Applications, ISL Report R 109/92, pp. 19.2–19.11, 1992.

MATHEMATICAL KNOWLEDGE REPRESENTATION

J. Calmet, K. Homann and I.A. Tjandra
University of Karlsruhe
Am Fasanengarten 5 — W-7500 Karlsruhe 1
Germany

Abstract

We report on the design of an environment that aims 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* is available for this purpose.

The second task is to define the concept of Mathematical Knowledge. This study is based upon the concept of Abstract Computational Structures. The inference procedure which enables the system to ensure that a mathematical operation on a given domain is valid is also taken into account.

1 Introduction

A computer algebra system performs computations on a given mathematical domain through operators that have well defined sets of properties. It is possible to formalize a concept of Mathematical Knowledge based upon the three entities: domain, operator, properties, as an abstract computational structure according to the model introduced by Bauer [Ba-Wo].

The properties of the operators are either known or acquired during a computation using methods coming from machine learning.

To represent and implement such a concept, we have designed a hybrid knowledge representation system, *MANTRA* [Bit89b, Bit90, Ca et al. 91c], with four innovative features: a unified semantics based upon a four-valued logic, communication among the implemented knowledge formalisms (logic, frames, semantic nets), decidability of the inference algorithms, a three level's architecture (one of these levels is a production rules system that enables to conceive expert systems). We do not use one of the well known systems which incorporate several methods for knowledge representation, such as the expert system shell KEE [KEE] for instance, because they lack both a representational theory

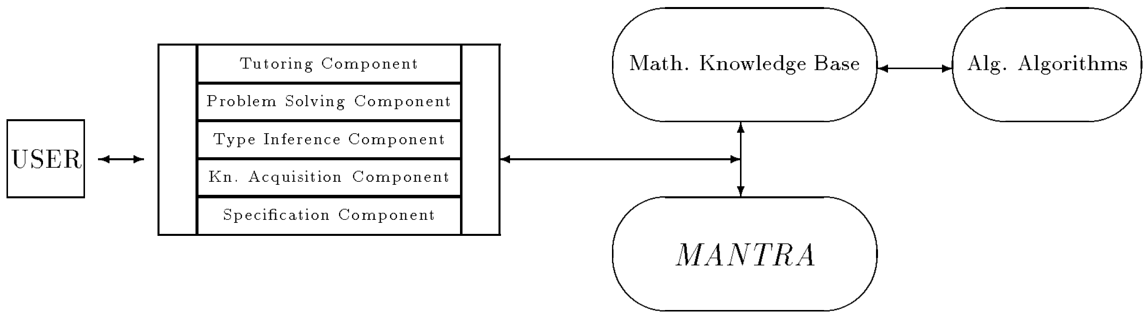

Figure 1: The structure of the environment

which explains which knowledge is to be represented by which formalisms and a common semantics. *MANTRA* possesses assertional, terminological, semantic network representation capabilities as well as production systems. It is a general purpose shell for knowledge representation systems and its use is not restricted to symbolic computation.

The environment for our project (Figure 1) is built upon *MANTRA* [Ca-Tj91a]. 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 Abstract Computational Structure (ACS), also based on other known "lower" ACS, taking into account the consistency of the set of properties of the operators.

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 explanation-based generalization [Ca-Tj90b, Ca-Tj91b].

The paper is organized as follows: In section 2 we describe *MANTRA*. In section 3 we give an outline of the formal definition of abstract computational structures and their representation. Section 4 deals with the method adopted to complete a set of properties. In section 5 we present some concluding comments and in particular we compare our approach with works that may look similar at first sight. This paper is merely an overview. But, more technical and complete descriptions are presented in the referenced papers.

2 MANTRA

The *MANTRA* system [Ca et al. 91c] is a hybrid system for knowledge representation with the following characteristics: All modules are semantically consistent and all the algorithms involved are decidable. The decidability requirement has been met with the adoption of a four-valued semantics based on the works of Patel-Schneider [PS], Frisch [Fr] and Thomason et al. [Th et al.]. This semantics is used throughout the system and ensures that it is semantically consistent.

The language can be thought of as an abstract data type (this is explained in [Ca et al. 91c]) allowing the creation and manipulation of knowledge bases. The *knowledge bases* consist of a set of knowledge base partitions, each associated to an independent formalism. The division of the language into several formalisms has two advantages: The computability problems associated to each formalism can be solved independently and the integration of new formalisms to the system is facilitated.

Each formalism is characterized by a set of definitions and a set of questions. The *definitions* allow the storage of knowledge into the knowledge bases and the *questions* allow the interrogation of these knowledge bases. Definitions are used to store knowledge only into the partition associated to the formalism, but questions can be directed to this partition or to a combination of two or more partitions of the knowledge base. The language is based on a new architecture, figure 2, consisting of three levels: The epistemological level, the logical level and the heuristic level.

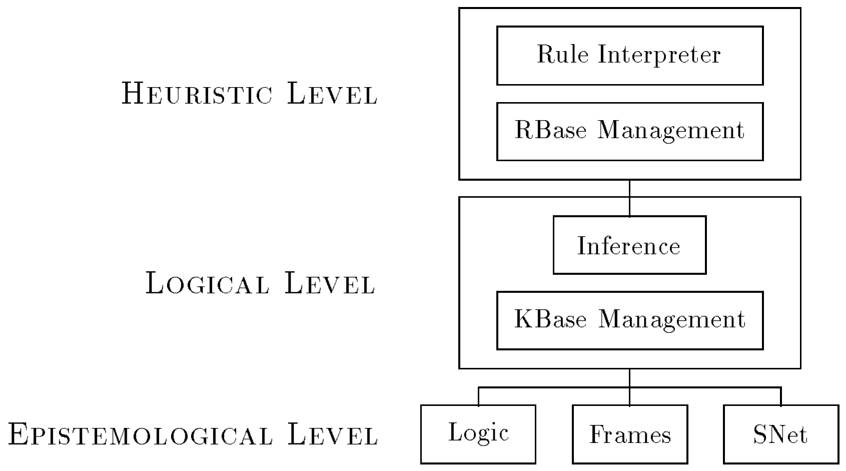

Figure 2: The architecture of *MANTRA*

The first level consists of three modules: an assertional module (the Logic module of figure 2), based on a decidable first-order logic language [PS], a frame module, based on the terminological box of Krypton [Br et al.], and a semantic network module, providing inheritance with exceptions [Et]. These modules offer several original features with respect to previously existing systems as described in [Bit90]. The primitives of the modules of this first level define the epistemological primitives of the language. These primitives are not complete expressions of the language but are used as parameters for the Ask and Tell primitives of the logical level.

The primitives of the *assertional module* correspond to the usual operators of the first-order logic languages, but the meaning of these operators is based on a four-valued semantics.

The primitives of the *frame module* allow for the definition of a terminology consisting of the purely intentional description of categories of objects. These categories are described by restricting the values of the properties of the objects forming them.

The primitives of the *semantic net module* manipulate the notions of *objects*, *classes* of objects and *hierarchies*. The notions of classes and hierarchies can be considered as explicit versions of the concepts and relations described in the frame module.

The second level introduces the notion of knowledge base. The language can be thought of as an abstract data type [Bit90] whose access functions are the primitives of this level. These primitives use the primitives of the first level as parameters. Two types of primitives are provided, **Tell** and **Ask**, these primitives are used, respectively, to store facts and to interrogate knowledge bases. The Ask primitives are defined in such a way that new facts can be inferred from evidence provided by the knowledge acquired only by one or by a combination of two of the first level modules.

Finally, the third level consists of primitives allowing the definition of *rule bases* [Lu-Ho]. These rules are formed by a conditional part consisting of queries to the knowledge bases defined in the logical level and an action part consisting of definitions. They can be manipulated and are used by the rule interpreter to automatically manipulate knowledge bases. The behaviour of the interpreter can be determined by choosing conflict resolution or control strategies in the recognize-act cycle. Every rule base consists of rules which can either represent epistemological knowledge, such as introducing domain knowledge in the knowledge bases, or they can specify strategies for the utilization of the logical level primitives.

This brief overview of *MANTRA* covers only those features which are used in the sequel.

3 Mathematical Domains of Computation

A mathematical domain of computation consists of a set of well-defined objects and of a set of operators over the objects which possess each 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 [Ba-Wo, Gu].

Various kinds of algebraic structures such as semigroups, monoids, groups, rings or fields to quote only a few, have to be considered. We regard these algebraic structures 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 Σ and $\mathcal{P}$ possess the following meanings:

- Σ 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 Σ. 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)

At this stage on must make two remarks. The first one is that the above definition could be interpreted as representing an equational theory in an associated term algebra. But, this is not the framework we are using. The second remark is that the complete formal definition of ACS's is too long to be given here. It can be found in [Ca-Tj91b].

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 3).

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.

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 [Ca et al. 90a].

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_Σ, the set of terms in a Σ-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 sense that

[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, Π is a set of context-free productions and S is the start symbol

1. It possesses the finite termination property, i.e. the iteration of the reduction process always terminates at an irreducible term after finitely many steps.

2. It possesses the unique property, i.e. different reduction beginning at the same term always terminates at the same irreducible term.

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.

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 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.

4 Completing a set of properties

Figure 3 shows the ACS for the concept of group, assuming that it does not inherit other domains. To complete the set of properties, which are expressed as equations, one can not rely on the Knuth-Bendix algorithm [Kn-Be] which is both inefficient and not practical for our purposes. Our approach can be sketched as follows.

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 Σ. 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

ACS group ≡ **mode** g ;

  based-on {} ;

  function

    f : (g,g) $\longrightarrow$ g

    inv : (g) $\longrightarrow$ g

    ne : () $\longrightarrow$ 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 3: Example

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 ∈ T_Σ 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.

If we restrict 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 then 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-Tj90b].

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.

5 Conclusion

We have depicted the possibility of using AI-methods to represent mathematical domains. Many problems have been omitted here. For instance, we can use the environment to free a user from explicitly entering the types of expressions. A solution is to use a type inference mechanism [Tj]. 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. Type inference is done by forward chaining, as *MANTRA* currently only supports 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 a set of equations modulo a relation, a syntactic problem. The currently existing completion methods have practical limitations due to the non-existence 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 [Ca-Tj90b]. The proposed method is very reminiscent of the method of explanation-based learning [Mi et al.].

To define mathematical knowledge within our environment enables to study in a consistent way different problems. Some are directly linked to Computer Algebra such as the design of the problem solving component or the design of a compiler suited to our approach [Tj]. This latter study relies upon the type system under development. A longer term goal is to design a computer algebra system tailored for this environment. Such a task has been undertaken but with a low priority. A trivial possibility is to embed an existing sytem into the first level of *MANTRA*. This has been achieved.

The learning capablities of our environment also permit to incorporate capabilities which are usually not connected with Computer Algebra such as the design of a theorem prover based upon learning methods. This task is just starting but with a high priority since this seems to be a new concept for the design of theorem provers.

Such an environment looks to be well suited to many applications in many fields from designing a system solving differential equations to designing a system for management decisions because of the integration of frames and semantic nets in a consistent environment.

We are not aware of works with similar diverse goals. *Nuprl* or *Ontic* are only considering mathematical problems on a restricted set of domains. Only the *TASSO* [Mi] project looks to have some similarities but, according to our understanding, in a restricted scope. The part covered in this paper has some analogy with *Axiom*, although we start from a different point of view, in the sense that categories in *Axiom* are partly what we call ACS's but they do not take into consideration the properties of operators.

References

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

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

[Bit89b] G. Bittencourt, *A Hybrid System Architecture and its Unified Semantics*, Proceedings of The Fourth International Symposium on Methodologies for Intelligent Systems, October 11 – 14, Charlotte, NC, USA, 1989.

[Bit90] G. Bittencourt, *An Architecture for Hybrid Knowledge Representation*, Ph D Thesis, Universität Karlsruhe, 1990.

[Br et al.] R.J. Brachman, V.P. Gilbert, H.J. Levesque, *An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of KRYPTON*, Proceedings of IJCAI 9, pp. 532 – 539, 1985.

[Ca et al. 90a] J. Calmet, G. Bittencourt, I.A. Tjandra, *A Framework for Representing Algebraic Knowledge Using a Hybrid Knowledge Representation System*, Proceedings of the fourth International Symposium on Knowledge Engineering, May 7 – 11, 1990, Barcelona – Spain.

[Ca-Tj90b] J. Calmet, I.A. Tjandra, *Learning Complete Computational Structures*, in Emrich et al (Eds), 5th International Symposium on Methodologies for Intelligent Systems (Selected Papers), Knoxville – USA, October 24 – 27, 1990, pp. 63 – 71, ICAIT.

[Ca-Tj91a] J. Calmet, I.A. Tjandra, *An AI Environment for Computer Algebra*, Proceedings of the International Conference on Artificial Intelligence in Mathematics, Glasgow – UK, April 3 – 5, 1991, pp 83 – 92.

[Ca-Tj91b] J. Calmet, I.A. Tjandra, *Representation of Mathematical Knowledge*, Z. W. Ras et al.. (Eds.) Proceedings of the 6th International Symposium on Methodologies for Intelligent Systems, Charlotte USA, October 16 – 19, 1991, Springer-Verlag.

[Ca et al. 91c] J. Calmet, G. Bittencourt, I.A. Tjandra, *MANTRA: A Shell for Hybrid Knowledge Representation*, Proceedings of the third International conference on Tools for Artificial Intelligence, San Jose USA, November 5 – 8, 1991, IEEE Computer Society Press.

[Ca-Tj91d] J. Calmet, I.A. Tjandra, *An Expert System for Correctness of Symbolic Computation*, Proceedings of the World Congress on Expert Systems, Orlando USA, December 16 – 18, 1991, Pergamon Press.

[Et] D.W. Etherington, *On Inheritance Hierarchies with Exceptions*, Proceedings of AAAI-83, pp. 104 – 108, 1986.

[Fr] A.M. Frisch, *Knowledge Retrieval as Specialized Inference*, Report No.214, Department of Computer Science, University of Rochester, May 1987.

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

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

[KEE] *KEE Software Development System User's Manual*, Intellicorp, Mountain View California, 1985.

[Lu-Ho] A. Lulay, K. Homann, *Entwurf und Implementierung der heuristischen Ebene eines hybriden Wissensrepräsentationssystems*, Diploma Thesis, Universität Karlsruhe, 1991.

[Mi] A. Miola, *Design and Implementation of Symbolic Computation Systems*, Proceedings of IFIP Working Conference on Programming Environments for High-Level Scientific Problem Solving, Karlsruhe, 1991.

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

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

[Th et al.] R.H. Thomason, J.F. Horty and D.S. Touretzky, *A Calculus for Inheritance in Monotonic Semantic Nets*, Technical Report CMU-CS-86-138, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1986.

[Tj] I.A.Tjandra, Forthcoming Dissertation Thesis.

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