INTEGRATING COMPUTER ALGEBRA INTO PROOF PLANNING

Manfred Kerber
School of Computer Science, The University of Birmingham, Birmingham B15 2TT, England, e-mail: `M.Kerber@cs.bham.ac.uk`
URL: `http://www.cs.bham.ac.uk/~mmk`

Michael Kohlhase and Volker Sorge
Fachbereich Informatik, Universität des Saarlandes, D-66141 Saarbrücken, Germany, e-mail: `{kohlhase|sorge}@ags.uni-sb.de`
URL: `http://jswww.ags.uni-sb.de/{~kohlhase|~sorge}`

(Received ..... ; Accepted in final form .....)

Abstract. Mechanised reasoning systems and computer algebra systems have different objectives. Their integration is highly desirable, since formal proofs often involve both of the two different tasks, proving and calculating. Even more importantly, proof and computation are often interwoven and not easily separable.

In this contribution we advocate an integration of computer algebra into mechanised reasoning systems at the proof plan level. This approach allows to view the computer algebra algorithms as *methods*, that is, declarative representations of the problem solving knowledge specific to a certain mathematical domain. Automation can be achieved in many cases by searching for a hierarchic *proof plan* at the method-level using suitable domain-specific control knowledge about the mathematical algorithms. In other words, the uniform framework of proof planning allows to solve a large class of problems that are not automatically solvable by separate systems.

Our approach also gives an answer to the correctness problems inherent in such an integration. We advocate an approach where the computer algebra system 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.

We present an implementation of our ideas and exemplify them using an automatically solved example.

> Changes in the criterion of 'rigour of the proof' engender major revolutions in mathematics.
>
> H. Poincaré, 1905

Key words: mechanised reasoning, computer algebra, hierarchical proof planning, proof checking.

1. Introduction

The computer and the development of high-level programming languages made possible the mechanisation of logic as well as the realisation of mechanical symbolic calculations, we could witness in the last forty years. This has lead to two rather disjoint academic fields: mechanised reasoning and computer algebra, which each have their own methods, interests and traditions, even though they share common roots: none of the two fields is imaginable without the underlying foundation of mathematical logic or the mathematical study of symbolic calculations (leading to such algorithms and methods as the determination of the GCD or the Gaußian elimination). Only in the last decade we have seen a move towards an integration of the fields driven by the insight that real-world formal problems often involve a mixture of both computation and reasoning, hence an integration of mechanised reasoning systems and computer algebra systems is highly desirable (cf. [Buc85]). This is the case in particular, since deduction systems are very weak, when it comes to computation with mathematical objects, and computer algebra systems manipulate highly optimised representations of these objects, but do not yield any formally checkable proof information (if they give any explanation at all).

In the remainder of this introduction we briefly summarise key points of mechanised reasoning systems as well as of computer algebra systems and then give a short preview on the integration approach advocated in this paper. By its nature, such a short description has to abstract from many details and to simplify considerably.

1.1. Mechanised Reasoning Systems

Mechanised reasoning systems (for short MRS in the following) are built with various 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 is to build a system where the user derives the proof, with the system guaranteeing its correctness. A third purpose consists in modelling 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. Let us roughly divide existing theorem-proving systems into three categories: machine-oriented theorem provers, proof checkers, and human-oriented (plan-based) theorem provers.

Normally all these systems do not exist in a pure form anymore, and in some systems like our own ΩMEGA system [BCF+97] 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. We think that the combination of these complementary approaches inherits more advantages than drawbacks, because for most tasks domain-specific as well as domain-independent problem-solving know-how is required and for difficult tasks more often than not an explicit user-interaction should be provided. While such an approach seems to be general enough to cope with any kinds of logic-level proofs, it neglects the fact that for many mathematical fields, the everyday work of mathematicians only partially consists in proving or verifying theorems. Calculation plays an equally important rôle. In some cases the tasks of proving theorems and calculating simplifications of certain terms can be separated from each other, but very often the tasks are interwoven and inseparable. In such cases an interactive theorem proving environment will only provide rather poor support to a user. Although theoretically any computation can be reduced to theorem proving, this is not practical for non-trivial cases, since the search spaces 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 procedures do not (and cannot) provide the control for doing a concrete computation.

1.2. Computer Algebra Systems

Early computer algebra systems (CAS for short) developed from collections of algorithms and data structures for the manipulation of algebraic expressions like the multiplication of polynomials, or the derivation and integration of functions [Hea95]. Abstractly spoken, the main objective of a CAS can be viewed in the simplification of an algebraic expression or the determination of a normal form. Today there is a broad range of such systems, ranging from very generally applicable systems to a multitude of systems designed for specific applications. Unlike MRS, CAS are used by many mathematicians as a tool in their everyday work, they are even widely applied in sciences, engineering and economics. Their high academic and practical standard reflects the fact that the study of symbolic calculation has long been an established and fruitful subfield of mathematics that has developed the mathematical theory and tools.

Most modern systems [Wol96, CGG$^+$92, JS92] have in common that the algebraic algorithms are integrated in a very comfortable graphical user interface that includes formula editing, visualisation of mathematical objects and even an interface to programming languages. As in the case of MRS the representation languages of CAS differ from system to system, which complicates the integration of such systems as well as the cooperation between them. This deficiency has been attacked in the OpenMath initiative [AvLS96], which strives for a standard CAS communication protocol. Currently the main emphasis is laid on standardising the syntax and the computational behaviour of the mathematical objects, while their properties or semantics are not considered. That means there is no explicit representation format for theorems, lemmata and proofs. Some specific systems allow to specify mathematical domains and theories. For instance in systems like MuPAD [Fuc96] or AXIOM [JS92], computational behaviour can be specified by attaching types and axiomatisations to mathematical objects; but this also falls short of a comprehensive representation of all relevant mathematics. Furthermore, almost all CAS fail to give an explanation or proof of their solution to the problem at hand, even though some mathematical theories like that of Gröbner bases can be successfully applied to theorem proving in elementary geometry [Cho88, Kap88, CGZ94, Wu94].

1.3. Contributions of this Paper

Not only the fact that a mutual simulation of the tasks of an MRS and a CAS can be quite inefficient, but more that the daily work of mathematicians is about proving *and* calculating points to the integration of such systems, since mathematicians want to have support in both of their main activities. Indeed two independent systems can hardly cover their needs, since in many cases the tasks of proving and calculating are hardly separable. As pointed out by Buchberger [Buc96a] 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.

Our paper addresses two immediate questions occurring in the integration of automated reasoning and computation systems.

- How can the algorithms be integrated, so that the underlying mathematical knowledge is mutually respected and a synergy effect is achieved?
- How can the correctness problem inherent in any such combination be addressed? In particular, how can results from the CAS be integrated into a proof without having to completely trust the CAS?

We advocate an integration of computer algebra into mechanised reasoning systems using the proof planning paradigm. This approach allows to encapsulate the computer algebra algorithms into *methods*, that is, declarative representations of the problem solving knowledge specific to a certain mathematical domain. The proof planning paradigm enables a user to guide a proof or to fully hand over the control to a planner, which in turn can use computer algebra systems, if the specifications for the corresponding algorithms are met. The use of hierarchic *proof plans* at the method-level gives a suitable granularity of integration, since it allows to directly use existing (human) control knowledge about the interplay of computation and reasoning.

A proper integration into the proof planning approach answers the question about the correctness automatically, since the corresponding questions are solved for proof planning. In this area a proof plan can either be rejected (the tactics are not executable, hence the plan cannot be used to build a proof) or executed. The later results either in a further planning phase to fill in possible gaps or in an accepted machine-checkable proof. Hence a proper integration requires that the computer algebra system produces high-level protocol information that can be processed by an interface to derive proof plans which themselves can be seamlessly integrated into the overall proof plan generated in the problem solving attempt. Since this can be expanded into an explicit, checkable proof in order to obtain a correctness guarantee for the combined solution, we have also given a principled answer to the correctness problem.

The feasibility of the approach advocated in the sequel has been verified by integrating a simple CAS into the ΩMEGA proof planning system. Therefore, we organise the paper around this experiment and describe the relevant features with a system perspective. Our approach requires a mode of the CAS that generates information from which it is possible to generate a proof plan. For that reason the integration of a standard CAS makes major adaptations unavoidable (in particular it is necessary to change the source code of these systems). Our approach is not committed to the particular systems involved, in particular, the work reported here should be understood rather as a proof of principle than as the development of a state-of-the-art integrated system.

Moreover, we will make the details of the approach more concrete by explaining them by means of an example that cannot easily be solved by either a mechanised reasoning system or a computer algebra system alone, but that needs the combined efforts of systems of each kind.

2. Related Work

We give a short description of some of the experiments to combine MRS and CAS and roughly categorise them into three classes with respect to the treatment of proofs that is adopted, that is, with respect to the correctness issue. In doing so we only describe in detail the approaches of integrating CAS into MRS, that is, essentially the MRS is the master and the CAS the slave, since our approach is also of this kind. With the same right, one can of course follow the converse direction, namely to approach the integration from the point of the CAS and indeed such approaches are also successfully undertaken (see e.g. [CZ92, Buc96]).

The question about the granularity of integration is treated uniformly by all these experiments. The application of the CAS is treated as another (derived) rule of inference at the level of the (tactic) calculus, so the granularity of integration depends on the granularity of the calculus or the tactics involved.

In the first category of attempts (see e.g. [HT93b, BHC95]) one essentially trusts that the CAS properly work, hence their results are directly incorporated into the proof. All these experiments are at least partly motivated by achieving a broader applicability range of formal methods and this objective is definitively achieved, since 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. This is not only a minor technical problem, but will remain unsolved for the foreseeable future since the complexity (not only the code complexity, but also the mathematical complexity) of a CAS does not permit a verification of the program itself with currently available program verification methods. Conceptually, the main contribution of such an integration is the solution of the software-engineering problem how to pass the control between the programs and translating results forth and back. While this is an important subproblem, it does not seem to cover the full complexity of the interaction of reasoning and computation found in mathematical theorem proving. In an alternative approach that formally respects correctness, but essentially trusts CAS, an additional assumption standing for the CAS is introduced, so that essentially formulae are derived that are proved modulo the correctness of the computer algebra system at hand (see e.g. [HT93b]).

The second category (for which [HT93a] is paradigmatic) is more conscious about the rôle 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. Indeed from the point of applicability, the results of the CAS help only in cases, where the verification of a result is simpler than its discovery, such as prime factorisations, solving equations, or symbolic integration. For other calculations, such as symbolic addition or multiplication of polynomials and differentiation, the verification is just as complex as the calculation itself, so that employing the CAS does not speed up the proof construction process. Typically in longer calculations, both types of sub-calculations are contained.

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] and been realised in NUPRL [Con86]. In this approach a constructive mechanised reasoning system is basically used as its own meta-system, the constructive features are exploited to synthesise a correct computer algebra system and due to bridge rules between ground and meta-system it is possible to integrate the so-built CAS that it can be directly used as a component. The theoretical properties of the meta-theoretic extension guarantee that if the original system was correct then the extended system is correct too. This method is the most appealing one from the viewpoint of correctness, although the assumption that the original (also rather complex) system must be correct can hardly be expected to be self-evident for any non-trivial system. A disadvantage compared to the other two approaches is that it is not possible to employ an existing CAS, but that it is necessary to (re)implement one in the strictly formal system given by the basic MRS. Of course this is subject to the limitations posed by the (mathematical and software engineering) complexities mentioned above.

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. In most cases, this is dealt with by meta-theoretic considerations why the algorithms are adequate. This lack of explanation makes it not only impossible for the average user to understand or convince himself of the correctness of the computation, but leaves any MRS essentially without any information why two terms should be equal. This is problematic, since computational errors have been reported even for well-tested and well-established CAS. From the reported categories of approaches only the last one seriously addresses this problem.

3. ΩMEGA as an Open System for Integrating Computation

ΩMEGA is a proof development system, based on the proof planning paradigm. In this section we describe its architecture and components and show how this supports the integration of computer algebra systems. Since the goal of this paper is not to present a system description of ΩMEGA, but to document the integration of computer algebra into it, we try to be as concise as possible and introduce the relevant parts only, the general architecture, the proof planner, and the integration possibilities for external reasoners.

3.1. The Proof Development Environment ΩMEGA

The entire process of theorem proving in ΩMEGA can be viewed as an interleaving process of proof planning, execution and verification centred around a hierarchical proof plan data structure.

Several integrated tools support the user in interacting with the system. Some of them are also available to the proof planner.

Theory Database

Since methods and control knowledge used in proof planning are mostly domain-specific, ΩMEGA organises the mathematical knowledge in a hierarchy of theories. Theories represent signature extensions, axioms, definitions, and methods that make up typical established mathematical domains. Each theorem has its home theory and therefore has access to the theory's signature extensions, axioms, definitions, and lemmata without explicitly introducing them. A simple inheritance mechanism allows to incrementally build larger theories from smaller parts.

We give an overview of the part of ΩMEGA's theory database that is necessary for solving our extended example in Figure 1.

Proof Explanation

Proof presentation is one important feature of a mathematical assistant that has been neglected by traditional deduction systems. ΩMEGA employs an extension of the PROVERB system [HF96] developed by our group that allows for presenting proofs and proof plans in natural language. In order to produce coherent texts that resemble those found in mathematical textbooks, PROVERB employs state-of-the-art techniques of natural language processing.

Due to the possibly hierarchical nature of ΩMEGA proofs, these can be verbalised at more than one level of abstraction, which can be selected by the user.

To summarise our view of proofs, for every theorem an explicit proof has to be constructed so that on the one hand it can be checked by a proof checker, on the other hand the system provides support to represent this proof in a high-level form that is easily readable by humans [HF96]. Neither the process of generating proofs nor that of checking them is fully replaced by the machine but only supported. If a human mathematician wants to see a proof, he/she can do so at an appropriate level of abstraction.

3.2. Proof Planning

The central data structure for the overall process is the *Proof plan Data Structure* ($\mathcal{PDS}$). This is a hierarchical data structure that represents a (partial) proof at different levels of abstraction (called *proof plans*). It is represented as a directed acyclic graph, where the nodes are justified by (LCF-style) tactic applications. Conceptually, each such justification represents a proof plan (the *expansion* of the justification) at a lower level of abstraction that is computed when the tactic is executed[1]. In ΩMEGA, we explicitly keep the original proof plan in an expansion hierarchy. Thus the $\mathcal{PDS}$ makes the hierarchical structure of proof plans explicit and retains it for further applications such as proof explanation or analogical transfer of plans.

Once a proof plan is completed, its justifications can successively be expanded to verify the well-formedness of the corresponding $\mathcal{PDS}$. This verification phase is necessary, since the correctness of the different components (in particular, that of external ones like automated theorem provers or computer algebra systems) cannot be guaranteed. When the expansion process is carried out down to the underlying ND-calculus (natural deduction), the soundness of the overall system relies solely on the correctness of the verifier and of ND. This also provides a basis for the controlled integration of external reasoning components if each reasoner's results can (on demand) be transformed into a sub-$\mathcal{PDS}$. The level to which the proofs have to be expanded depends on the sophistication of the proof checker. As pointed out by Barendregt [Bar96], a more complex proof-checker that accepts proofs in a more expressive formalism may drastically reduce the length of the communicated proofs. If the high-level justifications are not expanded but accepted as they are, our approach reduces to one in which the computer algebra system is fully trusted. In short, the hierarchical nature of the $\mathcal{PDS}$ supports the full spectrum of user preferences, from total trust in the CAS, over partial trust in certain levels to full expansion of the proofs in a detailed calculus level description that is machine checkable.

A $\mathcal{PDS}$ can be constructed by automated or mixed-initiative planning, or pure user interaction that can make use of the integrated tools. In particular, new pieces of $\mathcal{PDS}$ can be added by directly calling tactics, by inserting facts from a database, or by calling some external reasoner (cf. Section 3.3) such as an automated theorem prover or a computer algebra system. Automated proof planning is only adequate for

[1] This proof plan can be recursively expanded, until we have reached a proof plan that is in fact a fully explicit proof, since all nodes are justified by the inference rules of a higher-order variant of Gentzen's calculus of natural deduction (ND).

problem classes for which methods and control knowledge have already been established.

The goal of proof planning is to fill gaps in a given $\mathcal{PDS}$ by forward and backward reasoning [HKKR94] (proof plans were first introduced by Bundy, see [Bun88, BSvH$^+$93]). Thus from an abstract point of view the planning process is the process of exploring the search space of *planning states* that is generated by the *plan operators* in order to find a complete *plan* from a given *initial state* to a *terminal state*.

ΩMEGA's proof planner is an extension of the well-known STRIPS algorithm that can be evoked to construct a proof plan for a node g (the *goal node*) from a set I of *supporting nodes* (the initial state) using a set *Ops* of proof planning operators, here called methods. A *method* is a (partial) specification of a tactic in a meta-level language. In ΩMEGA planning is combined with hierarchical expansion of methods and precondition abstraction. The plans found by this procedure are directly incorporated into the $\mathcal{PDS}$ as a separate level of abstraction.

In this model, the actual reasoning competence of the planner and the user builds upon the availability of appropriate methods together with meta-level control knowledge that guides the planning. At the moment, ΩMEGA provides user-defined method ratings as a means of control and can use analogy as a control strategy of the planner. Two examples of methods are displayed in the section on the extended example, Section 3.4.

3.3. Integration of Computer Algebra Systems as External Reasoners

According to the different modes of ΩMEGA there are different levels on which an external reasoning system, RSys, can be integrated:

- **Interactive calls**, RSys is represented as a command `call-RSys` that invokes the reasoner on a particular subproblem and returns the result,
- **Proof planning**, RSys is represented as a method whose specification contains knowledge about the problem solving behaviour and option settings for RSys.
- **Justifications**, RSys can serve as a justification of a declaratively given subgoal that is left to be proved by RSys.

In any case, the proof found by RSys must eventually be transformed into a $\mathcal{PDS}$, since this is the proof-theoretic basis of ΩMEGA. For automated theorem provers like OTTER [McC94], we described the integration in [HKK$^+$94] and the necessary proof transformation to $\mathcal{PDS}$ in [HF96], so we will not pursue this matter here. The integration of

CAS follows the same paradigm and is the main topic of this paper, so we will develop the paradigm for the case of external computations in ΩMEGA. We will see examples for the three different levels of integrations of a CAS into ΩMEGA in the example in the next section, so we will not go into that here. This leaves us with the question of the transformation of the CAS results into $\mathcal{PDS}$.

If we take the idea of generating explicit $\mathcal{PDS}$ seriously also for computations we can neither just take existing systems nor follow the approach of meta-theoretic extensions, since ΩMEGA is a classical proof system and does not use constructive logic. On the other hand we cannot forgo using them even in cases, where the verification of a calculation is much easier than the calculation itself (e.g., integration of functions); the computation needed for verifying alone is in many cases still much too complicated to be automatically checked without any guidance. 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 then 70 single steps in the natural deduction calculus[2]. 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 the 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. Surely in most cases a user would not like to see proofs at a level where the binomial formula is explained (although a novice might want to). This means that a hierarchical approach to proof generation is appropriate, in which the abstraction level of the proof presentation can be chosen by the user.

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

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

produce a readable explanation or further expanded to a level that can be automatically checked by proof checkers. The level of abstraction on which the checking can take place, depends on the level of sophistication of the proof checker. For a naive proof checker, the proof must be expanded to an explicit calculus level. 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 mode of the CAS that gives enough information to generate a high-level description of the computation in terms of the mathematics involved. This is is described in the next section in detail. Before doing so we describe how the integrated system automatically solves an extended example from an economics examination.

3.4. Extended Example

The concrete task at hand is to minimise the costs for running a machine while producing a certain product.

Problem: *The output of a machine can range over a certain interval, the interval $I = [1,7]$. The cost of the product prod is determined by the costs of water and electricity for producing prod, which are given by the functions*

- $r_1 = (0.5d^2 + 3)\,\frac{m^3}{prod}$
- $r_2 = (4d^2 - 24d + 6)\,\frac{kWh}{prod}$

and the prices for water and electricity

- $p_1 = 2\,\frac{DM}{m^3}$
- $p_2 = 0.5\,\frac{DM}{kWh}$

Determine the output d in I of the machine such that the total costs are minimal. ■

This example serves our purposes for several reasons. Firstly, it allows us to show the interaction of proof planning with symbolic computation and the extraction of proof plans from calculations. Secondly, the mathematics involved is simple enough to be fully explained (only simple polynomial manipulations are necessary). Thirdly, it is not an example we created, but the problem is a slightly varied version of a minimisation problem from a masters examination in economics at the Universität des Saarlandes, Saarbrücken [WiW89].

In order to solve problems like this, we have integrated a simple CAS into ΩMEGA, called μ-CAS[3].

The μ-CAS-system is very simple and can at the moment only perform basic polynomial manipulations and differentiation, but it suffices for automatically solving the example at hand. Clearly, for a practical system for mathematical reasoning, a much more developed system like Maple [CGG+92], Reduce [Hea95], AXIOM [JS92], or Mathematica [Wol96] has to be integrated. The technicalities of the integration will be described in Section 4.

For the formalisation of the example, we use the theory mechanism of ΩMEGA to create a theory `economy` (see Figure 1) that contains the domain-specific knowledge (both the factual and the method knowledge) needed for the problem solution. Obviously, we need a background theory of *costs* in economics (that handles both numerical parts and denomination of cost functions) and one of *minimisation* of real functions, therefore, our theory inherits material from the theories `costs` and `calculus`. The `calculus` theory is provided by ΩMEGA and contains relevant parts of the knowledge of an elementary calculus textbook: For instance, the *real numbers* are introduced as a complete, dense archimedian *field* (based on elementary algebraic notions such as *groups* and *rings* defined in the respective theories). The set of real numbers (showing the existence of such a complete, dense archimedian field) are constructed as the quotient field of the ring of sequences of *rational* numbers over the ideal of null-sequences. The rational numbers in turn are constructed as signed fractions of natural numbers that are defined from the Peano axioms in theory `natural`. All of these mathematical theories are based on the theories `function`, `set`, and `relation`, that specify naive (simply typed) set theory and the properties of functions and relations on such sets. Finally, the whole hierarchy builds on the theory base, which declares the underlying logic by providing the logical connectives and quantifiers and the basic ND inference rules.

The theory `economy` provides a type v of units that covers the different units of denominations – in our example m^3 (for volume), kWh (for work), *prod* (for product) and *DM* (for the price). We then formalise prices as triples consisting of one real number and two units and cost functions as a real function together with two units (read as input/output units). Note, that just as in the real world, addition ($\oplus$)

[3] The μ-CAS system is part of the standard distribution of ΩMEGA which can be obtained from `http://www.ags.uni-sb.de/software/deduktion/omega`. The example is accessible as `WiWi-Exam` in the theory `economy`.

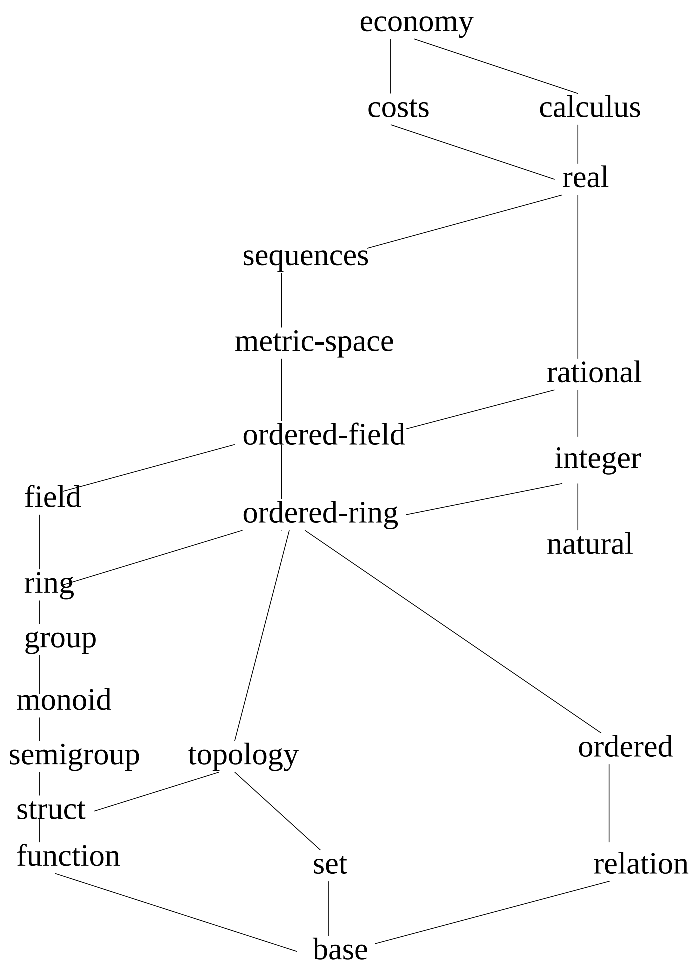

Figure 1. Theory hierarchy in ΩMEGA's knowledge base

multiplication ($\otimes$) and comparison of costs and cost functions is defined as that of their real parts with respect to the denominations. For these calculations we have the axioms CF1 and CF2. If two denominations differ, we can relate them by their prices, for this purpose we use axiom Pr.

$$
\begin{array}{lrcl}
\text{CF1} & cf(f,u,v) \oplus cf(g,u,v) & = & cf(f+g,u,v) \\
\text{CF2} & cf(f,u,v) \otimes cf(g,v,w) & = & cf(f \cdot g,u,w) \\
\text{Pr} & \mathit{price}(f,u,v) \Rightarrow cf(g,v,w) & = & cf(f \cdot g,u,w)
\end{array}
$$

Optimisation in `economy` is formalised by a predicate *Opt* on a cost function $cf(f, DM, prod)$ and an interval I that is true, whenever f has a total minimum[4] on I.

$$\text{O} \quad Opt(cf(f, DM, prod), I) \Leftrightarrow \exists x.\, \text{TotMin}(x, f, I)$$

Thus we can state the problem as the following formula[5]

$$\text{THM} \quad \mathcal{H} \vdash Opt([cf(\lambda d.\, 0.5d^2 + 3, m^3, prod) \oplus cf(\lambda d.\, 4d^2 - 24d + 6, kWh, prod)], [1, 7])$$

where $\mathcal{H}$ is a set of hypotheses that are needed for the complete proof, for instance the price axioms

$$P_{m^3} \quad price(2, DM, m^3)$$
$$P_{kWh} \quad price(0.5, DM, kWh)$$

The planner solves the problem by generating a high-level proof plan consisting of methods from its domain specific method base on economics exam questions[6].

We are going to outline this process by describing its major steps. In particular, we will demonstrate how the proof planner of ΩMEGA and the μ-CAS-system interact, and make explicit, on which entries of a mathematical database this interaction depends. The planner finds the following simple proof plan:

---

[4] The predicate TotMin and the problem solving knowledge related to it is inherited from the theory `calculus`.

[5] Actually the formalisation of the problem is not fully correct, since the examiner is not only interested in the proof that there exists such an x, but he/she wants to know the value of x as well as a proof that this value fits the requirements. Obviously, such an answer cannot be obtained from the formula here, but only from a proof that is constructive for the variable x, where we can extract a witness term. This is no problem for a CAS nor for an MRS based on constructive logic, but for a traditional MRS based on classical logic, the proof construction process has to be refined to guarantee constructivity for x. Note that the arguments, why the witness for x meets the requirements can still be classical and non-constructive. For ΩMEGA this means that the proof planner may only use methods in our proof plan that are constructive to get the wanted answer as presented here and not a non-constructive abstract argument. Finally note that this phenomenon is another argument in favour manipulating explicit proofs. Without this, one may find oneself in the position, that one is convinced (by meta-theoretic arguments) of the existence of a (constructive) proof, but in fact without one from which to extract a term witness to answer the exam question.

[6] Questions for certain standard exams are a good example for a very restricted mathematical domain, since the proofs and calculations involved are highly standardised. Therefore finding the proof plan in this example is not a big problem for ΩMEGA.

1 Mult-by-Price
2 Mult-by-Price
3 Add-by-Denom
4 Optimise
5 TotMin-Rolle

where the first three methods compute the actual cost function by adjusting the denominations and adding. Method 4 uses Axiom O for optimisation. As the example only contains polynomials of degree two, the planner selects a method `TotMin-Rolle` (cf. Figure 3) for finding total minima that makes implicit use of Rolle's theorem from the `calculus` theory:

> *Let f be a polynomial of degree two, then f has a total minimum at $x \in [a,b]$, iff f has a minimum at x and $f(a) \geq f(x) \leq f(b)$.*

Formally we get the following equivalence:

$$\text{TotMin} \quad \text{TotMin}(x, f, [a,b]) \Leftrightarrow x \in [a,b] \wedge \text{Min}(x,f) \wedge \\ f(x) \leq f(a) \wedge f(x) \leq f(b)$$

Note that Rolle's theorem is accessible in the current theory and, to ensure correctness, the database has to contain its formal proof.

Now let us take a closer look at some of the methods in order to get a feeling of how this initial proof plan can be expanded. In Figures 2 and 3 we have given slightly simplified presentations of the `Mult-by-Price` and `TotMin-Rolle` method[7].

The declaration slot of the method simply defines the meta-variables used in the body of the method. The premises, conclusions, and the constraint describe the applicability of the method. In the example of `Mult-by-Price`, for instance, line L_4 has to be present and to be an open subgoal, while L_1 and L_3 are lines that can be used in order to infer L_4. L_1 has to be given already, whereas L_3 is generated by the application of the method (indicated by the $\oplus$). Since the method is intended to prove L_4, after the application of the method, this line can be deleted from the current planning state (we indicate this by the $\ominus$). In the constraint slot further applicability criteria are described, which cannot be formulated in terms of proof line schemata. Declarations, premises, constraints, and conclusions form the specification part of the method. In order to be able to mechanically adapt methods the tactic part is further subdivided into the declarative content and the procedural content. (However, this particular feature is not important for the purpose of this paper.) In our examples the procedural content

[7] We have especially adjusted the syntax of the constraint in a way that is more comprehensive for the reader.

Method : `Mult-by-Price`	
Declarations	L_1, L_2, L_3, L_4: prln H_1, H_2, H_3: list(prln) J_1: just $f, g, v, w, \phi, \phi', \psi, \psi'$: variable
Premises	$L_1, \oplus L_3$
Constraint	$\psi \leftarrow (\mathsf{2ndarg}(\mathsf{termocc}(cf, \phi)) \neq DM \rightarrow \mathsf{termocc}(cf, \phi))$ $g \leftarrow \mathsf{1starg}(\psi)$ $v \leftarrow \mathsf{2ndarg}(\psi)$ $w \leftarrow \mathsf{3rdarg}(\psi)$ $\psi' \leftarrow cf(g \cdot f, DM, w)$ $\phi' \leftarrow \mathsf{replace}(\psi', \psi, \phi)$
Conclusions	$\ominus L_4$
Declarative Content	$(L_1)\ H_2 \quad \vdash price(f, DM, v) \quad (J_1)$ $(L_2)\ H_1, H_2 \quad \vdash cf(g, v, w) = \psi' \quad (\mathrm{Pr}\ L_1)$ $(L_3)\ H_3 \quad \vdash \phi' \quad$ (Call-CAS) $(L_4)\ H_1, H_2, H_3 \vdash \phi \quad (=\mathrm{subst}\ L_3\ L_2)$
Procedural Content	`schema − interpreter`

Figure 2. The `Mult-by-Price` method from theory `cost`.

consists of a `schema-interpreter`, which essentially inserts the declarative content (using the bindings made in the planning phase) at the correct place in the current partial proof tree. In the concrete example the lines L_1 through L_4 are inserted (Note that we adopted a linearised version of ND proofs as introduced in [And80]).

In order to understand to which piece of actual proof these methods evaluate, we have to examine the declarative content and the bindings performed in particular in the constraint. The constraint of the `Mult-by-Price`-method states a rather simple computation: if there is a cost function in the given open line which has a denomination other than DM, it is multiplied with the appropriate price. The multiplication of the real parts is carried out by the CAS and the corresponding cost function is constructed. As this point is crucial for understanding the working scheme of a method we will view the bindings in the constraint step by step: When applied to the current plan the method is matched with the open goals of the planning state. The first pass of the planner yields that L_4 can be matched with our theorem THM. Thus its formula $Opt([cf(\lambda d. 0.5d^2+3, m^3, prod) \oplus cf(\lambda d. 4d^2-24d+6, kWh, prod)], [1,7])$ is bound to the meta-variable ϕ. It is then examined to find an occurrence of a cost function. If such a subterm exists its arguments are bound to g, v, w and by matching line L_1 we receive the numerical part of *price* in f (if the appropriate price is not provided the application of the method would fail here). Afterwards the new cost function is computed (according to axiom Pr) in ψ' and finally ϕ' contains the result of replacing the old cost function in ϕ by ψ'. Hence in the first plan step the optimisation formula stored in ϕ' contains the cost function $cf(\lambda d. 1d^2+6, DM, prod)$ as a subterm.

With all these meta-variables instantiated the subproof contributed by the `Mult-by-Price`-method consists of lines L_2 and L_3 in the declarative content. Here we observe that L_2 results from applying the price-axiom Pr (which is fetched from the database) to line L_1. Furthermore note that in L_3 we have a call to the CAS as a justifying method for the line. This means that at this point in the proof planning procedure, the CAS is called in order to compute the product of price and original cost function. The line resulting from this calculation is then used as the new open subgoal in the planning state.

Summarising the effects of the method `Mult-by-Price` can be observed in two steps. First the goal line THM is justified with the method yielding the following subproof:

$$\begin{array}{lll}
\mathrm{L}_1 & \mathcal{H} \vdash Opt([cf(\lambda d. 1d^2+6, DM, prod) \oplus & (\mathrm{Open}) \\
 & \qquad cf(\lambda d. 4d^2-24d+6, kWh, prod)], [1,7]) & \\
\mathrm{THM} & \mathcal{H} \vdash Opt([cf(\lambda d. 0.5d^2+3, m^3, prod) \oplus & (\mathrm{MbP}\ L_1) \\
 & \qquad cf(\lambda d. 4d^2-24d+6, kWh, prod)], [1,7]) &
\end{array}$$

Then the method in the justification of line THM (which has been abbreviated due to a lack of space) could be expanded thereby inserting the intermediate steps as described above by instantiating the macro steps of the method. Note that the following expanded subproof is at a more detailed level of abstraction in the $\mathcal{PDS}$. In particular, the justification of THM itself is different at this level.

$$\begin{array}{llll}
\mathrm{P}_{m^3} & \mathrm{P}_{m^3} & \vdash price(2, DM, m^3) & (\mathrm{HYP}) \\
\mathrm{L}_2 & \mathcal{H} & \vdash cf(\lambda d. 0.5d^2+3, m^3, prod) = & (\mathrm{Pr}\ \mathrm{P}_{m^3}) \\
 & & \quad cf(\lambda d. 1d^2+6, DM, prod) & \\
\mathrm{L}_1 & \mathcal{H} & \vdash Opt([cf(\lambda d. 1d^2+6, DM, prod) \oplus & (\mathrm{Open}) \\
 & & \qquad cf(\lambda d. 4d^2-24d+6, kWh, prod)], [1,7]) & \\
\mathrm{THM} & \mathcal{H} & \vdash Opt([cf(\lambda d. 0.5d^2+3, m^3, prod) \oplus & (=\mathrm{subst} L_1 L_2) \\
 & & \qquad cf(\lambda d. 4d^2-24d+6, kWh, prod)], [1,7]) &
\end{array}$$

In the proof of THM, the method `Mult-by-Price` is applied twice in order to normalise both summands. To preserve space we will not present the next two methods of our proof plan as extensively as the `Mult-by-Price`-method. `Add-by-Denom` is very similar to `Mult-by-Price` and applies axiom CF1 inside the optimisation function *Opt* to compute the final cost function. In its course the CAS is called once to perform a polynomial addition. Then the `Optimise`-method simply introduces the definition for the *Opt* function of axiom O.

Method : `TotMin-Rolle`	
Declarations	$L_1, L_2, L_3, L_4, L_5, L_6, L_7, L_8, L_9, L_{10}, L_{11}$: prln H_1, H_2, H_3: list(prln) J_1, J_2: just a, b, f, x: variable y, ϕ, α, β: term
Premises	L_1, L_2
Constraint	$\mathsf{degree}(\phi) \doteq 2$ $y \leftarrow \mathsf{compute\_with\_CAS}(minimum, \phi)$
Conclusions	$\ominus L_{12}$
Declarative Content	(L_1) $H_1 \vdash \forall f. \forall x. (f'(x) = 0 \wedge f''(x) > 0) \Rightarrow \mathrm{Min}(x, f)$ (J_1) (L_2) $H_2 \vdash \forall a. \forall b. \forall x. x \in [a, b] \Leftrightarrow (a \leq x \wedge x \leq b)$ (J_2) (L_3) $H_3 \vdash \phi'(y) = 0$ (Call-CAS) (L_4) $H_3 \vdash \phi''(y) > 0$ (Call-CAS) (L_5) $H_3 \vdash \alpha \leq y$ (Simplify) (L_6) $H_3 \vdash y \leq \beta$ (Simplify) (L_7) $H_3 \vdash \phi(y) \leq \phi(\alpha)$ (Simplify) (L_8) $H_3 \vdash \phi(y) \leq \phi(\beta)$ (Simplify) (L_9) $H_3 \vdash \mathrm{Min}(y, \phi)$ $(L_1\ L_3\ L_4)$ (L_{10}) $H_3 \vdash y \in [\alpha, \beta]$ $(L_2\ L_5\ L_6)$ (L_{11}) $H_3 \vdash \mathrm{TotMin}(y, \phi, [\alpha, \beta])$ $(\mathrm{TotMin}\, L_7 L_8 L_9 L_{10})$ (L_{12}) $H_3 \vdash \exists x. \mathrm{TotMin}(x, \phi, [\alpha, \beta])$ $(\exists I\ L_{11})$
Procedural Content	`schema` $-$ `interpreter`

Figure 3. The `TotMin-Rolle` method from theory `calculus`.

Far more interesting than these two methods is the `TotMin-Rolle`-method as it contains a different example for the use of a CAS in ΩMEGA. Again the presentation of the method in Figure 3 is simplified.

The `TotMin-Rolle` method is applied at a stage of the proof where the actual minimum of the cost function has to be introduced. This task is fulfilled within the constraint of the method. The compute_with_CAS statement actually calls the CAS in quiet mode to compute the minimum of the function ϕ and store it in the meta-variable y. At this stage, the CAS is used as an oracle here, just as in [HT93a]. In our example the minimum of the cost function is at $y = 2$ and the ND-line of the form

$$\exists x. \mathrm{TotMin}(x, \lambda x. (3x^2 + (-12x + 9)), [1, 7])$$

will be transformed by eliminating the existentially quantified variable:

$$\mathrm{TotMin}(2, \lambda x. (3x^2 + (-12x + 9)), [1, 7])$$

The rest of the proof plan is devoted to proving that the result is actually a total minimum. This is done by using the definition for TotMin from the database and furthermore by using the definitions for minimum and interval which correspond to line L_1 and L_2 in the method `TotMin-Rolle`. These definitions are introduced in lines L_9 through L_{11} by applying them to the correct assertions given in lines L_3 through L_8. This is expressed by the justifications in the corresponding lines; for instance, the justification of line L_{10} states that we can infer $y \in [\alpha, \beta]$ from the lines L_5 and L_6 with the definition of interval in line L_2.

A closer look at the justifications of lines L_3 through L_8 reveals that these contain methods themselves. Lines L_3 and L_4 again depend on calculations of the CAS which computes the first and second derivative of our cost function. The justifications Simplify correspond to a method performing basic arithmetic simplifications and comparisons.

Consisting of only 5 methods the above proof plan gives the impression of a small proof and on an abstract level it is indeed; an experienced mathematician might not want to see more. But expanding the plan into a partially grounded ND proof gives it a length of 90 lines, containing lines justified by the CAS. The proof on this level may roughly correspond to a proof that a novice would like to see and that would form a reasonable solution of the exam problem once it is presented in natural language by the PROVERB system. By rerunning the CAS in a proof plan generating mode on the CAS-justifications and extracting proof plans, the proof can be expanded to a more detailed proof plan containing an account of the mathematics behind the calculations. This proof plan already contains 135 plan steps and – if the user does not feel comfortable with the level of detail yet – can then be expanded to a calculus-level ND proof of length 354. Note that even this proof is not a stand-alone proof of the minimisation theorem, but depends on the proofs of a number of lemmata from a database. Furthermore, in these proofs the simplification of ground arithmetic expressions is not expanded, for instance, into a representation involving zero and the successor function either, which would be necessary to obtain a detailed logic-level proof.

4. Integrating Computations into Explicit Proofs

In this section we describe SAPPER (**S**ystem for **A**lgorithmic **P**roof **P**lan **E**xtraction and **R**easoning), which generates proof plans from CAS output. As mentioned in Section 3.3, for the intended integration it is necessary to augment the CAS with mathematical information for a *proof plan generating mode* in order to achieve the proposed integration at the level of proofs. For the μ-CAS system, which we have developed to demonstrate the feasibility of the approach, this was rather simple, as we will demonstrate below. Enriching a state of the art CAS with such a mode for producing the necessary additional protocol information, would of course require a considerable amount of work.

4.1. Architecture

The SAPPER system can be seen as a generic interface for connecting ΩMEGA (or another proof plan-based mechanised reasoning system) with one or several computer algebra systems (see Figure 4). An incorporated CAS is treated as a slave to ΩMEGA which means that only the latter can call the first one and not vice versa. From the software engineering point of view, ΩMEGA and the CAS are two independent processes while the interface is a process providing a bridge for communication. Its rôle is to automate the broadcasting of messages by transforming output of one system into data that can be processed by the other[8].

Unlike other approaches (see [HC95, GPT96] for example) we do not want to change the logic inside our MRS. 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 ΩMEGA.

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 ΩMEGA and a CAS in both directions while the latter

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

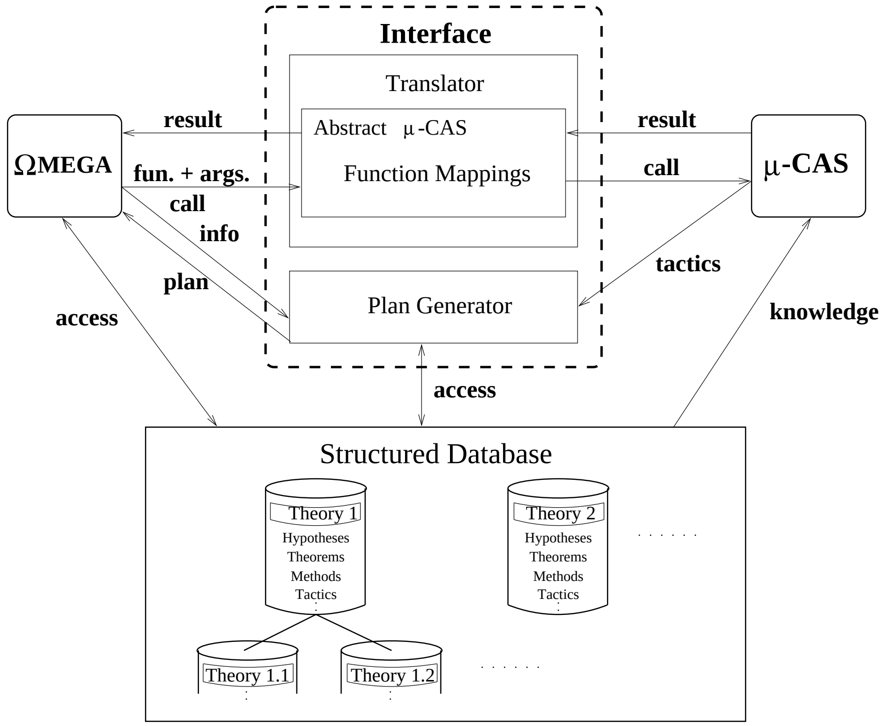

Figure 4. Interface between ΩMEGA and computer algebra systems

only transforms verbose output of the CAS to ΩMEGA 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*[9] – 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 ΩMEGA to a corresponding CAS-function and the type of its arguments. Furthermore it provides functionality to convert the given arguments of the mapped ΩMEGA function to CAS input. In the same fashion it transforms results of algebraic computations back into data that can be further processed by ΩMEGA. The functionality in this part of our interface offers us the possibility of connecting any CAS as a black box system, as in the first approach we have described in Section 2. For instance, we may want to use a very efficient system without a mode for generating proof plans in

[9] In a reimplementation of SAPPER we would probably use the OpenMath protocol [AvLS96] as a lingua franca on the CAS side.

proof search as black box system, and then another less efficient system with such a mode for the actual proof construction, once it is clear what the proof should look like. This corresponds to recent techniques used in knowledge based systems, where the explanation component is not just a trace of the rules applied during the search, but the explanation is reconstructed by an independent component.

The plan generator solely provides the machinery for our main goal, the proof plan extraction. Equipped with supplementary information on the proof by ΩMEGA 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 database of mathematical facts formalising the mathematics behind the particular CAS algorithms. Conceptually, this database together with the mappings governing the access, provides the semantics of the integration of ΩMEGA with a particular CAS. Thus expanding the plan generator is simply done by expanding the theory database by adding new tactics.

While ΩMEGA 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. Figure 4). The CAS does not interact with the database at all: it only has to know about it and references the logical objects (methods, tactics, theorems, or definitions) in the proof plan generating mode. Thus knowledge about the database is compiled a priori into the algebraic algorithms in order to document their calculations.

4.2. Proof Plan Extraction

Let us now take a closer look at the implementation of the proof plan generation in μ-CAS and at the expansion process of its output. This should demonstrate how proofs can be extracted from computer algebra calculation and provide an intuition on the requirements that our approach poses on the CAS side.

As an example we will consider a polynomial addition from the example above. Normally, an experienced mathematician would not like to see any proof at all for that, while a high-school student would like to. As we have seen in our example, the main purpose of the `Add-by-Demon`-method is to compute the final cost function $cf(\lambda d. (3d^2 - 12d + 9), DM, prod)$. This is done in μ-CAS by adding the two polynomials $\lambda d. d^2 + 6$ and $\lambda d. 2d^2 - 12d + 3$. In the remainder of this subsection we will expand this addition in several steps and thereby obtain a calculus level proof for the computation.

Before examining this example in detail, let us consider the general scheme of the proof plan generation inside the polynomial addition algorithm of μ-CAS. We first take a look at the different representations of a polynomial p in the variables $x_1, \ldots, x_r$: $p = \sum_{i=1}^{n} \alpha_i x_1^{e_{1_i}} \cdots x_r^{e_{r_i}}$. The logical language of ΩMEGA is a variant of the simply typed λ-calculus (indeed we use a stronger type system, but here we want to keep things as simple as possible), so the polynomials are represented as polynomial functions, that is, as λ-expression where the formal parameters $x_1, \ldots, x_r$ are λ-abstracted (mathematically, p is a function of r arguments):

$$p: \quad \lambda x_1 \cdots \lambda x_r.(+ (* \alpha_n (* (\uparrow x_1 \, e_{1_n}) \cdots)) \cdots (* \alpha_1 (* (\uparrow x_1 \, e_{1_1}) \cdots)),$$

For the notation, we use a prefix notation; the symbols $+$, $*$ and $\uparrow$ denote binary functions for addition, multiplication and exponentiation on the reals. In this representation, we can use β-reduction for the evaluation of polynomials.

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. Hence we get the following representation for p:

$$p: \quad ((\alpha_n \;\; e_{1_n} \cdots e_{r_n}) \cdots (\alpha_1 \;\; e_{1_1} \cdots e_{r_1}))$$

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[10] with recursive calls to itself. So let $p = \sum_{i=1}^{n} \alpha_i x_1^{e_{1_i}} \cdots x_r^{e_{r_i}}$ and $q = \sum_{i=1}^{m} \beta_i x_1^{f_{1_i}} \cdots x_r^{f_{r_i}}$. We have presented the algorithm in the jth component of p and the kth component of q in a LISP-like pseudo-code in Figure 5. 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.

Obviously, the only expansions of the original algorithm needed for the proof plan generation are the additional `(tactic...)` statements[11].

[10] We assume a lexicographic monomial order and employ it for ordering the exponents. Thus we make use of the operators $>$, $<$, and $=$ in an intuitive sense. Furthermore we can define the rank of a monomial as the vector given by its exponents and the rank of a polynomial as the maximum rank of its monomials with respect to the lexicographic monomial order.

[11] 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

$$
\begin{array}{l}
\texttt{(poly-add (p q)}\\
\quad (= (e_{1_j} \cdots e_{r_j})(f_{1_k} \cdots f_{r_k}))\\
\qquad \texttt{(tactic "mono-add")}\\
\qquad \texttt{(cons-poly}\ \ (\alpha_j + \beta_k) x_1^{e_{1_j}} \cdots x_r^{e_{r_j}}\\
\qquad\quad \texttt{(poly-add}\ \sum_{i=j+1}^{n} \alpha_i x_1^{e_{1_i}} \cdots x_r^{e_{r_i}}\ \ \sum_{i=k+1}^{m} \beta_i x_1^{f_{1_i}} \cdots x_r^{f_{r_i}}\texttt{))}\\
\quad (> (e_{1_j} \cdots e_{r_j})(f_{1_k} \cdots f_{r_k}))\\
\qquad \texttt{(tactic "pop-first")}\\
\qquad \texttt{(cons-poly}\ \ \alpha_j x_1^{e_{1_j}} \cdots x_r^{e_{r_j}}\\
\qquad\quad \texttt{(poly-add}\ \sum_{i=j+1}^{n} \alpha_i x_1^{e_{1_i}} \cdots x_r^{e_{r_i}}\ \ \sum_{i=k}^{m} \beta_i x_1^{f_{1_i}} \cdots x_r^{f_{r_i}}\texttt{))}\\
\quad (< (e_{1_j} \cdots e_{r_j})(f_{1_k} \cdots f_{r_k}))\\
\qquad \texttt{(tactic "pop-second")}\\
\qquad \texttt{(cons-poly}\ \ \beta_k x_1^{f_{1_k}} \cdots x_r^{f_{r_k}}\\
\qquad\quad \texttt{(poly-add}\ \sum_{i=j}^{n} \alpha_i x_1^{e_{1_i}} \cdots x_r^{e_{r_i}}\ \ \sum_{i=k+1}^{m} \beta_i x_1^{f_{1_i}} \cdots x_r^{f_{r_i}}\texttt{)))}
\end{array}
$$

Figure 5. Polynomial addition in μ-CAS.

They just produce the additional output by returning keywords of tactic names to the plan 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 now apply this algorithm to the two polynomials

$$p := x^2 + 6 \qquad\qquad q := 2x^2 - 12x + 3$$

we obtain the following proof plan:

```
(mono-add, pop-second, mono-add)
```

First the two quadratic monomials from p and q are added, then the linear term of q (the second argument) is raised, since it only appears in one argument, and finally the remaining monomials are added up.

In the case of the polynomial addition, each of the methods (proof plan operators) directly corresponds to a tactic with the same name, that is, the list of the three methods above directly represents a concrete proof plan for polynomial addition of the concrete polynomials p and

proof and thus knows on which monomials the algorithm works when returning a tactic. This way we need not to be concerned which form a monomial actually has during the course of the algorithm.

q. (In the following representation we omitted the context in which the polynomials are embedded in the actual proofs.)

$((x^2+6)+(2x^2-12x+3))$	
$(3x^2+(6+(-12x+3)))$	(`mono-add`)
$(3x^2-12x+(6+3))$	(`pop-second`)
$(3x^2-12x+9)$	(`mono-add`)

These four lines correspond to a step-by-step version of the basic High School algorithm. So far the expansion of the `call-cas`-method has been exclusively done by μ-CAS proof plan generation mode. But at this stage μ-CAS cannot provide us with any more details about the computation and the subsequent expansion of the next hierarchic level can be achieved without further use of a CAS.

Let us for instance 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:

$$(a+(b+c)) = (b+(a+c)) \tag{1}$$

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 equation (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. The `pop-second` step above can thus be expanded in a plan which reflects the single step applications of the laws of commutativity and associativity.

$(3x^2+(6+(-12x+3)))$	
$(3x^2+((6-12x)+3))$	(`associativity`)
$(3x^2+((-12x+6)+3))$	(`commutativity`)
$(3x^2-12x+(6+3))$	(`associativity`)

Assuming we have expanded the two `mono-add` tactics as well, we have constructed a representation of the proof at a level where it only needs the axioms in the polynomial ring. To finally expand this to a fully explicit calculus level proof, 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:

$\forall a \forall b. (a+b) = (b+a)$	(THM)
$\forall b. (6+b) = (b+6)$	($\forall$E 6)
$(6-12x) = (-12x+6)$	($\forall$E $-12x$)

Here, the justification (THM) in the first proof line indicates that the commutativity of + was imported from the theory `real` in ΩMEGAs mathematical database, where it was established as a theorem. The remaining lines are natural deduction inferences: universal eliminations that instantiate a with the number 6 and b with the term $-12x$.

Altogether this single application of the `pop-second`-tactic is equivalent to a calculus-level proof of 11 inference steps. The length of the subproof for this trivial polynomial addition is 43 single steps. This example shows how it is possible to mechanically construct a proof verifying the correctness of any particular CAS computation without verifying the CAS algorithm (or their implementation) in the general case.

However, the calculus level proofs for the computations are very long and rather boring and therefore hardly any human user might actually want to see much less read them. Therefore the PROVERB proof explanation system in ΩMEGA provides a more realistic alternative, since it gives the user access to representations of the parts of the proof on various levels of abstractions making use of the hierarchical structure of the underlying $\mathcal{PDS}$. For instance, it is then possible to present the computations with some intermediate steps, as it is customary in textbooks. For example, we could include the three steps of the High School algorithm mentioned above, to illustrate the polynomial addition. (The decision which steps should be included and which omitted, depends of course on the expertise of readers for which a particular proof presentation is intended.)

Despite all these abstractions in both developing and presenting the proof, we can still use any proof checker for ND-calculus to verify all steps including computations. Furthermore, if we assume we have a more sophisticated proof checker, for example one that works modulo the axioms of polynomial rings, it is also possible to check the proof on an abstract level. As already mentioned, the more sophisticated the proof checker is, the more concise the communicated proofs can be.

We have tested proof plan extraction from simple recursive and iterative CAS algorithms, where it works quite well, since these algorithms closely correspond to the mathematical definitions of the corresponding concepts. However, more complicated schemes like divide-and-conquer algorithms (for instance, the polynomial multiplication of Karatsuba and Ofman [KO63]) cannot be adapted to our approach so easily without extending the mathematical knowledge base by corresponding lemmata.

The example of the polynomial addition is surely a trivial one, we have chosen it solely for presentation reasons. In particular it is very likely to be correct in any real-world implementation, since it is well tested and does not depend on sophisticated mathematical theorems for which fuzzy boundary cases must be considered. For the sake of argument, let us assume an error in the implementation, for instance, in the second case of the polynomial addition algorithm in Figure 5 the `cons-poly` statement was forgotten, so that the algorithm has the following (incorrect) form

$$(> (e_{1_j} \cdots e_{r_j})(f_{1_k} \cdots f_{r_k}))$$
$$\texttt{(tactic "pop-first")}$$
$$\texttt{(poly-add } \sum_{i=j+1}^{n} \alpha_i x_1^{e_{1_i}} \cdots x_r^{e_{r_i}} \ \sum_{i=k}^{m} \beta_i x_1^{f_{1_i}} \cdots x_r^{f_{r_i}}\texttt{)}$$

In the computation of $((x^2+6)+(2x^2-12x+3))$ that we have discussed above, the second case is never used, and the computation would be correct although the program is not.

If we now change the order of addition of our polynomials p and q to $q+p$ we get the following incorrect result from the changed algorithm:

$$((x^2 + 6) + (2x^2 - 12x + 3)) = (3x^2 + 9)$$

Inserting the proof plan generated by the faulty algorithm then yields

$((2x^2 - 12x + 3) + (x^2 + 6))$	
$(3x^2 + ((-12x + 3) + 6))$	(`mono-add`)
$(3x^2 + (3 + 6))$	(`pop-first`)
$(3x^2 + 9)$	(`mono-add`)

In checking, the proof checker would see that the `pop-first` step is not justified, since the expansion corresponds to the application of the law of associativity. This would yield $((-12x+3)+6) = (-12x+(3+6))$ and thus would not be applicable during the expansion. Thus the proof plan and consequently the calculation would be rejected by ΩMEGA.

Note that in a large system with literally millions of possible cases, the correctness of a calculation like $(x^2+6)+(2x^2-12x+3)$ depends only on a tiny subset of the whole program. It is a strength of our approach, that only the calculations that are necessary for a given proof would be checked. This has the advantage that errors on different levels can be detected (in particular, on the levels of algorithms, of compilers, and of processors). Of course, for very long computations checking can be pretty expensive. Moreover, highly elaborated and efficient algorithms in state of the art CAS might be hard to augment with proof plan generation modes. As we have seen in the example above, the mathematical knowledge in the database has to reflect the mathematical knowledge in the algorithm in order to easily decorate the algorithms by a proof plan generation mode. However, to extend and prove corresponding lemmata is not a trivial task for sophisticated algorithms. In particular such an approach would go very much in the direction of program verification.

Even if it proves practically impossible to extract the 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 which the mathematics is easier and that are more easily decorated by a proof plan generation mode, for the proof extraction phase. Systems like *Axiom* [JS92] or *MuPAD* [Fuc96] seem to come closest among standard CAS to the needs for a proof plan generation, since one can already attach axiomatisations to algorithms.

5. Conclusion

In this work we have reported on an experiment of integrating a computer algebra system into the interactive proof development environment ΩMEGA, not only at the *system* level, but also at the level of *proofs*. The motivation for such an integration is the need for support of a human user when his/her proofs contain non-trivial computations. We have shown that the proof planning paradigm in general and the ΩMEGA system in particular provide an open environment for such an extended integration that supports different integration levels.

In our approach it is not possible to use a standard CAS for the integration as it is, since such a system provides answers, but no directly usable justifications from which proof plans can be extracted. This, however, 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 has a proof plan generation mode for the generation of communicable and checkable proofs. Since we want to achieve the 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 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 correspond to natural deduction proofs which can be mechanically checked by a simple proof checker. In the case that the CAS has made a mistake the proof checker will detect it.

The general idea and the fundamentals of the integration of a CAS into an MRS are independent from the concrete proof development environment ΩMEGA and the concrete computer algebra system μ-CAS. It can be realised in any plan-based theorem prover. Proof extraction can even be realised on any tactic-based system and with any CAS that can protocol its calculations in form of tactics. *Axiom* [JS92] and *MuPAD* [Fuc96] seem to be best suited for a corresponding extension since one can already attach axiomatisations to algorithms. If in addition the algorithms could be enriched in a way that they produce protocol information in every computation step, that is, state which of the attached axioms are used and what the particular instantiations are, the system would probably fit in with our approach pretty well.

A useful extension of our approach would consist in the usage of various algorithms for the same computation, for instance, one as a fast and efficient algorithm that is not suitable for knowledge extraction while searching for a proof. Afterwards, when actually documenting the whole proof a less efficient algorithm, which is optimised to find short proofs, can provide a complete proof plan.

Although the correctness issue can be achieved by a tactic-based approach as well and does not need the specifications that are used in proof planning, the full strength of an integration where considerable automated support is provided cannot be achieved on this level, since it is not possible to perform mechanical reasoning about the tactics. Such an automation can, however, be achieved by the proof planning approach, where the proof planner can automatically call a CAS procedure, when the conditions in the corresponding method are met. The usefulness of an integration on this level can already be seen in the case of our simple μ-CAS: After the integration we are able to prove optimisation problems which were out of reach without such a support. On the other hand, the system is able to give explanations of the involved computations at various levels of abstraction. A feature that is missing from todays CAS.

From our experiments we expect that the successful integration of any powerful computer algebra systems would considerably enhance the reasoning power of any mechanised reasoning system.

Acknowledgements The work presented in this paper was supported by the "Deutsche Forschungsgemeinschaft" in SFB 378, project OMEGA. It benefited a lot from discussions in the Calculemus interest group.

The authors would like to thank Lassaad Cheikhrouhou for his help with ΩMEGA's proof planner and with coding the methods for our examples. Furthermore, we would like to thank Deepak Kapur and the anonymous referees for carefully reading earlier versions of the paper and for their detailed comments that helped us to improve the presentation considerably.

References

And80. P. B. Andrews. Transforming matings into natural deduction proofs. In W. Bibel and R. Kowalski, editors, *Proceedings of the 5th CADE*, pages 281–292, Les Arc, France, 1980. Springer Verlag, LNCS 87.

AvLS96. J. Abbot, A. van Leeuwen, and A. Strotmann. Objectives of OpenMath. Technical Report 12, RIACA, Eindhoven, June 1996.

Bar96. H. Barendregt. *Computations and formal proofs in type theory.* Talk at the 2nd Meeting of the CALCULEMUS Project, Schloß Dagstuhl, Germany, 18.11.-20.11.1996. See also *The quest for correctness*, available as URL:
`ftp://ftp.cs.kun.nl/pub/CompMath.Found/quest.ps.Z`, 1996.

BCF+97. C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMEGA: Towards a mathematical assistant. In W. McCune, editor, *Proceedings of the 14th CADE*, pages 252–255, Townsville Australia, 1997. Springer Verlag, LNAI 1249.

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 S. Moore, editors, *The Correctness Problem in Computer Science*, pages 103–184. Academic Press, 1981.

BSvH+93. A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *AI*, **62**:185–253, 1993.

Buc85. B. Buchberger. Symbolic Computation (An Editorial). *J. Symbolic Computation*, **1**:1–6, 1985.

Buc96. B. Buchberger. Using *Mathematica* for Doing Simple Mathematical Proofs. Invited Talk at the 4th Tokyo Mathematica Users' Conference, November 2-3, 1996.

Buc96a. B. Buchberger. Mathematische Software-Systeme: Drastische Erweiterung des "Intelligenzniveaus" entsprechender Programme erwartet. *Informatik Spektrum*, **19/2**:100–101, 1996.

Bun88. A. Bundy. The use of explicit plans to guide inductive proofs. In E. L. Lusk and R. A. Overbeek, editors, *Proceedings of the 9th CADE*, pages 111–120, Argonne, Illinois, USA, 1988. Springer Verlag, LNCS 310.

CGG+92. B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt. *First leaves: a tutorial introduction to Maple V.* Springer Verlag, 1992.

CGZ94. S.-C. Chou, X.-S. Gao, and J.-Z. Zhang. *Machine Proofs in Geometry: Automated Production of Readable Proofs for Geometry Theorems.* World Scientific, Singapore, 1994.

Cho88. S.-C. Chou. *Mechanical geometry theorem proving.* Mathematics and its applications. D. Reidel Publishing Company, Dordrecht, 1988.

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*, 1991. Springer Verlag, LNCS 509.

Con86. R. L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, 1986.

CZ92. E. Clarke and X. Zhao. Analytica – A theorem prover in Mathematica. In *Automated Deduction*, pages 761–763, 11th International Conference on Automated Deduction, Saratoga Springs, New York, 15-18 June 1992.

Fuc96. B. Fuchssteiner et al. (The MuPAD Group). *MuPAD User's Manual.* John Wiley and Sons, 1996.

GPT96. F. Giunchiglia, P. Pecchiari, and C. Talcott. Reasoning theories – towards an architecture for open mechanized reasoning systems. In F. Baader and K. U. Schulz, editors, *Frontiers of combining systems (FroCoS-1) : 1st International Workshop*, pages 157–174 , Munich, Germany, 1996. Kluwer Academic Publishers.

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.

Hea95. A. C. Hearn. Reduce user's manual: Version 3.6. Technical Report, Rand Corporation, Santa Monica, CA, USA, 1995.

How88. D. J. Howe. Computational metatheory in Nuprl. in E. Lusk and R. Overbeek, editors, *Proceedings of the 9th CADE* pages 238–257, Argonne, Illinois, USA, 1988. Springer Verlag, LNCS 310.

HKK+94. X. Huang, M. Kerber, M. Kohlhase, E. Melis, D. Nesmith, J. Richts, and J. Siekmann. Ω-MKRP: A proof development environment. In A. Bundy, editor, *Proceedings of the 12th CADE*, pages 788–792, Nancy, 1994. Springer Verlag, LNAI 814.

HKKR94. X. Huang, M. Kerber, M. Kohlhase, and J. Richts. Adapting methods to novel tasks in proof planning. In B. Nebel and L. Dreschler-Fischer, editors, *KI-94: Advances in Artificial Intelligence – Proceedings of KI-94, 18th German Annual Conference on Artificial Intelligence*, pages 379–390, Saarbrücken, Germany, 1994. Springer Verlag, LNAI 861.

HF96. X. Huang and A. Fiedler. Presenting machine-found proofs. In M. A. McRobbie and J. K. Slaney, editors, *Proceedings of the 13th CADE*, pages 221–225, New Brunswick, New Jersey, USA, 1996. Springer Verlag, LNAI 1104.

HT93a. 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.

HT93b. J. Harrison and L. Théry. Reasoning about the reals: The marriage of HOL and Maple. In A. Voronkov, editor, *Proceedings of the 4th International Conference on Logic Programming and Automated Reasoning (LPAR'93)*, pages 351–353, St. Petersburg, Russia, 1993. Springer Verlag, LNAI 698.

JS92. R. D. Jenks and R. S. Sutor. *AXIOM: The Scientific Computation System.* Springer Verlag, 1992.

Kap88. D. Kapur. A refutational approach to theorem proving in geometry. *Artificial Intelligence*, **37**:61-93, 1988.

KO63. A. Karatsuba and Y. Ofman. *Multiplication of Multidigit Numbers by Automata*. Soviet Physics-Doklady, 1963.

Kow79. R. Kowalski. Algorithm = Logic + Control. *Communications of the Association for Computing Machinery*, **22**:424–436, 1979.

McC94. W. W. McCune. Otter 3.0 reference manual and guide. Technical Report ANL-94-6, Argonne National Laboratory, Argonne, Illinois, USA, 1994.

WiW89. Diplomthemen SS-89 Nr. 35. Fachschaft Wirtschaftswissenschaften, Universität des Saarlandes, Saarbrücken, Germany, 1989.

Wol96. S. Wolfram. *The Mathematica Book: Version 3.0*. Wolfram Media, Inc., Champaign, IL, 3rd edition, 1996.

Wu94. W. Wu. *Mechanical Theorem Proving in Geometries: Basic Principles*. Texts and monographs in symbolic computation. Springer, Wien, 1994.

Zip93. R. Zippel. *Effective Polynomial Computation*. Kluwer Academic Press, 1993.

Address for correspondence:
Manfred Kerber
School of Computer Science
The University of Birmingham
Birmingham B15 2TT, England
e-mail: M.Kerber@cs.bham.ac.uk
Tel: (+44)-121-414-4787
Fax: (+44)-121-414-4281