CSci 555-1: Functional Programming
Spring 2016

Assignment #3: Mealy Machine Simulator
Revised Deadline Friday, 22 April, 11:59 p.m.
(Five extra points if completed by original deadline of 19 April)

Note: This document will only display correctly in a browser that supports MathML.

Introduction

In this assignment, you are to design and implement an abstract data type to simulate a Mealy Machine. This kind of machine is a useful abstraction for simple controllers that listen for input events and respond by generating output events. For example in an automobile application, the input might be an event such as "fuel level low" and the output might be command to "display low-fuel warning message".

In the theory of computation (e.g., the general topic of the UM course CSCI 311, Models of Computation), a *Mealy Machine* is a finite-state automaton whose output values are determined both by its current state and the current input. It is a *deterministic finite state transducer* such that, for each state and input, at most one transition is possible.

Appendix A of the current CSci 311 textbook (Peter Linz, *Formal Languages and Automata*, 5th Ed., Jones & Bartlett, 2012) defines a Mealy Machine mathematically by a tuple

$$M = (Q, \Sigma, \Gamma, \delta, \theta, q_0)$$

where

Q is a finite set of internal states
Σ is the input alphabet (a finite set of values)
Γ is the output alphabet (a finite set of values)
$\delta : Q \times \Sigma \longrightarrow Q$ is the transition function
$\theta : Q \times \Sigma \longrightarrow \Gamma$ is the output function
q_0 is the initial state of M (an element of Q)

In an alternative formulation, the transition and output functions can be combined into a single function:

$$\delta : Q \times \Sigma \longrightarrow Q \times \Gamma$$

We often find it useful to picture a finite machine as a *transition graph* where the states are mapped to vertices and the transition function represented by directed edges between vertices labelled with the input and output symbols.

Tasks

Design and implement a Mealy Machine simulator as an immutable Scala “module” using functional programming techniques.

You may, but are not required to, use the doubly labelled Digraph abstract data type (presented in class) internally to represent the transition graph.

Define an appropriate interface to the module. The Mealy Machine abstract data type should have, at least, public operations to

- create a new machine (perhaps one with no states and no transitions)
- add a new state to the machine
- add a new transition to the machine
- add all reset transitions (that is, make the transition function a total function by adding any missing transitions from a state back to the initial state)
- set the “initial” state of the machine
- get the current state of the machine
- get a collection of all states of the machine
- get a collection of all inputs enabled for a given state
- move the machine forward one step given an input; the machine should generate the appropriate output and enable the next move from the destination state of this move (Note that, in a purely functional program, the “move” function probably needs to return two items–the next state and the output from the current move.)
- do other needed tasks (if any) to complete the abstraction or to make your implementation work conveniently with Scala

Define a function (either as a part of the abstract data type or that uses it) to execute the machine on a sequence of inputs, returning the corresponding sequence of outputs and enabling the machine to be continued with another sequence if needed.

Alternatively, you might want to split the above into two parts–a builder abstract data type that enables a machine to be defined and a separate abstract data type that executes a machine.

Remember, your design and implementation should avoid mutable state and imperative Scala code.

Design appropriate tests for your functions and test them thoroughly.

Document your program appropriately.

Submit the source code and documentation for your program and test driver, any needed instructions on how to run the program, and appropriate test output to Blackboard. Be sure to identify yourself in the materials turned in.