

Assignment09

Due: Sunday, 01.07.2012, 23:59:59 via SVN

Submit your solution for this assignment into your group's SVN as an **Eclipse Project** named "assignment09".
Programming exercises should be submitted as **Prolog files** in this project.
Theoretical exercises should be submitted as **PDF files** in this project.

Intro: Read the following carefully as a preparation for your tasks.

A meta-predicate is a predicate that calls (at least) one of its arguments in its body. The respective argument is said to be a meta-argument. E.g. `my_meta/1` is a metapredicate since its argument "Goal" is called in its body:

```
my_meta_1(Goal) :- ..., call(Goal).
```

An extended definition would also include every predicate that has arguments that are used to construct a term that is invoked via `call/1`. For instance, it would also consider the following examples as metapredicates:

```
my_meta_2(F,A) :- ..., functor(Goal,F,A), ..., call(Goal).
```

```
my_meta_3(List) :- ..., Goal =.. List, ..., call(Goal).
```

```
my_meta_4(Term) :- ..., arg(N,Term,Goal), ..., call(Goal).
```

So, in the general case, a metapredicate's head either shares a variable with the argument of a call in its body or it shares a variable with a term manipulation predicate that constructs the argument of the call. Note that the sharing relation can span an arbitrary number of intermediate literals, e.g. any chain of `functor/3`, `arg/3` and `=../2` invocations can be used to construct "Goal" in extensions of the examples above.

However, if there is no such connection to the head, the predicate is not a metapredicate. E.g. the following ones are not:

```
just_a_call(X) :- call( p(X) ).
```

```
just_a_call(X) :- call(G).% no connection of X and G
```

Task 1. *Metaprogramming* (10 Points)

a) Write a predicate

is_meta_argument(*Functor*,*Arity*, *MetaArgumentNumber*)

that verifies whether the predicate with functor *Functor* and arity *Arity* is a meta-predicate that has a meta-argument at position *MetaArgumentNumber*. That is, you must detect all the four cases introduced on the previous page and yield no false positive results on the two counterexample clauses. E.g.

?- is_meta_argument(*my_meta_1*,1, *MetaArgumentNumber*).

should succeed binding *MetaArgumentNumber* to 1.

Tip: Think of all the cases what can happen in place of “...” (conjunctions, disjunctions, negations, normal literals). Your predicate will look a lot like the meta-interpreters we discussed in the course.

See the file [metapred_finder.pl](#) for the solution to both assignments on this sheet.

Task 2. *Metaprogramming* (4+2 Points)

In SWI-Prolog you can find out about predefined meta-predicates using the call

?- predicate_property(*Head*,meta_predicate(*MetaSpec*)).

If *Head* is the head of a meta-predicate, the call succeeds and binds *MetaSpec* to a term that has the same functor and arity as *Head*. Each meta-argument position in *MetaSpec* is bound to a value *N* from 0 to 9 that indicates that the term passed at that position is extended by *N* arguments before being called. If *Head* is free, the call will enumerate all declared meta-predicates. See the manual entry for the “[meta_predicate/1](#)” declaration for more details.

a) Use the above built_in to write a predicate

is_declared_meta_argument(*Head*,*ArgNumber*,*N*)

that succeeds if *Head* is the head of a declared meta-predicate whose argument position *ArgNumber* is a meta-argument that is extended by *N* additional arguments before being called. Your predicate must succeed once for each meta-argument of *Head*.

b) Write a predicate that uses the one from a) to print out (one per line) all meta-predicates that do *not* extend their meta-arguments before calling them.

Invoke your predicate in a pristine SWI-Prolog process and hand in the output.

See the file [metapred_finder.pl](#) for the solution to both assignments on this sheet.