For this assignment, you will build a Scheme interpreter that will process strings as directed by a nondeterministic finite automaton. The work is divided into two parts.

1. The NFA will be represented as a list of lists. The first interior list will contain the accepting states, named with integers. For each state, there may be a subsequent list giving the available transitions from that state. Each of these transitions will itself be a list containing the source state as the first element and lists of destinations associated with a particular input string symbol. The special symbol ‘eps’ denotes a $\epsilon$-transition. Here are two examples.

   The NFA below has the representation ‘((4) (1 (a 1 2) (b 1) (c 1)) (2 (b 3)) (3 (c 4))) and accepts strings ending in abc. That is, it accepts strings $(a + b + c)^*abc$.

   \[\text{The transition diagram appears to the left. The accepting states are (4). The transitions from state 1 are (1 (a 1 2) (b 1) (c 1)). In particular, the transitions from state 1 available on an 'a' are 1 and 2; on a 'b' or a 'c', only destination 2 is available. The transitions for state 2 are (2 (b 3)), and so forth. Note that state 4 has no transitions and is simply missing from the list. I found it convenient to construct the machine in stages:}\]

   \[
   \begin{align*}
   \text{(define final '(4))} \\
   \text{(define trans1a '(a 1 2))} \\
   \text{(define trans1b '(b 1))} \\
   \text{(define trans1c '(c 1))} \\
   \text{(define trans2b '(b 3))} \\
   \text{(define trans3c '(c 4))} \\
   \text{(define trans1 (list 1 trans1a trans1b trans1c))} \\
   \text{(define trans2 (list 2 trans2b))} \\
   \text{(define trans3 (list 3 trans3c))} \\
   \text{(define term-abc (list final trans1 trans2 trans3))}
   \end{align*}
   \]

   Machine term-abc is then ‘((4) (1 (a 1 2) (b 1) (c 1)) (2 (b 3)) (3 (c 4))).

   \[
   \begin{align*}
   \text{A second example involves $\epsilon$-transitions, indicated by eps in the definition. Its diagram appears to the left, with the single accepting state 3, from which it is clear that the machine accepts } ab + a^*a. \text{ An incremental definition is} \\
   \begin{align*}
   \text{(define tt0a '(a 1))} \\
   \text{(define tt0eps '(eps 2))} \\
   \text{(define tt1b '(b 3))} \\
   \text{(define tt2a '(a 2 3))} \\
   \text{(define f2 '(3))} \\
   \text{(define mach2 (list f2 (list 0 tt0a tt0eps) (list 1 tt1b) (list 2 tt2a))})
   \end{align*}
   \]

   giving the complete definition ‘((3) (0 (a 1) (eps 2)) (1 (b 3)) (2 (a 2 3))).

The first part of this assignment is to write a Scheme function transitions that takes three arguments: a state, a symbol (or ‘eps), and a machine. It returns a list of possible destination states reachable from the argument state via the argument symbol. For example, the example machines described above deliver the following destinations.

(a) (transitions 1 'a term-abc) returns '(1 2)
(b) (transitions 2 'b term-abc) returns '(3)
(c) (transitions 2 'a term-abc) returns '()
(d) (transitions 3 'eps term-abc) returns '()
(e) (transitions 0 'eps mach2) returns '2)
2. Notice that the machine structure in the preceding part does \textit{not} contain a start state. For purposes of recursion, it is convenient to start the machine from various states, even though an accepting path from the true start state is necessary to accept a string.

For the second part of the assignment, you are to construct two mutually recursive functions: \texttt{nfa-execute} and \texttt{backtracker}. The first takes three arguments: a string (a simple list of atomic literals), a start state, and a machine. The machine is a structure as defined in the preceding question, and the start state must be one of its states. \texttt{nfa-execute} attempts to find an accepting path from its given start state to an accepting state. If it can find such a path, it returns a list of nodes along that path. If no such path exists, it returns the null list. For example,

\begin{verbatim}
(nfa-execute '(b b a a c b a c b a b c), 1, term-abc) returns '(1 1 1 1 1 1 1 1 2 3 4).
(nfa-execute '(b b a a c b a c b a b b), 1, term-abc) returns '().
\end{verbatim}

In these examples, we call \texttt{nfa-execute} with the true start state, and therefore we can interpret the result as a yes/no verdict on the membership of the given string in the language represented by the machine. However, \texttt{nfa-execute} can seek an accepting path from any node, and this flexibility allows us to recursively search for an accepting path.

\texttt{nfa-execute} achieves its goal by using the second function, \texttt{backtracker}, which in turn can call \texttt{nfa-execute}. Five arguments are necessary for a \texttt{backtracker} call: a string, a start state, a list of available destinations that can be reached by consuming the first string symbol, a list of available destinations that can be reached by a $\epsilon$-transition, and finally a machine. To advance, \texttt{nfa-execute} extracts the destination lists with the \texttt{transition} function of the preceding problem and then call the backtracker.

Suppose the call is \texttt{(backtracker string start trans epstrans machine)}. There are some initial checks. If \texttt{string} is null and \texttt{start} is an accepting state, then the backtracker return a singleton list containing \texttt{start}. If the string is not null and destinations are available in \texttt{trans}, the backtracker attempts to continue the path via \texttt{nfa-execute} on \texttt{(cdr string)}, \texttt{(car trans)}. If that fails, it tries a recursive call to itself using \texttt{string}, \texttt{start}, \texttt{(cdr trans)}, \texttt{epstrans}, \texttt{machine}. If \texttt{trans} is null, it proceeds in a similar fashion with \texttt{epstrans}, using the full string, of course, since \texttt{epstrans} contains destination states reachable via a $\epsilon$-transition. Backtracker exhausts all available possibilities from \texttt{trans} before trying those from \texttt{epstrans}. If none of these alternatives succeed, it returns the null list.

Here are a few more examples using the machines from the preceding question.

\begin{verbatim}
> (nfa-execute '(b b a a c b a c b a b c) 1 term-abc)
'(1 1 1 1 1 1 1 1 2 3 4)
> (nfa-execute '(b b a a c b a c b a b b) 1, term-abc) returns '().
> (nfa-execute '(a a a a a a a a a) 0 mach2)
'(0 2 2 2 2 2 2 2 3)
> (nfa-execute '(a b) 0 mach2)
'(0 1 3)
> (nfa-execute '(a b a a a b) 0 mach2)
'(0 2 3)
> (nfa-execute '(b a) 0 mach2)
'(0 2 3)
\end{verbatim}

You may assume that there are no cycles in the machine in which all links are epsilon-transitions.