# AN ARTIFICIAL INTELLIGENCE ENVIRONMENT FOR COMPUTER ALGEBRA

J. Calmet and I.A. Tjandra
Universität Karlsruhe, FRG

**Abstract**

We report on the design of an environment which allows to set Computer Algebra Systems (CAS) in the framework of Knowledge Representation Systems.

The first task was to design a general hybrid knowledge representation system capable of handling mathematical knowledge. This task has been completed and a system called MANTRA [Bit] is available for this purpose.

The second task is to define the concept of Mathematical Knowledge. This study is based on the definition of computing domains as Abstract Computational Structures. The inference procedure which permits the system to ensure that a mathematical operation on a given domain is valid is realized through learning methods.

## 1 Introduction

A mathematical domain of computation, consists of a set of well-defined objects and of a set of functions over the objects where the functions possess a certain set of properties. We regard the abstraction of a domain, i.e. without considering its particular implementation, as a unit of an abstract computational structure (ACS). Thus, a domain can be thought of as a model of an ACS under consideration [Gu 78].

CAS should be capable of managing and processing a very large variety of domains effectively, correctly and efficiently. Today CAS possess only a very small number of built in domains and, generally, there is no constructive way to ensure the correctness of computation on new user-defined domains.

We have designed a shell for knowledge representation which can be used as an environment for Computer Algebra capable of representing arbitrary mathematical domains linked by algebraic relationships, and of performing correct computations on these domains. A correct representation of domains is achieved through a constructive definition taking into account the functions of the domains and the properties. This includes consistency, completeness of properties and correctness of function implementations with respect to these properties. The main condition to be fulfilled in order to

perform a correct computation is tightly associated with a type inference mechanism, since mathematical expressions are, generally, not statically typed.

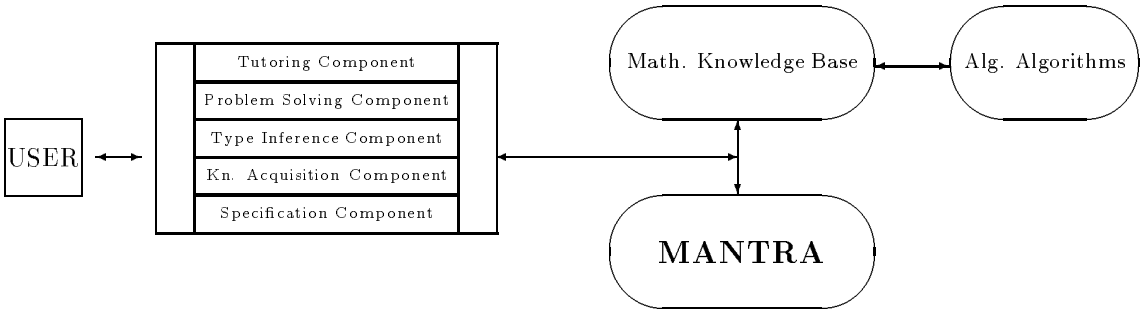


Figure 1: The structure of the environment

The environment (Figure 1) is built upon a hybrid knowledge representation system, called MANTRA. MANTRA was designed upon the following two principles: (i) several cooperating formalisms are better then a unique representation formalism, and (ii) a clear unified semantics explaining the meaning of the knowledge representation language is fundamental.

The specification component is used to specify and to represent arbitrary abstract computational structures and their domains. This component aids the user in building an ACS, also based on other known "lower" ACS, taking into account the consistency of the set of properties.

The learning component acts as an automatic knowledge acquisition tool in order to get a complete set of properties. This is performed by investigating the results of computations which are regarded as positive training instances. The method used is very reminiscent of the method of explanation-based generalization.

This paper proposes an overview of the project. In section 2 we describe MANTRA. In section 3 we present a formal definition of abstract computational structures and their representation. Section 4 deals with the method adopted to complete a set of properties. Finally, we present some concluding remarks on the design of the environment in section 5.

# 2 MANTRA

MANTRA (Modular Assertional, Semantic Network and Terminological Representation Approach) is a Shell for knowledge system, It is best described in [Bit] where a list of references is given. 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.

Logic, frames and semantic nets are used to express declarative knowledge and are called epistemological methods because they are associated to models in the external world. A production system represents a complete programming paradigm including the procedural aspect and is called a heuristic method because of this feature. One or more inference procedures are associated to each of the epistemological methods.

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 non-complete inference procedure and a query procedure which always terminates with an answer. They were introduced by Patel-Schneider [Pa]. These procedures are semantically defined through a four-valued logic approach [Be].

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 [Ho et al.] 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 weaken 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 knowledges from those acquired through the three methods separately. This interaction is defined not by the inference procedures 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.

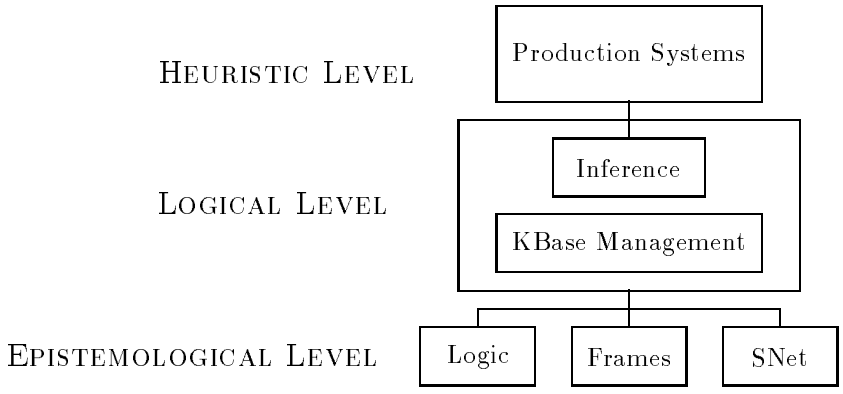


Figure 2: The architecture of MANTRA

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 for the first three 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. At this level the conflict resolution and flow control strategies applied during the execution of these production systems can be explicitly chosen through special primitives.

To facilitate the interconnection and communication between the different implemented methods a single data abstraction has been adopted: The directed graph. Directed graphs subsume several of the most commonly used data structures and they are also suitable for an interactive system because of their inherent graphical character.

The common basis for all the implemented inference procedures is the unification function. A specialized unification package has been implemented in KCL. It is based upon the almost linear algorithm of Martelli and Montanari [Ma,Mo].

# 3 Abstract Computational Structures

Because of the length imposed on the paper we omit the description of the basic concepts and notations which would help to better understand this section, they can be found in [Hu,Op] and [Ba,Wo].

Various kinds of algebraic structures such as semigroups, monoids, groups, rings or fields have to be considered. We regard these algebraic structures, which are second order types, as units of *abstract computational structures* since we do not take into account their implementation at this stage. The goal of introducing abstract computational structures is to group mathematical domains of computation in which the same operators with the same properties are defined.

We define an ACS as a pair: ACS = $\langle \Sigma, \mathcal{P} \rangle$ provided that $\Sigma$ and $\mathcal{P}$ possess the following meanings:

- $\Sigma$ is a *signature* that consists of
  - A *carrier* $\mathcal{S}$ that could include other *primitive carriers*
  - A set of *operator symbols* whose arguments belong to the carrier
- $\mathcal{P}$ is a set of rules (equations) determining the properties of the operations defined in the signature $\Sigma$. A property is denoted by a triple $p = \langle X, L, R \rangle$, where X is the set of variables occurring in the left-hand side ($L$) and right-hand side ($R$)

Based on other known ACS we can construct a new ACS by integrating all operators and properties possessed by the known ACS and by adding additional operators and properties into the new one. This implies that each ACS inherits all operators and properties possessed by the ACS upon which it is based.

In order to represent such ACS by means of the knowledge representation formalisms provided by Mantra we divide the construction of an ACS into the following parts: (i) ACS descriptor, (ii) ACS operators, (iii) ACS initial properties and (iv) ACS learned properties (see the example in figure 4).

Mantra provides a capability which allows us to build knowledge bases modularly. We can represent an ACS as a knowledge base that we call a *knowledge base module*. These modules, embodying the ACS and their models under consideration, are joined together into a semantic network preserving the relations among the ACS by means of the defined hierarchies. In the semantic network each node corresponds to a module and the links specify which entities are inherited by which modules within a particular hierarchy. Figure 3 shows an example.

An ACS descriptor is represented by a concept, using the **Tell** primitive provided by the knowledge base management at the logical level, possessing the following relations: (i) ACS-id, a unique identifier representing the name of the ACS, (ii) ACS-mode, a symbol representing $\mathcal{S}$, (iii) parameters, a list of other (known) ACS representing $\mathcal{S}$ and (iv) based-on, a list of other (known) ACS on which the ACS is based.

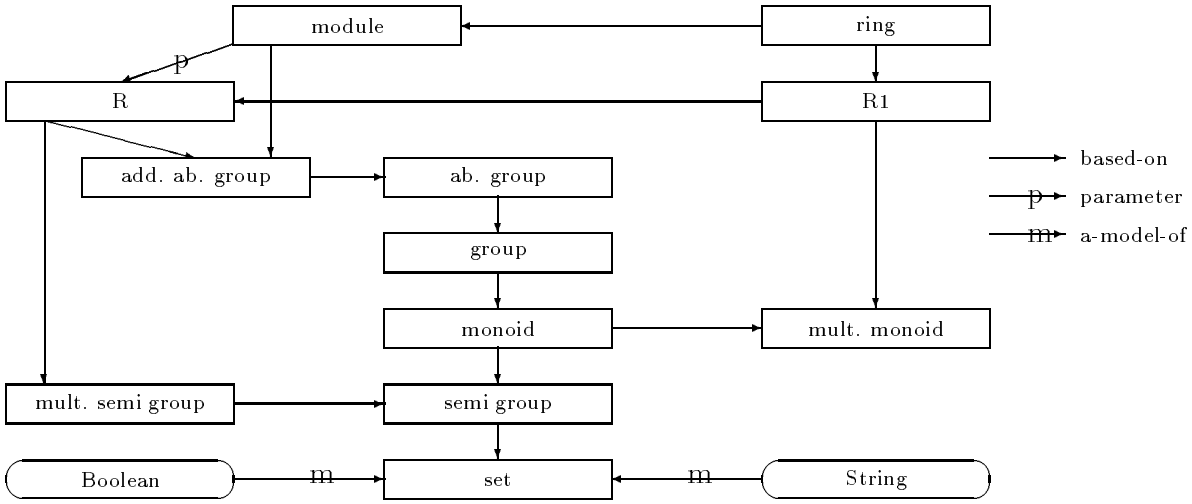


Figure 3: The semantic net of ACS and Domains

Similarly, each operator of an ACS is represented by means of a concept possessing the following relations: (i) operator-id, a symbol representing the name of an operator, (ii) domain, a list of other known ACS, which are elements of $\mathcal{S}$ , constituting the domain sorts of the operator, (iii) range, the range sort of the operator.

According to the definition of $T_\Sigma$, the set of terms in a $\Sigma$-domain, we can determine the language of terms by means of a context-free grammar[1]. We omit the technical details to construct such a context-free grammar. Using the grammar of an ACS we are able to build the parse tree of a term which plays a fundamental role in acquiring additional properties in order to complete the set of properties in an ACS.

Properties are specified into two parts: Initial properties and learned properties. The representation of both parts is the same. The initial properties are the basic properties to be possessed by the operators. The learned properties remain to be acquired in order to complete the set of properties. In the next section we deal with a method for completing a set of properties.

A property $p = \langle X, L, R\rangle$ is represented by means of a logic formula, using the assertional module at the logical level, where each variable in $X$ is bounded by a universal quantifier; $L$ and $R$ are treated as left-hand side and right-hand side respectively.

Another important role of properties is to aid the user in checking the consistency of the properties being specified. In order to check the consistence of $\mathcal{P}$, a set of properties, we check that $true = false$ is not deducible from $\mathcal{P}$.

In order to construct a domain with respect to a particular ACS we have to implement each relevant function through an operator symbol specified in the ACS. A domain consists of three parts: (i) domain descriptor, (ii) function descriptors and (ii) the implementation of each function. The implementation of functions is supported by an embedded Lisp programming environment and relies on the classical algebraic

[1]A context-free grammar is a Quadruple: $CFG = \langle N, T, \Pi, S\rangle$ where $N$ is a set of nonterminal symbols, $T$ is a set of terminal symbols, $\Pi$ is a set of context-free productions and $S$ is the start symbol

algorithms. The correctness of the function implementations with respect to the corresponding properties is verified by using a computational induction based on fixed point theory and structured induction. Figure 3 shows a semantic network representing ACS and their models. The nodes are represented by symbols identical to the symbols associated to the knowledge base modules.

# 4 Completing a set of properties

The next figure shows the ACS for the concept of group.

**ACS** group ≡ **mode** g ;

**based-on** {} ;

**function**

f : (g,g) ⟶ g
inv : (g) ⟶ g
ne : () ⟶ g ;

**initial properties**

$p_0$: ∀ x ∈ g : f(ne,x) = x
$p_1$: ∀ x ∈ g : f(inv(x),x) = ne
$p_2$: ∀ x,y,z ∈ g : f (f(x,y),z) = f(x,f(y,z)) ;

**learned properties**

$p_3$: inv(ne) = ne
$p_4$: ∀ x ∈ g : f(x,ne) = x
$p_5$: ∀ x ∈ g : inv(inv(x)) = x
$p_6$: ∀ x ∈ g : f(x,inv(x)) = ne
$p_7$: ∀ x,y ∈ g : f(inv(x),f(x,y)) = y
$p_8$: ∀ x,y ∈ g : f(x,f(inv(x),y)) = y
$p_9$: ∀ x,y ∈ g : inv(f(x,y)) = f(inv(y),inv(x));

**end-of-ACS**;

**figure 4: Example**

To complete the set of properties one can not rely on the Knuth-Bendix algorithm [Kn,Be] which is both inefficient and unpracticable for our purposes [Bi].

Let us suppose that we have an ACS for group as defined above and we assume that the learned properties have been acquired. Let $t_0$ = f(f(inv(f(y,f(inv(y),x))),x),f(ne,x)) and $t_{nf}$ = x, $t_0, t_{nf} \in T_\Sigma(X)$, be terms with respect to $\Sigma$. Since $t_0$ and $t_{nf}$ belong to the same congruence class, the interpretations of both terms yield the same result, i.e. $t_{nf} =_R t_0$. Therefore, it is conceivable to reduce a term to its canonical normal form before interpreting it. This reduction process is accomplished syntactically. In order to achieve $t_{nf}$ from $t_0$ the following reduction chain is processed.

$$t_0 \longrightarrow^{p_8} t_1 \longrightarrow^{p_1} t_2 \longrightarrow^{p_0} t_3 \longrightarrow^{p_0} t_{nf}$$

The underlying idea for the reduction of a term is that the problem of the validity of an equation in an ACS can be solved by the study of a congruence equation modulo a relation in the domain ($=_R$), a syntactic problem. In order to determine whether or not an equation is valid in the given ACS, i.e., whether or not two terms belong to the same congruence class we have to construct a **"complete"** set of properties.

Therefore, *Completion* means according to [Ba et al.] acquiring sufficiently many properties such that, in particular, any application of a property in a proof within the equational theory can be transformed into a reduction proof. Hence, a property can be eliminated during completion if there is already a reduction proof for it, or, more generally, if there is a "simpler" proof of the same property which we know will become a reduction proof eventually.

We just keep the properties defined by the user as they are, i.e., at the beginning we do not try to make the set of properties complete. Instead, the interpretations of terms should be used as positive training instances from which new properties could be acquired incrementally. The new acquired properties constitute the *learned properties part* as mentioned above. This can be achieved by introducing the following basic steps: (i) Before t $\in T_\Sigma$ is interpreted, it should be reduced using the existing properties. In this step, we should use heuristics to support or to accelerate the reduction process. (ii) Interpret $t$. (iii) Try to find a relation between the result and the term which has been interpreted by using the method of goal regression [Wa]. This involves traversing their parse trees in the top-down manner and replacing constants by variables consistently. (iv) The result of the generalization process, if it can be found, is an equation of the form *"generalized term = generalized result"* . According to the notion of completion, as mentioned above, the generalized equation remains to be proved in order to acquire new properties. (v) Integrate the new acquired properties into the learned properties part.

Restricting the properties to the following forms: (i) a left hand side of a property can be reduced by no property but itself and (ii) a right hand side of a property can not be reduced by a property we are capable of determining a method which can be applied to state and to solve the problems in steps 3 and 4.

The problem occurring in step 3 can be stated as follows: **given:** A term $L$ and a term $R$, the result of the interpretation of $L$ ($L, R \in T_\Sigma$), **determine:** $L' = R'$ $L', R' \in T_\Sigma(X)$, the generalization of the equation $L = R$.

An algorithmic method based upon a depth-first tree traversal has been designed to determine the generalization of $L = R$ [Ca,Tj].

The problem occurring in step 4 can be stated as follows : **given:** An equation $L' = R'$ as hypothesis, **determine:** The proof of the given equation.

According to the notion of completion, as mentioned above, an equation can be eliminated if there is already a rewrite proof for it. Our problem in this step leads to acquiring properties, which can be used to prove the given equation. [Ca,Tj]

# 5 Conclusion

We have investigated the possibility to use AI-methods to represent mathematical domains. Many problems have been omitted here. For instance, can we use the environment to free a user to explicitly enters the types of expressions. A solution is to use a type inference mechanism. Generally, this problem is undecidable, as the word problem of a Chomsky-0 language can be reduced to this problem. The proposed solution is mainly supported by the SNet module to embody subtype relations among the domains and by the production system to represent type inference rules, such as coercion rules, resolve rules and force rules. The inferencing of types is done by forward chaining, as MANTRA currently only support forward chaining. We omit the technical details on how to realize this component.

The idea behind completing a set of properties is that a semantic problem can be solved by the study of an equation congruence modulo a relation, a syntactic problem. The currently existing completion methods have practical limitations due to the nonexistence of a complete set of properties in many cases. We propose an approach to acquire the properties that can be used to prove a given equation. The major steps in acquiring such properties are the generalization and the proof of positive training instances. The proposed method is very reminiscent of the method of explanation-based learning [Mit et al.].

One of the further works in progress is to design the problem solving component. The main idea is that a, say mathematical or physical, problem can be solved by specifying the problem in the representation as defined in the environment and by using knowledge which has been stored in knowledge bases. Basically, it is very difficult to find a correct solution since, probably, the knowledge, which has been already stored, are not sufficient to be used to solve the given problem. A conceivable approach consists in integrating an intelligent knowledge acquisition tool to the component. The component could then interact with an expert in order to acquire additional knowledge required to solve the problem.

To implement effectively the proposed approach we must still solve several problems, such as to find an approximated criterion to terminate the property acquisition procedure when, for instance, "sufficiently" many properties have been acquired (generally this problem is undecidable). To have an efficient implementation with satisfactory running time must also be achieved. To find solution to such problems is the next step in this ongoing research project.

# References

[Ba et al.] Bachmaier, L., Dershowitz, N., Hsiang, J. : *Proof Orderings for Equational Proof* ; Proc. LISC 86, 346 —357.

[Ba,Wo] Bauer F.L., Woessner H.; *Algorithmische Sprache und Programmentwicklung (2nd Edition)* ; Springer-Verlag Berlin Heidelberg New York Tokyo, 1984.

[Bi] Bibel, W. : *Automated Theorem Proving* ; Vieweg; 1982.

[Be] Belnap N.D.: *A Useful Four-Valued Logic*; in "Modern Uses of Multiple-Valued Logic", ed. G. Epstein and J.M. Dunn, Boston: Reidel, 1977, pp. 8 – 37.

[Bit] Bittencourt G.: *An Architecture for Hybrid Knowledge Representation*; Ph D Thesis; Universität Karlsruhe; 1990.

[Bra et al.] Brachman R.J., Fikes R.E., Levesque H.J.: *KRYPTON: A Functional Approach to Knowledge Representation*; IEEE Computer, 16 (10), pp. 67 – 73, 1983.

[Ca,Tj] Calmet J., Tjandra I.A.: *Learning Complete Computational Structures*, In M.L. Emrich et al., editor, Fifth International Symposium on Methodologies for Intelligent Systems (Selected Papers), pp 63 –72, Knoxville, TN, 1990.

[Gu] Guttag J.V.: *The Algebraic Specification of Abstract Data Types*; Acta Informatica 10, 27–52, Springer-Verlag, 1978.

[Ho et al.] Horty J.F., Thomason R.H., Touretzky D.S.: *A Skeptical Theory of Inheritance in Nonmonotonic Semantic Nets*; Report CMU-CS-87-175, Carnegie-Mellon Univ., 1987.

[Hu,Op] Huet G., Oppen D.: *Equations and Rewrite Rules: a survey*; in Book R., editor, Formal Language Theory: Perspective and Open Problems, Academic Press, 1980.

[Kn,Be] Knuth D.E., Bendix P.B. : *Simple Word Problems in Universal Algebras*; OXFORD, 263 —298 ; 1967.

[Ma,Mo] Martelli A., Montanari U.: *An Efficient Unification Algorithm*; ACM-TOPLAS, 4(2), pp. 258 – 282, 1982.

[Mit et al.] Mitchell T.M., Keller R.M., Kedar-Ceballi S.T.: *Explanation-Based Generalization : A Unifying View*; Machine Learning 1 47 — 80 ; Kluwer Academic Publishers ; 1986.

[Pa] Patel-Schneider P.f.: *A Decidable First-Order Logic for Knowledge Representation*; Proceedings of IJCAI 9, pp. 455–458, 1985.

[Wa] Waldinger, R. : *Achieving Several Goals Simultaneously*; in Machine Intelligence 6 (eds. E. Elcock and D. Michie), 94 — 136, Ellis Horwood ; 1977.