# 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]`.

   Solution.

   (a) The terms are not unifiable. Using the unification algorithm from the lecture slides we in the first iteration obtain the set of equations $\{f(X) \doteq Y, X \doteq f(Z), f(Y) \doteq Z\}$. This set will then be simplified to $\{Y \doteq f(X), X \doteq f(Z), Z \doteq f(Y)\}$. This set is *not* in solved form (try to verify this using the formal definition), and we continue in case 5 of the algorithm with the choice $Y \doteq f(X)$. Since $Y$ occurs in $Z \doteq f(Y)$ on the right-hand-side this case is applicable, and we therefore replace every other occurence of $Y$ with $f(X)$, obtaining the set $\{Y \doteq f(X), X \doteq f(Z), Z \doteq f(f(X))\}$. This set of equations is also not in solved form, since no other case is applicable we continue with case 5 with the choice $Z \doteq f(f(X))$, and replace each occurence of $Z$ with $f(f(X))$, and obtain the new set $\{Y \doteq f(X), Z \doteq f(f(X), X \doteq f(f(f(X)))\}$. Again, this set of equations is not in solved form, and we proceed to case 5 with the choice $X \doteq f(f(f(X)))$. We then observe that $X$ is a proper subterm of $f(f(f(X)))$, and have no other choice than to answer "no" and abort the procedure.

   (b) Yes, these two terms are unifiable, and using the unification algorithm from the lecture slides we will first obtain the equations $\{f(X) \doteq Z, f(Y) \doteq Z, X \doteq W\}$, which will be simplified to $\{Z \doteq f(X), Z \doteq f(Y), X \doteq W\}$. Since this set is not in solved form we pick $Z \doteq f(X)$ and replace every other occurence of $Z$ by $f(X)$, and obtain the set $\{Z \doteq f(X), f(X) \doteq f(Y), X \doteq W\}$. Next, we pick $f(X) \doteq f(Y)$ and replace this equation (case 1) by $X \doteq Y$, and obtain $\{Z \doteq f(X), X \doteq Y, X \doteq W\}$. This set is not in solved form and we proceed to case 5 with the choice $X \doteq Y$, meaning that we replace every other occurence of $X$ with $Y$ and obtain the set $\{Z \doteq f(Y), X \doteq Y, Y \doteq W\}$. Last, we pick $Y \doteq W$ and replace every other occurence of $Y$ with $W$ and obtain the set $\{Z \doteq f(W), X \doteq W, Y \doteq W\}$. This set *is* in solved form and we are therefore done.

*wlodek.drabent@liu.se

†victor.lagerkvist@liu.se

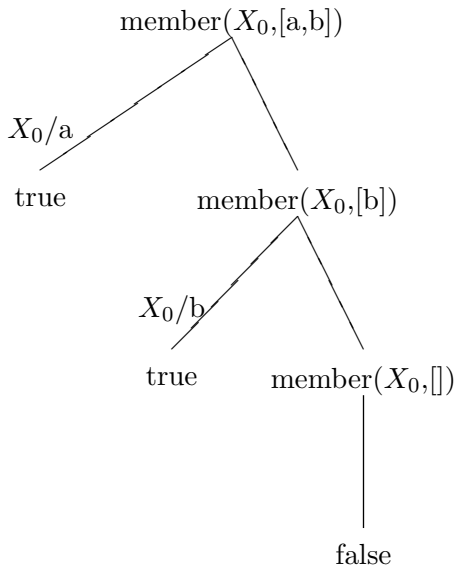


Figure 1: SLD-tree for the query `member(X, [a,b])`. The tree was drawn using the sldnftree library in SWI-Prolog.

(c) Yes, the two terms are unifiable, and if one uses the unification algorithm one obtains an MGU $\{X1 \doteq f(f(a,a), f(a,a)), X2 \doteq f(a,a), X3 \doteq a\}$.

(d) Yes, the two terms are unifiable. First, recall that a term `[A|As]` is only syntactic sugar for `.(A, As)`, and that `[X,Y|Xs]` can be simplified to `[X|[Y|XS]]`. Thus, before we start applying the unification algorithm we translate `[X,Y|Xs]` to `.(X,.(Y, Xs))`. The corresponding translation of `[a,b,c]` results in the term `.(a, .(b, .(c, [])))` where `[]`, the empty list, is treated as a constant symbol. If we then attempt to unify these terms, using the same approach as before, we will obtain the MGU $\{X \doteq a, Y \doteq b, Xs \doteq .(c, [])\}$.

(e) This exercise is in principle not harder than the previous one, but perhaps looks more complicated. We begin by expanding `[X,f(X)|X]` to the corresponding term: `.(X, .(f(X), X))`. We do the same thing with `[Z,Y,Z]` and obtain the term `.(Z,.(Y,.(Z,[])))`. Then we attempt to unify the two terms, using exactly the same approach as before, but discover that two two terms are not unifiable.

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).
```

Solution. See Figure 2. We omit the anonymous variable when drawing the SLD-tree. For the second query `member(a, Xs)` the resulting SLD-tree is infinite. This is due to the fact that `member(a, Xs)` can be resolved either via the fact, in which case we obtain the empty clause, or is resolved via the rule where unify `Xs` with `[_|L]`, and obtain the new goal `member(a, L)`. However, using this goal we can choose to use the rule again, and obtain the new goal `member(a, L0)`, which, again, may lead to a new goal `member(a, L1)`, and so on.

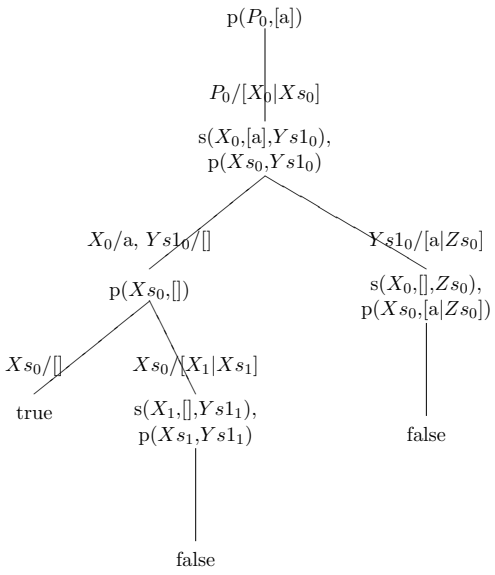


Figure 2: SLD-tree for the query `permutation(P, [a])`. The tree was drawn using the sldnftree library in SWI-Prolog, and the labels for several of the edges should for readability be placed to the left or right of the edges.

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])`.

Solution. See Figure 3 and Figure 3. To make the tree easier to draw we write `p` instead of `permutation`, and `s` instead of `select`.

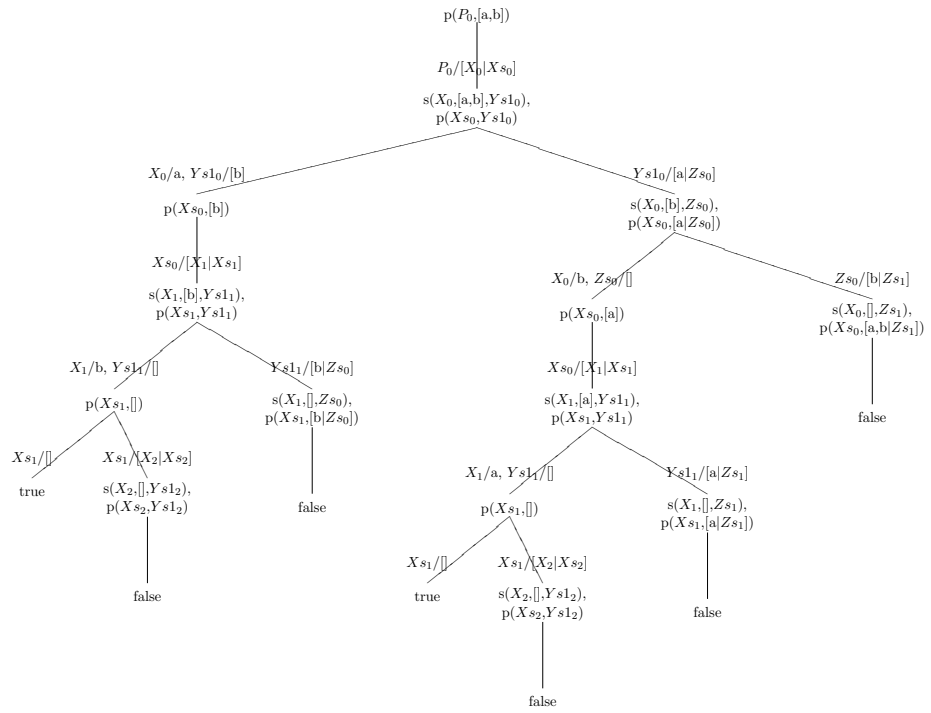


Figure 3: SLD-tree for the query `permutation(P, [a,b])`. The tree was drawn using the sldnftree library in SWI-Prolog, and the labels for several of the edges should for readability be placed to the left or right of the edges.

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`.

   Solution. For the first part, see exercise 2.3 in the lab compendium. For the second part, see the source file associated with this tutorial.

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))`?

   Solution. See the source file associated with this tutorial.

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.

Solution. See the source file associated with this tutorial.