Chapter 10

Recursively Enumerable Languages

10.2. (a) This approach would not work for recognizing $L_1 \cup L_2$. The reason is that if the input string is in L_2 but not in L_1, the TM might enter an infinite loop while simulating T_1 and thus never begin the simulation of T_2.

(b) This approach could be used to recognize $L_1 \cap L_2$. If the input string is in the language, the TM will accept it, since the simulations of T_1 and T_2 will both terminate successfully. In all other cases, one of the simulations will reject or enter an infinite loop, and in any such case the composite TM will fail to accept.

10.3. False. There is a language L that is not recursively enumerable, and if $L = \{x_1, x_2, \ldots\}$, then $L = \{x_1\} \cup \{x_2\} \cup \ldots$.

10.4. For each i, L_i' is the union of the L_j's with $j \neq i$. Therefore, L_i' is recursively enumerable. Since L_i is also, L_i is recursive by Theorem 10.5.

10.5. We sketch the proof that if there is a TM T enumerating L in canonical order, then L is recursive. First, in the case where L is finite, L is obviously recursive. Otherwise, given a string x, a TM T_1 can determine whether x is in L by executing T until one of the strings left on the tape of T is either x or a string that comes after x in canonical order. In the first case, $x \in L$, and in the second case $x \notin L$.

10.6. One direction here is simple: if there is a computable partial function f that is defined precisely at the points of L, then by definition of computable, a TM computing f accepts precisely the strings in L.

Suppose that T accepts L. We may simply modify T by making the TM erase the tape and leave the tape head on square 0 before accepting. Then the modified TM computes the partial function f, defined to be Λ at every point in L and undefined otherwise.

10.7. Suppose there is no longest path. Then the root node N_0 must have infinitely many descendants. Since N_0 has only a finite number of children, then one of them, say N_1, must have infinitely many descendants. Similarly, one of N_1's children, say N_2, must have infinitely many descendants. This reasoning can be repeated indefinitely, and it follows that the tree has the infinite path $N_0, N_1, \ldots$, contrary to the assumption.

10.8. One approach to both (a) and (b), which uses the fact that these languages are both recursive, is to choose a method of enumerating $\{0,1\}^*$ and for each string in the enumeration, list it if it is in the language we want. However, we give alternate algorithms, which are presumably at least a little more efficient.

(a) The set is

$$101, 101^2, 1^201, 101^3, 1^201^3, 1^301, 101^4, 1^201^5, 1^301^2, 1^401, \ldots$$

The "algorithm" (though of course it never terminates) is the following:

For $i = 1$ to ∞
  For $j = 1$ to i
    List the string 1^j01^k, where k is the smallest integer relatively prime to j for which 1^j01^k has not yet been listed

(b) For each nonnull string w, let $x_{1,w}$, $x_{2,w}$, ... be an enumeration, possibly including repetitions, of all strings containing the substring www. (For example, we could for each i list every string of length i $i+1$ times, and insert www in each of the $i+1$ positions.) Now let w_1, w_2, ... be an enumeration of the strings w in canonical order. Then use the algorithm

For $i = 1$ to ∞
  For $j = 1$ to i
    List the string $x_{j,w_{i+1-j}}$ *if* it has not previously been listed.

(c) Fermat's last theorem, of which a proof was published by Andrew Wiles in 1995, says that there are no such n's bigger than 2. If we pretend that we don't know this, we can select some enumeration of the 4-tuples (n, x, y, z), and for each such 4-tuple, list n if it is not already listed and $x^n + y^n = z^n$. If we do know the theorem, then we write the numbers 1 and 2, and we're done.

10.10. Given a number y, we can determine whether y is in the range of f by computing $f(0)$, $f(1)$, ..., until we find a j for which $f(j) \geq y$. If $f(j) > y$ for the first such j, y is not in the range.

10.11. (a) The variable D can be thought of as a "doubling" operator. The language is $\{a^n \mid n = 2^k \text{ for some } k \geq 0\}$.

(b) $\{a^n \mid n = 2^j3^k \text{ for some } j, k \geq 0\}$

(c) $\{x \in \{a, b, c\}^* \mid n_a(x) = n_b(x) = n_c(x)\}$

(d) $\{a^n \mid n = k^2 \text{ for some } k \geq 1\}$. The way to understand this is to consider the string $\alpha_k = LA^k * A^k * \ldots * A^k * R$, in which there are k groups of k A's, each group followed by $*$. I can be viewed as an "incrementing" variable, which operates on the string to produce α_{k+1}. After using the productions $S \to LA * R$, we have α_1. When an additional I is created in α_k, the two productions $I* \to A * IJ$ and $IR \to A * R$ allow I to add an extra A to each of the k groups and add an extra group (i.e., an extra $*$) at the end with one A in it. In addition, the I creates a J in each of the original groups, and once each J has worked its way to the end of the string, it leaves an extra A in the final group. The result is therefore $k+1$ groups of $k+1$ A's. The variable E is an "erasing" variable that destroys all the symbols except the A's, which can be replaced by a's.

10.12. (a) $\{a^nb^na^n \mid n \geq 0\}$

(b) $B \to b$

10.13. (a)

$$S \to FS_1 \mid \Lambda \qquad S_1 \to ABCDS_1 \mid ABCD$$
$$BA \to AB \qquad CA \to AC \qquad DA \to AD \qquad CB \to BC \qquad DB \to BD \qquad DC \to CD$$
$$FA \to A_1 \qquad A_1A \to A_1A_1 \qquad A_1B \to A_1B_1 \qquad B_1B \to B_1B_1$$
$$B_1C \to B_1C_1 \qquad C_1C \to C_1C_1 \qquad C_1D \to C_1D_1 \qquad D_1D \to D_1D_1$$
$$A_1 \to a \qquad B_1 \to b \qquad C_1 \to a \qquad D_1 \to b$$

The idea here is that the symbols A, B, C, and D rearrange themselves into the form $A^nB^nC^nD^n$. The auxiliary variables A_1, B_1, C_1, and D_1 are introduced in order to force this to happen: The original variables can be converted into the new ones only when they are in the right order, and the terminals can be produced only by the new variables. (If the original variables went directly to terminal symbols, in the same way as in Example 10.1, there would be the possibility of errors, because both a's and b's are allowed to appear in two different parts of the string.)

(b)

$$S \to FS_1 \mid \Lambda \qquad S_1 \to ABCS_1 \mid ABC$$
$$BA \to AB \qquad CA \to AC \qquad CB \to BC$$
$$FA \to A_1 \qquad A_1A \to A_1A_1 \qquad A_1B \to A_1B_1$$
$$B_1B \to B_1B_1 \qquad B_1C \to B_1C_1 \qquad C_1C \to C_1C_1$$
$$A_1 \to a \qquad B_1 \to a \mid b \qquad C_1 \to b$$

The principle in this grammar is the same as in part (a), except that this time the symbol B_1 can be replaced by either a or b.

(c)

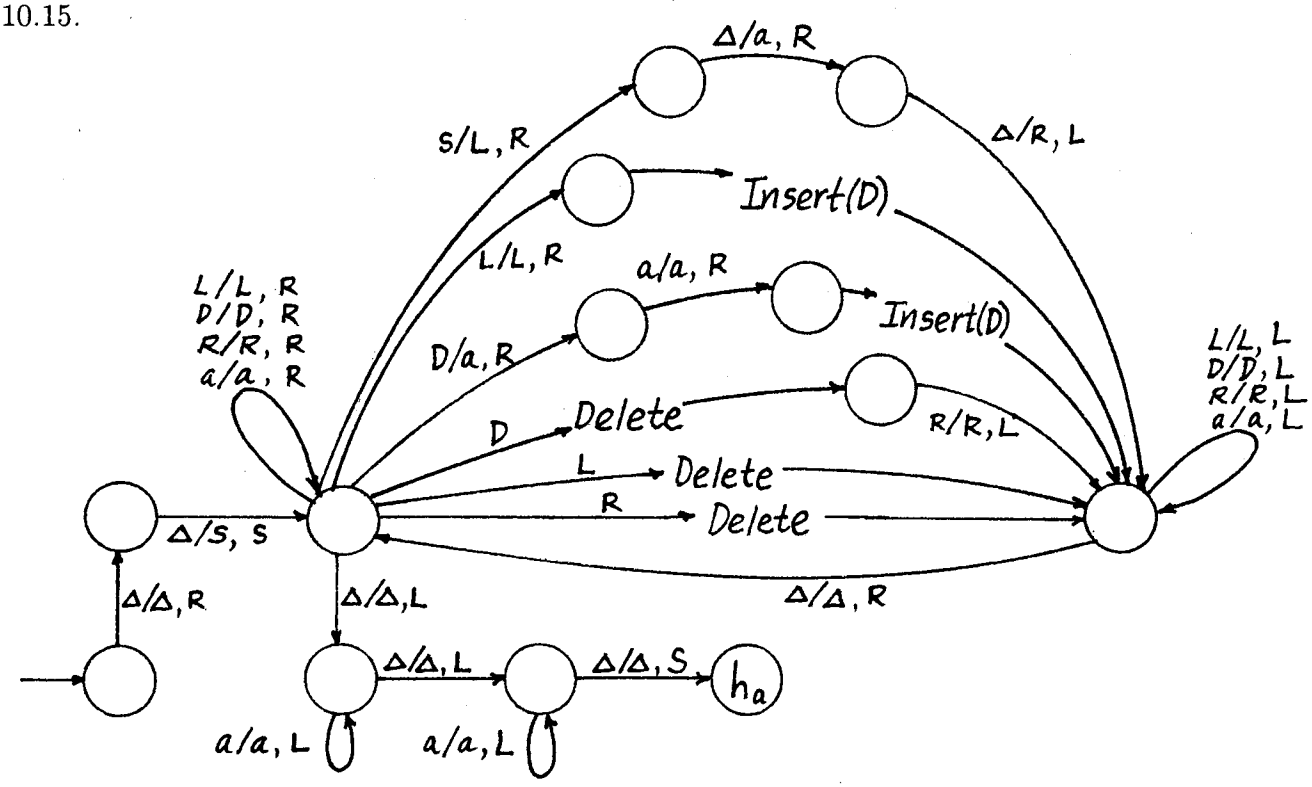
$$S \to LMN \qquad L \to LaA \mid LbB \mid \Lambda \qquad M \to \Lambda \qquad N \to \Lambda$$
$$Aa \to aA \qquad Ab \to bA \qquad Ba \to aB \qquad Bb \to bB$$
$$AM \to MaA \qquad BM \to MbB \qquad AN \to Na \qquad BN \to Nb$$

(d)

$$S \to LMN \qquad L \to LaA \mid LbB \mid \Lambda \qquad M \to \Lambda \qquad N \to \Lambda$$
$$Aa \to aA \qquad Ab \to bA \qquad Ba \to aB \qquad Bb \to bB$$
$$AM \to MA \qquad BM \to MB \qquad AN \to aNa \qquad BN \to bNb$$

Both parts (c) and (d) work on the principle of a variable (either A or B) migrating across the string and depositing the corresponding terminal in each of the three portions of the string. The difference is that in (c) the terminal is deposited at the beginning of each portion and in (d) it is deposited at the beginning of the first portion, the end of the second, and the beginning of the third.

10.15.



10.16.

$$
\begin{aligned}
S &\Rightarrow S(\Delta\Delta) \Rightarrow T(\Delta\Delta) \Rightarrow T(aa)(\Delta\Delta) \\
&\Rightarrow T(bb)(aa)(\Delta\Delta) \Rightarrow T(bb)(bb)(aa)(\Delta\Delta) \\
&\Rightarrow T(aa)(bb)(bb)(aa)(\Delta\Delta) \\
&\Rightarrow q_0(\Delta\Delta)(aa)(bb)(bb)(aa)(\Delta\Delta) \\
&\Rightarrow (\Delta\Delta)q_1(aa)(bb)(bb)(aa)(\Delta\Delta) \\
&\Rightarrow (\Delta\Delta)(a\Delta)q_2(bb)(bb)(aa)(\Delta\Delta) \\
&\Rightarrow (\Delta\Delta)(a\Delta)(bb)q_2(bb)(aa)(\Delta\Delta) \\
&\Rightarrow (\Delta\Delta)(a\Delta)(bb)(bb)q_2(aa)(\Delta\Delta) \\
&\Rightarrow (\Delta\Delta)(a\Delta)(bb)(bb)(aa)q_2(\Delta\Delta) \\
&\Rightarrow (\Delta\Delta)(a\Delta)(bb)(bb)q_3(aa)(\Delta\Delta) \\
&\Rightarrow (\Delta\Delta)(a\Delta)(bb)q_4(bb)(a\Delta)(\Delta\Delta) \\
&\Rightarrow (\Delta\Delta)(a\Delta)q_4(bb)(bb)(a\Delta)(\Delta\Delta) \\
&\Rightarrow (\Delta\Delta)q_4(a\Delta)(bb)(bb)(a\Delta)(\Delta\Delta) \\
&\Rightarrow (\Delta\Delta)(a\Delta)q_1(bb)(bb)(a\Delta)(\Delta\Delta) \\
&\Rightarrow (\Delta\Delta)(a\Delta)(b\Delta)q_5(bb)(a\Delta)(\Delta\Delta) \\
&\Rightarrow (\Delta\Delta)(a\Delta)(b\Delta)(bb)q_5(a\Delta)(\Delta\Delta) \\
&\Rightarrow (\Delta\Delta)(a\Delta)(b\Delta)q_6(bb)(a\Delta)(\Delta\Delta)
\end{aligned}
$$

$$
\begin{aligned}
&\Rightarrow (\Delta\Delta)(a\Delta)q_4(b\Delta)(b\Delta)(a\Delta)(\Delta\Delta)\\
&\Rightarrow (\Delta\Delta)(a\Delta)(b\Delta)q_1(b\Delta)(a\Delta)(\Delta\Delta)\\
&\Rightarrow (\Delta\Delta)(a\Delta)h(b\Delta)(b\Delta)(a\Delta)(\Delta\Delta)\\
&\Rightarrow (\Delta\Delta)h(a\Delta)h(b\Delta)(b\Delta)(a\Delta)(\Delta\Delta)\\
&\Rightarrow h(\Delta\Delta)h(a\Delta)h(b\Delta)(b\Delta)(a\Delta)(\Delta\Delta)\\
&\Rightarrow h(\Delta\Delta)h(a\Delta)h(b\Delta)h(b\Delta)(a\Delta)(\Delta\Delta)\\
&\Rightarrow h(\Delta\Delta)h(a\Delta)h(b\Delta)h(b\Delta)h(a\Delta)(\Delta\Delta)\\
&\Rightarrow h(\Delta\Delta)h(a\Delta)h(b\Delta)h(b\Delta)h(a\Delta)h(\Delta\Delta)\\
&\Rightarrow \Lambda h(a\Delta)h(b\Delta)h(b\Delta)h(a\Delta)h(\Delta\Delta)\\
&\Rightarrow \Lambda a h(b\Delta)h(b\Delta)h(a\Delta)h(\Delta\Delta)\\
&\Rightarrow \ldots \Rightarrow abba
\end{aligned}
$$

10.17. In the grammar in Example 10.2, F will be replaced by either A_1 or B_1, representing the first symbol (either a or b) in the first half. Similarly, M will be replaced by A_2 or B_2.

$$
\begin{gathered}
S \to A_1A_2 \mid B_1B_2\\
A_1 \to A_1aA \mid B_1aB \qquad B_1 \to A_1bA \mid B_1bB\\
Aa \to aA \qquad Ab \to bA \qquad Ba \to aB \qquad Bb \to bB\\
AA_2 \to A_2a \qquad AB_2 \to A_2b \qquad BA_2 \to B_2a \qquad BB_2 \to B_2b\\
A_1 \to a \qquad A_2 \to a \qquad B_1 \to b \qquad B_2 \to b
\end{gathered}
$$

10.18. (a) $S \to a \mid A_1A_2 \qquad A_1 \to A_1A \mid a \qquad A_2 \to a \qquad Aa \to aaA \qquad AA_2 \to aaA_2$
(d)

$$
\begin{gathered}
S \to a \mid A_LA_2aA_R \qquad A_L \to A_LI \mid E\\
Ia \to aI \mid IA_2 \to aA_2JI \qquad IA_R \to aA_2aA_R\\
Ja \to aJ \qquad JA_2 \to A_2J \qquad JA_R \to aA_R\\
Ea \to aE \qquad EA_2 \to aE \qquad EA_R \to aa
\end{gathered}
$$

The idea here is that at some stage, when the current string consists of n groups of n a's, each group be represented as $aa\ldots aA_2$, so that the variable A_2 represents the last a in the group. However, the first such group needs to begin with a special variable A_L to represent the first a in the entire string, and the last such group needs to end with a special variable A_R (instead of A_2) to represent the last a in the string. The case of only one a is taken care of separately. The variables I and J are used more or less as before, except that now each of them actually represents an a, not just a location in the string of a's. The production that is the most different, and perhaps requires a little more explanation, is $IA_R \to aA_2aA_R$. In the grammar in Exercise 10.11(d), the corresponding production was $IR \to A * R$; the

interpretation was that after I had already passed the last $*$, it formed a new group by creating a $*$ and leaving A before it. Here, A_R is within the last group; thus I must first add an a to this group (and at the same time change the A_R to A_2, since this group will no longer be the last one), then create the last group by creating a new A_R.

10.20. If G has the productions $S_1 \to BaB$, $aB \to Ba$, and $B \to b$, then $L(G) = \{bab, bba\}$. Either the grammar used in Chapter 6 to generate $L(G)L(G)$ or the one used to generate $L(G)^*$ would allow S_1S_1 to be generated, and would therefore allow the derivation $S_1S_1 \Rightarrow^* BaBBaB \Rightarrow^* BBBBaa \Rightarrow^* bbbbaa$. However, this string is in neither $L(G)L(G)$ not $L(G)^*$.

10.21. The basic idea is that if T can be restricted to the portion of the tape between square 0 and square $k|x|$, then by using several tracks on the tape it is possible to use another TM that is restricted to the portion between 0 and $|x|$. Therefore, L can be accepted by an LBA.

10.22. A nondeterministic TM can simulate derivations of G as in the proof of Theorem 10.8 but with the following modifications. It can mark a portion of the tape, whose length is $k|x|$, where x is the input string; this portion of the tape will be the one used for simulating the derivation. In addition, farther to the right on the tape, it can keep a record of every string that appears in a derivation. It rejects whenever either the string produced in the simulation is either longer than $k|x|$ or a string that has appeared at some previous step. Eventually, if it does not accept, it must reject in one of these two ways. Therefore, by Theorem 10.2, the language it accepts is recursive; however, it is easy to see that it accepts $L(G)$, since any string in $L(G)$ can be generated by a derivation in which no string is obtained for the second time.

10.23. The productions corresponding to $\delta(p, a) = (q, b, \mathrm{L})$ are

$$\begin{array}{rr} (\sigma_1\sigma_2)p(\sigma_3 a) \to q(\sigma_1\sigma_2)(\sigma_3 b) & (\sigma_1\langle\sigma_2)p(\sigma_3 a) \to q(\sigma_1\langle\sigma_2)(\sigma_3 b) \\ (\sigma_1\sigma_2)p(\sigma_3 a\rangle) \to q(\sigma_1\sigma_2)(\sigma_3 b\rangle) & (\sigma_1\langle\sigma_2)p(\sigma_3 a\rangle) \to q(\sigma_1\langle\sigma_2)(\sigma_3 b\rangle) \\ p(\sigma\langle a) \to q(\sigma\langle b)^L & p(\sigma\langle a\rangle) \to q(\sigma\langle b\rangle)^L \end{array}$$

Those corresponding to $\delta(p, a) = (q, b, S)$ are

$$p(\sigma_1 a) \to q(\sigma_1 b) \qquad p(\sigma_1\langle a) \to q(\sigma_1\langle b) \qquad p(\sigma_1 a\rangle) \to q(\sigma_1 b\rangle) \qquad p(\sigma_1\langle a\rangle) \to q(\sigma_1\langle b\rangle)$$

10.24. Let S be countable and $T \subseteq S$. If T is infinite, then S is certainly infinite also. Since S is countable, we may write $S = \{x_0, x_1, \ldots\}$, where $x_i \neq x_j$ if $i \neq j$. Let i_0 be the first subscript for which $x_{i_0} \in T$; let i_1 be the second such subscript; etc. Then $T = \{x_{i_1}, x_{i_2}, \ldots\}$. It follows that T is countably infinite, since the function that takes j to x_{i_j} is a bijection from $\mathcal{N}$ to T.

10.25. Suppose S is infinite. S must contain an element i_0, since otherwise there would be

a bijection from $\emptyset$ to S and S would be finite. In general, if $i_0, i_1, \ldots, i_k$ have been defined to be distinct elements of S, then there must be an element $i_{k+1} \in S$ that is not i_j for any $j \leq k$—otherwise there would be a bijection from $\{0, 1, \ldots, k\}$ to S. Therefore, S has a countably infinite subset $I = \{i_0, i_1, i_2, \ldots\}$. Now define a function $f : S \to S$ as follows: $f(x) = x$ for every $x \in S - I$, and for every $j \geq 0$, $f(i_j) = i_{j+1}$. Then f is a bijection from S to the proper subset $I \cup \{i_1, i_2, \ldots\}$ of S.

Conversely, if S is finite, then for some n, $S = \{i_0, i_1, \ldots, i_n\}$. It is not difficult to show using induction on n that there cannot be a bijection from S to a proper subset of S.

10.26. Suppose $f : S \to T$ is a bijection. If S is finite, then T is finite and therefore countable. If S is countably infinite, then there is a bijection $i : \mathcal{N} \to S$; the function $f \circ i : \mathcal{N} \to T$ is therefore a bijection, and T is countable. Finally, if T is countable, then since $f^{-1} : T \to S$ is a bijection, S is countable. Therefore, if S is uncountable, so is T.

10.27. The set $S \cap T$ is countable, since it is a subset of T. If $S - T$ were countable, then $(S - T) \cup (S \cap T) = S$ would be also.

10.28. First we can list the set $\mathcal{Q}_0$ of rational numbers x satisfying $0 \leq x < 1$ as follows:

$$0, 1/2, 1/3, 2/3, 1/4, 3/4, 1/5, 2/5, 3/5, 4/5, 1/6, 5/6, 1/7, \ldots$$

(The rule is to list the numbers in nondecreasing order of denominator, leaving out a fraction if it is numerically equal to one that has already appeared.)

Next we construct a list L that contains every integer infinitely often. One such list is

$$0, -1, 0, 1, -2, -1, 0, 1, 2, -3, -2, -1, 0, 1, 2, 3, -4, -3, -2, -1, 0, 1, 2, 3, 4, -5, \ldots$$

Finally, for an integer i, we let $\mathcal{Q}_i = \{i + x \mid x \in \mathcal{Q}_0\} = \{z \in \mathcal{Q} \mid i \leq z < i + 1\}$. Since $\mathcal{Q} = \cup_{i=-\infty}^{\infty} \mathcal{Q}_i$, we can create a list of the elements of $\mathcal{Q}$ by starting with the list L, and for each entry, if it is the jth occurrence of the integer i in L, replacing it by the jth element of $\mathcal{Q}_i$, where we use our listing of $\mathcal{Q}_0$ and the natural bijection from $\mathcal{Q}_0$ to $\mathcal{Q}_i$ determined by adding i.

Although it would be hard to write a formula for the nth element of the list, it is easy enough to find the nth element for a given value of n. Here are the first few:

$$0, -1, 1/2, 1, -2, -1/2, 1/3, 3/2, 2, -3, -3/2, -2/3, 2/3, 4/3, 5/2, 3, -4, \ldots$$

10.29. Recall the convention introduced in Chapter 9, by which there are fixed countably infinite sets $\mathcal{Q}$ and $\mathcal{S}$ from which we choose the states and the tape alphabet, respectively, of any Turing machine. For a given finite set of states, input alphabet, and tape alphabet, the number of Turing machines is finite. Since the set of finite subsets of $\mathcal{S}$ is countably infinite (see Exercise 10.31(b)), and once we fix the tape alphabet there are only a finite number of possible choices for the input alphabet, one application of Theorem 10.13 implies that there are only a countable number of Turing machines having a given set of states.

Since there are only a countably infinite number of finite subsets of $\mathcal{Q}$, another application implies that the set of Turing machines is finite.

10.30. (a) Define $f : S \to 2^{\mathcal{N}}$ as follows: for any $x \in S$, say $x = a_0, a_1, \ldots$, let $f(x)$ be the set $\{n \mid a_n = 1\}$. It is easy to see that f is a bijection.

(b) Suppose $S = \{s_0, s_1, \ldots\}$, and for each $j \geq 0$, let $s_{i,j}$ be the jth term of the sequence s_i. Then define x to be the sequence x_0, x_1, ..., where for each i, $x_i = 1 - s_{i,i}$. In other words, $x_i = 0$ if $s_{i,i} = 1$ and $x_i = 1$ if $s_{i,i} = 0$. Then for any i, since $x_i \neq s_{i,i}$, it follows that $x \neq s_i$. This shows that there can be no enumeration of the elements of S.

The reason this proof is essentially the same as the proof of Theorem 10.15 is that if we let $A_i = f(s_i)$ and $A = f(x)$, where $f : S \to 2^{\mathcal{N}}$ is the function described in the solution to (a), then

$$A = \{i \mid x_i = 1\} = \{i \mid s_{i,i} \neq 1\} = \{i \mid i \notin A_i\}$$

10.31. (a) Countable. This set is a subset of the set in part (b).

(b) Countable. The set is $\cup_{i=0}^{\infty} T_i$, where T_i is the set of all subsets of $\{0, 1, \ldots, i\}$. Each T_i is finite and therefore countable; therefore, their union is countable.

(c) Consider the following sets.

i) The set of nonempty subsets of $\mathcal{N}$
ii) The set of nonempty subsets of $\mathcal{N} - \{0\}$
ii) The set of partitions of $\mathcal{N}$ with two subsets

The first is uncountable, since $2^{\mathcal{N}}$ is. There is a bijection from the first to the second, since there is a bijection from $\mathcal{N}$ to $\mathcal{N} - \{0\}$, and so the second is uncountable. Finally, there is a bijection f from the second to the third. For any nonempty subset A of $\mathcal{N} - \{0\}$, let $f(A)$ be the partition consisting of A and $\mathcal{N} - A$. Then f is onto, because for any two-set partition of $\mathcal{N}$, one of the two sets is a nonempty set not containing 0. Also, f is 1-1, because if the two sets of a partition are A and $\mathcal{N} - A$, only one of them fails to contain 0, so that the partition cannot be both $f(A)$ and $f(\mathcal{N} - A)$.

It follows that the set of two-set partitions of $\mathcal{N}$ is uncountable. This is a subset of the set of all finite partitions, and thus the larger set is uncountable.

(d) A function from $\mathcal{N}$ to $\{0, 1\}$ is nothing more than a sequence a_0, a_1, ...of 0's and 1's. More precisely, there is a natural bijection from the set $\mathcal{F}$ of functions from $\mathcal{N}$ to $\{0, 1\}$ to the set S in Exercise 10.30. It follows from that exercise that $\mathcal{F}$ is uncountable.

(e) There is an obvious bijection f from this set to the set $\mathcal{N} \times \mathcal{N}$: if t is a function from $\{0, 1\}$ to $\mathcal{N}$, take $f(t)$ to be the pair $(t(0), t(1))$. Since $\mathcal{N} \times \mathcal{N}$ is countable, this set is.

(f) This contains the set in (d) as a subset and is therefore uncountable.

(g) Suppose $n \geq 0$, $A = \{a_0, a_1, \ldots, a_n\}$ is a nonempty finite subset of $\mathcal{N}$, and the elements are listed in decreasing order. Then a nonincreasing function f from $\mathcal{N}$ to $\mathcal{N}$ with range A determines a finite set $S_{A,f} = \{j_1, j_2, \ldots, j_n\}$, where $f(i) = a_0$ for $0 \leq i < j_1$, $f(i) = a_1$ for $j_1 \leq i < j_2$, ..., $f(i) = a_n$ for $j_n \leq i$. (If $n = 0$, the set $S_{A,f}$ is empty; otherwise, $j_1 > 0$.) Two different nonincreasing functions with range A determine two

different finite sets. In other words, there is a bijection from the set of nonincreasing functions with range A to a subset of the set of finite subsets of $\mathcal{N}$. It follows from the solution to Exercise 10.31(b) that the set of nonincreasing functions with range A is countable. Since the set of all nonempty finite sets A is also countable (by the same exercise), the set of all nonincreasing functions is a countable union of countable sets and is therefore countable.

(h) and (i) Both countable, since both are subsets of the set of recursively enumerable languages over $\{0, 1\}$.

10.32. The set of all subsets of the even integers has this property. It is uncountable because there is a bijection from $\mathcal{N}$ to the set of even integers, and $2^{\mathcal{N}}$ is uncountable. Its complement is uncountable, because it contains every nonempty subset of the odd integers, and the set of all such subsets is uncountable for the same reason that the set of subsets of the even integers is.

10.33. We know that the set of recursively enumerable languages is countable, and therefore the set of languages whose complement is recursively enumerable is countable. Therefore, the set of languages L so that either L or L' is recursively enumerable is countable. However, the set of languages is uncountable. Therefore, the set of languages L so that neither L nor L' is recursively enumerable is uncountable.

10.34. We consider the contrapositive of the statement. Consider a TM T that halts except on the input strings $x_1, x_2, \ldots, x_n$. We may construct another TM T_1 as follows. T_1 first operates like an FA in order to determine whether the input string is one of the x_i's. (The set $\{x_1, x_2, \ldots, x_n\}$ is a regular language.) If it is, T_1 rejects. Otherwise, T_1 returns the tape head to square 0 and begins to execute T on the original input. Clearly, T_1 accepts precisely the same strings that T does, but T_1 halts on every input. The conclusion is that any language that can be accepted by a TM that enters an infinite loop on only finitely many input strings is recursive.

10.35. Here is a sketch of the construction of a nondeterministic TM T to accept L_1L_2, given TMs T_1 and T_2 accepting L_1 and L_2, respectively. T nondeterministically inserts a blank into the input string at some point, so that the resulting tape is of the form $\Delta x_1 \underline{\Delta} x_2$. T simulates T_2 on the string x_2, and, if and when that simulation accepts, erases the portion of the tape beginning at the end of x_1 and simulates T_1 on x_1. If the input string is of the form x_1x_2, where each $x_i \in L_i$, then the choice of moves that causes the blank to be inserted between them will cause T to accept. If the input is not of this form, then no matter where the blank is inserted, at least one of the simulations will fail to accept.

10.36. The statement $E(f)$ is true if and only if f is Turing-computable.

Suppose first that f is computable. Then the argument sketched just before Theorem 10.6 can be carried out almost without any change to show that if L is recursive then L can be enumerated in order f. In addition, the argument given in the solution to Exercise 10.5 can be adapted to show the converse: the only requirement is that given two strings,

we need to be able to determine which comes first in the ordering f, and we can do this provided f is computable.

Now suppose that for any language L, L is recursive if and only if L can be enumerated in order f. Then in particular, since Σ^* is recursive, Σ^* can be enumerated in order f. This obviously implies that f is computable, since $f(i)$ is simply the ith string in the enumeration of Σ^*.

10.37. Suppose first that f is computable, say by the TM T_f. Then we can construct a TM T to accept $g(f)$ as follows. On an input string $x\#y$, T saves the string y and executes T_f on the string x. If this computation halts, T compares the resulting output to y and halts if and only if they agree. Then if $y = f(x)$, clearly T accepts $x\#y$, and otherwise, T doesn't—either because f is undefined on x, which means that T_f fails to halt on input x, or because $f(x)$ is defined but different from y.

Suppose on the other hand that $g(f)$ is recursively enumerable. Then there is a TM T accepting this set. If $g(f)$ were actually recursive, then in order to compute $f(x)$, our strategy would be to consider the strings $x\#y_1$, $x\#y_2$, ..., where y_1, y_2, ...is an enumeration of Σ^* in canonical order; we would try these strings in order as inputs to T until we found one that was in $g(f)$, and the second part of it would then be $f(x)$. (If we never found one in $g(f)$, the process would never halt, and this would be appropriate because $f(x)$ is undefined in this case.) Since $g(f)$ is assumed only to be recursively enumerable, we must refine this strategy in a way similar to the proof of Theorem 10.6. We execute one move of T on $x\#y_1$; then two moves of T on this string and one on $x\#y_2$; then three moves of T on $x\#y_1$, two on $x\#y_2$, and one on $x\#y_3$; and so forth. If $x\#y_i \in g(f)$ for some i, then we will eventually complete the computation by which T accepts this string, and we will have computed $f(x) = y_i$; otherwise, the process will never halt.

10.38. If L is the range of a computable partial function f, then let T be a TM computing f. If f were a total function, then a strategy to accept L would be, given some input x, to successively compute $f(y_1)$, $f(y_2)$, ...(where y_1, y_2, etc. is an enumeration of Σ^* in canonical order) until one of the values is x, and to accept x if and only if that happened. Since we can't be sure this strategy will work, we make the usual modification: execute T for one step on y_1; then execute T for two steps on y_1 and for one step on y_2; then execute T for three steps on y_1, two steps on y_2, and one step on y_3; etc. This will allow us to compare $f(y_i)$ to x for every i for which the computation of T halts, which means that if x is in the range of f the process will halt on the input string x. If x is not in the range of f, then we will never find an i for which $f(y_i) = x$, and the process will not halt.

Suppose on the other hand that L is recursively enumerable, and let T be a TM accepting L. L is the range of the partial function f, defined by $f(x) = x$ if $x \in L$ and undefined otherwise. We can attempt to compute f on an input x by executing T on x, halting with output x if this computation halts and failing to halt otherwise. Then this process computes f.

10.39. Suppose a virus tester IsSafe exists. Then there is a program D that, on input P, evaluates IsSafe(PP), and either prints out "XXX" (if the value is "NO") or alters the

operating system otherwise. Now consider program D acting on input D. If D is safe on input D, it is because IsSafe(DD) is "NO", which makes IsSafe incorrect. If input D causes D to alter the operating system, there are two possibilities: i) IsSafe(DD) is "YES" (which also means IsSafe is incorrect); or ii) the alteration of the operating system occurs as a result of invoking the program IsSafe, in which IsSafe is not safe. In any case, IsSafe cannot be both safe and correct. (Both the exercise and the solution are taken from the article "There Are No Safe Virus Tests," by William F. Dowling, found in the *American Mathematical Monthly*, November 1989, pp. 835-836.)

10.40. Suppose L is infinite and recursively enumerable. Then there is an algorithm for listing the elements of L. We modify the algorithm so that it examines the strings in the same order but lists each one only if it follows the most recently listed one in canonical order. The modified algorithm obviously lists a subset L_1 of L in canonical order. Moreover, L_1 is infinite, because for each string x, the set of elements of L that precede x in canonical order is finite. Therefore, by Theorem 10.7, L_1 is recursive.

10.41. (a)

$$S \to SB \mid SC \mid SABC \mid BC$$
$$AB \to BA \qquad AC \to CA \qquad BC \to CB$$
$$BA \to AB \qquad CA \to AC \qquad CB \to BC$$
$$A \to a \qquad B \to b \qquad C \to c$$

(b)

$$S \to ABCS \mid ABBCS \mid AABBCS \mid$$
$$BCS \mid BBCS \mid CS \mid BC$$
$$AB \to BA \qquad AC \to CA \qquad BC \to CB$$
$$BA \to AB \qquad CA \to AC \qquad CB \to BC$$
$$A \to a \qquad B \to b \qquad C \to c$$

It is fairly easy to see that any string generated by this grammar satisfies the desired inequalities. The other direction is less obvious. If we let n_A, n_B, and n_C be the number of A's, B's, and C's, respectively, in a string obtained by the first seven productions, and we let $x = n_A$, $y = n_B - n_A - 1$, and $z = 2n_C - n_B - 1$, then we may write the equations

$$\begin{array}{lllllllllllll} x & = & n_1 & + & n_2 & + & 2n_3 & & & & & & \\ y & = & & & n_2 & & & + & n_4 & + & 2n_5 & & \\ z & = & n_1 & & & & & + & n_4 & & & + & 2n_6 \end{array}$$

where, for each $i \leq 6$, n_i is the number of times the ith production is used. It is sufficient to show that for any choice of x, y, and z whose sum is even, there are nonnegative integer values for all the x_i's that satisfy the equations. This can be checked by considering cases.

When x, y, and z are all even, for example, it is easy to see that there is a solution in which n_1 is the minimum of x and z and $n_2 = n_4 = 0$.

(c)

$$S \to LA*R \qquad A \to a$$
$$L \to LA*I \qquad IA \to AI \qquad I* \to A*I \qquad IR \to R$$
$$L \to E \qquad EA \to AE \qquad E* \to E \qquad ER \to \Lambda$$

10.42. We will show that each production $\alpha \to \beta$, where $|\beta| \geq |\alpha|$, can be replaced by a set of productions of the desired form, so the new grammar generates the same language.

Suppose $\alpha = \gamma_1 A_1 \gamma_2 A_2 \gamma_3 \dots \gamma_n A_n \gamma_{n+1}$, where the A_i's are variables and each γ_i is a string of terminals. The first step is to introduce new variables $X_1, \dots, X_n$, and productions of the desired type that allow the string $\gamma_1 X_1 \gamma_2 \dots \gamma_n X_n \gamma_{n+1}$ to be generated. These new variables can appear only beginning with the string α, and will be used only to obtain the string β; the effect of this first step is to guarantee that none of the productions we are adding will cause strings not in the language to be generated. The first production is $\gamma_1 A_1 \gamma_2 \dots \gamma_n A_n \gamma_{n+1} \to \gamma_1 X_1 \gamma_2 \dots \gamma_n A_n \gamma_{n+1}$, the next allows A_2 to be replaced by X_2, and so forth; the nth production allows A_n to be replaced by X_n.

The second step is to introduce additional variables, and productions that allow us to generate a string $Y_1 Y_2 \dots Y_k$, where each Y_i is a variable and $k = |\alpha|$. For example, if $\gamma_1 = abc$, we can start with the productions $cX_1 \to Y_3 X_1$, $bY_3 \to Y_2 Y_3$, and $aY_2 \to Y_1 Y_2$. The variable Y_4 will be the same as X_1. Similarly, if $\gamma_2 = defg$, we would use $X_1 d \to X_1 Y_5$, $Y_5 e \to Y_5 Y_6$, etc. All these productions are of the right form, and they obviously allow us to obtain the string $Y_1 \dots Y_k$. The Y_i's, like the X_i's, are reserved for this purpose and are used nowhere else in the grammar.

The third step is to add still more productions that produce the string β. Let $\beta = Z_1 Z_2 \dots Z_k Z_{k+1} \dots Z_m$, where $m \geq k$ and each Z_i is either a variable or a terminal. The productions we need are $Y_1 Y_2 \to Z_1 Y_2$, $Y_2 Y_3 \to Z_2 Y_3$, $\dots$, $Y_{k-1} Y_k \to Z_{k-1} Y_k$, and $Z_{k-1} Y_k \to Z_{k-1} Z_k Z_{k+1} \dots Z_m$.

It is clear that all the productions we have added are of the proper form, and that they permit β to be derived from α. Furthermore, each of the new productions has a new variable on the right side except the last production mentioned above; and the left side of this last production can have been obtained only starting from the string α. The conclusion is that only the strings derivable in the original grammar can be derived in the new one.

10.45. (a) Let S_1 and S_2 be the start symbols of G_1 and G_2, respectively. Let G have new start symbol S, plus all the variables in G_1 and G_2, plus three other new variables L, M, and R. G will have all the productions in G_1 and G_2, together with the new productions

$$S \to LS_1 M S_2 R \qquad L\sigma \to \sigma L \qquad LM \to L \qquad LR \to \Lambda$$

(for every $\sigma \in \Sigma$). The idea is that M prevents any interaction between the productions of G_1 and those of G_2. L must eventually move to the right side of the string, in order to be eliminated. This means that it must first pass M, and once this has happened (using

the production $LM \to L$), there are never any variables preceding M in the string. This implies that the derivation in G_1 must be completed before $LM \to L$ is applied, and it is impossible for any production to be used whose left side involves both terminals produced by G_1 and variables in G_2.

(b) If S_1 is the start symbol of G_1, G will have the variables of G_1 as well as S (the new start symbol), S', L, M, and R. The productions of G will be

$$S \to \Lambda \mid LS' \qquad S' \to S_1MS' \mid S_1R$$
$$L\sigma \to \sigma L \qquad LM \to L \qquad LR \to \Lambda$$

The principle is similar to that in (a). From S we can get strings of the form Λ, LS_1R, LS_1MS_1R, $LS_1MS_1MS_1R$, etc. The derivation from one S_1 can't interfere with the derivation from another because of the presence of the M's.

10.46. The crucial aspect of the formula $A = \{i \mid i \notin A_i\}$ is that it specifies a function $f : \mathcal{N} \to \mathcal{N}$ so that for every i, $f(i) \in A_i$ if and only if $f(i) \notin A$. (The function is $f(i) = i$.) This allows us to conclude that for every i, $A \neq A_i$. The proof would work just as well with any other one-to-one function f. For example, we could let A be $\{2i \mid 2i \notin A_i\}$, or $\{3^i \mid 3^i \notin A_i\}$. (If f is not 1-1, this is not guaranteed to work. If $f(i) = f(j)$, for example, we might want $f(i)$ to be in A because it is not in A_i and $f(j)$ not to be in A because it is in A_j.)

10.47. (a) The function that takes s to $\{s\}$ is a bijection from S to a subset of 2^S. (b) To show that there is no bijection from S to 2^S, suppose f is such a bijection, and let $A = \{x \in S \mid x \notin f(x)\}$. Then since f is onto and $A \in 2^S$, $A = f(x_0)$ for some $x_0 \in S$. We ask whether x_0 can be an element of $f(x_0)$. On the one hand, if $x_0 \in f(x_0)$, then x_0 does not satisfy the defining condition for A, and so $x_0 \notin A = f(x_0)$. On the other hand, if $x_0 \notin f(x_0)$, then by definition of A, $x_0 \in A = f(x_0)$. Thus we have a contradiction, and we may conclude that the bijection f is impossible.

10.48. (a) For each n, there is a bijection between the $(n+1)$-fold Cartesian product $\mathcal{N} \times \mathcal{N} \times \ldots \times \mathcal{N}$ and the set of all integer polynomials of degree n. Therefore, the set of polynomials of degree n is countable, and thus the set of real numbers that are roots of such polynomials is countable. The set of all such numbers is the union of these sets, from $n = 0$ to ∞, and is therefore countable.

(b) For every subset $S = \{n_0, n_1, \ldots\}$ of $\mathcal{N}$ (assume the n_i's are listed in increasing order) we may consider the function f which is 0 for all i with $0 \leq i < n_0$, 1 for all i with $n_0 \leq i < n_1$, 2 for all i with $n_1 \leq i < n_2$, etc. This correspondence defines a bijection from $2^{\mathcal{N}}$ to a subset of the set of all nondecreasing functions from $\mathcal{N}$ to $\mathcal{N}$. Therefore, the set of such functions is uncountable.

(c) This set contains the one in Exercise 10.31(d) as a subset and is therefore uncountable.

(d) The set is finite, by an argument essentially the same as that in the solution to Exercise 10.31(g).

10.48. (e) Suppose f is periodic, and $P > 0$ is a *period* (i.e, $f(x + P) = f(x)$ for every x). Let $f_1 : \{0, 1, \ldots, P - 1\} \to \mathcal{N}$ be defined by $f_1(n) = f(n)$ for every n with $0 \leq n \leq P - 1$. Then if g is any periodic function other than f for which P is a period, the corresponding function g_1 is different from f_1. This means that there is a bijection from the set $\mathcal{F}_P$ of all functions having period P to the set of all functions from $\{0, 1, \ldots, P - 1\}$ to $\mathcal{N}$. Since the second set is countable (see part (e)), the set $\mathcal{F}_P$ is. It follows that the set of all periodic functions is also, since this set is the union of all the $\mathcal{F}_P$'s for $P = 1, 2, \ldots$.

(f) We can think of an eventually periodic function as a periodic function which, for some n, has had its values at the first n integers changed arbitrarily. For a given periodic function f, there are only a countable number of n's for which this could happen, and for each n only a countable number of ways of changing the function at the first n integers. Therefore, the set of functions that are eventually equal to f is countable. Since the set of periodic functions is countable, the set of all eventually periodic functions is countable.

(g) This set is a subset of (f) and is therefore countable.

10.49. (a) If we define $f_0 : A_0 \to B$ by $f_0(x) = f(x)$, then f_0 is obviously one-to-one, since f is. For any $x \in A_0$, $f_0(x) \in B_1$, because the number of ancestors of $f_0(x)$ is one plus the number of ancestors of x. Finally, for any $y \in B_1$, $y = f(x)$ for some $x \in A_0$. Therefore, f_0 is a bijection from A_0 to B_1. Similarly, g_0 is a bijection from B_0 to A_1. It is also easy to see that the function f_∞ defined by $f_\infty(x) = f(x)$ is a bijection from A_∞ to B_∞.

Now we define $F : A \to B$ by letting $F(x) = f(x)$ if $x \in A_0$ or $x \in A_\infty$ and $F(x) = g^{-1}(x)$ if $x \in A_1(x)$. It follows that F is a bijection from A to B.

(b) Let $f : A \to B$ and $g : B \to C$ be the two bijections. Then $g \circ f$ is a bijection from A to a subset of C. If there were a bijection $h : A \to C$, then h^{-1} would be a bijection from C to A. Thus $h^{-1} \circ g$ would be a bijection from B to a subset of A. The Schröder-Bernstein Theorem would then imply that there was a bijection from B to A, but we are assuming this is not the case.

(c) This is an immediate consequence of the Schröder-Bernstein Theorem.

(d) If A is an infinite set, A has a countably infinite subset B by Lemma 10.1. It follows that if A is uncountable and C is countable, A is larger than C. If A is infinite and C is countable, A cannot be smaller than C: If there is a bijection from A to a subset S of C, since S cannot be finite, S must be countably infinite, and therefore A is also.

10.50. The function $f : I \to I \times I$ defined by $f(x) = (x, 0)$ is a bijection from I into a subset of $I \times I$. Now we define a bijection g from $I \times I$ into a subset of I. Given an element (x, y) of $I \times I$, let $0.x_0x_1x_2\ldots$ and $0.y_0y_1y_2\ldots$ be the decimal expansions of x and y, respectively (that do not end in infinite strings of 9's), and let $g(x, y)$ be the real number with decimal expansion $0.x_0y_0x_1y_1\ldots$. Then $g(x, y) \in I$, and g is a bijection from $I \times I$ to a subset of I. (In fact, g is not onto, since for example the number with expansion $0.1119191919\ldots$ is not in the range of g.) It follows that there is a bijection from I to $I \times I$.

10.51. Suppose there is a 1-1 function f from B to A and none from A to B. Then f can be interpreted as a bijection from B to its range, which is a subset of A. If there is no 1-1

function from A to B, then there can certainly not be a bijection from B to A. Therefore, A is bigger than B.

Now suppose A is bigger than B. Then there is a bijection from B to a subset of A, which can be interpreted as a 1-1 function from B to A. Suppose there is a 1-1 function from A to B. Then it determines a bijection from A to a subset of B, and it follows from the Schröder-Bernstein Theorem that there is a bijection from A to B, which is impossible.