© 1996 BCS

Specification and Correctness Proof of a WAM Extension with Abstract Type Constraints*

Christoph Beierle¹ and Egon Börger²
¹Fachbereich Informatik, FernUniversität Hagen, Germany; ²Dipartimento di Informatica, Università di Pisa, Pisa, Italy

Keywords: Constraint logic programming; Abstract machine; WAM; Types; Type constraint; Correctness proof; Evolving algebras

Abstract. We provide a mathematical specification of an extension of Warren's Abstract Machine (WAM) for executing Prolog to type-constraint logic programming and prove its correctness. Our aim is 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.

In this paper, while leaving the details of the PAM’s type constraint representation and solving facilities to a sequel to this work, we keep the notion of types and dynamic type constraints 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. Since the type constraint handling is orthogonal to the compilation of predicates and clauses, we start from type-constraint Prolog algebras with compiled AND/OR structure that are derived from Börger’s and Rosenzweig’s corresponding compiled standard Prolog algebras. The specification of the type-constraint WAM extension is then given by a sequence of evolving algebras, each representing a refinement level, and 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, every compiler to the WAM extension with an abstract notion of types which satisfies the specified conditions, is correct.

Correspondence and offprint requests to: Christoph Beierle, Fachbereich Informatik, LG Praktische Informatik VIII, FernUniversität Hagen, Bahnhofstr. 48, D-58084 Hagen, Germany; e-mail: christoph.beierle@fernuni-hagen.de

* The first author was partially funded by the German Ministry for Research and Technology (BMFT) in the framework of the WISPRO Project (Grant 01 IW 206). He would also like to thank the Scientific Center of IBM Germany where the work reported here was started.
1. Introduction

Recently, Gurevich’s notion of evolving algebra [Gur88] 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 [Gur91]), but also for the description and analysis of implementation methods: Börger and Rosenzweig [BöR91, BöR92b, BöR92a] provide a mathematical elaboration of Warren’s Abstract Machine [War83, AiK91] 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 [Bör90a, Bör90b].

In this work 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 can be easily extended to cover also all extra-logical features (like the Prolog cut) whereas the WAM correctness proof of [Rus92] 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 (see e.g. [MyO84, GoM86, NaM88, Han88, Han91, Smo89]). 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 [Bei92, BBM91, Bei95] is a logic programming language that has a polymorphic, order-sorted type concept (similar to the slightly more general type concept of TEL [Smo88]) and a complete abstract machine implementation, called PAM [BMS91, BeM94] that is an extension of the WAM by the required polymorphic order-sorted unification. Our aim is to provide a full specification and correctness proof of the concrete PAM system.

Here we keep the notion of types and dynamic type constraints sufficiently abstract to allow applications to different constraint formalisms. Since the type constraint handling is orthogonal to the compilation of predicates and clauses, we start from type-constraint Prolog algebras with compiled AND/OR structure that are derived from Börger’s and Rosenzweig’s corresponding compiled standard Prolog algebras. The specification of the type-constraint WAM extension is then given by a sequence of evolving algebras, each representing a refinement level. For each refinement step a correctness proof is given. As a final result of this paper we obtain the theorem: For every such abstract type-constraint logic programming system L and for every compiler satisfying the specified conditions, compilation from L to the the WAM extension with an abstract notion of types is correct.

Although our description in this paper is oriented towards type constraints, it is modular in the sense that it can be extended to other constraint formalisms, like Prolog III [CoI90] or CLP(R) [JaL87, JMS90], as well. For instance, in [BöS95] a specification of the CLAM, an abstract machine for CLP(R), is given along these lines, together with a correctness proof for CLP(R) compilation. [Bei94] extends the work reported here by studying a general implementation scheme for CLP(X) and designing a generic extension WAM(X) of the WAM. Nevertheless, in order to avoid proliferation of different classes of evolving algebras, we will already speak here in terms of PROTOS-L and PAM algebras (instead of type-constraint Prolog and type-constraint WAM algebras).