Written Examination
DM509 Programming Languages

Department of Mathematics and Computer Science
University of Southern Denmark

Friday, January 23, 2009, 14:00–18:00

The exam set consists of 6 pages (including this front page), and contains 5 problems. The weight of each problem is listed as a percentage of the full set. The questions of each problem do not necessarily have equal weight. Note that most often questions in a problem can be answered independently from the other questions.

All written aids are allowed. Answering questions by references to material not listed in the course curriculum is not acceptable.

In answering this exam set, there is free choice between Danish and English.

Problem 1 (25%)

Question a: Implement a PROLOG predicate `get(L, I, X)` which is true if and only if `X` is the `I`'th element in the list `L`, counting from zero. You may assume that `I` is an integer which is at least zero and smaller than the length of `L`.

An example of the intended use of this predicate is

```
get([1,3,5,7], 2, X)
```

which should give `X = 5` as the only answer. □

Question b: Implement a PROLOG predicate `getset(L, C, R)` which is true if and only if `R` the list of elements from `L` with indices in `C`, in the same order.

An example of the intended use of this predicate is

```
getset([1,3,5,7], [0,3], R)
```

which should give `R = [1,7]` as the only answer. □

Question c: Implement a PROLOG predicate `findindices(L, X, C)` which is true if and only if `C` is the list of indices where `X` occurs in `L`.

An example of the intended use of this predicate is

```
findindices([1,2,2,1,3,2], 2, C)
```

which should give `C = [1,2,5]` as the only answer. □

Question d: Implement a PROLOG predicate `occurtwice(L, R)` which is true if and only if `R` is the list of elements occuring exactly twice in `L`.

An example of the intended use of this predicate is

```
occurtwice([1,2,3,4,2,1,3,2], R)
```

which should give `R = [1,3]` and/or `R = [3,1]` as the only answer(s). □

Problem 2 (25%)

Question a: Consider the following PROLOG program:

```
s(X,Y) :- q(X,Y).
s(0,0).

q(X,Y) :- i(X), !, j(Y).
q(3,3).

i(1).
i(2).
i(3).
j(1).
j(2).
```

Draw the search tree traversed by the PROLOG interpreter during repeated satisfaction of the goal `s(X,Y).` (by repeated use of ';'). Also, list all results (instantiations of `X` and `Y`). □

Question b: For the following pairs of PROLOG predicates, find a most general unifier or argue that none exists. Show the steps of the algorithm you use.

1. `g(h(Y), Z, c)` and `g(Z, h(X), Y)`
2. `f(g(X), X, Y)` and `f(Y, c, g(b))`
3. `f([H|T], 1, Y, [X,Y], H)` and `f(L, X, 2, T, 0)`

□

Question c: Convert the following expression to clausal form:

$$\forall X\,\forall Y\,\neg(\,p(X,Y)\;\Rightarrow\;\forall Y\,q(Y,Y)\,)$$

List the steps of your conversion. □

Problem 3 (10%)

Question a: We define a *run* in a list of elements to be a maximal subsequence of identical elements, i.e., `[1,2,2,3,1,1,1,4,4,2]` has six runs, namely `[1]`, `[2,2]`, `[3]`, `[1,1,1]`, `[4,4]`, and `[2]`. Define a HASKELL function `runLengths` which takes a list of integers as argument and produces a list of the lengths of the runs in the order they appear, i.e., `runLengths [1,2,2,3,1,1,1,4,4,2] = [1,2,1,3,2,1]`. □

Question b: Given a positive integer n, we define the *Collatz sequence from* n as follows: It is an infinite sequence $n_0, n_1, n_2, \ldots$ defined by $n_0 = n$ and for any $i \geq 1$, $n_i = f(n_{i-1})$ where

$$f(x) = \begin{cases} x/2, & \text{if } x \text{ is even} \\ 3x+1, & \text{otherwise} \end{cases}$$

In HASKELL, define a function `collatz` such that `collatz n` produces the infinite list $[n_0, n_1, n_2, \ldots]$.

For example, evaluating `collatz 6` should give the infinite list

```
[6, 3, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1, ...]
```

and the definition should be given in such a way that, for example, `take 5 (collatz 6)` will produce `[6, 3, 10, 5, 16]`. □

Problem 4 (25%)

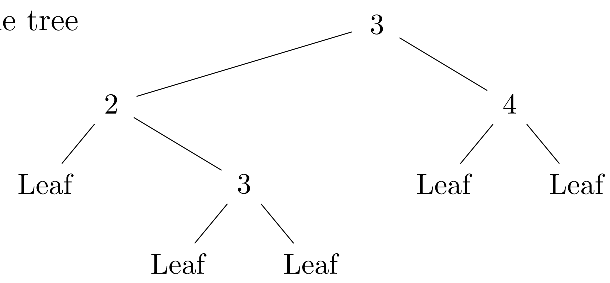
Consider the following data type of trees:

```
data Tree a = Node a (Tree a) (Tree a) | Leaf
```

Thus, the expression

```
ex = Node 3 (Node 2 Leaf (Node 3 Leaf Leaf)) (Node 4 Leaf Leaf)
```

corresponds to the tree



Question a: We define the *fringe* of a tree to be those nodes that have two leaves as children. Define a function

```
fringe :: Tree a -> [a]
```

which computes a list of all the values in the fringe nodes (with repetition, i.e., a value should appear in the result as many times as it appears in a fringe node). As an example, `fringe ex` should return `[3,4]`. □

The *level* of a node in a tree is defined as its distance from the root, i.e., the level of the root is zero, the level of a child of the root is one, the level of a grandchild of the root is two, etc.

Question b: Define a function

```
level :: Int -> Tree a -> [a]
```

which produces a list of all values in nodes at a particular level (again, with repetition).

As examples, `level 1 ex` should return `[2,4]` and `level 2 ex` should return `[3]`. □

Question c: Define a function `levels` that given a value and a tree computes a list of all the levels at which that value appears in the tree (again, with repetition). Also, state the most general type of the function `levels`.

As examples, `levels 2 ex` should return `[1]` and `levels 3 ex` should return `[0,2]`. □

Problem 5 (15%)

Question a: Find the most general type of each of the following two functions. Explain the steps in your derivation of the result.

1. `numberOf p xs = length (filter p xs)`
2. `restrict xs = map (\ (x,y) -> x < y) xs`

Recall that \ is the symbol in HUGS for λ and the types of the built-in functions are as follows:

- `length :: [a] -> Int`
- `filter :: (a -> Bool) -> [a] -> [a]`
- `map :: (a -> b) -> [a] -> [b]`

□

Question b: Consider the two different ways of adding elements pairwise from two lists which can be assumed to be of equal length:

```
s1 [] [] = []
s1 (n:ns) (m:ms) = (n+m) : (s1 ns ms)

h [] = []
h ((x,y):xs) = (x+y) : (h xs)

s2 ns ms = h (zip ns ms)
```

Prove by induction that on lists `xs` and `ys` of type `[Int]` and of the same length, `s1 xs ys` = `s2 xs ys`. Argue for the steps in your derivation. □