2. The off-line Turing machine is a Turing machine with transition function

$$\delta : Q \times \Sigma \times \Gamma \to Q \times \Gamma \times \{L, R\}.$$

For example,

$$\delta(q_i, a, b) = (q_j, c, L)$$

means that the transition rule can be applied only if the machine is in state q_i, and the read-only head of the input file sees an a and the read-write head of the tape sees a b. The symbol b on the input file will be replaced by a c, then the read-write head will move to the left. At this point, the input file's read-only head moves to the right and the control unit changes its state to q_j.

For the simulation by a standard Turing machine, the tape of the simulating machine is divided into two parts; one for the input file, the other the working tape. In any particular step, the Turing machine finds the input part, remembers the symbol there and erases it, then returns to its working part.

5. The machine has a transition function

$$\widehat{\delta} : Q \times \Gamma \to Q \times \Gamma \times I \times \{L, R\}$$

where $I = \{1, 2, ...\}$.

For example, a transition $\widehat{\delta}(q_i, a) = (q_j, b, 5, R)$ can be simulated by the standard Turing machine as

$$\begin{aligned} \widehat{\delta}(q_i, a) &= (q_{iR_1}, b, R) \\ \widehat{\delta}(q_{iR_1}, c) &= (q_{iR_2}, c, R) \\ \text{.....} \quad & \quad \text{.....} \\ \widehat{\delta}(q_{iR_4}, c) &= (q_j, c, R) \end{aligned}$$

for all $c \in \Sigma$.

10. The machine has transition function

$$\widehat{\delta} : Q \times \Gamma \times \Gamma \times \Gamma \to Q \times \Gamma \times \{L, R\}\,.$$

For example, $\widehat{\delta}\,(q_i, b, a, c) = (q_j, d, R)$ means that the machine is in state q_i, the read-write head sees a symbol a with a b on its left and a c in its right on the tape. Then the symbol a will be replaced with a d and the read-write head will move to its right. At this point the control unit changes its state to q_j.

The transition $\widehat{\delta}\,(q_i, (b, a, c)) = (q_j, d, R)$ can be simulated by a standard Turing machine as

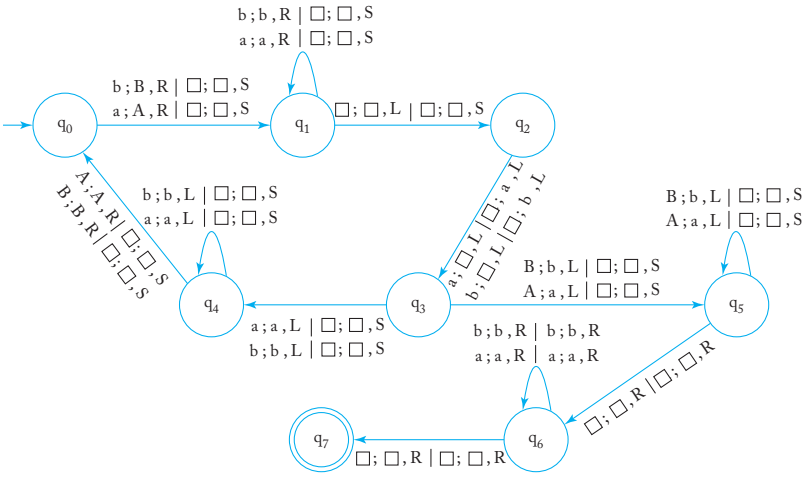
$$\begin{aligned} \delta(q_i, a) &= (q_{iL}, a, L) \\ \delta(q_{iL}, b) &= (q_{iR}, b, R) \\ \delta(q_{iR}, a) &= (q_{iR}, a, R) \\ \delta(q_{iR}, c) &= (q_{ic}, c, L) \\ \delta(q_{ic}, a) &= (q_j, d, R). \end{aligned}$$

Section 10.2

1. (c) The key is to split the input into two equal length parts. Keep the first part in tape 1 and move the second part into tape 2. Then we can compare the two tapes symbol by symbol without going back and forth like in the standard machine.

We start from the leftmost symbol on tape 1 and mark it with A for an a and B for a b. then replace the rightmost symbol by a blank after moving it to tape 2 in a reverse order. Repeat this process on tape 1 to mark the leftmost unmarked symbol and then move the rightmost symbol onto tape 2 and replace it by blank. We split the input strings into two equal length substrings when the process completes successfully with no unmarked symbols remain on tape 1.

The idea is straightforward, but the implementation tends to be somewhat long. A graph with $F = \{q_7\}$ is shown below.



2. A multitape off-line Turing machine is a machine that has a single input file with read-only head and n-tapes with each a read-write head. The transition function has form

$$\delta : Q \times \Sigma \times \Gamma^n \to Q \times \Gamma^n \times \{L, R\}^n.$$

For example, if $n = 2$, then the transition

$$\delta(q_i, a, b, c) = (q_j, d, e, R, L)$$

means that the transition rule can be applied only if the machine is in state q_i; the read-only head of the input file sees an a and the read-write head of tape 1 sees a b, and tape 2 sees a c. The symbol b on tape 1 will be replaced by a d with its read-write head moving to

the right. At the same time, the symbol c on tape 2 will be rewritten by an e and its read-write head will move to the left. Finally, the read-only head moves to right and the control unit then changes its state to q_j.

The simulation of the n-tape off-line machine by a standard machine using an $n+1$ tracks tape is the same as that for the $(n+1)$-tape Turing machine with the only exception that the first track corresponding to the read-only head will always move to right and never change its contents in any state. The description of the simulation for a multitape Turnig machine is given at the beginning of this section.

Chapter 11

Section 11.1

2. The union of two countable sets is countable and the set of all recursively enumerable languages is countable. If the set of all languages that are not recursively enumerable were also countable, then the set of all languages would be countable. But this is not the case.

4. Use the pattern in Figure 11.1 to list w_{ij}, where w_{ij} is the i-th word in L^j. This works because each L^j is enumerable.

11. A context-free language is recursive, so by Theorem 11.4 its complement is also recursive. Note, however, that the complement is not necessarily context free.

16. If $S_2 - S_1$ is a finite set, then it is countable. This means that $S_2 = (S_2 - S_1) \cup S_1$ must be countable. This is impossible because S_2 is known not countable. Therefore, S_2 must contain an infinite number of elements that are not in S_1.

19. Since every positive rational number can be represented by two integers we know from the enumeration procedure shown in Figure 10.1 that the set of positive rational numbers is countable. So is the set of non-positive rational numbers. Therefore, the the set of rational numbers is countable.

Section 11.2

1. A typical derivation:

$$\begin{aligned} S &\Rightarrow S_1B \Rightarrow aS_1bB \overset{*}{\Rightarrow} a^nS_1b^nB \Rightarrow a^{n-1}aS_1bb^{n-1}B \\ &\Rightarrow a^{n-1}aab^{n-1}B \Rightarrow a^{n+1}b^{n-2}bbB \\ &\overset{*}{\Rightarrow} a^{n+1}b^{n-2}b^{2m-1}bB \Rightarrow a^{n+1}b^{n-1}b^{2m}. \end{aligned}$$

From this, it is not hard to conjecture that $L(M) = \{a^{n+1}b^{n-1}b^{2m} : n \geq 1,\ m \geq 0\}$.

3. Formally, the grammar can be described by $G = (V, S, T, P)$, with $S \subseteq (V \cup T)^+$ and

$$L(G) = \left\{ x \in T^* : s \overset{*}{\Rightarrow}_G x \text{ for any } s \in S \right\}.$$

The unrestricted grammars in Definition 11.3 are equivalent to this extension because to any given unrestricted grammar we can always add starting rules $S_0 \to s_i$ for all $s_i \in S$.

6. To get this form of unrestricted grammars, insert dummy variables on the right whenever $|u| > |v|$. For example,

$$AB \to C$$

can be replaced by

$$\begin{aligned} AB &\to CD, \\ D &\to \lambda. \end{aligned}$$

The equivalence argument is straightforward.

Section 11.3

2. (a) A context-sensitive grammar is

$$\begin{aligned} S &\to aaAbbc|aab, \\ Ab &\to bA, \\ Ac &\to Bcc, \\ bB &\to Bb, \\ aB &\to aaAb, \\ aA &\to aa. \end{aligned}$$

3. (a) The solution can be made more intuitive by using variables subscripted with the terminal they are to create. An answer is

$$\begin{aligned} S &\to SV_aV_bV_c|V_aV_bV_c, \\ V_aV_b &\to V_bV_a, \\ V_aV_c &\to V_cV_a, \\ V_bV_a &\to V_aV_b, \\ V_bV_c &\to V_cV_b, \\ V_cV_a &\to V_aV_c, \\ V_cV_b &\to V_bV_c, \\ V_a &\to a, \\ V_b &\to b, \\ V_c &\to c. \end{aligned}$$

Chapter 12

Section 12.1

2. $\widehat{M}$ first saves its input and replaces it with w, then proceeds like M. If (M, w) halts, $\widehat{M}$ checks its original input and accepts it if it is of even length.

3. Given M and w, modify M to get $\widehat{M}$, which halts if and only if a special symbol, say an introduced symbol #, is written. We can do this by changing the halting configurations of M so that every halting configuration writes #, then stops. Thus, M halts implies that $\widehat{M}$ writes #, and $\widehat{M}$ writes # implies that M halts. Thus, if we have an algorithm that tells us whether or not a specified symbol a is ever written, we apply it to $\widehat{M}$ with $a = \#$. This would solve the halting problem.

9. Yes, this is decidable. The only way a dpda can go into an infinite loop is via λ-transitions. We can find out if there is a sequence of λ-transitions that (a) eventually produce the same state and stack top and (b) do not decrease the length of the stack. If there is no such sequence, then the dpda must halt. If there is such a sequence, determine whether the configuration that starts it is reachable. Since, for any given w, we need only analyze a finite number of moves, this can always be done

Section 12.2

4. Suppose we had an algorithm to decide whether or not $L(M_1) \subseteq L(M_2)$. We could then construct a machine M_2 such that $L(M_2) = \varnothing$ and apply the algorithm. Then $L(M_1) \subseteq L(M_2)$ if and only if $L(M_1) = \varnothing$. But this contradicts Theorem 12.3, since we can construct M_1 from any given grammar G.

7. If we take $L(G_2) = \Sigma^*$, the problem becomes the question of Theorem 12.3 and is therefore undecidable.

9. Again, the same type of argument as in Theorem 12.4 and Example 12.4, but there are several steps involved that make the problem a little more difficult.

(a) Start with the halting problem (M, w).

(b) Given any G_2 with $L(G_2) \neq \varnothing$, generate some $v \in L(G_2)$. This can always be done since G_2 is regular.

(c) As in Theorem 12.4, modify M to $\widehat{M}$ so that if (M, w) halts, then $\widehat{M}$ accepts v.

(d) From $\widehat{M}$ generates G_1, so that $L(\widehat{M}) = L(G_1)$.

Now, if (M, w) halts, then $\widehat{M}$ accepts v and $L(G_1) \cap L(G_2) \neq \varnothing$. If (M, w) does not halt, then $\widehat{M}$ accepts nothing, so that $L(G_1) \cap L(G_2) = \varnothing$. This construction can be done for any G_2, so that there cannot exist any G_2 for which the problem is decidable.

Section 12.3

2. A PC-solution is $w_3w_4w_1 = v_3v_4v_1$. There is no MPC solution because one string would have a prefix 001, the other 01.

6. (a) The problem is undecidable. If it were decidable, we would have an algorithm for deciding the original MPC problem. Given $w_1, w_2..., w_n$, we form $w_1^R, w_2^R..., w_n^R$ and use the assumed algorithm. Since $w_1w_i...w_k = \left(w_k^R...w_i^Rw_1^R\right)^R$, the original MPC problem has a solution if and only if the new MPC problem has a solution.

Section 12.5

2.

- **On a standard machine**. Find the middle of the string, then go back and forth to match symbols. Both parts take $O(n^2)$ moves.
- **On a two-tape deterministic machine**. Count the number of symbols. If the count is done in unary, this is an $O(n)$ operation. Next, write the first half on the second tape. This is also an $O(n)$ operation. Finally, you can compare in $O(n)$ moves.
- **On a single-tape nondeterministic machine**. You can guess the middle of the string nondeterministically in one move. But the matching still takes $O(n^2)$ moves.
- **On a two-tape nondeterministic machine**. Guess the middle of the string, then proceed as in a two-tape deterministic machine. Total effort required is $O(n)$.

Chapter 13

Section 13.1

2. Using the function *subtr* in Example 13.3, we get the solution

$$greater\,(x, y) = subtr\,(1, subtr\,(1, subtr\,(x, y)))\,.$$

4. Using the function *mult* in Example 13.2 and *subtr* in Example 13.3, we get the solution

$$equals\,(x, y) = mult(subtr\,(1, subtr(x, y)), subtr\,(1, subtr\,(y, x)))\,.$$

8. The function can be defined as

$$\begin{aligned} f(0) &= 1, \\ f(n+1) &= mult(2, f(n)). \end{aligned}$$

12. (a) Direct derivation from the definition of Ackermann's function gives

$$\begin{aligned} A(1,y) &= A(0, A(1, y-1)) \\ &= A(1, y-1) + 1 \\ &= A(1, y-2) + 2 \\ &\vdots \\ &= A(1,0) + y \\ &= y + 2. \end{aligned}$$

Section 13.2

1. (a) $\lambda,\ ab,\ aababb,\ aaababbaababbb$

8. The first few sentences derived from the axiom are found as

$$ab,\ a(ab)^2,\ a(a(ab)^2)^2,\ a(a(a(ab)^2)^2)^2....$$

Therefore, the language is

$$L = \{ab\} \cup L_1$$

where

$$L_1 = \{(a_n(a_{n-1}...(a_1(ab)^{k_1})...)^{k_{n-1}})^{k_n} : k_i = 2, a_i = a, i \leq n = 1, 2...\}.$$

Section 13.3

1.

$$\begin{aligned} &P_1 : S \to S_1 S_2, \\ &P_2 : S_1 \to aS_1, S_2 \to bS_2c, \\ &P_3 : S_1 \to \lambda, S_2 \to \lambda. \end{aligned}$$

6. The solution here is reminiscent of the use of messengers with context-sensitive grammars.

$$\begin{aligned} ab &\to x, \\ xb &\to bx, \\ xc &\to \lambda. \end{aligned}$$