Correctness proof for the WAM with types

Christoph Beierle  
IBM Germany, Scientific Center  
Inst. for Knowledge Based Systems  
P.O. Box 80 08 80  
D-7000 Stuttgart 80, Germany  
beierle@dsr

Egon Börger  
Dipartimento di Informatica  
Università di Pisa  
Corso Italia 40  
I-56100 Pisa, Italia  
boerger@dipisa.di.unipi.it

Abstract: We provide a mathematical specification of an extension of Warren's Abstract Machine for executing Prolog to type-constraint logic programming and prove its correctness. In this paper, we keep the notion of types and dynamic type constraints rather abstract to allow applications to different constraint formalisms like Prolog III or CLP(R). This generality permits us to introduce modular extensions of Börger's and Rosenzweig's formal derivation of the WAM. Starting from type-constraint Prolog algebras that are derived from Börger's standard Prolog algebras, the specification of the type-constraint WAM extension is given by a sequence of evolving algebras, each representing a refinement level. For each refinement step a correctness proof is given. Thus, we obtain the theorem that for every such abstract type-constraint logic programming system L and for every compiler satisfying the specified conditions, the WAM extension with an abstract notion of types is correct w.r.t. L. This is a first step towards our aim to provide a full specification and correctness proof of a concrete system, the PROTOS Abstract Machine (PAM), an extension of the WAM by polymorphic order-sorted unification as required by the logic programming language PROTOS-L.

1 Introduction

Recently, Gurevich's evolving algebra approach ([11]) has not only been used for the description of the (operational) semantics of various programming languages (Modula-2, Occam, Prolog, Prolog III, Smalltalk, Parlog, C; see [10]), but also for the description and analysis of implementation methods: Börger and Rosenzweig ([7,8]) provide a mathematical elaboration of Warren's Abstract Machine ([16], [1]) for executing Prolog. The description consists of several refinement levels together with correctness proofs, and a correctness proof w.r.t. Börger's phenomenological Prolog description ([5,6]).

In this paper we demonstrate how the evolving algebra approach naturally allows for modifications and extensions in the description of both the semantics of programming languages as well as in the description of implementation methods. Based on Börger and Rosenzweig's WAM description we provide a mathematical specification of a WAM extension to type-constraint logic programming and prove its correctness. Note that thereby our treatment covers also all extra-logical features (like the Prolog cut) whereas the WAM correctness proof of [12] deals merely with SLD resolution for Horn clauses.

The extension of logic programming by types requires in general not only static type checking, but types are also present at run time. For instance, if there are types and subtypes, restricting a variable to a subtype represents a constraint in the spirit of constraint logic programming. PROTOS-L ([2]) is a logic programming language that...
has a polymorphic, order-sorted type concept (derived from the slightly more general type concept of TEL [15], [14]) and a complete abstract machine implementation, called PAM ([13], [4]) that is an extension of the WAM by the required polymorphic order-sorted unification.

Our final aim is to provide a full specification and correctness proof of the concrete PAM system ([3]). In this paper, we keep the notion of types and dynamic type constraints rather abstract to allow applications to different constraint formalisms. Starting from type-constraint Prolog algebras that are derived from Börger's standard Prolog algebras, the specification of the type-constraint WAM extension is given by a sequence of evolving algebras, each representing a refinement level. We state precisely where no changes w.r.t. the WAM are needed (for instance the AND/OR structure can be taken essentially unchanged) and where - mostly orthogonal - extensions are required (in particular in the representation of terms). For each refinement step a correctness proof is given. As final result of this paper we obtain the main theorem: For every such abstract type-constraint logic programming system L and for every compiler satisfying the specified conditions the WAM extension with an abstract notion of types is correct w.r.t. L.

Although our description in this paper is oriented towards type constraints, it is modular in the sense that it should carry over to other constraint formalisms, like Prolog III or CLP(R), as well. Nevertheless, in order to avoid proliferation of different classes of evolving algebras, we will speak here in terms of PROTOS-L and PAM algebras (instead of type-constraint Prolog and type-constraint WAM algebras).

The paper is organized as follows: In Section 2 the derivation of PROTOS-L algebras from standard Prolog algebras is described. Section 3 defines PROTOS-L algebras with compiled AND/OR structure, and Section 4 introduces the representation of terms. The stack representation of environments and choicepoints is given in Section 5 which also contains the "Pure PROTOS-L" theorem stating the correctness of the PAM algebras developed so far w.r.t. the PROTOS-L algebras of Section 3. The notions of type constraint and constraint solving have been kept abstract through all refinement levels so far; thus, the development carried out in this paper applies to any type system satisfying the given abstract conditions. The introduction of PROTOS-L specific type constraint representation is carried out in [3].

To keep this paper within reasonable limits, we definitely suppose that the reader is familiar with [6] and [7,8] to which we will refer for many definitions and notations, including those concerning evolving algebras. The natural modularity of our approach will come out from presenting in this paper only those extensions and refinements which are needed in the presence of types. Due to severe space restrictions, examples could not be included in this paper; we refer the reader to [3].

2 PROTOS-L Algebras

2.1 Universes and Functions

The basic universes and functions in PROTOS-L algebras can be taken directly from the standard Prolog algebras ([5], [6]). In particular, we have the universes TERM and SUBST of terms and substitutions with a function

\[ \text{subres}: \quad \text{TERM} \times \text{SUBST} \to \text{TERM} \]

yielding \( \text{subres}(t,s) \), the result of applying \( s \) to \( t \).