

Solutions to Selected Exercises for Chapter 12 of R. Gregory Taylor, *Models of Computation and Formal Languages* (Oxford University Press: New York, 1998)

© Oxford University Press

Solutions to Exercises for § 12.1

12.1.2. The productions of Groups (6)–(9) are generally not context-free. The productions of Group (5), as well as the final production of Group (9), are not context-sensitive.

Solutions to Exercises for § 12.3

12.3.2 (c) Thus suppose that M is given a representation of 100 as input. Referring to the chart of (b), we see that there are $2^{\lfloor \log_2 100 \rfloor} - 1 = 2^6 - 1 = 63$ words of length 5 or less. It is thus reasonable to have M map argument 100 onto the 37th word over Σ^* of length 6. What sort of machine would M have to be in order to do this? Well, given input n , M might first calculate $\lfloor \log_2 n \rfloor$. It could then simulate the behavior of the machine of (b) on argument pair $\lfloor \log_2 n \rfloor$ and $n - (2^{\lfloor \log_2 n \rfloor} - 1)$.

(d) We start by considering the following table.

n	number of words w in Σ^* with $n_a(w) \leq n_b(w)$
0	1
1	1
2	3
3	4
4	11
...	...
odd k	2^{k-1}
even k	$2^{k-1} + \binom{k}{k/2} / 2$

So an elegant design might make use of this information. Surely easier to describe is a multitape machine M that would work as follows. Given input n and using ideas from (b) and (c), M would generate words of Σ^* in succession on one of its worktapes. After generating each word w , M would determine whether $n_a(w) \leq n_b(w)$. If so, then M would increment a counter on another of its worktapes. Continuing in this way, M would output the n^{th} word w generated such that $n_a(w) \leq n_b(w)$.

12.3.3 (a) $111 \rightarrow 111$
 $11 \rightarrow abb$
 $1 \rightarrow ab$

(c) Suppose that L contains n words $w_1, w_2, \dots, w_{n-1}, w_n$. The following $(n+1)$ -member production sequence will suffice.

$$\begin{aligned} I^{n+1} &\rightarrow I^{n+1} \\ I^n &\rightarrow w_n \\ I^{n-1} &\rightarrow w_{n-1} \\ &\dots \\ II &\rightarrow w_2 \\ I &\rightarrow w_1 \end{aligned}$$

Solutions to Exercises for § 12.4

12.4.1. In both directions, our proof will involve the simulation of one (single-tape) Turing machine by another (multitape) Turing machine. We shall want to appeal to the following items.

- Two equivalent conceptions of Turing machines as language acceptors were introduced: Turing machines may *accept by I* or they may *accept by terminal state*. Let L be a language over alphabet Σ . Then there exists a Turing machine that accepts L by I if and only if there exists a Turing machine that accepts L by terminal state. (This was Theorem 2.3.)
- We may assume without loss of generality that if Turing machine M does not accept word w , then M never halts if started scanning the leftmost symbol of w on an otherwise blank tape (see Exercise 2.10.8(c)).
- Suppose that Turing machine M enumerates language L . This means that M maps a subset \mathcal{N}'_1 of \mathcal{N}'_1 onto L . If \mathcal{N}'_1 is a proper subset of \mathcal{N}'_1 then there must exist natural numbers not mapped onto words of L . Suppose that k is such a number. Then, by Definition 12.1, we may assume that M never halts when started scanning the leftmost I of a representation of k on an otherwise blank tape.

Proof (reverse direction). Suppose that L is the language accepted by single-tape Turing machine M_{accept} . Without loss of generality we may assume that M_{accept} accepts by terminal state. By Theorem 12.3, it will be sufficient to describe a multitape Turing machine M_{enum} that enumerates L . Consider first the following description of M_{enum} , which does not quite work but which is a reasonable first approximation to something that does work. Initially, M_{enum} has a representation of natural number n on its input tape. On worktape₁, M_{enum} proceeds to generate, in sequence, words over Σ in order of increasing length; words of identical length are generated in lexicographic order. Thus, words $\epsilon, a, b, aa, ab, ba, bb, aaa, \dots$ are generated in succession. As each word w is generated, M_{enum} simulates M_{accept} on w so as to determine whether M_{accept} accepts w . Moreover, M_{enum} keeps track of how many generated words have been accepted by (the simulation of) M_{accept} so far. In the meantime, M_{enum} continues generating words over Σ , simulating M_{accept} and counting accepted words until it has identified the n^{th} word accepted by M_{accept} . At this point, M_{enum} writes the n^{th} word accepted to its output tape and halts.

Of course, the algorithm just described does not quite work since the simulation of M_{accept} 's behavior for any nonaccepted word will, by assumption, never terminate. We get around this difficulty by way using dovetailing. Thus M_{enum} 's operation may be described in phases as follows, where we begin writing w_i for the i^{th} word generated by M_{enum} on its worktape₁.

Phase 1. M_{enum} simulates M_{accept} on w_1 for one step only.

Phase 2. M_{enum} simulates M_{accept} on w_1 for two steps only and then simulates M_{accept} on w_2 for two steps only.

Phase 3. M_{enum} simulates M_{accept} on w_1 for three steps only, then simulates M_{accept} on w_2 for three steps only, and then simulates M_{accept} on w_3 for three steps only.

...

Phase k . M_{enum} simulates M_{accept} on each of w_1 through w_k , in turn, for k steps only.

Given argument n , M_{enum} will ultimately write, on its output tape, the n^{th} word—called it W_n —accepted during the course of this succession of partial simulations of M_{accept} . Note that, since M_{enum} can always recognize when its simulation of M_{accept} has entered a terminal state, M_{enum} will have no problem recognizing when a word has in fact been accepted by M_{accept} . Any word accepted by M_{accept} will be W_n for some n and hence will belong to the language enumerated by M_{enum} . Conversely, any word in the language enumerated by M_{enum} is so by virtue of M_{accept} 's having accepted it. By the Principle of Extensionality, the language accepted by M_{accept} is none other than the language enumerated by M_{enum} .

As for the **forward direction**, suppose that L is Turing-enumerated by single-tape Turing machine M_{enum} . We now describe a multitape Turing machine M_{accept} that accepts language L . As before, we shall first describe an idealization of M_{accept} that will not work in general, afterward suggesting how it can be altered so as to yield a machine that *will* work. M_{accept} starts with some given word w over Σ on its input tape. M_{accept} 's operation consists of generating representations of natural numbers $0, 1, 2, \dots$ in succession on one of its worktapes. As each natural number is generated, M_{accept} simulates the operation of M_{enum} so as to produce in succession W_0, W_1, W_2 , and so on upon another of its worktapes. (Again, we are writing W_i for that word over Σ that is the image of natural number i under $f_{M_{enum}}$.) After any such simulation has been completed, M_{accept} compares word W_i with input word w . If the two are identical, then M_{accept} writes a single 1 on its output tape and halts, thereby accepting w . On the other hand, if word W_i is not identical with w , then M_{accept} goes on to generate the next natural number $i+1$, simulate M_{enum} so as to obtain its image W_{i+1} , and then compare w with W_{i+1} . Clearly, if $w \in L$, then eventually w will be W_n for some finite n . On the other hand, if $w \notin L$, then M_{accept} will continue generating natural numbers, simulating the computation of their images, and comparing ad infinitum.

As described, M_{accept} will not quite work since, if M_{enum} just happens to be undefined for a particular k , then, by assumption, M_{accept} 's simulation of M_{enum} for argument k will never halt. This problem can be handled using dovetailing once again. Rather than simulating M_{enum} for argument $i+1$ only after its simulation of M_{enum} for argument i has been *completed*, M_{accept} will incrementally simulate M_{enum} for argument 0 , then for arguments 0 and 1 , then for arguments $0, 1$, and 2 , and so on. It is evident that the language accepted by M_{accept} will be none other than L . Q.E.D.

12.4.2. **Proof.** Suppose that L is r.e. This means that C_L is an r.e. set of natural numbers and that C_L is the image of some subset of \mathcal{N} under a partial recursive function f . By Theorem 3.3(a), f is Turing-computable. So let M be a Turing machine that computes f . We modify M in order to obtain a new machine M' . The modification amounts to appending to M a decoding routine that transforms natural number $f(n)$, which M produces as output, into that unique word w of L such that $\ulcorner w \urcorner = f(n)$. It should be clear that M' (Turing-)enumerates L . Hence L is Turing-enumerable and, by Theorem 12.4, Turing-acceptable as well. Q.E.D.

Here is an alternative proof that does not involve decoding gödel numbers, which we have avoided by appealing to Theorem 3.5.

Proof. Suppose that L is r.e. By Definition 12.4, this means that $C_L = \{\ulcorner w \urcorner \mid w \in L\}$ is an r.e. set of natural numbers. By Theorem 3.5, C_L is $Dom(f)$ for some partial recursive f . Of course, this f is Turing-computable as well. So let M be a Turing machine that computes it. By Exercise 2.10.8(a), we may assume that M 's computation never terminates for input $n \notin C_L$. We next describe a (multitape) Turing machine M^* that accepts L . Suppose that M^* starts scanning the leftmost symbol of word $w \in \Sigma^*$. First, M^* calculates $\ulcorner w \urcorner$ and then simulates M on it. If this simulation eventually halts, then by assumption $\ulcorner w \urcorner \in Dom(f) = C_L$ and $w \in L$. So, in such a case, M^* will write an accepting 1 to its output tape before halting. Otherwise, M^* simulates M for input $\ulcorner w \urcorner$ forever. It is apparent that M^* accepts L and that, consequently, L is Turing-acceptable. Q.E.D.

12.4.3 (a) Suppose, for the sake of producing a contradiction, that the class of phrase-structure languages is closed under relative complementation. Let A be an arbitrary phrase-structure language. We have $A^c = \Sigma^* \setminus A$ by definition. Since Σ^* is a phrase-structure language, it follows that A^c is also a phrase-structure. But A was an arbitrary phrase-structure language whose complement A^c has now been shown to be a phrase-structure language. In other words, our assumption has led us to the conclusion that the class of phrase-structure languages is closed under complement after all, contradicting Theorem 12.7. We conclude that, contrary to our assumption, the class of phrase-structure languages is *not* closed under relative complementation.

(b) The proof can be patterned on that of (a). It is convenient to use (a) together with the fact that $A \setminus B = (A \oplus B) \cap A$.

12.4.6. Our proof takes the form of a sequence of biconditionals.

L is a recursive language $\Leftrightarrow C_L$ is a recursive set of natural numbers (Definition 12.4)
 \Leftrightarrow both C_L and its complement are r.e. sets of natural numbers (Theorems 3.7 and 3.9)
 \Leftrightarrow both L and its complement are r.e. languages (Definition 12.4 and Remark 12.4.1)
 \Leftrightarrow both L and its complement are Turing-acceptable languages (Theorem 12.5)
 $\Leftrightarrow L$ is Turing-recognizable (Exercise 2.10.11) Q.E.D.

12.4.7. See the first three lines in the proof of Exercise 12.4.6 above.

Solutions to Exercises for § 12.5

12.5.1. Recall, first, that to pronounce a function effectively computable is just to say that one possesses some algorithm for computing its values. So suppose now that there exists an effective method \mathcal{M} for determining membership in $L(G^*)$ with $G^* = \langle \Sigma, \Gamma, \sigma, \Pi \rangle$. Let unary function f_{G^*} be defined by

$$f_{G^*}(\ulcorner w \urcorner_{\text{revised}}) = \begin{cases} 1 & \text{if } \sigma \Rightarrow^* w \\ 0 & \text{otherwise} \end{cases}$$

where $w \in \Sigma^*$. We wish to see that f_{G^*} is effectively computable. Here is an algorithm for computing it.

Algorithm for computing f_{G^*}
 Input: natural number n
 Output: 1 if $\sigma \Rightarrow^* w$ and 0 otherwise
 Begin
 find w such that $\ulcorner w \urcorner_{\text{revised}} = n$; /* There is exactly one such w . */
 apply \mathcal{M} to w ;
 if applying \mathcal{M} reveals that $w \in L(G^*)$ then return 1 else return 0
 End.

12.5.2. The answer to this question is no. In fact, the word problem for any context-sensitive language is solvable by Theorem 11.7. What Theorem 12.8 asserts is that the *general* word problem for phrase-structure languages is not solvable. In other words, there is no *general* algorithm that may be applied to *any* phrase-structure language L over Σ and to any word w over Σ so as to effectively decide whether $w \in L$. It is still entirely consistent with Theorem 12.8 that there should exist certain phrase-structure languages L' for which membership is effectively decidable by virtue of some algorithm that works *in that particular case*. By Theorem 11.7, any context-sensitive language is just such an L' .

In general, confusion will arise if one fails to distinguish between the word problem for a *particular* language L and the *general* word problem for a family of languages. We note the following.

- If the general word problem for a family of languages is solvable, then the word problem for any member of that family is obviously solvable as well.
- Conversely, if the word problem for a given language L is not solvable, then plainly the general word problem for any family of languages to which L belongs will not be solvable either.
- On the other hand, the general word problem for a given family may itself be unsolvable and yet the word problem for some particular member of that family may nonetheless be solvable. This was the issue above.

Solutions to Exercises for § 12.6

12.6.3. We have solution

$$S \rightarrow a'S'a' = abSba \rightarrow a'b'b'a' = abba$$

which corresponds to index sequence

$$\langle 1,4,16,9,5,9,15,12,13,8,13,12,16,9,10,10,9,2 \rangle$$

Once again, we note that, assimilating $\Sigma \cup \Gamma$ to $\Sigma' \cup \Gamma'$ and ignoring the duplication $a'b'b'a'abba$ at the very end, this sequence of sentential forms amounts to a derivation of $w=abba$ within G .

12.6.4 (a) L_3 is generated by the context-free grammar with productions

$$\begin{aligned} S &\rightarrow a_i S a_i \text{ for } i=1 \dots m \\ S &\rightarrow S_1 \\ S_1 &\rightarrow b_j S_1 b_j \text{ for } j=1 \dots n \\ S_1 &\rightarrow c \end{aligned}$$

It should also be clear that L_3 is accepted by a deterministic pushdown-stack automaton. It follows, by Theorem 10.13, that L_3^c is context-free.

(b) We describe a deterministic pushdown-stack automaton M that accepts $L_4 = \{ ucv^R \mid u \in L_1 \text{ and } v \in L_2 \}$ by empty stack, say. Here u is a word of the form $x_{i_1} x_{i_2} \dots x_{i_k} b_{i_k} \dots b_{i_2} b_{i_1}$. Roughly, M will push (representatives of) the symbols constituting word x_{i_1} as they are read. Later, M will pop all of them in reverse order upon reading symbol b_{i_1} . This association of word x_{i_1} and symbol b_{i_1} must be built into M . After encountering symbol c , M enters a second mode appropriate for verifying that the second half of its input word is of the form $v^R = (y_{i_1} y_{i_2} \dots y_{i_j} b_{i_j} \dots b_{i_2} b_{i_1})^R$ as required by the definition of L_4 . Now when b_{i_1} is read, (representatives of) each symbol within word y_{i_1} must be pushed—to be popped later just in case word y_{i_1} is itself encountered. Machine M is deterministic since the association of word pair $\langle x_{i_1}, y_{i_1} \rangle$ with symbol b_{i_1} is given in advance and hence can be incorporated into the design of M . It then follows, by Theorem 10.13, that both L_4 and L_4^c are context-free.

(c) (\Leftarrow) If $L_5 = \Sigma^*$, then plainly L_5 is regular.

(\Rightarrow) Note, first, that there is a 1-1 correspondence between solutions to PCP for word-pair sequence

$$\begin{aligned} &\langle x_1, y_1 \rangle \\ &\langle x_2, y_2 \rangle \\ &\dots \\ &\langle x_n, y_n \rangle \end{aligned} \tag{12.6.7}$$

and members of $L_5^c = (L_3^c \cup L_4^c)^c = L_3 \cap L_4$. Suppose, for the sake of proving a contradiction, that L_5 is regular but that $L_5^c \neq \emptyset$. By Theorem 9.8, L_5^c is regular. By the cited 1-1 correspondence and (the converse of) Remark 12.6.1, one can see that L_5^c must be infinite. Since L_5^c is regular and infinite, the Pumping Lemma for regular languages (Lemma 9.2) applies. It follows that there exist words u , w , and v with $w \neq \epsilon$ such that $uw^i v \in L_5^c = L_3 \cap L_4$ for all $i \geq 0$. But reflection on the definition of $L_3 \cap L_4$ shows that this is a contradiction. We must conclude that if L_5 is regular, then $L_5^c = \emptyset$, in which case $L_5 = \Sigma^*$.

(d) (\Rightarrow) To begin, we observe once again that there is a 1-1 correspondence between members of L_5^c and solutions of PCP for word-pair sequence (12.6.7). Next, suppose for the sake of deriving a contradiction that L_5^c is context-free but that $L_5^c \neq \emptyset$. By the fact that $L_3 \cap L_4 = (L_3^c \cup L_4^c)^c = L_5^c \neq \emptyset$ we see that solutions to PCP exist. But we know that if one solution exists, then infinitely many distinct solutions exist. This means that language $L_3 \cap L_4$ is infinite. Since it is context-free, it follows by the Pumping Lemma for Context-

Free Languages (Theorem 10.3) that there exist words $u, v, w, x,$ and y with v and x not both ε and such that $uv^iwx^iy \in L_3 \cap L_4$ for all $i \geq 0$. But reflection on the definition of $L_3 \cap L_4$ shows that this is a contradiction. We must conclude that if L_5^c is context-free, then $L_5^c = \emptyset$. The other direction (\Leftarrow) is trivial since \emptyset is context-free.

12.6.5 (a) We noted that there is a 1-1 correspondence between members of $L_3 \cap L_4 = L_5^c$ and solutions to PCP for word-pair sequence

$$\begin{array}{l} \langle x_1, y_1 \rangle \\ \langle x_2, y_2 \rangle \\ \dots \\ \langle x_n, y_n \rangle \end{array} \quad (12.6.7)$$

Also, by Exercise 12.6.4(c), language L_5 is context-free. It follows that any algorithm that could be used to decide whether an arbitrary context-free language has empty complement would in effect provide a decision procedure for PCP for an arbitrary sequence of word pairs. Consequently, by Theorem 12.9, there can be no such algorithm. Once again, we have reduced PCP for an arbitrary sequence of word pairs to a second problem and thereby shown that this second problem is unsolvable.

(b) Given an arbitrary sequence of word pairs, the language L_5 defined in terms of that sequence was seen to be context-free in Exercise 12.6.4(c). The mentioned general algorithm, if it existed, could then be used to determine whether $L_5^c = L_3 \cap L_4$ is infinite. If L_5^c turns out to be infinite, then, by the cited 1-1 correspondence, there exist infinitely many solutions to PCP for the sequence of word pairs that gave us L_5 . If not, then, by Remark 12.6.1, there exist no solutions to PCP for that sequence of word pairs. In other words, any algorithm that could be used to decide whether an arbitrary context-free language has infinite complement would in effect provide a decision procedure for PCP for an arbitrary sequence of word pairs.

(c) Suppose, for the sake of producing a contradiction, that we possess such an algorithm. Then for given word pair sequence

$$\begin{array}{l} \langle x_1, y_1 \rangle \\ \langle x_2, y_2 \rangle \\ \dots \\ \langle x_n, y_n \rangle \end{array} \quad (12.6.7)$$

we can apply our algorithm to context-free language L_5 , in particular, in order to determine whether L_5^c is regular. But

$$\begin{array}{ll} L_5^c \text{ is regular} & \Leftrightarrow L_5 \text{ is regular} & \text{by Theorem 9.8} \\ & \Leftrightarrow L_5 = \Sigma^* & \text{by Exercise 12.6.4(c)} \\ & \Leftrightarrow L_5^c = \emptyset & \\ & \Leftrightarrow \text{PCP for word pair sequence (12.6.7)} & \text{by 1-1 correspondence} \\ & \text{has no solutions} & \end{array}$$

In other words, the problem of deciding whether PCP for an arbitrary sequence of word pairs has a solution is reducible to the problem of deciding whether the complement of an arbitrary context-free language is regular. Consequently, since the former problem is not solvable, neither can the latter be solvable.

(d) This is similar to (c) except that we use Exercise 12.6.4(d). We note that

$$\begin{array}{ll} L_5^c \text{ is context-free} & \Leftrightarrow L_5^c = \emptyset & \text{by Exercise 12.4.6(d)} \\ & \Leftrightarrow \text{PCP for word pair sequence (12.6.7)} & \text{by 1-1 correspondence} \\ & \text{has no solutions} & \end{array}$$

Thus the problem of deciding whether PCP for an arbitrary sequence of word pairs has a solution is reducible to the problem of deciding whether the complement of an arbitrary context-free language is itself context-free. Since the former is not solvable, neither is the latter.

- 12.6.6 (a) This follows immediately from Exercise 12.6.5(a). After all, the answer to “Does L contain every word?” is yes if and only if the answer to the question “Is L^c empty?” is yes. Hence any procedure for deciding the former question provides a means of deciding the latter question as well.
 (b) Suppose that we possess such an algorithm. Then we can effectively decide whether the complement of a context-free language is regular, contradicting Exercise 12.6.5(c).

12.6.7. Suppose that we had such an algorithm. Then for an arbitrary sequence of word pairs

$$\begin{array}{l} \langle x_1, y_1 \rangle \\ \langle x_2, y_2 \rangle \\ \dots \\ \langle x_n, y_n \rangle \end{array} \quad (12.6.7)$$

we could apply this algorithm to context-free languages L_3 and L_4 so as to determine whether $L_3 \cap L_4 = L_5^c$ is context-free. By Exercise 12.6.4(d), L_5^c is context-free if and only if L_5^c is empty. So we, in effect, have an algorithm for deciding whether L_5^c is empty. But by the 1–1 correspondence between words of L_5^c and solutions to PCP for sequence (12.6.7), we have a decision procedure for PCP for an arbitrary sequence of word pairs, contradicting Theorem 12.9.

12.6.8. Suppose that we are in possession of such an algorithm and let L be an arbitrary context-free language. Language Σ^* is context-free, so we can apply our algorithm to Σ^* and L in order to determine whether $\Sigma^* \cap L = L$ is regular, contradicting Exercise 12.6.6(b).

- 12.6.9 (a) Suppose that we had an algorithm for deciding whether $L=L'$ for arbitrary context-free languages L and L' . Since Σ^* is context-free, we could use this algorithm to determine whether arbitrary context-free language L is identical with Σ^* , contrary to Exercise 12.6.6(a).
 (b) Use (a) and the Principle of Extensionality.

12.6.10 (a) We reduce PCP for an arbitrary sequence of word pairs to the problem of determining, of an arbitrary context-free grammar G , whether G is ambiguous. So let

$$\begin{array}{l} \langle x_1, y_1 \rangle \\ \langle x_2, y_2 \rangle \\ \dots \\ \langle x_n, y_n \rangle \end{array} \quad (12.6.7)$$

be an arbitrary sequence of word pairs over Σ . Next, let languages L_1 and L_2 be as in the proof of Theorem 12.10. Both were seen to be context-free. Suppose that $L_1=L(G_1)$ and that $L_2=L(G_2)$, where G_1 and G_2 are the context-free grammars of the proof of Theorem 12.10 except that we let their start symbols be S_1 and S_2 , respectively. We note at this point that neither G_1 nor G_2 is ambiguous. (Why not?) We easily construct a context-free grammar G^* with $L(G^*)=L_1 \cup L_2$. Namely, we simply add the two productions

$$S \rightarrow S_1 \mid S_2$$

to the combined productions of G_1 and G_2 . Since neither G_1 nor G_2 is ambiguous in and of itself, we see that language $L(G^*)$ is ambiguous if and only if there exists $w \in L_1 \cap L_2$, i.e., if and only if PCP for the given sequence of words pairs has a solution. (Why?) But the sequence of words pairs with which we began was completely arbitrary. In other words, any algorithm that would enable us to decide whether an arbitrary

context-free grammar is ambiguous could be used to decide whether PCP for an arbitrary sequence of word pairs has a solution. By Theorem 12.9, there can be no such algorithm.

(b) We first show that $L_5^c \neq \emptyset$ just in case $L_3 \cup L_4$ is inherently ambiguous. So assume an arbitrary sequence (12.6.7) of word pairs and suppose that $L_5^c = L_3 \cap L_4 \neq \emptyset$. It follows that an arbitrary context-free grammar G for context-free $L_3 \cup L_4$ must generate some word $w \in L_3 \cap L_4$. Reflection upon the definitions of L_3 and L_4 , however, reveals that w will have two parse trees relative to G —one by virtue of its membership in L_3 and another by virtue of its membership in L_4 . In other words, $L_3 \cup L_4$ is inherently ambiguous. Each step here being reversible, we may take ourselves to have shown that $L_5^c \neq \emptyset$ just in case $L_3 \cup L_4$ is inherently ambiguous. By the observed 1–1 correspondence between members of L_5^c and solutions to PCP for word-pair sequence (12.6.7), it follows from Theorem 12.9 that there can be no algorithm for deciding whether an arbitrary context-free language is inherently ambiguous.

Solutions to Exercises for § 12.7

- 12.7.1 (a) By Example 9.8.1 and Theorem 9.5, this language is type-2 but not type-3.
 (b) Example 9.8.1 makes it plain that both $\{a^n b^n \mid n \geq 0\}$ and $\{b^m a^m \mid m \geq 0\}$ are context-free languages. By closure under concatenation, so is $\{a^n b^n b^m a^m \mid n, m \geq 0\}$. The Pumping Lemma (Lemma 9.2) can be used to show that this language is not regular, however. Conclusion: language $\{a^n b^n b^m a^m \mid n, m \geq 0\}$ is type-2 but not type-3.
 (c) The Pumping Lemma (Lemma 10.3) can be used to show that this language is not type-2. It is easy to see that it is LBA-acceptable and hence type-1, however.
 (d) This language is finite and hence type-3, by Theorem 9.2.
 (e) By (d) and closure under complementation, this language is type-3.
 (f) We can see from Example 11.3.2 that this language is context-sensitive and hence type-1. This is an adequate characterization since Exercise 10.7.1(a) shows, in effect, that this language is not type-2.
 (g) This is a type-3 language denoted by regular expression $((ab)^{10})^*$.
 (h) By (g) and closure under complementation, this is another type-3 language.
 (i) This language is the intersection of the languages (a) and (h)—the first context-free, the second regular. By closure under regular intersection (Theorem 10.9), this language is type-2. The Pumping Lemma (Lemma 9.2) can be used to show that it is not type-3.
 (j) By Exercise 8.8.6, this language is Turing-acceptable but not Turing-recognizable and hence not context-sensitive. In other words, we have here a type-0 language that is not type-1.

12.7.3. **Proof.** Suppose that we possessed an algorithm that, applied to any phrase-structure grammar $G = \langle \Sigma, \Gamma, \sigma, \Pi \rangle$ and production $\pi \in \Pi$, would answer the question, “Is π ever used to generate a word over Σ^* ?” We show that this algorithm can be used to give a solution to the general word problem for phrase-structure grammars.

Suppose that $G = \langle \Sigma, \Gamma, S, \Pi \rangle$ is an arbitrary phrase-structure grammar and let w be any word over Σ^* . We show that we can decide whether G generates w . Let us use G to construct a new grammar $G_w = \langle \Sigma_w, \Gamma_w, S_w, \Pi_w \rangle$. Terminal alphabet Σ_w will just be Σ . As for nonterminal alphabet Γ_w , we first add to Γ the new start symbol S_w as well as endmarker $\$,$ say. In addition, for each symbol $s \in \Sigma = \Sigma_w$, alphabet Γ_w will contain copy \bar{s} . As for production set Π_w , we have all the productions that result from those of Π when each occurrence of any symbol $s \in \Sigma = \Sigma_w$ has been replaced with copy \bar{s} . At this point, the productions of Π_w contain no occurrences of symbols of $\Sigma = \Sigma_w$. So we next add to Π_w the two productions

$$S_w \rightarrow \$S\$ \quad \text{and} \quad \$\bar{w}\$ \rightarrow w$$

where \bar{w} is obtained from w by replacing any and all occurrences of symbols $s \in \Sigma$ with \bar{s} . This completes the construction of G_w . Finally, we note the following, recalling that w is a particular word over Σ .

- G_w generates word w if and only if G generates w .
- G_w generates word w if and only if production $\$\bar{w}\$ \rightarrow w$ is applied. It follows immediately that G generates w if and only if production $\$\bar{w}\$ \rightarrow w$ is used to generate, in G_w , some terminal string.

Consequently, if we can decide the latter question, then we can decide the former. Q.E.D.

- 12.7.4 (a) The answer is no. Suppose, for the sake of proving a contradiction, that every r.e. language were the intersection of two context-sensitive languages. By closure of the context-sensitive languages under intersection, it would follow that every r.e. language is context-sensitive and a fortiori that every recursive language is context-sensitive. By Theorem 12.6, however, the recursive languages are precisely the Turing-recognizable languages. So every Turing-recognizable language would be context-sensitive, contradicting Theorem 11.9.
- (b) Again, the answer is no. Suppose, for the sake of proving a contradiction, that every r.e. language were the intersection of two context-free languages. We have seen, however, that the word problem for context-free languages is solvable. But now, given our assumption, it should be clear that a general solution to the word problem for r.e. languages is now available, contradicting Theorem 12.8.