Written Examination
DM509 Programming Languages

Department of Mathematics and Computer Science
University of Southern Denmark

Friday, January 19, 2007, 09.00–13.00

The exam set consists of six pages (including this front page), and contains four questions. The weight of each question is as follows:

Question 1: 25%
Question 2: 20%
Question 3: 25%
Question 4: 30%

The parts of a question do not necessarily have equal weight. Note that often a part can be answered independently from the other parts.

All written aids are allowed. Unless otherwise stated in a question, use of results from the course textbooks, and of the standard libraries of the programming languages used, is allowed.

Question 1 (25%)

Part a: Make in Haskell a definition of a function

```
cuts :: [a] -> [([a],[a])]
```

such that `cuts list` is a list of tuples giving all the ways to divide `list` into two consecutive parts. As an example, `cuts [1,2,3]` should have the value

```
[([],[1,2,3]),([1],[2,3]),([1,2],[3]),([1,2,3],[])].
```

(or contain the same elements in some other ordering). □

Part b: Make in Haskell a definition of a function

```
shifts :: [a] -> [[a]]
```

such that `shifts list` is a list of all cyclic shifts of `list`. As an example, `shifts [1,2,3,4]` should have the value

```
[[1,2,3,4],[2,3,4,1],[3,4,1,2],[4,1,2,3]]
```

(or contain the same elements in some other ordering). □

Part c: Make in Haskell a definition of a function

```
permutations :: [a] -> [[a]]
```

such that `permutations list` is a list of all the permutations of `list`. As an example, `permutations [1,2,3]` should have the value

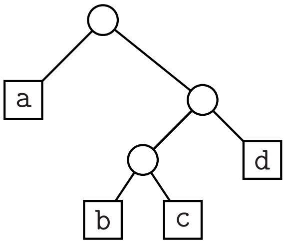
```
[[1,2,3],[2,1,3],[2,3,1],[1,3,2],[3,1,2],[3,2,1]]
```

(or contain the same elements in some other ordering). □

Question 2 (20%)

In Prolog, trees can be represented by structures. For instance, binary trees with elements at the leaves could be represented by structures with functor `node` and two components (left and right subtree) for internal nodes, and structures with functor `leaf` and one component (the element) for leaves.

In such a scheme, the tree



would be represented by

```
node(leaf(a),node(node(leaf(b),leaf(c)),leaf(d)))
```

Part a: Implement a Prolog predicate `height(Tree,N)` which, for `Tree` instantiated to a structure representing a tree according to the scheme above, is true iff `N` the height of the root. As usual, the height of a leaf is zero, and the height of an internal node is one larger than the largest height of its children. For the tree above, `N` should be `3`. □

Part b: Implement a Prolog predicate `flatten(Tree,List)` which, for `Tree` instantiated to a structure representing a tree according to the scheme above, is true iff `List` is the list of elements in the leaves of the tree, in left-to-right order. For the tree above, `List` should be `[a,b,c,d]`. □

Part c: Implement a Prolog predicate `sameshape(Tree1,Tree2)` which, for `Tree1` and `Tree2` instantiated to structures according to the scheme above, is true iff `Tree1` has the same tree structure as `Tree2`, but possibly different elements in the leaves.

As an example, if `Tree1` is instantiated to the structure representing the tree above, `sameshape(Tree1,Tree2)` should be true for `Tree2` instantiated to

```
node(leaf(1),node(node(leaf(x),leaf(2)),leaf(y)))
```

and false for `Tree2` instantiated to

```
node(node(node(leaf(a),leaf(b)),leaf(c)),leaf(d))
```

□

Question 3 (25%)

Part a: For the Prolog program below, state all results (i.e. all instantiations of `X` and `Y`) which will be produced by repeated satisfaction of the goal `f(X,b,Y)` (i.e. by repeated use of `';'`).

```
f(Z,b,Z) :- g(Z).
f(1,Z,1) :- g(Z).
g(a) :- !.
g(b).
g(c).
```

□

Part b: Convert the following predicate logic expression to clausal form:

$$\forall X(\neg(\forall Y(\exists Z((p(Y) \Rightarrow q(Z)) \land r(X)))))$$

Document the steps of your conversion. □

Part c: For each of the following pairs of Prolog predicates, find a most general unifier (with occurs-check), or argue that none exists. Explain each step of your derivations.

i) `q(q(X,Y),q(alice,bob))` and `q(Z,Z)`

ii) `alice(X,bob(X),X)` and `alice(bob(Y),Y,bob(Y))`

iii) `2=3` and `X=Y`

iv) `length([1,2,3])` and `3`

v) `append([1,2],[3,4],Z)` and `append(X,[3,4],[2,3])`

□

Question 4 (30%)

Part a: Consider the following Haskell definition:

```
f [] y     = y
f (x:xs) y = x:f y xs
```

Find the most general type of `f`. Explain each step of your derivation. ☐

Part b: Give the value of `f [1,2,3] [51,52,53,54]`, and explain the functionality of `f`. ☐

Part c: Answer the following:

i) Is `f` strict in its first argument?

ii) Is `f` strict in its second argument?

In each case, give arguments for your answer. ☐

Part d: Consider the following statement S:

```
length l1 + length l2 = length (f l1 l2)
```

where `length` is defined by

```
length:: [a] -> Int
length []     = 0
length (x:xs) = 1 + length xs
```

i) Show that statement S is true for `l1 = []`.

ii) Show that statement S is true for `l1 = (x:xs)` and `l2 = []`.

iii) Show that statement S is true for `l1 = (x:xs)` and `l2 = (y:ys)`, given that it is true for `l1 = xs` and `l2 = ys`.

Note that it follows [through an inductive proof with i) and ii) as base cases and iii) as the inductive step] that statement S is true for all finite lists `l1` and `l2`. No proof of this implication needs to be given. ☐

Part e: Make in Haskell a definition of a function `g` which for lists of equal length is an inverse to `f`. More precisely, it should be true that `(xs,ys) = g (f xs ys)` for finite lists `xs` and `ys` of equal length.

Only code needs to be given (no proof of the identity is required). □

Part f: The function `f` can also be defined using folding. Below is a possible code for such a variant `f1`:

```
f1 xs ys = (reverse mix) ++ yrest
   where
     (mix,yrest) = foldl h e xs
     e = ([],ys)
```

However, the binary function `h` used in the folding is not defined above. It has the following type:

```
h :: ([a],[a]) -> a -> ([a],[a])
```

Make in Haskell a definition of `h` such that `f xs ys` and `f1 xs ys` are the same for all finite lists `xs` and `ys`.

Only code needs to be given (no proof of the identity is required). □