McMaster University
Department of Computing and Software
Dr. W. Kahl

SFWR ENG 3E03
Exercise Sheet 5

Design and Selection of Programming Languages

11 October 2006

Exercise 5.1 — Haskell Evaluation (36% of Midterm 1, 2004)

Assume the following Haskell definitions to be given:

```
succ n = n+1                      -- reduce in one step, e.g.: succ 5 → 6

take :: Int -> [a] -> [a]
take 0 _      =  []
take _ []     =  []
take n (x:xs) =  x : take (n-1) xs

feed h q y =  q : feed h (q + y) (h y)
```

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

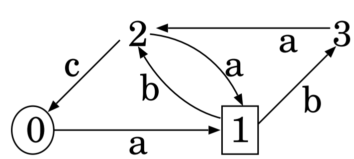
```
take 3 (feed succ 0 1)
```

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:

type *State* = *Int*
type *Symbol* = *Char*
type *TransRel* = [(*State* , *Symbol* , *State*)]
type *FSM* = (*State* , *TransRel* , [*State*])



(a) Define *fsm1* :: *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 *isDet* :: *FSM* → *Bool* such that *isDet fsm* evaluates to the Boolean value indicating whether the finite-state machine *fsm* is deterministic or not.

For example, *isDet fsm1* = **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 *TransRel*.
- Given a state, calculate all edges leaving that state in a *TransRel*.
- Given a *Symbol* and a *TransRel*, find all target nodes of edges with that symbol.
- Given a *State* and a *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:

```
swibble x y = [ ( x , y ) , ( x ++ "'", y + 1 ) ]
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 *input*s and also producing *output*s:

type *Transition state input output* = (*state* , *input*) → (*state* , *output*)

(a) Define a Haskell function

process :: *Transition state input output* → *state* → [*input*] → [*output*]

that calculates the list of outputs produced by a transition function given a starting state and a list of inputs.

Using *process* from (b) and prelude functions, the definition

runprocess :: *Transition state String String* → *state* → *IO* ( )
runprocess tr s = **do**
  hSetBuffering stdout LineBuffering −− requires: "import System.IO" at beginning of module
  interact (*unlines* ∘ *process tr s* ∘ *lines*)

allows *runprocess* to turn a transition with *String* inputs and outputs into a runnable program.

Try: *runprocess id* 0

(b) Define a transition function

countEcho :: *Transition Integer String String*

that keeps a counter as its state and otherwise just reproduces the input prefixed withline numbers as output.

Try: *runprocess countEcho* 0

(c) Define a transition function

trAdd :: *Transition Integer String String*

that uses the prelude functions *read* and *show* to add the *Integer* reading of the input to the accumulating state, and outputs that state as a string.

Try: *runprocess trAdd* 0

(d) Define a transition function

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: *runprocess polish* [ ]