Adam Gincel

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

4.1

a) 0100 is accepted

b) 011 is not accepted

c) No. The input only has one component; it’s not in a correct form.

d) No. The first component is not a regular expression, so the input is not in the correct form.

e) M’s language is not empty, so no.

f) M accepts the same language as itself, so yes.

4.2

We define a language L = {<M, R> | M is a DFA and R is a Regular Expression with L(M) = L(R)}

As seen previously, we can make a Turing Machine F that decides the language EQdfa = {<A, B> | A and B are DFAs and L(A) = L(B)}. We can make a TM T that decides L

T = “On input {M, R> where M is a DFA and R is a regular expression

Convert R into a DFA using the algorithm used in the proof of Kleene’s Theorem

Run TM F on <M, R>

If F accepts, accept. Otherwise, reject.

4.3

We know Edfa is decidable, and we can build a Turing Machine E that decides it. We can also construct a Turing Machine R that decides ALLdfa

R = On input <A> where A is a DFA

Construct a DFA B that recognizes the complement of L(A), by simply swapping accept/reject states

Run Turing Machine E on input <B>

If E accepts, accept. Otherwise, reject.

4.7

Say we try to list out every infinitely long binary string in a list. Once we think we have them all, we diagonally digits from each string, taking the first digit from the first string, the second digit from the second string, etc. At the end we would have an infinitely long binary string, and then we take its inverse, toggling each bit. We have now guaranteed we have a string that was not in our set already, even though we thought we had listed them all. This is possible forever, so it is impossible to begin to list every infinitely long binary string. B is uncountably infinite.

4.8

The natural numbers are countable, and T is a subset of the natural numbers (N). A subset of a countable set is countable, so T is countable.

4.16

A Turing machine T can be designed to decide A.

On a given input R

Construct a regular expression E that accepts every string containing the substring 111

Then test whether L(E) == L(R), using EQre decider W, proven below, accepting if W accepts and rejecting otherwise.

EQre = {<A, B> | A and B are Res and L(A) = L(B)}

EQre is a decidable language.

W = on A and B which are both regular expressions, convert them first to equivalent NFAs then algorithmically convert those NFAs to DFAs. Test these two DFAs using the decider that decides EQdfa, which we know already exists. If that decider accepts, accept. Otherwise, reject.

4.21

You can make a DFA N which accepts L(M)R intersected with the complement of L(M). From there, <M> only exists in the stated machine S iff <N> exists in EDFA. Because we can create N from M (since we can make DFAs for L(M)R and the complement of L(M)), S is decidable because EDFA is decidable.

Fibonacci Turing Machine:

name: Fibbonacci Recognizer

init: qLoad

accept: qAccept

//load 1 onto tape 3

qLoad,\_,\_,\_

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

qLoad,0,\_,\_

qCheck,0,\_,0,-,-,-

//Check if tape 1 == tape 3

qCheck,0,\_,0

qCheck,0,\_,0,>,-,>

qCheck,0,\_,\_

qReset,0,0,\_,-,-,<

qCheck,\_,\_,\_

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

qCheck,\_,\_,0

qReject,\_,\_,0,-,-,-

qCheck,0,0,0

qCheck,0,0,0,>,-,>

qCheck,0,0,\_

qReset,0,0,\_,-,<,<

qCheck,\_,0,\_

qAccept,\_,0,\_,-,-,-

qCheck,\_,0,0

qReject,\_,0,0,-,-,-

//get all tapes to left, then

//get next fib number

qReset,0,\_,\_

qReset,0,\_,\_,<,-,-

qReset,\_,0,\_

qReset,\_,0,\_,-,<,-

qReset,\_,\_,0

qReset,\_,\_,0,-,-,<

qReset,0,0,0

qReset,0,0,0,<,<,<

qReset,\_,0,0

qReset,\_,0,0,-,<,<

qReset,0,\_,0

qReset,0,\_,0,<,-,<

qReset,0,0,\_

qReset,0,0,\_,<,<,-

qReset,\_,\_,\_

qMath1,\_,\_,\_,>,>,>

//math1: get to tape 3 end

qMath1,0,0,0

qMath1,0,0,0,-,-,>

qMath1,0,0,\_

qMath2,0,0,\_,-,-,-

//Math2: Append tape2 to tape3

qMath2,0,0,\_

qMath2,0,0,0,-,>,>

qMath2,0,\_,\_

qMath3,0,\_,\_,-,<,<

//Math3: move back by tape2

qMath3,0,0,0

qMath3,0,0,0,-,<,<

qMath3,0,\_,0

qMath4,0,\_,0,-,>,-

//Math4: Write old tape3 to tape2

qMath4,0,0,0

qMath4,0,0,0,-,>,<

qMath4,0,\_,0

qMath4,0,0,0,-,>,<

qMath4,0,\_,\_

qCheck,0,\_,\_,-,<,>

BONUS:

name: Binary Comparison

init: qClearAZeros

accept: qAccept

qClearAZeros,0,\_,\_

qClearAZeros,\_,\_,\_,>,-,-

qClearAZeros,1,\_,\_

qReachBZeros,1,\_,\_,>,-,-

qReachBZeros,1,\_,\_

qReachBZeros,1,\_,\_,>,-,-

qReachBZeros,0,\_,\_

qReachBZeros,0,\_,\_,>,-,-

qReachBZeros,#,\_,\_

qClearBZeros,#,\_,\_,>,-,-

qClearBZeros,0,\_,\_

qClearBZeros,x,\_,\_,>,-,-

qClearBZeros,1,\_,\_

qCopyB,1,\_,\_,-,-,-

qCopyB,1,\_,\_

qCopyB,1,\_,1,>,-,>

qCopyB,0,\_,\_

qCopyB,0,\_,0,>,-,>

qCopyB,\_,\_,\_

qBackToA,\_,\_,\_,<,-,-

qBackToA,0,\_,\_

qBackToA,0,\_,\_,<,-,-

qBackToA,1,\_,\_

qBackToA,1,\_,\_,<,-,-

qBackToA,x,\_,\_

qBackToA,x,\_,\_,<,-,-

qBackToA,#,\_,\_

qCopyA,#,\_,\_,<,-,<

qCopyA,0,\_,0

qCopyA,0,0,0,<,<,<

qCopyA,0,\_,1

qCopyA,0,0,1,<,<,<

qCopyA,0,\_,\_

qCopyA,0,0,\_,<,<,<

qCopyA,1,\_,0

qCopyA,1,1,0,<,<,<

qCopyA,1,\_,1

qCopyA,1,1,1,<,<,<

qCopyA,1,\_,\_

qCopyA,1,1,\_,<,<,<

qCopyA,\_,\_,0

qClearInput,\_,\_,0,>,-,-

qCopyA,\_,\_,1

qClearInput,\_,\_,1,>,-,-

qCopyA,\_,\_,\_

qClearInput,\_,\_,\_,>,-,-

qClearInput,1,\_,\_

qClearInput,\_,\_,\_,>,-,-

qClearInput,0,\_,\_

qClearInput,\_,\_,\_,>,-,-

qClearInput,#,\_,\_

qClearInput,\_,\_,\_,>,-,-

qClearInput,x,\_,\_

qClearInput,\_,\_,\_,>,-,-

qClearInput,\_,\_,\_

qCompare,\_,\_,\_,-,>,>

//if bottom tape is at 1

qClearInput,1,\_,1

qClearInput,\_,\_,1,>,-,-

qClearInput,0,\_,1

qClearInput,\_,\_,1,>,-,-

qClearInput,#,\_,1

qClearInput,\_,\_,1,>,-,-

qClearInput,x,\_,1

qClearInput,\_,\_,1,>,-,-

qClearInput,\_,\_,1

qCompare,\_,\_,1,-,-,-

//if bottom tape is at 0

qClearInput,1,\_,0

qClearInput,\_,\_,0,>,-,-

qClearInput,0,\_,0

qClearInput,\_,\_,0,>,-,-

qClearInput,#,\_,0

qClearInput,\_,\_,0,>,-,-

qClearInput,x,\_,0

qClearInput,\_,\_,0,>,-,-

qClearInput,\_,\_,0

qCompare,\_,\_,0,-,-,-

//compare

qCompare,\_,1,\_

qResetForA,\_,1,\_,-,<,-

qCompare,\_,0,\_

qResetForA,\_,0,\_,-,<,-

qCompare,\_,\_,1

qResetForB,\_,\_,1,-,-,<

qCompare,\_,\_,0

qResetForB,\_,\_,0,-,-,<

qCompare,\_,1,1

qCompare,\_,1,1,-,>,>

qCompare,\_,0,0

qCompare,\_,0,0,-,>,>

qCompare,\_,1,0

qResetForA,\_,1,0,-,<,-

qCompare,\_,0,1

qResetForB,\_,0,1,-,-,<

qCompare,\_,\_,\_

qResetForA,\_,\_,\_,-,<,-

//Reset A to beginning for copy

qResetForA,\_,1,\_

qResetForA,\_,1,\_,-,<,-

qResetForA,\_,0,\_

qResetForA,\_,0,\_,-,<,-

qResetForA,\_,\_,\_

qAnswerA,\_,\_,\_,-,>,-

qResetForA,\_,1,1

qResetForA,\_,1,1,-,<,-

qResetForA,\_,0,1

qResetForA,\_,0,1,-,<,-

qResetForA,\_,\_,1

qAnswerA,\_,\_,1,-,>,-

qResetForA,\_,1,0

qResetForA,\_,1,0,-,<,-

qResetForA,\_,0,0

qResetForA,\_,0,0,-,<,-

qResetForA,\_,\_,0

qAnswerA,\_,\_,0,-,>,-

//Reset B to beginning for copy

qResetForB,\_,\_,1

qResetForB,\_,\_,1,-,-,<

qResetForB,\_,\_,0

qResetForB,\_,\_,0,-,-,<

qResetForB,\_,\_,\_

qAnswerB,\_,\_,\_,-,-,>

qResetForB,\_,1,1

qResetForB,\_,1,1,-,-,<

qResetForB,\_,1,0

qResetForB,\_,1,0,-,-,<

qResetForB,\_,1,\_

qAnswerB,\_,1,\_,-,-,>

qResetForB,\_,0,1

qResetForB,\_,0,1,-,-,<

qResetForB,\_,0,0

qResetForB,\_,0,0,-,-,<

qResetForB,\_,0,\_

qAnswerB,\_,0,\_,-,-,>

//Copy A to Input

qAnswerA,\_,1,\_

qAnswerA,1,1,\_,>,>,-

qAnswerA,\_,0,\_

qAnswerA,0,0,\_,>,>,-

qAnswerA,\_,\_,\_

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

qAnswerA,\_,1,1

qAnswerA,1,1,1,>,>,-

qAnswerA,\_,0,1

qAnswerA,0,0,1,>,>,-

qAnswerA,\_,\_,1

qAccept,\_,\_,1,-,-,-

qAnswerA,\_,1,0

qAnswerA,1,1,0,>,>,-

qAnswerA,\_,0,0

qAnswerA,0,0,0,>,>,-

qAnswerA,\_,\_,0

qAccept,\_,\_,0,-,-,-

//copy B to input

qAnswerB,\_,\_,0

qAnswerB,0,\_,0,>,-,>

qAnswerB,\_,\_,1

qAnswerB,1,\_,1,>,-,>

qAnswerB,\_,\_,\_

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

qAnswerB,\_,1,0

qAnswerB,0,1,0,>,-,>

qAnswerB,\_,1,1

qAnswerB,1,1,1,>,-,>

qAnswerB,\_,1,\_

qAccept,\_,1,\_,-,-,-

qAnswerB,\_,0,0

qAnswerB,0,0,0,>,-,>

qAnswerB,\_,0,1

qAnswerB,1,0,1,>,-,>

qAnswerB,\_,0,\_

qAccept,\_,0,\_,-,-,-