Adam Gincel

*I pledge my Honor that I have abided by the Stevens Honor System.*

Equal 0s and 1s Turing Machine

init: q0

accept: qAccept

q0,0,\_,\_

q0,0,0,\_,>,>,-

q0,1,\_,\_

q0,1,\_,1,>,-,>

q0,\_,\_,\_

q1,\_,\_,\_,-,<,<

q1,\_,0,1

q1,\_,x,x,-,<,<

q1,\_,0,\_

qReject,\_,x,\_,-,-,-

q1,\_,\_,1

qReject,\_,\_,x,-,-,-

q1,\_,\_,\_

qAccept,\_,\_,\_,-,-,-

AntiPalindrome Turing Machine

init: qCopy

accept: qAccept

qCopy,0,\_

qCopy,0,0,>,>

qCopy,1,\_

qCopy,1,1,>,>

qCopy,\_,\_

qReturn,\_,\_,-,<

qReturn,\_,0

qReturn,\_,0,-,<

qReturn,\_,1

qReturn,\_,1,-,<

qReturn,\_,\_

qTest,\_,\_,<,>

qTest,0,0

qReject,0,0,-,-

qTest,1,1

qReject,1,1,-,-

qTest,0,1

qTest,0,1,<,>

qTest,1,0

qTest,1,0,<,>

qTest,\_,\_

qAccept,\_,\_,-,-

3.2

//q4 means moving a 0 to the end

q4,0

q4,0,>

q4,1

q4,1,>

q4,#

q4,#,>

q4,\_

qReset,0,<

//q5 means moving a 1 to the end

q5,0

q5,0,>

q5,1

q5,1,>

q5,#

q5,#,>

q5,\_

qReset,1,<

qReset,0

qReset,0,<

qReset,1

qReset,1,<

qReset,#

qReset,#,<

qReset,x

q3,x,>

qClean,x

qClean,\_,<

qClean,\_

qReachString,\_,>

qReachString,\_

qReachString,\_,>

qReachString,0

qAccept,0,-

qReachString,1

qAccept,1,-

Reverse String Turing Machine

init: q0

accept: qAccept

q0,0 //get to # to prove valid string

q0,0,>

q0,1

q0,1,>

q0,\_ //only one string??

qReject,\_,-

q0,#

q1,#,> //write # at end

q1,#

q1,#,>

q1,0

q1,0,>

q1,1

q1,1,>

q1,\_

qLeft,#,<

qLeft,0

qLeft,0,<

qLeft,1

qLeft,1,<

qLeft,#

qLeft,#,<

qLeft,\_

q3,\_,>

q3,0

q4,x,>

q3,1

q5,x,>

q3,#

qClean,\_,<

b. 1#1

(q1, 1#1), (x, q3, #1), (x#, q5, 1), (x#, q6, x), (x#x, q7), (x, q1, #x), (x#, q8, x), (x#x, qAccept), String Accepted

c. 1##1

(q1, 1##1), (x, q3, ##1), (x#, q5, #1), (undefined behavior, q5 only responds to x or 1, so string is rejected.)

d. 10#11

(q1, 10#11), (x, q3, 0#11), (x0#, q5, 11), (x0#x, q6, 1), (x0, q7, #x1), (x, q1, 0#x1), (xx, q2, #x1), (xx#x, q4, 1), (undefined behavior, q4 has no response for 1, so string is rejected.)

e. 10#10

(q1, 10#10), (x, q3, 0#10), (x0#, q5, 10), (x0#x, q6, 0), (x0, q7, #x0), (x, q1, 0#x0), (xx, q2, #x0), (xx#x, q4, 0), (xx#xx, q6), (xx#, q7, xx), (xx, q1, #xx), (xx#, q8, xx), (xx#xx, qAccept), String Accepted

3.8

b. When you find a 1, x it out and go to the beginning of the string, then search through for a 1. If you find a 0, x it out then look for a second 0. If you find the second 0, go back to the beginning and look for a 1 again. If, while looking for a 1, you reach the end of the string, the string is valid. If, while looking for a 0, you reach the end of the string, your string is invalid, and you reject.

c. You run the previously described Turing Machine. If that machine accepts, you reject. If that machine rejects, you accept. This reduces the problem to a previously solved one.

3.11

A Turing Machine with doubly infinite tape is computationally identical in power to a regular Turing Machine. To start, to perfectly emulate the behavior of a regular Turing machine, the doubly infinite machine simply ignores the left half of its infinite tape. In addition, we know that doubly infinite machines are no more powerful than regular machines because a two-taped machine can perfectly emulate the behavior of a doubly infinite machine, and multi-taped machines have already been proven to be computationally identical to regular machines.

3.15 b.

Given two decidable languages A and B, we know there must be Turing machines that recognize them, we’ll call them Ma and Mb. Given an input in the form xy, there must be a place to split it in half where x exists in A and y exists in B. Any string only has a finite number of ways to be split, so this is possible deterministically or non-deterministically. We can make a machine, called Mc, that will split any input into x and y, then feed x into Ma and y into Mb. If they both accept, then Mc accepts, otherwise Mc rejects. This lets us see that the language AB is decidable.

3.15 c.

Similarly to the previous example, given a language L, we must be able to split a given input into substrings in the form L, LL, LLL, etc, to show they exist in L\*. A Machine ML\* would run as so: Run a machine ML that accepts a given string in language L for each substring of the input which exists in the in the form L\* (this can be done via brute force or non-deterministically.) If every possible split of the input has been tried and has failed, then reject. If one is found that accepts, accept.

3.15 d.

Very simply, a complement of a language is all things not in the language. Given a machine Mx that accepts all strings in a language X, we can find the complement of X by running a machine which rejects when Mx accepts and accepts when Mx rejects.

3.15 e.

Intersection is very easy to show. Given two machines Ma and Mb, which accept in Languages A and B respectively, we can find if a given input is in their intersection by running the same input through both machines Ma and Mb. If either machine rejects, we reject the input, but if both machines accept, the input is within the intersection of A and B.

**Extra Credit:**

qReset,H //get back to beginning

qReset,H,<

qReset,T

qReset,T,<

qReset,x

qReset,x,<

qReset,\_

qCheck,\_,>

qCheck,x //all xs valid

qCheck,x,>

qCheck,\_ //made it boys

qAccept,\_,-

qCheck,T //check if any Hs

qCanContinue,T,>

qCheck,H //flip this one

q0,H,-

qCanContinue,T

qCanContinue,T,>

qCanContinue,x

qCanContinue,x,>

qCanContinue,H

q0,H,-

qCanContinue,\_

qReject,\_,-

name: Coin Flip

init: q0

accept: qAccept

q0,H //scan right for H

q1,x,>

q0,T

q0,T,>

q0,x

q0,x,>

q1,H //flip right adjacent

q2,T,<

q1,T

q2,H,<

q1,x

q2,x,<

q1,\_

q2,\_,<

q2,x //move one left, guaranteed x

q3,x,<

q3,H //flip left adjacent

qReset,T,-

q3,T

qReset,H,-

q3,x

qReset,x,-

q3,\_

qReset,\_,-