CSCI3390-Assignment 2 Solutions

1 TMs for Deciding Languages

Write the specification of a Turing machine recognizing one of the following three languages. Do *one* of these problems. Options (b) and (c) are meant to be challenging and will get you extra-credit points. Part (a) can be approached very similarly to the way we handled the design of Turing machines that reversed the input, or determined if the number of as and bs was the same.

(a) $\{w\#w : w \in \{a,b\}^*\}$. That is, the input alphabet is $\{a,b,\#\}$, and the machine will accept a string if and only if it contains exactly one $\#$ and the portion before the $\#$ matches the portion after. Thus the machine should accept $ab\#ab$ but reject both $ab\#a$ and $ab\#\#ab\#$.

(b) $\{ww : w \in \{a,b\}^*\}$. So the machine determines if the first half of the string matches the second half. This is harder, because you don't have a convenient marker to tell you where the middle is.

(c) If $v \in \{0,1\}^*$, then we denote by $(v)_2$ the integer represented by v in binary. So for example, $(110)_2$ and $(0110)_2$ are both equal to the integer 6. The language is

$$\{u\#v : (u)_2 < (v)_2\}.$$

In other words, the machine takes as input two integers encoded in binary and determines is the first is less than the second. Thus it should accept $0110\#1000$ but reject both $111\#0110$ and $\#1\#0$.

Solution. I wimped out and only did the first one (I look forward to seeing the solutions of those who rose to the challenge of (b) or (c)). The strategy is pretty simple, and should be reminiscent of our reversal machine: Start in the left half,

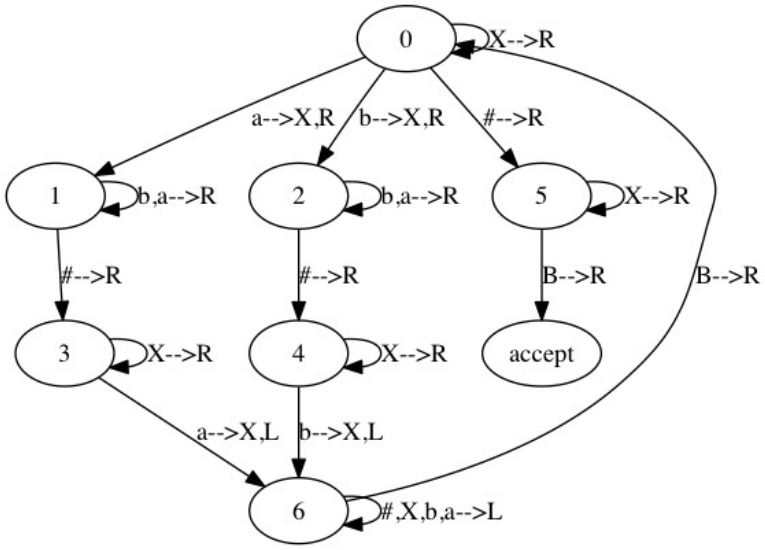

Figure 1: *Turing machine deciding the language in Problem 1(a)*

cross off the first symbol that is either a or b, and remember it in the state. Move right past the # , then continue right to the first a or b. If it matches the remembered character, cross it off, go back to the start, and repeat. If it doesn't match, reject.

If no a or b is found in the right half when the machine is searching for it, reject. If no a or b is found in the left half, and and a or b is found in the right half, reject. The remaining alternative is a scan that finds no occurrence of a or b in the left or right half, in which case the machine accepts.

The specification file is posted on the website. The state-transition diagram is pictured in Figure 1.

2 The move-over machine

The first step in simulating a two-tape machine by a one-tape machine is spreading out the original input string so that every second cell contains a blank. Here you are asked to show how this is done. Part (a) is a warmup, and if you solve both parts, you only need to submit your answer for (b).

(a) Design a Turing machine that puts a blank between its first input symbol and

second input symbol. That is, if the machine is started with

$$abba$$

on its tape, it will halt with

$$a\Box bba$$

on the tape.

(b) Design a Turing machine that puts a blank between every pair of consecutive input symbols. So if it starts with

$$abba$$

on its tape, it will halt with

$$a\Box b\Box b\Box a$$

on the tape.

Solution. For (a), move right to the second input symbol. Halt if it's blank; if not, write a blank, remembering the erased input symbol in the state, and move right. Then repeatedly write the remembered symbol, and remember the overwritten symbol in the state, until you overwrite a blank. It's ok to halt here, but to get ready for the next part, go left to the blank, move right and halt. For (b), instead of halting in the last step, return to the initial state. We only have to change one symbol in the specification file from (a) to do this!

The specifications are posted. The state diagrams are shown below.

3 One-way infinite tapes

Many texts give a different definition of the basic Turing machine: Instead of having the tape extend infinitely in both directions, it is only infinite toward the right. Thus there is a leftmost cell. We need to specify what happens if a transition tells the reading head to move left if it is positioned on the leftmost cell. Crash the program? Stay where it is? We will adopt the second of these rules: the machine will continue to run, but the reading head will not change its position.

The simulator program includes the option to run in 'one-way' mode. The way you do this (with, say, the `reverse.tm` specification) is by typing

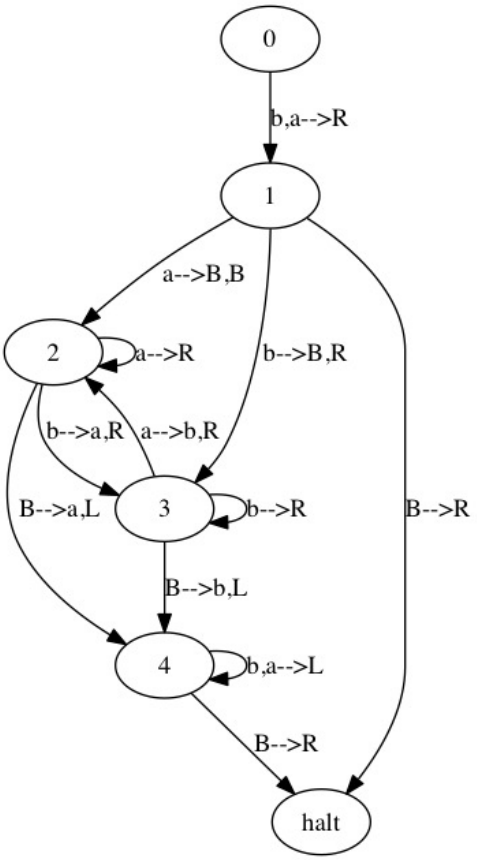

Figure 2: *Turing machine deciding the language in Problem 2(a)*

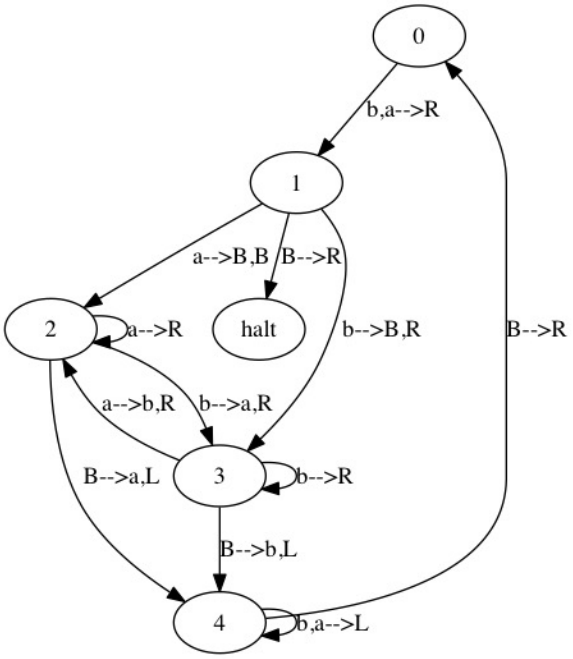

Figure 3: *Turing machine deciding the language in Problem 2(b). Observe that it is identical to the preceding machine, apart from the loop back to the initial state.*

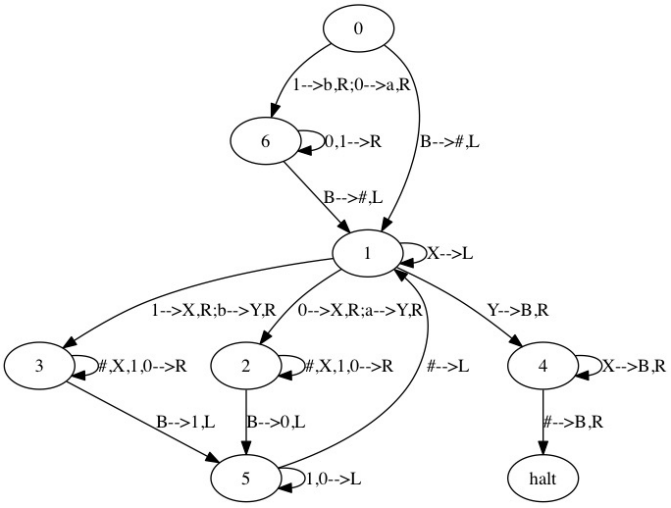

Figure 4: *Turing machine for reversal with a one-way infinite tape.*

```
runtm('reverse','110',bidirectional=False)
```

Try this out and make sure you understand the behavior you observe. Then modify the specification file `reverse.tm` (give it a new name) so that it reverses its input when run with these new rules. (HINT: You will have to somehow mark the leftmost cell of the tape.)

Solution. The two-way version of this problem halts when, while it searches for letters a, b to copy, encounters the blank cell to the left of the start of the original input. We cannot do this with the one-way tape, so instead we use special symbols for the leftmost cell of the tape, writing a in place of a, b in place of 0, and Y in place of X when writing to this cell. Thus the machine stops when it reaches the initial Y during a leftward scan. The state-transition diagram is shown in Figure 4, and the specification file is posted on the website.

An alternative solution is to shift the entire input rightward one cell, using the strategy from Problem 2(a), and then use the original version of the reversal machine.

4 *More on one-way infinite tapes

Prove that every Turing-recognizable language is recognized by a TM with a one-way infinite tape. You will have to describe a simulation, much in the 'higher-level' spirit of our proof that a two-tape machine can be simulated by a one-tape machine. (However, the simulation is easier, and the time penalty is much less.)

Solution. The problem is to show how to simulate a normal TM $\mathcal{M}$ by a one-way TM $\mathcal{M}'$. Students came up with several different strategies for this; I'll discuss one of these in detail and just mention some others briefly.

- This resembles the simulation method we used to simulate two-tape machines by 1-tape machines. We first alter the initial configuration of the tape with the move-over method (see Problem 2). As the simulation progresses, the symbols to the left of the initial head position are recorded in reversed order on alternate cells of the tape. Note that that leftmost cell of the 'right half', and the rightmost cell of the 'left half' are marked.

  We have to alter the states somewhat—essentially, each state of the original machine splits into two copies, one for when it is scanning a cell on the left half of the tape, one for the right half. (This can also be accomplished with additional markings on the tape symbols.) A transition of the form

  $$\delta(p, \beta) = (q, \gamma, R)$$

  of the original machine gives rise to several transitions of the new machine: If we are dealing with the 'right' copy of the state p, the machine will write γ, transition to the 'right' copy of q, and move two cells to the right. For the 'left' copy of p, the rightward transition induces motion two cells to the left. Something special has to happen when the reading head in the original machine moves from the left half to the right half, or vice-versa. In this case, a rightward transition of the original machine

  $$\delta(p, \beta) = (q, \gamma, R)$$

  also gives rise to a transition using the marked symbols

  $$\delta(p_{\text{left}}, \beta') = (q_{\text{right}}, \gamma', L).$$

- A variant of this interleaving method is to use a larger tape alphabet in which each tape cell of the one-tape machine contains a pair of symbols from the original machine.

Figure 5: *Simulation of a Turing machine with a two-way infinite tape by a machine with a one-way tape. The top diagram shows the initial preparation: the initial position and all the cells to the right are copied to alternating cells of the one-way tape, and the two leftmost cells are marked. The bottom figure shows a snapshot during the simulation. Cells to the left of the initial position of the two-way machine appear in reversed order in alternate cells of the one-way machine. If the two-way machine is about to move one cell to the right from the illustrated configuration, the one-way machine will move* **two** *cells to the* **left.**

- A very similar strategy is to split the two halves again and use a one-way two-tape machine.. A rightward transition of the original machine induces a rightward transition on the first tape, and a leftward transition on the second. We can then simulate the one-way two-tape machine by a one-way one-tape machine, just as we did in class with two bidirectional tapes. So the simulation here is a two-step process. (This also incurs a large time penalty in the second simulation of the two-tape machine by a one-tape machine.)

- The final method is to mark the left end of the tape, as usual. Whenever the original machine calls for a leftward transition off the leftmost cell, the new machine first shifts its entire tape contents one cell to the right, and then executes the transition. In so doing, we need to be sure that the symbol in the leftmost cell is always marked, and the in the other cells unmarked. We saw in 2(a) how to do the rightward shift. This method also incurs a large time penalty.

5 Hilbert's Tenth Problem

When we started the class, I mentioned the decision problem *Hilbert's Tenth Problem:* Given a multivariable polynomial with integer coefficients, something like

$$x^2y - 13x^3y^4 + 27,$$

determine whether there are integer values of x and y that make the polynomial evaluate to 0. The solution to Hilbert's problem is that this problem is undecidable. In terms of our formal definition, we can obviously encode polynomials as strings–the one above might be encoded as

```
x x y - 1 3 x x x y y y y +2 7.
```

The question then becomes whether the set of encodings of polynomials that have integer roots is a decidable language. The answer is 'no'. (This is a very difficult problem that took mathematicians many years to solve.)

Prove that it is a *Turing-recognizable* language. You should give a very high-level argument—so you don't need to talk about Turing machines at all. Just describe a procedure that will correctly answer 'yes' for exactly the polynomials that have integer roots. (This is actually an easy problem with practically no technical detail, intended to make sure that you understand the underlying concept of the difference between 'decidable' and 'Turing-recognizable'.)

Solution. If the polynomial has m variables, we substitute each m-tuple of integers $(i_1, i_2, \ldots, i_m)$ for the variables in the polynomial. If the result is zero, the algorithm says 'Yes'. This solution requires that we have some algorithm for generating the sequence of all m-tupes. Here is one such method: For each $i = 0, 1, 2, \ldots$, list all triples (there are only finitely many) such that the sum of the absolute values of the components is equal to i. With $m = 3$ this gives:

$$i = 0 : (0, 0, 0).$$

$$i = 1 : (\pm 1, 0, 0), (0, \pm 1, 0), (0, 0, \pm 1).$$

$$i = 2 : (\pm 2, 0, 0), (\pm 1, \pm 1, 0), (\pm 1, 0, \pm 1), (0, \pm 2, 0), (0, \pm 1, \pm 1,), (0, 0, \pm 2),$$

etc.

Thus if the polynomial has a root, this algorithm will find it and answer yes. If the polynomial has no root, the algorithm runs forever.

Note that it is crucial to be able to enumerate the triples in such a manner that we will never skip one. It is incorrect to say that you are testing the possible solutions in the order $(0, 0, 0), (1, 0, 0), (2, 0, 0)$, and so on, because you will never reach $(0, 1, 0)$ this way. It is not quite enough to say 'test all possible triples of integers', or even 'the set of triples of integers is countable, so they can be enumerated'. That's on the right track, but you have to describe an algorithm for performing the enumeration.