CENGAGE
Learning

# Instructor's Solutions Manual for Introduction to the Theory of Computation third edition

**Michael Sipser**

**Mathematics Department**
**MIT**

CENGAGE Learning®

# Preface

This Instructor's Manual is designed to accompany the textbook, *Introduction to the Theory of Computation, third edition*, by Michael Sipser, published by Cengage, 2013. It contains solutions to almost all of the exercises and problems in Chapters 0–9. Most of the omitted solutions in the early chapters require figures, and producing these required more work that we were able to put into this manual at this point. A few problems were omitted in the later chapters without any good excuse.

Some of these solutions were based on solutions written by my teaching assistants and by the authors of the Instructor's Manual for the first edition.

This manual is available only to instructors.

# Chapter 0

**0.1** **a.** The odd positive integers.
**b.** The even integers.
**c.** The even positive integers.
**d.** The positive integers which are a multiple of 6.
**e.** The palindromes over $\{\texttt{0,1}\}$.
**f.** The empty set.

**0.2** **a.** $\{1, 10, 100\}$.
**b.** $\{n|\ n > 5\}$.
**c.** $\{1, 2, 3, 4\}$.
**d.** $\{\texttt{aba}\}$.
**e.** $\{\varepsilon\}$.
**f.** $\emptyset$.

**0.3** **a.** No.
**b.** Yes.
**c.** $A$.
**d.** $B$.
**e.** $\{(\texttt{x}, \texttt{x}), (\texttt{x}, \texttt{y}), (\texttt{y}, \texttt{x}), (\texttt{y}, \texttt{y}), (\texttt{z}, \texttt{x}), (\texttt{z}, \texttt{y})\}$.
**f.** $\{\emptyset, \{\texttt{x}\}, \{\texttt{y}\}, \{\texttt{x}, \texttt{y}\}\}$.

**0.4** $A \times B$ has $ab$ elements, because each element of $A$ is paired with each element of $B$, so $A \times B$ contains $b$ elements for each of the $a$ elements of $A$.

**0.5** $\mathcal{P}(C)$ contains $2^c$ elements because each element of $C$ may either be in $\mathcal{P}(C)$ or not in $\mathcal{P}(C)$, and so each element of $C$ doubles the number of subsets of $C$. Alternatively, we can view each subset $S$ of $C$ as corresponding to a binary string $b$ of length $c$, where $S$ contains the $i$th element of $C$ iff the $i$th place of $b$ is $\texttt{1}$. There are $2^c$ strings of length $c$ and hence that many subsets of $C$.

**0.6** **a.** $f(2) = 7$.
**b.** The range $= \{6, 7\}$ and the domain $= \{1, 2, 3, 4, 5\}$.
**c.** $g(2, 10) = 6$.
**d.** The range $= \{1, 2, 3, 4, 5\} \times \{6, 7, 8, 9, 10\}$ and the domain $= \{6, 7, 8, 9, 10\}$.
**e.** $f(4) = 7$ so $g(4, f(4)) = g(4, 7) = 8$.

**0.7** The underlying set is $\mathcal{N}$ in these examples.

**a.** Let $R$ be the "within 1" relation, that is, $R = \{(a, b)|\ |a - b| \leq 1\}$.

**b.** Let $R$ be the "less than or equal to" relation, that is, $R = \{(a, b)|\ a \leq b\}$.

**c.** Finding a $R$ that is symmetric and transitive but not reflexive is tricky because of the following "near proof" that $R$ cannot exist! Assume that $R$ is symmetric and transitive and chose any member $x$ in the underlying set. Pick any other member $y$ in the underlying set for which $(x, y) \in R$. Then $(y, x) \in R$ because $R$ is symmetric and so $(x, x) \in R$ because $R$ is transitive, hence $R$ is reflexive. This argument fails to be an actual proof because $y$ may fail to exist for $x$.

Let $R$ be the "neither side is 1" relation, $R = \{(a, b)|\ a \neq 1 \text{ and } b \neq 1\}$.

**0.10** Let $G$ be any graph with $n$ nodes where $n \geq 2$. The degree of every node in $G$ is one of the $n$ possible values from 0 to $n - 1$. We would like to use the pigeon hole principle to show that two of these values must be the same, but number of possible values is too great. However, not all of the values can occur in the same graph because a node of degree 0 cannot coexist with a node of degree $n - 1$. Hence $G$ can exhibit at most $n - 1$ degree values among its $n$ nodes, so two of the values must be the same.

**0.11** The error occurs in the last sentence. If $H$ contains at least 3 horses, $H_1$ and $H_2$ contain a horse in common, so the argument works properly. But, if $H$ contains exactly 2 horses, then $H_1$ and $H_2$ each have exactly 1 horse, but do not have a horse in common. Hence we cannot conclude that the horse in $H_1$ has the same color as the horse in $H_2$. So the 2 horses in $H$ may not be colored the same.

**0.12** **a.** *Basis:* Let $n = 0$. Then, $S(n) = 0$ by definition. Furthermore, $\frac{1}{2}n(n + 1) = 0$. So $S(n) = \frac{1}{2}n(n + 1)$ when $n = 0$.

*Induction:* Assume true for $n = k$ where $k \geq 0$ and prove true for $n = k + 1$. We can use this series of equalities:

$$\begin{aligned} S(k+1) &= 1 + 2 + \cdots + k + (k+1) && \text{by definition} \\ &= S(k) + (k+1) && \text{because } S(k) = 1 + 2 + \cdots + k \\ &= \tfrac{1}{2}k(k+1) + (k+1) && \text{by the induction hypothesis} \\ &= \tfrac{1}{2}(k+1)(k+2) && \text{by algebra} \end{aligned}$$

**b.** *Basis:* Let $n = 0$. Then, $C(n) = 0$ by definition, and $\frac{1}{4}(n^4 + 2n^3 + n^2) = 0$. So $C(n) = \frac{1}{4}(n^4 + 2n^3 + n^2)$ when $n = 0$.

*Induction:* Assume true for $n = k$ where $k \geq 0$ and prove true for $n = k + 1$. We can use this series of equalities:

$$\begin{aligned} C(k+1) &= 1^3 + 2^3 + \cdots + k^3 + (k+1)^3 && \text{by definition} \\ &= C(k) + (k+1)^3 && C(k) = 1^3 + \cdots + k^3 \\ &= \tfrac{1}{4}(n^4 + 2n^3 + n^2) + (k+1)^3 && \text{induction hypothesis} \\ &= \tfrac{1}{4}((n+1)^4 + 2(n+1)^3 + (n+1)^2) && \text{by algebra} \end{aligned}$$

**0.13** Dividing by $(a - b)$ is illegal, because $a = b$ hence $a - b = 0$ and division by 0 is undefined.

# Chapter 1

**1.12** Observe that $D \subseteq \mathtt{b}^*\mathtt{a}^*$ because $D$ doesn't contain strings that have ab as a substring. Hence $D$ is generated by the regular expression $(\mathtt{aa})^*\mathtt{b}(\mathtt{bb})^*$. From this description, finding the DFA for $D$ is more easily done.

**1.14** **a.** Let $M'$ be the DFA $M$ with the accept and non-accept states swapped. We show that $M'$ recognizes the complement of $B$, where $B$ is the language recognized by $M$. Suppose $M'$ accepts $x$. If we run $M'$ on $x$ we end in an accept state of $M'$. Because $M$ and $M'$ have swapped accept/non-accept states, if we run $M$ on $x$, we would end in a non-accept state. Therefore, $x \notin B$. Similarly, if $x$ is not accepted by $M'$, then it would be accepted by $M$. So $M'$ accepts exactly those strings not accepted by $M$. Therefore, $M'$ recognizes the complement of $B$.

Since $B$ could be any arbitrary regular language and our construction shows how to build an automaton to recognize its complement, it follows that the complement of any regular language is also regular. Therefore, the class of regular languages is closed under complement.

**b.** Consider the NFA in Exercise 1.16(a). The string $a$ is accepted by this automaton. If we swap the accept and reject states, the string $a$ is still accepted. This shows that swapping the accept and non-accept states of an NFA doesn't necessarily yield a new NFA recognizing the complementary language. The class of languages recognized by NFAs is, however, closed under complement. This follows from the fact that the class of languages recognized by NFAs is precisely the class of languages recognized by DFAs which we know is closed under complement from part (a).

**1.18** Let $\Sigma = \{\mathtt{0}, \mathtt{1}\}$.

**a.** $\mathtt{1}\Sigma^*\mathtt{0}$

**b.** $\Sigma^*\mathtt{1}\Sigma^*\mathtt{1}\Sigma^*\mathtt{1}\Sigma^*$

**c.** $\Sigma^*\mathtt{0101}\Sigma^*$

**d.** $\Sigma\Sigma\mathtt{0}\Sigma^*$

**e.** $(\mathtt{0} \cup \mathtt{1}\Sigma)(\Sigma\Sigma)^*$

**f.** $(\mathtt{0} \cup (\mathtt{10})^*)^*\mathtt{1}^*$

**g.** $(\varepsilon \cup \Sigma)(\varepsilon \cup \Sigma)(\varepsilon \cup \Sigma)(\varepsilon \cup \Sigma)(\varepsilon \cup \Sigma)$

**h.** $\Sigma^*\mathtt{0}\Sigma^* \cup \mathtt{1111}\Sigma^* \cup \mathtt{1} \cup \varepsilon$

**i.** $(\mathtt{1}\Sigma)^*(\mathtt{1} \cup \varepsilon)$

**j.** $\mathtt{0}^*(\mathtt{100} \cup \mathtt{010} \cup \mathtt{001} \cup \mathtt{00})\mathtt{0}^*$

**k.** $\varepsilon \cup \mathtt{0}$

**l.** $(\mathtt{1}^*\mathtt{0}\mathtt{1}^*\mathtt{0}\mathtt{1}^*)^* \cup \mathtt{0}^*\mathtt{1}\mathtt{0}^*\mathtt{1}\mathtt{0}^*$

**m.** $\emptyset$
**n.** $\Sigma^{+}$

**1.20** **a.** ab, $\varepsilon$; ba, aba
**b.** ab, abab; $\varepsilon$, aabb
**c.** $\varepsilon$, aa; ab, aabb
**d.** $\varepsilon$, aaa; aa, b
**e.** aba, aabbaa; $\varepsilon$, abbb
**f.** aba, bab; $\varepsilon$, ababab
**g.** b, ab; $\varepsilon$, bb
**h.** ba, bba; b, $\varepsilon$

**1.21** In both parts we first add a new start state and a new accept state. Several solutions are possible, depending on the order states are removed.
**a.** Here we remove state 1 then state 2 and we obtain
$\mathtt{a}^*\mathtt{b}(\mathtt{a} \cup \mathtt{ba}^*\mathtt{b})^*$
**b.** Here we remove states 1, 2, then 3 and we obtain
$\varepsilon \cup ((\mathtt{a} \cup \mathtt{b})\mathtt{a}^*\mathtt{b}((\mathtt{b} \cup \mathtt{a}(\mathtt{a} \cup \mathtt{b}))\mathtt{a}^*\mathtt{b})^*(\varepsilon \cup \mathtt{a}))$

**1.22** **b.** $/\#(\#^*(\mathtt{a} \cup \mathtt{b}) \cup /)^*\#^+/$

**1.24** **a.** $q_1, q_1, q_1, q_1$; 000.
**b.** $q_1, q_2, q_2, q_2$; 111.
**c.** $q_1, q_1, q_2, q_1, q_2$; 0101.
**d.** $q_1, q_3$; 1.
**e.** $q_1, q_3, q_2, q_3, q_2$; 1111.
**f.** $q_1, q_3, q_2, q_1, q_3, q_2, q_1$; 110110.
**g.** $q_1$; $\varepsilon$.

**1.25** A *finite state transducer* is a 5-tuple $(Q, \Sigma, \Gamma, \delta, q_0)$, where
i) $Q$ is a finite set called the *states*,
ii) $\Sigma$ is a finite set called the *alphabet*,
iii) $\Gamma$ is a finite set called the *output alphabet*,
iv) $\delta \colon Q \times \Sigma \longrightarrow Q \times \Gamma$ is the *transition function*,
v) $q_0 \in Q$ is the *start state*.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0)$ be a *finite state transducer*, $w = w_1 w_2 \cdots w_n$ be a string over $\Sigma$, and $v = v_1 v_2 \cdots v_n$ be a string over the $\Gamma$. Then $M$ *outputs* $v$ if a sequence of states $r_0, r_1, \ldots, r_n$ exists in Q with the following two conditions:
i) $r_0 = q_o$
ii) $\delta(r_i, w_{i+1}) = (r_{i+1}, v_{i+1})$ for $i = 0, \ldots, n-1$.

**1.26** **a.** $T_1 = (Q, \Sigma, \Gamma, \delta, q_1)$, where
i) $Q = \{q_1, q_2\}$,
ii) $\Sigma = \{0, 1, 2\}$,
iii) $\Gamma = \{0, 1\}$,
iv) $\delta$ is described as

| | 0 | 1 | 2 |
|---|---|---|---|
| $q_1$ | ($q_1$,0) | ($q_1$,0) | ($q_2$,1) |
| $q_2$ | ($q_1$,0) | ($q_2$,1) | ($q_2$,1) |

v) $q_1$ is the start state.

**b.** $T_2 = (Q, \Sigma, \Gamma, \delta, q_1)$, where
i) $Q = \{q_1, q_2, q_3\}$,
ii) $\Sigma = \{\texttt{a}, \texttt{b}\}$,
iii) $\Gamma = \{\texttt{0}, \texttt{1}\}$,
iv) $\delta$ is described as

| | a | b |
|---|---|---|
| $q_1$ | ($q_2$,1) | ($q_3$,1) |
| $q_2$ | ($q_3$,1) | ($q_1$,0) |
| $q_3$ | ($q_1$,0) | ($q_2$,1) |

v) $q_1$ is the start state.

**1.29** **b.** Let $A_2 = \{www|\ w \in \{\texttt{0,1}\}^*\}$. We show that $A_2$ is nonregular using the pumping lemma. Assume to the contrary that $A_2$ is regular. Let $p$ be the pumping length given by the pumping lemma. Let $s$ be the string $\texttt{a}^p\texttt{ba}^p\texttt{ba}^p\texttt{b}$. Because $s$ is a member of $A_2$ and $s$ has length more than $p$, the pumping lemma guarantees that $s$ can be split into three pieces, $s = xyz$, satisfying the three conditions of the lemma. However, condition 3 implies that $y$ must consist only of as, so $xyyz \notin A_2$ and one of the first two conditions is violated. Therefore $A_2$ is nonregular.

**1.30** The error is that $s = \texttt{0}^p\texttt{1}^p$ *can* be pumped. Let $s = xyz$, where $x = \texttt{0}$, $y = \texttt{0}$ and $z = \texttt{0}^{p-2}\texttt{1}^p$. The conditions are satisfied because
i) for any $i \geq 0$, $xy^iz = \texttt{00}^i\texttt{0}^{p-2}\texttt{1}^p$ is in $\texttt{0}^*\texttt{1}^*$.
ii) $|y| = 1 > 0$, and
iii) $|xy| = 2 \leq p$.

**1.31** We construct a DFA which alternately simulates the DFAs for $A$ and $B$, one step at a time. The new DFA keeps track of which DFA is being simulated. Let $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ be DFAs for $A$ and $B$. We construct the following DFA $M = (Q, \Sigma, \delta, s_0, F)$ for the perfect shuffle of $A$ and $B$.
i) $Q = Q_1 \times Q_2 \times \{1, 2\}$.
ii) For $q_1 \in Q_1$, $q_2 \in Q_2$, $b \in \{1, 2\}$, and $a \in \Sigma$:

$$\delta((q_1, q_2, b), a) = \begin{cases} (\delta_1(q_1, a), q_2, 2) & b = 1 \\ (q_1, \delta_1(q_2, a), 1) & b = 2. \end{cases}$$

iii) $s_0 = (s_1, s_2, 1)$.
iv) $F = \{(q_1, q_2, 1)|\ q_1 \in F_1 \text{ and } q_2 \in F_2\}$.

**1.32** We construct an NFA which simulates the DFAs for $A$ and $B$, nondeterministically switching back and forth from one to the other. Let $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ be DFAs for $A$ and $B$. We construct the following NFA $N = (Q, \Sigma, \delta, s_0, F)$ for the shuffle of $A$ and $B$.
i) $Q = Q_1 \times Q_2$.
ii) For $q_1 \in Q_1$, $q_2 \in Q_2$, and $a \in \Sigma$:

$$\delta((q_1, q_2), a) = \{(\delta_1(q_1, a), q_2), (q_1, \delta_2(q_2, a))\}.$$

iii) $s_0 = (s_1, s_2)$.
iv) $F = \{(q_1, q_2)|\ q_1 \in F_1 \text{ and } q_2 \in F_2\}$.

**1.33** Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes $A$. Then we construct NFA $N = (Q', \Sigma, \delta', q_0', F')$ recognizing *DROP-OUT*$(A)$. The idea behind the construction is that $N$ simulates $M$ on its input, nondeterministically guessing the point at which the dropped out symbol occurs. At that point $N$ guesses the symbol to insert in that place, without reading any actual input symbol at that step. Afterwards, it continues to simulate $M$.

We implement this idea in $N$ by keeping two copies of $M$, called the top and bottom copies. The start state is the start state of the top copy. The accept states of $N$ are the accept states of the bottom copy. Each copy contains the edges that would occur in $M$. Additionally, include $\varepsilon$ edges from each state $q$ in the top copy to every state in the bottom copy that $q$ can reach.

We describe $N$ formally. The states in the top copy are written with a T and the bottom with a B, thus: $(\mathrm{T}, q)$ and $(\mathrm{B}, q)$.

i) $Q' = \{\mathrm{T}, \mathrm{B}\} \times Q$,

ii) $q'_0 = (\mathrm{T}, q_0)$,

iii) $F' = \{\mathrm{B}\} \times F$,

iv) $\delta'((\mathrm{T}, q), a) = \begin{cases} \{(\mathrm{T}, \delta(q, a))\} & a \in \Sigma \\ \{(\mathrm{B}, \delta(q, b)) \mid b \in \Sigma\} & a = \varepsilon \end{cases}$

$$\delta'((\mathrm{B}, q), a) = \begin{cases} \{(\mathrm{B}, \delta(q, a))\} & a \in \Sigma \\ \emptyset & a = \varepsilon \end{cases}$$

**1.35** Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes $A$. We construct a new DFA $M' = (Q, \Sigma, \delta, q_0, F')$ that recognizes $A/B$. Automata $M$ and $M'$ differ only in the sets of accept states. Let $F' = \{r \mid$ starting at $r$ and reading a string in $B$ we get to an accept state of $M\}$. Thus $M'$ accepts a string $w$ iff there is a string $x \in B$ where $M$ accepts $wx$. Hence $M'$ recognizes $A/B$.

**1.36** For any regular language $A$, let $M_1$ be the DFA recognizing it. We need to find a DFA that recognizes $A^R$. Since any NFA can be converted to an equivalent DFA, it suffices to find an NFA $M_2$ that recognizes $A^R$.

We keep all the states in $M_1$ and reverse the direction of all the arrows in $M_1$. We set the accept state of $M_2$ to be the start state in $M_1$. Also, we introduce a new state $q_0$ as the start state for $M_2$ which goes to every accept state in $M_1$ by an $\epsilon$-transition.

**1.39** The idea is that we start by comparing the most significant bit of the two rows. If the bit in the top row is bigger, we know that the string is in the language. The string does not belong to the language if the bit in the top row is smaller. If the bits on both rows are the same, we move on to the next most significant bit until a difference is found. We implement this idea with a DFA having states $q_0$, $q_1$, and $q_2$. State $q_0$ indicates the result is not yet determined. States $q_1$ and $q_2$ indicate the top row is known to be larger, or smaller, respectively. We start with $q_0$. If the top bit in the input string is bigger, it goes to $q_1$, the only accept state, and stays there till the end of the input string. If the top bit in the input string is smaller, it goes to $q_2$ and stays there till the end of the input string. Otherwise, it stays in state $q_0$.

**1.40** Assume language $E$ is regular. Use the pumping lemma to a get a pumping length $p$ satisfying the conditions of the pumping lemma. Set $s = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]^p \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]^p$. Obviously, $s \in E$ and $|s| \geq p$. Thus, the pumping lemma implies that the string $s$ can be written as $xyz$ with $x = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]^a, y = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]^b, z = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]^c \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]^p$, where $b \geq 1$ and $a + b + c = p$. However, the string $s' = xy^0z = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]^{a+c} \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]^p \notin E$, since $a + c < p$. That contradicts the pumping lemma. Thus $E$ is not regular.

**1.41** For each $n \geq 1$, we build a DFA with the $n$ states $q_0, q_1, \ldots, q_{n-1}$ to count the number of consecutive a's modulo $n$ read so far. For each character a that is input, the counter increments by 1 and jumps to the next state in $M$. It accepts the string if and only if the machine stops at $q_0$. That means the length of the string consists of all a's and its length is a multiple of $n$.

More formally, the set of states of $M$ is $Q = \{q_0, q_1, \ldots, q_{n-1}\}$. The state $q_0$ is the start state and the only accept state. Define the transition function as: $\delta(q_i, \mathtt{a}) = q_j$ where $j = i + 1 \bmod n$.

**1.42** By simulating binary division, we create a DFA $M$ with $n$ states that recognizes $C_n$. $M$ has $n$ states which keep track of the $n$ possible remainders of the division process. The start state is the only accept state and corresponds to remainder 0.

The input string is fed into $M$ starting from the most significant bit. For each input bit, $M$ doubles the remainder that its current state records, and then adds the input bit. Its new state is the sum modulo $n$. We double the remainder because that corresponds to the left shift of the computed remainder in the long division algorithm. If an input string ends at the accept state (corresponding to remainder 0), the binary number has no remainder on division by $n$ and is therefore a member of $C_n$.

The formal definition of $M$ is $(\{q_0, \ldots, q_{n-1}\}, \{\mathtt{0}, \mathtt{1}\}, \delta, q_0, \{q_0\})$. For each $q_i \in Q$ and $b \in \{\mathtt{0}, \mathtt{1}\}$, define $\delta(q_i, b) = q_j$ where $j = (2i + b) \bmod n$.

**1.43** Use the same construction given in the proof of Theorem 1.39, which shows the equivalence of NFAs and DFAs. We need only change $F'$, the set of accept states of the new DFA. Here we let $F' = \mathcal{P}(F)$. The change means that the new DFA accepts only when *all* of the possible states of the all-NFA are accepting.

**1.44** Let $A_k = \Sigma^* \mathtt{0}^{k-1}\mathtt{0}^*$. A DFA with $k$ states $\{q_0, \ldots, q_{k-1}\}$ can recognize $A_k$. The start state is $q_0$. For each $i$ from 0 to $k-2$, state $q_i$ branches to $q_{i+1}$ on 0 and to $q_0$ on 1. State $q_{k-1}$ is the accept state and branches to itself on 0 and to $q_0$ on 1.

In any DFA with fewer than $k$ states, two of the $k$ strings 1, 10, ... , $\mathtt{10}^{k-1}$ must cause the machine to enter the same state, by the pigeon hole principle. But then, if we add to both of these strings enough 0s to cause the longer of these two strings to have exactly $k - 1$ 0s, the two new strings will still cause the machine to enter the same state, but one of these strings is in $A_k$ and the other is not. Hence, the machine must fail to produce the correct accept/reject response on one of these strings.

**1.45** **b.** Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA recognizing $A$, where $A$ is some regular language. We construct $M' = (Q', \Sigma, \delta, q_0', F')$ recognizing $NOEXTEND(A)$ as follows:

i) $Q' = Q$
ii) $\delta' = \delta$
iii) $q_0' = q_0$
iv) $F' = \{q |\ q \in F$ and there is no path of length $\geq 1$ from $q$ to an accept state$\}$.

**1.47** To show that $\equiv_L$ is an equivalence relation we show it is reflexive, symmetric, and transitive. It is reflexive because no string can distinguish $x$ from itself and hence $x \equiv_L x$ for every $x$. It is symmetric because $x$ is distinguishable from $y$ whenever $y$ is distinguishable from $x$. It is transitive because if $w \equiv_L x$ and $x \equiv_L y$, then for each $z$, $wz \in L$ iff $xz \in L$ and $xz \in L$ iff $yz \in L$, hence $wz \in L$ iff $yz \in L$, and so $w \equiv_L y$.

**1.49** **a.** $F$ is not regular, because the nonregular language $\{\mathtt{ab}^n\mathtt{c}^n \mid n \geq 0\}$ is the same as $F \cap \mathtt{ab}^*\mathtt{c}^*$, and the regular languages are closed under intersection.

**b.** Language $F$ satisfies the conditions of the pumping lemma using pumping length 2. If $s \in F$ is of length 2 or more we show that it can be pumped by considering four cases, depending on the number of $\mathtt{a}$'s that $s$ contains.

i) If $s$ is of the form $\mathtt{b}^*\mathtt{c}^*$, let $x = \varepsilon$, $y$ be the first symbol of $s$, and let $z$ be the rest of $s$.

ii) If $s$ is of the form $\mathtt{ab}^*\mathtt{c}^*$, let $x = \varepsilon$, $y$ be the first symbol of $s$, and let $z$ be the rest of $s$.

iii) If $s$ is of the form $\mathtt{aab}^*\mathtt{c}^*$, let $x = \varepsilon$, $y$ be the first two symbols of $s$, and let $z$ be the rest of $s$.

iv) If $s$ is of the form $\mathtt{aaa}^*\mathtt{b}^*\mathtt{c}^*$, let $x = \varepsilon$, $y$ be the first symbol of $s$, and let $z$ be the rest of $s$.

In each case, the strings $xy^iz$ are members of $F$ for every $i \geq 0$. Hence $F$ satisfies the conditions of the pumping lemma.

**c.** The pumping lemma is not violated because it states only that regular languages satisfy the three conditions, and it doesn't state that nonregular languages fail to satisfy the three conditions.

**1.50** The objective of this problem is for the student to pay close attention to the exact formulation of the pumping lemma.

**c.** This language is that same as the language in in part (b), so the solution is the same.

**e.** The minimum pumping length is 1. The pumping length cannot be 0, as in part (b). Any string in $(\mathtt{01})^*$ of length 1 or more contains $\mathtt{01}$ and hence can be pumped by dividing it so that $x = \varepsilon$, $y = \mathtt{01}$, and $z$ is the rest.

**f.** The minimum pumping length is 1. The pumping length cannot be 0, as in part (b). The language has no strings of length 1 or more so 1 is a pumping length. (the conditions hold vacuously).

**g.** The minimum pumping length is 3. The string $\mathtt{00}$ is in the language and it cannot be pumped, so the minimum pumping length cannot be 2. Every string in the language of length 3 or more contains a $\mathtt{1}$ within the first 3 symbols so it can be pumped by letting $y$ be that $\mathtt{1}$ and letting $x$ be the symbols to the left of $y$ and $z$ be the symbols to the right of $y$.

**h.** The minimum pumping length is 4. The string $\mathtt{100}$ is in the language but it cannot be pumped (down), therefore 3 is too small to be a pumping length. Any string of length 4 or more in the language must be of the form $xyz$ where $x$ is $\mathtt{10}$, $y$ is in $\mathtt{11}^*\mathtt{0}$ and $z$ is in $(\mathtt{11}^*\mathtt{0})^*\mathtt{0}$, which satisfies all of the conditions of the pumping lemma.

**i.** The minimum pumping length is 5. The string $\mathtt{1011}$ is in the language and it cannot be pumped. Every string in the language of length 5 or more (there aren't any) can be pumped (vacuously).

**j.** The minimum pumping length is 1. It cannot be 0 as in part (b). Every other string can be pumped, so 1 is a pumping length.

**1.51** **a.** Assume $L = \{\mathtt{0}^n\mathtt{1}^m\mathtt{0}^n \mid m, n \geq 0\}$ is regular. Let $p$ be the pumping length given by the pumping lemma. The string $s = \mathtt{0}^p\mathtt{1}\mathtt{0}^p \in L$, and $|s| \geq p$. Thus the pumping lemma implies that $s$ can be divided as $xyz$ with $x = \mathtt{0}^a$, $y = \mathtt{0}^b$, $z = \mathtt{0}^c\mathtt{1}\mathtt{0}^p$, where $b \geq 1$ and $a + b + c = p$. However, the string $s' = xy^0z = \mathtt{0}^{a+c}\mathtt{1}\mathtt{0}^p \notin L$, since $a + c < p$. That contradicts the pumping lemma.

**c.** Assume $C = \{w |\, w \in \{\mathtt{0},\mathtt{1}\}^*$ is a palindrome$\}$ is regular. Let $p$ be the pumping length given by the pumping lemma. The string $s = \mathtt{0}^p\mathtt{1}\mathtt{0}^p \in C$ and $|s| \geq p$. Follow the argument as in part (a). Hence $C$ isn't regular, so neither is its complement.

**1.52** One short solution is to observe that $\overline{Y} \cap \mathtt{1}^*\mathtt{\#}\mathtt{1}^* = \{\mathtt{1}^n\mathtt{\#}\mathtt{1}^n |\, n \geq 0\}$. This language is clearly not regular, as may be shown using a straightforward application of the pumping lemma. However, if $Y$ were regular, this language would be regular, too, because the class of regular languages is closed under intersection and complementation. Hence $Y$ isn't regular.

Alternatively, we can show $Y$ isn't regular directly using the pumping lemma. Assume to the contrary that $Y$ is regular and obtain its pumping length $p$. Let $s = \mathtt{1}^{p!}\mathtt{\#}\mathtt{1}^{2p!}$. The pumping lemma says that $s = xyz$ satisfying the three conditions. By condition 3, $y$ appears among the left-hand 1s. Let $l = |y|$ and let $k = (p!/l)$. Observe that $k$ is an integer, because $l$ must be a divisor of $p!$. Therefore, adding $k$ copies of $y$ to $s$ will add $p!$ additional 1s to the left-hand 1s. Hence, $xy^{1+k}z = \mathtt{1}^{2p!}\mathtt{\#}\mathtt{1}^{2p!}$ which isn't a member of $Y$. But condition 1 of the pumping lemma states that this string is a member of $Y$, a contradiction.

**1.53** The language $D$ can be written alternatively as $\mathtt{0}\Sigma^*\mathtt{0} \cup \mathtt{1}\Sigma^*\mathtt{1} \cup \mathtt{0} \cup \mathtt{1} \cup \varepsilon$, which is obviously regular.

**1.54** The NFA $N_k$ guesses when it has read an a that appears at most $k$ symbols from the end, then counts $k-1$ more symbols and enters an accept state. It has an initial state $q_0$ and additional states $q_1$ thru $q_k$. State $q_0$ has transitions on both a and b back to itself and on a to state $q_1$. For $1 \leq i \leq k-1$, state $q_i$ has transitions on a and b to state $q_{i+1}$. State $q_k$ is an accept state with no transition arrows coming out of it. More formally, NFA $N_k = (Q, \Sigma, \delta, q_0, F)$ where

i) $Q = \{q_0, \dots, q_k\}$

ii) $\delta(q,c) = \begin{cases} \{q_0\} & q = q_0 \text{ and } c = \mathtt{a} \\ \{q_0, q_1\} & q = q_0 \text{ and } c = \mathtt{b} \\ \{q_{i+1}\} & q = q_i \text{ for } 1 \leq i < k \text{ and } c \in \Sigma \\ \emptyset & q = q_k \text{ or } c = \varepsilon \end{cases}$

iii) $F = \{q_k\}$.

**1.55** Let $M$ be a DFA. Say that $w$ *leads* to state $q$ if $M$ is in $q$ after reading $w$. Notice that if $w_1$ and $w_2$ lead to the same state, then $w_1p$ and $w_2p$ also lead to the same state, for all strings $p$.

Assume that $M$ recognizes $C_k$ with fewer than $2^k$ states, and derive a contradiction. There are $2^k$ different strings of length $k$. By the pigeonhole principle, two of these strings $w_1$ and $w_2$ lead to the same state of $M$.

Let $i$ be the index of the first bit on which $w_1$ and $w_2$ differ. Since $w_1$ and $w_2$ lead $M$ to the same state, $w_1\mathtt{b}^{i-1}$ and $w_2\mathtt{b}^{i-1}$ lead $M$ to the same state. This cannot be the case, however, since one of the strings should be rejected and the other accepted. Therefore, any two distinct $k$ bit strings lead to different states of $M$. Hence $M$ has at least $2^k$ states.

**1.60** **a.** We construct $M'$ from $M$ by converting each transition that is traversed on symbol $a \in \Sigma$ to a sequence of transitions that are traversed while reading string $f(a) \in \Gamma^*$. The formal construction follows.

Let $M = (Q, \Sigma, \delta, q_0, F)$. For each $a \in \Sigma$ let $z^a = f(a)$ and let $k_a = |z^a|$. We write $z^a = z_1^a z_2^a \ldots z_{k_a}^a$ where $z_i^a \in \Gamma$. Construct $M' = (Q', \Gamma, \delta', q_0, F)$. $Q' = Q \cup \{q_i^a |\, q \in Q, a \in \Sigma, 1 \leq i \leq k_a\}$

For every $q \in Q$,

$$\delta'(q, b) = \begin{cases} \{r |\, \delta(q, a) = r \text{ and } z^a = \varepsilon\} & b = \varepsilon \\ \{q_1^a |\, b = z_1^a\} & b \neq \varepsilon \end{cases}$$

$$\delta'(q_i^a, b) = \begin{cases} \{q_{i+1}^a |\, b = z_{i+1}^a\} & 1 \leq i < k_a \text{ and } b \neq \varepsilon \\ \{r |\, \delta(q, a) = r\} & i = k_a \text{ and } b = \varepsilon \end{cases}$$

**b.** The above construction shows that the class of regular languages is closed under homomorphism. To show that the class of non-regular languages is not closed under homomorphism, let $B = \{0^m 1^m |\, m \geq 0\}$ and let $f\colon \{0{,}1\} \longrightarrow \{2\}$ be the homomorphism where $f(0) = f(1) = 2$. We know that $B$ is a non-regular language but $f(B) = \{2^{2m} |\, m \geq 0\}$ is a regular language.

**1.61** **a.** $RC(A) = \{w_{i+1} \cdots w_n w_1 \cdots w_i |\, w = w_1 \cdots w_n \in A \text{ and } 1 \leq i \leq n\}$, because we can let $x = w_1 \cdots w_i$ and $y = w_{i+1} \cdots w_n$. Then $RC(RC(A))$ gives the same language because if $st = w_{i+1} \cdots w_n w_1 \cdots w_i$ for some strings $s$ and $t$, then $ts = w_{j+1} \cdots w_n w_1 \cdots w_j$ for some $j$ where $1 \leq j \leq n$.

**b.** Let $A$ be a regular language that is recognized by DFA $M$. We construct a NFA $N$ that recognizes $RC(A)$. In the idea behind the construction, $N$ guesses the cut point nondeterministically by starting $M$ at any one of its states. Then $N$ simulates $M$ on the input symbols it reads. If $N$ finds that $M$ is in one of its accept states then $N$ may nondeterministically reset $M$ back to it's start state prior to reading the next symbol. Finally, $N$ accepts its input if the simulation ends in the same state it started in, and exactly one reset occurred during the simulation. Here is the formal construction. Let $M = (Q, \Sigma, \delta, q_0, F)$ recognize $A$ and construct $N = (Q', \Sigma, \delta', r, F')$ to recognize $RC(A)$.

Set $Q' = (Q \times Q \times \{1, 2\}) \cup \{r\}$. State $(q, r, i)$ signifies that $N$ started the simulation in $M$'s state $q$, it is currently simulating $M$ in state $r$, and if $i = 1$ a reset hasn't yet occurred whereas if $i = 2$ then a reset has occurred.

Set $\delta'(r, \varepsilon) = \{(q, q, 1) |\, q \in Q\}$. This starts simulating $M$ in each of its states, nondeterministically.

Set $\delta'((q, r, i), a) = \{(q, \delta(r, a), i)\}$ for each $q, r \in Q$ and $i \in \{1, 2\}$. This continues the simulation.

Set $\delta'((q, r, 1), \varepsilon) = \{(q, q_0, 2)\}$ for $r \in F$ and $q \in Q$. This allows $N$ to reset the simulation to $q_0$ if $M$ hasn't yet been reset and $M$ is currently in an accept state.

We set $\delta'$ to $\emptyset$ if it is otherwise unset.

$F' = \{(q, q, 2) |\, q \in Q\}$.

**1.62** Assume to the contrary that $ADD$ is regular. Let $p$ be the pumping length given by the pumping lemma. Choose $s$ to be the string $1^p$=0+$1^p$, which is a member of $ADD$. Because $s$ has length greater than $p$, the pumping lemma guarantees that $s$ can be split into three pieces, $s = xyz$, satisfying the conditions of the lemma. By the third condition in the pumping lemma have that $|xy| \leq p$, it follows that $y$ is $1^k$ for some $k \geq 1$. Then $xy^2z$ is the string $1^{p+k}$=0+$1^p$, which is not a member of $ADD$, violating the pumping lemma. Hence $ADD$ isn't regular.

**1.63** Let $A = \{2^k |\ k \geq 0\}$. Clearly $B_2(A) = \texttt{10}^*$ is regular. Use the pumping lemma to show that $B_3(A)$ isn't regular. Get the pumping length $p$ and chose $s \in B_3(A)$ of length $p$ or more. We show $s$ cannot be pumped. Let $s = xyz$. For string $w$, write $(w)_3$ to be the number that $w$ represents in base 3. Then

$$\lim_{i\to\infty} \frac{(xy^iz)_3}{(xy^i)_3} = 3^{|z|} \quad \text{and} \quad \lim_{i\to\infty} \frac{(xy^{i+1}z)_3}{(xy^i)_3} = 3^{|y|+|z|}$$

so

$$\lim_{i\to\infty} \frac{(xy^{i+1}z)_3}{(xy^iz)_3} = 3^{|y|}.$$

Therefore for a sufficiently large $i$,

$$\frac{(xy^{i+1}z)_3}{(xy^iz)_3} = 3^{|y|} \pm \alpha$$

for some $\alpha < 1$. But this fraction is a ratio of two members of $A$ and is therefore a whole number. Hence $\alpha = 0$ and the ratio is a power of 3. But the ration of two members of $A$ also is a power of 2. No number greater than 1 can be both a power of 2 and of 3, a contradiction.

**1.64** Given an NFA $M$ recognizing $A$ we construct an NFA $N$ accepting $A_{\frac{1}{2}-}$ using the following idea. $M$ keeps track of two states in $N$ using two "fingers". As it reads each input symbol, $N$ uses one finger to simulate $M$ on that symbol. Simultaneously, $M$ uses the other finger to run $M$ backwards from an accept state on a guessed symbol. $N$ accepts whenever the forward simulation and the backward simulation are in the same state, that is, whenever the two fingers are together. At those points we are sure that $N$ has found a string where another string of the same length can be appended to yield a member of $A$, precisely the definition of $A_{\frac{1}{2}-}$.

In the formal construction, Exercise 1.11 allows us to assume for simplicity that the NFA $M$ recognizing $A$ has a single accept state. Let $M = (Q, \Sigma, \delta, q_0, q_{\text{accept}})$. Construct NFA $N = (Q', \Sigma, \delta', q'_0, F')$ recognizing the first halves of the strings in $A$ as follows:

i) $Q' = Q \times Q$.

ii) For $q, r \in Q$ define
$\delta'((q,r), a) = \{(u,v)|\ u \in \delta(q,a) \text{ and } r \in \delta(v,b) \text{ for some } b \in \Sigma\}$.

iii) $q_0{}' = (q_0, q_{\text{accept}})$.

iv) $F' = \{(q,q)|\ q \in Q\}$.

**1.65** Let $A = \{\texttt{0}^*\texttt{\#}\texttt{1}^*\}$. Thus, $A_{\frac{1}{3}-\frac{1}{3}} \cap \{\texttt{0}^*\texttt{1}^*\} = \{\texttt{0}^n\texttt{1}^n|\ n \geq 0\}$. Regular sets are closed under intersection, and $\{\texttt{0}^n\texttt{1}^n|\ n \geq 0\}$ is not regular, so $A_{\frac{1}{3}-\frac{1}{3}}$ is not regular.

**1.66** If M has a synchronizing sequence, then for any pair of states $(p,q)$ there is a string $w_{p,q}$ such that $\delta(p, w_{p,q}) = \delta(q, w_{p,q}) = h$, where $h$ is the home state. Let us run two copies of $M$ starting at states $p$ and $q$ respectively. Consider the sequence of pairs of states $(u,v)$ that the two copies run through before reaching home state $h$. If some pair appears in the sequence twice, we can delete the substring of $w_{p,q}$ that takes the copies of $M$ from one occurrence of the pair to the other, and thus obtain a new $w_{p,q}$. We repeat the process until all pairs of states in the sequence are distinct. The number of distinct state pairs is $k^2$, so $|w_{p,q}| \leq k^2$.

Suppose we are running $k$ copies of $M$ and feeding in the same input string $s$. Each copy starts at a different state. If two copies end up at the same state after some step, they will do exactly the same thing for the rest of input, so we can get rid of one of them. If $s$ is a synchronizing sequence, we will end up with one copy of $M$ after feeding in $s$. Now we will show how to construct a synchronizing sequence $s$ of length at most $k^3$.

i) Start with $s = \epsilon$. Start $k$ copies of $M$, one at each of its states. Repeat the following two steps until we are left with only a single copy of $M$.

ii) Pick two of $M$'s remaining copies ($M_p$ and $M_q$) that are now in states $p$ and $q$ after reading $s$.

iii) Redefine $s = sw_{p,q}$. After reading this new $s$, $M_p$ and $M_q$ will be in the same state, so we eliminate one of these copies.

At the end of the above procedure, $s$ brings all states of $M$ to a single state. Call that state $h$. Stages 2 and 3 are repeated $k-1$ times, because after each repetition we eliminate one copy of $M$. Therefore $|s| \leq (k-1)k^2 < k^3$.

**1.67** Let $C = \Sigma^* B \Sigma^*$. Then $C$ is the language of all strings that contain some member of $B$ as a substring, If $B$ is regular then $C$ is also regular. We know from the solution to Problem 1.14 that the complement of a regular language is regular, and so $\overline{C}$ is regular. It is the language of all strings that do not contain some member of $B$ as a substring. Note that $A$ *avoids* $B = A \cap \overline{C}$. Therefore $A$ *avoids* $B$ is regular because we showed that the intersection of two regular languages is regular.

**1.68** **a.** Any string that doesn't begin and end with 0 obviously cannot be a member of $A$. If string $w$ does begin and end with 0 then $w = 0u0$ for some string $u$. Hence $A = 0\Sigma^*0$ and therefore $A$ is regular.

**b.** Assume for contradiction that $B$ is regular. Use the pumping lemma to get the pumping length $p$. Letting $s = 0^p10^p$ we have $s \in B$ and so we can divide up $s = xyz$ according to the conditions of the pumping lemma. By condition 3, $xy$ has only 0s, hence the string $xyyz$ is $0^l10^p$ for some $l > p$. But then $0^l10^p$ isn't equal to $0^k1u0^k$ for any $u$ and $k$, because the left-hand part of the string requires $k = l$ and the right-hand part requires $k \leq p$. Both together are impossible, because $l > p$. That contradicts the pumping lemma and we conclude that $B$ isn't regular.

**1.69** **a.** Let $s$ be a string in $U$ whose length is shortest among all strings in $U$. Assume (for contradiction) that $|s| \geq \max(k_1, k_2)$. One or both of the DFAs accept $s$ because $s \in U$. Say it is $M_1$ that accepts $s$. Consider the states $q_0, q_1, \ldots, q_l$ that $M_1$ enters while reading $s$, where $l = |s|$. We have $l \geq k_1$, so $q_0, q_1, \ldots, q_l$ must repeat some state. Remove the portion of $s$ between the repeated state to yield a shorter string that $M_1$ accepts. That string is in $U$, a contradiction. Thus $|s| < \max(k_1, k_2)$.

**b.** Let $s$ be a string in $\overline{U}$ whose length is shortest among all strings in $\overline{U}$. Assume (for contradiction) that $|s| \geq k_1k_2$. Both of the DFAs reject $s$ because $s \in \overline{U}$. Consider the states $q_0, q_1, \ldots, q_l$ and $r_0, r_1, \ldots, r_l$ that $M_1$ and $M_2$ enter respectively while reading $s$, where $l = |s|$. We have $l \geq k_1k_2$, so in the sequence of ordered pairs $(q_0, r_0), (q_1, r_1), \ldots, (q_l, r_l)$, some pair must repeat. Remove the portion of $s$ between the repeated pair to yield a shorter string that both $M_1$ and $M_2$ reject. That shorter string is in $\overline{U}$, a contradiction. Thus $|s| < k_1k_2$.

**1.70** A PDA $P$ that recognizes $\overline{C}$ operates by nondeterministically choosing one of three cases. In the first case, $P$ scans its input and accepts if it doesn't contain exactly two #s. In the second case, $P$ looks for a mismatch between the first two strings that are separated by #. It does so by reading its input while pushing those symbols onto the stack until it reads #. At that point $P$ continues reading input symbols and matching them with symbols that are popped off the stack. If a mismatch occurs, or if the stack empties before $P$ reads the next `#`, or if $P$ reads the next `#` before the stack empties, then it accepts. In the third case, $P$ looks for a mismatch between the last two strings that are separated by `#`. It does so by reading its input until it reads `#` and the it continues reading input symbols while pushing those symbols onto the stack until it reads a second `#`. At that point $P$ continues reading input symbols and matching them with symbols that are popped off the stack. If a mismatch occurs, or if the stack empties before $P$ reaches the end of the input or if $P$ reaches the end of the input before the stack empties, then it accepts.

Alternatively, here is a CFG that generates $\overline{C}$.

$$\begin{aligned}
A &\to YDY\mathtt{\#}Y \mid Y\mathtt{\#}YDY \mid Y \mid Y\mathtt{\#}Y \mid Y\mathtt{\#}Y\mathtt{\#}Y\mathtt{\#}Z\\
D &\to XDX \mid \mathtt{0}E\mathtt{1} \mid \mathtt{1}E\mathtt{0}\\
E &\to XEX \mid \mathtt{\#}\\
X &\to \mathtt{0} \mid \mathtt{1}\\
Y &\to XY \mid \varepsilon\\
Z &\to Y\mathtt{\#}Z \mid \varepsilon
\end{aligned}$$

**1.71** **a.** Observe that $B = \mathtt{1}\Sigma^*\mathtt{1}\Sigma^*$ and thus is clearly regular.

**b.** We show $C$ is nonregular using the pumping lemma. Assume $C$ is regular and let $p$ be its pumping length. Let $s = \mathtt{1}^p\mathtt{0}\mathtt{1}^p$. The pumping lemma says that $s = xyz$ satisfying the three conditions. Condition three says that $y$ appears among the left-hand `1`s. We pump down to obtain the string $xz$ which is not a member of $C$. Therefore $C$ doesn't satisfy the pumping lemma and hence isn't regular.

**1.72** **a.** Let $B = \{\mathtt{0110}\}$. Then $\mathit{CUT}(B) = \{\mathtt{0110}, \mathtt{1010}, \mathtt{0011}, \mathtt{1100}, \mathtt{1001}\}$ and $\mathit{CUT}(\mathit{CUT}(B)) = \{\mathtt{0110}, \mathtt{1010}, \mathtt{0011}, \mathtt{1100}, \mathtt{1001}, \mathtt{0101}\}$.

**b.** Let $A$ be a regular language that is recognized by DFA $M$. We construct a NFA $N$ that recognizes $\mathit{CUT}(A)$. This construction is similar to the construction in the solution to Problem 1.61. Here, $N$ begins by nondeterministically guessing two states $q_1$ and $q_2$ in $M$. Then $N$ simulates $M$ on its input beginning at state $q_1$. Whenever the simulation reaches $q_2$, nondeterministically $N$ may switch to simulating $M$ at its start state $q_0$ and if it reaches state $q_1$, it again nondeterministically may switch to state $q_2$. At the end of the input, if $N$'s simulation has made both switches and is now in one of $M$'s accept states, it has completed reading an input $yxz$ where $M$ accepts $xyz$, and so it accepts.

**1.73** **a.** The idea here is to show that a DFA with fewer than $2^k$ states must fail to give the right answer on at least one string. Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA with $m$ states. Fix the value of $k$ and let $W = \{w \mid w \in \Sigma^k\}$ be the set of strings of length $k$.

For each of the $2^k$ strings $w \in W$, let $q_w$ be the state that $A$ enters after it starts in state $q_0$ and then reads $w$. If $m < 2^k$ then two different strings $s$ and $t$ must exist in $W$ where $A$ enters the same state, i.e., $q_s = q_t$. The string $ss$ is in $WW$ so $A$ must enter an accepting state after reading $ss$. Similarly, $st \notin WW$ so $A$ must enter a rejecting string after reading $st$. But $q_s = q_t$, so $A$ enters the same state after reading $ss$ or $st$, a contradiction. Therefore $m \geq 2^k$.

**b.** We can give an NFA $N = (Q, \Sigma, \delta, c_0, F)$ with $4k + 4$ states that recognizes $\overline{WW}$. The NFA $N$ branches into two parts. One part accepts if the inputs length isn't $2k$. The other part accepts if the input contains two unequal symbols that are $k$ separated.

Formally, let $Q = \{c_0, \ldots, c_{2k+1}, r, y_1, \ldots, y_k, z_1, \ldots, z_k\}$ and let $F = \{c_0, \ldots, c_{2k-1}, c_{2k+1}, y_k, z_k\}$.

For $0 \leq i \leq 2k$ and $a \in \Sigma$, set $\delta(c_i, a) = \{c_{i+1}\}$ and $\delta(c_{k+1}, a) = \{c_{k+1}\}$.
Additionally, $\delta(c_0, \varepsilon) = \{r\}$, $\delta(r, \mathtt{0}) = \{r, y_1\}$ and $\delta(r, \mathtt{1}) = \{r, z_1\}$.
Finally, for $0 \leq i < k-1$, set $\delta(y_i, a) = \{y_{i+1}\}$ and $\delta(z_i, a) = \{z_{i+1}\}$,
$\delta(y_{k-1}, \mathtt{1}) = \{y_k\}$, $\delta(y_k, a) = \{y_k\}$, and
$\delta(z_{k-1}, \mathtt{1}) = \{z_k\}$, $\delta(z_k, a) = \{z_k\}$.

# Chapter 2

**2.1** Here are the derivations (but not the parse trees).

**a.** $E \Rightarrow T \Rightarrow F \Rightarrow \texttt{a}$

**b.** $E \Rightarrow E\texttt{+}T \Rightarrow T\texttt{+}T \Rightarrow F\texttt{+}T \Rightarrow F\texttt{+}F \Rightarrow \texttt{a+}F \Rightarrow \texttt{a+a}$

**c.** $E \Rightarrow E\texttt{+}T \Rightarrow E\texttt{+}T\texttt{+}T \Rightarrow T\texttt{+}T\texttt{+}T \Rightarrow F\texttt{+}T\texttt{+}T \Rightarrow F\texttt{+}F\texttt{+}T \Rightarrow F\texttt{+}F\texttt{+}F \Rightarrow \texttt{a+}F\texttt{+}F \Rightarrow \texttt{a+a+}F \Rightarrow \texttt{a+a+a}$

**d.** $E \Rightarrow T \Rightarrow F \Rightarrow \texttt{(}E\texttt{)} \Rightarrow \texttt{(}T\texttt{)} \Rightarrow \texttt{(}F\texttt{)} \Rightarrow \texttt{((}E\texttt{))} \Rightarrow \texttt{((}T\texttt{))} \Rightarrow \texttt{((}F\texttt{))} \Rightarrow \texttt{((a))}$

**2.2** **a.** The following grammar generates $A$:

$$\begin{aligned} S &\to RT \\ R &\to \texttt{a}R \mid \varepsilon \\ T &\to \texttt{b}T\texttt{c} \mid \varepsilon \end{aligned}$$

The following grammar generates $B$:

$$\begin{aligned} S &\to TR \\ T &\to \texttt{a}T\texttt{b} \mid \varepsilon \\ R &\to \texttt{c}R \mid \varepsilon \end{aligned}$$

Both $A$ and $B$ are context-free languages and $A \cap B = \{\texttt{a}^n\texttt{b}^n\texttt{c}^n \mid n \geq 0\}$. We know from Example 2.36 that this language is not context free. We have found two CFLs whose intersection is not context free. Therefore the class of context-free languages is not closed under intersection.

**b.** First, the context-free languages are closed under the union operation. Let $G_1 = (V_1, \Sigma, R_1, S_1)$ and $G_2 = (V_2, \Sigma, R_2, S_2)$ be two arbitrary context free grammars. We construct a grammar $G$ that recognizes their union. Formally, $G = (V, \Sigma, R, S)$ where:

i) $V = V_1 \cup V_2$

ii) $R = R_1 \cup R_2 \cup \{S \to S_1,\ S \to S_2\}$
(Here we assume that $R_1$ and $R_2$ are disjoint, otherwise we change the variable names to ensure disjointness)

Next, we show that the CFLs are not closed under complementation. Assume, for a contradiction, that the CFLs are closed under complementation. Then, if $G_1$ and $G_2$ are context free grammars, it would follow that $\overline{L(G_1)}$ and $\overline{L(G_2)}$ are context free. We previously showed that context-free languages are closed under union and so $\overline{L(G_1)} \cup \overline{L(G_1)}$ is context free. That, by our assumption, implies that $\overline{\overline{L(G_1)} \cup \overline{L(G_1)}}$ is context free. But by DeMorgan's laws, $\overline{\overline{L(G_1)} \cup \overline{L(G_1)}} = L(G_1) \cap L(G_2)$. However, if $G_1$ and

$G_2$ are chosen as in part (a), $\overline{\overline{L(G_1)} \cup \overline{L(G_1)}}$ isn't context free. This contradiction shows that the context-free languages are not closed under complementation.

**2.4** **b.** $S \to \mathtt{0}R\mathtt{0} \mid \mathtt{1}R\mathtt{1} \mid \varepsilon$
$R \to \mathtt{0}R \mid \mathtt{1}R \mid \varepsilon$

**c.** $S \to \mathtt{0} \mid \mathtt{1} \mid \mathtt{00}S \mid \mathtt{01}S \mid \mathtt{10}S \mid \mathtt{11}S$

**e.** $S \to \mathtt{0}S\mathtt{0} \mid \mathtt{1}S\mathtt{1} \mid \mathtt{0} \mid \mathtt{1} \mid \varepsilon$

**f.** $S \to S$

**2.5** **a.** This language is regular, so the PDA doesn't need to use its stack. The PDA reads the input and uses its finite control to maintain a counter which counts up to 3. It keeps track of the number of 1s it has seen using this counter. The PDA enters an accept state and scans to the end of the input if it has read three 1s.

**b.** This language is regular. The PDA reads the input and keeps track of the first and last symbol in its finite control. If they are the same, it accepts, otherwise it rejects.

**c.** This language is regular. The PDA reads the input and keeps track of the length (modulo 2) using its finite control. If the length is 1 (modulo 2) it accepts, otherwise it rejects.

**d.** The PDA reads the input and pushes the symbols onto the stack. At some point it nondeterministically guesses where the middle is. It looks at the middle symbol. If that symbol is a 1, it rejects. If it is a 0 the PDA reads the rest of the string, and for each character read, it pops one element off of its stack. If the stack is empty when it finishes reading the input, it accepts. If the stack is empty before it reaches the end of the input, or nonempty when the input is finished, it rejects.

**e.** The PDA reads the input and pushes each symbol onto its stack. At some point it nondeterministically guesses when it has reached the middle. It also nondeterministically guesses whether string has odd length or even length. If it guesses even, it pushes the current symbol it's reading onto the stack. If it guesses the string has odd length, it goes to the next input symbol without changing the stack. Then it reads the rest of the input, and it compares each symbol it reads to the symbol on the top of the stack. If they are the same, it pops the stack, and continues reading. If they are different, it rejects. If the stack is empty when it finishes reading the input, it accepts. If the stack is empty before it reaches the end of the input, or nonempty when the input is finished, it rejects.

**f.** The PDA never enters an accept state.

**2.6** **b.**

$$\begin{aligned} S &\to X\mathtt{b}X\mathtt{a}X \mid T \mid U \\ T &\to \mathtt{a}T\mathtt{b} \mid T\mathtt{b} \mid \mathtt{b} \\ U &\to \mathtt{a}U\mathtt{b} \mid \mathtt{a}U \mid \mathtt{a} \\ X &\to \mathtt{a}X \mid \mathtt{b}X \mid \varepsilon \end{aligned}$$

**d.**

$$\begin{aligned} S &\to M\mathtt{\#}P\mathtt{\#}M \mid P\mathtt{\#}M \mid M\mathtt{\#}P \mid P \\ P &\to \mathtt{a}P\mathtt{a} \mid \mathtt{b}P\mathtt{b} \mid \mathtt{a} \mid \mathtt{b} \mid \varepsilon \mid \mathtt{\#} \mid \mathtt{\#}M\mathtt{\#} \\ M &\to \mathtt{a}M \mid \mathtt{b}M \mid \mathtt{\#}M \mid \varepsilon \end{aligned}$$

Note that we need to allow for the case when $i = j$, that is, some $x_i$ is a palindrome. Also, $\varepsilon$ is in the language since it's a palindrome.

**2.9** A CFG $G$ that generates $A$ is given as follows:

$G = (V, \Sigma, R, S)$, $V = \{S, E_{\texttt{ab}}, E_{\texttt{bc}}, C, A\}$, and $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$. The rules are:

$$\begin{aligned} S &\to E_{\texttt{ab}}C \mid AE_{\texttt{bc}} \\ E_{\texttt{ab}} &\to \texttt{a}E_{\texttt{ab}}\texttt{b} \mid \varepsilon \\ E_{\texttt{bc}} &\to \texttt{b}E_{\texttt{bc}}\texttt{c} \mid \varepsilon \\ C &\to C\texttt{c} \mid \varepsilon \\ A &\to A\texttt{a} \mid \varepsilon \end{aligned}$$

Initially substituting $E_{\texttt{ab}}C$ for $S$ generates any string with an equal number of a's and b's followed by any number of c's. Initially substituting $E_{\texttt{bc}}$ for $S$ generates any string with an equal number of b's and c's prepended by any number of a's.

The grammar is ambiguous. Consider the string $\varepsilon$. On the one hand, it can be derived by choosing $E_{\texttt{ab}}$C with each of $E_{\texttt{ab}}$ and $C$ yielding $\varepsilon$. On the other hand, $\varepsilon$ can be derived by choosing $AE_{\texttt{bc}}$ with each of $A$ and $E_{\texttt{bc}}$ yielding $\varepsilon$. In general, any string $\texttt{a}^i\texttt{b}^j\texttt{c}^k$ with $i = j = k$ can be derived ambiguously in this grammar.

**2.10**

1. Nondeterministically branch to either Stage 2 or Stage 6.
2. Read and push a's.
3. Read b's, while popping a's.
4. If b's finish when stack is empty, skip c's on input and *accept*.
5. Skip a's on input.
6. Read and push b's.
7. Read c's, while popping b's.
8. If c's finish when stack is empty, *accept*.

**2.11**

1. Place \$ and $E$ on the stack.
2. Repeat the following steps forever.
3. If the top of stack is the variable $E$, pop it and nondeterministically push either $E$+$T$ or $T$ into the stack.
4. If the top of stack is the variable $T$, pop it and nondeterministically push either $T{\times}F$ or $F$ into the stack.
5. If the top of stack is the variable $F$, pop it and nondeterministically push either ($E$) or a into the stack.
6. If the top of stack is a terminal symbol, read the next symbol from the input and compare it to the terminal in the stack. If they match, repeat. If they do not match, reject on this branch of the nondeterminism.
7. If the top of stack is the symbol \$, enter the accept state. Doing so accepts the input if it has all been read.

**2.12** Informal description of a PDA that recognizes the CFG in Exercise 2.3:

1. Place the marker symbol \$ and the start variable $R$ on the stack.
2. Repeat the following steps forever.
3. If the top of stack is the variable $R$, pop it and nondeterministically push either $XRX$ or $S$ into the stack.
4. If the top of stack is the variable $S$, pop it and nondeterministically push either a$T$b or b$T$a into the stack.
5. If the top of stack is the variable $T$, pop it and nondeterministically push either $XTX$, $X$ or $\varepsilon$ into the stack.
6. If the top of stack is the variable $X$, pop it and nondeterministically push either a or b into the stack.

**7.** If the top of stack is a terminal symbol, read the next symbol from the input and compare it to the terminal symbol in the stack. If they match, repeat. If they do not match, reject on this branch of the nondeterminism.

**8.** If the top of stack is the symbol $, enter the accept state. Doing so accepts the input if it has all been read.

**2.13** **a.** $L(G)$ is the language of strings of 0s and #s that either contain exactly two #s and any number of 0s, or contain exactly one # and the number of 0s on the right-hand side of the # is twice the number of 0s on the left-hand side of the #.

**b.** Assume $L(G)$ is regular and obtain a contradiction. Let $A = L(G) \cap \mathtt{0}^*\#\mathtt{0}^*$. If $L(G)$ is regular, so is $A$. But we can show $A = \{\mathtt{0}^k\#\mathtt{0}^{2k} \mid k \geq 0\}$ is not regular by using a standard pumping lemma argument.

**2.14**

$$\begin{aligned} S_0 &\to AB \mid CC \mid BA \mid BD \mid BB \mid \varepsilon \\ A &\to AB \mid CC \mid BA \mid BD \mid BB \\ B &\to CC \\ C &\to \mathtt{0} \\ D &\to AB \end{aligned}$$

**2.16** Let $G_1 = (V_1, \Sigma, R_1, S_1)$ and $G_2 = (V_2, \Sigma, R_2, S_2)$ be CFGs.

Construct a new CFG $G_\cup$ where $L(G_\cup) = L(G_1) \cup L(G_2)$. Let $S$ be a new variable that is neither in $V_1$ nor in $V_2$ and assume these sets are disjoint.

Let $G_\cup = (V_1 \cup V_2 \cup \{S\},\ \Sigma,\ R_1 \cup R_2 \cup \{r_0\},\ S)$, where $r_0$ is $S \to S_1 \mid S_2$.

We construct CFG $G_\circ$ that generates $L(G_1) \circ L(G_2)$ as in $G_\cup$ by changing $r_0$ in $G_\cup$ into $S \to S_1S_2$.

To construct CFG $G_*$ that generates the language $L(G_1)^*$, let $S'$ be a new variable not in $V_1$ and make it the starting variable. Let $r_0$ be $S' \to S'S_1 \mid \varepsilon$ be a new rule in $G_*$.

**2.17** Let $A$ be a regular language generated by regular expression $R$. If $R$ is one of the atomic regular expressions $b$, for $b \in \Sigma_\varepsilon$, construct the equivalent CFG $(\{S\}, \{b\}, \{S \to b\}, S)$. If $R$ is the atomic regular expressions $\emptyset$, construct the equivalent CFG $(\{S\}, \{b\}, \{S \to S\}, S)$. If $R$ is a regular expression composed of smaller regular expressions combined with a regular operation, use the result of Problem 2.16 to construct an equivalent CFG out of the CFGs that are equivalent to the smaller expressions.

**2.18** $S$ can generate a string of $T$s. Each $T$ can generate strings in $\{\mathtt{a}^m b^m \mid m \geq 1\}$. Here are two different leftmost derivations of ababab.
$S \Rightarrow SS \Rightarrow SSS \Rightarrow TSS \Rightarrow \mathtt{ab}SS \Rightarrow \mathtt{ab}TS \Rightarrow \mathtt{abab}S \Rightarrow \mathtt{abab}T \Rightarrow \mathtt{ababab}$.
$S \Rightarrow SS \Rightarrow TS \Rightarrow \mathtt{ab}S \Rightarrow \mathtt{ab}SS \Rightarrow \mathtt{ab}TS \Rightarrow \mathtt{abab}S \Rightarrow \mathtt{abab}T \Rightarrow \mathtt{ababab}$.
The ambiguity arises because $S$ can generate a string of $T$s in multiple ways. We can prevent this behavior by forcing $S$ to generate each string of $T$s in a single way by changing the first rule to be $S \to TS$ instead of $S \to SS$. In the modified grammar, a leftmost derivation will repeatedly expand $T$ until it is eliminated before expanding $S$, and no other option for expanding variables is possible, so only one leftmost derivation is possible for each generated string.

**2.19** Let $A$ be a CFG that is recognized by PDA $P$ and construct the following PDA $P'$ that recognizes $RC(A)$. For simplicity, assume that $P$ empties its stack when it accepts its input. The new PDA $P'$ must accept input $yx$ when $P$ accepts $xy$. Intuitively, would like $P'$ to start by guessing the state and the stack contents that $P$ would be in after reading $x$, so that $P'$ can simulate $P$ on $y$, and then if $P$ ends up in an accept state after reading $y$, we can simulate $P$ on $x$ to check that it ends up with the initially guessed start and stack contents. However, $P'$ has no way to store the initially guessed stack contents to check that it matches the stack after reading $x$. To avoid this difficulty, $P'$ guesses these stack symbols only at the point when $P$ would be popping them while reading $y$. It records these guesses by pushing a specially marked copy of each guessed symbol. When $P'$ guesses that it has finished reading $y$, it will have a copy of the stack that $P$ would have when it finishes reading $x$ but in reverse order, with marked symbols instead of regular symbols. Then, while $P'$ simulates $P$ reading $x$, whenever $P$ pushes a symbol, $P'$ may guess that it is one of the symbols that would have been popped while $P$ read $y$, and if so $P'$ pops the stack and matches the symbol $P$ would push with the marked symbol $P'$ popped.

**2.20** Let $B = \{1^m 2^n 3^n 4^m \mid n, m \geq 0\}$ which is a CFL. If *CUT*$(B)$ were a CFL then the language *CUT*$(B) \cap 2^*1^*3^*4^*$ would be a CFL because Problem 2.30 shows that the intersection of a CFL and a regular language is a CFL. But that intersection is $\{2^n 1^m 3^n 4^m \mid n, m \geq 0\}$ which is easily shown to be a non-CFL using the pumping lemma.

**2.21** We show that an ambiguous CFG cannot be a DCFG. If $G$ is ambiguous, $G$ derives some string $s$ with at least two different parse trees and therefore $s$ has at least two different rightmost derivations and at least two different leftmost reductions. Compare the steps of two of these different leftmost reductions and locate the first string where these reduction differ. The preceding string must be the same in both reductions, but it must have two different handles. Hence $G$ is not a DCFG.

**2.23** **a.** Let $B = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i \neq j \text{ for } i, j, k \geq 0\}$ and $C = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid j \neq k \text{ for } i, j, k \geq 0\}$. The following DPDA recognizes $B$. Read all a's (if any) and push them onto the stack. Read b's (if any) while popping the a's. If the b's finish while a's remain on the stack or if b's remain unread when the stack becomes empty, then scan to the end of the input and $accept$, assuming that the input is in $\mathtt{a}^*\mathtt{b}^*\mathtt{c}^*$ which was checked in parallel. Otherwise, $reject$.

A similar DPDA recognizes $C$. Thus $B$ and $C$ are DCFLs. However we have shown in the text that $A = B \cup C$ isn't a DCFL. Therefore the class of DCFLs is not closed under union.

**b.** We have shown in the text that the class of DCFLs is closed under complement. Thus, if the class of DCFLs were closed under intersection, the class would also be closed under union because $B \cup C = \overline{\overline{B} \cap \overline{C}}$, contradicting Part **a** of this problem. Hence the class of DCFLs is not closed under intersection.

**c.** Define $B$ and $C$ as in Part **a**. Let $B' = (\mathtt{01} \cup \mathtt{0})B$ and $C' = (\mathtt{10} \cup \mathtt{1})C$. Then $D = B' \cup C'$ is a DCFL because a DPDA can determine which test to use by reading the first few symbols of the input. Let $E = (\mathtt{1} \cup \varepsilon)D$ which is $(\mathtt{101} \cup \mathtt{10} \cup \mathtt{01} \cup \mathtt{0})B \cup (\mathtt{110} \cup \mathtt{11} \cup \mathtt{10} \cup \mathtt{1})C$. To see that $E$ isn't a DCFL, take $F = E \cap \mathtt{01}(\mathtt{abc})^*$. The intersection of a DCFL and a regular language is a DCFL so if $E$ were a DCFL then $F$ would be a DCFL but $F = \mathtt{01}A$ which isn't a DCFL for the same reason $A$ isn't a DCFL.

**d.** Define $C$ and $F$ as in Part **c**. The language $H = \mathtt{1} \cup \varepsilon \cup D$ is easily seen to be a DCFL. However $H^*$ is not a DCFL because $H^* \cap \mathtt{01}(\mathtt{abc})^* = F \cup \mathtt{1} \cup \varepsilon$ and the latter isn't a DCFL for the same reason that $F$ isn't a DCFL.

**e.** Define $B$ and $C$ as in Part **a**. Let $K = \mathtt{0}B^{\mathcal{R}} \cup \mathtt{1}C^{\mathcal{R}}$. First, show that $K$ is a DCFL by giving a DPDA which operates as follows. After reading the first symbol, the DPDA follows the deterministic procedure to recognize membership in $B^{\mathcal{R}}$ or in $C^{\mathcal{R}}$ depending on whether the first symbol is 0 or 1. Next, show that $K^{\mathcal{R}}$ is not a DCFL by showing how to convert a DPDA $P$ for $K^{\mathcal{R}}$ into a DPDA $P'$ for the language $A = B \cup C$ which the text shows isn't a DCFL. Modify $P$ so that it recognizes the endmarked version of $K^{\mathcal{R}}$. Thus $L(P) = K^{\mathcal{R}}\dashv = B\mathtt{0}\dashv \cup C\mathtt{1}\dashv$. It is enough to design $P'$ to recognize $A\dashv = B\dashv \cup C\dashv$ because $A\dashv$ is a DCFL iff $A$ is a DCFL. It simulates $P$ but also keeps additional information on the stack which says what $P$ would do if its next input symbol were 0 or it it were 1. Then when $P'$ reads ⊣, it can use this information to accept when $P$ would have accepted if the prior input symbol had been either a 0 or a 1.

**2.24** **a.** Let $E = \{w|\ w \text{ has an equal number of } \mathtt{a}\text{'s and } \mathtt{b}\text{'s}\}$. It is enough to show that $T \overset{*}{\Rightarrow} w$ iff $w \in E$. The forward direction is straightforward. If $T \overset{*}{\Rightarrow} w$ then $w \in E$ because every substitution adds an equal number of a's and b's. The reverse direction is the following claim.

*Claim:* If $w \in E$ then $T \overset{*}{\Rightarrow} w$.

Proof by induction on $|w|$, the length of $w$.

Basis, $|w| = 0$: Then $w = \varepsilon$ and the rule $T \to \varepsilon$ shows that $T \overset{*}{\Rightarrow} w$.

Induction step: Let $k > 0$ and assume the claim is true whenever $|w| < k$. Prove the claim is true for $|w| = k$.

Take $w = w_1 \cdots w_k \in E$ where $|w| = k$ and each $w_i$ is a or b. Let $x = w^{\mathcal{R}}$. For $1 \leq i \leq k$, let $a_i$ ($b_i$) be the number of a's (b's) that appear among the first $i$ symbols of $x$ and let $c_i = a_i - b_i$. In other words, $c_i$ is the running count of the excess number of a's over b's across $x$. Because $w \in E$ and $E$ is closed under reversal, we know that $c_k = 0$. Let $m$ be the smallest $i$ for which $c_i = 0$. Consider the case where $x_1 = \mathtt{a}$. Then $c_1 = 1$. Moreover, $c_i > 0$ from $i = 1$ to $m - 1$ and $c_{m-1} = 1$. Thus, $x_2 \cdots x_{m-1} \in E$ and $x_m = \mathtt{b}$ and then also $x_{m+1} \cdots x_k \in E$. We can write $x = \mathtt{a}x_2 \cdots x_{m-1}\mathtt{b}x_{m+1} \cdots x_k$ and so $w = (x_{m-1} \cdots x_k)^{\mathcal{R}}\mathtt{b}(x_2 \cdots x_{m-1})^{\mathcal{R}}\mathtt{a}$. The induction assumption implies that $T \overset{*}{\Rightarrow} (x_{m-1} \cdots x_k)^{\mathcal{R}}$ and $T \overset{*}{\Rightarrow} (x_2 \cdots x_{m-1})^{\mathcal{R}}$. Therefore the rule $T \to T\mathtt{b}T\mathtt{a}$ shows that $T \overset{*}{\Rightarrow} w$. A similar argument works for the case where $x_1 = \mathtt{b}$.

**b.** Omitted

**c.** The DPDA reads the input and performs the following actions for each symbol read. If the stack is empty or if the input symbol is the same as the top stack symbol, the DPDA pushes the input symbol. Otherwise, if the top stack symbol differs from the input symbol, it pops the stack. When it reads ⊣, it accepts if the stack is empty, and otherwise it doesn't accept.

**2.26** Following the hint, the modified $P$ would accept the strings of the form $\{\mathtt{a}^m\mathtt{b}^m\mathtt{c}^m|\ m \geq 1\}$. But that is impossible because this language is not a CFL.

**2.27** Assume that DPDA $P$ recognizes $B$ and construct a modified DPDA $P'$ which simulates $P$ while reading a's and b's. If $P$ enters an accept state, $P'$ checks whether the next input symbol is a c, and if so it simulates $P$ on bc (pretending it has read an extra b) and then continues to simulate $P$ on the rest of the input. It accepts only when $P$ enters an accept state after reading c's. Then $P'$ recognizes the non-CFL $\{\mathtt{a}^m\mathtt{b}^m\mathtt{c}^m|\ m \geq 1\}$, an impossibility.

**2.28** Assume to the contrary that DPDA $P$ recognizes $C$. For a state $q$ and a stack symbol $x$, call $(q, x)$ a *minimal pair* if when $P$ is started $q$ with $x$ on the top of its stack, $P$ never pops its stack below $x$, no matter what input string $P$ reads from that point on. In that case, the contents of $P$'s stack at that point cannot affect its subsequent behavior, so $P$'s subsequent behavior can depend only on $q$ and $x$. Additionally, call $(q, \varepsilon)$ a minimal pair. It corresponds to starting $P$ in state $q$ with an empty stack.

*Claim:* Let $y$ be any input to $P$. Then $y$ can be extended to $z = ys$ for some $s \in \{\texttt{0,1}\}^*$ where $P$ on $z$ enters a minimal pair.

To prove this claim, observe that if $P$ on input $y$ enters a minimal pair then we are done immediately, and if $P$ on input $y$ enters a pair $(q, x)$ which isn't minimal then some input $s_1$ exists on which $P$ pops its stack below $x$. If $P$ on input $ys_1$ then enters a minimal pair, we are again done because we can let $z = ys_1$. If that pair still isn't minimal then some input $s_2$ exists which take $P$ to an even lower stack level. If $P$ on $ys_1s_2$ enters a minimal pair, we are done because we can let $z = ys_1s_2$. This procedure must terminate with a minimal pair, because $P$'s stack shrinks at every step and will eventually become empty. Thus, the claim is proved.

Let $k = 1 + |Q| \times (|\Gamma| + 1)$ be any value which is greater than the total number of minimal pairs. For every $i \leq k$, let $y_i = \texttt{10}^i\texttt{1}$. The strings $y_i$ are all distinct, and no string $y_i$ is a prefix of some other string $y_j$. The claim shows that we can extend these to $z_1, \ldots, z_k \in \{\texttt{0,1}\}^*$ where each $z_i$ takes $P$ to a minimal pair. Because $k$ exceeds the number of minimal pairs, two of these strings, $z_i$ and $z_j$, lead to the same minimal pair. Observe that $P$ accepts both $z_i z_i^{\mathcal{R}}$ and $z_j z_j^{\mathcal{R}}$ because these strings are members of $C$. But because $z_i$ and $z_j$ lead to the same minimal pair, $P$'s will behave identically if we append the same string to either of them. Thus $P$ accepts input $z_i z_j^{\mathcal{R}}$ because it accepts input $z_j z_j^{\mathcal{R}}$. But $z_i z_j^{\mathcal{R}} \notin C$, a contradiction.

**2.31** The grammar generates all strings not of the form $\texttt{a}^k\texttt{b}^k$ for $k \geq 0$. Thus the complement of the language generated is $\overline{L(G)} = \{\texttt{a}^k\texttt{b}^k |\ k \geq 0\}$. The following grammar generates $\overline{L(G)}$: $\{\{S\}, \{\texttt{a}, \texttt{b}\}, \{S \to \texttt{a}S\texttt{b} \mid \varepsilon\}, S\}$.

**2.32** Let $P$ be a PDA that recognizes $A$. We construct a PDA $P'$ that recognizes $A/B$ as follows. It operates by reading symbols and simulating $P$. Whenever $P$ branches nondeterministically, so does $P'$. In addition, at every point $P'$ nondeterministically guesses that it has reached the end of the input and refrains from reading any more input symbols. Instead, it guesses additional symbols as if they were input and continues to simulate $P$ on this guessed input, while at the same time using its finite control to simulate a DFA for $B$ on the guessed input string. If $P'$ simultaneously reaches accept states in both of these simulations, $P'$ enters an accept state.

**2.33** The following CFG $G$ generates the language $C$ of all strings with twice as many a's as b's. $S \to \texttt{b}S\texttt{aa} \mid \texttt{aa}S\texttt{b} \mid \texttt{a}S\texttt{b}S\texttt{a} \mid SS \mid \varepsilon$.

Clearly $G$ generates only strings in $C$. We prove inductively that every string in $C$ is in $L(G)$. Let $s$ be a string in $C$ of length $k$.

*Basis* If $k = 0$ then $S = \varepsilon \in L(G)$.

*Induction step* If $k > 0$, assume all strings in $C$ of length less than $k$ are generated by $G$ and show $s$ is generated by $G$.

If $s = s_1 \ldots s_k$ then for each $i$ from 0 to $k$ let $c_i$ to be the number of a's minus twice the number of b's in $s_1 \ldots s_i$. We have $c_0 = 0$ and $c_k = 0$ because $s \in L(G)$. Next we consider two cases.

i) If $c_i = 0$ for some $i$ besides 0 and $k$, we divide into two substrings $s = tu$ where $t$ is the first $i$ symbols and $u$ is the rest. Both $t$ and $u$ are members of $C$ and hence are in $L(G)$ by the induction hypothesis. Therefore the rule $S \rightarrow SS$ show that $s \in L(G)$.

ii) If $c_i \neq 0$ for all $i$ besides 0 and $k$, we consider three subcases.

i. If $s$ begins with `b`, then all $c_i$ from $i = 1$ to $k-1$ are negative, because $c_1 = -2$ and jumping from a negative $c_i$ to a positive $c_{i+1}$ isn't possible because the $c$ values can increase only by 1 at each step. Hence $s$ ends with `aa`, because no other ending would give $c_k = 0$. Therefore $s = \texttt{b}t\texttt{aa}$ where $t \in C$. By the induction hypothesis, $t \in L(G)$ and so the rule $S \rightarrow \texttt{b}S\texttt{aa}$ shows that $s \in L(G)$.

ii. If $s$ begins with `a` and all $c_i$ are non-negative, then $s_2 = \texttt{a}$ and $s_k = \texttt{b}$. Therefore $s = \texttt{aa}t\texttt{b}$ where $t \in C$. By the induction hypothesis, $t \in L(G)$ and so the rule $S \rightarrow \texttt{aa}S\texttt{b}$ shows that $s \in L(G)$.

iii. If $s$ begins with `a` and some $c_i$ is negative, then select the lowest $i$ for which a negative $c_i$ occurs. Then $c_{i-1} = +1$ and $c_i = -1$, because the $c$ values can decrease only by 2 at each step. Hence $s_i = \texttt{b}$. Furthermore $s$ ends with `a`, because no other ending would give $c_k = 0$. If we let $t$ be $s_2 \cdots s_{i-1}$ and $u$ be $s_{i+1} \cdots s_{k-1}$ then $s = \texttt{a}t\texttt{b}u\texttt{a}$ where both $t$ and $u$ are members of $C$ and hence are in $L(G)$ by the induction hypothesis. Therefore the rule $S \rightarrow \texttt{a}S\texttt{b}S\texttt{a}$ shows that $s \in L(G)$.

**2.34** We construct a PDA $P$ that recognizes $C$. First it nondeterministically branches to check either of two cases: that $x$ and $y$ differ in length or that they have the same length but differ in some position. Handling the first case is straightforward. To handle the second case, it operates by guessing corresponding positions on which the strings $x$ and $y$ differ, as follows. It reads the input at the same time as it pushes some symbols, say `1`s, onto the stack. At some point it nondeterministically guesses a position in $x$ and it records the symbol it is currently reading there in its finite memory and skips to the #. Then it pops the stack while reading symbols from the input until the stack is empty and checks that the symbol it is now currently reading is different from the symbol it had recorded. If so, it accepts.

Here is a more detailed description of $P$'s algorithm. If something goes wrong, for example, popping when the stack is empty, or getting to the end of the input prematurely, $P$ rejects on that branch of the computation.

1. Nondeterministically jump to either 2 or 4.
2. Read and push these symbols until read #. *Reject* if # never found.
3. Read and pop symbols until the end of the tape. *Reject* if another # is read or if the stack empties at the same time the end of the input is reached. Otherwise *accept*.
4. Read next input symbol and push 1 onto stack.
5. Nondeterministically jump to either 4 or 6.
6. Record the current input symbol $a$ in the finite control.
7. Read input symbols until # is read.
8. Read the next symbol and pop the stack.
9. If stack is empty, go to 10, otherwise go to 8.
10. *Accept* if the current input symbol isn't $a$. Otherwise *reject*.

**2.35** We construct a PDA $P$ recognizing $D$. This PDA guesses corresponding places on which $x$ and $y$ differ. Checking that the places correspond is tricky. Doing so relies on the observation that the two corresponding places are $n/2$ symbols apart, where $n$ is the length of the entire input. Hence, by ensuring that the number of symbols between the guessed places is equal to the number other symbols, the PDA can check that the guessed places do indeed correspond, Here we give a more detailed description of the PDA algorithm. If something goes wrong, for example, popping when the stack is empty, or getting to the end of the input prematurely, $P$ rejects on that branch of the computation.

1. Read next input symbol and push `1` onto the stack.
2. Nondeterministically jump to either 1 or 3.
3. Record the current input symbol $a$ in the finite control.
4. Read next input symbol and pop the stack. Repeat until stack is empty.
5. Read next input symbol and push 1 onto the stack.
6. Nondeterministically jump to either 5 or 7.
7. *Reject* if current input symbol equals $a$.
8. Read next input symbol and pop the stack. Repeat until stack is empty.
9. *Accept* if input is empty.

Alternatively we can give a CFG for this language as follows.

$$\begin{aligned} S &\to AB \mid BA \\ A &\to XAX \mid \texttt{0} \\ B &\to XBX \mid \texttt{1} \\ X &\to \texttt{0} \mid \texttt{1} \end{aligned}$$

**2.38** Consider a derivation of $w$. Each application of a rule of the form $A \to BC$ increases the length of the string by 1. So we have $n-1$ steps here. Besides that, we need exactly $n$ applications of terminal rules $A \to a$ to convert the variables into terminals. Therefore, exactly $2n-1$ steps are required.

**2.39** **a.** To see that $G$ is ambiguous, note that the string

`if condition then if condition then a:=1 else a:=1`

has two different leftmost derivations (and hence parse trees):

**1.** ⟨STMT⟩
→ ⟨IF-THEN⟩
→ `if condition then` ⟨STMT⟩
→ `if condition then` ⟨IF-THEN-ELSE⟩
→ `if condition then if condition then` ⟨STMT⟩ `else` ⟨STMT⟩
→ `if condition then if condition then a:=1 else` ⟨STMT⟩
→ `if condition then if condition then a:=1 else a:=1`

**2.** ⟨STMT⟩
→ ⟨IF-THEN-ELSE⟩
→ `if condition then` ⟨STMT⟩ `else` ⟨STMT⟩
→ `if condition then` ⟨IF-THEN⟩ `else` ⟨STMT⟩
→ `if condition then if condition then` ⟨STMT⟩ `else` ⟨STMT⟩
→ `if condition then if condition then a:=1 else` ⟨STMT⟩
→ `if condition then if condition then a:=1 else a:=1`

**b.** The ambiguity in part a) arises because the grammar allows matching an `else` both to the nearest and to the farthest `then`. To avoid this ambiguity we construct a new grammar that only permits the nearest match, by disabling derivations which introduce an ⟨IF-THEN⟩

before an else. This grammar has two new variables: $\langle\text{E-STMT}\rangle$ and $\langle\text{E-IF-THEN-ELSE}\rangle$, which work just like their non-$\langle\text{E}\rangle$ counterparts except that they cannot generate the dangling $\langle\text{IF-THEN}\rangle$. The rules of the new grammar are the same as for the old grammar except we remove the $\langle\text{IF-THEN-ELSE}\rangle$ rule and add the following new rules:

$$\begin{aligned}
\langle\text{E-STMT}\rangle &\rightarrow \langle\text{ASSIGN}\rangle|\langle\text{E-IF-THEN-ELSE}\rangle\\
\langle\text{E-IF-THEN-ELSE}\rangle &\rightarrow \texttt{if condition then } \langle\text{E-STMT}\rangle \texttt{ else } \langle\text{E-STMT}\rangle\\
\langle\text{IF-THEN-ELSE}\rangle &\rightarrow \texttt{if condition then } \langle\text{E-STMT}\rangle \texttt{ else } \langle\text{STMT}\rangle
\end{aligned}$$

**2.40** I found these problems to be surprisingly hard.

**a.** Define an *a-string* to be a string where every prefix has at least as many a's as b's. The following grammar $G$ generates all a-strings unambiguously.

$$\begin{aligned}
M &\rightarrow A\mathtt{a}M \mid A\\
A &\rightarrow \mathtt{a}A\mathtt{b}A \mid \varepsilon
\end{aligned}$$

Here is a proof sketch. First we claim that $A$ generates all balanced a-strings unambiguously. Let $w = w_1 \cdots w_n$ be any string. Let $c_i$ be the number of a's minus the number of b's in positions 1 through $i$ in $w$. The *mate* of a at position $i$ in $w$ is the b at the lowest position $j > i$ where $c_j < c_i$. It is easy to show inductively that for any balanced a-string $w$ and any parse tree for $w$ generated from $A$, the rule $A \rightarrow \mathtt{a}A\mathtt{b}A$ generates the mated pairs at the same time, hence it divides the string in a unique way and consequently grammar is unambiguous.

Next we claim that $M$ generates all a-strings that have an excess of a's. Say that a is *unmated* if it has no mate. We can show inductively that for any a-string, the $M$ rule $M \rightarrow A\mathtt{a}M$ generates the unmated a's and the $A$ rules generates the mated pairs. The generation can be done in only one way so the grammar is unambiguous.

**b.**

$$\begin{aligned}
E &\rightarrow \mathtt{a}A\mathtt{b}E \mid \mathtt{b}B\mathtt{a}E \mid \varepsilon\\
A &\rightarrow \mathtt{a}A\mathtt{b}A \mid \varepsilon\\
B &\rightarrow \mathtt{b}B\mathtt{a}B \mid \varepsilon
\end{aligned}$$

Proof omitted.

**c.**

$$\begin{aligned}
E &\rightarrow \mathtt{a}A\mathtt{b}E \mid \mathtt{b}B\mathtt{a}E \mid F\\
A &\rightarrow \mathtt{a}A\mathtt{b}A \mid \varepsilon\\
B &\rightarrow \mathtt{b}B\mathtt{a}B \mid \varepsilon\\
F &\rightarrow \mathtt{a}AF \mid \varepsilon
\end{aligned}$$

Proof omitted.

**2.41** This proof is similar to the proof of the pumping lemma for CFLs. Let $G = (V, \Sigma, S, R)$ be a CFG generating $A$ and let $b$ be the length of the longest right-hand side of a rule in $G$. If a variable $A$ in $G$ generates a string $s$ with a parse tree of height at most $h$, then $|s| \leq b^h$.

Let $p = b^{|V|+2}$ where $|V|$ is the number of variables in $G$. Let $s_1 = \mathtt{a}^{2p!}\mathtt{b}^{p!}\mathtt{c}^{p!}$. Consider a parse tree for $s_1$ that has the fewest possible nodes. Take a symbol c in $s_1$ which has the longest path to $S$ in the parse tree among all c's. That path must be longer than $|V| + 1$ so some variable $B$ must repeat on this path. Chose a repetition that occurs among the lowest (nearest to the leaves) $|V| + 1$ variables. The upper $B$ can have at most $b^{|V|+2} \leq p$ `c`'s in the subtree below it and therefore it can have at most that number of `b`'s or else the tree surgery technique would yield a derived string with unequal numbers of `b`'s and `c`'s. Hence the subtree below the upper $B$ doesn't produce any `a`'s. Thus the upper and lower $B$'s yield a division of $s_1$ into five parts $uvxyz$ where $v$ contains only `b`'s and $y$ contains only `c`'s. Any other division would allow us to pump the result to obtain a string outside of $A$. Let $q = |v| = |y|$. Because $q \leq p$ we know that $q$ divides $p!$. So we can pump this string up and get a parse tree for $\mathtt{a}^{2p!}\mathtt{b}^{2p!}\mathtt{c}^{2p!}$.

Next let $s_2 = \mathtt{a}^{p!}\mathtt{b}^{p!}\mathtt{c}^{2p!}$ and carry out the same procedure to get a different parse tree for $\mathtt{a}^{2p!}\mathtt{b}^{2p!}\mathtt{c}^{2p!}$. Thus this string has two different parse trees and therefore $G$ is ambiguous.

**2.42** **a.** Assume $A$ is context-free. Let $p$ be the pumping length given by the pumping lemma. We show that $s = \mathtt{0}^p\mathtt{1}^p\mathtt{0}^p\mathtt{1}^p$ cannot be pumped. Let $s = uvxyz$.

If either $v$ or $y$ contain more than one type of alphabet symbol, $uv^2xy^2z$ does not contain the symbols in the correct order and cannot be a member of $A$. If both $v$ and $y$ contain (at most) one type of alphabet symbol, $uv^2xy^2z$ contains runs of `0`'s and `1`'s of unequal length and cannot be a member of $A$. Because $s$ cannot be pumped without violating the pumping lemma conditions, $A$ is not context free.

**d.** Assume $A$ is context-free. Let $p$ be the pumping length from the pumping lemma. Let $s = \mathtt{a}^p\mathtt{b}^p\mathtt{\#a}^p\mathtt{b}^p$. We show that $s = uvxyz$ cannot be pumped. Use the same reasoning as in part (c).

**2.43** Assume $B$ is context-free and get its pumping length $p$ from the pumping lemma. Let $s = \mathtt{0}^p\mathtt{1}^{2p}\mathtt{0}^p$. Because $s \in B$, it can be split $s = uvxyz$ satisfying the conditions of the lemma. We consider several cases.

i) If both $v$ and $y$ contain only `0`'s (or only `1`'s), then $uv^2xy^2z$ has unequal numbers of `0`s and `1`s and hence won't be in $B$.

ii) If $v$ contains only `0`s and $y$ contains only `1`s, or vice versa, then $uv^2xy^2z$ isn't a palindrome and hence won't be in $B$.

iii) If both $v$ and $y$ contain both `0`s and `1`s, condition 3 is violated so this case cannot occur.

iv) If one of $v$ and $y$ contain both `0`s and `1`s, then $uv^2xy^2z$ isn't a palindrome and hence won't be in $B$.

Hence $s$ cannot be pumped and contradiction is reached. Therefore $B$ isn't context-free.

**2.44** Assume $C$ is context-free and get its pumping length $p$ from the pumping lemma. Let $s = \mathtt{1}^p\mathtt{3}^p\mathtt{2}^p\mathtt{4}^p$. Because $s \in C$, it can be split $s = uvxyz$ satisfying the conditions of the lemma. By condition 3, $vxy$ cannot contain both `1`s and `2`s, and cannot contain both `3`s and `4`s. Hence $uv^2xy^2z$ doesn't have equal number of `1`s and `2`s or of `3`s and `4`s, and therefore won't be a member of $C$, so $s$ cannot be pumped and contradiction is reached. Therefore $C$ isn't context-free.

**2.45** Assume to the contrary that $F$ is a CFL. Let $p$ be the pumping length given by the pumping lemma. Let $s = \mathtt{a}^{2p^2}\mathtt{b}^{2p}$ be a string in $F$. The pumping lemma says that we can divide $s = uvxyz$ satisfying the three conditions. We consider several cases. Recall that condition three says that $|vxy| \leq p$.

i) If either $v$ or $y$ contain two types of symbols, $uv^2xy^2z$ contains some `b`'s before `a`'s and is not in $F$.

ii) If both $v$ and $y$ contain only a's, $uv^2xy^2z$ has the form $\texttt{a}^{2p^2+l}\texttt{b}^{2p}$ where $l \leq p < 2p$. But $2p^2 + l$ isn't a multiple of $2p$ if $l < 2p$ so $uv^2xy^2z \notin F$.

iii) If both $v$ and $y$ contain only b's, $uv^mxy^mz$ for $m = 2p^2$ has the form $\texttt{a}^i\texttt{b}^j$ where $i < j$ and so it cannot be in $F$.

iv) If $v$ contains only a's and $y$ contains only b's, let $t = uv^mxy^mz$. String $t$ cannot be member of $F$ for any sufficiently large value of $m$. Write $s$ as $\texttt{a}^{g+|v|}\texttt{b}^{h+|y|}$. Then $t$ is $\texttt{a}^{g+m|v|}\texttt{b}^{h+m|y|}$. If $t \in F$ then $g + m|v| = k(h + m|y|)$ for some integer $k$. In other words

$$k = \frac{g + m|v|}{h + m|y|}.$$

If $m$ is sufficiently large, $g$ is a tiny fraction of $m|v|$ and $h$ is a tiny fraction of $m|y|$ so we have

$$k = \frac{m|v|}{m|y|} = \frac{|v|}{|y|}$$

because $k$ is an integer. Moreover $k < p$. Rewriting the first displayed equation we have $g - hk = m(k|y| - |v|)$ which must be 0, due to the second displayed equation. But we have chosen $s$ so that $g - hk \neq 0$ for $k < p$, so $t$ cannot be member of $F$.

Thus none of the cases can occur, so the string $s$ cannot be pumped, a contradiction.

**2.46** $L(G)$ is the language of strings of 0s and #s that either contain exactly two #s and any number of 0s, or contain exactly one # and the number of 0s to the right of the # is twice the number of 0s to the left. First we show that three is not a pumping length for this language. The string 0#00 has length at least three, and it is in $L(G)$. It cannot be pumped using $p = 3$, because the only way to divide it into $uvxyz$ satisfying the first two conditions of the pumping lemma is $u = z = \varepsilon$, $v = \texttt{0}$, $x = \texttt{\#}$, and $y = \texttt{00}$, but that division fails to satisfy the third condition.

Next, we show that 4 is a pumping length for $L(G)$. If $w \in L(G)$ has length at least 4 and if it contains two #s, then it contains at least one 0. Therefore, by cutting $w$ into $uvxyz$ where either $v$ or $y$ is the string 0, we obtain a way to pump $w$. If $w \in L(G)$ has length at least 4 and if it contains a single #, then it must be of the form $\texttt{0}^k\texttt{\#0}^{2k}$ for some $k \geq 1$. Hence, by assigning $u = \texttt{0}^{k-1}$, $v = \texttt{0}$, $x = \texttt{\#}$, $y = \texttt{00}$, and $z = \texttt{0}^{2k-2}$, we satisfy all three conditions of the lemma.

**2.47** Assume G generates a string $w$ using a derivation with at least $2^b$ steps. Let $n$ be the length of $w$. By the results of Problem 2.38, $n \geq \frac{2^b+1}{2} > 2^{b-1}$.

Consider a parse tree of $w$. The right-hand side of each rule contains at most two variables, so each node of the parse tree has at most two children. Additionally, the length of $w$ is at least $2^b$, so the parse tree of $w$ must have height at least $b+1$ to generate a string of length at least $2^b$. Hence, the tree contains a path with at least $b+1$ variables, and therefore some variable is repeated on that path. Using a surgery on trees argument identical to the one used in the proof of the CFL pumping lemma, we can now divide w into pieces $uvxyz$ where $uv^ixy^iz \in G$ for all $i \geq 0$. Therefore, $L(G)$ is infinite.

**2.48** Let $F = \{\texttt{a}^i\texttt{b}^j\texttt{c}^k\texttt{d}^l \mid i, j, k, l \geq 0$ and if $i = 1$ then $j = k = l\}$. $F$ is not context free because $F \cap \texttt{ab}^*\texttt{c}^*\texttt{d}^* = \{\texttt{ab}^n\texttt{c}^n\texttt{d}^n \mid n \geq 0\}$, which is not a CFL by the pumping lemma, and because the intersection of a CFL and a regular language is a CFL. However, $F$ does satisfy the pumping lemma with $p = 2$. ($p = 1$ works, too, with a bit more effort.)

If $s \in F$ has length 2 or more then:

i) If $s \in \mathtt{b}^*\mathtt{c}^*\mathtt{d}^*$ write $s$ as $s = rgt$ for $g \in \{\mathtt{b}, \mathtt{c}, \mathtt{d}\}$ and divide $s = uvxyz$ where $u = r$, $v = g$, $x = y = \varepsilon$, $z = t$.

ii) If $s \in \mathtt{ab}^*\mathtt{c}^*\mathtt{d}^*$ write $s$ as $s = \mathtt{a}t$ and divide $s = uvxyz$ where $u = \varepsilon$, $v = \mathtt{a}$, $x = y = \varepsilon$, $z = t$.

iii) If $s \in \mathtt{aaa}^*\mathtt{b}^*\mathtt{c}^*\mathtt{d}^*$ write $s$ as $s = \mathtt{aa}t$ and divide $s = uvxyz$ where $u = \varepsilon$, $v = \mathtt{aa}$, $x = y = \varepsilon$, $z = t$.

In each of the three cases, we may easily check that the division of $s$ satisfies the conditions of the pumping lemma.

**2.49** Let $G = (\Sigma, V, T, P, S)$ be a context free grammar for $A$. Define $r$ to be the length of the longest string of symbols occurring on the right hand side of any production in $G$. We set the pumping length $k$ to be $r^{2|V|+1}$. Let $s$ be any string in $A$ of length at least $k$ and let $T$ be a derivation tree for $A$ with the fewest number of nodes. Observe that since $s \geq k$, the depth of $T$ must be at least $2|V| + 1$. Thus, some path $p$ in $T$ has length at least $2|V| + 1$. By the pigeonhole principle, some variable $V'$ appears at least three times in $p$. Label the last three occurrences of $V'$ in $p$ as $V_1$, $V_2$, and $V_3$. Moreover, define the strings generated by $V_1$, $V_2$, and $V_3$ in $T$ as $s_1$, $s_2$, and $s_3$ respectively. Now, each of these strings is nested in the previous because they are all being generated on the same path. Suppose then that $s_1 = l_1 s_2 r_1$ and $s_2 = l_2 s_3 r_2$. We now have three cases to consider:

i) $|l_1 l_2| \geq 1$ and $|r_1 r_2| \geq 1$: In this case, there is nothing else to prove since we can simply pump in the usual way.

ii) $|l_1 l_2| = 0$: Since we defined $T$ to be a minimal tree, neither $r_1$ nor $r_2$ can be the empty string. So, we can now pump $r_1$ and $r_2$.

iii) $|r_1 r_2| = 0$: This case is handled like the previous one.

**2.51** Let $A$ and $B$ be defined as in the solution to Problem 2.50. Let $D$ be the shuffle of $A$ and $B$. Then let $E = D \cap ((\mathtt{0} \cup \mathtt{1})(\mathtt{a} \cup \mathtt{b}))^*$ The language $E$ is identical to the language $C$ in Problem 2.50 and was shown there to be non-context free. The intersection of a CFL and a regular language is a CFL, so $D$ must not be context free, and therefore the class of CFLs isn't closed under the shuffle operation.

**2.52** The language $C$ is context free, therefore the pumping lemma holds. Let $p$ be the pumping length and $s \in C$ be a string longer then $p$, so it can be split in $uvxyz$ such that $uv^ixy^iz \in C$ for all $i > 0$, where $|vy| > 0$. All prefixes are in $C$ so all $uv^i \in C$ and thus the regular language $u(v)^* \subseteq C$. If $v \neq \varepsilon$, that language is infinite and we're done. If $v = \varepsilon$, then $y \neq \varepsilon$, and the infinite regular language $uvx(y)^* \subseteq C$.

**2.53** **a.** Let $B = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k |\ i \neq j \text{ or } i \leq k\}$ which is a CFL. Let $s = \mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \in B$. Assume $k \geq 2$. Let $s' = \mathtt{a}^i\mathtt{b}^j\mathtt{c}^{k-1}$ which is a prefix of $s$. Show that $s' \in B$ if $i \neq j$ or $i \neq k$.

**1.** If $i \neq j$ then $s' \in B$.

**2.** If $i < k$ then $i \leq (k-1)$ so $s' \in B$.

**3.** If $i > k$ then $i \neq j$ (because $s \in B$) so $s' \in B$.

If $i = j = k$ then $s$ has no prefix in $B$ because removing some of the c's would yield a string that fails both conditions of membership in $B$. Thus, $(\mathit{NOPREFIX}(B) \cap \mathtt{a}^*\mathtt{b}^*\mathtt{ccc}^*) \cup \mathtt{abc} \cup \varepsilon = \{\mathtt{a}^i\mathtt{b}^i\mathtt{c}^i |\ i \geq 0\}$ which isn't a CFL. Therefore, $\mathit{NOPREFIX}(B)$ cannot be a CFL.

**b.** Let $C = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k |\ i \neq j \text{ or } i \geq k\}$ which is a CFL. Let $s = \mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \in C$. If $i \neq j$ or if $i > k$ then $s\mathtt{c} \in C$. But if $i = j = k$ then $s$ has no extension in $C$. Therefore, $\mathit{NOEXTEND}(C) = \{\mathtt{a}^i\mathtt{b}^i\mathtt{c}^i |\ i \geq 0\}$ which isn't a CFL.

**2.54** Assume $Y$ is a CFL and let $p$ be the pumping length given by the pumping lemma. Let $s = \mathtt{1}^{p+1}\mathtt{\#1}^{p+2}\mathtt{\#}\cdots\mathtt{\#1}^{5p}$. String $s$ is in $Y$ but we show it cannot be pumped. Let $s = uvxyz$ satisfying the three conditions of the lemma. Consider several cases.

i) If either $v$ or $y$ contain #, the string $uv^3xy^3z$ has two consecutive $t_i$'s which are equal to each other. Hence that string is not a member of $Y$.

ii) If both $v$ and $y$ contain only 1s, these strings must either lie in the same run of 1s or in consecutive runs of 1s within $s$, by virtue of condition 3. If $v$ lies within the runs from $\mathtt{1}^{p+1}$ to $\mathtt{1}^{3p}$ then $uv^2xy^2z$ adds at most $p$ 1s to that run so that it will contain the same number of 1s in a higher run. Therefore the resulting string will not be a member of $Y$. If $v$ lies within the runs from $\mathtt{1}^{3p+1}$ to $\mathtt{1}^{5p}$ then $uv^0xy^0z$ subtracts at most $p$ 1s to that run so that it will contain the same number of 1s in a lower run. Therefore the resulting string will not be a member of $Y$.

The string $s$ isn't pumpable and therefore doesn't satisfy the conditions of the pumping lemma, so a contradiction has occurred. Hence $Y$ is not a CFL.

**2.55** **a.** First note that a PDA can use its stack to simulate an unbounded integer counter. Next suppose $A \subseteq \{\mathtt{0},\mathtt{1}\}^*$ is recognized by a DFA $D$. Clearly, a binary string $x$ is in *SCRAMBLE*$(A)$ iff $x$ has the same length and same number of 1s as some $w$ that is accepted by $D$. Thus, a PDA $M$ for *SCRAMBLE*$(A)$ operates on input $x$ by nondeterministically simulating $D$ on every possible $|x|$-long input $w$ using its stack as a counter to keep track of the difference between the number of 1s in $w$ and the number of 1s in $x$. A branch of $M$'s computation accepts iff it reaches the end of $x$ with the stack recording a difference of 0 and the simulation of $D$ on an accepting state.

**b.** If $|\Sigma| > 2$, then the claim in part (a) is false. For example, consider the regular language $A = (\mathtt{abc})^*$ over the ternary alphabet $\Sigma = \{\mathtt{a},\mathtt{b},\mathtt{c}\}$.

$$\textit{SCRAMBLE}(A) = \{x \in \Sigma^* |\ x \text{ has the same number of a's, b's and c's.}\}$$

This language is not context-free.

**2.56** Let $M_A$ be a DFA that recognizes $A$, and $M_B$ be a DFA that recognizes $B$. We construct a PDA recognizing $A \diamond B$. This PDA simulates $M_A$ on the first part of the string pushing every symbol it reads on the stack until it guesses that it has reached the middle of the input. After that it simulates $M_B$ on the remaining part of the string popping the stack for every symbol it reads. If the stack is empty at the end of the input and both $M_A$ and $M_B$ accepted, the PDA accepts. If something goes wrong, for example, popping when the stack is empty, or getting to the end of the input prematurely, the PDA rejects on that branch of the computation.

**2.57** Suppose $A$ is context-free and let $p$ be the associated pumping length. Let $s = \mathtt{1}^{2p}\mathtt{0}^p\mathtt{1}^p\mathtt{1}^{2p}$ which is in $A$ and longer than $p$. By the pumping lemma we know that $s = uvxyz$ satisfying the three conditions. We distinguish cases to show that $s$ cannot be pumped and remain in $A$.

- If $vxy$ is entirely in the last two thirds of $s$, then $uv^2xy^2z$ contains 0s in its first third but not in its last third and so is not in $A$.
- Otherwise, $vxy$ intersects the first third of $s$, and it cannot extend beyond the first half of $s$ without violating the third condition.
  - If $v$ contains both 1s and 0s, then $uv^2xy^2z$ contains 0s in its first third but not in its last third and so is not in $A$.
  - If $v$ is empty and $y$ contains both 0s and possibly 1s, then again $uv^2xy^2z$ contains 0s in its first third but not in its last third and is not in $A$.

- Otherwise, either $v$ contains only 1s, or $v$ is empty and $y$ contains only 1s. In both cases, $uv^{1+6p}xy^{1+6p}z$ contains 0s in its last third but not in its first third and so is not in $A$.

A contradiction therefore arises and so $A$ isn't a CFL.

**2.58** **a.**

$$\begin{aligned} S &\rightarrow XSX \mid T\mathtt{1} \\ T &\rightarrow XT \mid X \\ X &\rightarrow \mathtt{0} \mid \mathtt{1} \end{aligned}$$

Here $T$ is any nonempty string, and $S$ is any string with $T$1 in the middle (so the 1 falls at least one character to the right of the middle).

**b.** Read the input until a 1 appears, while at the same time pushing 0s. Pop two 0s. Continue reading the input while popping the stack in an accept state until reach the end of the input. If the stack empties before reaching the end of the input, then do not read any more input (i.e., reject).

**2.59** **a.**

$$\begin{aligned} S &\rightarrow T \mid \mathtt{1} \\ T &\rightarrow XXTX \mid XTXX \mid XTX \mid X\mathtt{1}X \\ X &\rightarrow \mathtt{0} \mid \mathtt{1} \end{aligned}$$

**b.** Assume that $C_2$ is a CFG and apply the pumping lemma to obtain the pumping length $p$. Let $s = \mathtt{0}^{p+2}\mathtt{1}\mathtt{0}^{p}\mathtt{1}\mathtt{0}^{p+2}$. Clearly, $s \in C_2$ so we may write $s = uvxyz$ satisfying the three conditions of the pumping lemma. If either $v$ or $y$ contains a 1, then the string $uxz$ contains fewer than two 1s and thus it cannot be a member of $C_2$. By condition three of the pumping lemma, parts $v$ and $y$ cannot together contain 0s from both of the two outer runs of 0s. Hence in the string $uv^2xy^2z$ the 1s cannot both remain in the middle third and so $uv^2xy^2z$ is not in $C_2$.