Metastructures vs. Attributed Variables in the Context of Extensible Unification

Christian Holzbaur

Austrian Research Institute for Artificial Intelligence, and
Department of Medical Cybernetics and Artificial Intelligence
University of Vienna
Freyung 6, A-1010 Vienna, Austria
email: christian@ai.univie.ac.at

Abstract. We relate two mechanisms which aim at the extension of logic programming languages. The first mechanism directly extends syntactic unification through the introduction of a data type, whose (unification) semantics are specified through user-defined predicates. The second mechanism was utilized for the implementation of coroutining facilities, and was independently derived with optimal memory management for various Prolog extensions in mind. Experience from the application of both mechanisms to the realization of CLP languages, without leaving the logic programming context, enables us to reveal similarities and the potential with respect to this task. Constructive measures that narrow or close the gap between the two conceptual schemes are provided.

1 Introduction

As a serious user of two rather similar mechanisms — as far as their applications are concerned — we think that it is useful to expose this similarity in some detail. Both mechanisms provide means for the extension of logic programming languages.

Metastructures as introduced by Neumerkel (1990) aim at extensions to Prolog's builtin unification through user-defined behavior of metastructures during unification. A refined version of the concept of metastructures was used in (Holzbaur 1990) for the specification and implementation of a variety of instances of the general CLP scheme (Jaffar and Michaylov 1987).

More or less at the same time, the data type attributed variable was introduced by Hoitouze. Memory management issues as early reset and variable shunting by the garbage collector were addressed in (Hoitouze 1990). The behavior of attributed variables during unification was not mentioned. However, regarding applications, Hoitouze also proposed the use of attributed variables for the implementation of delayed computations, reversible modification of terms, variable typing, and others.

Earlier, Carlsson (1987) used a data type suspension, which was incorporated into SICStus Prolog (Carlsson and Widen 1990) for the implementation of coroutining facilities. As far as we can tell — as a third party — the data structures attributed variable and suspension are the same. The difference between Hoitouze's and Carlsson's exposition is that the former put some emphasis on the data type as such and on memory management. The latter used it as a low level primitive for the
implementation of mechanisms that necessitated the specification of the behavior of the data type during unification.

In the following sections we will have a closer look on metastructures and attributed variables. In particular, we compare them with regard to their behavior during unification and their potential for the implementation of CLP languages.

2 Metastructures

Metastructures are ordinary, non-variable Prolog terms with the sole difference that they can be detected as members of this special sort. Metastructures are introduced by a declaration :- meta functor N/A, where N/A denotes any functor. The behavior of metastructures during unifications can be specified precisely through a Prolog meta interpreter which makes unification explicit (Holzbaur 1990). The meta interpreter implements the unification table from (Neumerkel 1990) and makes some further conventions integral parts of the specification:

- Unifications between variables and metastructures just produce a binding as usual. If a metastructure is to be unified with an ordinary term, the reaction to this event is given by a user-supplied predicate meta term unify/2. Similarly, unifications between two metastructures are covered by the user-supplied predicate meta meta unify/2, the arguments being the two metastructures involved.
- Once extensible unification is put into force, we have a problem passing metastructures to the user-supplied predicates, without triggering further calls to them in a nonterminating fashion. Neumerkel solved the problem by the introduction of a builtin predicate ===/2, which behaves as =/2, but treats metastructures as ordinary structures. In addition he has to rely on the programmers discipline: Nothing but variables may be used as formal arguments in the definition of the two user-supplied predicates. Access to the components of metastructures is via ===/2. The disadvantages of this solution are that the user has to be very careful, and that indexing does not apply.
Therefore, we specified the following mechanism: Calls to the two user-supplied predicates meta term unify/2 and meta meta unify/2 are made with syntactic unification in force. The encapsulation effect of this solution is at least as strong as the one with ===/2, as the only means to get access to the 'internals' of a metastructure.
- The last part of the specification stemmed from a typical application of metastructures in the context of the implementation of CLP languages. It is covered in detail in the next section.

2.1 Metastructures and Reversible Modification

The implementations of many CLP instances require the functionality that is achieved with destructive updates in traditional, procedural realizations. In logic programming, this functionality is provided in a sound fashion by either copying or modifying by variable substitution.

The latter option can be applied to metastructures through the convention that one particular argument of the structure is a free variable, which will eventually be