

# CS 496: Assignment 1

Due: February 21, 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.

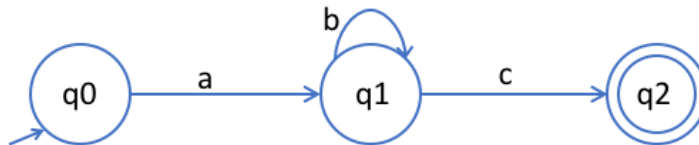
**Under absolutely no circumstances code can be exchanged between students.** Excerpts of code presented in class can be used.

**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 (given that you have already taken CS334) introduction below.

A *finite automaton* (FA) is a simple machine for recognizing sequences of symbols that follow a certain pattern. For example, suppose we have an alphabet of three symbols: 'a', 'b' and 'c'. We can picture an automaton that recognizes all the strings 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 fo



An automaton is a tuple  $(Q, \Sigma, \delta, q_0, F)$  where:

- $Q$  is a finite set of states;
- $\Sigma$  is a finite set of input symbols called the alphabet;
- $\delta : Q \times \Sigma \rightarrow Q$  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 designated start state (indicated with a short arrow) is  $q_0$ . The set of final states is  $F = \{q_2\}$ . These are indicated by writing two concentric circles.

**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 processed. For example, the arrow from  $q_0$  to  $q_1$  labeled “a” indicates that our automaton, if it is in state  $q_0$  and the next symbol in the input is an “a”, then it can move to state  $q_1$  and pass on to process the next symbol in the input.

## 3 Simulating Automata

### 3.1 Encoding Automata

We will use the following user defined datatypes.

```

1 type symbol = char
2 type input = char list
3
4 type state = string
5
6 (* transition function *)
7 type tf = (state * symbol * state) list
8
9 (* start state * transition function * end state *)
10 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`.

```

1 let a = {states = ["q0"; "q1"; "q2"];
2           start = "q0";
3           tf = [("q0", 'a', "q1"); ("q1", 'b', "q1"); ("q1", 'c', "q2")];
4           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.

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:

```

1 # apply_transition_function f "q0" 'a';;
2 - : state option = Some "q1"
3 utop # apply_transition_function f "q0" 'b';;
4 - : state option = None
5 utop # apply_transition_function f "q0" 'c';;
6 - : 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:

```

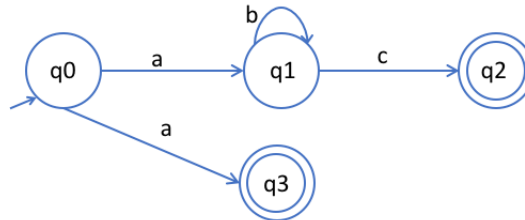
1 utop # accept a @@ input_of_string "abbc";;
2 - : bool = true
3 utop # accept a @@ input_of_string "ac";;
4 - : bool = true
5 utop # accept a @@ input_of_string "a";;
6 - : bool = false
7 utop # accept a @@ input_of_string "bb";;
8 - : bool = false

```

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. `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 is deterministic. The following one is non-deterministic.



Hint: first implement a function `next : tf -> state -> symbol -> state list` that returns the list of all the states that are successors of some given state with some given symbol.

4. `valid : fa -> bool`

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

- (a) The start state belongs to set of states;
- (b) The final states belong to set of states; and
- (c) It is deterministic.

5. `reachable : fa -> state list`

Reports list of states that are reachable from the start state.

6. `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  $\delta$  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.