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

Michael Sipser

Mathematics Department

MIT

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 $\{0,1\}$.
f. The empty set.

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

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

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 ith element of C iff the ith place of b is $\mathtt{1}$. 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 $\Sigma = \{0, 1\}$.

a. $1\Sigma^*0$

b. $\Sigma^*1\Sigma^*1\Sigma^*1\Sigma^*$

c. $\Sigma^*0101\Sigma^*$

d. $\Sigma\Sigma0\Sigma^*$

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

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

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

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

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

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

k. $\varepsilon \cup 0$

l. $(1^*01^*01^*)^* \cup 0^*10^*10^*$

m. $\emptyset$
n. Σ^+

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

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

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; ε.

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) Σ is a finite set called the *alphabet*,
iii) Γ 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 Σ, and $v = v_1 v_2 \cdots v_n$ be a string over the Γ. 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) $\Sigma = \{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$,
iii) $\Gamma = \{\mathtt{0}, \mathtt{1}\}$,
iv) δ 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) $\Sigma = \{\mathtt{a}, \mathtt{b}\}$,

iii) $\Gamma = \{\mathtt{0}, \mathtt{1}\}$,

iv) δ 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 $A_2 = \{www |\ w \in \{\mathtt{0},\mathtt{1}\}^*\}$. 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 $\mathtt{a}^p\mathtt{b}\mathtt{a}^p\mathtt{b}\mathtt{a}^p\mathtt{b}$. 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 $s = \mathtt{0}^p\mathtt{1}^p$ *can* be pumped. Let $s = xyz$, where $x = \mathtt{0}$, $y = \mathtt{0}$ and $z = \mathtt{0}^{p-2}\mathtt{1}^p$. The conditions are satisfied because

i) for any $i \geq 0$, $xy^iz = \mathtt{0}\mathtt{0}^i\mathtt{0}^{p-2}\mathtt{1}^p$ is in $\mathtt{0}^*\mathtt{1}^*$.

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 ε 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: (T, q) and (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 ϵ-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 $A_k = \Sigma^* \mathtt{0} \Sigma^{k-1} \mathtt{0} \Sigma^*$. 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, ... , 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 ≥ 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 $\{\mathtt{a}\mathtt{b}^n\mathtt{c}^n \mid n \geq 0\}$ is the same as $F \cap \mathtt{a}\mathtt{b}^*\mathtt{c}^*$, 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 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 $\mathtt{a}\mathtt{b}^*\mathtt{c}^*$, 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 $\mathtt{a}\mathtt{a}\mathtt{b}^*\mathtt{c}^*$, 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 $\mathtt{a}\mathtt{a}\mathtt{a}^*\mathtt{b}^*\mathtt{c}^*$, 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 $(\mathtt{0}\mathtt{1})^*$ of length 1 or more contains $\mathtt{0}\mathtt{1}$ and hence can be pumped by dividing it so that $x = \varepsilon$, $y = \mathtt{0}\mathtt{1}$, 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 $\mathtt{0}\mathtt{0}$ 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 $\mathtt{1}\mathtt{0}\mathtt{0}$ 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 $\mathtt{1}\mathtt{0}$, y is in $\mathtt{1}\mathtt{1}^*\mathtt{0}$ and z is in $(\mathtt{1}\mathtt{1}^*\mathtt{0})^*\mathtt{0}$, which satisfies all of the conditions of the pumping lemma.

i. The minimum pumping length is 5. The string $\mathtt{1}\mathtt{0}\mathtt{1}\mathtt{1}$ 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 $C = \{w |\ w \in \{0,1\}^* \text{ is a palindrome}\}$ is regular. Let p be the pumping length given by the pumping lemma. The string $s = 0^p10^p \in C$ 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 $\overline{Y} \cap 1^*\#1^* = \{1^n\#1^n |\ n \geq 0\}$. 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 $s = 1^{p!}\#1^{2p!}$. 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, $xy^{1+k}z = 1^{2p!}\#1^{2p!}$ 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 $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 \cup \varepsilon$, 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) $Q = \{q_0, \ldots, q_k\}$

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 $M' = (Q', \Gamma, \delta', q_0, F)$. $Q' = Q \cup \{q_i^a |\ q \in Q, a \in \Sigma, 1 \le i \le k_a\}$

For every $q \in Q$,

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

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

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 $B = \{0^m 1^m |\ m \ge 0\}$ and let f: $\{0,1\} \longrightarrow \{2\}$ be the homomorphism where $f(0) = f(1) = 2$. We know that B is a non-regular language but $f(B) = \{2^{2m} |\ m \ge 0\}$ is a regular language.

1.61 **a.** $RC(A) = \{w_{i+1} \cdots w_n w_1 \cdots w_i |\ w = w_1 \cdots w_n \in A \text{ and } 1 \le i \le n\}$, 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 $1 \le j \le n$.

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 δ' 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 $1^p\texttt{=}\texttt{0}\texttt{+}1^p$, 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 $|xy| \le p$, it follows that y is 1^k for some $k \ge 1$. Then xy^2z is the string $1^{p+k}\texttt{=}\texttt{0}\texttt{+}1^p$, 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 $B_2(A) = \mathtt{10}^*$ 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
$\delta'((q, r), a) = \{(u, v) | \ u \in \delta(q, a)$ and $r \in \delta(v, b)$ for some $b \in \Sigma\}$.
iii) $q_0{}' = (q_0, q_{\text{accept}})$.
iv) $F' = \{(q, q) | \ q \in Q\}$.

1.65 Let $A = \{\mathtt{0}^*\mathtt{\#1}^*\}$. Thus, $A_{\frac{1}{3}-\frac{1}{3}} \cap \{\mathtt{0}^*\mathtt{1}^*\} = \{\mathtt{0}^n\mathtt{1}^n | \ n \geq 0\}$. Regular sets are closed under intersection, and $\{\mathtt{0}^n\mathtt{1}^n | \ n \geq 0\}$ 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 $\overline{C}$.

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

1.71 **a.** Observe that $B = \texttt{1}\Sigma^*\texttt{1}\Sigma^*$ 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 $s = \texttt{1}^p\texttt{01}^p$. The pumping lemma says that $s = xyz$ satisfying the three conditions. Condition three says that y appears among the left-hand 1s. 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 $B = \{\texttt{0110}\}$. Then $CUT(B) = \{\texttt{0110}, \texttt{1010}, \texttt{0011}, \texttt{1100}, \texttt{1001}\}$ and $CUT(CUT(B)) = \{\texttt{0110}, \texttt{1010}, \texttt{0011}, \texttt{1100}, \texttt{1001}, \texttt{0101}\}$.

b. Let A be a regular language that is recognized by DFA M. We construct a NFA N that recognizes $CUT(A)$. 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 $W = \{w |\ w \in \Sigma^k\}$ 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
$\delta(z_{k-1}, \mathtt{1}) = \{z_k\}$, $\delta(z_k, a) = \{z_k\}$.