Handout 8b: Turing Machines Review: Solutions to Practice Problems

Andrew Jin and Anastasija Tortevska

COMS 3261 Fall 2022

1. Consider the input-output TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{halt})$ where $Q = \{q_0, q_1, q_{halt}\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \sqcup\}$, and δ is given by:
$\delta(q_0, 0) = (q_0, 0, R)$, $\delta(q_0, 1) = (q_0, 1, R)$, $\delta(q_0, \sqcup) = (q_1, \sqcup, L)$
$\delta(q_1, 0) = (q_{halt}, 1, R)$, $\delta(q_1, 1) = (q_1, 0, L)$, $\delta(q_1, \sqcup) = (q_{halt}, \sqcup, L)$

   (a) Provide the complete sequence of configurations of M when ran on input 100.
   What is the output of M on this input?

   Solution:
   $q_0 100 \Rightarrow 1q_0 00 \Rightarrow 10q_0 0 \Rightarrow 100q_0 \Rightarrow 10q_1 0 \Rightarrow 101q_{halt}$
   The output of M on 100 is 101.

   (b) What is the output of M on 10011? on input 11?

   Solution:
   The output of M on 10011 is 10100.
   The output of M on 11 is 10.

   (c) What function is computed by M?

   Solution:
   The function computed by M is the following:
   On input 1^n: output 10^{n-1}
   On input ε: output ε
   On input $x \neq 1^n, x \neq \varepsilon$: treat x as a binary number and output $x+1$.

2. Let $\Sigma = \{\#, 0, 1\}$. Provide an *implementation level* description of a input-output TM that computes the function

$$f(\#\langle x\rangle) = \begin{cases} \#\langle x/2\rangle & \text{if } x \text{ is even} \\ \#\langle 3x+1\rangle & \text{otherwise} \end{cases}$$

where $\langle x\rangle$ stands for the binary representation of the number x.

(For example, if the TM starts with #100 on the tape it should halt with #10 on the tape; if it starts with #11, it should halt with #1010.)

You may use a TM with more than one tape – in this case the output should be written on the first tape.

Solution:

We will use a 3-tape TM. The intuition is to first check whether the input x is even (the last digit is 0) or odd (the last digit is 1). If the former, we replace the terminating 0 with a blank (dividing by 2 in binary) and halt. If the latter, we duplicate the input on a second tape and add a terminating 0 to the original (multiplying by 2 in binary). We now have $2x$ on the first tape and x on the second tape. We then add a leading zero to the first tape and two leading zeroes to the second so that we can sum the newly aligned tapes to get $3x$, to which we finally add 1. The third tape is used to keep track of carry bits in the addition step.

More specifically, construct machine M as follows: On input $\#\langle x\rangle$, where $\langle x\rangle$ is a binary string:

(a) Make sure the first character is a #, then scan the remaining binary number on the input tape from left to right until the last digit is reached. If it is a 0 (i.e., the number is even), replace the 0 with a blank and halt. (If it is a 1, go on.)

(b) Write "#00" to the second tape, then copy the contents of the first tape onto the second tape after the '#00".

(c) Shift the contents of the first tape after the # over to the right by 1. Make the new first symbol after the # on the first tape a "0".

(d) Replace the first blank on the first tape with a 0.

(e) Write a # to the third tape for each symbol on the first tape, then replace the rightmost # with a 0.

(f) Add the contents of the second tape to the first tape by repeating the following two steps until all the digits of the second tape are crossed off:

   i. Starting with the rightmost digit on the second tape that has not been crossed off, add it and the digits from the other two tapes in the corresponding place (i.e. the same number of spaces to the left of the first blank on their tape), write the result to that place on the first tape, and write any carry to the third tape:

A. If all of the three digits are 0, write 0 to that place on the first tape and replace the rightmost # on the third tape with a 0.

B. If only one of them is 1, write 1 to that place on the first tape and replace the rightmost # on the third tape with a 0.

C. If only two of them are 1, write 0 to that place on the first tape and replace the rightmost # on the third tape with a 1.

D. If all three of them are 1, write 1 to that place on the first tape and replace the rightmost # on the third tape with a 1.

ii. Cross off the rightmost digit on the second tape that has not been crossed off.

(g) Add 1 to the number on the first tape by starting with the rightmost digit and checking if it is 0. If it is 0, change it to 1 and go to the final step. If it is 1, change it to 0, proceed to the next rightmost digit and repeat this step.

(h) If the first symbol after the # on the first tape is a 0, delete it and shift the remainder of the tape one to the left. Halt.

3. Show that $E_{CFG} = \{\langle G\rangle | G \text{ is a CFG and } L(G) = \phi\}$ is decidable.

   Solution:

   Observe that $L(G)$ is the empty set (generates no strings) if it's impossible to generate a terminal from the start state S, no matter how you follow the rules. For example, the set of rules

   - $S \to A$
   - $A \to A$
   - $B \to b$

   generated no strings because you will be stuck following

   $$S \to A \to A \to A \to ...$$

   Even though there's a rule $B \to b$ that can get you a terminal, you can never reach B from S.

   To show that this problem is decidable, we note that there must be a finite number of rules. Thus we can look through each of these rules and see if we can reach a string of terminals by following them. We start by marking variables that can generate a string of terminals. We then repeat this process again and again, checking all the variables that are within i steps of being able to generate a string of terminals. When no new variable is added, we check if the start variable has been marked. If so, the language has at least one string in it, so we reject. In more detail, consider the following algorithm.

   M on input $< G >$ where $G = (V, \Sigma, R, S)$ is a CFG:

   (a) Set V_1 = set of all variables A in V for which there is a rule $A \to w$ where $w \in \Sigma^*$. Mark all variables in V_1

   (b) Set $i = 1$

   (c) Repeat until no new variable is marked:
      - Set V_{i+1} = set of all variables A in V which are not already in V_i, and for which there's a rule $A \to w$ where $w \in (\Sigma \cup V_i)^*$.
      - Mark all variables in V_{i+1}
      - $i \leftarrow i + 1$

   (d) If S is marked, reject. Else, accept.

   The above is a decider, since for any valid $\langle G\rangle$ the set of variables in the CFG G is finite, and so the loop can be exeuted only finitely many times. Each iteration of the loop requires going through a finite set of the rules of G. If the input was not a valid encoding of a CFG, it can be tested and rejected immediately. In any case, every step of the above algorithm terminates in finite time, so this is a decider.

Observe that a variable is in the set V_i is i steps away from being able to generate a string of terminals.

$x \notin E_{CFG} \implies$ either (a) $x \neq \langle G \rangle$ (not a valid encoding), or (b) $x = \langle G \rangle$ for a CFG $G = (V, \Sigma, R, S)$ and there exists some derivation $S \rightarrow^* w$ where $w \in \Sigma^*$. In case (a), x is rejected immediately. In case (b), since the derivation is of finite numbre of steps m, it means that if S was not marked before, it will be marked when the loop is executed for the nth time. Hence, the algorithm rejects in step (d). In any case, x is rejected.

$x \in E_{CFG} \implies x = \langle G \rangle$ for a CFG $G = (V, \Sigma, R, S)$, but there is no derivation starting from S that yields a string of terminals. Hence, when the loops finishes executing, S will not be marked, and the algorithm accepts in step (d).

4. Let $L = \{\langle M, k\rangle | M$ is a TM, k is a positive integer, and there exists an input to M that makes M run for at least k steps$\}$
   Prove that that L is decidable.

   First attempt
   We can construct a TM that decides L. It will do the following:

   (a) On input $\langle M, k\rangle$ where M is a TM:
   (b) For all strings $w_i, i \in \mathbb{N}$
       - run M on w_i for k steps. If M doesn't terminate on w_i within k steps, accept.
   (c) If we're finished enumerating and M terminated within k steps every time, reject

   Problem: When do we reject? We need to enumerate through an infinite number of strings! Therefore, we will never arrive at step (c). This only recognizes the language, not decides it
   Observation: Each "step" of a TM's computation means it executes a single transition function. Each time a transition function is executed, the head of the TM can only move at most a single space in either direction. Even if the TM head moves right on every single transition, then in k steps, the head could only ever look at the first $k + 1$ characters on the tape. Therefore, to determine whether the machine ever runs for more than k steps or whether it halts within k steps on every input, anything on the tape after the first $k + 1$ symbols is not relevant. Thus, we don't need to enumerate over any strings that are longer than $k + 1$ characters long.

   Second attempt
   We can construct a TM that decides L. It will do the following:

   (a) On input $\langle M, k\rangle$ where M is a TM:
   (b) For all strings w_i where $len(w_i) \leq k + 1$:
       - run M on w_i for k steps. If M doesn't terminate on w_i within k steps, accept.
   (c) If we're finished enumerating and M terminated within k steps every time, reject

   Proof that this is a decider:
   $x \in L \Rightarrow M$ runs for at least k steps on some string, therefore there is a string w_i such that $len(w_i) \leq k + 1$ where the computation of M on w_i will not have terminated after k steps. Therefore, the program will accept on some iteration.
   $x \notin L \Rightarrow M$ terminates within k steps on every input string. Since we are testing a finite number of strings and M terminates on all of them, we will reach step (c) and then reject.