Written Examination
DM22 Programming Languages

Department of Mathematics and Computer Science
University of Southern Denmark

Monday, June 12, 2006, 09.00–13.00

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

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

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%)

A pair of primes which differ by two are denoted *twin primes*. As an example, (17,19) is a pair of twin primes.

Recall that in an exercise during the course, the infinite list `primes :: [Int]` of all primes was defined.

Part a: Assuming `primes` is already defined, make in Haskell a definition of

```
twinprimes :: [(Int,Int)]
```

such that `twinprimes` is the infinite list[1] of pairs of twin primes. □

For a finite list, its *statistics* is a list of tuples telling how many times each list element appears. As an example, for the list `['a','b','c','a','a','c']` the statistics is `[('a',3),('b',1),('c',2)]`. The statistics for the empty list is defined as the empty list. We will only consider statistics for lists whose elements are members of the `Ord` class.

Part b: Make in Haskell a definition of

```
statistics :: Ord a => [a] -> [(a,Int)]
```

such that `statistics l` is the statistics for the list `l`. □

Ternary trees are trees where each internal node has three children. Assume a data type for ternary trees is defined by

```
data TTree a = TLeaf a | TNode (TTree a) (TTree a) (TTree a)
```

As usual, we define the depth of the root of a tree to be zero, the depth of its children to be one, and so forth.

Part c: Make in Haskell a definition of

```
depths :: TTree a -> [Int]
```

such that `depths t` is the list where the ith element is the number n_i of nodes (internal nodes as well as leaves) at depth $i = 0, 1, 2, \ldots$ in a tree `t`. It is sufficient that `depths t` is defined for finite trees `t` (in which case `depths t` is a list of length $k + 1$, where k is the maximal depth of a leaf in the tree). □

[1]More precisely, *potentially* infinite list, since it is a famous open problem in mathematics whether there are infinitely many pairs of twin primes.

Question 2 (25%)

A *merge* of two lists is another list containing each element exactly once, where the elements of each of the first two lists appear in order (but not necessary consecutively). As an example, one possible merge of the lists `[a,b,c,d]` and `[x,y,z]` is the list `[a,b,x,c,y,z,d]`.

Part a: Implement a Prolog predicate `merge(L1,L2,L3)` which is true iff `L3` is a merge of `L1` and `L2`. The predicate must be able to generate (as instantiations of `L3`) all merges of given lists `L1` and `L2` by repeated use of `';'`. □

In the game of *Buzz*, players alternate in saying aloud the next positive integer, except that integers divisible by either 3 or 7 (or both) should be replaced by the word "buzz".

Part b: Implement a Prolog predicate `buzz` which is always true, and where satisfaction of the goal `buzz` as a side effect prints on the screen the infinite sequence `1 2 buzz 4 5 buzz buzz 8 buzz 10 11 buzz 13 buzz buzz...` representing an infinite game of *Buzz*. □

The Danish version of the game is known as *Bum*, and is slightly different: "buzz" is replaced by "bum", there is only one divisor d to check for (not two), and integers where d occurs as a digit are *also* replaced by "bum". The number d is an integer between 1 and 9, and is decided upon when the game starts.

Part c: Implement a Prolog predicate `bum(D,N,L)` which is true iff `L` is the list of the first `N` entries in a game of *Bum* with digit `D` (where `N` and `D` are assumed to be instantiated).

As an example, satisfaction of the goal `bum(3,15,L)` should instantiate `L` to `[1,2,bum,4,5,bum,7,8,bum,10,11,bum,bum,14,bum]`. □

Question 3 (30%)

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 `t(X,Y)` (i.e. by repeated use of `';'`).

```
t(X,Y):- s(X),!,v(Y),s(X).
s(1).
s(2).
v(a).
v(b):- !.
v(c).
```

□

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

$$\forall X(\forall Y(p(X,Y) \Leftrightarrow \neg(\exists Z(q(X,Z)))))$$

Document the steps of your conversion. □

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

i) `q(g(X),X)` and `q(Y,g(Y))`

ii) `s(h(h(X)),h(Y),Z)` and `s(h(h(Y)),h(h(Z)),h(h(T)))`

iii) `X+Y` and `Y+X`

iv) `X*(Y+Z)` and `X*Y + X*Z`

□

Part d: Consider the Haskell function signatures

```
map :: (a -> b) -> [a] -> [b]
sum :: Num a => [a] -> a
```

Find the most general type of the expression `map sum`. Explain each step of your derivation. □

Part e: Consider the following Haskell definition:

```
m f g [] = []
m f g (x:xs) = f x : m g f xs
```

Find the most general type for `m` (explaining each step of your derivation), and describe its functionality. □

Question 4 (20%)

Consider the following two definitions `sum1` and `sum2` for a function summing the elements of a list.

```
sum1, sum2:: Num a => [a] -> a

sum1 []     = 0
sum1 (x:xs) = x + sum1 xs

sum2 = foldr (+) 0

foldr f e []     = e
foldr f e (x:xs) = f x (foldr f e xs)
```

Part a: Prove that for all finite lists `xs`, the following holds:

```
sum1 xs = sum2 xs
```

□

Part b: Prove the same for all infinite lists `xs`. □

Part c: Prove that `sum1 = sum2`. □