CS 496: Assignment 1
Due: 5 February, 11:55pm

1 Assignment Policies

Collaboration Policy. Homework will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems or programming. Use of the Internet is allowed, but should not include searching for existing solutions.

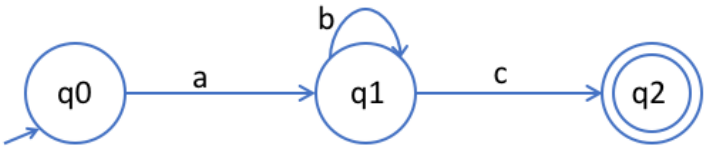
Under absolutely no circumstances code can be exchanged between students. Excerpts of code presented in class can be used.

Assignments from previous offerings of the course must not be re-used. Violations will be penalized appropriately.

2 Assignment

This assignment is about coding with lists, tuples and records. The setting is that of Finite Automata. We'll give a brief introduction below.

A *finite automaton* (FA) is a simple machine that recognizes sequences of symbols from a given alphabet. For example, suppose we have an alphabet of three symbols: 'a', 'b' and 'c'. Below is a pictorial representation of a FA that recognizes all the sequences of symbols of that alphabet that start with an 'a', have zero or more 'b's in the middle, and then end with a 'c', as follows:



The designated *start state* is q_0; this is indicated with a short arrow. The state q_2 is a *final state* and is indicated with two concentric circles. A more precise definition of Finite Automata follows. A *finite automaton* is a tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of states;

- Σ is a finite set of input symbols called the alphabet;
- $\delta : Q \times \Sigma \to Q$ is a transition function;
- $q_0 \in Q$ is an initial or start state; and
- $F \subseteq Q$ is a set of accept states.

In our example, $Q = \{q_0, q_1, q_2\}$. The set of final states is $F = \{q_2\}$.

How an automaton operates. Given an input sequence of symbols, the automaton attempts to recognize it. Initially the automaton is in the start state and an input pointer points to the first symbol in the input sequence. It then follows the transition function (*cf.* the "arrows" in the figure above), changing states according to the next symbol in the sequence to be processes. For example, the arrow from q0 to q1 labeled "a" indicates that our automaton, if it is in state q0 and the next symbol in the input is an "a", then it "moves" to state q1 and passes on to process the next symbol in the input.

3 Simulating Automata

3.1 Encoding Automata

We will use the following user defined datatypes.

```
type symbol = char
type input = char list

type state = string

(* transition function *)
type tf = (state * symbol * state) list

(* start state * transition function * end state *)
type fa = { states: state list; start:state; tf: tf; final: state list}
```

Below is how we encode the automaton from the figure above. We give it a name, `a`, so that can refer to it later. Note that `a` is a record and its type is `fa`.

```
let a = {states = ["q0";"q1";"q2"];
         start = "q0";
         tf = [("q0",'a',"q1"); ("q1",'b',"q1"); ("q1",'c',"q2")];
         final = ["q2"]}
```

3.2 Your Task

Implement the following functions. You should work on the file `fa_stub.ml`. First rename it to `fa.ml` and then start completing the exercises given below. Unless otherwise stated, you may assume that any function that has a FA as input, that FA is valid (in the sense explained below).

1. `apply_transition_function : tf -> state -> symbol -> state option`

   The expression `apply_transition_function f st sym` applies the transition function `f` to the symbol `sym` assuming that the current state is `st`. For example, if `f` is the transition function of `a`, then:

```
# apply_transition_function f "q0" 'a';;
- : state option = Some "q1"
# apply_transition_function f "q0" 'b';;
- : state option = None
# apply_transition_function f "q0" 'c';;
- : state option = None
# apply_transition_function f "this_state_does_not_exist" 'c';;
- : state option = None
```

For this exercise, you may assume the FA is deterministic.

2. `accept : fa -> input -> bool`

   Determine whether a word is accepted by a finite automaton. Here are some examples:

```
# accept a (input_of_string "abbc");;
- : bool = true
# accept a (input_of_string "ac");;
- : bool = true
# accept a (input_of_string "a");;
- : bool = false
# accept a (input_of_string "bb");;
- : bool = false
```

   The `input_of_string` function is just a helper function that translates strings to lists of symbols. It is provided in the stub so you need not implement it.

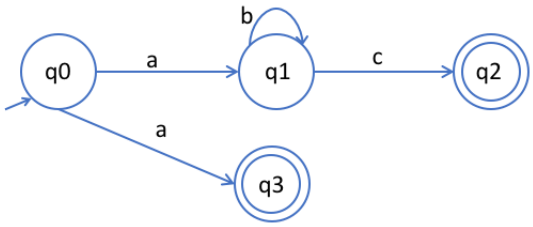
3. `next : tf -> state -> symbol -> state list`

   This functions returns the list of all the states that are successors of some given state with some give symbol. For example,

```
# next a.tf "q1" 'c';;
- : state list = ["q2"]
# next a.tf "q1" 'b';;
- : state list = ["q1"]
# next a.tf "q1" 'a';;
- : state list = []
```

4. `deterministic : fa -> bool`

   This function checks whether the given automaton is non-deterministic or not. A *non-deterministic FA* is one in which there is a state that has two or more successor states with the same symbol. Our first example introduced above is deterministic. The following one is non-deterministic.



5. `valid : fa -> bool`

   Implement `valid` that checks for validity. A FA is said to be *valid* if

   (a) The list of states has no duplicates;

   (b) The start state belongs to set of states;

   (c) The final states belong to set of states; and

   (d) It is deterministic.

6. `reachable : fa -> state list`

   Reports list of states that are reachable from the start state. Note that the start state is stored in the `start` field of the FA. Here is an example:

```
# reachable a;;
- : state list = ["q0"; "q1"; "q2"]
```

7. `non_empty : fa -> bool`

   Determines whether a FA accepts at least one word. Hint: make sure that at least one final state is reachable from the start state.

8. `remove_dead_states : fa -> fa`

   Removes all dead (i.e. unreachable) states from a valid FA. This includes removing them from the set of states, removing the transitions in δ that involve dead states and also removing them from the set of final states.

4 Submission instructions

Submit a single file named `hw1.ml` through Canvas. No report is required. Your grade will be determined as follows:

- You will get 0 points if your code does not compile.
- Partial credit may be given for style, comments and readability.