Written Examination, December 19th, 2006 Course no. 02153

The duration of the examination is 4 hours.

Course Name: Declarative Modelling

Allowed aids: All written material

The problem set consists of 7 problems which are weighted as follows:
Problem 1: 10%, Problem 2: 15%, Problem 3: 10%, Problem 4: 15%,
Problem 5: 25%, Problem 6: 10%, Problem 7: 15%

Marking: 13-scale.

Problem 1 (10%)

In the following it can be assumed that all elements of the lists are integers.

Question 1.1

Write a Prolog program `cutoff` such that `cutoff(+List1,?List2)` succeeds if and only if `List2` is the longest prefix of `List1` without negative elements.

Sample Prolog queries:

```
?- cutoff([1,-2,3],[1]).

Yes

?- cutoff([1,-2,3],[1,-2,3]).

No
```

Problem 2 (15%)

In the following a Prolog program is said to be deterministic if and only if it does not succeed more than once.

Consider the following basic predicates:

```
member(H,[H|_]).
member(H,[_|T]) :- member(H,T).

append([],U,U).
append([H|T],U,[H|V]) :- append(T,U,V).
```

Here `member(?Elem,?List)` succeeds if and only if `Elem` can be unified with one of the members of `List` and `append(?List1,?List2,?List3)` succeeds if and only if `List3` unifies with the concatenation of `List1` and `List2`.

Consider the following fragment of a boxing club database:

```
beat(a,[b,c,d]).
beat(b,[]).
beat(c,[d]).
beat(d,[b]).
beat(e,[a]).
```

Hence boxer `a` has beaten `b`, `c` and `d`, whereas `b` has not beaten anyone.

 ... Continued on next page

Question 2.1

Write a deterministic Prolog program `fighter(+Boxer)` corresponding to the following definition: X is a fighter if and only if X has been beaten by someone and X has beaten someone.

The predicate `member` might be useful.

Question 2.2

Write a deterministic Prolog program `count` that prints the number of beaten boxers for each boxer as follows:

```
?- count.
3 a
0 b
1 c
1 d
1 e

Yes
```

Notice that the Prolog query must always succeed.

Question 2.3

The president of the boxing club wants a Prolog program that can divide a group of boxers in two groups such that each group has at least one fighter.

Consider the following Prolog program:

```
divide(L,A,B) :- append(A,B,L), checkfighter(A), checkfighter(B).

checkfighter(A) :- member(X,A), fighter(X), !.
```

State the exact sequence of solutions for the Prolog query:

```
?- divide([a,b,c,d,e],A,B).
```

Question 2.4

Does the Prolog program `divide` work as the president of the boxing club expects?

Provide a brief explanation.

 ... Continued on next page

Problem 3 (10%)

Consider the following fragment of a food ingredient database:

```
ingredient(pizza,ham).
ingredient(pizza,sauce).
ingredient(pizza,cheese).
ingredient(ham,meat).
ingredient(ham,salt).
ingredient(cheese,milk).
ingredient(cheese,salt).
ingredient(sauce,tomato).
ingredient(sauce,water).
ingredient(sauce,salt).
```

Hence `pizza` contains the ingredients `ham`, `sauce` and `cheese`. An ingredient may contain other ingredients, for example `ham` contains the ingredients `meat` and `salt`.

Question 3.1

Write a Prolog program `component` such that `component(?Term1,?Term2)` succeeds if and only if `Term1` is an ingredient in `Term2` either directly or indirectly because it is a component of an ingredient in `Term2`.

Sample Prolog queries:

```
?- component(salt,pizza).

Yes

?- component(jam,pizza).

No
```

Question 3.2

State the exact sequence of solutions for the Prolog query:

```
?- component(X,pizza).
```

 ... Continued on next page

Problem 4 (15%)

Consider the following formula:

$$\exists x \forall y p(x,y) \rightarrow \forall y \exists x p(x,y)$$

Question 4.1

Use refutation and the systematic construction of a semantic tableau. State whether this shows that the formula is valid or not.

Question 4.2

Use refutation, skolemization and the general resolution procedure. State whether this shows that the formula is valid or not.

 ... Continued on next page

Problem 5 (25%)

In this problem we consider evaluation of arithmetical expressions (type `expr`) constructed from numbers and identifiers using operators for addition and subtraction and *let-expressions* with local declarations (type `decl`), as declared by the following mutually recursive datatype declarations.

```
datatype expr =
    N of int                                (* Number          *)
  | I of string                             (* Identifier      *)
  | Add of expr * expr                      (* Addition        *)
  | Sub of expr * expr                      (* Subtraction     *)
  | Let of decl * expr                      (* let-expression  *)

and decl = D of (string * expr) list        (* declaration list *)
```

Expressions are evaluated in an *environment* (type `env`) which associates integer values to strings representing identifiers. We represent an environment by a list of pairs:

```
type env = (string * int) list;
```

For example, using a suitable, informal expression-notation

```
'z + (let x=3, y=x+1 in x+y)'
```

should evaluate to 9 in an environment where `z` is 2.

1. Give the SML value corresponding to `'z + (let x=3, y=x+1 in x+y)'`.
2. Declare SML functions `lookup` and `update` and state their types.

   The function `lookup` should for a given string x and a given environment *env* give the integer value associated to x in *env*. An exception should be raised if no value is associated to x in *env*.

   The function `update` should for a given string x, integer n and environment *env*, give a new environment, which is as *env* except that n is associated to x.
3. Declare mutually recursive functions:

   `E: expr -> env -> int` and `Extend: decl -> env -> env`

   The function `E` should evaluate an expression in a given environment.

   The function `Extend` should, given a declaration list and an environment, produce a new updated environment, where the declarations are taken into account.
4. We want to extend the language with an `if-then-else` expression, to allow expressions like `if x<y and z=5 then y else x+2`, written in a suitable concrete syntax.

   Extend your program to incorporate this extension.

 ... Continued on next page

Problem 6 (10%)

Consider the following SML declarations:

```
fun f x y = if x <= y then y :: f x (y-1) else [];

fun g h ([], _)         = false
  | g h (_, [])         = false
  | g h (x::xs, y::ys) = h(x,y)
                          orelse g h (x::xs, ys)
                          orelse g h (xs, y::ys);
```

1. Give the type of `f`, and give the values of the expressions `f 1 0` and `f 1 3`. Describe what `f` computes.
2. Give the type of `g`, and describe what `g` computes.
3. Give the type of `g (fn (x,y) => x=y)`, and describe what `g (fn (x,y) => x=y)` computes.

Problem 7 (15%)

Consider the following SML declarations:

```
fun take(_, [])     = []
  | take(0, _)      = []
  | take(i, x::xs) = x :: take(i-1, xs);

fun drop(_, [])     = []
  | drop(0, xs)     = xs
  | drop(i, x::xs) = drop(i-1, xs);

infix @; fun       [] @ ys = ys
  | (x::xs) @ ys = x::(xs @ ys);
```

Prove that

$$\texttt{take}(k, xs) \mathbin{@} \texttt{drop}(k, xs) = xs$$

holds for every list xs and every non-negative integer k.