ematical Assistants – Issues and possible Solutions", 11th European Conference on Artificial Intelligence, Amsterdam, pp. 48–53, 1994.

Towards an Intelligent Mathematical Environment

Bridging the Gap between Theorem Proving and Symbolic Mathematical Computing

KARSTEN HOMANN JACQUES CALMET

Universität Karlsruhe
Institut für Algorithmen und Kognitive Systeme
Postfach 69 80 · D-76128 Karlsruhe · Germany
{homann,calmet}@ira.uka.de

Abstract

One of the purposes of an intelligent assistant for Mathematics is to support formal reasoning and efficient computation in mathematics and mathematical applications. Such an assistant requires to represent complex mathematical knowledge and to perform symbolic mathematical computations in an environment which also provides sophisticated AI techniques.

We describe and outline the advantages of an intelligent mathematical environment which combines automated theorem proving and symbolic mathematical computing.

1 Introduction

The dream of an intelligent assistant for mathematicians has motivated researchers for decades. One of the purposes of a comfortable interactive environment is to support formal reasoning and efficient computation in mathematics and mathematical applications. Such an assistant requires to represent complex mathematical knowledge and to perform symbolic mathematical computations in an environment which also provides sophisticated AI techniques, e.g. automated theorem proving, machine learning, planning.

Computer algebra systems (CAS) usually offer a powerful collection of algebraic algorithms and a straightforward programming language. In classical systems the mathematical knowledge, e.g. definitions of mathematical structures, properties of operators on a domain, domain and range of algorithms and their mathematical specification, is hidden in the algebraic algorithms. AXIOM [JeSu92] allows the definition of new abstract data types including properties of operators and started a new generation of systems, but no AI methods (e.g. automated theorem proving, learning) are available. CAS are very efficient to compute symbolic solutions by given algorithms but cannot derive new theorems or lemmas.

On the other hand, automated theorem provers (ATP) have shown remarkable results in proving non-trivial mathematical theorems. However, they lack some mathematical knowledge, algebraic algorithms, intelligible representations and proofs, are hard to use, or compute huge search spaces.

A promising approach consists in the integration of theorem proving and symbolic mathematical computing into a common environment. We report on such an environment called $\lambda\epsilon\mu\mu\alpha$[1] which enables to rely on algebraic algorithms, to derive theorems, to deal with both

[1] $\mathcal{L}$earning $\mathcal{E}$nvironment for $\mathcal{M}$athematic and $\mathcal{M}$athematical $\mathcal{A}$pplications

vertical and inclusion polymorphisms, to learn and to apply equation schemata. The explicit formalization of mathematical dependencies provides new possibilities to explain the solution steps.

2 An Intelligent Environment for Symbolic Mathematical Computing

An environment for solving mathematical problems which integrates theorem proving, symbolic computing, explanation-based learning and a knowledge representation system is given in figure 1. The schema-based representation of mathematical structures and algorithms enables the representation of meta-knowledge, e.g. constraints of parameters, dependencies of algorithms and theorems.

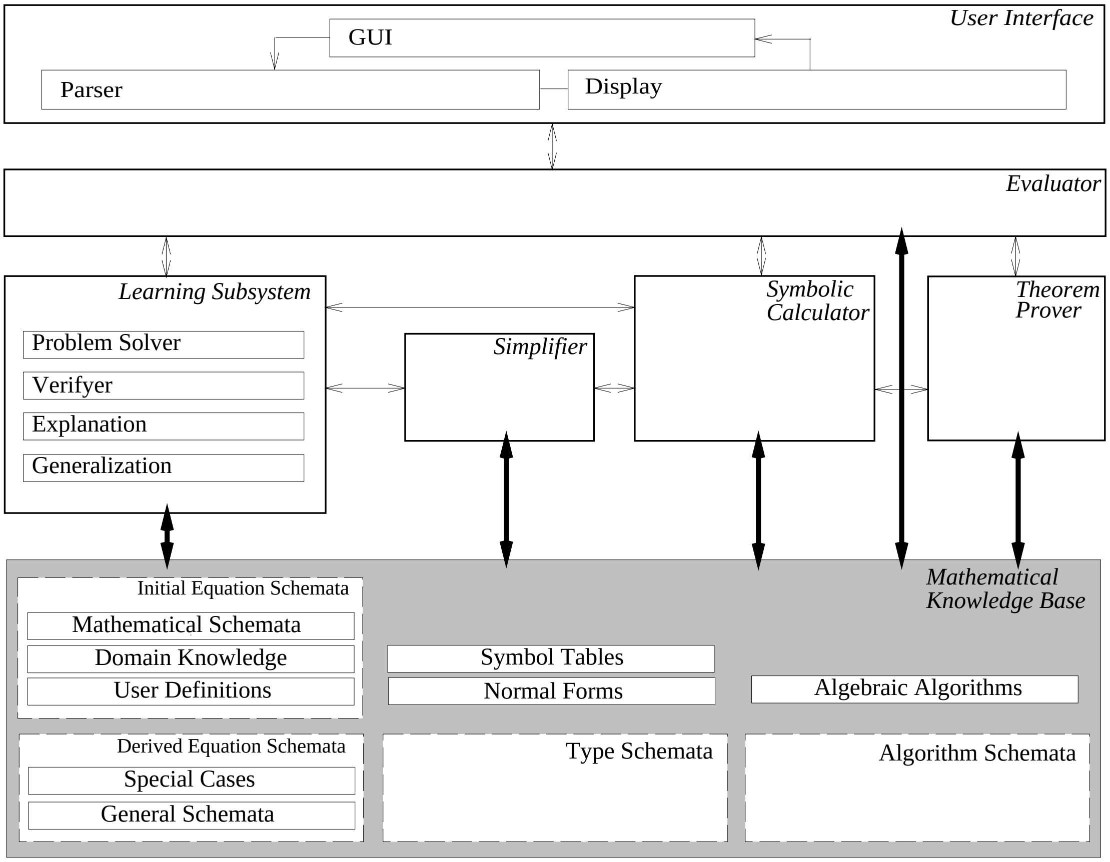

Figure 1: Architecture of the intelligent environment for symbolic computing

The user interface offers frames and graphs for handling schemata and displays explanations about solutions of specific problems. An evaluator solves these problems through a theorem prover, algebraic algorithms (symbolic calculator), and applying equation schemata (learning subsystem). The knowledge base consists of symbol tables, normal forms of the simplifier, algebraic algorithms of the symbolic calculator, algorithm schemata for the specification of algorithms, type schemata for abstract computational structures, as well as initial and derived equation schemata for simplifying expressions.

Equation schemata consist of mathematical rewriting rules which model domain knowledge, and user defined laws. New equation schemata can be learned by generalizing specialized solutions using explanation-based learning (EBL). Given problems are solved by applying schemata to eliminate obstacles [Shav90] in the calculation of unknown properties of a variable. An explanation why this is an appropriate solution to the problem is generated, the achieved schema is generalized to solve other problems, and finally, the knowledge base of equation schemata is updated by the new generalized schema.

The user doesn't receive any information about the solution steps from the system, e.g. why is the output the solution of the given problem, or how to find the solution of a problem. Therefore, algorithms are represented in terms of schemata, too. An example schema of the general gcd algorithm is given in figure 2. It allows the representation of meta-knowledge like:

- *Name*, a unique identifier of the schema with variable bindings
- *Signature*, describes the types of input and output
- *Constraints*, imposed on the domains and ranges
- *Definition*, mathematical description of the output
- *Subalgs*, list of subalgorithms describing the embedded subtasks
- *Theorems*, describing properties of the algorithm
- *Function*, name of the corresponding executable algebraic function to compute the output.

Name	$\mathsf{gcd}(?a, ?b) = ?g$
Signature	$?A \times ?A \rightarrow ?A$
Constraints	isa $(?A$, EuclideanRing)
Definition	$(?g \mid ?a) \wedge (?g \mid ?b) \wedge (\forall c \in ?A : (c \mid ?a) \wedge (c \mid ?b) \Rightarrow (c \mid ?g))$
Subalgs	
Theorems	$\mathrm{gcd}(u, v) = \mathrm{gcd}(v, u)$ $\mathrm{gcd}(u, v) = \mathrm{gcd}(v, u \; mod \; v)$ $\mathrm{gcd}(u, 0) = u$
Function	

Figure 2: Schema of algorithm gcd

A proposed interaction between a symbolic calculator (SC) and an automated theorem prover is illustrated in figure 3. Algorithms can be used for the efficient computation of predicates when proving theorems. A simple interaction needs to transfer all necessary knowledge and parameters to the SC. This is avoided when a common knowledge base is used, and a direct link between SC and ATP allows the immediate call of an algorithm within a proof, e.g. OTTER 3.0 [McCu94] allows the introduction of user-defined algorithms which must be identified by a special character (e.g. $GCD). CAS provide an extensive collection of very efficient mathematical algorithms, thus reimplementation is neither necessary nor meaningful.

Example:
OTTER allows the definition of simple functions, e.g. factorial. The performance can be increased strongly by calling the efficient factorial algorithm available in the SC.

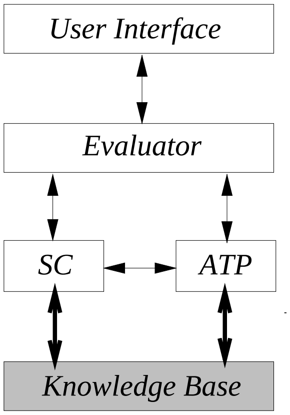

Figure 3: Interaction between algorithms and theorems

```
factorial(x) =        % factorial for nonnegative integers
   $IF($EQ(x,0),
       1,
       $PROD(x, factorial($DIFF(x,1)))).
```

The application of theorems is useful even when running algebraic algorithms (e.g. verification of conditions, properties of objects). The advantage lies in using the powerful reasoning capabilities of the theorem prover in the SC.

Example:
A condition in an algebraic algorithm, e.g. "if #IsNormal(G,H) then..." [2], can be verified by theorems of an ATP, e.g. try to prove that all subgroups of index 2 are normal.

A complete integration of algorithms and theorems is achieved by combining both interactions. At any step, arbitrary combinations of algorithms and theorems can be applied to solve a given problem. Such a feature combines the respective advantages, but requires to fit the SC and the ATP to a common knowledge representation.

Example:
Berlekamp's factorization algorithm contains the condition "if #SquareFree(p) then..." can be verified using a theorem $\forall f \in \mathbb{Z}_p[x] : SquareFree(f) \Leftrightarrow \$GCD(f, f') = 1$. The SC can be used to compute the derivation of p and the gcd.

3 A Knowledge-Based Type System

The theory of algebraic specification provides a good framework to design the type system of a mathematical assistant. The specification language FORMAL-Σ [CaTj93] has been designed to represent the mathematical knowledge. It is well-suited to specify mathematical domains of computations, e.g. finite groups, polynomial rings, which are inherently modular. An algebraic specification introduces constants, operators and properties in their intended interpretation and enables the re-use of subspecifications within a specification in accordance with the dependencies between particular specification modules of an abstract computational structure (ACS).

A type schema represents such a module and consists of:

[2] The special character # indicates a call to the theorem prover.

- *Name*, a unique identifier
- *Based-on*, a list of inhereted ACS
- *Parameter*, a list of ACS which are parameters
- *Sorts*, declaration of new sorts
- *Operators*, declarations of new operators
- *InitialProps*, initial properties.

Figure 4 show the schemata of some example ACS (more details may be found in [CHT92]). These definitions build a based-on hierarchy of the mathematical domains of computation (figure 5).

Name	Monoid
Based-On	SemiGroup
Sorts	Mo $ne \in$ Elt
Operators	
InitialProps	$\forall x \in$ Elt: $ne\ f\ x = x$

Name	Group
Based-On	Monoid
Sorts	Gr
Operators	inv _ :: Elt $\rightarrow$ Elt
InitialProps	$\forall x \in$ Elt: $inv(x)\ f\ x = ne$

Name	Ring
Based-On	MultSemiGroup (*rename:* $(f, \times)$, $(ne, 1)$) AddAbelianGroup (*rename:* $(f, +)$, $(ne, 0)$, $(inv, \Leftrightarrow)$
Sorts	Ri
Operators	
InitialProps	$\forall x, y, z \in$ Elt: $x \times (y + z) = (x \times y) + (x \times z)$ $\forall x, y, z \in$ Elt: $(y + z) \times x = (y \times x) + (z \times x)$

Figure 4: Type schemata for Monoid, Group, and Ring.

4 Conclusion

We have very briefly outlined the main features of an intelligent mathematical environment which combines automated theorem proving and symbolic mathematical computing. The architecture of the environment integrates a subsystem for learning equation schemata by EBL and an automated theorem prover for the derivation of theorems based on the definitions and theorems of algorithm schemata and properties of operators of type schemata. Explanations on the solutions to a given problem are given by graphs of dependencies between algorithms, mathematical definitions and types.

The foreseen advantages and benefits of this approach consist in being able to address the following problems:

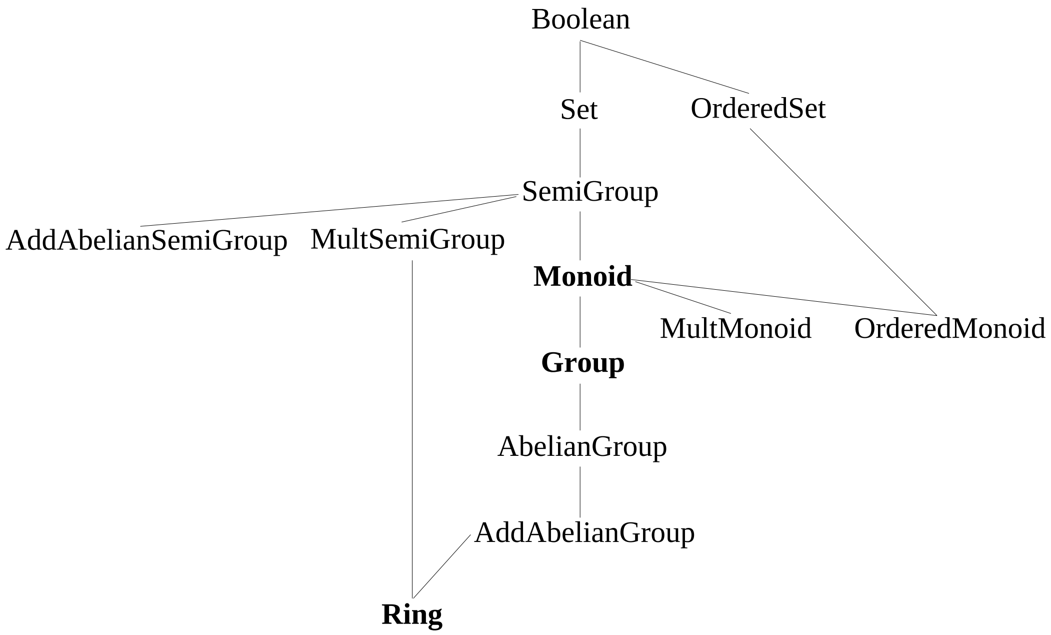

Figure 5: Hierarchy of type schemata

- problem solving by combining automated theorem proving and symbolic calculations
- modifying EBL to incrementally complete the properties of operators in ACS,
- extraction of mathematical schemata from algebraic algorithms,
- explanation of the solutions.

The next tasks within this on-going project are to design a common language for SC and ATP, the generation of algorithms from theorems and their verification and the integration of the learning component.

References

[CHT92] J. Calmet, K. Homann, I.A. Tjandra, *Unified Domains and Abstract Computational Structures*, in J. Calmet (ed.), International Conference on Artificial Intelligence and Symbolic Mathematical Computing, Karlsruhe, August 3–6, 1992, LNCS 737, pp 166–177, Springer, 1993.

[CaTj93] J. Calmet, I.A. Tjandra, *A Unified-Algebra-Based Specification Language for Symbolic Computing*, in A. Miola (ed.), Design and Implementation of Symbolic Computation Systems, LNCS 722, pp 122–133, Springer, 1993.

[JeSu92] R.D. Jenks, R.S. Sutor, *AXIOM*, Springer, 1992.

[McCu94] W.W. McCune, *OTTER 3.0 Reference Manual and Guide*, Technical Report ANL-94/6, Argonne National Laboratory, 1994.

[Shav90] J.W. Shavlik, *Extending Explanation-Based Learning by Generalizing the Structure of Explanations*, Pitman, London, 1990.