In J. Calmet, J.A. Campbell (Eds.), Integrating Symbolic Mathematical Computation and Artificial Intelligence, Proceedings of 2nd International Conference on Artificial Intelligence and Symbolic Mathematical Computing (AISMC-2), Karlsruhe, Springer, LNCS 958, pp. 18–29, 1995.

Combining Theorem Proving and Symbolic Mathematical Computing

Karsten Homann and Jacques Calmet

Universität Karlsruhe
Institut für Algorithmen und Kognitive Systeme
Am Fasanengarten 5 · 76131 Karlsruhe · Germany
{homann,calmet}@ira.uka.de

Abstract. An intelligent mathematical environment must enable symbolic mathematical computation and sophisticated reasoning techniques on the underlying mathematical laws. This paper disscusses different possible levels of interaction between a symbolic calculator based on algebraic algorithms and a theorem prover. A high level of interaction requires a common knowledge representation of the mathematical knowledge of the two systems. We describe a model for such a knowledge base mainly consisting of type and algorithm schemata, algebraic algorithms and theorems.

Keywords: theorem proving, symbolic mathematics, knowledge representation

1 Introduction

The dream of "doing" mathematics on a computer is progressively becoming true. Ideally, an intelligent assistant for doing mathematics will be a user friendly interactive environment allowing to perform computations, to prove theorems and to support formal reasoning and advanced tutoring. Such an environment must thus rely on some sophisticated AI techniques, e.g. automated theorem proving, machine learning and planning.

At present, two clases of mathematical computations can be efficiently performed. On one side, computer algebra systems (CAS) usually offer a large collection of powerful algebraic algorithms and a straightforward programming language. In classical systems the mathematical knowledge, e.g. definitions of mathematical structures, properties of operators of a domain, domains of computation, range of algorithms and their mathematical specification, is hidden in the algebraic algorithms. Axiom [JeSu92] allows the definition of abstract data types including operators and domains of computation. However, no AI methods (e.g. automated theorem proving, learning) are provided. CAS are very efficient in computing symbolic solutions through given algorithms but cannot derive new theorems or lemmas. On the other side, automated theorem provers (ATP) have achieved remarkable results in proving non-trivial mathematical theorems. But they lack embedded mathematical knowledge such as algebraic algorithms

or intelligible representations of proofs and they are difficult to use. Moreover, they require huge search spaces.

It is thus natural to integrate theorem proving and symbolic mathematical computing in a common environment. We report on such an environment, $\lambda\epsilon\mu\mu\alpha$[1], which enables to compute with algebraic algorithms, to derive theorems, to deal with vertical or inclusion polymorphisms, and to learn and apply equation schemata. The explicit formalization of mathematical dependencies provides new possibilities to explain the solution steps.

This paper is structured as follows. Section 2 illustrates different levels of interaction between a symbolic calculator using algebraic algorithms and a theorem prover. A common mathematical knowledge base stores the domain specific problem solving knowledge and is described in section 3. An overview of $\lambda\epsilon\mu\mu\alpha$ in section 4 is followed by some concluding remarks in the last section.

2 Mathematical Problem Solving by Algorithms and Theorems

When solving problems, mathematicians follow a 'Mathcycle' [VeVe94]: conception, naive formulation, exploration, tentative proof, formulation, proof, publication, education, and use. Many packages which aid mathematicians in some of these steps have been developed, e.g. AM [LeBr84] for concept formulation, CAS for application of algorithms, ATP for verification and discovery of theorems, specification languages and knowledge representation. However, few mathematicians use these systems as everyday tools, because of some severe drawbacks which make them hard to use.

Classical CAS provide thousands of sophisticated algebraic algorithms which are difficult to handle by users. On the one hand, it is hard to select the appropriate algorithm from the amount of available algorithms, on the other hand, the interpretation of the solution needs deep mathematical understanding. The user doesn't receive any information about the solution steps from the system (Why is the output the solution to the given problem, or how to find the solution to a problem?). The mathematical knowledge is embedded implicitly in the algorithms and is inaccessible to the user, e.g. commutativity of polynomial addition, axioms of groups. However, the algorithmic encoding leads to high efficiency.

In traditional theorem provers it is difficult to specify axioms in the provers language, usually a first-order language and a normal form. Therefore, the representation of mathematical concepts (e.g. gcd, finite fields) is awkward and unnatural. Provers usually lack embedded algebraic algorithms. Although OTTER [McCu94] allows the declaration of user-defined functions together with their corresponding argument and result types, the extension of the system by new algorithms is very expensive, i.e. new implementation of the function and recompilation of the whole system. ATP compute huge search spaces and are inefficient. Additionaly, long and complex proofs are difficult to understand and

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

should provide representations that point out the essential steps of the proof. However, their success in proving hard mathematical theorems is impressing. In contrast to algorithmic problem solvers, theorem provers provide proofs to explain their solutions.

We propose the integration of CAS and ATP. This integration can be achieved in different ways as illustrated in figure 1.

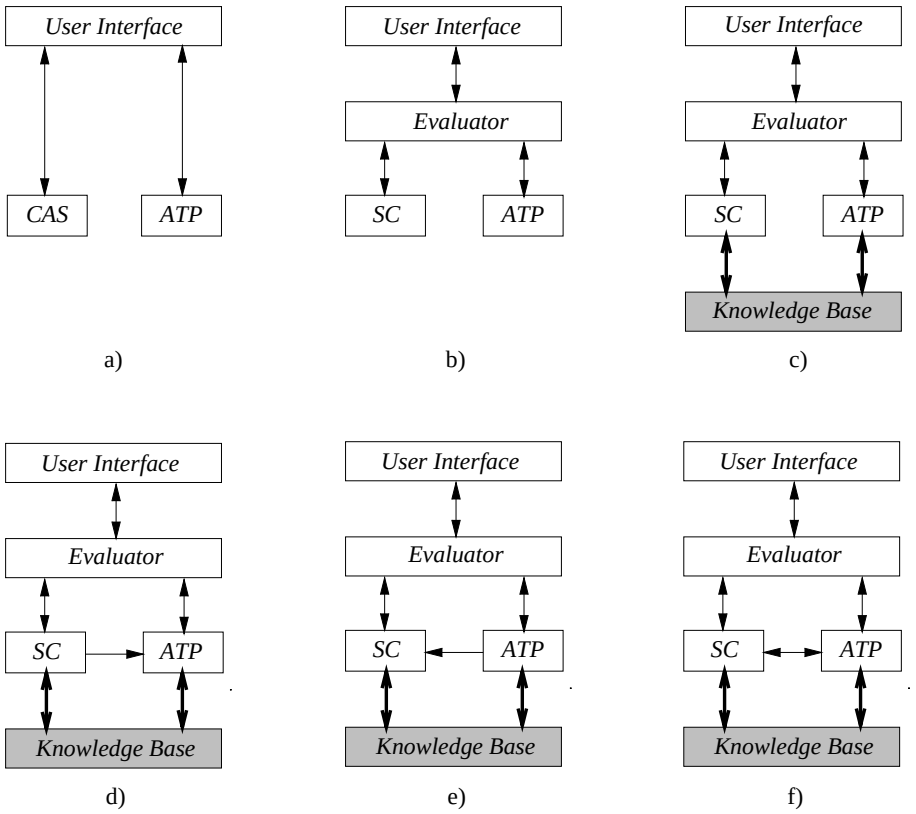

Fig. 1. Forms of interaction between algorithms and theorems

a) In the simplest case of interaction the user interface only provides a link to a CAS and an ATP, respectively. A user can access both systems and can apply algorithms or theorems to solve a given problem, depending on the class of the problem. Only this type of interaction allows the use of arbitrary CAS and ATP. However, the systems do not interact directly and a user must be familiar with both systems. Such an architecture combines the advantages, but also the drawbacks.

Example:

CAS: adding 0 to a polynomial in a polynomial ring is done by a function.
ATP: tries to prove that a left neutral element is also right neutral in a group.

b) Symbolic calculator (SC) and ATP can be extended by a common control unit (evaluator). This evaluator controls the selection of the modules by meta-knowledge on all functions and predicates. It also controls the application of algebraic algorithms in the SC and of theorems. The mathematical knowledge is represented separately in each module.

c) Algorithms often require type information about their arguments to be applicable. This information can be derived by theorems assuming that both modules share a common knowledge representation. This mathematical knowledge base also consists of meta-information on algorithms in form of algorithm schemata (cf. next section, e.g. figure 2).

Name	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	$\gcd(u, v) = \gcd(v, u)$ $\gcd(u, v) = \gcd(v, u \bmod v)$ $\gcd(u, 0) = u$
Function	

Fig. 2. Schema of algorithm gcd

Algebraic algorithms offer no capabilities to explain their solutions. These explanations can be generated using the theorem prover and the mathematical specification of the algorithms. The knowledge representation of both symbolic calculator and theorem prover must be adjusted to a common representation. With this form of interaction, theorems are not available within algorithms because the SC cannot access directly the ATP.

d) As mentioned in b), algorithms can be used for the efficient computation of predicates when proving theorems. However, the interaction in b) 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 from SC to ATP allows the immediate call of an algorithm out of a proof. New versions of theorem provers (e.g. OTTER 3.0) allow the introduction of user-defined algorithms which must be identified by a special character (e.g. $GCD). The extension of the prover requires the recompilation of the whole system and each algorithm has to be implemented in C. CAS provide an extensive collection of very efficient mathematical algorithms, thus reimplementation

is neither necessary nor meaningful. This kind of interaction would lead to a strong improvement of the efficiency of a theorem prover.

Example:

SC: various efficient algorithms for gcd calculation.
ATP: OTTER allows the definition of simple functions (e.g. gcd in figure 3). The performance can be increased strongly by calling instead the adequate gcd algorithm of the SC.

```
gcd(x,y) =       % greatest common divisor for nonnegative integers
   $IF($EQ(x,0),
       y,
       $IF($EQ(y,0),
           x,
           $IF($LT(x,y),
               gcd(x,$DIFF(y,x)),
               gcd(y,$DIFF(x,y))))).
```

Fig. 3. Definition of function gcd in OTTER

e) The application of theorems is useful even when running algebraic algorithms (e.g. verification of conditions, properties of objects). This kind of integration (same is true for f)) requires to redesign new algorithms to use the prover. The advantage lies in using the powerful reasoning capabilities of the theorem prover in the SC.

Example:

SC: ... if #IsNormal(G,H) then ... [2]
ATP: tries to prove that all subgroups of index 2 are normal (figure 4).

f) A complete integration of algorithms and theorems is achieved by combining d) and e). At any step, arbitrary combinations of algorithms and theorems can be applied to solve a given problem. This combines the advantages of a) to e), but requires to fit SC and ATP to a common knowledge representation.

Example:

SC: in Berlekamp algorithm ... if #SquareFree(p) then ...
ATP: $\forall f \in \mathbb{Z}_p[x] : SquareFree(f) \Leftrightarrow \$GCD(f, f') = 1$.
The SC can be used to compute the derivation of p and the gcd.

We have shown different levels of interaction between SC and ATP. The complete integration in f) requires the development of a new common semantics of SC and ATP, the reengineering of some algorithms, and the common explicit

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

```
%  existence of inverse
     4 P(x,g(x),e).
%  closure
     5 P(x,y,f(x,y)).
%  associative property
     6 -P(x,y,u) | -P(y,z,v) | -P(u,z,w) | P(x,v,w).
     7 -P(x,y,u) | -P(y,z,v) | -P(x,v,w) | P(u,z,w).
%  the operation is well defined
    10 -P(z,y,x) | -P(w,y,x) | EQUAL(z,w).
%  Denial of the theorem
    28 H(b).
    29 P(b,g(a),c).                    PROOF:
    30 P(a,c,d).
    31 -H(d).                          195 (29,7,4,5,37,33) P(c,a,b).
%  demodulators                        213 (195,7,30,5) P(d,a,f(a,b)).
    33 EQUAL(f(x,e),x).                716 (213,6,5,4,38,39) P(d,g(b),e).
    37 EQUAL(g(g(x)),x).               721 (716,10,4) EQUAL(d,b).
    38 EQUAL(g(f(x,y)),f(g(x),g(y))).  722 (721) EQUAL(d,b).
    39 EQUAL(f(x,f(g(x),y)),y).        729 (31,722,28) .
```

Fig. 4. Proof using hyperresolution and standard p-formulation of the theorem: all subgroups of index 2 are normal.

representation of objects and mathematics in a mathematical knowledge base. The construction of this memory is described in the next section.

3 The Mathematical Knowledge Base

The mathematical knowledge base consists of type schemata, algorithm schemata, algebraic algorithms, theorems, symbol tables, and normal forms. In this paper, we will not discuss the representation of algebraic algorithms and theorems, because they are exclusively used by the prover or CA engine. Thus, a unique treatment, e.g. by defining theorem schemata, is desirable but does not improve the collaboration of both systems. However, it would be required to verify algorithms and generate theorems automatically.

The theory of algebraic specification provides a good framework to design the type system of a mathematical assistant. We adopt the specification language Formal-Σ [CaTj93] 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 reuse 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:

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

Figure 5 shows the schemata of some selected ACS (more details may be found in [CHT92]). These definitions build a based-on hierarchy of the mathematical domains of computation (figure 6).

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:* $(f, +)$, $(ne, 0)$, $(inv, -)$)
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: $(y + z) \times x = (y \times x) + (z \times x)$

Fig. 5. Type schemata for Monoid, Group, and Ring.

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. They allow 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 domain and range
- *Definition*, mathematical description of the output
- *Subalgs*, list of subalgorithms describing the embedded subtasks

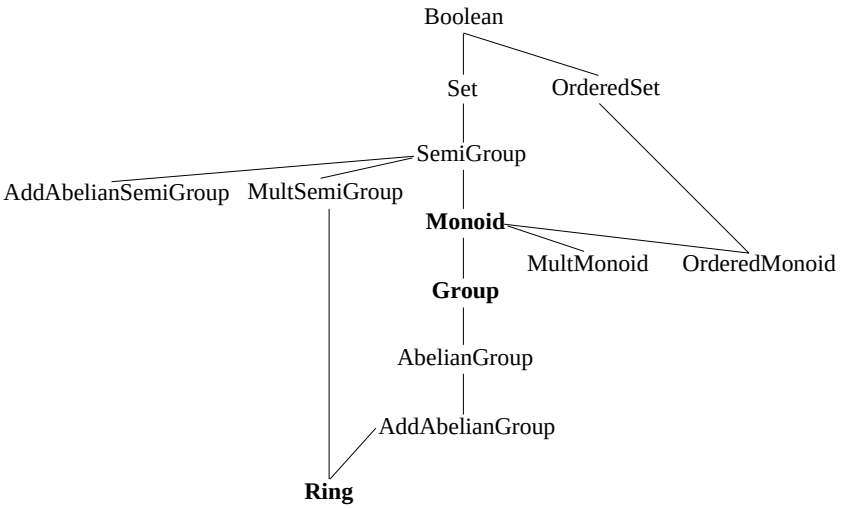

Fig. 6. Hierarchy of type schemata

- *Theorems*, describing properties of the algorithm
- *Function*, name of the corresponding executable algebraic function to compute the output.

Name	gcd-primitive($?a, ?b$) $=?g$
Signature	$?A \times ?A \rightarrow ?A$
Constraints	isa ($?A$, UnivariatePolynomial(x, UFD))
Definition	
Subalgs	primitive-part pseudo-remainder content gcd multiply
Theorems	
Function	GcdPrimitive

Fig. 7. Schema of algorithm gcd-primitive

Similarly to type and equation schemata, algorithm schemata build a hierarchy of specialized versions, and specializations inherit definitions and theorems from more general algorithms. Examples of algorithm schemata are given in figures 2 & 7, figure 8 describes parts of the hierarchy of algorithm schemata. New properties of algorithms can be derived by the theorem prover.

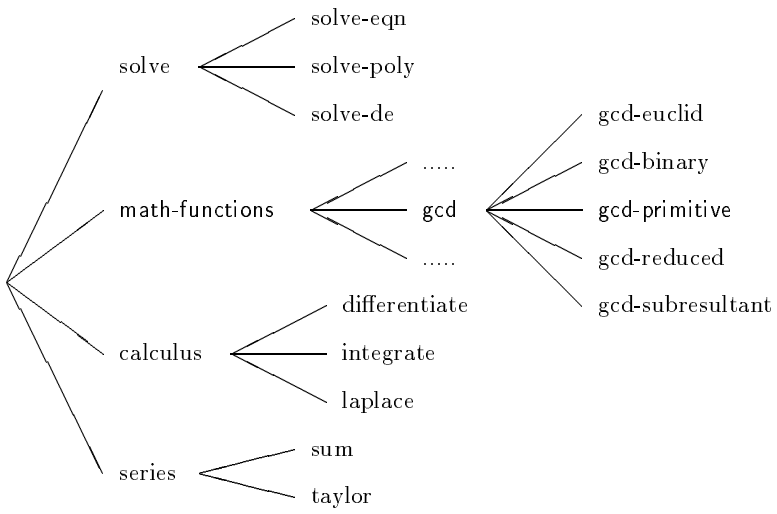

Fig. 8. Hierarchy of algorithm schemata

We introduced different kinds of interactions between SC and ATP. The mathematical knowledge is represented in a common knowledge base which consists of symbol tables, normal forms, theorems, algebraic algorithms, type and algorithm schemata. An environment corresponding to the interaction described in figure 1 f) is introduced in the next section.

4 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 9. 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.

The user interface offers frames and graphs for handling schemata and displays the explanations about solutions of specific problems. An evaluator solves these problems by using a theorem prover and a 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. Given

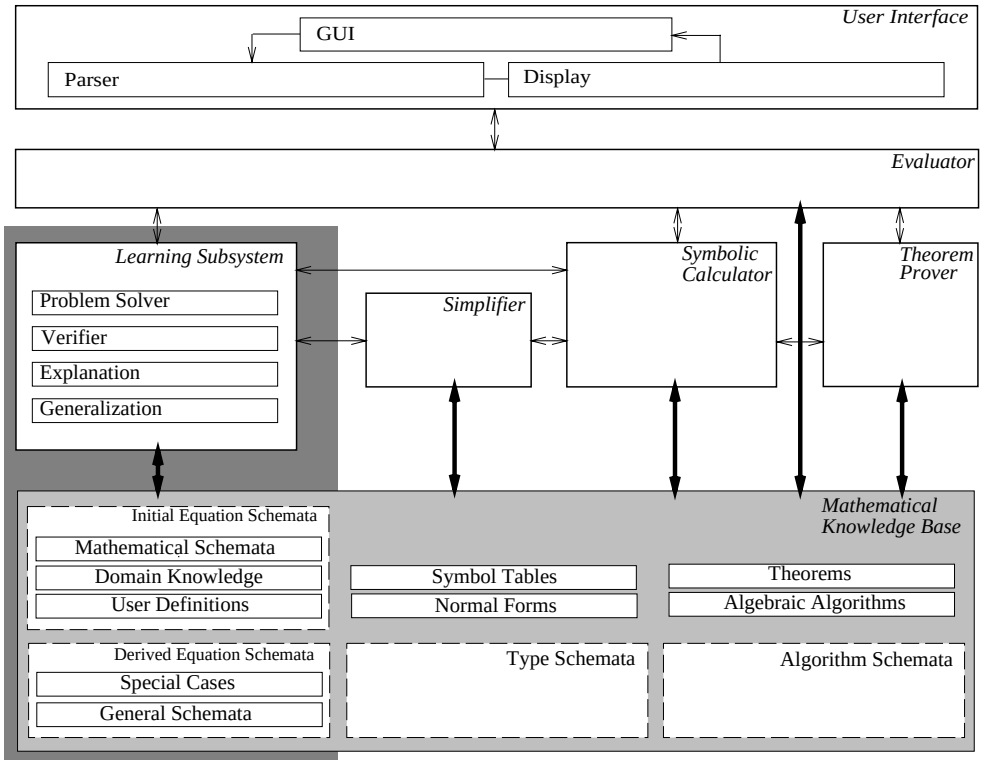

Fig. 9. Architecture of the intelligent environment for symbolic computing

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 with the new generalized schema.

5 An Example

For the purpose of having prototype systems, we created two ad hoc interfaces between theorem provers and CAS. These interfaces are controled by a common evaluator and implemented in CLisp and C respectively, however, our aim is to approach the integration at a higher level of interaction corresponding to figure 1 f).

The first prototype combines DTP [Gedd94], a simple first-order theorem prover, and Magma [BoCa94], a CAS for computations in algebra and particularly in group theory. The application of Magma algorithms is guided by DTP, which can solve new problems, e.g. finding elements with minimal index, and

prove difficult properties by induction and applications of algorithms. Additionaly, the tremendous knowledge about groups represented in MAGMA is accessible by the prover.

Another prototype combines ISABELLE [Paul94], a generic theorem prover supporting set theory, type theory, higher order logic ..., and MAPLE [Heck93], a well known commercial CAS. Many domains of computation were defined in the provers language (e.g. numbers, polynomials) and new theorems were proven by the cooperation with MAPLE.

Example:

$$\text{Proof of } \forall n \geq 5 \ : \ n^5 \leq 5^n$$

by using ISABELLEs induction theorem:

```
[| n: Nat; a: Nat; P(a);
   !!x. [| x: Nat; a <= x; P(x) |]  ==>  P(x + 1);
   a <= n |]  ==>  P(n)
```

start: $5^5 \leq 5^5$, true by reflexivity of $\leq$
step and new goal: $n^5 \leq 5^n \Rightarrow (n+1)^5 \leq 5^{(n+1)}$

MAPLE is used to expand both sides:
$n^5 + 5 * n^4 + 10 * n^3 + 10 * n^2 + 5 * n + 1 \leq 5 * 5^n$
and the prover is trying to prove the upper bounds:
$n^5 \leq 5^n$ (1), $5 * n^4 \leq n^5 \leq 5^n$ (2), $10 * n^3 \leq n^5 \leq 5^n$ (3),
$10 * n^2 \leq n^5 \leq 5^n$ (4), $5 * n + 1 \leq n^5 \leq 5^n$ (5).
Again, these bounds are proven by an interaction of both systems, e.g. the second bound is derived by ISABELLEs rule

```
le_mult
[| a: Nat; b: Nat; c: Nat; d: Nat;
   a <= b; c <= d |]  ==>  a * c <= b * d
```

and $5 \leq n$ (given) and $n^4 \leq n^4$ (reflexivity). Finally, MAPLE transforms the conclusion $5 * n^4 \leq n * n^4$ to $5 * n^4 \leq n^5$.

6 Conclusion

We have outlined several advantages of combining theorem proving and symbolic mathematical computing. On the one hand, computer algebra systems profit from theorem provers, e.g. by explanations of the solution of algorithms and verification of properties of mathematical objects. On the other hand, they offer an extensive collection of efficient mathematical algorithms which can improve the efficiency of the theorem prover.

A high level of interaction requires a common representation of the mathematical knowledge of the two systems. Such a knowledge base mainly consists of type and algorithm schemata, algebraic algorithms and theorems. The adopted

specification language for the specification of type schemata provides executability and offers a type system for both symbolic calculator and theorem prover.

Among the work in progress is the design of a "language" for the environment whose semantics allows a consistent treatment of algorithms and theorems, tools for the verification of algorithm schemata, extraction and learning of theorems out of algebraic algorithms, generation of algorithms from theorems, interaction of the learning component and the theorem prover and applications of the environment.

References

[BoCa94] W. Bosma, J. Cannon, *Handbook of MAGMA Functions*, Sydney, 1994.

[CHT92] J. Calmet, K. Homann, I.A. Tjandra, *Unified Domains and Abstract Computational Structures*, in J. Calmet, J.A. Campbell (eds.), 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.

[Gedd94] D. Geddis, *The DTP Manual*, Stanford University, 1994.

[Heck93] A. Heck, *Introduction to MAPLE*, Springer, 1993.

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

[LeBr84] D.B. Lenat, J.S. Brown, *Why AM and EURISKO Appear to Work*, Artificial Intelligence 23, pp. 269–294, Elsevier, 1984.

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

[Paul94] L.C. Paulson, *ISABELLE: A Generic Theorem Prover*, LNCS 828, Springer, 1994.

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

[VeVe94] A. Vella, C. Vella, *Artificial Intelligence and the Mathcycle*, in J.H. Johnson, S. McKee, A. Vella (eds.), Artificial Intelligence in Mathematics, Oxford University Press, 1994.

This article was processed using the LaTeX macro package with LLNCS style