Written Examination
DM 509 Programming Languages
– Solution –

Department of Mathematics and Computer Science
University of Southern Denmark

Monday, January 10, 2011, 09:00 – 13:00

This exam set consists of 7 pages (including this front page) and contains a total of 5 problems. Each problem is weighted by the given percentage. The individual questions of a problem are not necessarily weighted equally.

Most questions in a problem can be answered independently from the other questions of the same problem.

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

You may answer the exam in English or in Danish.

This document contains the essential parts of the solutions to the exam set identified above. Note that many times, there are several possible solutions, and this document just lists one. Also, perfect answers to some of the exam questions should contain explanations which are generally omitted in this document. Finally, this document has not been scrutinized in the same meticulous manner as an exam set and may contain typos, etc.

Problem 1 (20%)

Question a: Implement a Prolog predicate `dropFirst/3` such that `dropFirst(N, L, M)` is true if, and only if, `M` is the list obtained by dropping the first `N` elements of the list `L`.

For example, the query

```
?- dropFirst(2, [6,3,4,5,2,1], M).
```

should yield the answer `M = [4,5,2,1]`. Likewise, the query

```
?- dropFirst(4, [1,2,3], M).
```

should yield the answer `M = []`.

Possible Solution:

```
dropFirst(_,[],[]) :- !.
dropFirst(0,L,L).
dropFirst(N,[_|L],M) :- N > 0, N1 is N-1, dropFirst(N1,L,M).
```

Question b: Implement a Prolog predicate `takeNth/3` such that `takeNth(N, L, M)` is true if, and only if, `M` is the list obtained by taking every `N`-th element from the list `L`.

For example, the query

```
?- takeNth(2, [6,3,4,5,2,1], M).
```

should yield the answer `M = [6,4,2]`. Likewise, the query

```
?- takeNth(3, [1,2,3,4], M).
```

should yield the answer `M = [1,4]`.

Possible Solution:

```
takeNth(_,[],[]).
takeNth(N,[X|L],[X|M]) :- dropFirst(N,[_|L],L1), takeNth(N,L1,M).
```

Question c: A fraction $\frac{a}{b}$ can be represented by the term `a/b`. Note that instead of "/" one could also use "-" or "+".

Implement a Prolog predicate `add/3` such that `add(X,Y,Z)` is true if, and only if, `Z` is the fraction obtained by adding `X` and `Y`.

For example, the query

```
?- add(1/6, 3/10, F).
```

should yield the answer `F = 28/60`. Likewise, the query

```
?- add(3/1, 1/2, F).
```

should yield the answer `F = 7/2`.

Possible Solution:

```
add(A/B,C/D,X/Y) :- X is A*D+C*B, Y is B*D.
```

Question d: A heterosquare is a matrix of dimension $n \times n$ containing all numbers from 1 to n^2 such that the sums of all rows and of all columns are pairwise different.

The following is an example of a heterosquare of dimension 2×2. Note that $1+2 = 3$, $3+4 = 7$, $1+3 = 4$, and $2+4 = 6$, i.e., we have the sums 3, 7, 4, and 6, which are all pairwise different.

1	2
3	4

We represent such a square as a list of concatenated rows, i.e., the above square would be represented as follows.

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

Implement a Prolog predicate `hetero/1` such that the query `?- hetero(L).` has exactly those lists `L` as answers that represent heterosquares of dimension 2×2.

You may (but do not have to) use constraint logic programming for your implementation.

Possible Solution:

```
hetero(L) :- L = [A,B,C,D],
  S = [R1,R2,C1,C2],
  fd_domain(L,1,4),  fd_domain(S,1,7),
  fd_all_different(L),  fd_all_different(S),
  A+B #= R1,  C+D #= R2,  A+C #= C1,  B+D #= C2,
  fd_labeling(L).
```

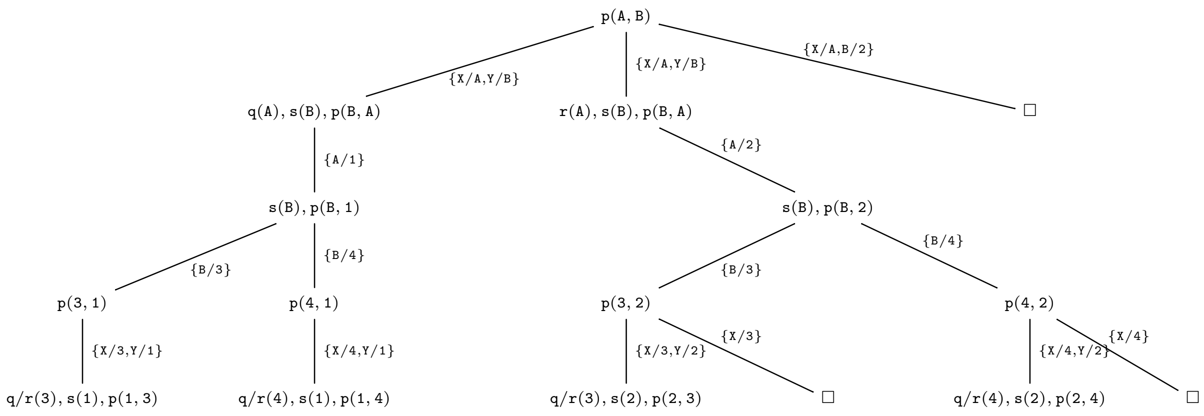
Problem 2 (25%)

Question a: Consider the following Prolog program.

```
p(X,Y) :- q(X), s(Y), p(Y,X).
p(X,Y) :- r(X), s(Y), p(Y,X).
p(X,2).
q(1).
r(2).
s(3).
s(4).
```

Draw the SLD tree for the query `?- p(A,B).` and list all answers with the instantiations of `A` and `B`.

Possible Solution:



Here, `q/r(...), ...` represents the two nodes `q(...), ...` and `r(...), ...` obtained by unifying with the first two clauses from `p`.

The answers returned by Prolog are:

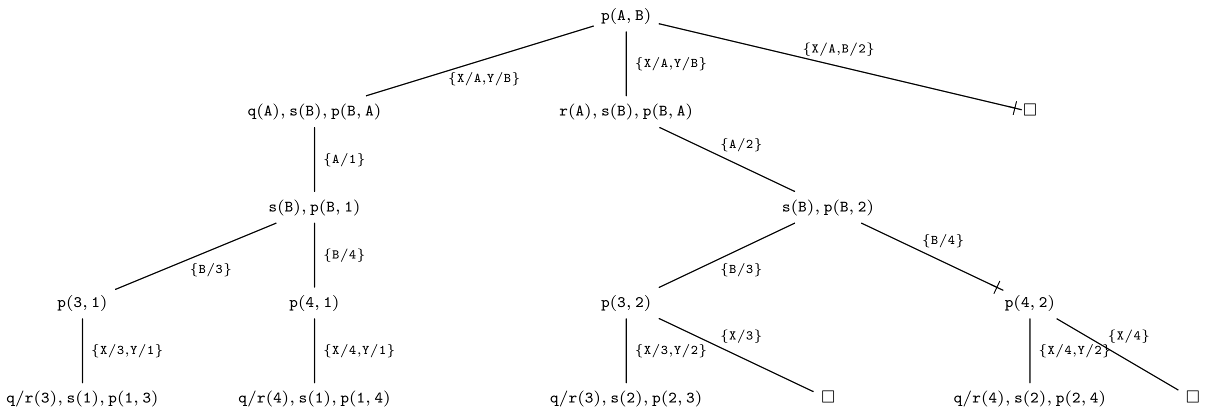
```
A = 2, B = 3
A = 2, B = 4
B = 2
```

Question b: We now introduce a cut into the body of the second clause from Question a, i.e., we now have the following Prolog clauses for `p/2`.

```
p(X,Y) :- q(X), s(Y), p(Y,X).
p(X,Y) :- r(X), s(Y), p(Y,X), !.
p(X,2).
```

Indicate in the SLD tree of Question a which branches are cut and list all remaining answers with the instantiations of `A` and `B`.

Possible Solution:



The only answer returned by Prolog is:

```
A = 2, B = 3
```

Question c: For the following pairs of Prolog terms, find a most general unifier or argue that none exists. Show the steps of the algorithm. In case of success, give the resulting substitution. In case of failure, state if it is an occur failure or a clash failure.

1. `p(f(b),a,Y)` and `p(f(Y),X,b)`
2. `q(g(X),g(Y),g(a))` and `q(g(A),A,g(X))`
3. `r(a,Z,[X,Y])` and `r(X,Y,[X|Z])`

Possible Solution:

1. SUBSTITUTION {X/a, Y/b}

$$\begin{array}{rl}
 & \{\mathtt{p(f(b),a,Y)} \overset{?}{=} \mathtt{p(f(Y),X,b)}\} \\
\Rightarrow (\mathit{DECOMPOSE}) & \{\mathtt{f(b)} \overset{?}{=} \mathtt{f(Y)}, \mathtt{a} \overset{?}{=} \mathtt{X}, \mathtt{Y} \overset{?}{=} \mathtt{b}\} \\
\Rightarrow (\mathit{DECOMPOSE}) & \{\mathtt{b} \overset{?}{=} \mathtt{Y}, \mathtt{a} \overset{?}{=} \mathtt{X}, \mathtt{Y} \overset{?}{=} \mathtt{b}\} \\
\Rightarrow (\mathit{ELIMINATE}) & \{\mathtt{b} \overset{?}{=} \mathtt{b}, \mathtt{a} \overset{?}{=} \mathtt{X}, \mathtt{Y} \overset{?}{=} \mathtt{b}\} \\
\Rightarrow (\mathit{DELETE}) & \{\mathtt{a} \overset{?}{=} \mathtt{X}, \mathtt{Y} \overset{?}{=} \mathtt{b}\} \\
\Rightarrow (\mathit{ORIENT}) & \{\mathtt{X} \overset{?}{=} \mathtt{a}, \mathtt{Y} \overset{?}{=} \mathtt{b}\}
\end{array}$$

2. CLASH FAILURE

$$\begin{array}{rl}
 & \{\mathtt{q(g(X),g(Y),g(a))} \overset{?}{=} \mathtt{q(g(A),A,g(X))}\} \\
\Rightarrow (\mathit{DECOMPOSE}) & \{\mathtt{g(X)} \overset{?}{=} \mathtt{g(A)}, \mathtt{g(Y)} \overset{?}{=} \mathtt{A}, \mathtt{g(a)} \overset{?}{=} \mathtt{g(X)}\} \\
\Rightarrow (\mathit{DECOMPOSE}) & \{\mathtt{X} \overset{?}{=} \mathtt{A}, \mathtt{g(Y)} \overset{?}{=} \mathtt{A}, \mathtt{g(a)} \overset{?}{=} \mathtt{g(X)}\} \\
\Rightarrow (\mathit{DECOMPOSE}) & \{\mathtt{X} \overset{?}{=} \mathtt{A}, \mathtt{g(Y)} \overset{?}{=} \mathtt{A}, \mathtt{a} \overset{?}{=} \mathtt{X}\} \\
\Rightarrow (\mathit{ELIMINATE}) & \{\mathtt{X} \overset{?}{=} \mathtt{A}, \mathtt{g(Y)} \overset{?}{=} \mathtt{A}, \mathtt{a} \overset{?}{=} \mathtt{A}\} \\
\Rightarrow (\mathit{ORIENT}) & \{\mathtt{X} \overset{?}{=} \mathtt{A}, \mathtt{g(Y)} \overset{?}{=} \mathtt{A}, \mathtt{A} \overset{?}{=} \mathtt{a})\} \\
\Rightarrow (\mathit{ELIMINATE}) & \{\mathtt{X} \overset{?}{=} \mathtt{A}, \mathtt{g(Y)} \overset{?}{=} \mathtt{a}, \mathtt{A} \overset{?}{=} \mathtt{a})\}
\end{array}$$

3. OCCUR FAILURE

$$\begin{array}{rl}
 & \{\mathtt{r(a,Z,[X,Y])} \overset{?}{=} \mathtt{r(X,Y,[X|Z])}\} \\
= & \{\mathtt{r(a,Z,.(X,.(Y,[])))} =?\mathtt{r(X,Y,.(X,Z))}\} \\
\Rightarrow (\mathit{DECOMPOSE}) & \{\mathtt{a} \overset{?}{=} \mathtt{X}, \mathtt{Z} \overset{?}{=} \mathtt{Y}, \mathtt{.(X,.(Y,[]))} \overset{?}{=} \mathtt{.(X,Z)}\} \\
\Rightarrow (\mathit{DECOMPOSE}) & \{\mathtt{a} \overset{?}{=} \mathtt{X}, \mathtt{Z} \overset{?}{=} \mathtt{Y}, \mathtt{X} \overset{?}{=} \mathtt{X}, \mathtt{.(Y,[])} \overset{?}{=} \mathtt{Z}\} \\
\Rightarrow (\mathit{DELETE}) & \{\mathtt{a} \overset{?}{=} \mathtt{X}, \mathtt{Z} \overset{?}{=} \mathtt{Y}, \mathtt{.(Y,[])} \overset{?}{=} \mathtt{Z}\} \\
\Rightarrow (\mathit{ELIMINATE}) & \{\mathtt{a} \overset{?}{=} \mathtt{X}, \mathtt{Z} \overset{?}{=} \mathtt{Y}, \mathtt{.(Y,[])} \overset{?}{=} \mathtt{Y}\} \\
\Rightarrow (\mathit{ORIENT}) & \{\mathtt{a} \overset{?}{=} \mathtt{X}, \mathtt{Z} \overset{?}{=} \mathtt{Y}, \mathtt{.Y} \overset{?}{=} \mathtt{.(Y,[])}\}
\end{array}$$

Problem 3 (15%)

Question a: Define a HASKELL function `max` which takes two non-negative positive integers and determines the maximum of the two.

For example, `max 2 3 = 3` and `max 5 0 = 5`.

Here, you may not use any pre-defined functions except for `(+)` and `(-)`. In particular, you may not use `(>)` or any other comparison operator.

Possible Solution:

```
max 0 y = y
max x 0 = x
max x y = 1 + max (x-1) (y-1)
```

Question b: Define a HASKELL function `maxList` which takes two lists of non-negative integers of same length and builds a list, which at each position contains the maximum of the elements of the two argument lists.

For example, `maxList [1,6,3] [2,4,5] = [2,6,5]`.

You should use the function `max` from Question a.

Possible Solution:

```
maxList = zipWith max
```

Alternative Solution:

```
maxList [] [] = []
maxList (x:xs) (y:ys) = max x y : maxList xs ys
```

Question c: Define a HASKELL function `transpose` which takes a matrix of integers represented as a list of rows and computes the transposed matrix.

For example, `transpose [[1,2],[3,4]] = [[1,3],[2,4]]` and `transpose [[1,2,3,4],[5,6,7,8]] = [[1,5],[2,6],[3,7],[4,8]]`.

Possible Solution:

```
transpose [] = []
transpose ([]:mss) = transpose mss
transpose mss = map (\x:xs) -> x) mss : transpose (map (\(x:xs) -> xs))
```

Question d: Give a HASKELL declaration for the infinite list `powersOf2` of all strings consisting of just "*" with a length that is a power of 2, i.e., a declaration of the form "`powersOf2 = ...`".

For example, `take 5 powersOf2` should return the following list.

```
["*","**","****","********","****************"]
```

for the following (standard definition) of `take`:

```
take 0 _ = []
take _ [] = []
take (n+1) (x:xs) = x : take n xs
```

Possible Solution:

```
powersOf2 = "*" : map (\x -> x++x) powersOf2
```

Problem 4 (20%)

Question a: Consider the following data type for multisets, i.e., for sets that can contain an element multiple times.

```
data MultiSet a = a -> Integer
```

Thus, the expression `\x -> 0` represents the empty multiset $\{\}$ while the expression `\x -> if x == 4 then 1 else if x == 1 then 3 else 0` represents the multiset $\{1, 1, 1, 4\}$.

Functions that work on these multisets need to create, apply, or modify functions. The following declarations for `emptyMS`, `frequency`, and `insert` define the empty multiset, a function returning the multiplicity of an element, and insert an element into a multiset, respectively:

```
emptyMS = \x -> 0
frequency x ms = ms x
insert x ms = \y -> ms y + if x == y then 1 else 0
```

Define a HASKELL function `union` which takes two `MultiSet a` and produces the union of the two multisets.

For example,
`union (insert 1 (insert 1 emptyMS)) (insert 4 (insert 1 emptyMS))`
should return a function representing the multiset $\{1, 1, 1, 4\}$.

Possible Solution:

```
union ms1 ms2 = \x -> ms1 x + ms2 x
```

Question b: Consider an alternative representation of multisets as lists of pairs where $[(4,1),(1,3)]$ represents the multiset $\{1,1,1,4\}$.

Define a HASKELL function `toList` which takes a multiset represented by a value of type `[(a,Integer)]` and returns a list of type `[a]` that contains each element of the multiset as many times as specified.

For example, `toList [(4,1),(1,3)]` could return `[4,1,1,1]` or `[1,1,1,4]`. Note that the order is not important.

Possible Solution:

```
toList = foldr (\(x,n) xs -> replicate n x ++ xs) []
```

Alternative Solution:

```
toList [] = []
toList ((_,0):xs) = toList xs
toList ((x,n):xs) = x : toList ((x,n-1):xs)
```

Question c: Declare a HASKELL data type `MSList a` using a `data` declaration to represent values of type `[(a,Integer)]` by self-defined data constructors.

Possible Solution:

```
data MSList a = EmptyMS | Insert a Integer (MSList a)
```

Problem 5 (20%)

Question a: Find the most general type for each of the following two HASKELL functions. You may assume that `False,True::Bool`, `1::Int`, and `(:)::a -> [a] -> [a]`.

- `f = \x -> if x 1 then False else True`
- `g (x:xs) y = x (g xs y)`

Explain your reasoning.

Possible Solution:

- Assume `f::a` and `x::b`. As `x` is applied to `1` and must return a value of type `Bool`, we have that `b` unifies with `Int -> Bool`. The value of the `if ...then ...else ...` has the same type as `True` and `False`, i.e, `Bool`. Thus `a` has to unify with `(Int -> Bool) -> Bool`.

  ```
  f::(Int -> Bool) -> Bool
  ```

- Assume `g::c -> d -> a`, `x::h`, `xs::i`, and `y::b`. Because of the subexpression `(x:xs)` we need to unify `k -> [k] -> [k]` with `h -> i -> c` and obtain `x::k`, `xs::[k]`, and `g::[k] -> d -> a`. By the second argument of `g` we obtain `g::[k] -> b -> a`. As `x` is applied to `g xs y`, we need to unify `k` with `l -> m`, `l` with `a`, and `m` with `a`.

  ```
  g::[a -> a] -> b -> a
  ```

Question b: Consider the two following ways of defining HASKELL functions for mapping a function to all elements of a list.

```
map1 f [] = []
map1 f (x:xs) = f x : map1 f xs

map2 f = foldr help [] where
  help x xs = f x : xs

foldr g h [] = h
foldr g h (x:xs) = g x (foldr g h xs)
```

Prove by induction that for all `f` of type `[a -> b]` and `ys` of type `[a]`, these two definitions yield the same result, i.e., `map1 f ys = map2 f ys`.

Possible Solution:

- base case (`ys = []`)

  $\underline{\texttt{map1 f []}}$ `= [] = foldr help [] [] =` $\underline{\texttt{map2 f []}}$

- step case (assume theorem holds for `ys`, show it holds for `y:ys`)
  Using the definition of `map1` and the induction hypothesis we obtain:

  $\underline{\texttt{map1 f (y:ys)}}$ `= f y : map1 f ys = f y : map2 f ys`

  Next, we use the definition of `map2` and the definition of `help`:

  `f y : map2 f ys = f y : foldr help [] ys = help y (foldr help [] ys)`

  Now, we use the definitions of `foldr` and `map 2` backwards:

  `help y (foldr help [] ys) = foldr help [] (y:ys) =` $\underline{\texttt{map2 f (y:ys)}}$

  Thus, we have shown that $\underline{\texttt{map1 f ys}}$ `=` $\underline{\texttt{map2 f ys}}$.