A Knowledge-Based Type System for Computer Algebra

Indra A. Tjandra, Karsten Homann[†], Jacques Calmet[†]

Concordia University
Department of Computer Science
Montreal PQ H3G 1M8, Canada
ono@cs.concordia.ca

[‡] Universität Karlsruhe
Institut für Algorithmen und Kognitive Systeme
Postfach 6980 · 76128 Karlsruhe · Germany
homann@ira.uka.de, calmet@dkauni2.bitnet

Extended Abstract

A novel framework, Formal, for specifying mathematical domains of computation and their inherently related type inference mechanisms as well as for transforming those specifications into knowledge bases is introduced. This framework [Tja] aims at designing an environment for reasoning about knowledge in symbolic computing. It involves an algebraic specification language, a method to transform specifications into knowledge bases and a hybrid knowledge representation system as well, cf. figure 1.

The specification language Formal-Σ [CT93] provides modular and well-structured specifications. It is well-suited to specify *"mathematical objects"* and, particularly, to specify the parametric and the inclusion polymorphisms in a unified way. The underlying formalism is based upon the so-called homogeneous *" unified algebras"* allowing the treatment of sorts as values. Since algebraic specifications are the most appropriate formalism for embodying abstract algebras, e.g. group, ring, field or module, and concrete algebras, e.g. polynomial rings, vector spaces or matrices, this specification language can be regarded in the context of knowledge acquisition.

There are many motivations and approaches to the executability of algebraic specifications. Our approach is among the compilational ones. It consists in compiling a term to a representation in some model, e.g. the execution model is the semantics of the language of a hybrid knowledge representation. Such a language is of particular interest as an execution model since its data domain is the algebra of terms.

In view of achieving the executability of specifications we have also developed the transformation method Formal-Θ providing a capability for compiling a non-executable specification into an executable one, i.e. into a knowledge base that can be processed by the inference machine of the hybrid knowledge representation system Mantra [CTB91]. In Mantra four different knowledge representation formalisms are integrated: First-order logic, terminological languages, semantic networks and production systems.

The role of Formal-χ consists in processing queries given by the user, e.g. to simplify a term or to define the type of an expression.

In this extended abstract we give an overview of Formal-Σ, Mantra and Formal-Θ. For the sake of clarity and of simplicity technical details are omitted.

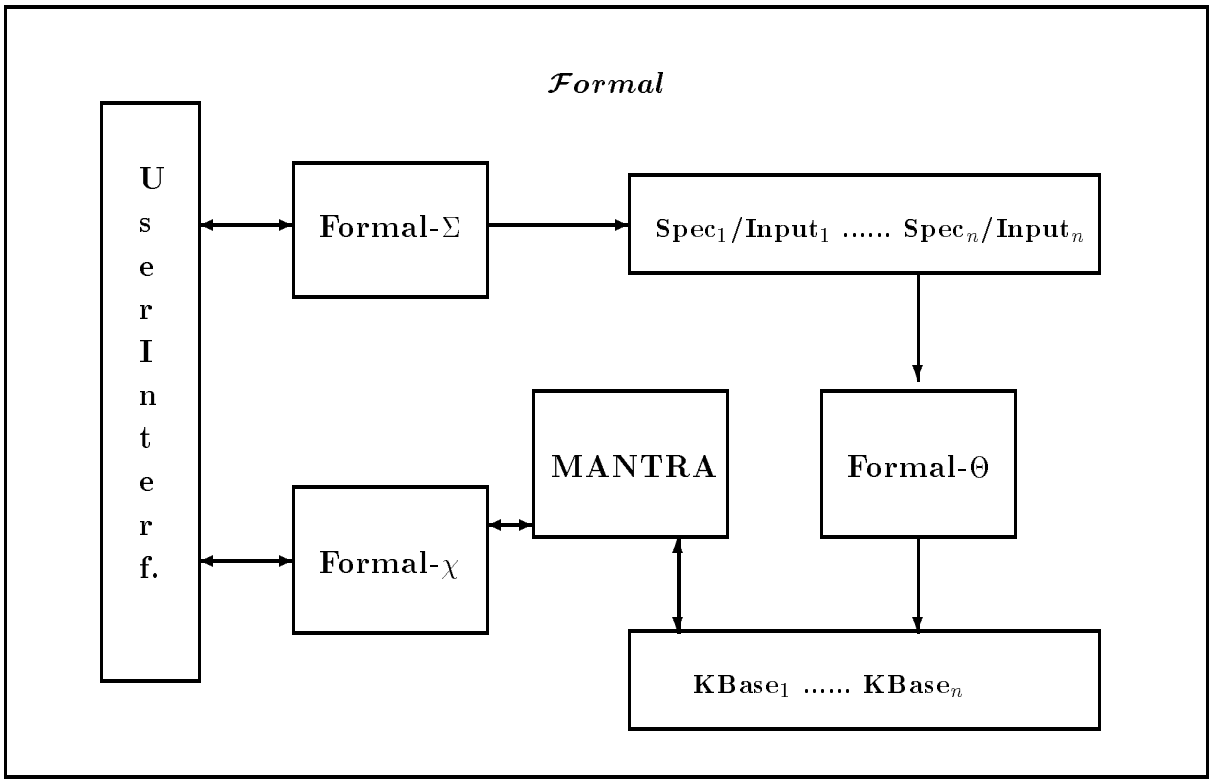

Figure 1: Overview of FORMAL

FORMAL-Σ

The need of specifying *mathematical domains of computation* in symbolic computing arises from the fact that correct nontrivial computations are performed in well-defined proper domains.

The main goal of designing FORMAL-Σ consists in providing a tool for the specifications of mathematical domains, taking into account the properties of function symbols, as well as for the specifications of type inference mechanisms involving parametric and inclusion polymorphism[1].

The following examples give a brief overview of the specifications of `semi group`, `monoid` and `group`. The properties of function symbols are represented in the body of **Clauses**. A specification is represented by a module.

```
(Module SemiGroup
 (Define (Constants SemiGroup)
         (Operations (o (SemiGroup SemiGroup) -> SemiGroup))
         (Clauses (Imply (: (a b c) SemiGroup)
                         (= (o (o a b) c) (o a (o b c)))))))
```

[1] We use the notions of parametric and inclusion polymorphism according to those introduced in [CW85].

```
(Module Monoid
 (Union (SemiGroup)
        (Define (Constants (=< Monoid SemiGroup)
                           (: Neutral Monoid))
                (Clauses (Imply (: a Monoid) (and (= (o Neutral a) a)
                                                  (= (o a Neutral) a)))))))

(Module Group
(Union (Monoid)
       (Define (Constants (=< Group Monoid))
               (Operations (inv Group -> Group))
               (Clauses (Imply (: (a b) S) (and (= (op (inv a) a) Neutral)
                                                (= (op a (inv a)) Neutral)))))))
```

The Module `SemiGroup` is specified as a *basic module* possessing the constant **Semi Group**. It also possesses the function symbol `o` that is represented in the **Operation** part together with its functionality. The property of the function symbol is expressed by means of Horn Clauses with equality. The module construct **Union** is used to embed modules into a particular module.

First of all, we want to depict the major prerequisites for achieving our goal. Accordingly, we have to provide a suitable formalism that can be adopted in order to design FORMAL-Σ.

Basically, the initial step of constructing a mathematical domain of computation comprises the determination of its *abstract specification* consisting of the declarations of the sorts, operations and properties under consideration. Such specifications of abstract mathematical domains can rely on the definition of abstract structures as, usually, found in any text-book on abstract algebra. Regarding these abstract structures, e.g. the abstract structure of `Module` that is based on additive Abelian group and possesses `Ring` as its formal parameter, it turns out that the specification language to be designed should be able to handle *structured* (or *modularly-built*) and *parameterized* specifications.

Moreover, it is very convenient to make use of properties imposed on an abstract structure, e.g. in order to draw a conclusion[2], whether two ground terms have the same value or two terms are equal in the initial algebra of the specification, or in order to find the value of a given term, etc. Such requirements illustrate why the executability of specifications could be very helpful.A feasible way to realize this task consists in *transforming* the specifications into a particular executable form, e.g. transformation of ASF-specifications into Prolog clauses [BHK89]. Hence, we have to take into consideration that the formalism of FORMAL should be tailored in such a way, or it should be *operational* enough, so that it allows a straightforward transformation.

An other important aspect to design the specification language concerns the *inclusion polymorphism* and also the *type inference engine*, as specifications of abstract mathematical domains and their interrelationships represent hierarchies involving multiple inheritance. Thus, the specification of a semantics of inclusion polymorphism involves operations that map sorts to sorts, e.g. sort constructors. Hence, we need a specification formalism allowing sorts to be treated as values. The main idea consists in representing the carrier of a structure including sorts not only elements of data, e.g. homogeneous algebras.

[2] Assuming that the set of properties is complete.

The prerequisites cited above motivated us to adopt the framework for algebraic specification involving the so-called *"unified algebras"* [Mos89]. FORMAL-Σ is intended to be used as a tool for specifying mathematical domains of computation which are embodied as parameterized modules. The semantics of FORMAL-Σ is determined by a meaning function mapping models into models. This kind of function is also used in EXTENDED-ML. In contrast to the subsort concept, treated in OBJ, we deal with subsorts using the partial ordering defined in unified algebras allowing the specification of type hierarchies with multiple inheritance and type constructions. This is required in symbolic computing. In order to describe a method to tranform a specification to a knowledge base of MANTRA we shall give a brief overview of MANTRA in the next section.

MANTRA

MANTRA (Modular Assertional, Semantic Network and Terminological Representation Approach) is a Shell for knowledge systems [Bit90]. The main characteristics of MANTRA are: (i) The introduction of a multilevel architecture for hybrid systems together with a methodology to define a unified semantics for knowledge representation methods and their interaction, (ii) The integration of two features which are usually not found in hybrid systems: Inheritance with exception and heuristic programming, (iii) The extension of the Frame terminological language to accept n-valued relations instead of binary roles only, (iv) The semantic definition of non-monotonic inheritance with exceptions using the four-valued logic approach, (v) All algorithms for inference procedures are decidable, (vi) High interaction among the different knowledge representations covered by the system, (vii) Unifying four-valued semantics.

First order logic offers a very powerful inference procedure allowing to generate all of the entailed (i.e. implicitly represented) knowledge by a given amount of explicitly represented knowledge. The problem of verifying if a given element of knowledge is entailed by a certain amount of explicit knowledge is, however, not decidable in first order logic. To avoid this problem we adopt a *weak* inference procedure and a query procedure which always terminates. They were introduced by Patel-Schneider [PS90]. These procedures are semantically defined through a four-valued logic approach.

Our frame method is a terminological language as in Brachman [Br et al.] But it extends its capabilities by accepting n-valued relations. The main inference procedure is the subsumption procedure. A concept or relation subsumes another one if the former includes the latter.

Semantic networks are methods based on the abstraction of nodes and edges. The method adopted in our approach allows to define hierarchies with exception. The inheritance procedure is the main inference procedure available in this method. It allows to verify if a class is a subclass of another one given an explicit hierarchy consisting of either default or exception links. The so-called skeptical inheritance approach has been selected within, again, the four-valued semantic approach.

These three epistemological methods are not very powerful when standing alone because of the adoption of the four-valued semantics which weakens the deductive power of the system. But, this choice was mandatory in order to have only decidable inference algorithms and a semantically sound system. The deductive power is, however, very much increased by the association of the three methods. This association is performed through the definition of hybrid inference procedures allowing to generate new knowledge from those acquired through the three methods separately. This interaction is defined not by the inference procedures

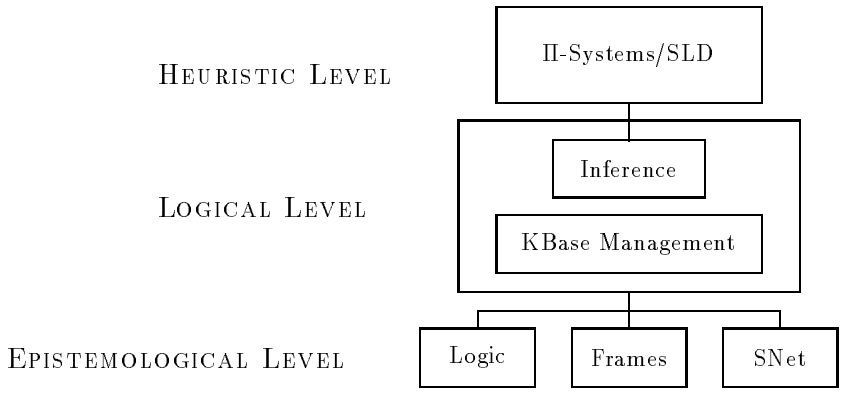

Figure 2: Architecture of Mantra

themselves but by the semantic specification of the results that these procedures should provide.

The architecture of Mantra is defined along three levels: the epistemological level, the logical level and the heuristic level as shown in figure 2.

The epistemological level includes the three methods described above and it is extendible. Each of the corresponding module is thus constructed around some epistemological notions: predicates, functions and constants in the logic module, concepts and relations in the frames module, classes and hierarchies in the semantic net module. Syntactically, each one of these modules consists of a set of primitives which are used to manipulate the epistemological notions. The four-valued semantics enables to model ignorance and inconsistency and thus provides a semantics for an incomplete inference mechanism.

The second level introduces the concept of knowledge base. Two primitives are used to store facts and to interrogate knowledge bases respectively. There are eight possible interactions among modules: logic-frame, logic-semantic networks (SN), frame-SN, SN-logic, SN-frame, logic-frame-SN, frame-logic-SN and SN-logic-frame. All these interactions have been semantically defined and algorithms have been proposed and proven sound and complete with respect to the semantic definition.

The third level is the heuristic level. It consists of primitives allowing the definition of production systems and Horn clauses taking into account all formalisms provided by the logical level.

In the following section will shall give an overview on transforming a specification to a knowledge base of Mantra.

Formal-Θ

A transformation is defined over the basic modules in the sense that modules possessing module constructs, e.g. `union`, must initially be converted to semantically equivalent basic modules by using the underlying semantic functions.

A specification is transformed into a knowledge base according to its syntax tree in a bottom up manner. Each transformation step makes use of a particular transformation rule

represented in the following form:

$$\frac{I}{\mathbf{R}\quad 0} \left\langle \begin{array}{l} C_1 \\ \cdots \\ C_n \end{array} \right.$$

The above form is also equivalent to the following one:

$$\frac{\{C_1, \cdots, C_n\}}{\vdash R[I, O]}$$

The intended meaning of such a rule is the following. I and O are called input and output scheme respectively. I is part of a specification in FORMAL-Σ and O is the corresponding representation of I in the language of MANTRA. A program scheme is a term from $W(PL \cup X)$, the term algebra over $PL \cup X$, i.e. a term over PL (programming language) containing free variables from a countable set X of typed scheme parameters. $C_1 \cdots C_n$ are applicability conditions which are Horn clauses over an enrichment of $PL \cup X$, i.e. they may contain additional syntactic and semantic predicates over program schemes.

A transformation rule is correct if it constitutes a valid inference, i.e. if the program schemes I and O are in the semantic relation indicated by R whenever the applicability conditions are valid.

The transformation of a specification can be outlined as follows:

- The signature coinciding with the main construction of a specification consisting of the module identifier, formal parameters and the function symbols are represented by means of frames provided at the epistemological level of MANTRA.
- The carrier, that is a distributive lattice (in a unified algebra), is also modelled by frames.
- The Horn clauses imposed on the specification is represented by Horn clauses in MANTRA at the heuristic level. As MANTRA does not allow equations we extend the approach to integrating functional programming into logic programming proposed by van Emden and Yukawa [VY87] in such a way, that equations can also be treated in MANTRA.

The correctness of the transformation of a specification, i.e. the specification and its corresponding representation in MANTRA are semantically equivalent, is verified by giving as proof for each transformation rule that there is a morphism from I to O according to the semantic predicate R.

References

[BHK89] J.A. Bergstra, J. Heering, and P. Klint. *Algebraic Specification.* Addison-Wesley Publising Company, 1989.

[Bit90] G. Bittencourt. *An Architecture for Hybrid Knowledge Representation.* PhD thesis, Universität Karlsruhe, Institut für Algorithme und Kognitive Systeme, 1990.

[CT93] J Calmet and I.A. Tjandra. A unified-algebra-based specification language for symbolic computing. In A. Miola, editor, *Design and Implementation of Symbolic Computation Systems*. To appear in LNCS Springer-Verlag, 1993.

[CTB91] J. Calmet, I.A. Tjandra, and G. Bittencourt. Mantra: A shell for hybrid knowledge representation. In *IEEE-Conference on Tools for AI*, pp 164 –171. IEEE, IEEE Computer Society Press, 1991.

[CW85] L. Cardelli and P. Wagner. On understanding types, data abstraction and polymorphism. *Computing Survey*, 17(4), pp 471 – 522, 1985.

[Mos89] P.D. Mosses. Unified algebras and institutions. In *Logics in Computer Science*, pp 304 – 312. IEEE Press, 1989.

[PS90] P.F. Patel-Schneider. A decidable first-order logic for knowledge representation. *Journal of Automated Reasoning*, 6, pp 361 – 388, 1990.

[Tja] I.A. Tjandra. *Algebraic Specification of Mathematical Domains of Computation and Polymorphic Types in Computer Algebra*. PhD thesis, Universität Karlsruhe, Institut für Algorithme und Kognitive Systeme, 1993.

[VY87] M.H. VanEmden and K. Yukawa. Logic programming with equations. *The Journal of Logic Programming*, 4, pp 265 – 288, 1987.