CS 311-E01 Formal Languages and Automata
Project
(Due: 9 AM, Monday, 7/13/2015)

Once a finite state automaton (FSA) is designed, its transition diagram can be translated in a straightforward manner into program code. However, this translation process is considerably tedious if the FSA is large and troublesome if the design is modified. The reason is that the transition information and mechanism are combined in the translation.

To do it differently, we can design a general data structure such as a table or a list to hold the transition information, and to implement in program code only a general transition mechanism. Using this approach, the resulted program is not only smaller, but it can also be modified easily to simulate a different FSA by just changing the transition information and the general data structure holds. We shall refer to such a program as a ***universal finite state automaton***.

This assignment consists of two parts:

Part 1: Implement a universal finite state automaton. To convince you it is not a difficult task, I have suggested an algorithm below. Note that your program has to be designed to handle the FSA where the input transition function is partial, which means there is a default dead-end, or trap state.

```
1   state = initial_state;   exit = false;
2   while not exit do
3   begin
4       symbol = getNextSymbol();
5       if symbol is in alphabet then begin
6           state = getNextState(state, symbol);
7           if state is dead_end then begin exit = true; reject; end
8           end
9       else begin
10          exit = true;
11          if symbol is not the endmarker then reject;
12          else if state is final then accept;
13          else reject;
14      end //if
15  end //while
```

The above algorithm shows how to process one input string and correctly determine whether the string is in the language. You need to augment the algorithm so that your program is able to read in different FSA descriptions from an input file and simulate one machine at a time with any number of test strings.

Part 2: Implement the subset construction algorithm to convert an NFSA without ε-moves to an equivalent DFSA and test the resulting DFSA with the above universal finite state automaton.

Algorithm SubsetConstruction

```
1    dfsa.state[0] = {0};
2    dfsa.num_state = 0;  dfsa.current_state = 0;
3    while dfsa.current_state ≤ dfsa.num_state do
4    begin
5       for each symbol in alphabet do
6       begin
7          ComputeDfsaNextState ();
8          if dfsa.next_state is not a new state then
9             Store (dfsa.current_state, symbol, the dfsa state number of dfsa.next_state)
10         else begin
11            dfsa.num_state = dfsa.num_state + 1;
12            dfsa.state[dfsa.num_state] = dfsa.next_state;
13            Store (dfsa.current_state, symbol, dfsa.num_state)
14         end //if
15      end //for
16      dfsa.current_state = dfsa.current_state + 1;
17   end //while
```

```
method ComputeDfsaNextState ()
1    begin
2       dfsa.next_state = ∅;
3       for each nfsa.state in dfsa.state[dfsa.current_state]
4       do
5          dfsa.next_state = dfsa.next_state ∪ nfsaNextState (nfsa.state, symbol);
6       return dfsa.next_state;
7    end //method
```

You must represent each FSA in the input file with the following format:

(1) The number of states, say N.

For ease of implementation, number states from 0 to N-1, with 0 representing the initial state, and N the dead-end state.

(2) The set of final states.

You need a boolean array FINAL [0..N-1].

(3) The alphabet.

Symbols in the alphabet should be numbered internally so that the value returned by the function getNextSymbol is an integer.

(4) A sequence of transitions of the form (p,a,q).

The triple (p,a,q) means that in state p, looking at input symbol a, the FSA will change its state to q. For this project, store this information in a table, say next_state, so that the value returned by the function getNextState is next_state [state, symbol].

For a nondeterministic machine, transitions will be of the form (p, a, q_1, q_2, …, q_n), which means that in state p, looking at input symbol a, the FSA will change its state to q_1, q_2, …, or q_n.

Test your program with the following 8 FSA and put all 8 machines in one input file.

(1) A DFSA which recognizes the set of strings over {a,b} such that each a in the string is immediately preceded and immediately followed by a b. **Strings** to be tested: a, b, bab, abb, bbaba, babaab, bbbababab, bbbbb

(2) A DFSA which recognizes the set of strings over {a,b} that do not have 3 consecutive b's. **Strings** to be tested: ε, bb, aaab, babbb, ababab, bbaabba, abababbba, bbb

(3) A DFSA which recognizes the set of strings over {0,1} with a 0 as the 4th from the last symbol. **Strings** to be tested: 11, 000, 10110, 00110, 01110101, 11001110, 010101010, 0000

(4) A DFSA which recognizes all identifiers that begin with a letter (both upper and lower) or an underscore, followed by any combination of letters, digits, and underscores. **Strings** to be tested: 1st_Assignment, FORTRAN, _finite_automaton, program, X3Y7, _, X=90, X*Y

(5) An NFSA with at least one nondeterministic state which recognizes the set of strings over {0,1} such that the third symbol from the right end is 1. **Strings** to be tested: 010100, 011, 0000111, 1111000, 0, 1, ε, 1010100

(6) An NFSA M = ({0,1,2,3,4}, {0,1}, δ, 0, {2,4}) where δ is defined as follows. **Strings** to be tested: ε, 0101, 10110, 1010010, 011000, 00, 1, 111

	0	1
0	0,3	0,1
1	-	2
2	2	2
3	4	-
4	4	4

(7) An NFSA M = ({0,1,2,3,4}, {a,b,c}, δ, 0, {4}) where δ is defined as follows. **Strings** to be tested: a, ba, ac, abc, cabca, acbac, bacbc, aabbcc

	a	b	c
0	0,1	0,2	-
1	-	3	2
2	3,4	2	-
3	-	-	0,4
4	-	-	0

(8) An NFSA with at least one nondeterministic state which recognizes the set of strings over {0,1} such that the fifth symbol from the right end is 0. **Strings** to be tested: ε, 1010, 0000111, 11111000, 01110, 0101010, 000, 1

Sample input of a DFSA:

```
2
1
0 1
(0 0 0)
(0 1 1)
(1 0 1)
(1 1 0)
1000
10001
……..
```

Corresponding output:

```
Finite State Automaton #1.
1) number of states: 2
2) final states: 1
3) alphabet: 0, 1
4) transitions:
        0   0   0
        0   1   1
        1   0   1
        1   1   0
5) strings:
        1000            Accept
        10001           Reject
        ……..
```

Sample input of a NFSA:

```
4
3
0 1
(0  0  0)
(0  1  0  1)
(1  0  2)
(1  1  2)
(2  0  3)
(2  1  3)
010100
011
……..
```

Corresponding output:

Finite State Automaton #5.

1) number of states: 4
2) final states: 3
3) alphabet: 0, 1
4) transitions:

```
0  0  0
0  1  0  1
1  0  2
1  1  2
2  0  3
2  1  3
```

The equivalent DFSA by subset construction:

1) number of states: 8

```
state 0:  {0}
state 1:  {0  1}
state 2:  {0  2}
state 3:  {0  1  2}
state 4:  {0  3}
state 5:  {0  1  3}
state 6:  {0  2  3}
state 7:  {0  1  2  3}
```

2) final states: 4 5 6 7
3) transitions:

```
state 0:  0  0,  1  1
state 1:  0  2,  1  3
state 2:  0  4,  1  5
state 3:  0  6,  1  7
state 4:  0  0,  1  1
state 5:  0  2,  1  3
state 6:  0  4,  1  5
state 7:  0  6,  1  7
```

4) strings:

010100	Accept
011	Reject
……..	

Submit on Bb and turn in the printout of (1) program code with proper comments including your name and clear instructions to compile, link, and run your program, (2) one input file containing the description of the above 8 machines and corresponding test strings, and (3) output of executions.

Grading:
84% – correctness
16% – program structure and readability, required input/output format

Your project must be your own work. Copying programs or parts of programs will receive a zero grade and will be reported to the CS department and the University.

Absolutely no late project will be accepted for grading. If you cannot complete the project by the due date, then submit whatever you have completed. Partial credit will be given for reasonable partial solutions.