CS337 Project 3
Finite State Machines

TA: Yan Li (yanli@cs.utexas.edu)

Release Date: Oct. 27, 2008 (Monday)
Due Date: Nov. 10, 2008 11:59 PM (Monday)

1 Overview

In this project, you and your partner (please find a partner early!) will create some simple Finite State Machines(FSMs), then implement a finite state machine interpreter to test your FSMs on input strings. This project consists of three parts:

part 1: Create seven finite state machines to match the given string patterns. You do not need to turnin anything for this part but they will be used for part 2.

part 2: Translate the diagram of each FSM from part 1 into a format (saved into a file for each FSM) that can be read by a computer program.

part 3: Implement a program that reads any description files of FSMs (obtained from part 2 or any other .fsm files in the same format) and a file of strings, and prints if your finite state machines accepts the input strings.

The alphabet for this project is (Note: no uppercase letter in this alphabet and space is not part of the alphabet.):

`abcdefghijklmnopqrstuvwxyz0123456789~!@#%^&*()-+{}.,`

There are four **uppercase** letters we will use as shorthand for different sets of the alphabet which you should use in your FSM description files in part 2. And you MUST implement these shortcuts in part 3.

('A') Alphabetic: `abcdefghijklmnopqrstuvwxyz`

('D') Digit: `0123456789`

('N') Non-Zero: `123456789`

('S') Symbolic: `~!@#%^&*()-+{}.,`

2 Part 1

Use the notation described in class to create a unique finite state diagram for each of the following seven string descriptions. Number the state in each FSM as 1, 2, 3, ... So the start state for each machine should be 1. Refer to Figure 1 for an example of a finite state diagram. You do not need to turn in anything for this part. But these diagrams will be used in Part 2.

NOTE: You do not need to make reject state in the FSMs. Draw the transitions only for valid inputs and any other input not shown would be considered as going to the reject state.

Warning: The empty string must be rejected by all Finite State Machines.

1. Accept only integers, where integers are defined as either a single 0, or a nonzero digit followed by any number of digits. Examples: 0, 3802 will be accepted. But 061 should be rejected.

2. Accept only comma-separated integers (use integer as defined above). Examples: $2,000$ or $6,500,000$ will be accepted. But 3000 and $3,00,0$ must be rejected. Note that the commas must be in the thousandth, millionth, billionth, etc. positions. If the number is 999 or less it should be accepted.

3. Accept signed and unsigned integers (use integer as defined above). Examples: -5, $+45$, 45, and 0 will be accepted. But $+0$, -45.5, and -0.3 must be rejected.

4. Accept any string that uses the defined alphabet if it contains a keyword "begin" and/or "end". Examples "erikbegin", "xendd" and "ebeginend" will be accepted. But "began" must be rejected.

5. Accept unsigned floating point numbers which MUST contain one decimal point in the number. With floating point numbers, a 0 can be the leading digit but only if it is the only digit before the decimal. Examples: 0.003, 0.20, 0.000, 3.54 will be accepted. But 0..03, 1., 55, and 01.33 must be rejected.

6. Accept a string with up to 3 levels of balanced parentheses, ignoring the characters between, before, or after the parentheses. Reject the string if the parentheses are unbalanced, if there are no parentheses in the string, or if the nesting is deeper than three. Examples: "compute((1+2)+((5+2*8+2)))", and "((()))" will be accepted. But "((((4))))", "(()())", and "4+3" will be rejected.

7. Accept a string of digits if they are non-increasing. Examples: 5443, 4, 22 and 631 will be accepted. But 234 must be rejected.

3 Part 2

In this part, you will convert each of the finite state machine diagrams from part 1 into a text file (.fsm) that can be read by a computer program. The 7 .fsm files must be called "machine1.fsm", ..., "machine7.fsm". The .fsm file format to describe a FSM is as follows.

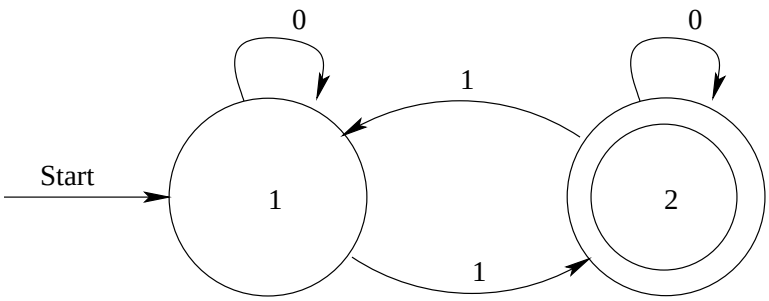

Figure 1: An FSM that accepts strings with an odd number of 1s.

Each line in the file represents a single state in the FSM diagram. Below is the file called `machine0.fsm` which is the machine-readable form of the diagram in Figure 1:

```
1 1:0 2:1
2 1:1 2:0 X
```

Each line has the following format:

```
[State Name] [NextState Name]:[Transitions] [Accepting State?]
```

The first token in each line is the state name. For your implementation you can expect to see only unsigned numbers for state names. In the above example, the first line describes state 1, the second line describes state 2. The state number will not exceed exceed a Java int.

The Finite State Machine Interpreter will always look for, and start with, the state labeled 1. The states do not have to be in order, and there can be gaps in the numbering. A state can only be listed once per file.

After each state name, there can be an arbitrary number of additional tokens. There are two types of tokens, Accepting State Flags, and Next Step Pairs.

An Accepting State Flag (ASF) is the capital letter 'X'. When an ASF is on the line, it means the current state is an accepting state.

Note: The flag can appear before, after, or anywhere in between other Next Step Pair tokens.

In the example file, state 2 has an ASF, so if the input string terminates in state 2, the string is accepted. State 1 does not have an ASF, so if the string ends there, it will be rejected.

A Next Step Pair consists of the next state name, followed by a colon (":") and then the valid characters that allow the transition from current state to next state to take place. There must not be any spaces between the name, colon, and transition characters. In the example, state 1 transitions back to state 1 if a '0' is encountered–a loop. State 1 transitions to state 2 if a '1' is encountered.

If [Transitions] has more than one symbol then you should write all the symbols together without any space. For example, in a FSM M, if state 5 transitions to state 1 when the input character is either 'a' or 'b'; and state 5 transitions to state 2 if a '0' or '1' is encountered, then the line in the corresponding M.fsm file will be as follows:

```
5 1:ab 2:01
```

Remember the shorthand mentioned in Part 1: If the next character in the string is not mentioned in the current state, then the string is immediately rejected (i.e., you should not make a reject state).

You should use the four **uppercase** letters described in Section 1 as a shorthand for different sets of the alphabet in your .fsm files.

For example, To create an FSM that accepts strings which have a non-zero digit as the first character, and zero or more characters after it, you would create a .fsm file with two lines:

```
1 2:N
2 X 2:ADS
```

4 Part 3

In this part, you need to write a Java program "FSM.java" to read ANY .fsm files in the format described in Part 2 (not only the seven .fsm files obtained from Part 2. I will test your code using some FSMs not in Part 1.) and a file "tests.txt" which contains a string on each line.

All the file names are passed from the command line as input parameters to the program "FSM.java". The "FSM.java" should be able to accept one or more, up to at least 10 .fsm files and one test string file. Do NOT implement an interactive prompt to ask for input file names.

Your program should compile and run using the following command (this example has three .fsm files plus the test string file):

```
javac FSM.java
java FSM machine1.fsm machine5.fsm machine9.fsm tests.txt
```

You need to create your own test strings, saved in the file "tests.txt". Assume that my "tests.txt" will have blank lines, and end of line spaces that may need to be trimmed.

You need to evaluate each text string with each FSM you create and display which machines accept the input. The program output should be displayed on the screen.

If you implemented both of the example .fsm files above and created a "tests.txt" that contained:

```
11001101
11111
1thisshouldfailmachine0
```

then the output should look like:

```
machine0.fsm Accepted: 11001101
machine1.fsm Accepted: 11001101
machine0.fsm Accepted: 11111
machine1.fsm Accepted: 11111
machine1.fsm Accepted: 1thisshouldfailmachine0
```

If the string is rejected by a machine, do NOT print anything. The format of the above example output should be strictly followed and they should be case-sensitive.

NOTE: There are four shortcut characters you need to implement, as mentioned in Section 1. The uppercase letter 'A' should cover all alphabetic characters, 'D' for 0 to 9, 'N' for 1 to 9, and 'S' for symbols. You should NOT implement any other shortcuts.

5 Turning in your project

5.1 Files to be submitted

List of the files you must submit:
(The names of the files should be exactly the same as follows and case-sensitive):

1. emails.txt
   "emails.txt" content and format: (You MUST follow the exact format below! Otherwise, no comment and grade will be sent to your and your partner's emails and I will not post on egradebook this time about the comments.):
   ```
   your_CS_account your_email
   partner_CS_account partner_email
   ```

   Example "emails.txt" (Note: There is a space between your_CS_account and your_email):
   ```
   yanli yanli@cs.utexas.edu
   burt burtkk@gamil.com
   ```

2. readme.txt (Do not forget Section#!)

   ```
   NAME of partners;
   Section # of partners;
   EID of partners;
   CODING STATUS: (very important!);
   ```

   IMPORTANT: Summarize your code status in "readme.txt" file. If your code works perfectly, explicitly saying this in the readme.txt. If you code does not work (can not compile, or can not run, or only work for some FSMs or can not output anything, etc.), you need to specifically explain this: what have been done so far and what have not been done, what is working and what is not working to avoid confusion! Put proper comments in all the code files.

3. machine1.fsm
4. machine2.fsm
5. machine3.fsm

6. machine4.fsm
7. machine5.fsm
8. machine6.fsm
9. machine7.fsm
10. FSM.java
11. tests.txt

5.2 turnin

To submit your project, use Linux "turn-in" program exactly as follows. Do NOT create a directory like in project 1 this time!

```
turnin --submit yanli project3 emails.txt readme.txt machine1.fsm
machine2.fsm machine3.fsm machine4.fsm machine5.fsm machine6.fsm
machine7.fsm FSM.java tests.txt
```

To confirm your submission:

```
turnin --list yanli project3
```

ONLY ONE submission per group!

1. Please do NOT submit your files with a directory this time (so it is different from project 1!). Please strictly follow the above command as it is.
2. Please do not turn in a tarred or compressed version of your files.
3. Please strictly follow the files names. Otherwise, your submission may not be graded.
4. No late submission is allowed. The turn-in program will be turned off after the deadline time. And no email submission is accepted.
5. You must make sure your program will work correctly on CS Linux machines. All the text files should be generated using LINUX OS.

6 Questions and Project Clarifications

Please stick to all specifications as written. Changes and clarifications regarding this project will be mailed to you through Blackboard. Though I do not expect any changes, you are responsible for any modifications found there.

Good Luck!