Project 1: Turing Machines & DFA's

Spring 2022 - due April 15, 2020

Brian Lucero

David Cortez

Ricardo Barojas

Your submission should contain code and documentation as a `.pdf` file, a `.ipynb` file, or a tarball (`.tar.gz`, `.tgz`, `.tar.Z`, `.tar.bz2`, or `.tar.xz`) file containing the following items:

1. A description of how DFAs and their input are represented as binary strings

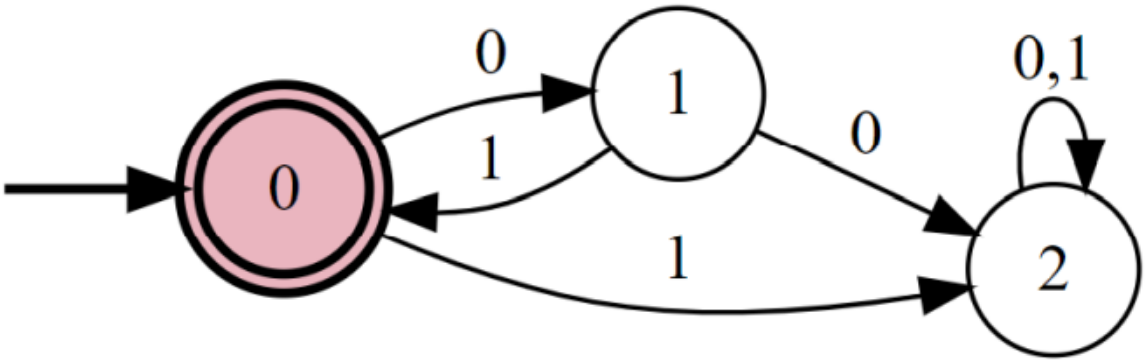
The input string is 0s,1s separated by # from the encoded DFA.

Each **state** and their **transitions** are represented as a tuple (**a, b, c**) encoded as $1^a01^b01^c$ for **a, b, c** $\in$ in N

- Each final **state**, d, is represented as 1^d0 for, d $\in$ N
- In the tuple (**a,b,c**), **a** represents the current state, **b** represents the new state after reading a **0** input, and **c** represents the new state after reading a **1** input.
- The 00 token string deliminates objects T and S

Example:

input#10110111011011101011101110111001 = T: (1, 11, 111,11,111,1,111,111,111), S: {1} = T: ((q0, q1, q2), (q1, q2, q0), (q2, q2, q2)), S: {q0}



2. Versions of the DFAs in Figures 6.3 and 6.4 of the textbook and several input strings using the representations you have defined

Figure 6.3 **input#**101011011011010011

Accepted:
001#101011011011010011
010#101011011011010011
Rejected:
011#101011011011010011

Figure 6.4 **input#**10111011110111011110110110101111011110111101111001

Accepted:
010#10111011110111011110110110101111011110111101111001
010010#101110111101110111101101101011110111101111011111001
Rejected:
101#10111011110111011110110110101111011110111101111001

3. Your Turing Machine or NAND-TM program to simulate execution of a DFA given its representation and input

The URL link below contains the code for the Turing Machine that we designed for this project:

http://turingmachinesimulator.com/shared/bjfvosxivo

There is a text file included in the github repository for this project at:

https://github.com/13rianlucero/439-project1/blob/main/TuringMachineWebsiteCode.txt

4. A description of the design of your Turing Machine, explaining how it accomplishes its task

The machine starts by reading in the input up to the # sign and isolating it on a second tape.

```
6  //qcopyinput
7  qcopyinput, 1,_,_
8  qcopyinput, _,1,_,>,>,-
9
10 qcopyinput, 0,_,_
11 qcopyinput, _,0,_,>,>,-
12
13 qcopyinput, #,_,_
14 qreturninput, #,_,_,-,<,-
```

Then we read in the first state taken as the starting state and copy it onto our third tape as the current state.

```
26 //qreadstate
27 qreadstate, 1,1,_
28 qreadstate, 1,1,1,>,-,>
29
30 qreadstate, 0,1,_
31 qinputcheck, 0,1,_,>,-,<
32
33 qreadstate, 1,0,_
34 qreadstate, 1,0,1,>,-,>
35
36 qreadstate, 0,0,_
37 qinputcheck, 0,0,_,>,-,<
```

Then we check the first bit of the input, and if it's a 0 we take the first argument of the state and a 1 means we take the second argument. The decision to combine what is normally two

separate rule checks into one is because a dfa looking at binary inputs only has to check between 0 and 1, and combining these into one motion saves execution time.

```
39 //qinputcheck
40 qinputcheck, 1,0,1
41 qcopystate, 1,0,1,-,>,-
42
43 qinputcheck, 1,1,1
44 q1start,1,1,1,-,-,-
45
46 qinputcheck, 1,0,_
47 qcopystate, 1,0,_,-,>,-
48
49 qinputcheck, 1,1,_
50 q1start,1,1,_,-,>,-
51
```

```
 99  //q1start
100  q1start, 1,1,1
101  q1start,1,1,1,>,-,-
102
103  q1start, 1,1,_
104  q1start,1,1,_,>,-,-
105
106  q1start, 1,_,_
107  q1start,1,_,_,>,-,-
108
109  q1start, 1,0,1
110  q1start,1,0,1,>,-,-
111
112  q1start, 1,0,_
113  q1start,1,0,_,>,-,-
114
115  q1start,0,1,1
116  qcopystate,0,1,1,>,-,-
117
118  q1start,0,0,1
119  qcopystate,0,0,1,>,-,-
120
121  q1start,0,1,_
122  qcopystate,0,1,_,>,-,-
123
124  q1start,0,0,_
125  qcopystate,0,0,_,>,-,-
126
127  q1start,0,_,_
128  qcopystate,0,_,_,>,-,-
```

After taking either the first or second argument, the old current state is cleared from the third tape and replaced with the new argument.

```
270
271  //qclearstate
272  qclearstate, 1,_,1
273  qclearstate, 1,_,_,-,-,<
274
275  qclearstate, 0,_,1
276  qclearstate, 0,_,_,-,-,<
277
278  qclearstate, 1,0,1
279  qclearstate, 1,0,_,-,-,<
280
281  qclearstate, 0,0,1
282  qclearstate, 0,0,_,-,-,<
283
284  qclearstate, 0,1,1
285  qclearstate, 0,1,_,-,-,<
286
287  qclearstate, 1,1,1
288  qclearstate, 1,1,_,-,-,<
289
290  qclearstate, 1,_,_
291  qinputcheck, 1,_,_,-,-,<
292
293  qclearstate, 0,_,_
294  qinputcheck, 0,_,_,-,-,<
295
296  qclearstate, 1,0,_
297  qinputcheck, 1,0,_,-,-,<
298
299  qclearstate, 0,0,_
300  qinputcheck, 0,0,_,-,-,<
301
302  qclearstate, 0,1,_
303  qinputcheck, 0,1,_,-,-,<
304
305  qclearstate, 1,1,_
306  qinputcheck, 1,1,_,-,-,<
307
```

Then the first and third tape reset to their origins and start to loop through checking for the current state. To avoid looking at an entire dfa rule as 3 states during this process, the machine only checks every 3rd state on the tape.

```
194  //qstatecheck
195  qstatecheck,1,1,1
196  qstatecheck,1,1,1,>,-,>
197
198  qstatecheck,1,0,1
199  qstatecheck,1,0,1,>,-,>
200
201  qstatecheck,0,1,1
202  qresetcurrent,0,1,1,-,-,-
203
204  qstatecheck,0,0,1
205  qresetcurrent,0,0,1,-,-,-
206
207  qstatecheck,0,1,_
208  qclearstate,0,1,_,>,-,<
209
210  qstatecheck,0,0,_
211  qclearstate,0,0,_,>,-,<
212
213  qstatecheck,1,1,_
214  qresetcurrent,1,1,_,>,-,<
215
216  qstatecheck,1,0,_
217  qresetcurrent,1,0,_,>,-,<
218
219  qstatecheck,1,_,1
220  qchecksuccess1,1,_,1,-,-,-
221
```

```
222  //qnextstate1
223  qnextstate1,1,1,1
224  qnextstate1,1,1,1,>,-,-
225
226  qnextstate1,1,0,1
227  qnextstate1,1,0,1,>,-,-
228
229  qnextstate1,0,1,1
230  qnextstate2,0,1,1,>,-,-
231
232  qnextstate1,0,0,1
233  qnextstate2,0,0,1,>,-,-
234
235  //qnextstate2
236  qnextstate2,1,1,1
237  qnextstate2,1,1,1,>,-,-
238
239  qnextstate2,1,0,1
240  qnextstate2,1,0,1,>,-,-
241
242  qnextstate2,0,1,1
243  qstatecheck,0,1,1,>,-,-
244
245  qnextstate2,0,0,1
246  qstatecheck,0,0,1,>,-,-
247
```

After it finds the matching state between the first and third tapes, it repeats the process of checking the next input bit on the second tape and taking either the first or second argument as the new current state and resetting to the origin of the first and third tape.

After the second tape has hit a blank representing the end of any inputs, the machine will lock the current state as the end state and loop through the first tape looking for “00” to represent the start of the success states.

```
52  //inputcheck - go to success
53  qinputcheck, 1,_,1
54  qchecksuccess, 1,_,1,-,-,-
55
56  qinputcheck, 1,_,1
57  qchecksuccess,1,_,1,-,-,-
58
59  qinputcheck, 1,_,_
60  qchecksuccess, 1,_,_,-,-,-
61
62  qinputcheck, 1,_,_
63  qchecksuccess,1,_,_,-,-,-
64
```

```
308  //qchecksuccess1
309  qchecksuccess1,1,_,1
310  qchecksuccess1,1,_,1,>,-,-
311
312  qchecksuccess1,0,_,1
313  qcheck0,0,_,1,>,-,-
```

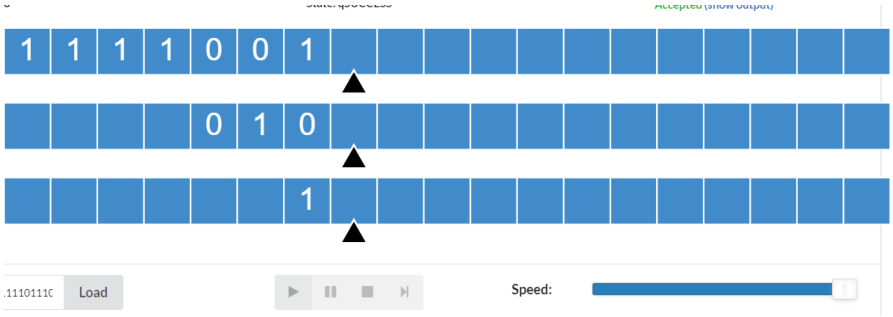
The machine checks for “00” by finding any 0 during the loop and running a check to see if the next bit is also a 0. If not, the machine continues to loop.

```
315  //qcheck0
316  qcheck0,0,_,1
317  qchecksuccess2,0,_,1,>,-,-
318
319  qcheck0,1,_,1
320  qchecksuccess1,1,_,1,>,-,-
```

After finding a “00” the machine moves to the second phase of checksuccess, looping through the succes states and comparing them to the end state. If a success state matches the end state, the machine moves to the qSUCCESS state representing a success. If no match is found, the machine moves to qfailstate, representing a failure.

```
322   //qchecksuccess2
323   qchecksuccess2,1,_,1
324   qchecksuccess2,1,_,1,>,-,>
325
326   qchecksuccess2,1,_,_
327   qresetsuccess,1,_,_,>,-,<
328
329   qchecksuccess2,_,_,1
330   qfailstate,_,_,1,>,-,>
331
332   qchecksuccess2,1,1,_
333   qfailstate,1,1,_,>,-,>
334
335
336   qchecksuccess2,_,_,_
337   qSUCCESS,_,_,_,-,-,-
338
339 //qresetsuccess
340 qresetsuccess,1,_,1
341 qresetsuccess,1,_,1,>,-,<
342
343 qresetsuccess,0,_,1
344 qresetsuccess,0,_,1,-,-,<
345
346 qresetsuccess,0,_,_
347 qchecksuccess2,0,_,_,>,-,>
348
349 qresetsuccess,_,_,_
350 qfailstate,_,_,_,-,-,-
```

After the DFA has been simulated, the output of the machine is saved to the third tape.



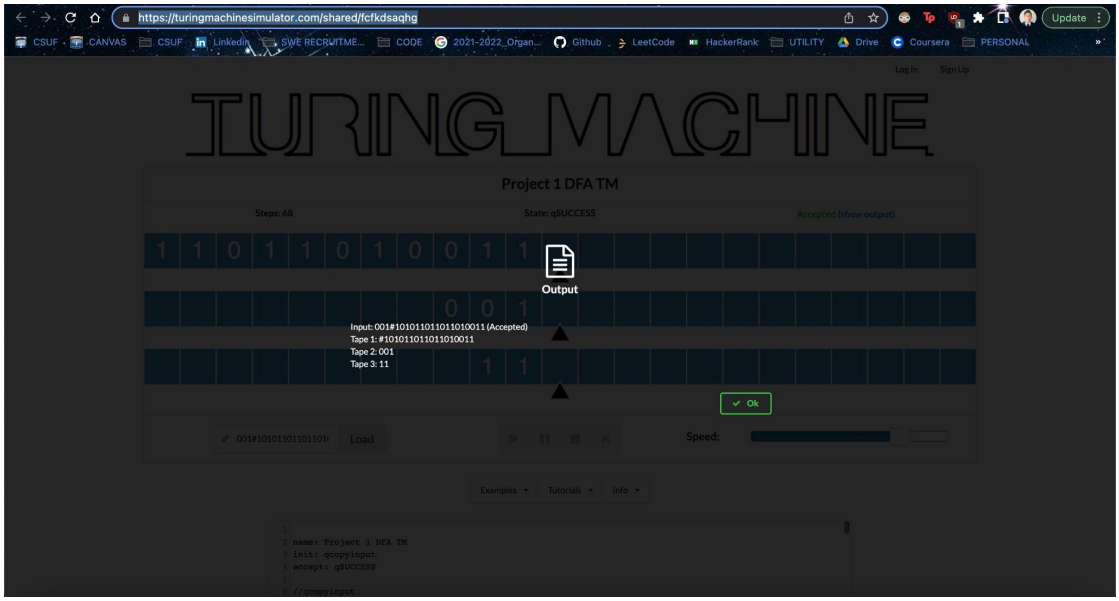
5. Output showing the execution of the Turing Machine or NAND-TM program on several input strings for each of the DFAs in Figures 6.3 and 6.4 of the textbook

For each DFA, we tested both accepted and rejected states

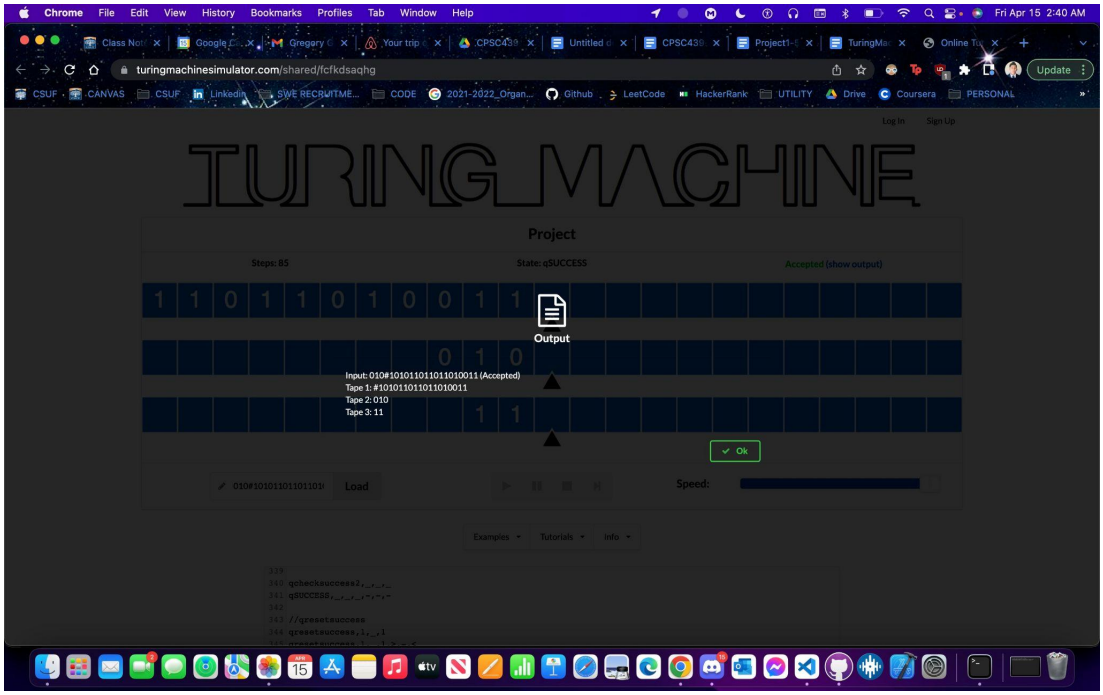
Figure 6.3 **input#**1010110110110100 11

Accepted:

001#1010110110110100 11



010#101011011011010011



Rejected:

011#101011011011010011

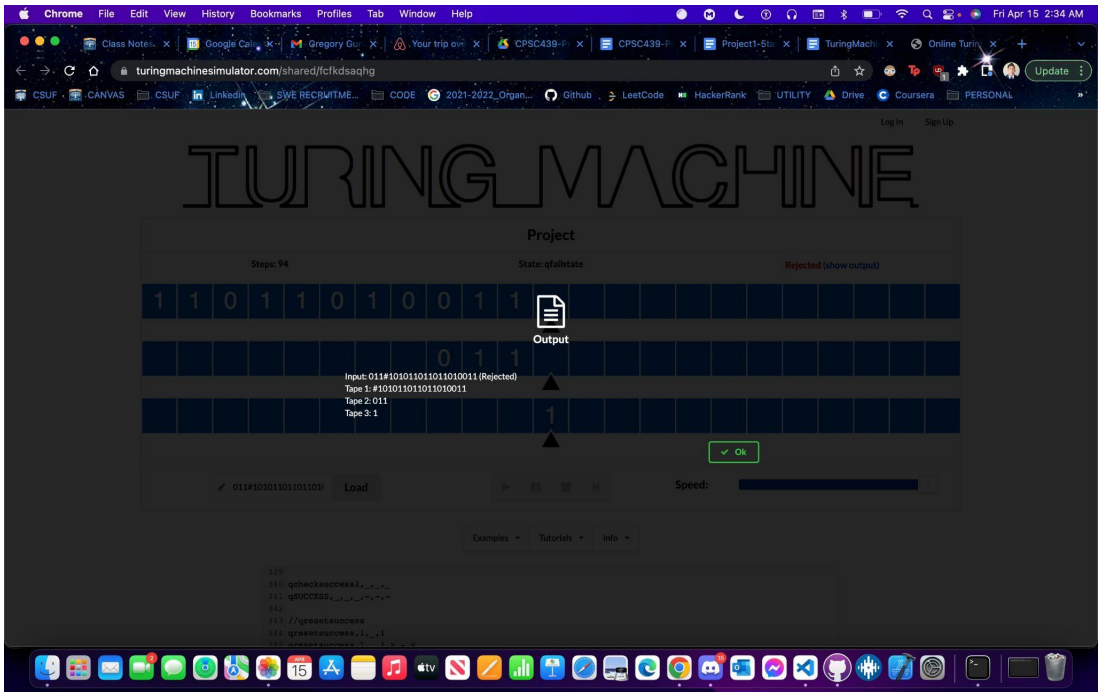
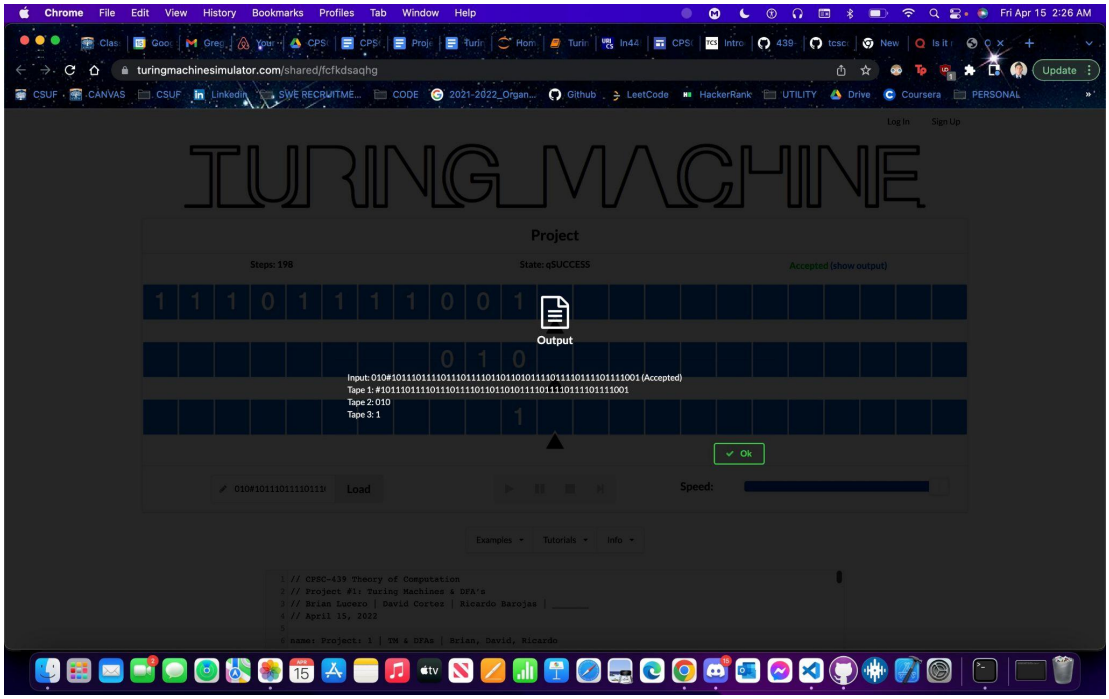


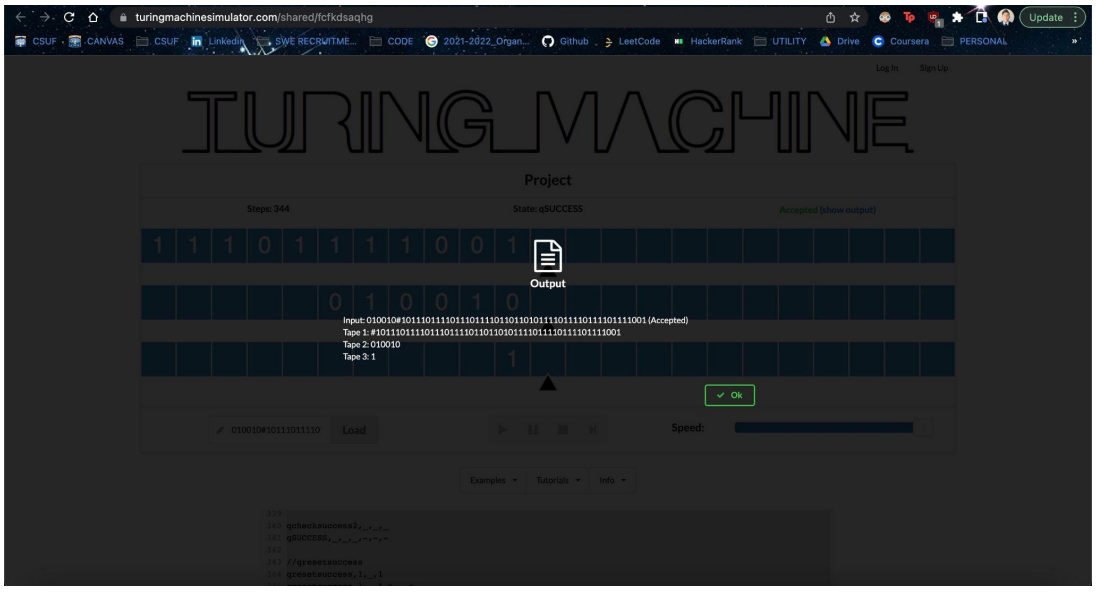
Figure 6.4
input#10111011110111011110110110101111011110111101111001

Accepted:

010#10111011110111011110110110101111011110111101111001



010010#10111011110111011110110110101111011110111101111001



Rejected:

101#10111011110111011110110110101111011110111101111001