Homework #2: CMPT-413

Distributed on Mon, Jan 19; Due on Mon, Jan 26

Anoop Sarkar – anoop@cs.sfu.ca

(1) Submit file name: `fsm-recognize.pl`

Download the Perl program `fsm-nogen.pl`. This file contains an implementation of the algorithm `NDRecognize` from Jurafsky and Martin (see Figure 2.21 on page 44) minus the function `generateNewStates`. You have to add that function.

The program takes as input finite-state machine (FSM) described as the following Perl data (available as the file `fsm-ex1.pl` from the course web page):

```
$startstate = 1;
$finalstate[3] = 'true';
$edges[1] = [2];
$edges[2] = [2,3];
$input[1][2] = ['a'];
$input[2][2] = ['a','b'];
$input[2][3] = ['b'];
```

This FSM corresponds to the usual graphical representation shown in Figure 1. Note that the `transition-table` data structure used in the Jurafsky and Martin textbook has been replaced here with two arrays: `@edges` provides a list of possible transitions from a state, and `@input` provides the possible inputs to match from the tape on each transition from one state to another.

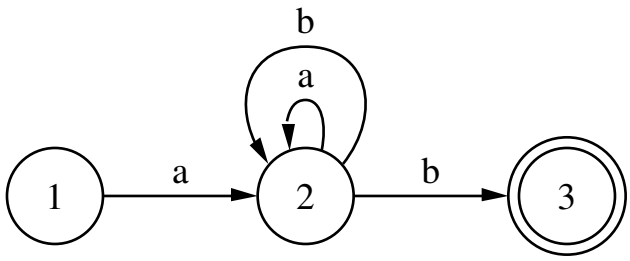

Figure 1: An example finite-state machine for Question 1.

a. Copy `fsm-nogen.pl` to a new file `fsm-recognize.pl` and add the function `generateNewStates` based on the description given in the textbook. Once you add this function you should be able to run the following command:

`$ perl fsm-recognize.pl fsm-ex1.pl` ⟨*cr*⟩

Note that ⟨*cr*⟩ is the *carriage return* or *enter* key. It takes as input any string from standard input. Enter a string and then press ⟨*cr*⟩. If the string is accepted the algorithm will display a trace of its execution and say `yes` if the string is accepted by the FSM or `no` otherwise. Given below is a sample trace of the execution of `fsm-recognize.pl` (when the missing function `generateNewStates` has been added) for the input string `abab`.

```
$ echo "abab" | perl fsm-recognize.pl fsm-ex1.pl
currentItem=1, index=-1 tape=abab
agenda=
currentItem=2, index=0 tape=bab
agenda=
currentItem=2, index=1 tape=ab
agenda=3 1
currentItem=3, index=1 tape=ab
agenda=2 2
currentItem=2, index=2 tape=b
agenda=
currentItem=2, index=3 tape=
agenda=3 3
currentItem=3, index=3 tape=
accept state=3
yes
```

b. Modify `fsm-recognize.pl` so that it can handle transitions on the empty string ” and transitions on strings with more than one character (e.g. a transition on the string ’ab’) . Use the FSMs in `fsm-ex2.pl` and `fsm-ex3.pl` to test your implementation.

(2) Submit file name: `fst-recognize.pl`

Consider the following representation in Perl of a finite state transducer:

```
$startstate = 1;
$finalstate[3] = 'true';
$edges[1] = [2];
$edges[2] = [2,3];
$input[1][2] = ['a'];     $output[1][2] = ['0'];
$input[2][2] = ['a','b']; $output[2][2] = ['0','1'];
$input[2][3] = ['b'];     $output[2][3] = ['1'];
```

It is a slight extension of the representation used for finite state automata. The only addition is the `@output` array which is analogous to the `@input` array. The above Perl data represents the finite state transducer shown in Figure 2.

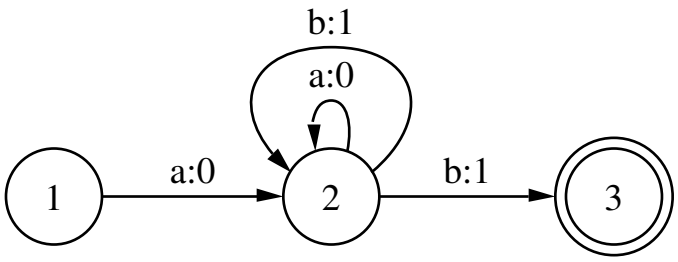

Figure 2: An example finite-state transducer for Question 2.

Just as finite state automata (FSA) recognize strings, a finite state transducer (FST) can be viewed as a recognizer over *pairs* of strings. For example, the pair of strings (*abab*, 0101) is accepted by the above FST. This means we can extend the algorithm `NDRecognize` for FSA to an algorithm called `fstRecognize` which can be used to decide whether a *string pair* is accepted by a given FST.

The process of accepting a pair of strings will be represented in Perl with the use of two 'tapes'. An input tape which will be a reference to a list of elements from the input alphabet of the FST, and an output tape which is a reference to a list of elements from the output alphabet of the FST. A Perl implementation of the algorithm `fstRecognize` uses the definition of the FST as Perl data provided above. It takes the input tape and the output tape as parameters and returns `1` if the pair of strings is accepted by the FST and `0` otherwise.

For example,

`$inputTape = ['a', 'b', 'a', 'b'];`
`$outputTape = ['0', '1', '0', '1'];`
`fstRecognize($inputTape, $outputTape)` returns `1`

Just as in the previous question, download the program `fst-nogen.pl`. This file contains the Perl code for `fstRecognize` which uses two additional functions: `acceptState` and `generateNewStates`. You are also given the definition of `acceptState`.

Copy the file `fst-nogen.pl` to the file `fst-recognize.pl`. Modify this file to provide Perl code for the function `generateNewStates` that completes the implementation of FST recognition.

`fst-recognize.pl` takes as input the input and the output string as two lines from standard input. Enter the input string, press ⟨*cr*⟩ then enter the output string, press ⟨*cr*⟩. If the pair of strings is accepted by the FST the program will output `yes` or `no` otherwise.

Use the FSTs in `fst-ex1.pl`, `fst-ex2.pl`, `fst-ex3.pl`, `fst-ex4.pl` to test your implementation. Here are some examples of how a successful implementation should run:

```
$ perl fst-recognize.pl fst-ex1.pl
abb
011
input=abb output=011
yes

$ perl fst-recognize.pl fst-ex3.pl
ab
0
input=ab output=0
yes

$ perl fst-recognize.pl fst-ex4.pl
mouse+N+PL
mouse
input=mouse+N+PL output=mouse
no

$ perl fst-recognize.pl fst-ex4.pl
mouse+N+PL
mice
input=mouse+N+PL output=mice
yes
```