

CS 30151 Spring 2008  
Project 2  
Due on Friday, March 28 at 11:59pm

## Project Description

This project consists of developing the second simulator for this course, pushdown automata (PDA). The overall specifications and guidelines are similar to project 1, and it should be possible to reuse parts of the code written for simulating DFA for this assignment.

Project 2 is to create a program that simulates deterministic pushdown automata (DPDA). The program is to read a description of a DPDA and one or more inputs for it. It will process an input and output the execution path and other information about it. After processing one input, the program should reset the automaton (this includes emptying the stack if necessary) 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 6 parts of a DFA. The format is as follows:

- 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 “A:” will list the input 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 “Z:” will list the stack alphabet  $\Gamma$ , where the symbols are separated with a comma (with no spaces) and the list is terminated by the end-of-line character.
- 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, the expected symbol at the top of the stack (which is being popped), the resulting state, and the symbol to be written on the stack (all with no spaces). For example, we might have a line T:q1,0,a,q2,b. 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, the input character is a valid character from the input alphabet, and both stack symbols are from the stack alphabet.
- 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.

The transition rules might include the empty string, which will be denoted in the machine description by  $\epsilon$ . This will be a special character which will not be permitted to appear as the name of a machine’s state or as a symbol of the input or stack alphabets.

Since the project deals with deterministic PDA, upon reading all transition rules from the machine's description, the program must verify that computation can proceed deterministically. That is, no configuration has choice of more than one move, as specified in the definition of a DPDA.

Also note that, unlike a DFA, a DPDA will not automatically read the next input symbol and the symbol at the top of the stack after each transition. Instead, the transition rules specified for the current state will determine whether the machine is to read the next input character, pop the stack, or do both. Assume that the machine will make as many transitions as possible if its description contains transitions on the empty string.

Stack is assumed to be empty at the beginning of each simulation. If more than one input is specified, the simulator must empty the stack after each simulation and reset the state.

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 presented in class:

```

Q:q0,q1,q2,q3,q4
A:{,(,},)
Z:{,(,$
T:q0,e,e,q1,$
T:q1,(,e,q2,(
T:q1,{,e,q2,{
T:q2,{,(,q3,(
T:q2,{,{,q3,{
T:q3,e,e,q2,{
T:q2,(,{,q4,{
T:q2,(,(,q4,(
T:q4,e,e,q2,(
T:q2,},{,{,q2,e
T:q2,),(,q2,e
T:q2,e,$,q1,$
S:q0
F:q0,q1

```

### 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:

1. the initial state followed by a semicolon and a single space;
2. the input character used in the transition (if none was used, `e` is to be printed) followed by a semicolon and a single space;
3. the stack symbol popped during the transition (if none, `e` is to be printed) followed by a semicolon and a single space;
4. the resulting state followed by a semicolon and a single space;
5. and the contents of the entire stack starting from the top, with each symbol separated by a comma (no spaces).

If the end of the tape is reached and the machine is in an accept state, it should print `ACCEPT` on a line by itself, and should output `REJECT` at the end of execution otherwise. Then machine will reset and proceed to processing the next input tape if more inputs are remaining. For example, given the DPDA above, if the input is:

```
2
{,(,)},}
(,(
```

the program's output will be:

```
q0; e; e; q1; $
q1; {; e; q2; {,$
q2; (; {; q4; {,$
q4; e; e; q2; (,{,$
q2; ); (; q2; {,$
q2; }; {; q2; $
q2; e; $; q1; $
ACCEPT
```

```
q0; e; e; q1; $
q1; (; e; q2; (,$
q2; (; (; q4; (,$
q4; e; e; q2; ((,$
REJECT
```

### Program submission and execution

The program should be submitted using your dropbox for this course. The name of the submission should be `proj2`; that is, create a directory named `proj2`, place all of the files to be submitted into that directory and submit it. 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 `dpda` (if the programming language you are using does not require compilation, Makefile is not needed). The program should take one argument, which is the name of the file containing a DPDA 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 `dpda` (or `dpda.java` for Java).

```
> ./dpda dpda1.txt < input1.txt > output1.txt
```

This command line corresponds to program execution using files `dpda1.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 the algorithms and the data structures used (for storing and verifying the program description, for the simulation itself such as the stack data structure, etc.)
- 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.