Exercise 5.1 — Haskell Evaluation  (36% of Midterm 1, 2004)
Assume the following Haskell definitions to be given:

\[
succ \ n = n + 1
\]

\[
take \ :: \ \text{Int} \to \ \text{[a]} \to \ \text{[a]}
\]

\[
take \ 0 \ _ = [ ]
\]

\[
take \ _ \ [ ] = [ ]
\]

\[
take \ n \ (x:xs) = x : take \ (n-1) \ xs
\]

\[
\text{feed} \ h \ q \ y = q : \text{feed} \ h \ (q + y) \ (h \ y)
\]

Simulate Haskell evaluation for the following expression (write down the sequence of intermediate expressions):

\[
\text{take 3 (feed succ 0 1)}
\]

Note:  You may introduce abbreviations for repeated subexpressions, or use repetition marks for material that is unchanged from the previous line. In particular, write “s” instead of “succ”!

Exercise 5.2 — Finite-State Machines  (25% of Midterm 1, 2004)
Let the following type synonyms be given, as in the presentation in the first lecture:

\[
\text{type State} = \text{Int}
\]

\[
\text{type Symbol} = \text{Char}
\]

\[
\text{type TransRel} = \{ (\text{State}, \text{Symbol}, \text{State}) \}
\]

\[
\text{type FSM} = (\text{State}, \text{TransRel}, \text{[State]})
\]

(a) Define \( fsm1 :: \text{FSM} \) such that it represents the finite-state machine drawn above (with start state circled and end states in boxes):

(b) Define the Haskell function \( \text{isDet} :: \text{FSM} \to \text{Bool} \) such that \( \text{isDet} \ fsm \) evaluates to the Boolean value indicating whether the finite-state machine \( fsm \) is deterministic or not.

For example, \( \text{isDet} \ fsm1 = \text{False} \) since there are two \( b \)-edges from state 1 to different nodes.

**Hint:** Define auxiliary functions! For example:
- Calculate all start nodes of transitions in a \( \text{TransRel} \).
- Given a state, calculate all edges leaving that state in a \( \text{TransRel} \).
- Given a \( \text{Symbol} \) and a \( \text{TransRel} \), find all target nodes of edges with that symbol.
- Given a \( \text{State} \) and a \( \text{TransRel} \), find out whether any edges from that state violate determinacy.

Other functions may be useful, too. **Document your functions!**
Exercise 5.3 — Haskell Typing  (19% of Midterm 1, 2004)
Provide detailed derivations of the Haskell types of the following functions:
\[ \text{swibble } x \ y = [ ( x \ , \ y ) \ , \ ( x \ \_\_\_"\" \ , \ y + 1 ) ] \]
\[ \text{swoon } g \ h = [ g \ ( (1 + ) \ . \ h ) ] \]

Exercise 5.4  (Skeleton file is on the course page)
We define a type of transition functions that define state transitions triggered by inputs and also producing outputs:
\[
\text{type Transition state input output } = (\text{state, input}) \to (\text{state, output})
\]
(a) Define a Haskell function
\[
\text{process :: Transition state input output } \to \text{state } \to [\text{input}] \to [\text{output}]
\]
that calculates the list of outputs produced by a transition function given a starting state and a list of inputs.

Using \text{process} from (b) and prelude functions, the definition
\[
\text{runprocess :: Transition state String String } \to \text{state } \to \text{IO ()}
\]
\[
\text{runprocess tr s = do}
\]
\[
\text{hSetBuffering stdout LineBuffering} -- \text{requires: “import System.IO” at beginning of module}
\]
\[
\text{interact (unlines } \circ \text{process tr s } \circ \text{lines)}
\]
allows \text{runprocess} to turn a transition with \text{String} inputs and outputs into a runnable program.
Try: \text{runprocess id 0}

(b) Define a transition function
\[
\text{countEcho :: Transition Integer String String}
\]
that keeps a counter as its state and otherwise just reproduces the input prefixed with line numbers as output.
Try: \text{runprocess countEcho 0}

(c) Define a transition function
\[
\text{trAdd :: Transition Integer String String}
\]
that uses the prelude functions \text{read} and \text{show} to add the \text{Integer} reading of the input to the accumulating state, and outputs that state as a string.
Try: \text{runprocess trAdd 0}

(d) Define a transition function
\[
\text{polish :: Transition [ Integer] String String}
\]
that implements a reverse Polish notation calculator by pushing number inputs on the stack, always outputing the top of the stack (if present), and interpreting \(+, -, *, /\) as taking their arguments from the stack and pushing the result back onto the stack.
Try: \text{runprocess polish []}