A Constructive Type System Based on Data Terms

Hans-Joachim Goltz
German National Research Center for Computer Science
GMD–FIRST, Rudower Chaussee 5
D – 1199 Berlin, Germany

Abstract
The syntax and semantics of a new kind of a type system are described and type unification is discussed. Special algebraic structures are used for the semantics. The universe of such a structure consists of ground data terms (for the representation of the ground objects) and of ground type terms (for the representation of the types). A binary function "'" is used to determine the elements belonging to a type. This function is defined by special rewrite rules called type rules. The main aim of this approach is computing with types. The type unification can generate the elements belonging to the intersection of two types.

1 Introduction
The syntax and semantics of a new kind of type system are described in the present paper, a unification algorithm for the unification of typed terms is given, and properties of this algorithm are discussed. In the suggested type system, types are subsets of the universe of discourse and are defined by type rules. The elements belonging to a type can be generated by using these type rules.

In general, there are several objectives for introducing types, e.g. type checking (avoiding useless inferences, avoiding program errors), optimized compilation, definition of data structures, computing with types (as an extension of type inference). In our approach, the main aim is computing with types. Especially, data terms can be generated during the inferences. The other objectives mentioned above are supported, too.

Structuring the universe of discourse is a successful method in the field of representing knowledge and automated reasoning. The universe is subdivided into a certain number of subuniverses. Subuniverses may contain one another, may partially overlap, or may be disjoint. Relations between subuniverses form a special kind of information, sometimes called taxonomic information, which helps to avoid
meaning-less or wrong conclusions. The benefit of using such information is widely recognized within the field of knowledge representation and automated theorem proving (see e.g. [21],[26]). The introduction of types and sorts provides an appropriate means for such a structuring of the universe. Especially, the incorporation of types or sorts into logic programming is of great interest. Many research activities have been devoted to this task and several proposals for various type systems have been made. The majority of the approaches are based on many-sorted logic or order-sorted logic. In these approaches either a common universe of all ground elements is assumed (then types are interpreted by subsets of this domain) or there are different domains for different types.

In the following some important papers on type systems in logic programming are shortly reviewed. Mycroft and O'Keefe ([22]) have proposed a polymorphic type system for Prolog based on many-sorted logic. A formal semantics of this approach is discussed in [18]. Different domains are associated with distinct types and type declarations form an integral part of programs. Subset relations between types are not allowed. A typed functional extension of Logic programming is defined in [23]. This language is essentially based on Horn clause logic with equality and a polymorphic type system that is an extension of Mycroft and O'Keefe's system.

In order-sorted type systems different types may be related by an inclusion relation ([24]). OBJ is a logic programming language based on order-sorted equational logic ([17]). Programs are order-sorted equational specifications and computation is a form of equational deduction by rewriting. Logic Programming with polymorphically order-sorted types is proposed in [25]. Type functions must be monotonic in their arguments with respect to inclusion order. In [13] the subtype order is defined by Horn clauses and the monotony of type functions is not required. PROTOS-L ([2]) is a logic programming language based on the foundations given in [25].

In [1] another direction of an approach for types in logic programming is given. Ordinary first-order terms are replaced by record structures and a special type unification is used. A further direction is proposed in [6] and [5]. Types are defined by functions on terms (by means of equations) and it is shown that order-sortedness can easily be expressed within the framework of equational logic programming. Equations are also used for the type system in [14], but there is a clear distinction between types and functions (and predicates). The subsort relationship is described by equations and the type structure is specified by a many-sorted signature with equational axioms.

The basic idea of a type system discussed in this paper is very similar to the approach given in [6] and [5], respectively. Types can be defined by special rewrite rules (called type rules). In contrast to [6] and [5], a special semantics and a special type unification are defined. Our main aim is the definition of subsets of the universe by means of type rules such that the elements belonging to a subset can be generated by these rewrite rules, and computing with such subsets. These subsets are regarded as types.

In the following we discuss how rewrite rules can be used for the definition of types. For instance, the type of natural numbers can be defined by the following rewrite rules: