CSCE 428/828 Project: DFA Simulation

Part 1: Proposal. Due Friday 3/10/2017
Part 2: Implementation and Report. Due 4/19/2017
In this project you will develop a program to simulate a given DFA. Your program can check whether an input string is accepted by the given DFA. Your program can also check if two given DFAs are equivalent (and give counterexamples if they are not).

**Implementation.** You will write a program $P$ to do two tasks:

1. (25 pts) **Check for membership:** Given as input the description of a DFA $M$ (described below) and a string $s$, your program $P$ returns *Yes* if $M$ accepts $s$ and *No* otherwise.

2. (50 pts) **Check for equivalence:** Given as input the descriptions of two DFA’s $M_1$ and $M_2$, your program $P$ returns *Yes* if $M_1 \equiv M_2$ (i.e., recognizing the same language) and *No* otherwise. Moreover, in the case $M_1 \neq M_2$, $P$ returns a counterexample string $s$ such that $P(M_1, s) \neq P(M_2, s)$.

In addition, you will need to provide at least 3 input examples showing how your program works. For example, for task 1, create your own DFA $M$ (e.g., get some DFA’s from the textbook or class notes) and create several example inputs that are accepted/rejected by $M$. Similarly for task 2, create several $M_1$’s and compare them with various $M_2$’s that are equivalent and not equivalent to $M_1$.

**Writing.** As usual, you will need to clearly explain your work. You will explicitly do these tasks:

1. (10 pts) **Proposal:** before doing the implementation, think about how you would solve these problems. Write (about) a half a page explaining how you plan to solve the implementation tasks 1 and 2 above. Also mention what programming language you plan to use.

2. (15 pts) **Report:** After doing the implementation, write (about) a page describing how your program works. E.g., how do you check for membership and equivalence? How do you generate counterexamples? Anything interesting about your design? Any potential problems?

**Extra Credits.** (50 pts) Read following paper and implement the described algorithm on synthesizing a DFA by asking membership questions and using counterexamples. “Learning regular sets from queries and counterexamples” by Dana Angluin (1987).

**Due Dates.**


**Specifications.** The description of a DFA is given in a *.dfa text file and uses the format illustrated in the following example

```plaintext
# This DFA recognizes \{ x in \{0,1\}* \mid x does not end in 000 \}

states:       qe,   # last bit was a 1 or non-existent
              q0,   # last two bits were 10
              q00,  # last three bits were 100
              q000  # last three bits were 000

input_alphabet:  0,1

start_state:    qe    # no last bit when we start
```

1
accept_states: qe,q0,q00 # accept as long as the last three bits weren't 000

delta: # if we see a 1, reset
qe,1  -> qe
q0,1  -> qe
q00,1 -> qe
q000,1 -> qe

# if we see a 0, count one more 0 than before
qe,0  -> q0
q0,0  -> q00
q00,0 -> q000

# until we get to three, and then just remember
q000,0 -> q000

Notes.

• The alphabet cannot contain whitespace chars, e.g., no input strings such as 0 101.

• You can use any language you want, however your program must run on a CSE machine. You must provide a runTask.sh script that (compiles and) runs your program. This script allows the grader to run your code regardless of what language/compiler is used. So you might have your script call make and then run the compiled program on the first argument to the script. E.g., all the grader has to run is ./runTask.sh dfa_file input_string

• Here are some examples of how we run your program

/path/to/runTask.sh M1.dfa 01010101
Yes
/path/to/runTask.sh M1.dfa 0101000
No
/path/to/runTask.sh M1.dfa M1.dfa
Yes
/path/to/runTask.sh M1.dfa M2.dfa
Yes # M2 is not equivalent to M1
010 # a counterexample showing /path/to/runTask.sh M1.dfa 010 gives diff result than /path/to/runTask.sh M2.dfa

• Turn in your project using CSE Handin