# Computer Supported Mathematics with ΩMEGA

Jörg Siekmann, Christoph Benzmüller, Serge Autexier

*Universität des Saarlandes and DFKI GmbH, Saarbrücken, Germany*

## Abstract

Classical automated theorem proving of today is based on ingenious search techniques to find a proof for a given theorem in very large search spaces – often in the range of several billion clauses. But in spite of many successful attempts to prove even open mathematical problems automatically, their use in everyday mathematical practice is still limited.

The shift from search based methods to more abstract planning techniques however opened up a paradigm for mathematical reasoning on a computer and several systems of that kind now employ a mix of interactive, search based as well as proof planning techniques.

The ΩMEGA system is at the core of several related and well-integrated research projects of the ΩMEGA research group, whose aim is to develop system support for a working mathematician as well as a software engineer when employing formal methods for quality assurance. In particular, ΩMEGA supports proof development at a human-oriented abstract level of proof granularity. It is a modular system with a central proof data structure and several supplementary subsystems including automated deduction and computer algebra systems. ΩMEGA has many characteristics in common with systems like NuPrL, CoQ, Hol, Pvs, and Isabelle. However, it differs from these systems with respect to its focus on *proof planning* and in that respect it is more similar to the proof planning systems Clam and λClam at Edinburgh.

*Email addresses:* `siekmann@dfki.de` (Jörg Siekmann), `chris@ags.uni-sb.de` (Christoph Benzmüller), `autexier@ags.uni-sb.de` (Serge Autexier).

*URLs:* `www-ags.dfki.uni-sb.de` (Jörg Siekmann), `www.ags.uni-sb.de/~chris` (Christoph Benzmüller), `www.ags.uni-sb.de/~autexier` (Serge Autexier).

# 1   Introduction

The vision of computer-supported mathematics and a system which provides integrated support for all work phases of a mathematician has always fascinated researchers in artificial intelligence, particularly in the deduction systems area, and more recently in mathematics as well. The dream of mechanizing (mathematical) reasoning dates back to Gottfried Wilhelm Leibniz in the 17th century with the touching vision that two philosophers engaged in a dispute would one day simply code their arguments into an appropriate formalism and then *calculate* (Calculemus!) who is right. At the end of the 19th century modern mathematical logic was born with Frege's Begriffsschrift and an important milestone in the formalization of mathematics was Hilbert's program and the 20th century Bourbakism.

With the logical formalism for the representation and calculation of mathematical arguments emerging in the first part of the twentieth century it was but a small step to implement these techniques now on a computer as soon as it was widely available.

In 1954 Martin Davis' Presburger Arithmetic Program was reported to the US Army Ordnance and the Dartmouth Conference in 1956, which is not only known for giving birth to artificial intelligence in general but also more specifically for the demonstration of the first automated reasoning programs for mathematics by Herb Simon and Alan Newell.

However, after the early enthusiasm of the 1960s, in particular the publication of the resolution principle in 1965 [84], and the developments in the 70s a more sober realization of the actual difficulties involved in automating everyday mathematics set in and the field increasingly fragmented into many subareas which all developed their specific techniques and systems [1].

It is only very recently that this trend appears to be reversed, with the CALCULEMUS [2] and MKM [3] communities as driving forces of this movement. In CALCULEMUS the viewpoint is bottom-up, starting from existing techniques and tools developed in the computer-algebra and deduction systems communities. MKM[3] approaches the goal of computer-based mathematics for the new millennium by a complementary top-down approach starting from existing, mainly pen and paper based mathematical practice down to system support.

---

[1]  The history of the field is presented in a classical paper by Martin Davis [35] and also in [36] and more generally in his history of the first electronic computers [37]. Another source is Jörg Siekmann [86] and more recently [87].

[2]  `www.calculemus.org`

[3]  `www.mkm-ig.org`

The ΩMEGA project aims at an integrated approach since its start in the mid 80s and it is deeply rooted in both initiatives. The ΩMEGA system is at the core of the project and it has many characteristics in common with systems like NuPrL [1], CoQ [34], Hol [47], Pvs [79], and Isabelle [80,78]. However, it differs from these systems with respect to its focus on *proof planning* and in that respect it is more similar to the proof planning systems Clam and λClam at Edinburgh [83,29]. In this article we shall first provide an overview of the main developments of the ΩMEGA project and then point to current research and some future goals.

## 2   ΩMEGA

The ΩMEGA project represents one of the major attempts to build an all encompassing assistance tool for the working mathematician or for the formal work of a software engineer. It is a representative of systems in the paradigm of *proof planning* and combines interactive and automated proof construction for domains with rich and well-structured mathematical knowledge. The inference mechanism at the lowest level of granularity is an interactive theorem prover based on a higher-order natural deduction (ND) variant of a soft-sorted version of Church's simply typed $\lambda$-calculus [33]. The logical language, which also provides some support for partial functions, is called $\mathcal{POST}$, for **p**artial functions and **o**rder **s**orted **t**ype theory. The search for a proof, however, is usually conducted at a higher level of granularity defined by *tactics* and *methods*. Automated proof search at this 'abstract' (i.e., less granular) level is called *proof planning* (see Section 2.3). Proof construction is also supported by already proven assertions, i.e. theorems and lemmata, and by calls to external systems to simplify or solve subproblems. Resource-guided search for applicable tactics, methods, and external systems is conducted by ΩAnts, an agent-based reasoning system.

### 2.1   System Overview

At the core of ΩMEGA is the *proof plan data structure* $\mathcal{PDS}$ [32], in which *proofs* and *proof plans* are represented at various levels of granularity (see Figure 1). The $\mathcal{PDS}$ is a directed acyclic graph, where *open nodes* represent unjustified propositions that still need to be proved and *closed nodes* represent propositions that are already proved. The proof plans are developed and classified with respect to a taxonomy of mathematical theories in the mathematical knowledge base MBase [42,56]. The user of ΩMEGA, or the proof planner Multi [73,64], or else the agent-based reasoning system ΩAnts [19] modify the $\mathcal{PDS}$ during proof development until a complete proof plan, where
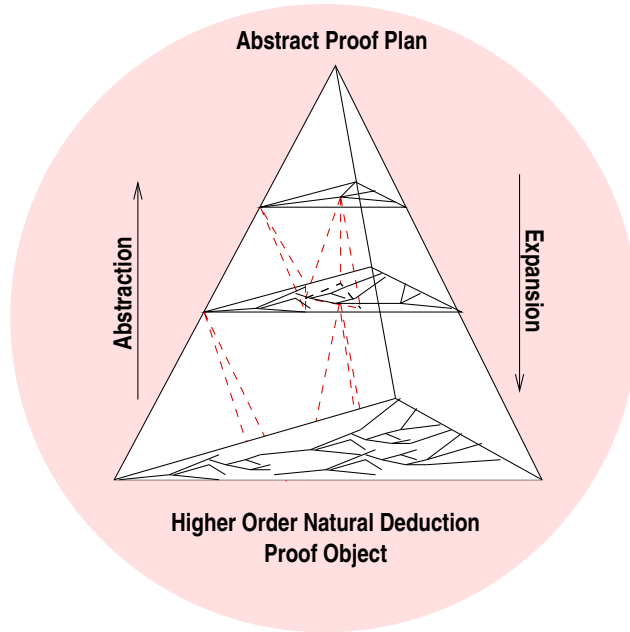
Figure 1. The proof plan datastructure $\mathcal{PDS}$ is at the core of the $\Omega$MEGA system.

all nodes are closed, has been found. They can also invoke external reasoning systems, whose results are included in the $\mathcal{PDS}$ after appropriate transformation. Once a complete proof plan at an appropriate level of granularity has been found, this plan must be expanded by sub-methods and sub-tactics into lower levels of granularity until finally a proof at the level of the logical calculus is established. After expansion of these high-level proofs to the underlying ND-calculus, the $\mathcal{PDS}$ can be checked by $\Omega$MEGA's proof checker.

Hence, there are two main tasks supported by this system, namely (i) to find a proof plan, and (ii) to expand this proof plan into a calculus-level proof; and both jobs can be equally difficult and time consuming. Task (ii) employs a combination of an LCF-style tactic based expansion mechanism as well as deductive proof search in order to generate a lower-level proof object. It is a design objective of the $\mathcal{PDS}$ that various *proof levels* coexist with their respective dynamic relationships being maintained.

The graphical user interface $\mathcal{L}\Omega\mathcal{UI}$ [90] provides both a graphical and a tabular view of the proof under consideration, and the interactive proof explanation system *P.rex* [40,39,41] generates a natural language presentation of the proof (see Figures 5 and 6).

The previously monolithic system has been split up and separated into several independent modules (see Figure 2), which are connected via the mathematical software bus MATHWEB-SB [99]. An important benefit is that MATHWEB-SB modules can be distributed over the Internet and are then remotely accessible by other research groups as well. There is now a very active MathWeb
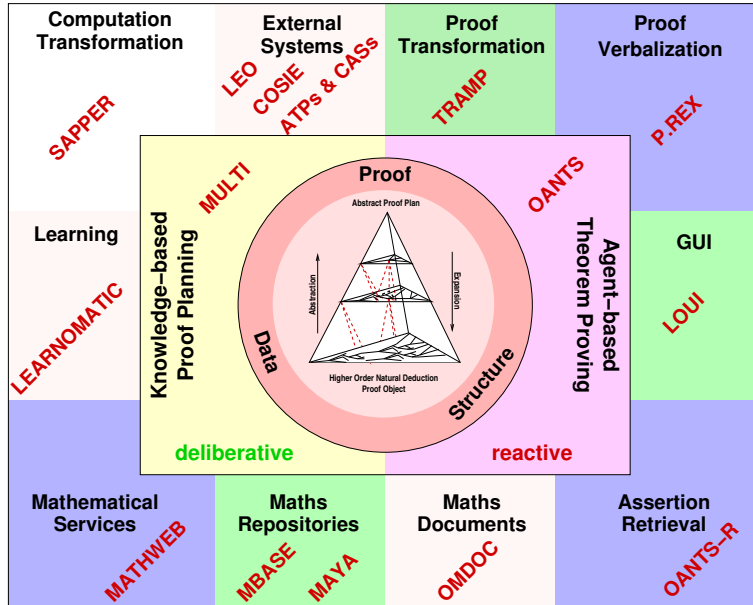
Figure 2. The vision of an all encompassing mathematical assistance environment: we have now modularized and out-sourced many of the support tools (whose names are printed in red) such that they can also be used by other systems via the MATH-WEB-SB software bus.

user community with sometimes several thousand theorems and lemmata being proven per day. Many theorems are generated automatically as (currently non-reusable and non-indexed) subproblems in natural language processing (see the Doris system [4]), proof planning and verification tasks.

## 2.2 Proof Objects

The central data structure for the overall search is the proof plan data structure $\mathcal{PDS}$ in Figure 1 and the subsystems cooperate to construct a proof whose status is stored again in the $\mathcal{PDS}$. The facilities provided by the subsystems include support for interactive and mixed-initiative theorem proving by the user, the proof planner, and by external systems such as automated theorem provers and computer algebra systems. These facilities require, in particular, the representation of proof steps at different levels of granularity ranging from abstract, human-oriented reasoning to logic-level justifications.

Therefore $\Omega$MEGA provides a hierarchical proof plan data structure that represents a (partial) proof at different levels of granularity (called partial proof plans). Technically, the $\mathcal{PDS}$ is a directed acyclic graph consisting of nodes, justifications and hierarchical edges (see [32] for more details). Each node rep-

---

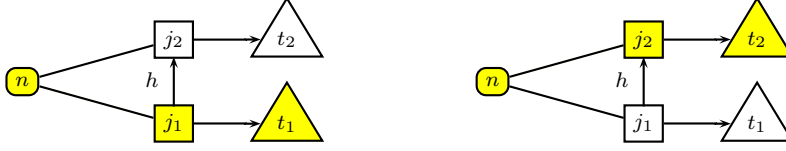[4] `www.cogsci.ed.ac.uk/~jbos/doris/`

Figure 3. Representation of a $\mathcal{PDS}$ node with justifications at different levels of granularity.

resents a sequent and can be *open* or *closed*. An open node corresponds to a sequent that is to be proved and a closed node to a sequent which is already proved or reduced to other sequents using an inference rule $R := \frac{A_1 \ldots A_k}{B}$; where $R$ may represent a calculus rule, a tactic, a method, or a call to an external system. Such a rule denotes that we can conclude $B$ from $A_1, \ldots, A_k$ or reading it the other way round that $B$ can be reduced to $A_1, \ldots A_k$. Thus, an inference step is represented by a justification $R$ which connects a node $n_b$ containing the sequent $B$ to nodes $n_1, \ldots, n_k$ containing the sequents $A_1, \ldots A_k$. If a node has more than one outgoing justification, each of them represents a proof attempt of the sequent stored in the source node, but at different granularity. These justifications are ordered with respect to their granularity using hierarchical edges. A hierarchical edge connects two justifications $j_1$ and $j_2$ with the meaning that justification $j_1$ represents a more detailed proof attempt than justification $j_2$. Thus, $\Omega$MEGA's $\mathcal{PDS}$ explicitly maintains the original proof plan as well as intermediate expansion layers in an expansion hierarchy.

Normally, the user wants to see the proof only at a specific level of granularity and therefore he can chose the granularity by selecting the justification for each node in the $\mathcal{PDS}$. Figure 3 shows an example of how the selection of a justification of a node determines the level of granularity. It shows a node $n$ with two outgoing justifications $j_1$ and $j_2$, which are connected by a hierarchical edge $h$ from $j_1$ to $j_2$ indicating that $j_1$ is a more granular justification than $j_2$. The user can decide whether he wants to see the more detailed version of the proof given by $j_1$ (and its subtree $t_1$) or the more abstract version given by $j_2$ (and its subtree $t_2$). The two different possible selections are shaded. Selecting the justifications for each node the user gets a view into the $\mathcal{PDS}$-graph, called a $\mathcal{PDS}$-view (see Figure 4), at the selected level of granularity.

Note that in contrast to the traditional LCF approach, it is not mandatory to immediately expand a high-level proof plan to a lower-level, because we explicitly represent the high-level proof plans in the $\mathcal{PDS}$ and thus conceptually separate plan formation from plan validation (by recursive expansion). Validation of proof plans can thus be postponed and executed at any time later on. In case of an unsuccessful expansion attempt, $\Omega$MEGA's $\mathcal{PDS}$ provides mechanisms which change the status of the affected proof nodes from *justified*, i.e. *closed*, to *open* and then consistently clean up all structures, which
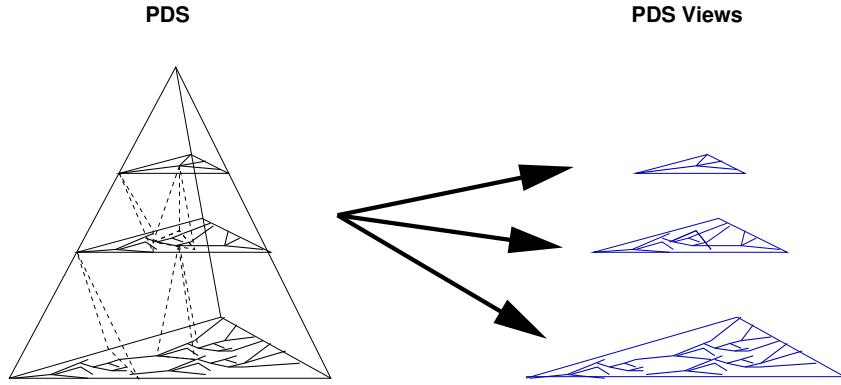
Figure 4. Possible views of proofs at different levels of granularity inside a $\mathcal{PDS}$

depend on these nodes. Thus, failing expansion may in particular introduce new gaps into a previously closed proof plan and hence proof planning has to start again in order to fill the gaps and search for a new plan.

Because the $\mathcal{PDS}$ represents the dependencies among goals and subgoals as well as between high-level inference rules and lower-level inference rules, we can traverse the datastructure in many ways for different purposes like visualization, proof explanation, natural language generation and dependency-directed pruning of the proof object.

In summary, coexistence of several granularity levels and the dynamical maintenance of their relationship is a central and distinguishing design objective of $\Omega$MEGA's $\mathcal{PDS}$. The $\mathcal{PDS}$ makes the hierarchical structure of proof plans explicit and retains it for further applications such as proof expansion, proof explanation with *P.rex* or an analogical transfer of plans.

Currently, however, we can not change the representation *inside* of a proof node, which is still something to be desired. For example, it would be nice to be able to change the logical propositions in naive set theory into Venn diagrams such that a diagrammatic reasoning system could be used. Support for representational shifts of this kind in combination with different levels of granularity is future work.

The proof object generated by $\Omega$MEGA for the theorem "$\sqrt{2}$ is irrational", which has a well known human proof of less than a dozen lines, is recorded in a technical report [14], where the unexpanded and the expanded proof objects are presented in great detail: The most abstract proof at the level of the proof plan has about twenty steps and the fully expanded proof has about 750. The final proof in natural language generated by the $\Omega$MEGA-system is shown in Figure 6. A general presentation of this interesting case study is [88].

7

ΩMEGA's main focus is on knowledge-based proof planning [25,26,74], where proofs are not conceived in terms of low-level calculus rules, but at a less detailed granularity, i.e. a more abstract level, that highlights the main ideas and de-emphasizes minor logical or mathematical manipulations on formulae.

Knowledge-based proof planning is a paradigm in automated theorem proving, which swings its motivational pendulum back to the AI origins in that it employs and further develops many AI principles and techniques such as hierarchical planning, knowledge representation in frames and control rules, constraint solving, tactical theorem proving, and meta-level reasoning. It differs from traditional search based techniques in automated theorem proving not least in its level of granularity: The proof of a theorem is planned at an abstract-level where an outline of the proof is found first. This outline, that is, the abstract proof plan, can be recursively expanded to construct a proof within a logical calculus provided the expansion of the proof plan does not fail. The plan operators, called *methods*, represent mathematical techniques familiar to a working mathematician. While the knowledge of a mathematical domain as represented by methods and control rules is specific to the mathematical field, the representational techniques and reasoning procedures are general-purpose. For example, one of our first case studies [74] used the limit theorems proposed by Woody Bledsoe [23] as a challenge to automated reasoning systems. The general-purpose planner makes use of the mathematical domain knowledge of $\epsilon$-$\delta$-proofs and of the guidance provided by declaratively represented control rules, which correspond to mathematical intuition about how to prove a theorem in a particular situation. These rules are the basis for our meta-level reasoning and the goal-directed behavior.

Domain knowledge is encoded into methods, control rules, and strategies. Moreover, methods and control rules can employ external systems (e.g., one method is to call one of the computer algebra systems) and make use of the knowledge in these systems. ΩMEGA's multi-strategy proof planner MULTI [73,64] searches then for a plan using the acquired methods and strategies guided by the control knowledge in the control rules.

### 2.3.1   AI Principles in Proof Planning

A *planning problem* is a formal description of an *initial state*, a *goal*, and some *operators* that can be used to transform the initial state via some intermediate states to a state that satisfies the goal. Applied to a planning problem, a *planner* returns a sequence of *actions*, that is, instantiated operators (i.e. methods), which reach a goal state from the initial state when executed. Such a

sequence of actions is called a *solution plan*.

Proof planning considers mathematical theorems as planning problems [25]. The initial state of a proof planning problem consists of the proof *assumptions* of the theorem, whereas the goal is the *theorem* itself. The operators in proof planning are the methods, traditionally they are tactics augmented by pre- and postconditions.

In $\Omega$MEGA, proof planning is the process that computes actions, that is, instantiations of methods, and assembles them sequentially in order to derive a theorem from a set of assumptions. The effects and the preconditions of an action in proof planning are formulae in the higher-order language $\mathcal{POST}$, where the effects are considered as logically inferable from the preconditions using this method. A proof plan under construction is represented in the proof plan data structure $\mathcal{PDS}$, which consists initially of an open node containing the conjecture to be proven, and closed, i.e. justified nodes for the proof assumptions. The introduction of a method changes the $\mathcal{PDS}$ by adding new proof nodes and justifying the effects of the method by applications of the method to its premises. The aim of the proof planning process is to reach a *closed* $\mathcal{PDS}$, that is, a $\mathcal{PDS}$ without open nodes. The *solution proof plan* produced is then a record of the sequence of actions that lead to a closed $\mathcal{PDS}$.

By allowing for forward and backward methods, $\Omega$MEGA's proof planner MULTI combines forward and backward state-space planning. Thus, a *planning state* in MULTI is a pair of the current world state and the current goal state. The initial world state consists of the given proof assumptions and is transferred by forward methods into a new world state. The goal state consists of the initial open node and is transferred by backward methods into a new goal state containing new open nodes. From this point of view the aim of proof planning is to compute a sequence of actions that derives a current world state in which all the goals are satisfied.

As opposed to precondition achievement planning (e.g., see [96]), effects of methods in proof planning do not cancel each other. For instance, a method with effect $\neg F$ introduced for an open node $L_1$ does not threaten the effect $F$ introduced by another method for an open node $L_2$. Dependencies among open nodes result from shared variables for witness terms and their constraints. Constraints can, for instance, be instantiations for the variables but they can also be mathematical constraints such as $x < c$, which states that, whatever the instantiation for $x$ is, it has to be smaller than $c$. The constraints created during the proof planning process are collected in the constraint store of the $\mathcal{CoSIE}$ system [76,100], which is a domain-independent extension of existing propagation-based constraint solvers. The extension turned out to be necessary, since proof planning has peculiar requirements that are not met by off-

the-shelf constraint solvers: $\mathcal{CoSIE}$ computes symbolic constraint inferences while respecting the logical side-conditions of proof planning, for instance, the Eigenvariable condition and the logical dependencies between constraints and their context. The search procedure of $\mathcal{CoSIE}$ computes logically correct instantiations for the meta-variables.

A proof-planning method is applicable only if its constraints are consistent with the constraints collected so far. Dependencies among goals with shared variables are difficult to analyze and can cause various kinds of failures in a proof planning attempt (see [63] for more details).

### 2.3.2  Methods, Control Rules, and Strategies

*Methods* are traditionally perceived as tactics in tactical theorem proving [78] augmented with preconditions and effects, called *premises* and *conclusions*, respectively. A method represents a large inference of the conclusion from the premises based on the body of the tactic. For instance, NotI-m is a (very low-level) method whose purpose is to prove a goal $\Gamma \vdash \neg P$ by contradiction. If NotI-m is applied to a goal $\Gamma \vdash \neg P$ then it closes this goal and introduces the new goal to prove falsity, $\bot$, under the assumption $P$, that is, $\Gamma, P \vdash \bot$. Thereby, $\Gamma \vdash \neg P$ is the conclusion of the method, whereas $\Gamma, P \vdash \bot$ is the premise of the method. NotI-m is a *backward* method, which reduces a goal (the conclusion) to new goals (the premises). *Forward* methods, in contrast, derive new conclusions from given premises. For instance, =Subst-m performs equality substitutions, for example, by deriving from the two premises $\Gamma \vdash P[a]$ and $\Gamma \vdash a = b$ the conclusion $\Gamma \vdash P[b]$ where an occurrence of $a$ is replaced by an occurrence of $b$. Note that NotI-m and =Subst-m are simple examples of domain-independent, logic-related methods, which are needed in addition to domain-specific, mathematically motivated methods as illustrated below in Section 2.3.3. Knowledge-based proof planning expands on these ideas and allows for more general mathematical methods to be encapsulated into the proof planning *methods*.

*Control rules* represent mathematical knowledge about how to proceed in the proof planning process. They can influence the planner's behavior at choice points (e.g., which goal to tackle next or which method to apply next) by preferring members of the corresponding list of alternatives (e.g., the list of possible goals or the list of possible methods). This way promising search paths are preferred and the search space can be pruned.

*Strategies* employ a fixed set of methods and control rules and, thus, tackle a theorem by some mathematical standard that happens to be typical for this theorem. The reasoning as to which strategy to employ on a problem is an explicit choice point in MULTI. In particular, MULTI can backtrack from a

chosen strategy and commence search with different strategies.

Detailed discussions of $\Omega$MEGA's method and control rule language can be found in [63,65]. A detailed introduction to proof planning with multiple strategies is given in [73,64] and more recently in [69]. In the following we briefly sketch how proof planning with generic and domain specific methods along with domain specific control strategies can be applied to plan "irrationality of $\sqrt[j]{l}$"-conjectures for arbitrary natural numbers $j$ and $l$ (see also [88]).

### 2.3.3 Exploiting Domain Specific Knowledge: Proof Planning $\sqrt[j]{l}$-Problems

$\Omega$MEGA can successfully proof plan and proof/disprove the irrationality of $\sqrt[j]{l}$ for arbitrary natural numbers $j$ and $l$. In order to find a general approach to tackle these problems, we first showed the challenge problem "$\sqrt{2}$ is irrational" (see [97]) and then analyzed proofs for statements such as $\sqrt{8}$, $\sqrt{(3 \cdot 3) - 1}$, or $\sqrt[3]{2}$. We found that some of the concepts and inference steps we used for $\sqrt{2}$ are particular to this problem and do not generalize whereas others do. Thus, the analysis led to some generalized concepts, theorems, and proof steps, which we encoded into methods and control rules, which together form one *planner strategy* for this kind of problems. We shall now discuss the acquired methods and control rules.

The essential idea of the proofs is as follows:

(1) Use the MBASE-theorem `RAT-CRITERION` (it states that for each rational number $x$, there are integers $y$ and $z$, such that $x \cdot y = z$, where $y$ and $z$ have no common divisor besides 1) and construct an indirect proof.

(2) In order to derive the contradiction show that the two witnesses (i.e. the existential variables $y$ and $z$) in `RAT-CRITERION`, which are supposed to have no common divisor, actually do have a common divisor $d$.

(3) In order to find a common divisor transform equations (for example, $\sqrt{2} \cdot n = m \longrightarrow 2 \cdot n^2 = m^2$), derive new divisor statements (for example, from $2 \cdot n^2 = m^2$ derive that $m^2$ has divisor 2, or from the statement that $m^2$ has divisor 2 derive that $m$ has divisor 2), and derive from given divisor statements new representations of terms, which can be used again for equational transformations (for example, from the statement that $m$ has divisor 2 derive that $m = 2 \cdot k$ for some $k$).

Note that we are particularly interested in prime divisors, since only for prime numbers $d$ is it true that if $d$ is a divisor of $m^j$ then $d$ is also a divisor of $m$. A corresponding theorem is available in $\Omega$MEGA's knowledge base MBASE.

To realize the first idea (1), the planner MULTI has to decide for an indirect

proof, apply the theorem `RAT-CRITERION`, and derive $l \cdot n^j = m^j$ for integers $m$ and $n$, which are supposed to have no common divisor. These steps are canonical for arbitrary $\sqrt[j]{l}$ problems. Hence, we could implement them all into one method. However, to avoid the well known problem of over-fitting methods, i.e. to make them special just for a particular theorem, we decided to employ already existing methods from other domains: NotI-m (contradiction of negated statements), MAssertion-m (apply a theorem or an axiom from the theory), ExistsE-Sort-m (decompose existentially quantified formulae), AndE-m (decompose conjunctions).

The application of the methods ExistsE-Sort-m, AndE-m, and NotI-m do not need any further control, but the application of MAssertion-m has to be guided by selecting the theorem or axiom to be applied. This is achieved by a control rule `apply-ratcriterion`, which determines that the theorem `RAT-CRITERION` should be used for MAssertion-m, whenever there is a goal formula $\sqrt[j]{l}$.

The second idea (2) is realized with the method ContradictionCommonDivisor-m. When MULTI tries to apply the method it searches first for an assumption stating that two terms $t_1, t_2$ have no common divisor, and then it searches for two (derived) assumptions stating that $t_1$ and $t_2$ both have a divisor $d$. This method is not guided by control rules, but MULTI tries to apply it to some derived assumptions in each planning cycle.

The third idea (3) of the proof technique is encoded into several collaborating methods: TransFormEquation-m, =Subst-m, PrimeFacsProduct-m, PrimeDivPower-m, and CollectDivs-m. The method TransFormEquation-m contains knowledge about suitable equational transformations for our problem domain. It is applied to an equation and derives a new equation. For instance, TransFormEquation-m derives $l \cdot n^j = m^j$ from $\sqrt[j]{l} \cdot n = m$, or it derives $n^2 = 2 \cdot k^2$ from $2 \cdot n^2 = (2 \cdot k)^2$. The method =Subst-m performs equality substitutions.

PrimeFacsProduct-m and PrimeDivPower-m encapsulate the knowledge of how to derive divisor statements. PrimeFacsProduct-m is applied to equations $x = l \cdot y$ (or $l \cdot y = x$) and derives a new assumption which is a conjunction of statements that $x$ has particular prime divisors. The method employs MAPLE to compute the prime divisors of $l$ using MAPLE's function `with(numtheory, factorset)`. It derives that $x$ has to have all prime divisors of $l$. For instance, from $2 \cdot n^2 = m^2$ PrimeFacsProduct-m derives that $m^2$ has the prime divisor 2, from $6 \cdot n^2 = m^2$ it derives that $m^2$ has the prime divisors 2 and 3. PrimeDivPower-m is applied to an assumption that states that $y^j$ has prime divisor $d$ and derives that $y$ has prime divisor $d$.

For a term $t$ CollectDivs-m searches for assumptions stating that $t$ has some prime divisors. Then, it computes different possible representations of $t$ based on the set of the prime divisors $\{p_1, \ldots, p_n\}$. That is, for each subset $\{p'_1, \ldots,$

$p'_{n'}\}$ of $\{p_1, \ldots, p_n\}$ it adds a new assumption $t = p'_1 \cdot \ldots \cdot p'_{n'} \cdot c'$ for some integer $c'$.

TransFormEquation-m, PrimeFacsProduct-m and PrimeDivPower-m are applied whenever possible and no guidance is required. The application of the method CollectDivs-m, however, is guided by the control rule `apply-collectdivs`, which prefers CollectDivs-m with respect to a term $t$ as soon as there are assumpitions stating that $t$ has some prime divisors. The application of =Subst-m is guided by the control rule `apply-=subst`, which states that, after an application of CollectDivs-m, the method =Subst-m should be applied in order to use the equations resulting from CollectDivs-m. When a method such as =Subst-m, PrimeFacsProduct-m, or PrimeDivPower-m is applied to some premises, then the same method is afterwards applicable again to the same premises, deriving the same result. To avoid endless loops of such methods, we added the control rule `reject-loop`, which blocks the repeated application of a forward method to the same premises.

### 2.4  ΩANTS: *Agent-oriented Theorem Proving*

ΩANTS has originally been developed to support interactive theorem proving [18] and later its was extended to a fully automated reasoning system [19,92]. The basic idea of ΩANTS is to encapsulate each inference rule into a pro-active agent, which checks automatically for its own applicability. For each proof situation the $\mathcal{PDS}$ is continously checked by these agents and thus composes a ranked list of potentially applicable inference rules. In this process all calculus rules, tactics, external system calls and methods, collectively called *inference rules*, are uniformly viewed with respect to three sets: premises, conclusions, and additional parameters. The elements of these three sets are called *arguments* of the inference rule and they usually depend on each other. An inference rule is applicable if at least some of its arguments can be instantiated with respect to the given proof context. The task of the ΩANTS-system is now to determine the applicability of inference rules by computing instantiations for their arguments.

The ΩANTS-architecture consists of two layers. On the bottom layer, possible instantiations of the arguments of individual inference rules are computed. In particular, each inference rule is associated with a blackboard and some concurrent processes, one for each argument of the inference rule. The role of every process is to compute possible instantiations for its designated argument of the inference rule, and to record these on the blackboard. The computation is carried out with respect to the given proof context and exploits the information already present on the blackboard, that is, argument instantiations computed by other processes. On the upper layer, the information from

the lower layer is used for computing and heuristically ranking the inference rules that are applicable in the current proof state. The heuristically most promising rule is then applied to the central proof object and the data on the blackboards is cleared for the next round of computation.

$\Omega$Ants uses resource reasoning to guide the search [22]. The integration of external reasoning systems such as automated theorem provers, computer algebra systems, or model generators into the architecture of $\Omega$Ants presupposes the declaration of some resource limits these reasoning agents are allowed to spend (e.g., by specifying time-outs). The external systems are encapsulated into inference rules, usually one for each system. For example, an inference rule modeling the application of an ATP has its conclusion argument set as "open goal". A process can then place this open goal onto the blackboard, where it is picked up by a process that applies the prover to it. Any computed proof or partial proof from the external system is again written onto the blackboard from where it is subsequently inserted into the $\mathcal{PDS}$ when the inference rule is applied. While this setup enables proof construction by a collaborative effort of diverse reasoning systems, the cooperation is achieved via the central $\mathcal{PDS}$. This means that all partial results have to be translated back and forth between the syntaxes of the integrated systems and the representation language of the $\mathcal{PDS}$. In some cases efficient communication between inference systems is difficult to achieve [15]. Therefore we have recently developed an alternative model of cooperating systems in $\Omega$Ants which has been successfully applied to the combination of automated higher-order and first-order theorem provers [20].

*2.5 External Systems*

Proof problems require many different skills for their solution and it is desirable to have access to several systems with complementary capabilities, to orchestrate their use, and to integrate their results. $\Omega$mega interfaces heterogeneous external systems such as *computer algebra systems (CASs)*, higher- and first-order *automated theorem proving systems (ATPs)*, *constraint solvers (CSs)*, and *model generation systems (MGs)*.

Their use is twofold: they may provide a solution to a subproblem, or they may give hints for the control of the search for a proof. In the former case, the output of an incorporated reasoning system is translated and inserted as a subproof into the $\mathcal{PDS}$. This is beneficial for interfacing systems that operate at different levels of granularity, and also for a human-oriented display and inspection of a partial proof. In particular we can now check the soundness of each contribution by expanding the inserted subproof to a basic logic-level proof in the $\mathcal{PDS}$ and then verify it by $\Omega$mega's proof checker.

Currently, the following external systems are integrated and used in ΩMEGA:

**CASs** provide symbolic computation, which can be used in two ways: first, to compute hints to guide the proof search (e.g., witnesses for existential variables), and, second, to perform some complex algebraic computation such as to normalize or simplify terms. In the latter case the symbolic computation is directly translated into proof steps in ΩMEGA. CASs are integrated via the transformation and translation module SAPPER [91]. Currently, ΩMEGA uses the systems MAPLE [30] and GAP [85].

**ATPs** are employed to solve subgoals. Currently ΩMEGA uses the first-order provers BLIKSEM [38], EQP [60], OTTER [61], PROTEIN [10], SPASS [95], WALDMEISTER [50], the higher-order systems TPS [2], and $\mathcal{LEO}$ [16,11], and we plan to incorporate VAMPIRE [82]. The first-order ATPs are connected via TRAMP [62], which is a proof transformation system that transforms resolution-style proofs into assertion-level ND-proofs which can then be integrated into ΩMEGA's $\mathcal{PDS}$. TPS already provides ND-proofs, which can be further processed and checked with little transformational effort [12].

**MGs** provide either witnesses for free (existential) variables, or counter-models, which show that some subgoal is not a theorem. Hence, they help to guide the proof search. Currently, ΩMEGA uses the model generators SATCHMO [58] and SEM [98].

**CSs** construct mathematical objects with theory-specific properties as witnesses for free (existential) variables. Moreover, a constraint solver can help to reduce the proof search by checking for inconsistencies of constraints. Currently, ΩMEGA employs $\mathcal{CoSIE}$ [76,100], a constraint solver for inequalities and equations over the field of real numbers.

*2.6   Interface and System Support*

ΩMEGA's graphical user interface $\mathcal{LΩUI}$ [90] displays the current $\mathcal{PDS}$ in multiple modalities: a graphical map of the proof tree, a linearized presentation of the proof nodes with their formulae and justifications, a term browser, and a natural language presentation of the proof via *P.rex* (see Figures 5 and 6).

When inspecting a part of a proof, the user can switch between alternative levels of granularity coexisting in the $\mathcal{PDS}$, for example, by expanding an abstract justification of a proof node into its associated, less abstract partial subproof, which causes appropriate changes in the other presentation modes. Moreover, an interactive natural language *explanation* of the proof is provided by the system *P.rex* [40,39,41], which is adaptive in the following sense: it explains a proof step at the most abstract level (which the user is assumed to know) and then reacts flexibly to questions and requests, possibly at a lower level of granularity, for example, by detailing some ill-understood subproof.
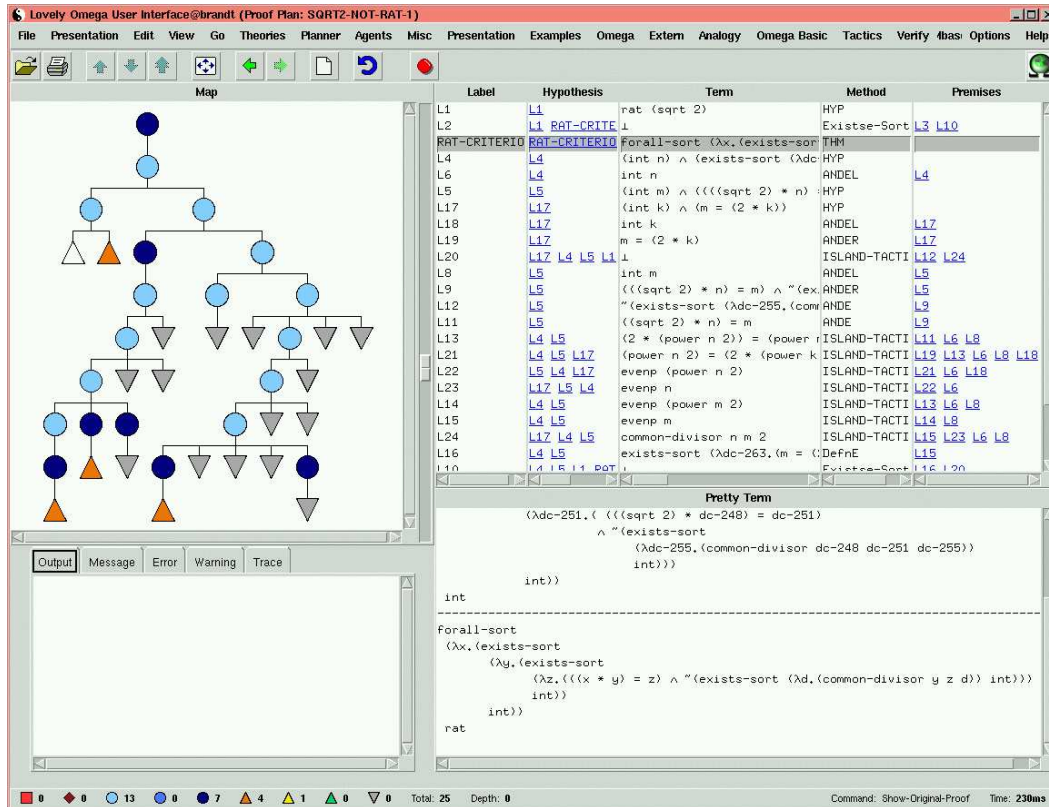
Figure 5. Multi-modal proof presentation in the graphical user interface $\mathcal{L\Omega UI}$.

Another system support is the guidance mechanism provided by the suggestion module $\Omega$Ants (see Section 2.4), which searches pro-actively for possible actions that may be helpful in finding a proof and presents them in a preference list.

### 2.7   Case Studies

Early developments of proof planning in Alan Bundy's group at Edinburgh used proofs by induction as their favorite case studies [25]. The $\Omega$mega system has been used in several other case studies, which illustrate in particular the interplay of the various components, such as proof planning supported by heterogeneous external reasoning systems.

A typical example for a class of problems that cannot be solved by traditional automated theorem provers is the class of $\epsilon$–$\delta$–proofs [74,71]. This class was originally proposed by Woody Bledsoe [23] as a challenge and it comprises theorems such as LIM+ and LIM*, where LIM+ states that the limit of the sum of two functions equals the sum of their limits and LIM* makes the corresponding statement for multiplication. The difficulty of this domain arises
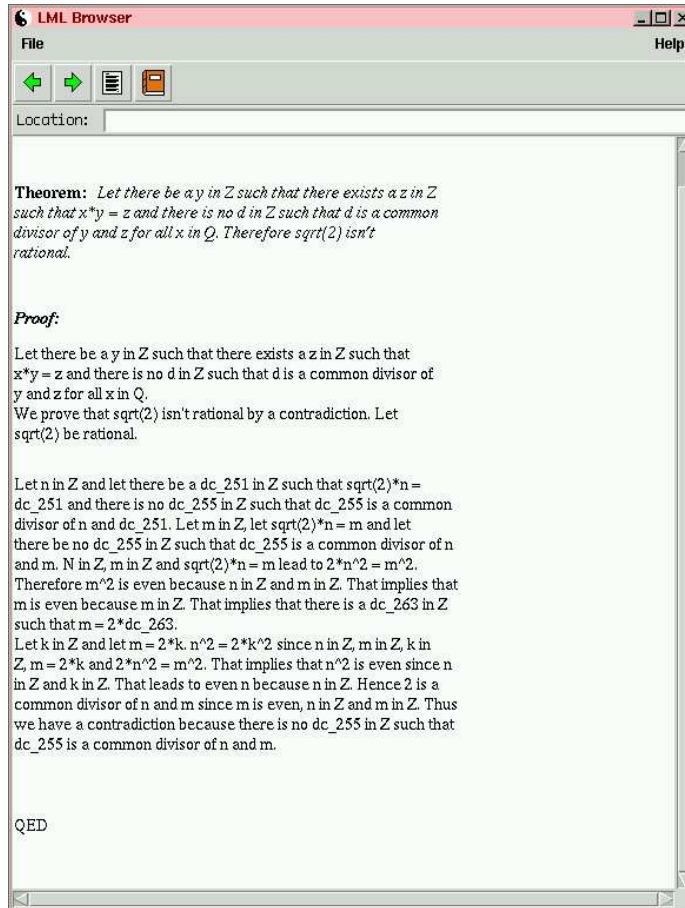
16

Figure 6. Natural language proof presentation by *P. rex* in $\mathcal{L\Omega UI}$.

from the need for arithmetic computation in order to find a suitable instantiation of free (existential) variables (such as a $\delta$ depending on an $\epsilon$). Crucial for the success of $\Omega$MEGA's proof planning is the integration of suitable experts for these tasks: the arithmetic computation is done by the computer algebra system MAPLE, and an appropriate instantiation for $\delta$ is computed by the constraint solver $\mathcal{CoSIE}$. We have been able to solve all challenge problems suggested by Bledsoe and many more theorems in this class taken from a standard textbook on real analysis [9].

Another class of problems we tackled with proof planning is concerned with residue classes [67,66]. In this domain we showed theorems such as: "the residue class structure $(\mathbb{Z}_5, \bar{+})$ is associative", "it has a unit element", and similar properties, where $\mathbb{Z}_5$ is the set of all congruence classes modulo 5 (i.e. $\{\bar{0}_5, \bar{1}_5, \bar{2}_5, \bar{3}_5, \bar{4}_5\}$) and $\bar{+}$ is the addition on residue classes. We have also investigated whether two given structures are isomorphic or not and altogether we have proved more than 10,000 theorems of this kind (see [92]). Although the problems in this domain are not too difficult and still within the success range of a traditional automated theorem prover, it was nevertheless an interesting

17

case study for proof planning, since multi-strategy proof planning generated substantially different proofs based on entirely different proof ideas.

Another important proof technique is Cantor's diagonalization technique and we also developed methods and strategies for this class [31]. Important theorems we have been able to prove are the undecidability of the halting problem and Cantor's theorem (cardinality of the set of subsets), the non-countability of the reals in the interval $[0, 1]$ and of the set of total functions, and similar theorems.

Finally, a good example for a standard proof technique is the excess-literal-number technique. This is routinely used for completeness proofs of refinements of resolution, where the theorem is usually first shown at the ground level using the excess-literal-number technique and then ground completeness is lifted to the predicate calculus level. We have done this for many refinements of resolution with $\Omega$MEGA [45].

However, $\Omega$MEGA's main aim is to become a proof assistant for the working mathematician. Hence, it should support interactive proof development at a human-oriented level of granularity. The already mentioned theorem that $\sqrt{2}$ is irrational, and its well-known proof dating back to the School of Pythagoras, provides an excellent challenge to evaluate whether this ambitious goal has been reached. In [97] seventeen systems that have solved the $\sqrt{2}$-problem show their results. The protocols of their respective sessions have been compared on a multi-dimensional scale in order to assess the "naturalness" by which real mathematical problems of this kind can be shown. This represents an important shift of emphasis in the field of automated deduction away from the somehow artificial problems of the past – as represented, for example, in the test set of the TPTP library [93] – back to real mathematical challenges. We participated in this case study essentially with three different contributions. Our initial contribution was an interactive proof in $\Omega$MEGA without adding any special domain knowledge to the system. This demonstrates the use of $\Omega$MEGA as a tactical theorem prover (see [14]). The most important albeit not entirely new lesson to be learned from this experiment is that the level of granularity common in most automated and tactical theorem proving environments is far too low. While our proof representation in this first study is already an abstraction (called the *assertion level* in [51]) from the calculus level typical for most ATPs, it is nevertheless clear that as long as a system does not hide all these excruciating details, no working mathematician will feel inclined to use such a system. In fact, this is in our opinion one of the critical impediments for using first-order ATPs and one, albeit not the only one, of the reasons why they are not used as widely as computer algebra systems. This is the crucial issue of the $\Omega$MEGA project and our main motivation for departing from the classical paradigm of automated theorem proving about fifteen years ago.

Our second contribution to the case study of the $\sqrt{2}$-problem is based on interactive *island planning* [70], a technique that expects an outline of the proof, i.e. the user provides main subgoals, called *islands*, together with their assumptions. In fact, we are able to proof plan arbitrary $\sqrt[j]{l}$-problems as sketched in Section 2.3.3. Hence, the user can write down *his* proof idea in a natural way with as many gaps as there are open at this first stage of the proof. Closing the gaps is ideally fully automatic, in particular, by exploiting external systems. However, for difficult theorems it is necessary more often than not that the user provides additional information and applies the island approach recursively. In comparison to our first tactic-based solution the island style supports a much more abstract and user-friendly interaction level. The proofs are now at a level of granularity similar to proofs in mathematical textbooks.

Our third contribution to the case study of the $\sqrt{2}$-problem are fully automatically planned and expanded proofs of $\sqrt[j]{l}$-problems for arbitrary natural numbers $j$ and $l$. The details of this important case study, that shows best what can (and what cannot) be achieved with current proof planning technology are presented in [88], [89], and [14].

## 2.8 Discussion

### 2.8.1 Proof-Planning as an Alternative Approach to Automated Theorem Proving?

The most important question to ask here is: Can we find the essential and creative steps automatically, for example, for the $\sqrt{2}$-problem discussed in Section 2.3.3? The answer is yes, as we have shown in [88]. However, while we can answer the question in the affirmative, not every reader may be convinced, as our solution touches upon a subtle point, which opens the Pandora Box of critical issues in the paradigm of proof planning [28]: It is always easy to write some specific methods, which perform just the steps in the interactively found proof and then calls the proof planner MULTI to fit the methods together into a proof plan for the given problem. This, of course, shows nothing of substance: Just as we could write down all the definitions and theorems required and sufficient for the problem in first-order predicate logic and then hand them to a first-order prover [5], we would just hand-code the final solution into appropriate methods.

Instead, the goal of the game is to find *general* methods for a whole class of theorems within some theory that can solve not only this particular problem,

---

[5] This was done when OTTER tackled the $\sqrt{2}$-problem; see [97] for the original OTTER case study and [14] for its replay with ΩMEGA.

but also all the other theorems in that class. While our approach essentially follows the proof idea of the interactively constructed proof for the $\sqrt{2}$-problem, it relies essentially on more general concepts.

However, this is certainly not the end of the story. In order to evaluate the appropriateness of a proof planning approach we suggest the following four criteria:

(1) How general and how rich in mathematical content are the methods and control rules?
(2) How much search is involved in the proof planning process?
(3) What kind of proof plans, that is, what kind of proofs, can we find?
(4) If the proof planning procedure fails on some given conjecture, how likely is it that the given conjecture is not a theorem?

These criteria should allow us to judge how general and how robust our solution is. The art of proof planning is to acquire domain knowledge that, on the one hand, comprises meaningful mathematical techniques and powerful heuristic guidance, and, on the other hand, is general enough to tackle a broad class of problems. For instance, as one extreme, we could have methods that encode ΩMEGA's ND-calculus and we could run MULTI without any control. This approach would certainly be very general, but MULTI would fail to prove any interesting problems. As the other extreme, we could cut a known proof into pieces, and code the pieces as methods. Guided by control rules that always pick the next right piece of the proof, MULTI would assemble the methods again to the original proof without performing any search. However, in that case if MULTI fails to find a proof then it is not unlikely that the conjecture is nevertheless a theorem.

### 2.8.2 What lessons have we learned?

The problem domains on which proof planning has been applied so far are small but nevertheless typical. Some interesting observations gained from this experience are the following:

(1) The devil is in the detail, that is, it is always possible to hide the crucial creative step (represented as a specific method or represented in the object language by an appropriate lemma) and to pretend a level of generality that has not actually been achieved. To evaluate a solution *all* tactics, methods, theorems, lemmata and definitions have to be made explicit.
(2) The enormous distance between the well-known (top-level) proof of the Pythagorean School, which consists of about a dozen proof steps in comparison to the final (non-optimized) proof at ΩMEGA's ND-calculus level with about 750 inference steps is striking. This is, of course, not a new

insight. While mathematics can *in principle* be reduced to purely formal logic-level reasoning as demonstrated by Russell and Whitehead as well as the Hilbert School, nobody would actually want to do so *in practice* as the Bourbaki group of French mathematicians states explicitly: The first quarter of the first volume in the several dozen volume set on the foundation of mathematics starts with elementary, logic-level reasoning and then proceeds with the crucial sentence [24]: "No great experience is necessary to perceive that such a project [of complete formalization] is absolutely unrealizable: the tiniest proof at the beginning of the theory of sets would already require several hundreds of signs for its complete formalization."

(3) Finally and more to the general point of interest in mathematical support systems: Now that we can prove theorems in the $\sqrt[j]{l}$-problem class, the skeptical reader may still ask: *So what?* Will this ever lead to a *general* system for mathematical proof assistance?

We have shown that the class of $\epsilon$-$\delta$-proofs for limit theorems can indeed be solved with a few dozen mathematically meaningful methods and control rules (see [74,72,63]). Similarly, the domain of group theory with its class of residue theorems can be formalized with even fewer methods (see [68,66,67]). [6] An interesting observation is also that these methods by and large correspond to the kind of mathematical knowledge a freshman would have to learn to master this level of professionalism.

Do the above observations now hold for our $\sqrt[j]{l}$-problems? The unfortunate answer is probably *No!* Imagine the subcommittee of the United Nations in charge of the maintenance of the global mathematical knowledge base in a hundred years from now. Would they accept the entry of our methods, tactics and control rules for the $\sqrt[j]{l}$-problems? Probably not!

Factual mathematical knowledge is preserved in books and monographs, *but the art of doing mathematics* [81,49] is passed on by word of mouth from generation to generation. The methods and control rules of the proof planner correspond to important mathematical techniques and "ways to solve it" [81], and they make this implicit and informal mathematical knowledge explicit and formal.

The theorems about $\sqrt[j]{l}$-problems are shown by contradiction, that is, the planner derives a contradiction from the equation $l \cdot n^j = m^j$, where $n$ and $m$ are integers with no common divisor. However, these problems belong to the more general class to determine whether two complex mathematical objects $\mathcal{X}$ and $\mathcal{Y}$ are equal. A general mathematical principle for comparison of two

---

[6] The generally important observation is not, of course, whether we need a dozen or a hundred methods, but that we don't need a few thousand or a million. A few dozen methods seem to be generally enough for a restricted mathematical domain.

complex objects is to look at their characteristic properties, for example, their normal forms or some other uniform notation in the respective theory.

And this is the crux of the matter: to find general mathematical principles and encode them into appropriate methods, control rules and strategies such that an appropriately large *class of problems* can be solved with these methods.

## 3   The Future: What next?

The longterm goal of the $\Omega$MEGA project is an integrated environment of tools supporting a wide range of typical mathematical activities. Examples of mathematical activities are computing, proving, solving, modeling, verifying, structuring, searching, inventing, publishing, explaining, illustrating, etc. We anticipate that in the long run assistance systems for mathematics will change mathematical practice and they will have a strong societal impact, not least in the sense that a powerful infrastructure for mathematical research and education will become commercially available. Computer supported mathematical reasoning tools and integrated assistance systems will be further specialized to have a strong impact also in many other theoretical fields such as safety and security verification of computer software and hardware, theoretical physics and chemistry and other related subjects.

The research questions we plan to investigate in the immediate future arise from the following scenario of preparing a mathematical research article with formalized content in a textbook style and in professional type-setting quality.

*Mathematical Research Article Preparation Scenario.* The author starts writing a new mathematical document in a format suitable for publication by using mathematical concepts from different mathematical domains. New mathematical concepts or lemmata introduced in the paper should result in corresponding new formal objects. Furthermore, when writing the document appropriate service tools can be used to compute intermediate results for an illustrating example, querying mathematical databases for mathematical publications introducing similar concepts and send subproblems to be solved to special reasoning or computation systems. Proofs of lemmata and theorems contained in the document should be amenable to formal proof checking techniques such that the submitted paper can be proof checked semi-automatically by the journal. A long-term goal may be fully automated verification.

## 3.1 Formalization and Proving at a Higher Level of Granularity

Mathematical reasoning with the $\Omega$MEGA system is at the comparatively high level of the proof planning methods. However, as these methods have to be expanded eventually to our base-level ND-calculus, the system still suffers from the effect and influence this logical representation has. In contrast, the proofs developed by a mathematician, say for a mathematical publication, and the proofs developed by a student in a mathematical tutoring system are typically developed at a less fine-grained argumentative level. This level has been formally categorized as *proofs at the assertion level* [51]. While so far assertion level proofs needed to be constructed from the underlying ND-calculus proof in $\Omega$MEGA, the recently developed CoRE system [3,4] supports proof construction directly on the assertion level and defines a communication infrastructure, i.e. a mediator, between the user and the automatic reasoning procedures. Currently, we exchange $\Omega$MEGA's ND-calculus by the CoRE calculus, which supports the presentation of the proof state via relevant contextual information about possible proof continuations and also supports hierarchical proof development. The proof theory of CoRE is uniform for a variety of logics and exploits proof-theoretic annotations in formulas for an assertion-level contextual reasoning style.

An unfortunate aspect of typical mathematical proofs is their *under-specification*,[7] for example, missing references to premise assertions, to rule and instantiation specifications or simply the specific part of the formula the author is talking about. One particular challenge here is to define an appropriate proof format which allows to represent human-constructed proofs as they are and to develop means to resolve the under-specification later by deductive methods. First steps in that direction and a description of the types of under-specifications can be found in [5,13].

## 3.2 Mathematical Knowledge Representation

A mathematical proof assistance system relies upon different kinds of knowledge: First, of course, the formalized mathematical domain as organized in structured theories of definitions, lemmata, and theorems. Secondly, there is mathematical knowledge on how to prove a theorem, which is encoded in tactics and methods, in $\Omega$ANTS agents, in control knowledge and in strategies.

---

[7]  "Under-specification" is a technical term borrowed from research on the semantics of natural language. Roughly it means that certain aspects in the semantic representation of a natural language utterance are left uninterpreted, such that their proper treatment can be deferred to later stages of processing in which more contextual information is available.

This type of knowledge can be general, theory specific or even problem specific.

The integration of a mathematical proof assistant into the typical and everyday activities of a mathematician requires, however, other types of knowledge as well. For example, a tutoring system for maths students may rely upon a database with different samples of proofs and proof plans linked by meta-data in order to advise the student. Another example is the support for mathematical publications: The documents containing both formalized and non-formalized parts need to be related to specific theories, lemmata, theorems, and proofs. This raises the research challenge on how the usual structuring mechanisms for mathematical theories (such as theory hierarchies or the import of theories via renaming or general morphisms) can be extended to tactics and methods as well as to proofs, proof plans and mathematical documents. Furthermore, changing any of these elements requires maintenance support as any change in one part may have consequences in other parts. For example, the validity of a proof needs to be checked again after changing parts of a theory, which in turn may affect the validity of the mathematical documents. Thus, technology supporting the *management of change* [7,8,6,52,77], originally developed for evolutionary formal software engineering at the DFKI[8], will now be integrated into the $\Omega$MEGA system as well.

Hierarchically structured mathematical knowledge, i.e. an ontology of mathematical theories and assertions has initially been stored in $\Omega$MEGAs hardwired mathematical knowledge base. This mathematical knowledge base was later (end of the 90s) out-sourced and linked to the development of MBASE [43]. We now assume that a mathematical knowledge base also maintains domain specific control rules, strategies, and linguistic knowledge. While this is not directly a subject of research in the $\Omega$MEGA project, relying here on other groups of the MKM community and especially the OMDOC format[9], we shall nevertheless concentrate on one aspect, namely how to find the appropriate information as outlined in the next paragraph.

### 3.2.1 Semantic Mediators for Mathematical Knowledge Bases

Knowledge acquisition and retrieval in the currently emerging large repositories of formalized mathematical knowledge should not be based purely on syntactic matching, but it needs to be supported by *semantic* mediators.

To prove a mathematical theorem in a particular domain is initially blind. Indeed, in order to prevent a search space explosion, only part of the relevant knowledge is made available at the start. For instance, in the $\Omega$MEGA system the proof planner MULTI selects a subset of the available knowledge which

---

[8] `www.dfki.de`

[9] `www.mathweb.org/omdoc/`

consists, for each theorem, of a set of assertions (axioms, definitions, lemmata), tactics and proof-planning methods. As this selection is naturally incomplete, there is the need to incrementally incorporate additional knowledge if needed.

We are working on appropriately limited higher-order reasoning agents for domain- and context-specific retrieval of mathematical knowledge from a mathematical knowledge base. For this we shall adapt a two stage approach as in [17], which combines syntactically oriented pre-filtering with semantic analysis. The pre-filter employ efficiently processable criteria based on meta-data and ontologies that identify sets of candidate theorems of a mathematical knowledge base that are potentially applicable to a focused proof context. The higher-order agents then act as post-filters to exactly determine the applicable theorems of this set.

### 3.3 MathServ: A Global Web for Mathematical Services

The Internet provides a vast collection of data and computational resources. For example, a travel booking system combines different information sources, such as the search engines, price computation schemes, and the travel information in distributed very large databases, in order to answer complex booking requests. The access to such specialized travel information sources has to be planned, the obtained results combined, and, in addition, the consistency of time constraints has to be guaranteed. We want to transfer and apply this methodology to mathematical problem solving and develop a system that plans the combination of several mathematical information sources (such as mathematical databases), computer algebra systems, and reasoning processes (such as theorem provers or constraint solvers). Based on the well-developed MATHWEB-SB network of mathematical services, the existing client-server architecture will be extended by advanced problem solving capabilities and semantic brokering of mathematical services (see [101]).

### 3.4 Support for Mathematical Activities

Proof construction is an important but only a small part of a much wider range of mathematical activities an assistance system for mathematics should support.

#### 3.4.1 Certified Mathematics Texts

A mathematician or software engineer writes a paper usually in a LaTeX-like environment. The definitions, lemmata, theorems and especially their proofs
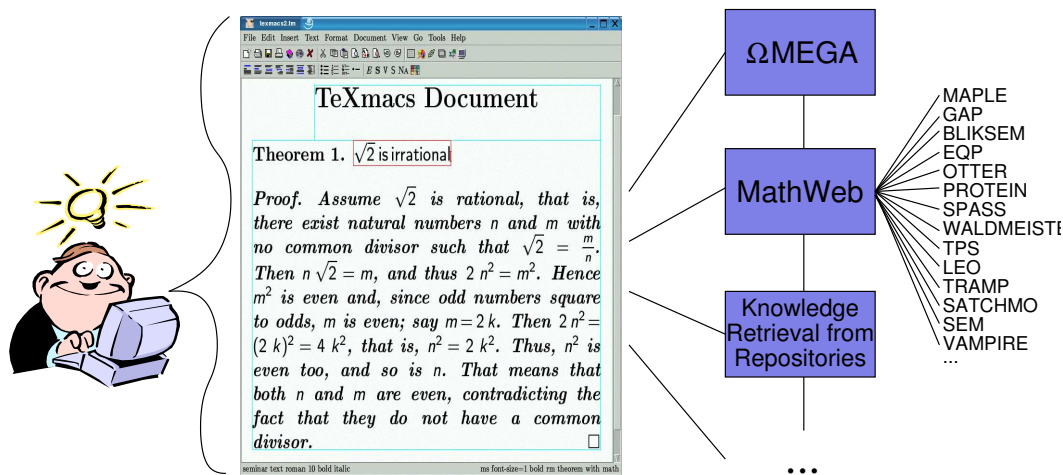
Figure 7. Documents in TeXmacs: The user will be supported by different mathematical reasoning services that "understand" the document content.

give rise to extensions of the original theory he started with. If the proofs of the new theorems and their consisteny with previous assertions are computer checked, we have mathematical documents in a publishable style which in addition are formally validated, hence obtaining *certified mathematical documents.* A first step in that direction is currently under development by linking the WYSIWYG mathematical editor TEXMACS [94] with the ΩMEGA system (see Figure 7).

The TEXMACS-system provides LaTeX-like editing and macro-definition features, and we are defining macros for theory-specific knowledge such as types, constants, axioms, and lemmata. This allows us to translate new textual definitions and lemmata into the formal representation, as well as to translate (partial) textbook proofs into (partial) proof plans.

### 3.4.2 Mathematical Advice in Tutoring Systems

We are also involved in the DFKI project ActiveMath [75], which develops an e-learning tool for tutoring maths students, in particular in advising a student how to prove a theorem. This scenario is currently also under investigation in the DIALOG [10] project [13,21] and, aside from all linguistic analysis problems, gives rise to the problem to bridge the gap between the human style of proofs and machine-oriented proof representations. Human-authored proofs are often imprecise in several respects, namely (i) the used inference rule is not men-

---

[10] The DIALOG project is a collaboration between the Computer Science and Computational Linguistics departments of Saarland University as part of the Collaborative Research Center on *Resource-Adaptive Cognitive Processes*, SFB 378 (`www.coli.uni-saarland.de/projects/sfb378/`).

tioned, (ii) some of the premises needed for a step in the derivation are not mentioned, and (iii) some steps of the derivation are completely omitted.

Another interesting and novel application for theorem proving systems in the DIALOG project is proof step evaluation (see [21]): Each proof step uttered by a student within a tutorial context has to be analyzed with respect to the following criteria:

*Soundness*: Can the proof step be reconstructed by a formal inference system and logically and tutorially verified?

*Granularity*: Is the 'argumentative complexity' or 'size' of the proof step logically and tutorially acceptable?

*Relevance*: Is the proof step logically and tutorially useful for achieving the final goal?

# References

[1] S. Allen, R. Constable, R. Eaton, C. Kreitz, and L. Lorigo. The Nuprl open logical environment. In McAllester [59].

[2] P.B. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and H. Xi. TPS: A theorem proving system for classical type theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.

[3] S. Autexier. *Hierarchical Contextual Reasoning*. PhD thesis, Computer Science Department, Saarland University, Saarbrücken, Germany, 2003.

[4] S. Autexier. The CoRe calculus. In Robert Nieuwenhuis, editor, *Proceedings of the 20th Conference on Automated Deduction (CADE-20)*, volume 3632 of *LNAI*, pages 84–98, Tallinn, Estonia, July 2005. Springer.

[5] S. Autexier, C. Benzmüller, A. Fiedler, H. Horacek, and Q. Bao Vo. Assertion-level proof representation with under-specification. *Electronic Notes in Theoretical Computer Science*, 93:5–23, 2003.

[6] S. Autexier and D. Hutter. Maintenance of formal software development by stratified verification. In M. Baaz and A. Voronkov, editors, *Proceedings of LPAR'02*, LNCS, Tbilissi, Georgia, September 2002. Springer.

[7] S. Autexier, D. Hutter, T. Mossakowski, and A. Schairer. The development graph manager MAYA. In H. Kirchner and C. Ringeissen, editors, *Proceedings 9th International Conference on Algebraic Methodology And Software Technology (AMAST'02)*, volume 2422 of *LNCS*. Springer, September 2002.

[8] S. Autexier and T. Mossakowski. Integrating HOL-CASL into the development graph manager MAYA. In A. Armando, editor, *Proceedings of FROCOS'02*, volume 2309 of *LNAI*, pages 2–17. Springer, April 2002.

[9]  R. Bartle and D. Sherbert. *Introduction to Real Analysis*. Wiley, 2nd edition, 1982.

[10] P. Baumgartner and U. Furbach. PROTEIN, a PROver with a Theory INterface. In Bundy [27], pages 769–773.

[11] C. Benzmüller. *Equality and Extensionality in Higher-Order Theorem Proving*. PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, 1999.

[12] C. Benzmüller, M. Bishop, and V. Sorge. Integrating TPS and ΩMEGA. *Journal of Universal Computer Science*, 5:188–207, 1999.

[13] C. Benzmüller, A. Fiedler, M. Gabsdil, H. Horacek, I. Kruijff-Korbayova, M. Pinkal, J. Siekmann, D. Tsovaltzi, B. Quoc Vo, and M. Wolska. Tutorial dialogs on mathematical proofs. In *Proceedings of IJCAI-03 Workshop on Knowledge Representation and Automated Reasoning for E-Learning Systems*, pages 12–22, Acapulco, Mexico, 2003.

[14] C. Benzmüller, A. Fiedler, A. Meier, and M. Pollet. Irrationality of $\sqrt{2}$ — a case study in ΩMEGA. Seki-Report SR-02-03, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2002.

[15] C. Benzmüller, M. Jamnik, M. Kerber, and V. Sorge. Experiments with an agent-oriented reasoning system. In Franz Baader, Gerhard Brewka, and Thomas Eiter, editors, *KI 2001: Advances in Artificial Intelligence, Joint German/Austrian Conference on AI, Vienna, Austria, September 19-21, 2001, Proceedings*, number 2174 in LNAI, pages 409–424. Springer, 2001.

[16] C. Benzmüller and M. Kohlhase. LEO — a higher-order theorem prover. In Kirchner and Kirchner [54].

[17] C. Benzmüller, A. Meier, and V. Sorge. Bridging theorem proving and mathematical knowledge retrieval. In D. Hutter and W. Stephan, editors, *Festschrift in Honour of Jörg Siekmann's 60s Birthday*, number 2605 in LNAI. Springer, 2004.

[18] C. Benzmüller and V. Sorge. A blackboard architecture for guiding interactive proofs. In Giunchiglia [46].

[19] C. Benzmüller and V. Sorge. Ωants – An open approach at combining Interactive and Automated Theorem Proving. In Kerber and Kohlhase [53].

[20] C. Benzmüller, V. Sorge, M. Jamnik, and M. Kerber. Can a higher-order and a first-order theorem prover cooperate? In F. Baader and A. Voronkov, editors, *Proceedings of the 11th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR)*, volume 3452 of *LNAI*, pages 415–431. Springer, 2005.

[21] C. E. Benzmüller and Q. B. Vo. Mathematical domain reasoning tasks in natural language tutorial dialog on proofs. In M. Veloso and S. Kambhampati, editors, *Proceedings of the Twentieth National Conference on Artificial*

Intelligence (AAAI-05), pages 516–522, Pittsburgh, Pennsylvania, USA, 2005. AAAI Press / The MIT Press.

[22] Christoph Benzmüller and Volker Sorge. Critical agents supporting interactive theorem proving. In Pedro Borahona and Jose J. Alferes, editors, *Proceedings of the 9th Portuguese Conference on Artificial Intelligence (EPIA'99)*, number 1695 in LNAI, pages 208–221, Evora, Portugal, 1999. Springer.

[23] W. Bledsoe. Challenge problems in elementary calculus. *Journal of Automated Reasoning*, 6:341–359, 1990.

[24] N. Bourbaki. Theory of sets. In *Elements of Mathematics*, volume 1. Addison-Wesley, 1968.

[25] A. Bundy. The use of explicit plans to guide inductive proofs. In Lusk and Overbeek [57], pages 111–120.

[26] A. Bundy. A science of reasoning. In G. Plotkin J.-L. Lasser, editor, *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–199. MIT Press, 1991.

[27] A. Bundy, editor. *Proceedings of the 12th Conference on Automated Deduction*, number 814 in LNAI. Springer, 1994.

[28] A. Bundy. A critique of proof planning. In *Computational Logic: Logic Programming and Beyond*, number 2408 in LNCS, pages 160–177. Springer, 2002.

[29] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam System. In M. Stickel, editor, *Proceedings of the 10th Conference on Automated Deduction*, number 449 in LNCS, pages 647–648, Kaiserslautern, Germany, 1990. Springer.

[30] B. Char, K. Geddes, G. Gonnet, B. Leong, M. Monagan, and S. Watt. *First leaves: a tutorial introduction to Maple V*. Springer, 1992.

[31] L. Cheikhrouhou and J. Siekmann. Planning diagonalization proofs. In Giunchiglia [46], pages 167–180.

[32] L. Cheikhrouhou and V. Sorge. PDS — A Three-Dimensional Data Structure for Proof Plans. In *Proceedings of the International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications (ACIDCA'2000)*, Monastir, Tunisia, 22–24 March 2000.

[33] A. Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[34] Coq Development Team. *The Coq Proof Assistant Reference Manual*. INRIA, 1999-2003. See `coq.inria.fr/doc/main.html`.

[35] M. Davis. The prehistory and early history of automated deduction. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning*, volume 2 Classical Papers on Computational Logic 1967–1970 of *Symbolic Computation*. Springer, 1983.

[36] M. Davis. The early history of automated deduction. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 1, pages 3–15. Elsevier Science, 2001.

[37] M. Davis, editor. *The Undecidable: Basic Papers on undecidable Propositions, unsolvable Problems and Computable Functions*. Dover Publication, New York, February 2004.

[38] H. de Nivelle. Bliksem 1.10 user manual. Technical report, Max-Planck-Institut für Informatik, 1999.

[39] A. Fiedler. Dialog-driven adaptation of explanations of proofs. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1295–1300, Seattle, WA, 2001. Morgan Kaufmann.

[40] A. Fiedler. P.rex: An interactive proof explainer. In Goré et al. [48].

[41] A. Fiedler. *User-adaptive proof explanation*. PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2001.

[42] A. Franke and M. Kohlhase. System description: MBase, an open mathematical knowledge base. In McAllester [59].

[43] Andreas Franke and Michael Kohlhase. System description: Mbase, an open mathematical knowledge base. In McAllester [59].

[44] H. Ganzinger, editor. *Proceedings of the 16th Conference on Automated Deduction*, number 1632 in LNAI. Springer, 1999.

[45] H. Gebhard. Beweisplanung für die Beweise der Vollständigkeit verschiedener Resolutionskalküle in ΩMEGA. Master's thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, 1999.

[46] F. Giunchiglia, editor. *Proceedings of 8th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA'98)*, number 1480 in LNAI. Springer, 1998.

[47] M. Gordon and T. Melham. *Introduction to HOL – A theorem proving environment for higher order logic*. Cambridge University Press, 1993.

[48] R. Goré, A. Leitsch, and T. Nipkow, editors. *Automated Reasoning — 1st International Joint Conference, IJCAR 2001*, number 2083 in LNAI. Springer, 2001.

[49] J. Hadamard. *The Psychology of Invention in the Mathematical Field*. Dover Publications, New York, USA; edition 1949, 1944.

[50] Th. Hillenbrand, A. Jaeger, and B. Löchner. System description: Waldmeister — improvements in performance and ease of use. In Ganzinger [44], pages 232–236.

[51] X. Huang. Reconstructing Proofs at the Assertion Level. In Bundy [27], pages 738–752.

[52] D. Hutter. Management of change in structured verification. In *Proceedings of Automated Software Engineering, ASE-2000*. IEEE, 2000.

[53] M. Kerber and M. Kohlhase, editors. *8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus-2000)*. AK Peters, 2000.

[54] C. Kirchner and H. Kirchner, editors. *Proceedings of the 15th Conference on Automated Deduction*, number 1421 in LNAI. Springer, 1998.

[55] H. Kirchner and C. Ringeissen, editors. *Frontiers of combining systems: Third International Workshop, FroCoS 2000*, volume 1794 of *LNAI*. Springer, 2000.

[56] M. Kohlhase and A. Franke. MBase: Representing knowledge and context for the integration of mathematical software systems. *Journal of Symbolic Computation; Special Issue on the Integration of Computer Algebra and Deduction Systems*, 32(4):365–402, September 2001.

[57] E. Lusk and R. Overbeek, editors. *Proceedings of the 9th Conference on Automated Deduction*, number 310 in LNCS, Argonne, Illinois, USA, 1988. Springer.

[58] R. Manthey and F. Bry. SATCHMO: A theorem prover implemented in Prolog. In Lusk and Overbeek [57], pages 415–434.

[59] D. McAllester, editor. *Proceedings of the 17th Conference on Automated Deduction*, number 1831 in LNAI. Springer, 2000.

[60] W. McCune. Solution of the Robbins problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997.

[61] W. W. McCune. Otter 3.0 reference manual and guide. Technical Report ANL-94-6, Argonne National Laboratory, Argonne, Illinois 60439, USA, 1994.

[62] A. Meier. TRAMP: Transformation of Machine-Found Proofs into Natural Deduction Proofs at the Assertion Level. In McAllester [59].

[63] A. Meier. *Proof Planning with Multiple Strategies*. PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2004.

[64] A. Meier and E. Melis. MULTI: A multi-strategy proof planner (system description). In Robert Nieuwenhuis, editor, *Proceedings of the 20$^{th}$ Conference on Automated Deduction (CADE-20)*, volume 3632 of *LNAI*, pages 250–254, Tallinn, Estonia, July 2005. Springer.

[65] A. Meier, E. Melis, and M. Pollet. Towards extending domain representations. Seki Report SR-02-01, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2002.

[66] A. Meier, M. Pollet, and V. Sorge. Classifying Isomorphic Residue Classes. In R. Moreno-Diaz, B. Buchberger, and J.-L. Freire, editors, *A Selection of Papers from the 8th International Workshop on Computer Aided Systems Theory (EuroCAST 2001)*, number 2178 in LNCS, pages 494–508. Springer, 2001.

[67] A. Meier, M. Pollet, and V. Sorge. Comparing Approaches to the Exploration of the Domain of Residue Classes. *Journal of Symbolic Computation, Special Issue on the Integration of Automated Reasoning and Computer Algebra Systems*, 34(4):287–306, October 2002. Steve Linton and Roberto Sebastiani, eds.

[68] A. Meier and V. Sorge. Exploring properties of residue classes. In Kerber and Kohlhase [53].

[69] Andreas Meier, Erica Melis, and Jörg Siekmann. Proof planning with multiple startegies. Submitted to Artificial Intelligence.

[70] E. Melis. Island planning and refinement. Seki-Report SR-96-10, Department of Computer Science, Saarland University, Saarbrücken, Germany, 1996.

[71] E. Melis. AI-techniques in proof planning. In H. Prade, editor, *Proceedings of the 13th European Conference on Artifical Intelligence*, pages 494–498, Brighton, UK, 1998. John Wiley & Sons, Chichester, UK.

[72] E. Melis. AI-techniques in proof planning. In *European Conference on Artificial Intelligence*, pages 494–498, Brighton, 1998. Kluwer.

[73] E. Melis and A. Meier. Proof planning with multiple strategies. In J. Loyd, V. Dahl, U. Furbach, M. Kerber, K. Lau, C. Palamidessi, L.M. Pereira, Y. Sagivand, and P. Stuckey, editors, *First International Conference on Computational Logic (CL-2000)*, number 1861 in LNAI, pages 644–659, London, UK, 2000. Springer.

[74] E. Melis and J. Siekmann. Knowledge-based proof planning. *Artificial Intelligence*, 115(1):65–105, 1999.

[75] E. Melis and J. Siekmann. Activemath: An intelligent tutoring system for mathematics. volume 3070, pages 91–101. Springer-Verlag, 2004.

[76] E. Melis, J. Zimmer, and T. Müller. Integrating constraint solving into proof planning. In Kirchner and Ringeissen [55].

[77] T. Mossakowski, S. Autexier, and D. Hutter. Extending development graphs with hiding. In Heinrich Hussmann, editor, *Proceedings of Fundamental Approaches to Software Engineering (FASE 2001)*, number 2029 in LNCS, pages 269–283, Genova, April 2001. Springer.

[78] T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic.* Number 2283 in LNCS. Springer, 2002.

[79] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T. Henzinger, editors, *Computer-Aided Verification, CAV '96*, number 1102 in LNCS, pages 411–414, New Brunswick, NJ, 1996. Springer.

[80] L. Paulson. *Isabelle: A Generic Theorem Prover.* Number 828 in LNCS. Springer, 1994.

[81] G. Polya. *How to Solve it.* Princeton University Press, 1973.

[82] A. Riazanov and A. Voronkov. Vampire 1.1 (system description). In Goré et al. [48].

[83] J. Richardson, A. Smaill, and I. Green. System description: Proof planning in higher-order logic with λClam. In Kirchner and Kirchner [54].

[84] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of ACM*, 12(1):23–41, 1965.

[85] M. Schönert et al. *GAP – Groups, Algorithms, and Programming.* Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 1995.

[86] J. Siekmann. Geschichte des Automatischen Beweisens (History of Automated Deduction). In *Deduktionssysteme, Automatisierung des Logischen Denkens.* R. Oldenbourg Verlag, 2nd edition, 1992. Also in English with Elsewood.

[87] J. Siekmann. History of computational logic. In D. Gabbay and J. Woods, editors, *The Handbook of the History of Logic*, volume I-IX. Elsevier, 2004. To appear.

[88] J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, I. Normann, and M. Pollet. Proof development in OMEGA: The irrationality of square root of 2. In F. Kamareddine, editor, *Thirty Five Years of Automating Mathematics*, Kluwer Applied Logic series (28), pages 271–314. Kluwer Academic Publishers, 2003. ISBN 1-4020-1656-5.

[89] J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, and M. Pollet. Proof development with OMEGA: Sqrt(2) is irrational. In M. Baaz and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 9th International Conference, LPAR 2002*, number 2514 in LNAI, pages 367–387. Springer, 2002.

[90] J. Siekmann, S. Hess, C. Benzmüller, L. Cheikhrouhou, A. Fiedler, H. Horacek, M. Kohlhase, K. Konrad, A. Meier, E. Melis, M. Pollet, and V. Sorge. $\mathcal{LOUI}$: $\mathcal{L}$ovely $\Omega$MEGA $\mathcal{U}$ser $\mathcal{I}$nterface. *Formal Aspects of Computing*, 11:326–342, 1999.

[91] V. Sorge. Non-Trivial Computations in Proof Planning. In Kirchner and Ringeissen [55].

[92] V. Sorge. ΩANTS — A Blackboard Architecture for the Integration of Reasoning Techniques into Proof Planning. PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2001.

[93] G. Sutcliffe, C. Suttner, and T. Yemenis. The TPTP problem library. In Bundy [27].

[94] J. van der Hoeven. GNU TeXmacs: A free, structured, wysiwyg and technical text editor. In *Actes du congrès Gutenberg*, number 39-40 in Actes du congrès Gutenberg, pages 39–50, Metz, May 2001.

[95] C. Weidenbach, B. Afshordel, U. Brahm, C. Cohrs, Th. Engel, E. Keen, C. Theobalt, and D. Topic. System description: SPASS version 1.0.0. In Ganzinger [44], pages 378–382.

[96] D. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.

[97] F. Wiedijk. The seventeen provers of the world. To appear in Lecture Notes in Artificial Intelligence, Springer, 2005.

[98] J. Zhang and H. Zhang. SEM: A system for enumerating models. In C. S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 298–303, Montreal, Canada, 1995. Morgan Kaufmann, San Mateo, California, USA.

[99] J. Zimmer and M. Kohlhase. System description: The Mathweb Software Bus for distributed mathematical reasoning. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, number 2392 in LNAI, pages 138–142. Springer, 2002.

[100] J. Zimmer and E. Melis. Constraint solving for proof planning. *Journal of Automated Reasoning*, 33:51–88, 2004.

[101] Jürgen Zimmer. A Framework for Agent-based Brokering of Reasoning Services. In Raul Monroy, Gustavo Arroyo-Figueroa, Luis Enrique Sucar, and Juan Humberto Sossa Azuela, editors, *MICAI*, volume 2972 of *Lecture Notes in Computer Science*. Springer, 2004.