Metaterms with Several Attributes

Pascal Brisset
ECRC
Arabellastraße 17, D-81925 Munich 81, Germany
brisset@ecrc.de

Abstract

Metastructures and metaterms have been proposed to extend Prolog systems [6, 5]. They enable special purpose unifications to be specified. They are useful to implement for example coroutining and new domains of computation. In both cases, information is attached to variables, which then are encoded with metaterms or metastructures. This information must be taken into account during unification.

Problems appear when one wants to use metaterms for different issues: how to unify metaterms of different kinds with a single unification handler? These problems especially appear in a modular programming with independent pieces of code, where it is impossible to write a single handler for all uses of metaterms hidden in modules. In this paper, we propose a solution where there is only one kind of metaterms which has several facets, each facet serving a special issue. Then each facet is independently unified. This solution has been implemented in the ECRC Prolog system ECL^{i}PS^e.

Keywords Metaterms, Metastructures, Attributed Variable, Extended Unification, Modular Programming.

1 Introduction

Implementing extensions in a Prolog system at the Prolog level needs to extend the standard unification. Tools are needed to extend the unification. The two usual examples are coroutining and extension of the domain of computation. In the former case, the implementation of `freeze/2`, the unification has to *wake* some goals. In the latter case, for example the implementation of finite domains, unification has to solve the equations which define the equality in the new domain.

Neumerkel [6] proposes *metafunctors* which are used to construct *metastructures*. Then, unification of metastructures is handled by a user-defined predicate. Meier [5] proposes another solution he called *metaterms*. In this solution, it is possible to *attach* a term to one variable. When a variable with an attached term has to be unified, a user-defined predicate is called. Holzbaur roughly gives the same solution [2]. The two approaches are based on the same low-level data-structure, the attributed variable [4]. With metastrutures, only the attribute is shown at the Prolog level. With metaterms, both the variable and the attribute can be handled.

Both metastructures and metaterms exhibit problems when they are used for different issues. For example, when metaterms are used to implement coroutining and the finite domains, the user predicate which unifies them has to know the data-structure of the delayed goals and the one of the finite domains.

In this paper, we describe the problem and propose a solution to solve it. Examples are given with the ECL^{i}PS^e syntax.

2 Using Metaterms

2.1 One Example

In his thesis, Holzbaur [1] uses extended unification to implement constraints [3]. He gives a full specification of constraints over finite domains using metastructures. In this scheme he has metafunctors and a specific Prolog predicate to unify them.

Holzbaur needs two metafunctors. The first one, called `$dom/2`, is used to encode finite domains. The second one, called `$fwd/3`, is used to encode the coroutining: the main argument is a list of delayed goals.

When two metastructures are about to be unified, the user-defined predicate `meta_meta_unify` is called. This predicate has to handle the three cases of combining the `$dom/2` and `$fwd/3` metafunctors (`$dom` and `$dom`, `$dom` and `$fwd`, `$fwd` and `$fwd`). When a `$dom/2` term has to be unified with a `$fwd/3` term, both are unified to a new term with a new metafunctor `$dom$fwd/4`. Actually, `meta_meta_unify` has to handle six cases (three new cases with `$dom$fwd`).

2.2 The Problem

There are at least three problems from a software engineering point of view in the Holzbaur's solution:

- Metastructures which serve different purposes are mixed: It is confusing to treat finite domains and delayed goals in the same place.
- There is a lot of cases to treat in metastructure unification: It needs a large piece of code.
- Each time a new kind of metastructure is introduced, the unification handler has to be modified.

These problems come from the fact that there is one single handler. Then, the handler has to know how all the metastructures have to be handled and combined. With such a solution, it is not possible to extend again an extended unification: The original aim is lost.

2.3 A solution

If metaterms are different, they have to be unified by different procedures, but there are problems to combine different kinds of metaterms.

We propose to have only one kind of metaterms which has several facets, each facet serving a special issue. We call these facets of the metaterm the *attributes*. In the Holzbaur's application, there would be two attributes, one for the domain and one for the delayed goals. A metaterm may have one of the two attributes or both.

For metaterm unification, one user-defined predicate is called for each attribute. So there is a generic handler which is called when a metaterm is about to be unified. This *generic* handler parses all the attributes and calls a *local* handler for each of them. Note that a local handler is called for each potential attribute even if the concerned attribute is not set [1].

Each attribute is independently accessed and unified: this means that we want to mix independent extensions which do not interact.

The solution has the following advantages:

- Issues are not mixed: in the Holzbaur's application, waking goals is not mixed with checking if a value belongs to a domain.

[1] It is an arbitrary choice to do the checking in the local handlers and not in the generic one.

- There is a limited number of cases for each local handler. It is only necessary to check if the attribute is set or not (so three cases for each local handler, no attributes set, one attribute set, two attributes set)
- When adding a new attribute, there is no need to modify the existing local handlers.

3 One Handler for Several Attributes

In this section, we give the whole implementation of the proposed solution in the ECL^{i}PS^e system [5]. First, we present the metaterms of the system which have a single attribute. Then, we show how to use them to simulate metaterms with several attributes.

3.1 Syntax of Metaterms in ECL^{i}PS^e

Metaterms can be defined as the attributed variables of Le Huitouze which are lifted at the Prolog level. A metaterm is written:

```
Variable{Attribute}
```

Hence, a predicate which attaches an attribute to a variable is:

```
attach_attribute(Variable, Attribute) :-
  Variable = _{Attribute}.
```

Access to the attribute is done using pattern-matching, an ad hoc mechanism like the `===` of Neumerkel [6] which does "syntactic unification".

As in the solutions proposed by Neumerkel, unification of metaterms leads to a call to a user-defined predicate. The argument of the called goal is a list of pairs: the first argument of the pair is the attribute of the metaterm which is unified with the second argument of the pair (which may also be a metaterm). For example if `meta_unify/1` is the predicate attached to the handler of metaterm unification, the following unification

```
f(M1{A1}, M2{A2}) = f(1, M3{A3})
```

causes the following call

```
meta_unify([[A1 | 1], [A2 | M3]])
```

Note that contrary to the proposals of Neumerkel and Holzbaur, the binding of the metaterm is done before the handler is called.

3.2 Attribute Declaration

We now present the multi attributes solution, implemented on top of the single attribute metaterms.

The user who wants to use metaterms has to declare a *name* for his own attribute. For the Holzbaur's example (constraints over finite domains), we have the following declarations:

```
:- attribute(domain).
:- attribute(delay).
```

respectively for the encoding of domains and coroutining.

3.3 Setting and getting an attribute

In order to have independent attributes in the same metaterm, the data-structure is hidden. This means that the user cannot access directly his attribute but has to *ask* for it using its name.

So we propose two built-in predicates to get and set an attribute:

```
set_attribute(Var, Attr_Name, Attr_Value).
get_attribute(Meta, Attr_Name, Attr_Value).
```

`set_attribute/3` allows the user to add an attribute to a variable or a metaterm. In the variable case, a metaterm is then created. In the metaterm case, the old attribute is overwritten by the new one. `get_attribute/3` allows the user to get his attribute from a metaterm.

For example, the `freeze/2` predicate is written:

```
freeze(X, Goal) :-
  var(X), !,
  set_attribute(X, delay, Goal).
freeze(_X, Goal) :-
  Goal.
```

In a metaterm, an attribute which has not been set is a free variable.

3.4 Unifying metaterms

The user who uses metaterms (and who declares his own attribute) has to write a handler for this attribute. The user of the attribute `x` has to define a handler `x_unify/2`:

```
x_unify(Attribute, Term)
```

This handler is called each time a metaterm is unified. The first argument is the concerned attribute, the second argument is the term the metaterm has been unified with. This term may be a metaterm.

For example, for the coroutining handling, where the attribute is a list of delayed goals, we have the following handler:

```
delay_unify(Attr1, Term) :-
  is_meta(Term), !,     % Unification between two metaterms
  get_attribute(Term, delay, Attr2),
  delay_delay(Attr1, Attr2, Term).
delay_unify(Attr, Term) :-
  delay_term(Attr, Term)
```

The term is checked to know if it is also a metaterm. If so, its attribute is extracted (`get_attribute`). Then the `delay_delay/3` and `delay_term/2` predicates are the `combine_attributes/2` and `verify_attributes/2` of Holzbaur [2]. The difference is that the user handler is always called when a metaterm is unified, even if this metaterm does not contain the concerned attribute. So the first thing the user handler has to do is to check if the attribute has been set or not.

If we want to wake goals only when the variable is instantiated then we have the following code:

```
delay_term(Attr, _Term) :-
  var(Attr), !.              % No attribute
delay_term(Goals, _Term) :-
  wake(Goals).

delay_delay(Attr1, Attr2, Term) :-
  var(Attr1) -> Goals1 = []  % No attribute in the first metaterm
        ; Goals1 = Attr1,
  var(Attr2) -> Goals2 = []  % No attribute in the second metaterm
        ; Goals2 = Attr2,
```

```
append(Goals1, Goals2, Goals),
is_meta(Term) -> set_attribute(Term, delay, Goals)
   ; true.      % The metaterm has been bound by another handler
```

3.5 Implementation Details

An implementation has been done at Prolog level on top of the metaterms of ECL^{i}PS^e.

Data-Structure The single original attribute of metaterms is a compound term (functor `attributes`). Arguments of this compound term are the different attributes. The arity of this compound term is a global variable which is incremented each time a new attribute is declared. A table maintains the association between the attribute names and the corresponding fields in the compound term.

For example, the following sequence:

```
attribute(delay), attribute(domain), set_attribute(X, domain, [1, 2])
```

gives the following term:

```
X{attributes(_Free, [1, 2])}
```

Access to Attributes The association name-field is known as soon as the attribute is declared. Then it is possible to use at compile-time the association table for the `get` and `set` commands: using a macro mechanism, a name of an attribute is replaced by the associated index in the compound term. Hence, there is no overhead at run-time.

For example

```
get_attribute(X, domain, Dom)
```

is replaced at compile-time by

```
get_the_attribute(X, A), arg(2, A, Dom)
```

where `arg` just get the right field in the compound term.

Generic Handler The handler which is called by the ECL^{i}PS^e kernel when a metaterm is unified, is now the generic handler which calls the local handlers. The simplest way to implement is to have a loop over the attributes. In order to avoid some overheads in the control of the loop, this generic handler is regenerated and compiled each time a new attribute is declared: the loop is compiled into the conjunction of calls to the local handlers.

4 Conclusion

We have shown that extending unification with metaterms (or metastructure) is not straight-forward as soon as it is necessary to implement independent extensions. A lot of problems are encountered if a single metaterm unification handler is used for metaterms with several purpose.

We have proposed a solution where metaterms have several independent attributes where a local handler is dedicated to each attribute. With this solution, information encoded in metaterms which serve different issues is not mixed, and it is possible to add new attributes without changing the existing handlers. This solution has been tried in the ECRC Prolog System ECL^{i}PS^e to handle finite domains and coroutining.

Acknowledgement The author wants to thank J. Schimpf, M Meier and M. Wallace for fruitful discussions, M. Ducassé and C. Gervet for her helpful comments.

References

[1] C. Holzbaur. Specification of constraint based inference mechanisms through extended unification. Dissertation, University of Vienna, 1990.

[2] C. Holzbaur. Metastructures vs. attributed variables in the context of extensible unification. In M. Bruynooghe and M. Wirsing, editors, *4th Int. Work. Programming Languages Implementation and Logic Programming, LNCS 631*. Springer-Verlag, 1992.

[3] J. Jaffar and S. Michaylov. Methodology and implementation of a CLP system. In J.L. Lassez, editor, *4th Int. Conf. Logic Programming*. MIT Press, 1987.

[4] S. Le Huitouze. A new data structure for implementing extensions to Prolog. In P. Deransart and J. Małuszyński, editors, *2nd Int. Work. Programming Languages Implementation and Logic Programming, LNCS 456*, pages 136–150. Springer-Verlag, 1990.

[5] M. Meier and J. Schimpf. An architecture for Prolog extentions. In E. Lemma and P. Mello, editors, *Third Int. Workshop on Extensions of Logic Programming, LNAI 660*, pages 319–338. Springer-Verlag, 1992.

[6] U. Neumerkel. Extensible unification by metastructures. In M. Bruynooghe, editor, *2nd Workshop on Meta-Programming in Logic Programming*, pages 352–363, Leuven, Belgium, 1990. K.U. Leuven, Dept. of Computer Science.