

CS 30151 Spring 2008  
Project 1  
Due on Friday, February 8 at 11:59pm

## Introduction

This project is the first part of the multi-step automata simulator we will be developing this semester. The overall project will consist of simulators for the following major classes of automata:

- finite automata;
- push-down automata;
- and Turing machines.

The projects will have compatible inputs and outputs to permit building on previous parts of this multi-step project. In addition to helping in understanding the concepts, the projects should be useful for verifying homework assignments.

## Project Description

Project 1 consists of developing a program that simulates deterministic finite automata (DFA). The program is to read a description of a DFA and one or more inputs for it (i.e., a single machine definition should support multiple test cases). It then will process an input and output the execution path and other information about it. After processing an input, the program should reset the automaton and proceed with the next input.

## Machine description

A machine description will be provided in a file. The file will consist of text lines defining all 5 parts of a DFA. The format is as follows:

- A line starting with “A:” will list the alphabet  $\Sigma$ , where the symbols are separated with a comma (with no spaces) and the list is terminated by the end-of-line character. For example, the alphabet for binary strings will be given as **A:0,1**.
- A line starting with “Q:” will list the set of states, where the states are separated with a comma (with no spaces) and the list is terminated by the end-of-line character. For example, a list of states can look like **Q:q1,q2,q3**.
- A line starting with “T:” provides a single transition rule. The format of the line is the starting state followed by a comma, the input symbol followed by a comma, and the resulting state (with no spaces). For example, we might have a line **T:q1,0,q2**. There will be multiple lines of this form per machine description. The program must check that both of the states given in the transition function are valid states from the  $Q$  set and the input character is a valid character from the alphabet. Once the input is complete, the program should also check that all transitions were given (i.e., one rule for all state-symbol pairs).
- A line starting with “S:” will specify the start state. For example, we might have **S:q1**. The program must check that the state given is a valid state.

- A line starting with “F:” will specify the list of accept states. Examples are F:q1,q2 and F:q2.

There will be a description of a single machine in a file. The file name will be given as an argument to the program. As a sample description, consider the machine given in Figure 1.4 of the textbook (page 34):

```
A:0,1
Q:q1,q2,q3
T:q1,0,q1
T:q1,1,q2
T:q2,0,q3
T:q2,1,q2
T:q3,0,q2
T:q3,1,q2
S:q1
F:q2
```

### Tape description

Once the program is initialized with an automaton description, it will take tape input from the standard input. It should silently wait for input without any prompts. The format is as follows:

- The first line will contain a single integer by itself indicating how many tape inputs follow.
- If the first line contained a number  $n$ , then the next  $n$  lines will contain tape inputs, one sequence per line. Each line will consist of a list of alphabet symbols, each separated by a comma. For example, if the alphabet is  $\Sigma = \{0, 1\}$ , then one input line could be: 0,0,1,1,0.

### Program output

The program should print all information to the standard output. For each input tape, the program’s output will consist of a trace of program execution. Traces corresponding to different inputs should be separated by a blank line. For each input tape, for each transition (one transition per line) the machine is to print the starting state followed by a comma, the current input character followed by a comma, and the resulting state. When the end of the tape is reached, if the machine is in an accept state, it should print **ACCEPT** on a line by itself. And if the machine is not in an accept state when the end of the tape is reached, it should output **REJECT**. Then machine will reset its state and proceed to processing the next input tape if more inputs are remaining. For example, given the DFA above, if the input is:

```
2
1,1
0,1,0
```

the program’s output will be:

q1,1,q2  
q2,1,q2  
ACCEPT

q1,0,q1  
q1,1,q2  
q2,0,q3  
REJECT

## Program submission and execution

The program should be submitted using your dropbox for this course. The name of the submission should be `proj1`. The program can be written in a programming language of your choice, but **it must run on Linux cluster machines**. So make sure that it compiles and runs in one of the Fitzpatrick labs. If we are unable to compile or run, the **project risks getting zero credit**.

Your submission must include a Makefile with rules to compile the program. Typing `make` in the submission directory should produce an executable named `dfa`. The program should take one argument, which is the name of the file containing a DFA description (according to the format specified). The tape inputs will be provided from the standard input and all output should go to the standard output.

To test the programs, we will use input and output redirection. It is recommended that you test the program using the same mechanism on the sample files (provided) and your own inputs. The program can then be tested as follows:

```
> make
```

This should produce a program executable called `dfa` (or `dfa.java` for Java).

```
> ./dfa dfa1.txt < input1.txt > output1.txt
```

This command line corresponds to program execution using files `dfa1.txt` and `input1.txt`, which are supplied as a part of this project. After the execution, your program's output will be written to file `output1.txt`. The contents of that file should be identical to the contents of the file with the same name supplied as a part of the project. If the program cannot be run as above and requires special handling (i.e., usage of Java interpreter), include such instructions in a `README` file (note that in case of Java, the above command will simply be prefixed with `"java "`).

## Project report

You are encouraged to experiment with the project running it on different automata and inputs. Besides the program itself, your submission should include a project report (a PDF document) describing the following:

- The language(s) and platform used for testing.
- A high level description of the program including major data structures.
- The test programs and tape inputs used to verify correctness of the simulation.
- Any major difficulties encountered or any notable implementation decisions.

To summarize, the submission will include:

- the source file(s);
- Makefile;
- README file if necessary;
- a PDF report file;
- supplements to the report such as test files.