Written Examination
DM509 Programming Languages

Department of Mathematics and Computer Science
University of Southern Denmark

Saturday, January 19, 2008, 10.00–14.00

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

Question 1: 20%
Question 2: 25%
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 (20%)

Consider the following PROLOG script.

```
p(X,Y) :- q(X),r(Y).
p(X,X) :- q(X).
q(a).
q(b).
r(1).
r(2).
r(3).
```

Part a: Draw the entire search tree resulting from repeated satisfaction of the goal `?- p(A,B)` (by repeatedly entering `;` in `gprolog`).

A node in your tree should represent the current goal and every edge should represent a successful unification; the results of which should be listed on the corresponding edge. Recall that examples of search trees are given in the notes.

Also state all answers (instatiantions of `A` and `B`) reported by `gprolog` during the repeated satisfaction. □

Now consider the following altered script.

```
p(X,Y) :- q(X),!,r(Y).
p(X,X) :- q(X).
q(a).
q(b).
r(1).
r(2) :- !.
r(3).
```

Part b: Show which parts of the search tree from **Part a** are removed due to the added cuts. □

Question 2 (25%)

In cryptography, a Caesar cipher is one of the simplest and most widely known encryption techniques. The method is named after Julius Caesar, who used it to communicate with his generals. It is a substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet in a circular fashion. In this question we consider the alphabet to be the lower case letters 'a'...'z'. For example, with a shift of 3, 'a' would be replaced by 'd', 'b' by 'e', and 'z' by 'c'.

In this question we will implement Caesar cipher encryption using Prolog. For this we will consider messages to be a list of single lowercase characters, e.g. the message "Prolog" will be written `[p,r,o,l,o,g]`. To convert a message in this form into a list of character codes, we may use the built-in predicate `char_code/2`, that is true iff the single lowercase character given as the first argument has the character code given as the second argument, e.g. `char_code(a,97)` is true, but `char_code(b,97)` is not.

The predicate can be used to convert between characters and character codes, e.g. by `char_code(a,A)`, and vice versa, but fails if neither of its arguments are instantiated. The character codes of 'a'...'z' are 97...122.

Part a: Implement a Prolog predicate

```
charCodes(Chars,Codes) :- ...
```

that is true iff `Chars`, a list of single lowercase characters, corresponds to `Codes`, a list of character codes. That is, the first element of `Chars` must have the first element of `Codes` as its character code, the second element of `Chars` must have the second element of `Codes` as its character code, etc.

Your implementation should make use of `char_code/2` and recursion. □

For the next part, assume that a Prolog predicate

```
rotateRight(L1,L2,Places)
```

is available. Assume the predicate is true iff `L2`, a list of numbers, is equal to `L1`, another list of numbers, rotated `Places` right within the range $[97, 122]$. For example, satisfying the goal `rotateRight([97,98,122],L2,3)` will instatiate `L2` to `[100,101,99]`.

Part b: Implement a Prolog predicate

```
encrypt(Plain,Crypt,Shift) :- ...
```

that is true iff and only iff `Crypt`, a list of single lowercase characters, is the Caeser cipher encryption with shift `Shift` of `Plain`.

The idea is, that we can use this predicate to encrypt a message, e.g. instatiate `C` to `[s,u,r,o,r,j]` by satisfying `encrypt([p,r,o,l,o,g],C,3)`. □

Part c: Implement the PROLOG predicate `rotateRight(L1,L2,Places)`. *Hint: The operator `mod` is used to calculate the modulus of two numbers. For example, `M is P mod Q`.* □

Part d: Using your implementations of `charCodes/2`, `encrypt/3` and `rotateRight/3`, is it possible to instatiate `P` to `[p,r,o,l,o,g]` by attempting satisfaction of the goal `encrypt(P,[s,u,r,o,r,j],3)`? Explain your answer. □

Question 3 (25%)

Part a: Find a most general unifier of the following pairs of predicates or argue that none exists. Explain each step of your derivations.

a) $p(q(A), B)$ and $p(C, q(C))$.

b) $p(x, q(A), A)$ and $p(C, q(C), y)$.

c) $p(A, q(A))$ and $p(r(B), B)$.

d) $t(M/a)$ and $t(b/N)$.

□

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

$$\exists A (\forall B (p(A, B) \wedge (q(B) \Rightarrow r(A)))).$$

Explain each step of your conversions. □

Recall the definition of the Haskell library function `map`:

```
map :: (a -> b) -> [a] -> [b]
map f xs = [f x|x <- xs]
```

Part c: Find the most general type of the following Haskell function:

```
mm f g []     = []
mm f g (x:xs) = map f ((g x):(mm f g xs))
```

Explain your reasoning. □

Question 4 (30%)

In this problem we consider the whole numbers $\mathbb{Z}$ represented in Haskell by

```
data Whole = Zero | Succ Whole | Pred Whole deriving Show
```

We will only consider `Whole` values that contain a finite number of the type constructors `Succ` and `Pred`. Recall that the textbook has a similar definition of the natural numbers.

As an example of how to interpret a value w of type `Whole` as an integer value in $\mathbb{Z}$, consider

$w =$ `Succ (Succ (Pred (Succ Zero)))`

The integer interpretation of w is then

$$1 + (1 + (-1 + (1 + 0))) = 2,$$

i.e., starting from 0, add 1 for each occurence of `Succ` and substract 1 for each occurence of `Pred`.

Part a: Define in Haskell a function

```
wholeToInt :: Whole -> Int
```

that converts its argument to the corresponding integer value. □

Naturally, we would like to convert integer values into `Whole` values as well. However, this presents a problem; there are several `Whole` representations of the same integer!

Part b: Show that using the above data type, there are infinitely many distinct representations of a number $z \in \mathbb{Z}$. *Hint: Consider a representation of z as a `Whole` w. Show that any such representation may be extended into a new representation.* □

For a `Whole` w, we define the *reduced form* to be the unique representation of w that contains only `Succ` if w represents a positive number, only `Pred` if w represents a negative number, or neither if w represents 0. For example, the reduced form representation of 2 is `Succ (Succ Zero)`.

You are not required to show that such a representation exists nor that it is unique.

Part c: Define in Haskell a function

```
intToWhole :: Int -> Whole
```

that converts its argument to the reduced form `Whole` representation of the same value. □

Part d: Show by induction over `w` that

```
wholeToInt (intToWhole w) = w
```

Hint: Use two inductions; one for the positive whole numbers and one for the negative. □

Part e: Define in Haskell a function

```
reduce :: Whole -> Whole
```

that converts an arbitrary `Whole` w to its corresponding reduced form. This conversion must be done without using other datatypes than `Whole`, i.e. you are not allowed to first convert w to an `Int`. □