第四章

4.1

a. Yes
b. No
c. No
d. No
e. No
f. Yes

4.2 Let $EQ_{\mathsf{DFA,REX}} = \{\langle A, R\rangle \mid A \text{ is a DFA}, R \text{ is a regular expression and } L(A) = L(R)\}$. The following TM E decides $EQ_{\mathsf{DFA,REX}}$.

E = "On input $\langle A, R\rangle$:

1. Convert regular expression R to an equivalent DFA B using the procedure given in Theorem 1.28.
2. Use the TM C for deciding EQ_{DFA} given in Theorem 4.5, on input $\langle A, B\rangle$.
3. If R accepts, *accept*. If R rejects, *reject*."

4.3 Let $ALL_{\mathsf{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA that recognizes } \Sigma^*\}$. The following TM L decides ALL_{DFA}.

L = "On input $\langle A\rangle$ where A is a DFA:

1. Construct DFA B that recognizes $\overline{L(A)}$ as described in Exercise 1.10.
2. Run TM T from Theorem 4.4 on input $\langle B\rangle$, where T decides E_{DFA}.
3. If T accepts, *accept*. If T rejects, *reject*."

4.4 Let $A\varepsilon_{\mathsf{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG that generates } \varepsilon\}$. The following TM V decides $A\varepsilon_{\mathsf{CFG}}$.

V = "On input $\langle G\rangle$ where G is a CFG:

1. Run TM S from Theorem 4.6 on input $\langle G, \varepsilon\rangle$, where S is a decider for A_{CFG}.
2. If S accepts, *accept*. If S rejects, *reject*."

4.5 Let $INFINITE_{\mathsf{DFA}} = \{\langle A \rangle | \ L(A) \text{ is an infinite language}\}$. The following TM I decides $INFINITE_{\mathsf{DFA}}$.

I = "On input $\langle A \rangle$ where A is a DFA:

1. Let k be the number of states of A.
2. Construct a DFA D that accepts strings of length $\geq k$.
3. Construct a DFA M such that $L(M) = L(A) \cap L(D)$.
4. Run TM T from Theorem 4.4 on input $\langle M \rangle$, where T decides E_{DFA}.
5. If T accepts, *reject*. If T rejects, *accept*."

If A accepts a string with length at least k, this string can be must cause A to enter the same state at least twice, so it can be pumped up to obtain infinitely many strings that are accepted by A. Conversely, if A accepts infinitely many strings, it must accept strings of arbitrarily long lengths, in particular, a string of length at least k.

4.6

a. g is one-to-one. f is not one-to-one because $f(1) = f(3)$.

b. g is onto. f is not onto because there does not exist $x \in X$ such that $f(x) = 10$.

c. g is a correspondence because g is one-to-one and onto. f is not a correspondence because f is not one-to-one and onto.

4.7 Suppose $\mathcal{B}$ is countable and a correspondence $f : \mathcal{N} \longrightarrow \mathcal{B}$ exists. We construct x in $\mathcal{B}$ that is not paired with anything in $\mathcal{N}$. Let $x = x_1 x_2 \ldots$. Let $x_i = 0$ if $f(i)_i = 1$, and $x_i = 1$ if $f(i)_i = 0$ where $f(i)_i$ is the ith bit of $f(i)$. Therefore, we ensure that x is not $f(i)$ for any i because it differs from $f(i)$ in the ith symbol, and a contradiction occurs.

4.8 We demonstrate a one-to-one $f : T \longrightarrow \mathcal{N}$. Let $f(i, j, k) = 2^i 3^j 5^k$. Function f is one-to-one because if $a \neq b$, $f(a) \neq f(b)$. Therefore, T is countable.

4.9 Let $\sim$ be the binary relation "is the same size". In other words If $A \sim B$, A and B are the same size. We show that $\sim$ is an equivalence relation. First, $\sim$ is reflexive because the identity function $f(x) = x, \forall x \in A$ is a correspondence $f : A \longrightarrow A$. Second, $\sim$ is symmetric because any correspondence has an inverse, which itself is a correspondence. Third, $\sim$ is transitive because if $A \sim B$ via correspondence f, and $B \sim C$ via correspondence g, then $A \sim C$ via correspondence $f \circ g$ (the composition of f and g). Because $\sim$ is reflexive, symmetric, and transitive, $\sim$ is an equivalence relation.

4.10

The language we are concerned with is

$$INFINITE_{\text{DFA}} = \{ \langle A \rangle \ : \ A \text{ is a DFA and } L(A) \text{ is an infinite language} \} .$$

Want: To show that $INFINITE_{\text{DFA}}$ is decidable, meaning to construct a TM M that decides $INFINITE_{\text{DFA}}$. This means that for all DFAs A we want

- If $L(A)$ is an infinite language then $M(\langle A \rangle)$ accepts
- If $L(A)$ is a finite language then $M(\langle A \rangle)$ rejects.

Idea: Given $\langle A \rangle$ where A is a DFA, we want to be able to determine whether the number of strings that A accepts is finite or infinite. As per the Computability Crib Sheet, TM M_{dfa} decides the language

$$A_{\text{DFA}} = \{ \langle A, w \rangle \ : \ A \text{ is a DFA and } A \text{ accepts } w \} .$$

One way to approach the design of a TM M to decide $INFINITE_{\text{DFA}}$ is, on input $\langle A \rangle$, start a loop over all $w \in \Sigma^*$, with an iteration of the loop running $M_{\text{dfa}}(\langle A, w \rangle)$ to test whether or not $w \in L(A)$. The problem with this is that there are infinitely many w and our machine will not halt. Instead, we claim the following, and use it in the construction. We will then return to justify the claim.

Claim: Suppose A is a DFA with p states and alphabet Σ. Let

$$B_A = \{ w \in \Sigma^* \ : \ w \in L(A) \text{ and } p \leq |w| \leq 2p \} .$$

Then $L(A)$ is infinite if and only if $B_A \neq \emptyset$. ▌

This means that to test whether or not $L(A)$ is infinite, we need only test whether or not B_A is empty. The latter can be done using the machine M_{dfa} since B_A is a finite set. We now provide the construction.

Construction: M = On input $\langle A \rangle$, where A is a DFA

Let p be the number of states in DFA $\langle A \rangle$
Let Σ be the alphabet of DFA A
For each string $w \in \Sigma^*$ such that $p \leq |w| \leq 2p$ do:
    Run $M_{\text{dfa}}(\langle A, w \rangle)$
    If it accepts then accept
EndFor
Reject

Correctness: The code above accepts $\langle A \rangle$ if and only if the set B_A is non-empty, and is thus correct due to the Claim stated above. We will now justify the claim. Since the claim is an "if and only if" there are two parts to our proof of the claim.

In the first part of the proof, we assume that $B_A \neq \emptyset$. We want to show that $L(A)$ is infinite. Note that the number of states p in the DFA is the p of the Pumping Lemma (Theorem 1.37, page 78) applied to the regular language $L(A)$. Since B_A is non-empty, there is some string $s \in B_A$. By definition of B_A we have $s \in L(A)$ and $|s| \geq p$, so there exist x, y, z such that $s = xyz$ and the three conditions of the Pumping Lemma hold. The first condition says that $xy^iz \in L(A)$ for all $i \geq 0$. This means that $L(A)$ is infinite.

In the second part of the proof, we assume that $L(A)$ is infinite. We want to show that $B_A \neq \emptyset$. Let s be a shortest string in $L(A)$ such that $|s| \geq p$. (Such an s exists since $L(A)$ is infinite.) We If $|s| \leq 2p$ then $s \in B_A$ so we have shown $B_A \neq \emptyset$. Now, suppose towards a contradiction that $|s| > 2p$. Apply the Pumping Lemma to the regular language $L(A)$ to write $s = xyz$ such that the three conditions of the Pumping Lemma hold. Setting $i = 0$ in the first condition tells us that $xz \in L(A)$. The third condition implies $|y| \leq p$. So $|xz| = |s| - |y| > 2p - p = p$, meaning xz is a string in $L(A)$ with length strictly greater than p, but is shorter than s. This contradicts the assumption that s was the shortest such string.

4-11

Solution

- Build a Turing machine that will do the following.
  - Read $\langle M \rangle$ and create an equivalent context-free grammar G.
  - Convert G to Chomsky Normal Form. Call it G'.
  - Do a breadth-first search of the grammar rules of G' looking for recursion.
  - That is, does there exist a derivation $A \stackrel{+}{\Rightarrow} uAv$?
  - If so, then accept $\langle M \rangle$.
  - If not, then reject $\langle M \rangle$.

4.12

The following TM X decides A.

X = "On input $\langle M \rangle$ where M is a DFA:

1. Construct a DFA O that accepts any string containing an odd number of 1s.
2. Construct DFA B such that $L(B) = L(M) \cap L(O)$.
3. Run TM T from Theorem 4.4 on input $\langle B \rangle$, where T decides E_{DFA}.
4. If T accepts, *accept*. If T rejects, *reject*."

4.13

We observe that $L(R) \subseteq L(S)$ if and only if $\overline{L(S)} \cap L(R) = \emptyset$. The following TM X decides A.

X = "On input $\langle R, S \rangle$ where R and S are regular expressions:

1. Construct DFA E such that $L(E) = \overline{L(S)} \cap L(R)$.
2. Run TM T from Theorem 4.4 on input $\langle E \rangle$, where T decides E_{DFA}.
3. If T accepts, *accept*. If T rejects, *reject*."

4.14

We have shown in Problem 2.17 that if C is a context-free language and R is a regular language, then $C \cap R$ is context free. Therefore $\mathtt{1}^* \cap L(G)$ is context free. The following TM X decides A.

X = "On input $\langle G \rangle$ where G is a CFG:

1. Construct CFG H such that $L(H) = \mathtt{1}^* \cap L(G)$.
2. Run TM R from Theorem 4.7 on input $\langle H \rangle$, where R decides E_{CFG}.
3. If R accepts, *reject*. If T rejects, *accept*."

4.15 Let $A = \{v \mid v \text{ is of the form } ww^{\mathcal{R}} \text{ for } w \in \{0,1\}^*\}$. A is a CFL. Problem 2.17 showed that $B \cap A$ is context free if B is regular. The following TM X decides E.

X = "On input $\langle M \rangle$ where M is a DFA:

1. Let $C = L(M) \cap A$. Let G be the CFG of C.
2. Run TM R from Theorem 4.7 on input $\langle G \rangle$, where R decides E_{CFG}.
3. If R accepts, *reject*. If R rejects, *accept*."

X decides E because a DFA M accepts some string of the form $ww^{\mathcal{R}}$ if and only if $L(M) \cap A \neq \emptyset$.

4.16

The following TM X decides A.

X = "On input $\langle R \rangle$ where R is a regular expression:

1. Construct DFA E that accepts $\Sigma^*\mathtt{111}\Sigma^*$.
2. Construct DFA B such that $L(B) = L(R) \cap L(E)$.
3. Run TM T from Theorem 4.4 on input $\langle B \rangle$, where T decides E_{DFA}.
4. If T accepts, *reject*. If T rejects, *accept*."

4.17

For any DFAs A and B, $L(A) = L(B)$ if and only if A and B accept the same strings up to length mn, where m and n are the numbers of states of A and B, respectively. Obviously, if $L(A) = L(B)$, A and B will accept the same strings. If $L(a) \neq L(B)$, we need to show that the languages differ on some string s of length at most mn. Let t be a shortest string on which the languages differ. Let l be the length of t. If $l \leq mn$ we are done. If not, consider the sequence of states $q_0, q_1, \ldots, q_l$ that A enters on input t, and the sequence of states $r_0, r_1, \ldots, r_l$ that B enters on input t. Because A has m states and B has n states, only mn distinct pairs (q, r) exist where q is a state of A and r is a state of B. By the pigeon hole principle, two pairs of states (q_i, r_i) and (q_j, r_j) must be identical. If we remove the portion of t from i to $j - 1$ we obtain a shorter string on which A and B behave as they would on t. Hence we have found a shorter string on which the two languages differ, even though t was supposed to be shortest. Hence t must be no longer than mn.

4.18

We need to prove both directions. To handle the easier one first, assume that D exists. A TM recognizing C operates on input x by going through each possible string y and testing whether $\langle x, y \rangle \in D$. If such a y is ever found, *accept*; if not, just continue searching.

For the other direction, assume that C is recognized by TM M. Define a language B to be $\{\langle x, y \rangle | \ M \text{ accepts } x \text{ within } |y| \text{ steps}\}$. Language B is decidable, and if $x \in C$ then M accepts x within some number of steps, so $\langle x, y \rangle \in B$ for any sufficiently long y, but if $x \notin C$ then $\langle x, y \rangle \notin C$ for any y.

4-19

We will show a decidable language L and a homomorphism h such that $h(L)$ is undecidable

- Let $L = \{xy \mid x \in \{0,1\}^*, y \in \{a,b\}^*, x = \langle M, w \rangle$, and y encodes an integer n such that the TM M on input w will halt in n steps $\}$
- L is decidable: can simply simulate M on input w for n steps
- Consider homomorphism h: $h(0) = 0$, $h(1) = 1$, $h(a) = h(b) = \epsilon$.
- $h(L) =$ HALT which is undecidable. □

4.20

Let A and B be two languages such that $A \cap B = \emptyset$, and $\overline{A}$ and $\overline{B}$ are recognizable. Let J be the TM recognizing $\overline{A}$ and K be the TM recognizing $\overline{B}$. We will show that the language decided by TM T separates A and B.

T = "On input w:

1. Simulate J and K on w by alternating the steps of the two machines.
2. If J accepts first, *reject*. If K accepts first, *accept*."

The algorithm T terminates because $\overline{A} \cup \overline{B} = \Sigma^*$. So either J or K will accept w eventually. $A \subseteq C$ because if $w \in A$, w will not be recognized by J and will be accepted by K first. $B \subseteq \overline{C}$ because if $w \in B$, w will not be recognized by K and will be accepted by J first. Therefore, C separates A and B.

4.21

For any language A, let $A^{\mathcal{R}} = \{w^{\mathcal{R}} | \ w \in A\}$. If $\langle M \rangle \in S$, then $L(M) = L(M)^{\mathcal{R}}$. The following TM T decides S.

T = "On input $\langle M \rangle$, where M is a DFA:

1. Construct DFA N that recognizes $L(M)^{\mathcal{R}}$.
2. Run TM F from Theorem 4.5 on $\langle M, N \rangle$, where F is the Turing machine deciding EQ_{DFA}.
3. If F accepts, *accept*. If F rejects, *reject*."

4.23 The following procedure decides $AMBIG_{\text{NFA}}$. Given an NFA N, we design a DFA D that simulates N and accepts a string iff it is accepted by N along two different computational branches. Then we use a decider for E_{DFA} to determine whether D accepts any strings.

Our strategy for constructing D is similar to the NFA-to-DFA conversion in the proof of Theorem 1.39. We simulate N by keeping a pebble on each active state. We begin by putting a red pebble on the start state and on each state reachable from the start state along ε transitions. We move, add, and remove pebbles in accordance with N's transitions, preserving the color of the pebbles. Whenever two or more pebbles are moved to the same state, we replace its pebbles with a blue pebble. After reading the input, we accept if a blue pebble is on an accept state of N or if two different accept states of N have red pebbles on them.

The DFA D has a state corresponding to each possible position of pebbles. For each state of N, three possibilities occur: It can contain a red pebble, a blue pebble, or no pebble. Thus, if N has n states, D will have 3^n states. Its start state, accept states, and transition function are defined to carry out the simulation.

4-24

This problem requires you to recognize that if C is a context free language and R is a regular language, then C∩R is context free. You have to show this, but it has been done for you as the solution to problem 2.18.

"On input <G>:

1. Construct CFG H st L(H) = 1*∩L(G).
2. Test whether L(H) = ∅, using the E_{CFG} decider R from Theorem 4.8.
3. If R accepts, *reject*; if R rejects, *accept*."

Another solution：

Let $\mathcal{P}$ be the set of all pushdown automata. Let the language $\mathcal{U} = \{x \in \mathcal{P} | x$ has a useless state$\}$. To show that $\mathcal{U}$ is decidable we design a Turing machine which accepts only strings in $\mathcal{U}$. From the book we know that the question of whether a PDA has an empty language is decidable, and we may reduce the question of whether a given state q is useless to this question by making q the only accept state and then determining whether the resulting PDA has an empty language. If it does, then q is a useless state.

Our Turing machine may therefore solve the question of whether there are *any* useless states by performing this test for each state in order.

4.25 The language of all strings with an equal number of 0s and 1s is a context-free language, generated by the grammar $S \to 1S0S \mid 0S1S \mid \varepsilon$. Let P be the PDA that recognizes this language. Build a TM M for BAL_{DFA}, which operates as follows. On input $\langle B \rangle$, where B is a DFA, use B and P to construct a new PDA R that recognizes the intersection of the languages of B and P. Then test whether R's language is empty. If its language is empty, *reject*; otherwise, *accept*.

4．26

Ans. Let $PAL_{DFA} = \{\langle M \rangle \mid M$ is a DFA that accepts some palindrome$\}$. To show PAL_{DFA} is decidable, we construct a decider D for PAL_{DFA} as follows (Let K be a TM that decides E_{CFG}):

$D =$ "On input $\langle M \rangle$,

1. Construct a PDA P such that $L(P) = \{w \mid w$ is a palindrome$\}$
2. Construct a PDA P' such that $L(P') = L(P) \cap L(M)$
3. Convert P' into an equivalent CFG G
4. Use K to check if $L(G)$ is empty.
5. If $L(G)$ is empty, reject. Else, accept.

In the above TM, Step 1 can be done in finite steps. Step 2 is based on Prob 2.18 and can be done in finite steps. Step 3 is the conversion of PDA into an equivalent CFG, which can be done in finite steps (See Notes 8, pages 28–34). Step 4 is done in finite steps, because the decider K can check whether the language of a CFG is empty (For the existence of K, see Notes 12, pages 20–21). In summary, D runs in finite steps for any input, and is thus a decider.

4.27

Let $E = \{\langle M \rangle \mid M$ is a DFA that accepts some string with more 1s than 0s$\}$. Show that E is decidable. (Hint: Theorems about CFLs are helpful here.)

Answer: Let $A = \{x \mid x$ has more 1s than 0s$\}$. The language A is context-free, as we can easily construct a PDA to recognize A. Now, we construct the TM M below to decide E as follows:

$M =$ "On input $\langle M \rangle$ where M is a DFA:

1. Construct $B = A \cap L(M)$. Note that B is CFL, since $L(M)$ is regular and A is CFL.
2. Test whether B is empty.
3. If yes, **reject**. Otherwise, **accept**.

4-28

$C = \{< G, x > | G$ is a CFG that generates some string w, where x is a substring of w $\}$.
The following TM M decides C.
$M =$"On input $< G, x >$ where G is a CFG:

1. Let the regular expression $E = \Sigma^* x \Sigma^*$.
2. Construct a CFG H such that $L(H) = L(E) \cap L(G)$.
3. Run TM R from Theorem 4.8 on input $< H >$.
4. If R accepts, reject; otherwise, accept."

Note that L(H) is a context-free language because L(E) is regular and L(G) is context-free. If G generates some string w containing x as a substring, then $w \in L(H)$. Hence $L(H) \neq \phi$ and thus R rejects and M accepts. Conversely, if G never generates a string w containing x as a substring, no any $w \in L(G)$. Since $L(H) = \phi$, R will accept and M will reject. As a result, M decides C.

4.29

Ans. Let C be the language

$C_{CFG} = \{\langle G, k\rangle \mid G \text{ is a CFG and } L(G) \text{ contains exactly } k \text{ strings where } k \geq 0 \text{ or } k = \infty\}$

In this problem, we are given a decider D that decides if the language of a CFG is infinite. Then, we can show that C is decidable, by finding a corresponding decider F as follows:

F = "On input $\langle G, k\rangle$,

1. Use D to check if $L(G)$ is an infinite set.
2. There are four cases:
   i. If yes, and $k = \infty$, accept.
   ii. If yes, but $k \neq \infty$, reject.
   iii. If no, but $k = \infty$, reject.
   iv. If no, and $k \neq \infty$, continue.
3. Compute the pumping length p for the grammar G.
4. Set *count* to be 0.
5. For $x = 1, 2, \ldots, p$
   i. For all string s with length $= x$,
      Check if s can be generated by G; if so, increment *count* by 1.
6. If *count* $= k$, accept. Else, reject.

In the above TM F, Steps 1–2 correctly answer the case where $L(G)$ is an infinite set, or $k = \infty$. So, after Step 2, we only deal with a grammar G whose language is a finite set, and our task is to check whether the language size is exactly k. To do so, the loop in Step 5 counts all string that can be generated by G, whose length is at most the pumping length p. Because we know that $L(G)$ is finite, we are sure that no strings of $L(G)$ can be longer than p. In other words, the value *count* correctly computes the exact number of strings in $L(G)$. So, Step 6 can check correctly answer the case when $L(G)$ is finite, and k is finite.

Finally, it is easy to check that each step runs in finite number of steps. Thus, F is a decider, so that C is a decidable language.

4.30

Let A be a Turing-recognizable language consisting of descriptions of Turing machines, $\{\langle M_1\rangle, \langle M_2\rangle, \ldots\}$, where every M_i is a decider. Prove that some decidable language D is not decided by any decider M_i whose description appears in A.† (Hint: You may find it helpful to consider an enumerator for A, and re-visit the diagonalization technique.)

Answer: Since A is Turing-recognizable, there exists an enumerator E that enumerates it. In particular, we let $\langle M_i\rangle$ be the i^{th} output of E (note: $\langle M_i\rangle$ may not be distinct).

Let $s_1, s_2, s_3 \ldots$ be the list of all possible strings in $\{0,1\}^*$. Now, we define a TM D as follows:

$D =$ "On input w:

1. If $w \notin \{0,1\}^*$, **reject**.
2. Else, w is equal to s_i for a specific i.
3. Use E to enumerate $\langle M_1\rangle, \langle M_2\rangle, \ldots$ until $\langle M_i\rangle$.
4. Run M_i on input w.
5. If M_i accepts, **reject**. Otherwise, **accept**."

4.31:

Say that a variable A in a CFG G is *usable* if it appears in some derivation of some string $w \in L(G)$. Formulate this problem as a language and show that it is decidable.

The language is:

$$\text{USABLE}_{CFG} = \{\langle G, A\rangle : G \text{ is a CFG}, A \text{ usable for } G\}.$$

To show that this is decidable, I make the following:

Claim: Variable A is usable for CFG G if and only if there exists a derivation $S \Rightarrow^* xAy$ of G such that $L(x)$, $L(A)$, and $L(y)$ are all nonempty.

(By $L(x)$ for a sentential form x I mean the language: $\{w \in \Sigma^* : x \Rightarrow^* w\}$.)

To prove the claim, suppose first of all that A is usable for G. Then there is a derivation $S \Rightarrow^* w$ in which A appears. For A to appear in a derivation means that it must have the form $S \Rightarrow^* xAy \Rightarrow^* w$. In this case, we must have $w = utv$, where $x \Rightarrow^* u$, $A \Rightarrow^* t$, and $v \Rightarrow^* v$. But this implies $u \in L(x)$, $t \in L(A)$, and $v \in L(y)$, so all three are nonempty.

Conversely, suppose that there is a derivation $S \Rightarrow^* xAy$ such that $L(x)$, $L(A)$, and $L(y)$ are all nonempty. Then there exist some $u \in L(x)$, $t \in L(A)$, and $v \in L(y)$, hence there are derivations $x \Rightarrow^* u$, $A \Rightarrow^* t$, and $y \Rightarrow^* v$. But then there is clearly also a derivation $S \Rightarrow^* xAy \Rightarrow^* utv$, which shows that A is usable for G. ∎

We now make use of the above claim to show that USABLE_{CFG} is decidable. A decider for USABLE_{CFG} works as follows:

- On input $\langle G, A \rangle$:
  - Construct the directed graph which has as its nodes the nonterminals of G and which has an edge from B to C exactly when G has some production of the form $B \to xCy$. This graph has the property that there is a path from S to A if and only if there is some derivation of G of the form $S \Rightarrow^* xAy$.
  - Determine if there is a path in the graph from S to A. This can be done by your favorite version of depth-first or breadth-first search. If there is no path from S to A, then G has no derivation of the form $S \Rightarrow^* xAy$, so *reject*.
  - If there is a path from S to A, then by choosing a production of G corresponding to each edge in the path, we obtain a derivation $S \Rightarrow^* xAy$.
  - Use a decider for the language E_{CFG} to check whether $L(x)$, $L(A)$, and $L(y)$ are all nonempty. (To check whether $L(A) \neq \emptyset$ just apply the decider to a modification of G obtained by changing the start symbol to A. To check whether $L(x)$ or $L(y)$ is nonempty it is necessary to check *all* nonterminals in x and y.)
  - If all three of $L(x)$, $L(A)$, and $L(y)$ are nonempty, then *accept*, otherwise *reject*.

Note that it is not enough for the decider to just check the reachability of A; *i.e.* whether there is a derivation $S \Rightarrow^* xAy$, because if any of $L(x)$, $L(A)$, or $L(y)$ is empty, then this derivation cannot be extended to the derivation of a sentence $w \in \Sigma^*$, hence it will not establish the usability of A.