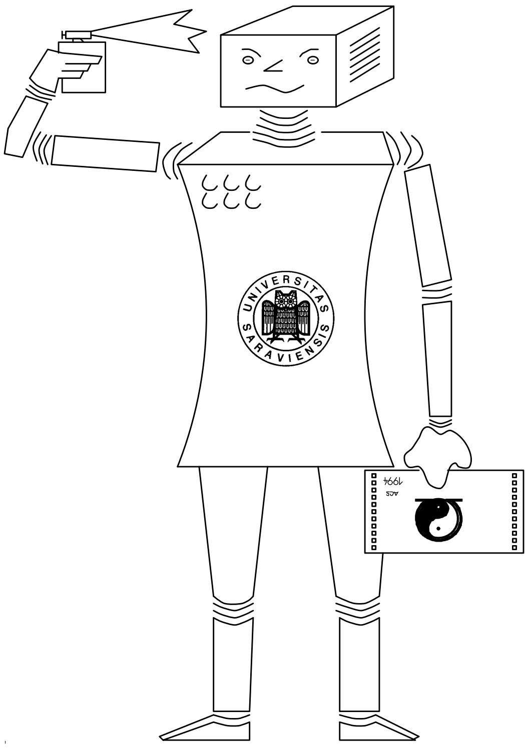
UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D-66041 SAARBRÜCKEN
GERMANY
WWW: http://jswww.cs.uni-sb.de/pub/www/



SEKI Report

An Integration of Mechanised Reasoning and Computer Algebra that Respects Explicit Proofs

Manfred Kerber, Michael Kohlhase, and Volker Sorge

SEKI Report SR-96-06

An Integration of Mechanised Reasoning and Computer Algebra that Respects Explicit Proofs

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

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

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. Even more importantly, proof and computation are often interwoven and not easily separable. 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. This can be achieved 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. We present an implementation of our ideas and exemplify them using an automatically solved extended example.

Changes in the criterion of 'rigour of the proof' engender major revolutions in mathematics

H. Poincaré, 1905

1 Introduction

The dream of machine assistance in doing mathematics by far predates the advent of electronic computers: it is at least as old as the invention of the first working mechanical calculators by Schickard (1623), Pascal (1642), and Leibniz (1671). Although the technical realisation was quite restricted in the seventeenth century, the dream already then went far beyond performing the fundamental operations of arithmetic for relatively small numbers. Leibniz himself already had the idea to use such a calculator for proving mathematical theorems. He wanted to develop a *lingua characteristica universalis* and a corresponding *calculus ratiocinator*, whose alleged purpose was to solve mathematical and everyday problems stated in the lingua universalis by mere calculations (*"calculemus"* [Lei, Lei86]). This dream has inspired philosophers and logicians over the centuries, it can be seen as a driving force behind the rapid development of logic, starting with the works of Boole [Boo54] and Frege [Fre79] in the last century. In the first part of our century, mechanical calculators were perfected and their development has come to an end. The computer and the development of high-level languages were prerequisites for the mechanisation of logic as well as for the realisation of mechanical symbolic calculations, we could witness in the last forty years. Of course none of the two fields, neither Mechanised Reasoning nor Computer Algebra, 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).

In the remainder of this introduction we briefly summarise key points of mechanised reasoning systems as well as of computer algebra systems. By its nature, such a short description has to abstract from many details and oversimplify the historic development.

1.1 Mechanised Reasoning Systems

Mechanised Reasoning Systems (for short MRS in the following) can be considered as an attempt to realise Leibniz' dream of a calculus ratiocinator (at least partially, for instance for a formal application area like mathematics). The emergence of MRS heavily relies on the vehement development of modern logic starting with Boole [Boo54] and Frege [Fre79] and continued by the epochal work in the first third of this century (compare [WR10, Göd30, Her30, Gen35] in order to mention only some of the main works). This theoretical work was continued in early artificial intelligence (AI) research with the implementation of inference machines, which were among the first existing AI systems [NSS57].

A theorem-proving system may be 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 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. 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, that is, systems based upon some computer-oriented inference system. The main break-through in this branch of automated reasoning can be seen as the invention of the resolution principle [Rob65] by which the search spaces involved with proof search in a calculus could drastically be reduced. The most important aspect at the calculus is that it replaces non-deterministic instantiation of variables by algorithmic unification. Modern systems derive their strength from their ability to maintain and manipulate very large search spaces. Their strength can be truly remarkable, but the general complexity results demonstrate clearly that no algorithm can be constructed to practically solve arbitrary tasks (even propositional logic is in a class that is generally considered intractable).

Interactive proof checking and *proof development systems* have been built with the aim of achieving a new standard of rigour in mathematical proof. As pointed out by de Bruijn, the developer of one of the earliest systems, AUTHOMATH, only a small part of mathematical literature today is absolutely flawless. To improve this situation, interactive proof checkers have been developed that carry out the meticulous final checking. In more recent systems like NUPRL [Con86], ISABELLE [Pau90], and IMPS [FGT90], a lot of attention has been paid to the user. In particular, these systems are no longer mere interactive proof checkers but normally incorporate some human-oriented proof techniques that are encoded and represented in so-called *tactics*, first used in LCF [GMW79]. Tactics are programs that manipulate the current state of the proof not only by the application of a single calculus step, but by a whole series of such steps. In this way one user interaction, namely the call of a single tactic, results in a sequence of steps. In all these systems, the proof is essentially found by the user with a little help from the machine, rather than vice versa: a lot of machine support with a little (conceptual) help from the user.

Human-oriented theorem-proving systems have attracted growing attention after the initial enthusiasm for machine-oriented theorem provers died down and the limitations of later systems became more apparent. With such systems (e.g. MUSCADET [Pas89]), one tries to model the dynamic search process for a proof as well. A particular way for achieving this goal is to extend tactics to so-called *methods* by adding specifications. Intuitively speaking, a method contains a piece of knowledge for solving or simplifying problems or transforming them into a form that is easier to solve. Specifications of methods essentially consist of pre- and postconditions for the method to allow connecting them to proof plans. These proof plans are the basic elements of a more general planning framework (e.g. CLAM [BvHHS90, BSvH$^+$93]). The reasoning power of such plan-based systems is not derived from a (theoretically) complete underlying reasoning calculus, but relies on domain-specific problem solving knowledge. Incidently, theoretical completeness of the methods is of minor importance practically. What matters is, whether a problem solver, either human or computer, has the necessary domain-specific problem solving knowledge that offers the chance to solve the problem with a reasonable amount of search. Since such domain-specific knowledge is inevitably limited for non-trivial domains and hence practically incomplete, a declarative approach to representing methods that enables a mechanical modification and adaptation to novel tasks has been developed [HKRS94].

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. We think that the combination of these complementary approaches inherits more the advantages than the drawbacks, because for most tasks domain-specific as well as domain-independent problem-solving know-how is required and for difficult task 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 proofs, it neglects the fact that for many mathematical fields, the everyday work of mathematicians does only partially consist in proving theorems. Calculation plays an equally important role. 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 much 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 can not) provide the control for doing a concrete computation.

1.2 Computer Algebra Systems

A principal motivation for building the first computers was to facilitate the myriads of numerical computations that were necessary during World War II in particular in order to decode secret messages. But the pioneers like Turing and von Neumann, clearly saw that the possible applications are much broader. With the development of high-level computer languages such as LISP, it became quite easy to implement symbolic algorithms for the manipulation of algebraic expressions like the multiplication of polynomials, or the derivation and integration of functions. Early Computer Algebra Systems (CAS for short) have developed from collections of such algorithms and data structures [Hea87].

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 to 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 sub-field of mathematics that has developed the mathematical theory and tools. Today many algorithms are added directly by the mathematically trained specialists without any deviation over software engineers. This guarantees a fast information flow. In addition, many advanced systems have reached a user friendliness that it is not only possible to use the systems without being a software engineer, but that it is also possible to easily add new algorithms without knowing much of the internal details of the system. Some mathematical theories are theoretically and algorithmically very well understood like the theory of Gröbner bases such that they can be successfully applied in areas that seem on the first view not directly related like geometrical theorem proving.

Most modern systems [Wol91, CGG+92, Dav92] 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 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 [AvLS95], in which one strives for a standard of the CAS communication protocols. Currently the main emphasis is laid on standardising the syntax and the computational behaviour of the mathematical objects, while their properties 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 [Dav92], 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.

As one of its possible application areas, CAS have been successfully applied to prove mathematical theorems and indeed theorem proving can theoretically be reduced to symbolic computation (and as Gödel has shown, even to numerical computation). This is highly successful for certain areas like geometric theorem proving [Wan91, Ueb95], but it is not practical for the general case.

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.

In the next section we will further motivate the integration of mechanised reasoning systems and computer algebra systems and discuss possible ways of their integration. In Section 3 we advocate a particular approach built on top of the proof planning paradigm and describe the implementation of a prototype. Finally in Section 4 we illustrate the approach by an extended example.

2 Syntheses of Reasoning and Calculation

In this section we again motivate the integration of mechanised reasoning and computer algebra, furthermore we describe some of the most prominent attempts of integrating CAS and MRS in the next subsection. The second subsection is devoted to the philosophy of mathematics, since it has a great influence on the architecture of systems that support mathematicians in their work. This fact has led—to our knowledge—to more discussions in the field of mechanised reasoning than in that of computer algebra. This may be due to the fact that it directly affects the very sensible notions of proof, truth, and correctness. Since it also directly affects the possible syntheses of mechanised reasoning systems and computer algebra systems, we give a short, simplified recapitulation of this discussion. Then we adopt one of the philosophical positions and show how the basic features of our own MRS meets this position and describe the main problems that arise from any integration of a CAS into such an MRS. We conclude this section by relating the existing approaches to the philosophy of mathematics and give an alternative approach to such an integration.

2.1 Motivation and Related Approaches

Traditional MRS are very weak, when it comes to computation with mathematical objects. In contrast, CAS manipulate highly optimised representations of the objects and are therefore very useful for solving subgoals in mathematical deduction. Thus they should be part of any mathematical assistant system used for verifying intuitive mathematical proofs. Moreover, looking into mathematical textbooks reveals that for many areas neither computation nor purely logical deduction dominate proofs. 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 give a short description of some of these experiments and roughly categorise them 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. 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) 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.

The second category [HT93a] 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. Indeed from the point of applicability, the results of the CAS help only in cases, where the verification of a result has a lower complexity 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 such as solving a minimisation problem both types of sub-calculations are contained. While this should provide at least some hints to an MRS for the construction of a proof, it is normally not trivial to separate the different parts. 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]).

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 construct 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 from the viewpoint of correctness most appealing, although the assumption that the original system must be correct can hardly be expected to be true 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.

Before introducing our own approach we will use the next subsection to discuss some philosophical aspects of proof and correctness, which directly influence basic architecture decisions of our work.

2.2 The Notion of Proof and Correctness

Mathematics generally enjoys the prestige of being the *correct* scientific discipline *par excellence*. This reputation comes from the requirement that every claim must be justified by a rigorous proof. The ultimate goal of many MRS is to support mathematicians in the task of constructing such a proof. This is not trivial, since in traditional mathematical practice, proofs are not given in terms of single calculus rules but at a level of abstraction that conveys the main ideas. This procedure is based on the conviction that a detailed logic-level proof could be generated if necessary, which, however, would be too boring. In addition, most mathematicians have no interest in "formal" (logical-level) proofs. The correctness of their relatively informal proofs is usually guaranteed by a social process of critical reviewing. It turns out nevertheless that this social process often fails to reach its goal: in most cases this is only caused by minor and reparable errors, but from time to time a false theorem is assumed to have been proved. The history of Euler's polyhedron theorem is a well-known story of such repeated falsification and patching [Lak76]. The development of mathematical logic and in particular of automated reasoning systems can be viewed as an attempt to achieve a new quality of correctness. The philosophy behind this enterprise lies in the belief that a meticulous machine guarantees the correctness by carrying out only correct deduction steps on a calculus level.

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[1]. 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. Please note that essentially all three categories of approaches enumerated in subsection 2.1 are of the first kind, that is, they assume the trustworthiness of the CAS and/or the MRS. A partial exception is the practically not feasible variant of the second category, were proofs using computer algebra calculation have to be completed by the MRS

[1]However, not that if the metatheory is constructive itself, such as in [ASG96], then the meta-level proof of soundness of the extensions (such as decision procedures) can be used to extract a transformation procedure of object level proofs, that can in turn be used to complete the gaps left by the decision procedures

before being accepted, as well as the third one, which can relatively easily be extended to produce detailed proofs.

2.3 The Proof Development Environment Ω-MKRP

In our own MRS, the Ω-MKRP-system to be described in the following, we advocate the stricter nominalist approach (the second approach described in the previous subsection). 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 the Ω-MKRP proof development environment [HKK+94], 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 Figure 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). If this is not the case, e.g., for automated theorem provers based on resolution, a transformation into the natural deduction formalism must be carried out. To summarise our view of proofs, in Ω-MKRP for any 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. Neither the process of generating proofs nor that of checking them is replaced by the machine but only supported. If a human mathematician want to see a proof, he/she can do so on an appropriate level of abstraction.

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.

To understand the basics of the proof planning process, please remember that the goal of proof planning is to fill gaps in a given partial proof tree by forward and backward rea-

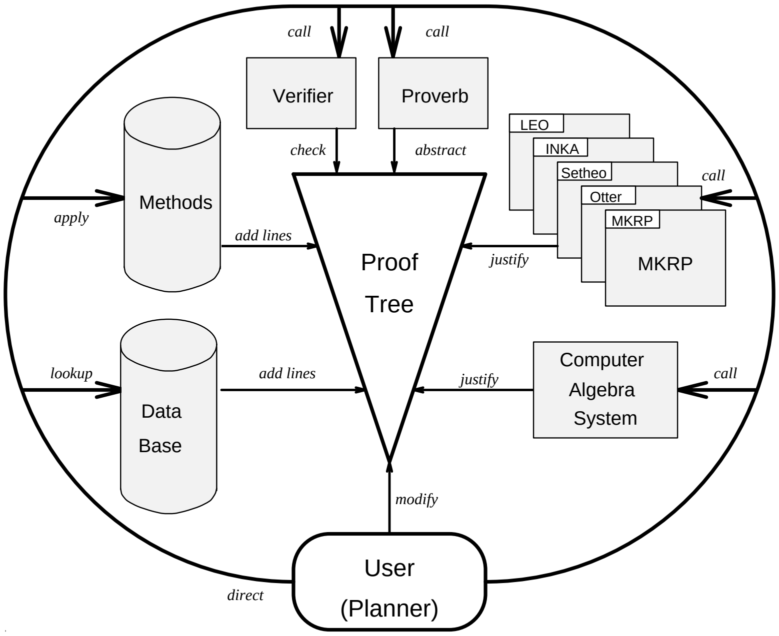


Figure 1: Architecture of Ω-MKRP

soning [HKKR94] (proof plans were first introduced by Bundy see [BvHHS90, 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* (i.e. for a linear planner, a sequence of instantiated plan operators) from a given *initial state* to a *terminal state*.

Specifically a *planning state* contains a subset of proof lines which are formulated in Gentzen's natural deduction (ND for short) calculus in the current partial proof that correspond to the boundaries of a gap in the proof. This subset can be divided into *open lines* (that must be proved to bridge the gap) and *support lines* (that can be used as premises to bridge it). The initial planning state consists of all lines in the initial problem; the assumptions are the support lines and the conclusion is the only open line. The terminal planning state is reached when there is no more open line in the planning state.

Once a complete proof plan is found, all methods (i.e. their tactics) in the proof plan are successively executed in order to construct a calculus level proof. The verification phase, which follows the application of the methods, may result in a recursive call to the planner or in backtracking. While a recursive call refines a plan and models hierarchical planning, the backtracking rejects the plan and calls the proof planner in order to find an alternative one.

The most interesting component in our declarative hierarchical approach to proof planning is that of a *method*, an operator for plan construction. Formally, a method is defined as a 6-tuple with the components:

Declarations: A signature that declares the meta-variables used in the method,

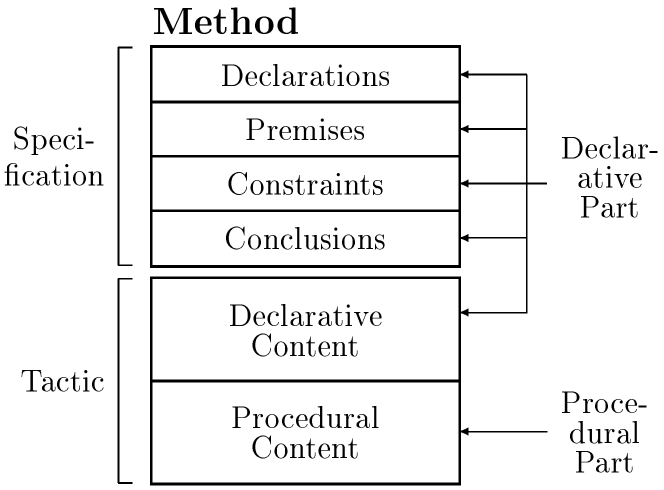
Premises: Schemata of proof lines which are used by this method to prove the conclusions,

Constraints: Additional restrictions on the premises and the conclusions, which cannot be formulated in terms of proof line schemata,

Conclusions: Schemata of proof lines which this method is designed to prove,

Declarative Content: A piece of declarative knowledge interpreted by the procedural content. This slot is currently restricted to schemata of partial proofs,

Procedural Content: Either a standard procedure interpreting the declarative content, or a special purpose inference procedure.



Two examples of methods are displayed in the section on the extended example, Section 4.

2.4 Integrating Computations into Explicit Proofs

If we take the idea of generating explicit proofs seriously also for computations and do not want to give up the nominalist paradigm, we can neither just take existing systems nor follow the approach of meta-theoretic extensions, since Ω-MKRP 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

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

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.

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.

3 SAPPER

In this section we describe SAPPER (**S**ystem for **A**lgorithmic **P**roof **P**lan **E**xtraction and **R**easoning), which 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 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.

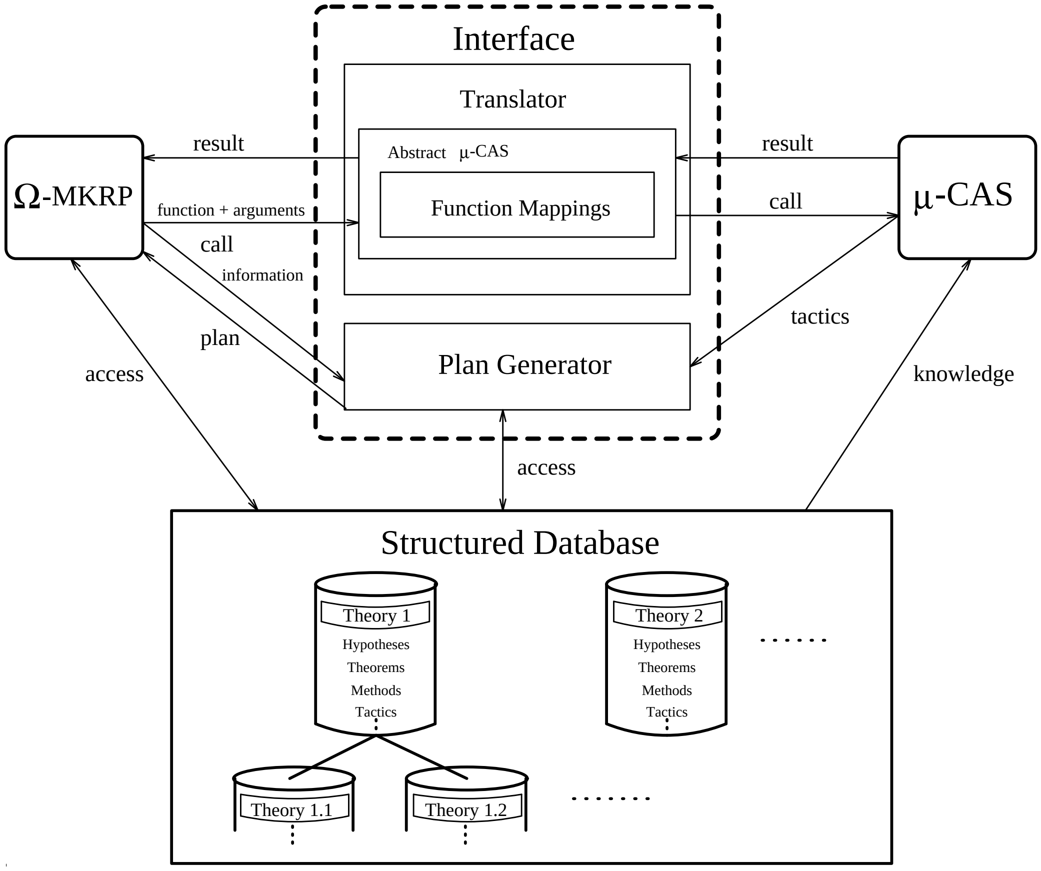


Figure 2: Interface between Ω-MKRP and Computer Algebra Systems

3.1 Architecture

The SAPPER system can be seen as a generic interface for connecting Ω-MKRP with one or several Computer Algebra Systems (see Figure 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 other[3].

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.

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

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*[4]—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 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, as in the first approach we have described in Section 2.1. 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

[4] In a reimplementation of SAPPER we would use the well-established OPEN MATH protocol [AvLS95] as a lingua franca on the CAS side.

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. Figure 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.

3.2 A First Example of the Integration

As a first 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 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 Ω-MKRP is a variant of the simply typed λ-calculus, so the polynomials are represented as λ-expression where the formal parameters $x_1, \ldots, x_r$ 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 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. 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[5] 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 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.

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

Figure 3: Polynomial addition in μ-CAS.

Obviously, the only expansions of the original algorithm needed for the verbose mode are the additional `(tactic ...)` statements[6]. 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 apply this algorithm to the simple polynomials

$$p := 3x^2 + 2x + 5 \qquad\qquad q := 5x^3 + x + 2$$

[5]We employ the intuitive well-ordering on the exponents and thus make use of the operators $>, <, =$ in an intuitive sense.

[6]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 to be concerned what form a monomial actually has during the course of the algorithm.

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.

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:

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

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. 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 embedded in contexts of course.)

$$\begin{array}{ll} ((3x^2 + 2x + 5) + (5x^3 + x + 2)) & \\ (5x^3 + ((3x^2 + 2x + 5) + (x + 2))) & \texttt{(pop-second)} \end{array}$$

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

$$\begin{array}{ll} ((3x^2 + 2x + 5) + (5x^3 + x + 2)) & \\ (((3x^2 + 2x + 5) + 5x^3) + (x + 2)) & \texttt{(associativity)} \\ ((5x^3 + (3x^2 + 2x + 5)) + (x + 2)) & \texttt{(commutativity)} \\ (5x^3 + ((3x^2 + 2x + 5) + (x + 2))) & \texttt{(associativity)} \end{array}$$

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:

$$\begin{array}{ll} \forall a \forall b\textbf{.}\, (a + b) = (b + a) & \texttt{(HYP)} \\ \forall b\textbf{.}\, ((3x^2 + 2x + 5) + b) = (b + (3x^2 + 2x + 5)) & (\forall\texttt{E}\ \ (3x^2 + 2x + 5)) \\ ((3x^2 + 2x + 5) + 5x^3) = (5x^3 + (3x^2 + 2x + 5)) & (\forall\texttt{E}\ \ 5x^3) \end{array}$$

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. As the proof itself is rather boring we omit presenting it here and refer to the more elaborated and interesting example in Section 4.

3.3 Refinements, and Problems of the Basic Idea

In order to keep the communication overhead low and improve the structure of proofs, the verbose output and the resulting proof plan should be hierarchically structured. For instance an algorithm that calls sub-procedures can represent these calls explicitly in a verbose output of `call-cas` but run the sub-algorithm themselves in quiet mode. Thus, during the expansion of the corresponding proof plan, the CAS is called on the appropriate subproblems, allowing the generation of further sub-proof-plans, and so on. This very simple mechanism allows to map the natural hierarchical (modular) structure of the algorithm onto the proof plan, which then contains levels of abstraction ranging from a single step (completely trusting the computation of the CAS) over intermediate levels to a full ND proof. Moreover, since the proof plan expansion is only carried out by need, the user can select the level up to which he/she wants to expand the proof and which calculations he/she feels comfortable with.

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 in System like *Axiom* [Dav92] or *MuPAD* [Fuc96] to return domain specific tactics in the verbose output.

4 Extended Example

In this section we present an example that cannot easily be solved by either a mechanised reasoning system or a computer algebra system, but that needs the combined efforts of two systems of each kind. The concrete task at hand is to minimise the costs for running a machine while producing a certain product. This example serves our purposes for several reasons. Firstly, it allows us to show the interaction of proof planning 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 a real-world example, 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 the following we solve this problem using our SAPPER-system. First we present the problem.

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*

- $r_1 = (4d^2 - 18d + 7)\,\frac{l}{prod}$
- $r_2 = (0.5d^2 - 1.5d + 0.5)\,\frac{kWh}{prod}$

and the prices for water and electricity

- $p_1 = 0.5\,\frac{DM}{l}$
- $p_2 = 2\,\frac{DM}{kWh}$

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

In order to solve this problem, we obviously need a theory for the formalisation of this problem that can handle both numerical parts and denomination of cost functions. For this purpose we apply the theory of optimisation problems in economy that provides a type υ of units and a type κ of costs. υ covers the different units of denominations—in our example l (for volume), kWh (for work), $prod$ (for the number of products) and DM (for the price)—while the cost-type κ can intuitively be understood as that of pairs of real numbers and units. Cost functions are determined by a real function and a pair of units (read as input/output units) and are therefore of type $\kappa \to \kappa$. Note, that just as in the real world, addition ($\oplus$) 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 P.

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

Optimisation 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 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[7]

$$\text{T} \quad \mathcal{H} \vdash Opt([cf(\lambda d.\,4d^2 - 18d + 7, l, prod) \oplus cf(\lambda d.\,0.5d^2 - 1.5d + 0.5, kWh, prod)], [1,7])$$

[7] 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 Ω-MKRP 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 of the nominalist point of view. A realist may find himself in the position, that he/she 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.

where $\mathcal{H}$ is a set of hypotheses that are needed for the complete proof (11 of these hypotheses are needed only for the polynomial manipulations of the CAS) and for instance the price axioms[8]

$$price(\lambda d. 0.5, DM, l) \qquad price(\lambda d. 2, DM, kWh)$$

The planner solves the problem, by generating a high-level proof plan consisting of methods from it's domain specific method base on economics exam questions[9].

We are going to outline this process by describing its major steps. In particular, we will demonstrate how the proof planner of Ω-MKRP and the Computer Algebra component of the SAPPER-system interact, and make explicit, on which entries of a mathematical data-base this interaction depends. The planner finds the following simple proof plan:

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 5) for finding total minima that makes implicit use of Rolle's theorem

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

Formally we get the following equivalence:

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

Note that the proof of Rolle's theorem has to be accessible in the current theory, and furthermore, the data-base has to contain a formal proof in order to ensure correctness.

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 4 and 5 we have given slightly simplified presentations of the `Mult-by-Price` and `TotMin-Rolle` method[10].

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

[8] The formalisation allows arbitrary rational functions, but we only need constant ones here.

[9] 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 Ω-MKRP.

[10] 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 $f, g, v, w, I, \phi, \phi', \psi, \psi'$: variable			
Premises	$L_1, \oplus L_3$			
Constraint	$\psi \leftarrow$ (2ndarg(termocc(cf, ϕ)) $\neq DM \rightarrow$ termocc(cf, ϕ)) $g \leftarrow$ 1starg(ψ) $v \leftarrow$ 2ndarg(ψ) $w \leftarrow$ 3rdarg(ψ) $\psi' \leftarrow cf(g \cdot f, DM, w)$ $\phi' \leftarrow$ replace(ψ', ψ, ϕ)			
Conclusions	$\ominus L_4$			
Declarative Content	(L_1)	H_2	$\vdash price(f, DM, v)$	(J_1)
	(L_2)	H_1, H_2	$\vdash cf(g, v, w) = \psi'$	$(\mathrm{P}(L_1))$
	(L_3)	H_3	$\vdash Opt(\phi', I)$	(Call-CAS)
	(L_4)	H_3	$\vdash Opt(\phi, I)$	(=subst L_3 L_9)
Procedural Content	`schema − interpreter`			

Figure 4: The `Mult-by-Price` method

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_3, 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.

The tactic part of the method plays its role not in the planning phase but in the execution phase. It consists of the procedural content, which consists for our examples 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 in the cost function of our optimisation problem 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 T. Thus our cost function $[cf(\lambda d. 4d^2 - 18d + 7, l, prod) \oplus cf(\lambda d. 0.5d^2 - 1.5d + 0.5, kWh, prod)]$ is bound to the metavariable ϕ. By binding its arguments to g, v, w and matching line L_1 we receive the numerical part of *price* in f. Afterwards the new cost function is computed (according to axiom P) 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 value cost function stored in ϕ' is $[cf((\lambda d.4d^2-18d+7\cdot\lambda d.0.5, DM, prod)\oplus cf(\lambda d.0.5d^2-1.5d+0.5, kWh, prod)]$.

With all these metavariables 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 P (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 plan 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.

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.

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 Ω-MKRP. Again the presentation of the method in Figure 5 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 metavariable y. In our example the minimum of the cost function is at $y = 2$ and the ND-line of the form

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

will be transformed by eliminating the existentially quantified variable:

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

The rest of the proof plan is devoted to proving that the result actually a total minimum. This is done by using the definition for TotMin from above which must be applicable in 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.

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	degree$(\phi) \doteq 2$	
	$y \leftarrow$ compute with CAS$(minimum, \phi)$	
Conclusions	$\ominus L_{12}$	
Declarative Content	(L_1) $H_1 \vdash \forall f. \forall x. (f'(x) = 0 \land f''(x) > 0) \Rightarrow \mathrm{Min}(x, f)$	(J_1)
	(L_2) $H_2 \vdash \forall a. \forall b. \forall x. x \in [a, b] \Leftrightarrow (a \leq x \land x \leq b)$	(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 5: The `TotMin-Rolle` method

Consisting of only 5 methods the above proof plan gives the impression of a small proof. But expanding the plan into a partially grounded ND proof gives it a length of 90 lines, containing lines justified by the CAS. By rerunning the CAS in verbose-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 data-base. 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.

5 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 has a verbose 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 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 correspond to natural deduction proofs which can be mechanically checked. In the case that the CAS has made a mistake the proof checker can detect it on this level. The usefulness of the integration 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.

The general idea and the fundamentals of the integration of a CAS into an MRS is independent from the concrete interactive proof development environment Ω-MKRP and the concrete computer algebra system μ-CAS. It can be realised in any tactic-based system and with any CAS that can protocol its calculations in form of tactics. Moreover, the interface provides not only the opportunity of reusing tactics for several algorithms and systems, but also to compute more than one plan for a subproof or to plan subproofs separately. Set in a distributed system architecture one could think of working on a proof in Ω-MKRP interactively, while at the same time subproofs of this very proof are planned on a different machine. It would also be possible to use 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 but more verbose algorithm can provide a complete proof plan.

Such an interface and protocol has turned out to be straightforward for simple iterative and recursive algorithms but may yield problems for more complicated schemes like divide-and-conquer. Such problems and possible solutions are currently a matter of research.

References

[And80] P. B. Andrews. Transforming matings into natural deduction proofs. In W. Bibel and R. Kowalski, editors, *Proceedings of the 5^{th} International Conference on Automated Deduction (CADE–5)*, volume 87 of *Lecture Notes in Computer Science*, pages 281–292, Les Arc, Frankreich, 1980. Springer Verlag, Berlin.

[ASG96] A. Armando, A. Smaill, and J. Gallagher. Automating the synthesis of decision procedures in a constructive metatheory. In *Proceedings of AI/MATH-96*, Florida, 1996. Extended version submitted to the Annals of Mathematics and Artificial Intelligence.

[AvLS95] J. Abbot, A. van Leeuwen, and A. Strotmann. Objectives of openmath. *Journal of Symbolic Computation*, 11, 1995.

[BC92] David A. Basin and Robert L. Constable. Metalogical frameworks. Technical Report MPI-I-92-205, Max-Planck-Institut für Informatik, Im Stadtwald, Saarbrücken, Germany, February1992.

[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.

[Boo54] George Boole. *An Investigation of The Laws of Thought.* Macmillan, Barclay, & Macmillan, Cambridge, United Kingdom; reprinted by Dover Publication, New York, USA, 1958, 1854.

[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, Berlin, Germany, LNAI 449.

[CGG+92] Bruce W. Char, Keith O. Geddes, Gaston H. Gonnet, Benton L. Leong, Michael B. Monagan, and Stephen M. Watt. *First leaves: a tutorial introduction to Maple V.* Springer Verlag, Berlin, 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*, volume 509 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, June 1991.

[Con86] Robert L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1986.

[CZ92] Edmund Clarke and Xudong 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 Juni 1992.

[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.

[FGT90] William M. Farmer, Joshua D. Guttman, and F. Javier Thayer. IMPS: An interactive mathematical proof system. In Mark E. Stickel, editor, *Proceedings of the 10th CADE*, pages 653–654, Kaiserslautern, Germany, 1990. Springer Verlag, Berlin, Germany, LNAI 449.

[Fre79] Gottlieb Frege. Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. Halle, 1879. reprint in: Begriffsschrift und andere Aufsätze, J. Angelelli, editor, Hildesheim. See also in Logiktexte, Karel Berka, Lothar Kreiser, editors, pages 82–112.

[Fuc96] Benno Fuchssteiner et al. (The MuPAD Group). *MuPAD User's Manual*. John Wiley and Sons, Chichester, New York, first edition, 1996.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen I & II. *Mathematische Zeitschrift*, **39**:176–210, 572–595, 1935.

[GMW79] Michael Gordon, Robin Milner, and Christopher Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*. LNCS 78. Springer Verlag, Berlin, Germany, 1979.

[Göd30] Kurt Gödel. Die Vollständigkeit der Axiome des logischen Funktionenkalküls. *Monatshefte für Mathematik und Physik*, **37**:349–360, 1930.

[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, 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.

[Her30] Jacques Herbrand. Recherches sur la théorie de la démonstration. *Sci. Lett. Varsovie, Classe III sci. math. phys.*, **33**, 1930.

[How88] D.J. Howe. Computational metatheory in Nuprl. in Lusk and Overbeek eds, *Proceedings of CADE 9* pages 238–257. Springer-Verlag, 1988.

[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, Berlin, Germany, LNAI 814.

[HKKR94] Xiaorong Huang, Manfred Kerber, Michael Kohlhase, and Jörn Richts. Adapting methods to novel tasks in proof planning. In Bernhard Nebel and Leonie 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, Berlin, Germany, LNAI 861.

[HKRS94] Xiaorong Huang, Manfred Kerber, Jörn Richts, and Arthur Sehn. Planning mathematical proofs with methods. *Journal of Information Processing and Cybernetics, EIK*, **30**(5-6):277–291, 1994.

[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)*, volume 780 of *Lecture Notes in Computer Science*, pages 174–184. Springer Verlag, Berlin, New York, 1993.

[HT93b] John Harrison and Laurent 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)*, volume 698 of *LNAI*, pages 351–353, St. Petersburg, Russia, July 1993. Springer Verlag.

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

[Kow79] Robert Kowalski. Algorithm = Logic + Control. *CACM*, 1979.

[Lak76] Imre Lakatos. *Proofs and Refutations—The Logic of Mathematical Discovery.* Cambridge University Press, 1976.

[Lei] Gottfried Wilhelm Leibniz. Historia et commendatio linguae characteristicae universalis quae simul sit ars inveniendi et judicandi. German translation in Gottfried Wilhelm Leibniz: Gott – Geist – Güte: Eine Auswahl aus seinen Werken, pages 163–173, Bertelsmann Verlag, Gütersloh, Germany, 1947. not dated.

[Lei86] Gottfried Wilhelm Leibniz. Projet et essais pour arriver à quelque certitude pour finir une bonne partie des disputes et pour avancer l'art d'inventer. In Karel Berka and Lothar Kreiser, editors, *Logiktexte*, chapter I.2, pages 16–18. Akademie-Verlag, german translation, 1983, Berlin, Germany, 1686.

[NSS57] Allen Newell, Cliff Shaw, and Herbert Simon. Empirical explorations with the logic theory machine: A case study in heuristics. In *Proceedings of the 1957 Western Joint Computer Conference*, New York, USA, 1957. McGraw-Hill. reprinted in Computers and Thought, Edward A. Feigenbaum, Julian Feldman, editors, New York, 1963.

[Pas89] Dominique Pastre. Muscadet: An automated theorem proving system using knowledge and metaknowledge in mathematics. *AI*, **38**(3):257–318, 1989.

[Pau90] Lawrence C. Paulson. Isabelle: The next 700 theorem provers. *Logic and Computer Science*, pages 361–386, 1990.

[Pel91] Francis Jeffry 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, San Mateo, California, USA.

[Rob65] John Alan Robinson. A machine oriented logic based on the resolution principle. *Journal of the ACM*, **12**:23–41, 1965.

[Ueb95] J. Ueberberg. Interactive theorem proving and computer algebra. In J. Calmet and J. A. Campbell, editors, *Integrating Symbolic Mathematical Computation and Artificial Intelligence; Second International Conference, AISMC-2 Cambridge, United Kingdom, August 3-5, 1994; Selected Papers*, volume 958 of *Lecture Notes in Computer Science*, pages 1–9, Springer Verlag, Berlin, 1995.

[Wan91] Dongming Wang. Reasoning about geometric problems using algebraic methods. Technical report, RISC-Linz, Linz, 1991.

[WiW89] Diplomthemen SS-89 Nr. 35. Fachschaft Wirtschaftswissenschaften, Universität des Saarlandes, 1989.

[Wol91] S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer.* Addison-Wesley, 1991.

[WR10] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*, volume I. Cambridge University Press, Cambridge, United Kingdom; second edition, 1910.

[Zip93] Richard Zippel. *Effective Polynomial Computation.* Kluwer Academic Press, Boston; Dordrecht; London, 1993.