Concepts of Programming Languages, CSCI 305, Fall 2021
Lab 3, Traversing a DFA using Scheme, Sept. 10

The text shows (page 550) a DFA (Deterministic Finite Automaton) which recognizes all strings of 0s and 1s with an even number of 0s and an even number of 1s, possibly interleaved. The list encoding of for this DFA is in Figure 11.2, page 550. The list contains three elements:

1. Start state/current state
2. Transition function that is a list of pairs. The first element in each pair is a pair itself. The interpretation of ((q3 1) q2) is that if the DFA is in state q3 and a 1 is read, the DFA goes into state q2.
3. List of final states

1. Define the constant, *evenDFA* that encodes the start state, the transitions for the DFA that accepts all strings with an even number of 0's and an even number of 1's, and a list of the final states for this DFA.
2. Define the following helper functions:
   a. *current-state* – takes an encoded DFA and extracts the current state from it
   b. *transition-function* – takes an encoded DFA and extracts the transition function from it
   c. *final-states* – takes an encoded DFA and extracts the list of final states from it
   d. *infinal?* – takes an encoded DFA and sees if the current state is in the list of final states. This function can return #t and #f, or the current state and #f (the current state would be seen as #t.
   e. *move* – takes an encoded DFA and a symbol and returns the encoded DFA where the current state has changed from where it was to where it would be after following the symbol transition.
3. Now that the helper functions are completed, create a driver for traversing the DFA. This can be called *simulate* and takes an encoded DFA (starting at the start state) and a list of 0's and 1's representing the input string. The function *simulate* returns the list of states traversed, beginning with the start state and ending with either accept or reject. For example,

   (simulate evenDFA `(0 1 0 1 0 0))

   returns

   '(q0 q2 q3 q1 q0 q2 q0 accept)

   If the function encounters an error, it can return the string 'error for each of the remaining input symbols. For example,

   ```
   (simulate evenDFA '(0 0 3 0 0 0) `())
   ```

   returns

   ```
   (q0 q2 q0 error error error error reject)
   ```

Run the following simulations to verify that you get the output shown.

```
(simulate evenDFA `())   => '(q0 accept)

(simulate evenDFA `(0))   => '(q0 q2 reject)

(simulate evenDFA `(0 1 0 1 0 0)) => '(q0 q2 q3 q1 q0 q2 q0 accept)

(simulate evenDFA `(0 0 3 0 0 0))
              => '(q0 q2 q0 error error error error reject)
```

```
;Create the data structure and let the current state be the start state.
(define evenDFA
  '(q0            ; current_state
    (             ; transition function
     ((q0 0) q1) ((q0 1) q2) ((q1 0) q0) ((q1 1) q3)
     ((q2 0) q3) ((q2 1) q0) ((q3 0) q2) ((q3 1) q1)
    )
    (q0)          ; set of final states
    )
  )

; Define a procedure current-state that takes a dfa
; and returns the current state from that dfa.
(define current-state
  (lambda (dfa)
    (car dfa)))

; Define a procedure transition-function that takes
; a dfa and returns the transition function of that
; dfa.
(define transition-function
  (lambda (dfa)
    (cadr dfa)))

; Define a procedure final-states that takes
; a dfa and returns the final states of that
; dfa.
(define final-states
  (lambda (dfa)
    (caddr dfa)))


; Define a procedure final? that takes
; a dfa and returns #f if the current state
; is not in the list of final states, and something
; non-false, if it is.
(define final?
  (lambda (dfa)
    (member (current-state dfa)
            (final-states dfa))))
```

```scheme
; Define a procedure move that takes
; a dfa and a symbol, and returns the
; encoded DFA where the state has been
; updated.
; Define a procedure move that takes
; a dfa and a symbol, and returns the
; encoded DFA where the state has been
; updated.
(define move
  (lambda (dfa symbol)
    (let* (
          (cs    (current-state dfa))       ; current state
          (trans (transition-function dfa)) ; transitions
          (tran  (assoc (list cs symbol) trans)); transition or #f
          )
      (list
          (if (equal? tran #f) 'error (cadr tran))  ;new state or error
               trans                                ; transition function
              (final-states dfa)                    ; final states
           ) ; list
      ) ; let
  )
)



; Define the simulate procedure that takes
; a dfa and a list of symbols, representing
; the string being read. It returns a list
; with each of the states traversed by the
; dfa when reading the symbols, followed
; by accept or reject.
(define simulate
  (lambda (dfa input)
    (if (null? input)
        (if (final? dfa)     ; Base case is when the input string is empty.
            (list (current-state dfa) 'accept)
            (list (current-state dfa) 'reject))
        (cons (current-state dfa)
              (simulate (move dfa (car input)) (cdr input))
         )  ; Insert the state into the list created by the recursive call.
        ) ; if
    ) ; lambda
  ) ; define



; test cases
(simulate evenDFA `())
;        => (q0 accept)

(simulate evenDFA `(0))
;        => (q0 q2 reject)


(simulate evenDFA `(0 1 0 1 0 0))
;        => (q0 q2 q3 q1 q0 q2 q0 accept)

(simulate evenDFA `(0 0 3 0 0 0))
;        => (q0 q2 q0 error error error error reject)
```