

Problem Set 3 Solutions

Problem 1 Exercise 3.5, page 148 of the text.

The key to the solution is to closely examine the formal definition of a Turing machine as given in Definition 3.1 page 128 of the text as well as the ensuing discussion until the end of page 129. The point of the exercise is to learn how the material there contains all you need to answer the following.

a. Can a Turing machine ever write the blank symbol on its tape?

Yes. By looking at the definition of the transition function, you can see that the machine can write any symbol from the tape alphabet to the tape. The blank symbol is mandated to be part of the tape alphabet (although it is not part of the input alphabet).

b. Can the tape alphabet Γ be the same as the input alphabet Σ ?

No. The tape alphabet always contains the blank symbol, but the input alphabet cannot contain this symbol. If the blank symbol were part of the input alphabet, the Turing machine would never know when the input actually ends!

c. Can a Turing machine's read head *ever* be in the same location in two successive steps?

Yes. But the only time this can happen is when the Turing machine is on the first tape square and it tries to move left. It will stay in place instead of falling off the tape. However, if it is not on the leftmost square, then in the next move the tape head cannot remain in the same location.

d. Can a Turing machine contain just a single state?

No. A Turing machine must contain at least two states: an accept state and a reject state. Because being in either of these states halts the computation, a different start state would be necessary in order for the TM to read any input whatsoever.

Problem 2 Exercise 3.7, page 148 of the text.

M_{bad} = The input is a polynomial p over variables x_1, \dots, x_k .

Try all possible settings of x_1, \dots, x_k .

Evaluate p on all these settings.

If any of these setting evaluates to 0, *accept*; else *reject*.

This description has several problems which result in its not being a proper piece of code or pseudocode. The loop structure is too vague to really understand what it does. Does the machine

first write all possible values of the variable to tape and then evaluate each one of them? Or does it produce the values of the variables and then evaluate those assignments before considering the next assignment? In either case, the machine may get in an infinite loop. The most serious problem is in the last step. There is no condition when the machine can reject. The machine cannot try all possible assignments of the variables, because there are an infinite number of them.

Note: If you are careful about how the loop is structured, you can make a TM that *recognizes* the language this machine attempts to decide.

Problem 3 Problems 3.14(c) and 3.15(c), page 149 of the text.

3.14 (c) The class of decidable languages is closed under the Kleene star operation.

We need to show that if A is any decidable language, then A^* is also decidable. We use the usual template.

Given: Assume A is decidable. This means that there is a TM M which decides A . This in turn means that for all strings $w \in \Sigma^*$

- If $w \in A$ then $M(w)$ accepts.
- If $w \notin A$ then $M(w)$ rejects.

Want: To show that A^* is decidable. To do this we need a TM M^* that decides A^* , meaning for all $W \in \Sigma^*$

- If $W \in A^*$ then $M^*(w)$ accepts.
- If $W \notin A^*$ then $M^*(W)$ rejects.

Construction: $M^* =$ On input a string W

If $W = \varepsilon$ then *accept*

Let n be the length of W

For $k = 1$ to n do

For each possible string-sequence $X_1, \dots, X_k \in \Sigma^* - \{\varepsilon\}$ satisfying $X = X_1X_2 \dots X_k$ do

Run $M(X_j)$ for $j = 1$ to k .

If all of these computations accept then *accept*.

End For

End For

Reject.

Correctness: By definition of the Kleene Star operation, $W \in A^*$ if and only if $W = \varepsilon$ or $W = X_1 \dots X_k$ for some k and some non-empty strings $X_1, \dots, X_k \in A$. The machine M^* above tries all possible choices of X_1, \dots, X_k . It is correct, meaning always terminates with the right answer, because there are only a finite number of ways to divide a string of finite length into non-empty substrings.

3.15 (c) The class of recognizable languages is closed under the Kleene star operation.

We need to show that if A is any recognizable language, then A^* is also recognizable. We use the usual template.

Given: Assume A is recognizable. This means that there is a TM M which recognizes A . This in turn means that for all strings $w \in \Sigma^*$

- If $w \in A$ then $M(w)$ accepts.
- If $w \notin A$ then $M(w)$ either rejects or loops.

Want: To show that A^* is recognizable. To do this we need a TM M^* that recognizes A^* , meaning for all $W \in \Sigma^*$

- If $W \in A^*$ then $M^*(w)$ accepts.
- If $W \notin A^*$ then $M^*(W)$ either rejects or loops.

Construction: You may think that the construction provided in the above solution continues to work. This is not true. The problem is that when we run M , we no longer have a guarantee that it will ever halt. This will potentially cause our machine to enter an infinite loop before it checks all the substrings that it needs to check. We can solve this by putting another loop around the entire algorithm that will break M 's computation if it runs too long. The construction follows.

M^* = On input a string W

```

If  $W = \varepsilon$  then accept
Let  $n$  be the length of  $W$ 
count  $\leftarrow 1$ ;  $h \leftarrow 0$ 
While ( $h = 0$ ) do
  For  $k = 1$  to  $n$  do
    For each possible string-sequence  $X_1, \dots, X_k \in \Sigma^* - \{\varepsilon\}$  satisfying  $X = X_1X_2 \dots X_k$  do
       $h \leftarrow 1$ 
      For  $j = 1, \dots, k$  do
        Run  $M(X_j)$  for count steps
        If it has not halted at this point, or has rejected, let  $h \leftarrow 0$ 
      EndFor
      If  $h = 1$  then accept.
    End For
  End For
  count  $\leftarrow$  count + 1
EndWhile

```

Correctness: We know that $W \in A^*$ if and only if $W = \varepsilon$ or $W = X_1 \dots X_k$ for some k and some non-empty strings $X_1, \dots, X_k \in A$. The machine M^* above tries all possible choices of X_1, \dots, X_k . For each candidate choice, it runs M on each X_i for count steps to see whether or not M accepts, and then, in the next iteration of the while loop, increments count. If $W \in A^*$ then eventually count will be large enough that, for a choice of k and X_1, \dots, X_k satisfying $W = X_1 \dots X_k$, machine $M(X_i)$ will halt within count steps for all i . At that point, M^* will halt and accept. If $W \notin A^*$ then regardless of the value of count, for all k and all X_1, \dots, X_k , there will be some i such that $M(X_i)$ either rejects or does not halt, and hence will not accept in count steps. So M^* will loop.

(It turns out M^* never rejects. It accepts if $W \in A^*$ and loops otherwise. But that's ok.)

Problem 4 Exercise 4.4, page 169 of the text.

We are considering the language

$$A_{\varepsilon_{\text{CFG}}} = \{ \langle G \rangle : G \text{ is a CFG that generates } \varepsilon \}$$

Want: To show that $A_{\varepsilon_{\text{CFG}}}$ is decidable, meaning to construct a TM M that decides $A_{\varepsilon_{\text{CFG}}}$. This means that for all CFGs G we want

- If G generates ε then $M(\langle G \rangle)$ accepts
- If G does not generate ε then $M(\langle G \rangle)$ rejects.

Construction: This is a simple example of how you need to remember and exploit facts already proved or stated about decidability in proving the decidability of new languages. In particular, using a TM from the Computability Crib Sheet as a subroutine, our TM is: $M =$ On input $\langle G \rangle$

If $M_{\text{cfg}}(\langle G, \varepsilon \rangle)$ accepts then accept
Else reject

Correctness: From the definitions of the languages we can see that $\langle G \rangle \in A_{\varepsilon_{\text{CFG}}}$ if and only if $\langle G, \varepsilon \rangle \in A_{\text{CFG}}$. So the above is correct.

Problem 5 Exercise 4.5, page 169 of the text.

The language we are concerned with is

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

Want: To show that $\text{INFINITE}_{\text{DFA}}$ is decidable, meaning to construct a TM M that decides $\text{INFINITE}_{\text{DFA}}$. 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 $\text{INFINITE}_{\text{DFA}}$ 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  $\Sigma$  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.

Problem 6 Problem 4.10, page 169 of the text.

We are considering the language

$$A = \{ \langle M \rangle : M \text{ is a DFA which doesn't accept any string containing an odd number of 1s} \}.$$

Want: To show that A is decidable, meaning to construct a TM M_A that decides A . This means that for all DFAs M we want

- If M doesn't accept any string containing an odd number of 1s then $M_A(\langle M \rangle)$ accepts.
- If M accepts some string containing an odd number of 1s then $M_A(\langle M \rangle)$ rejects.

Construction: Note that the language

$$\text{Odd} = \{ w \in \{0, 1\}^* : w \text{ contains an odd number of 1s} \}$$

is regular. This means we can let D_{Odd} be a DFA that recognizes Odd. Using various facts from the Computability Crib Sheet we define TM M_A as follows: $M_A = \text{On input } \langle M \rangle$ where M is a DFA

Let $\langle N \rangle \leftarrow T_{\text{dfa}}^{\cap}(\langle M, D_{\text{Odd}} \rangle)$
 Run $M_{\text{dfa}}^{\emptyset}(\langle A \rangle)$
 If it accepts then accept else reject

Correctness: We note that $\langle M \rangle \in A$ iff $L(M) \cap \text{Odd} = \emptyset$. Our TM M_A above accepts input $\langle M \rangle$ iff $L(M) \cap \text{Odd} = \emptyset$.

Problem 7 Exercise 5.2, page 195 of the text.

Want: To show that

$$EQ_{\text{CFG}} = \{ \langle G_1, G_2 \rangle : G_1, G_2 \text{ are CFGs and } L(G_1) = L(G_2) \}$$

is co-recognizable. In other words, we want to show that the complement of the above language, namely the language

$$A = \{ \langle G_1, G_2 \rangle : G_1, G_2 \text{ are CFGs and } L(G_1) \neq L(G_2) \}$$

is recognizable. Meaning we want to construct a TM M such that for all CFGs G_1, G_2

- If $L(G_1) \neq L(G_2)$ then $M(\langle G_1, G_2 \rangle)$ accepts
- If $L(G_1) = L(G_2)$ then $M(\langle G_1, G_2 \rangle)$ rejects or loops.

Construction: Using various TMs from the Computability Crib Sheet, our TM M is as follows: $M = \text{On input } \langle G_1, G_2 \rangle$ where G_1, G_2 are CFGs

For all w in Σ^* do
 If $M_{\text{cfg}}(\langle G_1, w \rangle)$ accepts then let $d_1 = 1$ else let $d_1 = 0$
 If $M_{\text{cfg}}(\langle G_2, w \rangle)$ accepts then let $d_2 = 1$ else let $d_2 = 0$
 If $d_1 \neq d_2$ then accept

The above machine tests all strings $w \in \Sigma^*$ one by one. In an iteration, it tests whether w is generated by one grammar but not the other. If so it accepts. If not, it does not reject; rather it continues with the loop.

Correctness: $L(G_1) \neq L(G_2)$ iff there is some w such that one of the following holds: (1) Either $w \in L(G_1)$ and $w \notin L(G_2)$ OR (2) $w \notin L(G_1)$ and $w \in L(G_2)$. Our machine M searches for a w satisfying this property. If it finds one, it halts and accepts. If not, it does not halt.

