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

**SFWR ENG 3E03**
**Exercise Sheet 5**

# Design and Selection of Programming Languages

12 October 2005

**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 — 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 5.3 — Defining Haskell Functions (20% of Midterm 1, 2004)**

Define the follwing Haskell functions (the solutions are independent of each other):

(a) $sum :: [\,Integer\,] \to Integer$

such that $sum\ xs$ evaluates to the sum of all elements of the list $xs$.

(b) $all :: (\,a \to Bool\,) \to [\,a\,] \to Bool$

such that $all\ p\ xs$ evaluates to **True** if $p$ considered as a predicate holds for all elements of $x$, and to **False** if there is at least one element in $xs$ for which $p$ does not hold.

E.g., $all\ (>1)\ [2..10] = \textbf{True}$

(c) *selMod* :: *Integer* → [ *Integer* ] → [ *Integer* ]

such that *selMod k xs* selects from the list *xs* all those elements that are equivalent to *k* modulo *k* + 1, e.g.,

*selMod* 2 [2, 3, 8, 1, 2, 5] = [2, 8, 2, 5]

(d) *sources* :: *Eq a* ⇒ [ ( *a*, *a* ) ] → [ *a* ]

such that *sources ps* returns the *sources* of the graph *ps*.

Here, the list *ps* of pairs is considered as representing a simple graph by representing each edge from node *x* to node *y* by the pair ( *x*, *y* ).

The *context* "*Eq a* ⇒" just means that you may use the equality test for elements of type *a*, i.e., (==) :: *a* → *a* → *Bool*.

Example: *sources* [ (2,3), (3,4), (1,4), (1,5), (2,5) ] = [2,1]

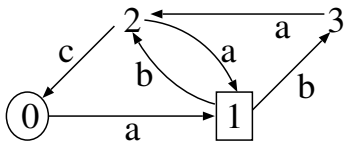
(The order is irrelevant.)

**Exercise 5.4 — 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 following finite-state machine (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 leaving state 1, there are two *b*-edges directed towards 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 leaving that state violate determinacy.

Other functions may be useful, too. **Document your functions!**