CSCI3390-Assignment 2.

due Tuesday, September 18

This is the only assignment in which you will actually have to design Turing machines. For these problems (1,2 and 3) you must use the simulator software provided and submit your specification files as part of your solutions. Note that there are several options for Problems 1 and 2.

By the way, the simulator will reject if an applicable transition is not found, so you don't have to explicitly give the transitions into the reject state. This is the same convention we use in drawing the state-transition diagrams.

Problem 4 asks you for a higher-level argument, describing a simulation of one kind of machine by another, and Problem 5 asks you for a very high-level argument, in which you need not talk about Turing machines at all.

In terms of difficulty, options (b) and (c) for Problem 1 are pretty involved, and Problem 4 requires more of you than the others. Problem 5 requires something *different* from you–an intuitive understanding of the terms *decidable* and *Turing recognizable.*

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 $\#1\#0$.

2 The move-over machine

A key step in our proof that a two-tape machine can be simulated 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.

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

```
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.)

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.)

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 ... there is no solution: There is no general algorithm for solving this problem. 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.)

(a) 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 and never answer ‘yes’ for polynomials that have no 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’.)