Problem Set for Tutorial 4 — TDDD08

Wlodek Drabent*[1] and Victor Lagerkvist†[1]

[1]Department of Computer and Information Science, Linköping University, Linköping, Sweden

1. Determine which of the following pairs of terms that are unifiable, and provide the most general unifier (mgu) in case there is one:
   (a) `p(f(X),X,f(Y))` and `p(Y,f(Z),Z)`.
   (b) `p(f(X),f(Y),X)` and `p(Z,Z,W)`.
   (c) `p(X1,X2,X3)` and `p(f(X2,X2),f(X3,X3),a)`.
   (d) `[X,Y|Xs]` and `[a,b,c]`.
   (e) `[X,f(X)|X]` and `[Z,Y,Z]`.

2. Draw the SLD-tree for the program below and the query `member(X,[a,b])`, and sketch an SLD-tree for the query `member(a, Xs)`.

```
member(X, [X|_]).
member(X, [_|L]) :- member(X,L).
```

3. Consider the following definitions of `permutation/2` and `select/3`.

```
%permutation(Xs,Ys) is true if Xs is a permutation of Ys.
permutation([], []).
permutation([X|Xs], Ys) :- select(X, Ys, Ys1), permutation(Xs, Ys1).

%select(X, Ys, Ys1) is true if Ys1 is the list obtained by removing an
%occurrence of X in Ys.
select(X, [X|Ys], Ys).
select(X, [Y|Ys], [Y|Zs]) :- select(X, Ys, Zs).
```

   Draw the SLD-tree for the query `permutation(P, [a,b])` using Prolog's selection rule of always choosing the leftmost atom in the node of the tree. To make the drawing simpler, begin by drawing the SLD-tree for the query `permutation(P, [a])`. Then, when you draw the SLD-tree for the query `permutation(P, [a,b])` you can re-use the SLD-tree for `permutation(P, [a])`.

*wlodek.drabent@liu.se
†victor.lagerkvist@liu.se

4. Assume that we represent simple arithmetical expressions as terms as follows: `var(x)` is an arithmetical term if `x` is a non-numerical atom, `num(n)` is an arithmetical term if `n` is an integer, and if T_1 and T_2 are arithmetical terms then $T_1 + T_2$ and $T_1 * T_2$ are arithmetical terms. For example, `var(x) + var(y) * var(z) + num(10)` would be an example of an arithmetical term. Note that an arithmetic term of this form can only be evaluated to a concrete value once the values of all involved variables have been specified. A *binding environment* is a data structure which for each variable x contains a numerical value n.

   (a) Design a representation of a binding environment which supports the following operations:

      i. `init(B)` is true if `B` is an empty binding environment (according to your own definition).

      ii. `get(B, X, Value)` is true if the arithmetical variable `X` has the value `Value` in the binding environment `B`.

      iii. `set(B1, X, Value, B2)` is true if `B2` is the resulting of adding/updating the arithmetical variable `X` to the value `Value` in the binding environment `B1`.

   (b) Define the following predicates:

      i. `evaluate(B, Term, Result)` is true if the result of evaluating the arithmetical term `Term` with respect to the binding environment `B` is `Result`.

      ii. `assign(B1, X, Term, B2)` is true if `B2` is the binding environment resulting from updating the binding environment `B1` by assigning `X` the the value resulting from evaluating `Term` with respect to `B1`.

5. In tutorial 2 this representation for trees with data in the leaves was introduced:

```
tree(l(_)).
tree(t(L, R)) :- tree(L), tree(R).
```

   Define the following relations as Prolog programs:

   (a) `leftmost(Tree,Leaf)` — `Leaf` is the leftmost leaf in `Tree`.

   (b) `rigthmost(Tree,Leaf)` — correspondingly.

   (c) `leaves(Tree,Leaves)` — `Leaves` is a list of the leaves (from left to right, i.e. infix order) in `Tree`. **For an extra challenge:** is it possible to solve this without using `append/3` or any auxiliary list processing predicate? Why, or why not?

   (d) `mirror(Tree1,Tree2)` — the trees are mirror images of each other. Hint: what is the mirror image of `l(X)`, and what is the mirror image of `t(l(X), l(Y))`?

6. In exercise 4 we used Prolog's built-in support for arithmetics, but it is possible to represent numbers in logic programming through terms. For example, we could introduce the atom `zero` to denote the number 0, and then introduce a function symbol s so that the term `s(zero)` represents 1, the term `s(s(zero))` represents 2, and so on. We could then define a predicate which recognises all natural numbers (in this representation) as follows.

```
isnumber(zero).
isnumber(s(N)) :- isnumber(N).
```

Define the following relations as Prolog programs:

(a) `greater(X,Y)` — the number represented by `X` is greater than the number represented by `Y`.

(b) `add(X,Y,Z)` — `Z` is the representation of the sum of the numbers represented by `X` and `Y`.

(c) `mul(X,Y,Z)` — correspondingly for multiplication.