

Integrating Computer Algebra with Proof Planning

Manfred Kerber¹, Michael Kohlhase^{*2}, Volker Sorge²

¹ School of Computer Science, The University of Birmingham
Birmingham B15 2TT, England
M.Kerber@cs.bham.ac.uk

² Fachbereich Informatik, Universität des Saarlandes
D-66141 Saarbrücken, Germany
{kohlhase|sorge}@cs.uni-sb.de

Abstract. Mechanised reasoning systems and computer algebra systems have apparently different objectives. Their integration is, however, highly desirable, since in many formal proofs both of the two different tasks, proving and calculating, have to be performed. In the context of producing reliable proofs, the question how to ensure correctness when integrating a computer algebra system into a mechanised reasoning system is crucial. In this contribution, we discuss the correctness problems that arise from such an integration and advocate an approach in which the calculations of the computer algebra system are checked at the calculus level of the mechanised reasoning system. We present an implementation which achieves this by adding a verbose mode to the computer algebra system which produces high-level protocol information that can be processed by an interface to derive proof plans. Such a proof plan in turn can be expanded to proofs at different levels of abstraction, so the approach is well-suited for producing a high-level verbalised explication as well as for a low-level (machine checkable) calculus-level proof.

1 Introduction

Mechanised Reasoning Systems (for short MRS in the following) may be built with various concrete purposes in mind. One goal is the construction of an autonomous theorem prover, whose strength achieves or even surpasses the ability of human mathematicians. Another may be to build a system where the user derives the proof, with the system guaranteeing its correctness. A third purpose might be the modelling of human problem-solving behaviour on a machine, that is, cognitive aspects are the focus. Advanced theorem proving systems often try to combine the different goals, since they can complement each other in an ideal way. However the ultimate motivation behind all of these systems is to give the human user assistance in formal (mathematical) reasoning tasks, and in particular theorem proving.

While all of the the approaches are in principle general enough to cope with any kind of proofs, they often neglect the fact that for many mathematical fields, everyday reasoning only partially consists in proving theorems. Calculation plays an equally important role, and mathematicians want to have support in both

* This work was supported by the Deutsche Forschungsgemeinschaft in SFB 314 (D2)

activities. More often than not the tasks of proving theorems and calculating simplifications of certain terms are interwoven and inseparable. In such cases traditional MRS will only provide rather poor support to a user, since they are very weak, when it comes to computation with mathematical objects.

In contrast, computer algebra systems (CAS in the following) manipulate highly optimised representations of the objects, which makes them very efficient. However they only provide answers to computations, but no proofs.

Although theoretically any computation can be reduced to theorem proving, this is not practical for non-trivial cases using traditional MRS, since the search spaces there are intractable. For many of these tasks, however, no search is necessary at all, since there are numerical or algebraic algorithms that can be used. If we think of Kowalski's equation "Algorithm = Logic + Control" [Kow79], general purpose MRS procedures do not (and can not) provide the control for doing the concrete computations that are explicitly encoded in CAS.

All of these facts point to the usefulness of an integration of CAS into MRS, where the former are used for solving subgoals in mathematical deduction. Consequently several experiments on combining CAS and MRS have been carried out recently. As pointed out by Buchberger [Buc96] the integration problem is still unsolved, but it can be expected that a successful combination of these systems will lead to "a drastic improvement of the intelligence level" of such support systems.

We will give an overview over these experiments in Sect. 3, advocate a particular approach built on top of the proof planning paradigm and describe the implementation of a prototype in Sect. 4. But let us first take a closer look at MRS in general and our Ω -MKRP system in particular.

2 Mechanised Reasoning and the Role of Proofs

Let us roughly divide existing theorem-proving systems into three categories: machine-oriented theorem provers, proof checkers, and human-oriented (plan-based) theorem provers. By *machine-oriented theorem provers* we mean theorem provers based on computational logic theories such as resolution, paramodulation, or the connection method. *Interactive proof checking* and *proof development systems* have been developed to carry out the meticulous final checking of proofs. And finally *human-oriented theorem-proving systems* incorporate some human problem solving behaviour. This is achieved for instance by using tactics, programs that manipulate proof-states by a series of calculus steps (first used in LCF [GMW79]) or by proof planning with so-called methods, which are tactics extended by specifications (e.g. CLAM [BvHHS90, BSvH⁺93]).

Normally all these systems do not exist in a pure form anymore, and in some systems like our own Ω -MKRP system [HKK⁺94] it is explicitly tried to combine the reasoning power of automated theorem provers as logic engines, the specialised problem solving knowledge of the proof planning mechanism, and the interactive support of tactic-based proof development environments.

MRS can also be classified with respect to the role proofs play in them. There are essentially two different views of proofs, the *realist's* (also called *Platonist*) and the *nominalist's* [Pel91]. A realist accepts abstract properties of proofs, in

particular he/she is satisfied with the evidence of the existence of a proof in order to accept the truth of a theorem. A proof for a nominalist, on the other hand, makes only sense with respect to a particular calculus, hence he/she only accepts concrete proofs formulated in this calculus. The advantage of adopting the realist position is that reasoning systems can be built (and meta-theoretically extended) without bothering about the concrete construction of proofs. The advantage of the nominalist position is that it preserves the tradition that proofs can be communicated: The nominalist position guarantees the correctness of machine generated proofs without violating an essential of the traditional notion of proof, namely the possibility to communicate them. Furthermore explicit proofs can be checked by simple proof checkers and this seems currently to be the only way to ensure the correctness of proofs generated by large computing systems, which are inevitably error-prone.

In accordance with the two philosophical positions there are two possible ways to use an MRS: as trustworthy black box (trustworthy, for instance, since there are a lot of meta-arguments, why the system works properly) or as a system that produces communicable and checkable proofs.

In the Ω -MKRP-system to be described in the following, we advocate the stricter nominalist approach. The main reason for this is the reliability argument. Clearly, the sheer size of the systems involved prohibits to come up with provably correct MRS, but in particular makes it impossible to add different systems to an MRS like an external CAS *without* giving up any correctness requirement. In Ω -MKRP, a human user can apply different integrated tools to manipulate a proof tree, which stores the current partial natural deduction proof. In particular, new pieces of proof can be added by calling so-called methods, programs that store some proof information, by inserting facts from a data base, or by calling some external automated theorem prover (see Fig. 1). Furthermore the user can call the Proverb system for the generation of an abstract proof that can be verbalised. Finally, there is the possibility to call a computer algebra system as we will describe in the rest of the paper.

Since the correctness of the different components (in particular of the external ones) cannot be guaranteed, the final proof has to be checked by a simple verifier equipped with a fixed set of natural deduction rules. The soundness of the overall system only relies on the correctness of this checker and the correctness of the natural deduction rules. The price we have to pay for this is that in our approach each component must protocol its results in some universal format (in our case, a variant of Gentzen's calculus NK of natural deduction).

We are not going to present Ω -MKRP in great detail, most of its components are quite standard and are not important for the purpose of this paper. A main component, which is important for the integration of computer algebra into Ω -MKRP is its planning component. The entire process of theorem proving in Ω -MKRP can be viewed as an *interleaving process* of proof planning, method execution (the plan operators are called methods and essentially are tactics plus specifications) and verification. In particular, this model ascribes a problem-solver's reasoning competence to the existence of methods together with a planning mechanism that uses these methods for proof planning.

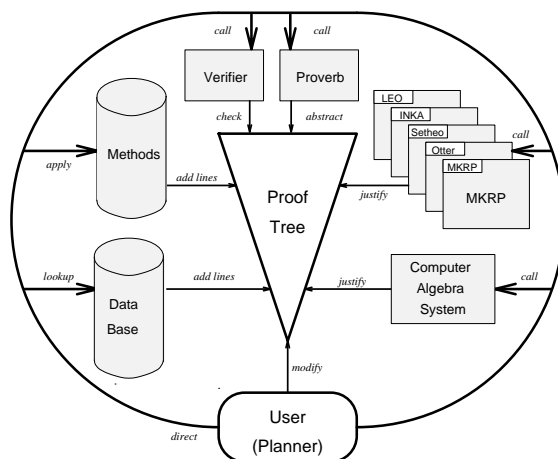


Fig. 1. Architecture of Ω -MKRP

3 Syntheses of Reasoning and Calculation

We will now categorise the experiments of integrating MRS and CAS into three categories with respect to the treatment of proofs that is adopted.

In the attempts belonging to the first category (see e.g. [CZ92, BHC95]) one essentially trusts that the CAS properly work, hence their results are directly incorporated into the proof. The range of mathematical theorems that can be formally proved by the system combinations is much greater than that provable by MRS alone. However, CAS are very complex programs and therefore only trustworthy to a limited extent, so that the correctness of proofs in such a hybrid system can be questioned. The second category [HT93] is more conscious about the role of proofs, and only uses the CAS as an oracle, receiving a result, whose correctness can then be checked deductively. While this certainly solves the correctness problem, this approach only has a limited coverage, since even checking the correctness of a calculation may be out of scope of most MRS, when they don't have additional information. A third approach of integrating computer algebra systems into a particular kind of mechanised reasoning system, consists in the meta-theoretic extension of the reasoning system as proposed for instance in [BM81, How88]. In this approach a constructive mechanised reasoning system is basically used as its own meta-system, the constructive features are exploited to construct a correct computer algebra system and due to bridge rules it can directly be used in the base system.

The main problem of integrating CAS into MRS without violating correctness requirements is that CAS are generally highly optimised towards maximal speed of computation but not towards generating explanations of the computations involved. Not only if one is interested in correctness, but in particular if explanations for theorems and calculations are required, this can turn out to be a major problem; in such a case explicit proofs (calling for a nominalist approach) are essential.

If we take the idea of generating explicit proofs and explanations seriously

also for computations and do not want to give up the nominalist paradigm, we can neither just take existing CAS nor follow the approach of meta-theoretic extensions, since Ω -MKRP is a classical proof system and does not use constructive logic. But we cannot forgo using them either, if we want to tackle nontrivial computations inside proofs. For instance even the proof for the binomial formula $(x + y)^2 = x^2 + 2xy + y^2$ (a trivial problem for any computer algebra system) needs more than 70 single steps in the natural deduction calculus³. Thus using theorem provers or rewriting systems to find such proofs can produce unnecessarily large search spaces and thus absorb valuable resources. On the other hand such proofs show a remarkable resemblance to algebraic calculations themselves and suggest the use of a CAS not only to instantly compute the result of the given problem, but also to guide a proof in the way of exploiting the implicit knowledge of the algorithms. We propose to do this extraction of information not by trying to reconstruct the computation in the MRS after the result is generated—as we have seen, even in case of a trivial example for a CAS this may turn out to be a very hard task for an MRS—but rather to extend the CAS algorithm itself so that it produces some logically usable output alongside the actual computation.

The novel contribution of our approach is to use the mathematical knowledge implicit in the CAS to extract proof plans that correspond to the mathematical computation in the CAS. So essentially the output of a CAS should be transferable into a sequence of tactics, which presents a high-level description for the proof of correctness of the computation the CAS has performed. Note that this does not prove general correctness of the algorithms involved, instead it only gives a proof for a particular instance of computation. The high-level description can then be used to produce a readable explanation or evaluated to check the proof. If we want to check the whole derivation, these proof plans can then be expanded into detailed natural deduction proofs. The decision to extract proof plans rather than concrete proofs from the CAS is essential to the goal of being verbose without transmitting too much detail.

For our purpose, we need different modes, in which we can use the CAS. Normally, during a proof search, we are only interested in the result of a computation, since the assumption that the computation is correct is normally justified for established CAS. When we want to understand the computation—in particular, in a successful proof—we need a verbose mode of the CAS that gives enough information to generate a high-level description of the computation in terms of the mathematics involved. How this can be achieved is described in the next section in detail.

4 SAPPER – Integrating Computations into Proofs

The SAPPER system (**S**ystem for **A**lgorithmic **P**roof **P**lan **E**xtraction and **R**easoning), integrates a prototypical computer algebra system into a proof plan-based mechanised reasoning system. The system is kept generic, but for the concrete integration we have used the Ω -MKRP-system as MRS and a self-written

³ Proofs of this length are among the hardest ever found by totally automatic theorem provers without domain-specific knowledge.

CAS, called μ -CAS. As mentioned in the previous section, for the intended integration it is necessary to augment the CAS with mathematical information for a *verbose mode* in order to achieve the proposed integration at the level of proofs. The μ -CAS-system is very simple and can at the moment only perform basic polynomial manipulations and differentiation, but it suffices for demonstrating the feasibility of our approach. Clearly, for a practical system for mathematical reasoning, a much more developed system like Maple [CGG⁺92], Reduce [Hea87], or Mathematica [Wol91] has to be integrated. Enriching such a large CAS with a corresponding verbose mode for producing additional protocol information, would of course require a considerable amount of work.

The SAPPER system can be seen as a generic interface for connecting Ω -MKRP with one or several CAS (see Fig. 2). An incorporated CAS is treated as a slave to Ω -MKRP which means that only the latter can call the first one and not vice versa. From the software engineering point of view, Ω -MKRP and the CAS are two independent processes while the interface is a process providing a bridge for communication. Its role is to automate the broadcasting of messages by transforming output of one system into data that can be processed by the other⁴.

Unlike other approaches (see [GPT94, HC95] for example) we do not want to change the logic inside our prover. In the same line, we do not want to change the computational behaviour of the computer algebra algorithms. In order to achieve this goal the trace output of the algorithm is kept as short as possible. In fact most of the computations for constructing a proof plan is left to the interface. The proof plans can directly be imported into Ω -MKRP.

This makes the integration independent of the particular systems, and indeed all the results below are independent of the CAS employed and make only some general assumptions about the MRS (such as being proof plan-based). Moreover, the interface approach helps us to keep the CAS free of any logical computation, for which such a system is not intended anyway. Finally, the interface minimises the required changes to an existing CAS, while maintaining the possibility of using the CAS stand-alone. The only requirement we make for integrating a particular CAS is that it has to produce enough protocol information so that a proof plan can be generated from this information. The proof plan in turn can be expanded by the MRS into a proof verifying the concrete computation.

The interface itself can be roughly divided into two parts; the *translation part*, and the *plan-generator*. The first performs syntax translations between Ω -MKRP and a CAS in both directions while the latter only transforms verbose output of the CAS to Ω -MKRP proof plans. Clearly only the translation part depends on the particular CAS that is invoked.

For the translations a collection of data structures—called *abstract CAS*—is provided each one referring to a particular connected CAS (or just parts of one). The main purpose of these structures is to specify function mappings, relating a particular function of Ω -MKRP to a corresponding CAS-function and the type of its arguments. Furthermore it provides functionality to convert the given arguments of the mapped Ω -MKRP function to CAS input. In the same fashion it transforms results of algebraic computations back into data that can be further processed by Ω -MKRP. These syntax transformations are implemented by two

⁴ This is an adaptation of the general approach on combining systems in [CMP91].

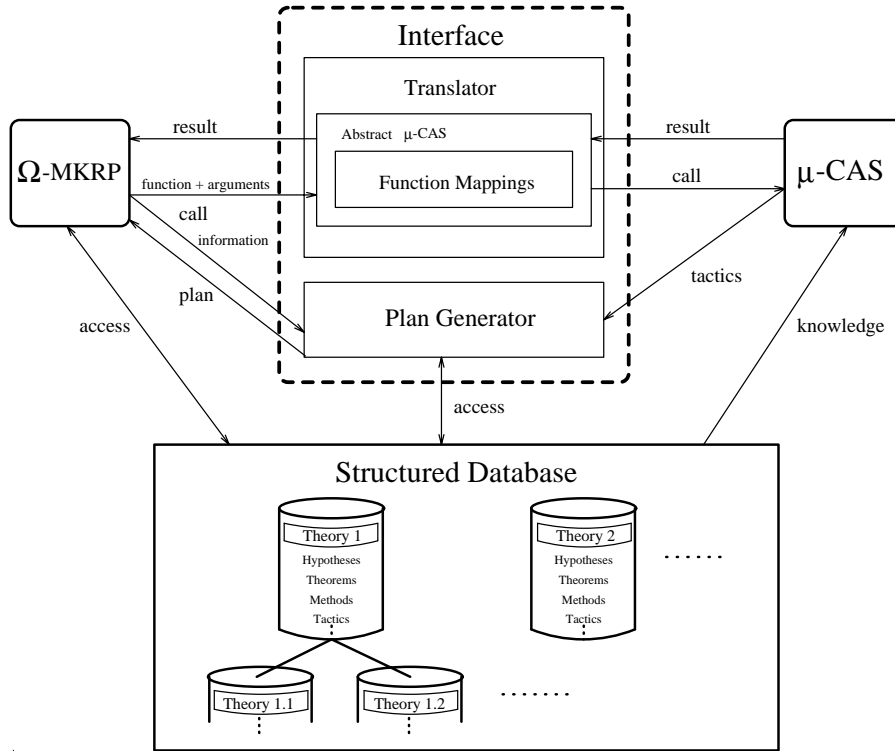


Fig. 2. Interface between Ω -MKRP and Computer Algebra Systems

generic functions, one for each direction of data flow. The single methods for each function are selected according to the type of objects send to and returned from the CAS. It is possible to expand the translation unit generically by simply adding new abstract CAS and methods for the two translation functions. The functionality in this part of our interface offers us the possibility of connecting any CAS as a black box system. For instance, we may want to do use a very efficient system without verbose mode for proof search as black box system, and then another less efficient system with verbose mode for the actual proof construction, once it is clear what the proof should look like.

The plan-generator solely provides the machinery for our main goal, the proof plan extraction. Equipped with supplementary information on the proof by Ω -MKRP it records the output produced by the particular algebraic algorithm and converts it into a proof plan. Here the requirements of keeping the CAS side free of logical considerations and on the other hand of keeping the interface generic seem conflicting at the first glance. However this conflict can be solved by giving both sides of the interface access to a data base of mathematical facts formalising the mathematics behind the particular CAS algorithms. Conceptually, this data base together with the mappings governing the access, provides the semantics of the integration of Ω -MKRP with a particular CAS. Thus expanding the plan-generator is simply done by expanding the database by adding new tactics.

Such a database is needed independently of the usage in the integration of CAS by the proof planner for storing and structuring the definitions, theorems, proofs, and methods of a given mathematical domain. In fact, to Ω -MKRP the mathematics behind the CAS algorithms is just another special domain. The data-base is structured in a hierarchical system of theories and sub-theories. Such a theory is a collection of definitions, type information and axioms, theorems, and lemmata derivable from these axioms. Moreover it contains methods, tactics, etc. usable by various kinds of proof planners (including the plan-generator for a particular CAS). In this setting an Ω -MKRP proof plan obtains a natural hierarchy corresponding to the structure of the theories. In particular, the hierarchical structure of this data-base is a valuable source for guiding the search for proof plans.

While Ω -MKRP itself can access the complete database, SAPPER's plan-generator in the interface is only able to use tactics and lookup hypotheses of a theory (cf. Fig. 2). The CAS does not interact with the data base at all: it only has to know about it and references the logical objects (methods, tactics, theorems or definitions) in the verbose mode. This verbose information of the CAS is returned as strings consisting of the name of the object and its appropriate arguments. Thus knowledge about the data base is compiled a priori into the algebraic algorithms in order to document their calculations.

5 An Example of the Integration

As an example of the integration we consider the case of extracting proof plans from a recursive algorithm for adding polynomials. Even though this algorithm is quite trivial from the CAS point of view, we present it here, since it sheds some light on the spirit of the integration.

Let us now take a look at the different representations of a polynomial $p(x_1, \dots, x_r) = \sum_{i=1}^n \alpha_i x_1^{e_{1i}} \cdots x_r^{e_{ri}}$ in the variables x_1, \dots, x_r . The logical language of Ω -MKRP is a variant of the simply typed λ -calculus, so the polynomials are represented as λ -expression where the formal parameters x_1, \dots, x_r are λ -abstracted (mathematically, p is a function of r arguments):

$$p : \lambda x_1 \cdots \lambda x_r. (+ (* \alpha_n (* (\uparrow x_1 e_{1n}) \cdots)) \cdots (* \alpha_1 (* (\uparrow x_1 e_{11}) \cdots))),$$

For the notation, we use a prefix notation; the symbols $+$, $*$ and \uparrow denote binary functions for addition, multiplication and exponentiation on the rationals. In this representation, we can use β -reduction for the evaluation of polynomials, but we have to define a special function for polynomial addition for any arity r .

In μ -CAS, we use a variable dense, expanded representation as an internal data-structure for polynomials (as described in [Zip93] for instance). Thus every monomial is represented as a list containing its coefficient together with the exponents of each variable. p is represented as $((\alpha_n e_{1n} \cdots e_{rn}) \cdots (\alpha_1 e_{11} \cdots e_{r1}))$ for instance.

Let us now turn to the actual μ -CAS algorithm for polynomial addition. This simple algorithm adds polynomials p and q by a case analysis on the

exponents with recursive calls to itself. So let $p = \sum_{i=1}^n \alpha_i x_1^{e_{1i}} \cdots x_r^{e_{ri}}$ and $q = \sum_{i=1}^m \beta_i x_1^{f_{1i}} \cdots x_r^{f_{ri}}$. We have presented the algorithm in the j th component of p and the k th component of q in a LISP-like pseudo-code in Fig. 3. Intuitively, the algorithm proceeds by ordering the monomials, advancing the leading monomial either of the first or the second arguments; in the case of equal exponents, the coefficients of the monomials are added.

```

(poly-add (p q)
  (= (e1j ... erj)(f1k ... frk))
    (tactic mono-add)
    (cons-poly (αj + βk)x1e1j ... xrerj
      (poly-add  $\sum_{i=j+1}^n \alpha_i x_1^{e_{1i}} \cdots x_r^{e_{ri}}$   $\sum_{i=k+1}^m \beta_i x_1^{f_{1i}} \cdots x_r^{f_{ri}}$ ))
  (> (e1j ... erj)(f1k ... frk))
    (tactic pop-first)
    (cons-poly αjx1e1j ... xrerj
      (poly-add  $\sum_{i=j+1}^n \alpha_i x_1^{e_{1i}} \cdots x_r^{e_{ri}}$   $\sum_{i=k}^m \beta_i x_1^{f_{1i}} \cdots x_r^{f_{ri}}$ ))
  (< (e1j ... erj)(f1k ... frk))
    (tactic pop-second)
    (cons-poly βkx1e1k ... xrerk
      (poly-add  $\sum_{i=j}^n \alpha_i x_1^{e_{1i}} \cdots x_r^{e_{ri}}$   $\sum_{i=k+1}^m \beta_i x_1^{f_{1i}} \cdots x_r^{f_{ri}}$ ))

```

Fig. 3. Polynomial addition in μ -CAS.

Obviously, the only expansions of the original algorithm needed for the verbose mode are the additional (tactic ...) statements⁵. They just produce the additional output by returning keywords of tactic names to the tactic generator and do not have any side effects. In particular, the computational behaviour of the algorithm does not have to be changed at all. If we apply the algorithm to the simple polynomials $p := 3x^2 + 2x + 5$ and $q := 5x^3 + x + 2$ we obtain the following verbose output: (pop-second, pop-first, mono-add, mono-add).

First the cubic monomial from q (the second argument) and then the quadratic one from p (the first argument) are raised, since they only appear in one argument, and finally the remaining monomials are summed up.

⁵ Observe that in this case, the called tactics do not need any additional arguments, since our plan-generator in the interface keeps track of the position in the proof and thus knows on which monomials the algorithm works when returning a tactic. This way we need not be concerned what form a monomial actually has during the course of the algorithm.

In this simple case, each of the verbose keywords directly corresponds to a tactic with the same name, so the verbose output directly represents a proof plan for polynomial addition of the concrete polynomials p and q .

Let us now take a look at the `pop-second` tactic to understand its logical content. The tactic itself describes a reordering in a sum that looks in the general case as follows:

$$(a + (b + c)) = (b + (a + c)) \quad (1)$$

For the current example we can view a and c as arbitrary polynomials and b as a monomial of rank greater than that of the polynomial a . It is now obvious that the behaviour of `pop-second` is determined by the pattern of the sum it is applied to. If in (1) the polynomial c does not exist, `pop-second` is equivalent to a single application of the law of commutativity. Otherwise, like in our example, the tactic performs a series of commutativity and associativity steps. In the example the tactic can be expanded in two steps. First we have a one step inference with `pop-second` corresponding to two lines in our proof. (In the actual proofs, the terms are of course embedded in contexts.)

$$\begin{array}{l} ((3x^2 + 2x + 5) + (5x^3 + x + 2)) \\ (5x^3 + ((3x^2 + 2x + 5) + (x + 2))) \end{array} \quad (\text{pop-second})$$

Expanding this plan adds two intermediate lines to the proof which then reflects the single step applications of the laws of commutativity and associativity.

$$\begin{array}{l} ((3x^2 + 2x + 5) + (5x^3 + x + 2)) \\ (((3x^2 + 2x + 5) + 5x^3) + (x + 2)) \quad (\text{associativity}) \\ ((5x^3 + (3x^2 + 2x + 5)) + (x + 2)) \quad (\text{commutativity}) \\ (5x^3 + ((3x^2 + 2x + 5) + (x + 2))) \quad (\text{associativity}) \end{array}$$

Finally to receive the complete calculus level proof for the computation step described by the `pop-second` tactic we further expand all three justifications of the above lines. This leads to a sequence of eliminations of universally quantified variables in the corresponding hypothesis, the axioms of commutativity and associativity. In our example the commutativity axiom would be transformed in the following fashion:

$$\begin{array}{l} \forall a \forall b. (a + b) = (b + a) \quad (\text{HYP}) \\ \forall b. ((3x^2 + 2x + 5) + b) = (b + (3x^2 + 2x + 5)) \quad (\forall E (3x^2 + 2x + 5)) \\ ((3x^2 + 2x + 5) + 5x^3) = (5x^3 + (3x^2 + 2x + 5)) \quad (\forall E 5x^3) \end{array}$$

Altogether this single application of the `pop-second`-tactic is equivalent to a calculus-level proof of 11 inference steps. The overall proof of this trivial polynomial addition has a length of 47 single step.

6 Conclusion

In this work we reported on an experiment of integrating a computer algebra system into the interactive proof development environment Ω -MKRP, not only at the *system* level, but also at the level of *proofs*. The motivation for such an integration is the need for a support of a human user when his/her proofs contain non-trivial computations. Unfortunately, we could not use a standard CAS for the integration, since such a system provides answers, but no justifications,

which turned out to be essential in an environment that is built to construct communicable and checkable proofs.

In order to achieve a solution that is compatible with such a strong requirement, we have adopted a generic approach, where the only requirement for the CAS is that it provides a verbose mode for the generation of communicable and checkable proofs. Since we want to achieve two goals simultaneously, namely to have high-level descriptions of the calculations of the CAS for communicating them to human users as well as low-level ones for a mechanical checking, we represent the protocol information in form of high-level hierarchical proof plans, which can be expanded to the desired detail. Fully expanded proof plans are natural deduction proofs which can be mechanically checked. In the case that the CAS has made a mistake in the computation, the proof checker can detect it on this level. Thus our approach provides a paradigm of verification of calculations that is much simpler than that of verifying the correctness of the CAS itself. The usefulness of the integration for proof development can already be seen in the case of our simple μ -CAS. After the integration we are able to prove optimization problems which were out of reach without such a support.

We have tested proof plan extraction from simple recursive and iterative CAS algorithms, where it works quite well. However, more complicated schemes like divide-and-conquer algorithms (e.g. the polynomial multiplication of Karatsuba and Ofman [KO63]) cannot be adapted to our approach so easily. Highly elaborated and efficient algorithms in systems like *Mathematica* [Wol91] or *Maple* [CGG⁺92] might be hard to augment with verbose modes. However, even if it proves impossible to extract verbose information that is valuable at the conceptual, mathematical level, it is always possible to reserve these elaborated techniques for the quiet mode used in proof discovery, and use more basic algorithms for the proof extraction phase. It may even be, that algorithms using elaborated data-structures and calculation schemes can contribute internal information that improves informed runs of simpler algorithms. It furthermore would be desirable to make use of sophisticated type systems and axiom specifications like those in *Axiom* [Dav92] or *MuPAD* [Fuc96] to return domain specific tactics in the verbose output.

References

- [BHC95] C. Ballarin, K. Homann, and J. Calmet. Theorems and algorithms: An interface between Isabelle and Maple. In A. H. M. Levelt, editor, *Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC'95)*, pages 150–157. ACM Press, 1995.
- [BM81] R. S. Boyer and J. S. Moore. Metafunctions. In R. S. Boyer and J. Strother Moore, editors, *The Correctness Problem in Computer Science*, pages 103–184. Academic Press, 1981.
- [BSvH⁺93] Alan Bundy, Andrew Stevens, Frank van Harmelen, Andrew Ireland, and Alan Smaill. Rippling: A heuristic for guiding inductive proofs. *AI*, **62**:185–253, 1993.
- [Buc96] Bruno Buchberger. Mathematische Software-Systeme: Drastische Erweiterung des “Intelligenzniveaus” entsprechender Programme erwartet. *Informatik Spektrum*, 19/2:100–101, 1996.

- [BvHHS90] Alan Bundy, Frank van Harmelen, Christian Horn, and Alan Smaill. The OYSTER-CLAM system. In Mark E. Stickel, editor, *Proceedings of the 10th CADE*, pages 647–648, Kaiserslautern, Germany, 1990. Springer-Verlag.
- [CGG⁺92] Bruce W. Char, Keith O. Geddes, Gaston H. Gonnnet, Benton L. Leong, Michael B. Monagan, and Stephen M. Watt. *First leaves: a tutorial introduction to Maple V*. Springer-Verlag, 1992.
- [CMP91] D. Clément, F. Montagnac, and V. Prunet. Integrated software components: A paradigm for control integration. In *Proceedings of the European Symposium on Software Development Environments and CASE Technology*, Springer-Verlag, LNCS 509, 1991.
- [CZ92] Edmund Clarke and Xudong Zhao. Analytica-A theorem prover in mathematics. In *Proceedings of the 11th CADE*, pages 761–763, Saratoga Springs, New York, 1992, Springer-Verlag.
- [Dav92] J. H. Davenport. The AXIOM system. AXIOM Technical Report TR5/92 (ATR/3) (NP2492), Numerical Algorithms Group, Inc., Downer's Grove, IL, USA and Oxford, UK, 1992.
- [Fuc96] Benno Fuchssteiner et al. (The MuPAD Group). *MuPAD User's Manual*. John Wiley and Sons, Chichester, first edition, 1996.
- [GMW79] Michael Gordon, Robin Milner, and Christopher Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*. LNCS 78. Springer-Verlag, 1979.
- [GPT94] Fausto Giunchiglia, Paolo Pecchiari, and Carolyn Talcott. Reasoning theories – towards an architecture for open mechanized reasoning systems. IRST-Technical Report 9409-15, IRST (Istituto per la Ricerca Scientifica e Tecnologica), Trento, Italy, June 1994.
- [HC95] K. Homann and J. Calmet. An open environment for doing mathematics. In M. Wester, S. Steinberg, and M. Jahn, editors, *Proceedings of 1st International IMACS Conference on Applications of Computer Algebra*, Albuquerque, USA, 1995.
- [Hea87] A. C. Hearn. Reduce user's manual: Version 3.3. Technical Report, Rand Corporation, Santa Monica, CA, USA, 1987.
- [HKK⁺94] Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Erica Melis, Dan Nesmith, Jörn Richts, and Jörg Siekmann. Ω -MKRP: A proof development environment. In Alan Bundy, editor, *Proceedings of the 12th CADE*, pages 788–792, Nancy, 1994. Springer-Verlag, LNAI 814.
- [How88] D. J. Howe. Computational Metatheory in Nuprl. In Lusk and Overbeek, editors, *Proceedings of CADE 9*, pages 238–257, 1988. Springer-Verlag.
- [HT93] J. Harrison and L. Théry. Extending the HOL theorem prover with a computer algebra system to reason about the reals. In C.-J. H. Seger J. J. Joyce, editor, *Higher Order Logic Theorem Proving and its Applications (HUG '93)*, pages 174–184, 1993. Springer-Verlag, LNCS 780.
- [KO63] A. Karatsuba and Y. Ofman. *Multiplication of Multidigit Numbers by Automata*. Soviet Physics-Doklady, 1963.
- [Kow79] Robert Kowalski. Algorithm = Logic + Control. *CACM*, 1979.
- [Pel91] Francis Jeffrey Pelletier. The philosophy of automated theorem proving. In John Mylopoulos and Ray Reiter, editors, *Proceedings of the 12th IJCAI*, pages 538–543, Sydney, 1991. Morgan Kaufmann.
- [Wol91] S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley, 1991.
- [Zip93] Richard Zippel. *Effective Polynomial Computation*. Kluwer Academic Press, 1993.