JÖRG SIEKMANN, CHRISTOPH BENZMÜLLER, ARMIN FIEDLER, ANDREAS MEIER, IMMANUEL NORMANN, AND MARTIN POLLET

PROOF DEVELOPMENT WITH ΩMEGA: THE IRRATIONALITY OF $\sqrt{2}$

The well-known theorem asserting the irrationality of $\sqrt{2}$ was proposed as a case study for a comparison of fifteen (interactive) theorem proving systems [Wiedijk, 2002]. This represents an important shift of emphasis in the field of automated deduction away from the somehow artificial problems of the past back to real mathematical challenges.

We present an overview of the ΩMEGA system as far as it is relevant for the purpose of this paper, and then we discuss three different styles of proof development in ΩMEGA using the example of the irrationality of $\sqrt{2}$: The first follows the traditional tactical theorem proving approach without any mathematical knowledge, the second employs the idea of interactive island proof planning, and the third is a fully automated proof based on planning with ΩMEGA's proof planner MULTI. Moreover, we illustrate the expansion and subsequent verification of the proofs at the logic level. We also discuss the knowledge engineering process by which the main ideas of the interactive island proof are generalized into respective proof methods, such that automatic proof planning becomes feasible for this domain.

1 ΩMEGA

The ΩMEGA proof development system [Siekmann *et al.*, 2002] 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 the working mathematician.

ΩMEGA is a mathematical assistant tool that supports proof development in mathematical domains at a user-friendly level of abstraction. It is a modular system with a central proof data structure and several supplementary subsystems. ΩMEGA has many characteristics in common with systems like NUPRL [Allen *et al.*, 2000], COQ [Coq Development Team, 1999-2003], HOL [Gordon and Melham, 1993], PVS [Owre *et al.*, 1996], and Isabelle [Paulson, 1994; Nipkow *et al.*, 2002]. However, it differs significantly 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

Fairouz Kamareddine (ed.),
Thirty Five Years of Automating Mathematics *271–314.*

λCLAM at Edinburgh [Richardson *et al.*, 1998; Bundy *et al.*, 1990]. We shall now present an overview of the architecture of the ΩMEGA system and show some of its novel features, which include facilities to access several external reasoning systems and to integrate their results into a single proof structure; substantial support for interactive proof development through some non-standard inspection facilities and for guidance in the search for a proof; and finally methods to develop proofs at a human-oriented, higher level of abstraction.

1.1 System Overview

The ΩMEGA project currently represents one of the largest attempts worldwide to build an assistant tool for the working mathematician. It is a representative of systems in the new 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 abstraction is an interactive theorem prover based on a higher-order natural deduction (ND) variant of a soft-sorted[1] version of Church's simply typed λ-calculus [Church, 1940]. The logical language, which also supports partial functions, is called $\mathcal{POST}$, for ***p***artial functions ***o***rder ***s***orted ***t***ype theory. While this represents the "machine code" of the system the user will seldom want to see, the search for a proof is usually conducted at a higher level of abstraction defined by *tactics* and *methods*. Automated proof search at this abstract level is called *proof planning* (see Section 1.3). Proof construction is also supported by already proven assertions and theorems and by calls to external systems to simplify or solve subproblems, as will be shown in Section 1.2.

At the core of ΩMEGA is the *proof plan data structure* $\mathcal{PDS}$ [Cheikhrouhou and Sorge, 2000], in which proofs and *proof plans* are represented at various levels of granularity and abstraction. 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, which is currently being replaced by the mathematical knowledge base MBASE [Franke and Kohlhase, 2000; Kohlhase and Franke, 2001]. The user of ΩMEGA, or the proof planner MULTI [Melis and Meier, 2000], or else the suggestion mechanism Ω-ANTS [Benzmüller and Sorge, 2000] modify the $\mathcal{PDS}$ during proof development until a complete proof plan 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 the most appropriate level of abstraction has been found, this plan must be expanded by sub-methods

[1] Unary predicates are interpreted as sorts and theorems of a certain syntactical form as sort declarations. Sort inferences using this information are explicit proof steps.

and sub-tactics into lower levels of abstraction 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 ΩMEGA'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 an LCF-style tactic expansion mechanism, proof search or a combination of both 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 relationships being dynamically maintained. Failing expansions of proof steps typically lead to open nodes at a lower proof level and thus result in an incomplete proof. The reasons for the failure can in principle be analyzed in the $\mathcal{PDS}$ in the sense of proof critics [Ireland and Bundy, 1996], however, this option has not yet been further explored.

User interaction is supported by the graphical user interface $\mathcal{L}\Omega\mathcal{UI}$ [Siekmann *et al.*, 1999], which provides both a graphical and a tabular view of the proof under consideration, and the interactive proof explanation system *P.rex* [Fiedler, 2001a; Fiedler, 2001b], which provides the user with a natural-language presentation of the proof.

The previously monolithic system has been split up and separated into several independent modules, which are connected via the mathematical software bus MATHWEB-SB [Zimmer and Kohlhase, 2002]. An important benefit is that MATHWEB modules can be distributed over the Internet and are then remotely accessible by other research groups as well. There is a very active user community with sometimes several thousand theorems and lemmata being proved per day; most theorems are generated automatically as (currently non-reusable and non-indexed) subproblems in natural language processing (see the Doris system [Doris, 2001]), proof planning [Meier *et al.*, 2002b; Sorge, 2001], and verification tasks.

1.2 External Systems

Proof problems require many different skills for their solution. Therefore, it is desirable to have access to several systems with complementary capabilities, to orchestrate their use, and to integrate their results. ΩMEGA 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 abstraction, and also for a human-oriented

display and inspection of a partial proof. Importantly, it also enables us to check the soundness of each contribution by refining the inserted subproof to a logic-level proof to be verified by ΩMEGA's proof checker.

Currently, the following external systems are integrated 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 [Sorge, 2000]. Currently, ΩMEGA uses the systems MAPLE [Char *et al.*, 1992] and GAP [Schönert and others, 1995].

ATPs are employed to solve subgoals. Currently ΩMEGA uses the first-order provers BLIKSEM [de Nivelle, 1999], EQP [McCune, 1997], OTTER [McCune, 1994], PROTEIN [Baumgartner and Furbach, 1994], SPASS [Weidenbach *et al.*, 1999], WALDMEISTER [Hillenbrand *et al.*, 1999], the higher-order systems TPS [Andrews *et al.*, 1996], and $\mathcal{LEO}$ [Benzmüller and Kohlhase, 1998; Benzmüller, 1999], and we plan to incorporate VAMPIRE [Riazanov and Voronkov, 2001]. The first-order ATPs are connected via TRAMP [Meier, 2000], which is a proof transformation system that transforms resolution-style proofs into assertion-level ND proofs to be integrated into ΩMEGA's $\mathcal{PDS}$. TPS already provides ND proofs, which can be further processed and checked with little transformational effort [Benzmüller *et al.*, 1999].

MGs provide either witnesses for free (existential) variables, or countermodels, which show that some subgoal is not a theorem. Hence, they help to guide the proof search. Currently, ΩMEGA uses the MGs SATCHMO [Manthey and Bry, 1988] and SEM [Zhang and Zhang, 1995].

CSs construct mathematical objects with theory-specific properties as witnesses for free (existential) variables. Moreover, a CS can help to reduce the proof search by checking for inconsistencies of constraints. Currently, ΩMEGA employs $\mathcal{CoSIE}$ [Melis *et al.*, 2000], a CS for inequalities and equations over the field of real numbers.

1.3 Proof Planning

ΩMEGA's main focus is on knowledge-based proof planning [Bundy, 1988; Melis and Siekmann, 1999], where proofs are not conceived in terms of low-level calculus rules, but at a much higher level of abstraction that highlights

the main ideas and de-emphasizes minor logical or mathematical manipulations on formulae.

Knowledge-based proof planning is a new paradigm in automated theorem proving, which swings the motivational pendulum back to its 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 abstraction: the proof of a theorem is planned at an abstract level where an outline of the proof is found. This outline, that is, the abstract proof plan, can be recursively expanded to construct a proof within a logical calculus provided the proof plan does not fail. The plan operators represent mathematical techniques familiar to a working mathematician. While the knowledge of such a mathematical domain as represented within 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 [Melis and Siekmann, 1999] used the limit theorems proposed by Woody Bledsoe [Bledsoe, 1990] as a challenge to automated reasoning systems. The general-purpose planner makes use of this mathematical domain knowledge 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 provide a basis for meta-level reasoning and goal-directed behavior.

In ΩMEGA, domain knowledge is encoded in methods, control rules, and strategies. Moreover, methods and control rules can employ external systems (e.g., computer algebra systems) and make use of the knowledge in these systems. ΩMEGA's multi-strategy proof planner MULTI [Melis and Meier, 2000] searches then for a plan using the acquired methods and strategies guided by the control knowledge in the control rules.

AI Principles in Proof Planning

In AI, 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, which reach a goal state from the initial state when executed. Such a sequence of actions is also called a *solution plan.*

A simple, yet very influential language is the STRIPS representation [Fikes and Nilsson, 1971; Fikes *et al.*, 1972]. Formalized in propositional logic, STRIPS describes the initial state by a set of ground literals. A goal is described by a conjunction of positive literals. Operators in STRIPS have *preconditions* and *effects*, which formalize to which states the operator can

be applied and how these states are changed by its application, respectively. Whereas the preconditions of an operator are represented by a conjunction of positive literals, the effects are represented by a conjunction of positive and negative literals. The positive literals in the operator's effects are called the *add list* of the operator, while the negative literals are called the *delete list* of the operator.

The classical approach to planning problems is *precondition achievement planning* [Drummond, 1994], which goes back to the General Problem Solver, GPS [Newell and Simon, 1963]. The planning process starts from the goal, which is considered as an unachieved precondition. First, the add list of all operators is checked to see whether it contains an effect that achieves some literal of the goal. Then, one such operator is chosen and appropriately instantiated, and the resulting action is inserted into the plan under development, thus satisfying part of the goal. The preconditions of the introduced action become new unsatisfied preconditions of the plan, and the planning process recurses on those.

Proof planning considers mathematical theorems as planning problems [Bundy, 1988]. 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.

In ΩMEGA, proof planning is the process that computes actions, that is, instantiations of methods, and assembles them in order to derive a theorem from a set of assumptions. The effects and the preconditions of an action in proof planning are proof nodes with formulae in the higher-order language $\mathcal{POST}$, where the effects are considered as logically inferable from the preconditions. A proof plan under construction is represented in the proof plan data structure $\mathcal{PDS}$ (see Section 1.5). Initially, the $\mathcal{PDS}$ consists of an open node containing the statement to be proved, and closed, that is, justified, nodes for the proof assumptions. The introduction of an action changes the $\mathcal{PDS}$ by adding new proof nodes and justifying the effects of the action by applications of the method of the action 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 actions ΩMEGA's proof planning combines forward and backward state-space planning. Thus, a *planning state* 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 transfered by forward actions into a new world state. The goal state consists of the initial open node and is transfered by backward actions 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 in the current goal state are satisfied.

As opposed to precondition achievement planning (e.g., see [Weld, 1994]),

effects of methods in proof planning do not cancel each other. For instance, an action with effect $\neg F$ introduced for the open node L_1 does not threaten the effect F introduced by another action for the open node L_2. Dependencies among open nodes result from shared variables for witness terms and their constraints. Constraints can be, for instance, 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 a constraint store. An action introducing new constraints 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. First results about how to analyze and deal with such failures are discussed in [Meier, 2003].

Methods, Control Rules, and Strategies

In ΩMEGA, *methods* can be perceived as tactics in tactical theorem proving augmented with preconditions and effects, called *premises* and *conclusions*, respectively. A method represents the inference of the conclusion from the premises. For instance, NotI-M is a 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 by deriving from 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. Examples of the latter will be discussed in detail in Section 3.

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. We shall discuss examples for control rules in Section 3.1.

Strategies employ different sets of methods and control rules and, thus, tackle the same problem in different ways. The reasoning as to which strategy to employ on a problem is an explicit choice point in MULTI. In partic-

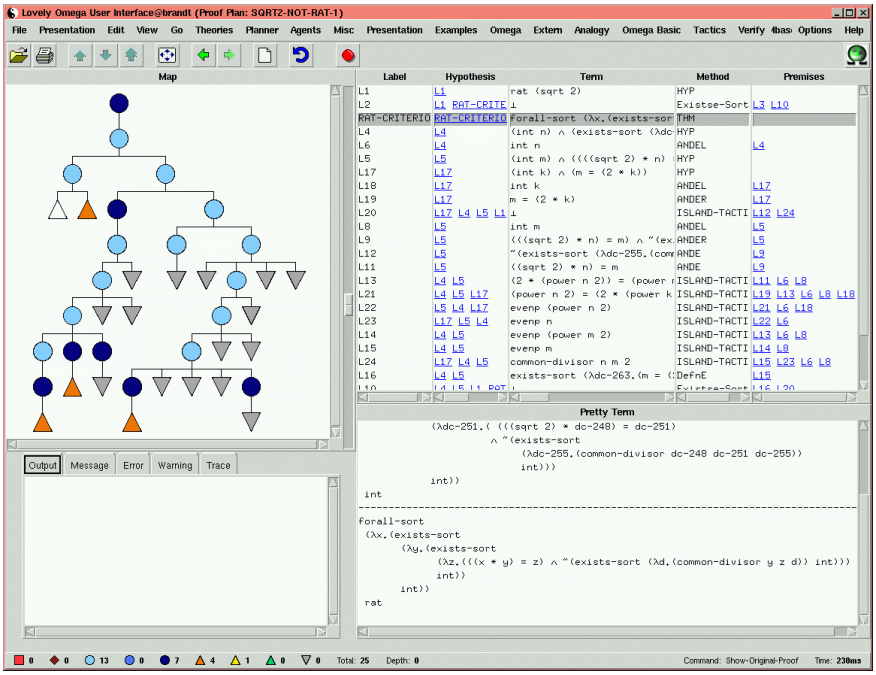

Figure 1. Multi-modal proof presentation in the graphical user interface $\mathcal{LOUI}$.

ular, MULTI can backtrack from chosen strategies and search at the level of strategies.

Detailed discussions of ΩMEGA's method and control rule language can be found in [Meier *et al.*, 2002a]. A detailed introduction to proof planning with multiple strategies is given in [Melis and Meier, 2000].

1.4 Interface and System Support

ΩMEGA's graphical user interface $\mathcal{LOUI}$ [Siekmann *et al.*, 1999] displays the current proof state 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 Fig. 1 and 2).

When inspecting portions of a proof by these facilities, the user can switch between alternative levels of abstraction, for example, by expanding a node in the graphical map of the proof tree, 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* [Fiedler, 2001a;

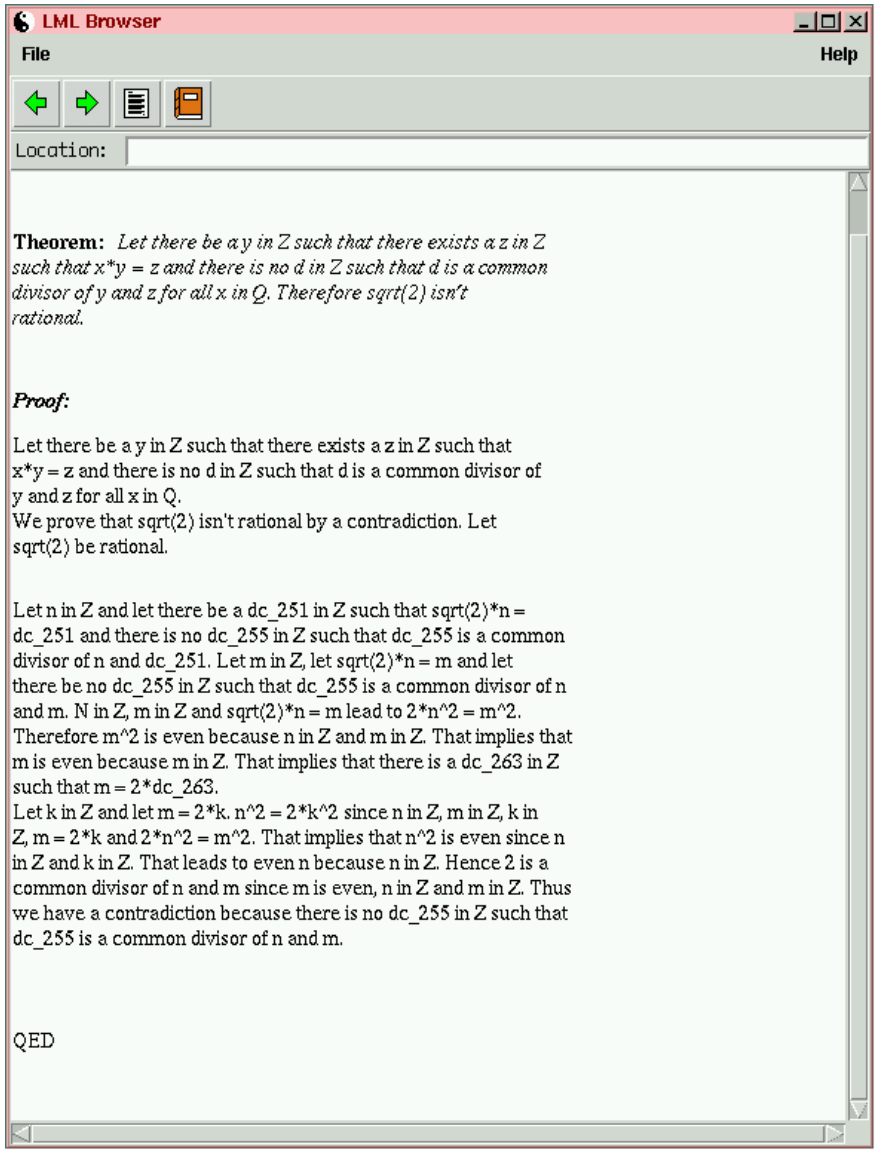

Figure 2. Natural language proof presentation by *P.rex* in $\mathcal{LOUI}$.

Fiedler, 2001b; Fiedler, 2001c], 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 lower levels of abstractions, for example, by detailing some ill-understood subproof.

Another system support feature of ΩMEGA is the guidance mechanism provided by the suggestion module Ω-ANTS [Benzmüller and Sorge, 1998; Benzmüller and Sorge, 2000; Sorge, 2001], which searches proactively for possible actions that may be helpful in finding a proof and orders them in a preference list. Examples for such actions are an application of a particular calculus rule, the call of a tactic or a proof method as well as a call of an external reasoning system, or the search for and insertion of facts from the knowledge base MBASE. The general idea is the following: every inference rule, tactic, method or external system is "agentified" in the sense that every possible *action* searches concurrently for the fulfillment of its application conditions and once these are satisfied it suggests its execution. User-definable heuristics select and display the suggestions to the user. Ω-ANTS is based on a hierarchical blackboard, which collects the data about the current proof state.

1.5 Proof Objects

The central data structure for the overall search is the proof plan data structure $\mathcal{PDS}$. This is a hierarchical data structure that represents a (partial) proof at different levels of abstraction (called partial proof plans). Technically, it is an acyclic graph, where the nodes are justified by tactic applications. Conceptually, each such justification represents a proof plan (the expansion of the justification) at a lower level of abstraction, which is computed when the tactic is executed. In ΩMEGA, we explicitly keep the original proof plan as well as intermediate expansion layers in an expansion hierarchy. The coexistence of several abstraction levels and the dynamical maintenance of their relationship is a central design objective of ΩMEGA's $\mathcal{PDS}$. Thus the $\mathcal{PDS}$ makes the hierarchical structure of proof plans explicit and retains it for further applications such as proof explanation with *P.rex* or analogical transfer of plans. The lowest level of abstraction of a $\mathcal{PDS}$ represents the ND calculus.

The proof object generated by ΩMEGA for the "irrationality of $\sqrt{2}$" theorem is recorded in a technical report [Benzmüller *et al.*, 2002], where the unexpanded and the expanded proof objects are presented in great detail, that is in a little less than a thousand proof steps.

1.6 Case Studies

Early developments of proof planning in Alan Bundy's group at Edinburgh used proofs by induction as their favorite case studies [Bundy, 1988]. The ΩMEGA 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 ϵ–δ–proofs [Melis and Siekmann, 1999; Melis, 1998]. This class was originally proposed by Woody Bledsoe [Bledsoe, 1990] 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 from the need for arithmetic computation in order to find a suitable instantiation of free (existential) variables (such as a δ depending on an ϵ). Crucial for the success of ΩMEGA'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 δ 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 [Bartle and Sherbert, 1982].

Another class of problems we tackled with proof planning is concerned with residue classes [Meier *et al.*, 2002b; Meier *et al.*, 2001]. In this domain we show 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 $\{\bar{0}_5, \bar{1}_5, \bar{2}_5, \bar{3}_5, \bar{4}_5\}$ 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 [Sorge, 2001]). Although the problems in this domain are still within the range of difficulty a traditional automated theorem prover can handle, it was nevertheless an interesting 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 [Cheikhrouhou and Siekmann, 1998]. 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 candidate for a standard proof technique are completeness proofs for 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. We have done

this for many refinements of resolution with ΩMEGA [Gebhard, 1999].

2 A CASE STUDY: $\sqrt{2}$ IS NOT RATIONAL

ΩMEGA's main aim is to become a proof assistant tool for the working mathematician. Hence, it should support interactive proof development at a user-friendly level of abstraction. The mathematical theorem that $\sqrt{2}$ is not rational, 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 the remainder of the paper, we will refer to this proof problem as the $\sqrt{2}$-*problem*. In [Wiedijk, 2002] fifteen systems that have solved the $\sqrt{2}$-problem show their respective 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 proved within the respective system.

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 [Sutcliffe *et al.*, 1994] — back to real mathematical challenges.

We participated in this case study essentially with three different contributions. Our initial contribution was an interactive proof in ΩMEGA without adding special domain knowledge to the system. For further details on this case study, which particularly demonstrates the use of ΩMEGA as a usual tactical theorem prover, we refer to [Benzmüller *et al.*, 2002]. The most important albeit not entirely new lesson to be learned from this experiment is that the level of abstraction common in most automated and tactical theorem proving environments is far too low. While our proof representation is already an abstraction (called the *assertion level* in [Huang, 1994]) 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 ATPs and one, albeit not the only one, of the reasons why they are not used as widely as, say, computer algebra systems.

This is the crucial issue in the ΩMEGA 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* [Melis, 1996], a technique that expects an outline of the proof and has the user provide main subgoals, called *islands*, together with their assumptions. The details of the proof, eventually down to the logic level, are postponed. 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 the external systems interfaced to ΩMEGA. 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 abstraction similar to proofs in mathematical textbooks.

Our third contribution to the case study of the $\sqrt{2}$-problem is a fully automatically planned and expanded proof of the theorem as presented in Section 3.

In the following sections we shall first describe the problem formalization (Section 2.1). Then, we shall present part of the tactic-level proof (Section 2.2) and the interactive island approach (Sections 2.3 to 2.5). Finally, in Section 3 we show the fully automated solution and we provide a generalization of it covering a whole class of similar problems. The actual challenge, attributed to the Pythagorean School, is as follows:

THEOREM. $\sqrt{2}$ is irrational.

Proof. [by contradiction] Assume $\sqrt{2}$ is rational, that is, there exist natural numbers m, n with no common divisor such that $\sqrt{2} = m/n$. Then $n\sqrt{2} = m$, and thus $2n^2 = m^2$. Hence m^2 is even and, since odd numbers square to odds, m is even; say $m = 2k$. Then $2n^2 = (2k)^2 = 4k^2$, that is, $n^2 = 2k^2$. Thus, n^2 is even too, and so is n. That means that both n and m are even, contradicting the fact that they do not have a common divisor. ■

2.1 Problem Formalization

The theorem is initially formulated in ΩMEGA's knowledge base as an open problem in the theory `REAL`. The problem is encoded in $\mathcal{POST}$ syntax, which is the logical input language for ΩMEGA:

```
(th~defproblem sqrt2-not-rat (in real)
  (conclusion (not (rat (sqrt 2))))
  (help "sqrt 2 is not a rational number."))
```

The concepts of the rational numbers (`rat`) and the square root (`sqrt`) are defined in the knowledge base as well. Since they are not needed in the interactive session at this abstract level and because of lack of space, we do not display them here (cf. [Benzmüller *et al.*, 2002] for the details).

To prove the given problem, further mathematical knowledge is required. Our proof employs the definition of evenness (`evenp`) and some theorems about rational numbers, evenness, and common divisors (`common-divisor`). However, the definition of `sqrt` is not needed in the main proof, because we use the computer algebra system MAPLE to justify the transformation

of $n\sqrt{2} = m$ into $2n^2 = m^2$. To do so, expressions in ΩMEGA such as $\sqrt{2}$ are mapped to corresponding MAPLE representations, and MAPLE uses its own built-in knowledge to manipulate them. Using and verifying these computation steps requires the expansion of MAPLE's computations to the calculus layer in ΩMEGA. As shown in [Sorge, 2000], this can be done by replaying MAPLE's computation by special computational tactics in ΩMEGA, which may also unfold some definitions such as `sqrt`. These tactics and their expansions are part of the SAPPER system and correspond directly to the mathematical definitions available in ΩMEGA's knowledge base. For example, the number 2 is defined in theory `NATURAL` as $s(s(0))$. Again, this knowledge is only required when expanding the abstract proof to the basic calculus layer and it is not visible to the user at this stage. However, SAPPER is still too weak in general, this is a subject of further development.

We now give examples for definitions and theorems used in our proofs of the $\sqrt{2}$-problem.

```
(th~defdef evenp (in integer)
  (definition
     (lam (x  num) (exists-sort (lam (y num) (= x (times 2 y))) int)))
  (help "Definition of even."))



(th~deftheorem rat-criterion (in real)
  (conclusion (forall-sort (lam (x num)
                (exists-sort (lam (y num)
                  (exists-sort (lam (z num)
                    (and (= (times x y) z)
                         (not (exists-sort (lam (d num)
                                                (common-divisor y z d))
                                int))))
                     int))
                  int))
                rat))
  (help "x rational implies there exist integers y,z which
         have no common divisor and furthermore z=x*y."))



(th~deftheorem square-even (in integer)
  (conclusion (forall-sort (lam (x num) (equiv (evenp (power x 2))
                                               (evenp x)))
              int))
 (help "x is even, iff x^2 is even."))
```

2.2 Tactical Theorem Proving in ΩMEGA

One way to construct proofs in ΩMEGA interactively is based on traditional tactical theorem proving. When employed in this mode ΩMEGA is comparable to many other interactive systems like NUPRL [Allen *et al.*, 2000], COQ [Coq Development Team, 1999-2003], HOL [Gordon and Melham, 1993], PVS [Owre *et al.*, 1996], and Isabelle [Paulson, 1994; Nipkow *et al.*, 2002]. However, a characteristic special to ΩMEGA is that several tools

(e.g., the various external systems) are available to support the interactive proof development process (cf. Section 1.2).

Our first case study on the $\sqrt{2}$-problem used the tactical theorem proving approach and proved the theorem in 33 interactive steps. These steps were automatically recorded by ΩMEGA in a so-called replay file, which contains all information that is needed to automatically replay a proof.[2] Here, we only sketch the interactions required for the problem by presenting the content of this replay file. A detailed description of the interactive proof can be found in [Benzmüller *et al.*, 2002].

```
Step 0:   OMEGA-BASIC PROVE (SQRT2-NOT-RAT)
Step 1:   DECLARATION DECLARE ((CONSTANTS (M NUM) (N NUM) (K NUM)))
Step 2:   RULES NOTI default default
Step 3:   MBASE IMPORT-ASS (RAT-CRITERION)
Step 4:   TACTICS FORALLE-SORT default default ((SQRT 2)) default
Step 5:   TACTICS EXISTSE-SORT default default (N) default
Step 6:   TACTICS ANDE default default default
Step 7:   TACTICS EXISTSE-SORT (L7) default (M) default
Step 8:   TACTICS ANDE* (L8) (NIL)
Step 9:   OMEGA-BASIC LEMMA default ((= (POWER M 2) (TIMES 2 (POWER N 2))))
Step 10:  TACTICS BY-COMPUTATION (L13) ((L11))
Step 11:  OMEGA-BASIC LEMMA (L9) ((EVENP (POWER M 2)))
Step 12:  RULES DEFN-CONTRACT default default default
Step 13:  OMEGA-BASIC LEMMA (L9) ((INT (POWER N 2)))
Step 14:  TACTICS WELLSORTED default default
Step 15:  TACTICS EXISTSI-SORT (L15) ((POWER N 2)) (L13) (L16) default
Step 16:  MBASE IMPORT-ASS (SQUARE-EVEN)
Step 17:  TACTICS ASSERT ((EVENP M)) ((SQUARE-EVEN L10 L14)) (NIL)
Step 18:  RULES DEFN-EXPAND (L17) default default
Step 19:  TACTICS EXISTSE-SORT default default (K) default
Step 20:  TACTICS ANDE (L19) default default
Step 21:  OMEGA-BASIC LEMMA default ((= (POWER N 2) (TIMES 2 (POWER K 2))))
Step 22:  TACTICS BY-COMPUTATION (L23) ((L13 L22))
Step 23:  OMEGA-BASIC LEMMA default ((EVENP (POWER N 2)))
Step 24:  RULES DEFN-CONTRACT default default default
Step 25:  OMEGA-BASIC LEMMA (L20) ((INT (POWER K 2)))
Step 26:  TACTICS WELLSORTED (L26) ((L21))
Step 27:  TACTICS EXISTSI-SORT default ((POWER K 2)) (L23) default default
Step 28:  TACTICS ASSERT ((EVENP N)) ((SQUARE-EVEN L6 L24)) (NIL)
Step 29:  MBASE IMPORT-ASS (EVEN-COMMON-DIVISOR)
Step 30:  OMEGA-BASIC LEMMA (L20) ((INT 2))
Step 31:  TACTICS WELLSORTED (L28) (NIL)
Step 32:  TACTICS ASSERT (FALSE) ((EVEN-COMMON-DIVISOR L10 L6 L12 L17 L27 L28)) (NIL)
Step 33:  RULES WEAKEN default default
```

The lesson to be learned from this protocol is that the wrong level of abstraction is still common in most automated and tactical theorem proving environments. This is our conviction even though De Bruijn's conjecture that the formalized proof object is at most a linear blow-up of the informal mathematical proof can be argued to actually hold for this example.

In the following section, we shall show how a proof at a more user-friendly level of abstraction can be achieved.

[2]The structure of a line entry of a replay file is as follows: First an identifier of the command category (e.g., `MBASE`) is given. Then follows the command to be executed (e.g., `IMPORT-ASS`) and a sequence of parameters (e.g., `RAT-CRITERION`). The parameter information "`default`" leaves the choice of the parameter to ΩMEGA.

2.3 Interactive Island Proof Development in ΩMEGA

The $\sqrt{2}$-problem can also be solved interactively in ΩMEGA along the lines of the previously given textbook proof (cf. the introduction of Section 2). Due to space restrictions we cannot show the proof development using ΩMEGA's graphical user interface $\mathcal{LOUI}$, but the more cumbersome command line interface of the emacs editor. Otherwise we would have to show a $\mathcal{LOUI}$ screen shot for every user interaction. Thus, the following presentation gives an insufficient impression of the interaction with ΩMEGA, which is in the style of the final island proof plan in $\mathcal{LOUI}$ and *P.rex* as shown in Fig. 1 and 2.

For every command we show both the newly introduced proof nodes and the previously open proof nodes that are closed by the command. The input to ΩMEGA (entered after the `OMEGA` prompt) and its output are given in `typewriter` font.

We present the steps of the proof in a linearized style and therefore call proof nodes also *proof lines*. In the following, we shall write proof lines as $L \quad (\Delta) \vdash \varphi \quad \mathcal{R}$, where L is a unique label, $(\Delta) \vdash \varphi$ denotes that the formula φ can be derived from the formulae whose labels are in the list Δ, and $\mathcal{R}$ is the justification for this derivation of φ from Δ by naming the used inference rule, tactic, or method along with parameters and premises.

Step 0 We start by loading the theory `REAL`, in which the problem is declared.

```
OMEGA: load-problems real
;;; Rules loaded for theory REAL.
;;; Theorems loaded for theory REAL.
;;; Tactics loaded for theory REAL.
;;; Methods loaded for theory REAL.
;;; Strategies loaded for theory REAL.
```

Now, we set the focus on our problem and declare some constant symbols, which we shall use later.

```
OMEGA: prove sqrt2-not-rat
Changing to proof plan SQRT2-NOT-RAT-1
SQRT2-NOT-RAT () |- (NOT (RAT (SQRT 2)))                                    OPEN

OMEGA: declare (constants (m num) (n num) (k num))
```

Step 1: We prove the goal indirectly, that is, we use the inference rule `noti`.

```
OMEGA: noti
NEGATION (NDLINE) A negated line: [SQRT2-NOT-RAT]
FALSITY (NDLINE) A falsity line: [()]

L1 (L1)           |- (RAT (SQRT 2))                                          HYP
L2 (L1)           |- FALSE                                                  OPEN
SQRT2-NOT-RAT () |- (NOT (RAT (SQRT 2)))                                NOTI: (L2)
```

Step 2: We load from the database the theorem `RAT-CRITERION`, which 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 beside 1 (as the formalization of `common-divisor` states, cf. [Benzmüller *et al.*, 2002]). As a side effect, the newly introduced proof line containing that theorem is implicitly added to the hypotheses lists of all other proof lines. The retrieval of this theorem and its application in the appropriate context is non-trivial, since the only syntactical criterion that can be exploited to restrict the potentially large set of applicable theorems is the predicate symbol `RAT`. Automatic acquisition of theorems is a challenging problem and requires a combination of syntax-oriented and semantics-oriented tools. First ideas in this direction are presented in [Benzmüller *et al.*, 2003], assertion application based on resolution is described in [Vo *et al.*, 2003]. However, dynamic retrieval of appropriate assertions from a very large data base with mathematical knowledge is unsolved.

```
OMEGA: import-ass rat-criterion

RAT-CRITERION (RAT-CRITERION) |- (FORALL-SORT ([X].                          THM
                                   (EXISTS-SORT ([Y].
                                     (EXISTS-SORT ([Z].
                                       (AND (= (TIMES X Y) Z)
                                            (NOT (EXISTS-SORT ([D].
                                                   (COMMON-DIVISOR Y Z D))
                                                   INT))))
                                       INT))
                                     INT))
                                   RAT)
```

Step 3: We eliminate the sorted universal quantifier by instantiating its variable `X` with $\sqrt{2}$. This step is again non-trivial. A naive approach to automation, however, is to identify and subsequently instantiate terms of appropriate sort occurring in the proof context.

```
OMEGA: foralle-sort
UNIV-LINE (NDLINE) Universal line: [RAT-CRITERION]
LINE (NDLINE) A line: [()]
TERM (TERM) Term to substitute: (sqrt 2)
SO-LINE (NDLINE) A line with sort: [L1]
```

```
L3 (L1) |- (EXISTS-SORT ([DC-248].                    FORALLE-SORT: ((SQRT 2))
             (EXISTS-SORT ([DC-251].                        (RAT-CRITERION L1)
               (AND (= (TIMES (SQRT 2) DC-248) DC-251)
                    (NOT (EXISTS-SORT ([DC-255].
                           (COMMON-DIVISOR DC-248 DC-251 DC-255))
                           INT))))
               INT))
             INT)
```

Step 4: We eliminate the two sorted existential quantifiers by introducing the constants n and m. Since the quantifiers are soft-sorted, this introduces the additional information that n and m are integers. The newly introduced hypotheses L4 and L5, which express this sort information, are decomposed right away.

```
OMEGA: mexistse-sort*
CONCLINE (NDLINE) Conclusion Line.: [L2]
EXLINE (NDLINE) An existentially quantified line: [L3]
SUBGOAL (NDLINE) Subgoal Line.: [()]
PARAMETER (TERMSYM-LIST) Termsym List.: [(dc-2481 dc-2511)](n m)

L4  (L4)       |- (AND (INT N)                                              HYP
                      (EXISTS-SORT ([DC-251].
                        (AND (= (TIMES (SQRT 2) N) DC-251)
                             (NOT (EXISTS-SORT ([DC-255].
                                    (COMMON-DIVISOR N DC-251 DC-255))
                                    INT))))
                        INT))
L6  (L4)       |- (INT N)                                        ANDEL: (L4)
L5  (L5)       |- (AND (INT M)                                           HYP
                      (AND (= (TIMES (SQRT 2) N) M)
                           (NOT (EXISTS-SORT ([DC-255].
                                  (COMMON-DIVISOR N M DC-255))
                                  INT))))
L8  (L5)       |- (INT M)                                        ANDEL: (L5)
L9  (L5)       |- (AND (= (TIMES (SQRT 2) N) M)                  ANDER: (L5)
                      (NOT (EXISTS-SORT ([DC-255].
                             (COMMON-DIVISOR N M DC-255))
                             INT)))
L10 (L4 L5 L1) |- FALSE                                                  OPEN
L2  (L1)       |- FALSE                    Existse-Sort*-m: ((N M)) (L3 L10)
```

Step 5: Line L9 is further decomposed:

```
OMEGA: ande
CONJUNCTION (NDLINE) Conjunction to split: [L9]
LCONJ (NDLINE) Left conjunct: [()]
RCONJ (NDLINE) Right conjunct: [()]

L11 (L5) |- (= (TIMES (SQRT 2) N) M)                              ANDE: (L9)
L12 (L5) |- (NOT (EXISTS-SORT ([DC-255].                          ANDE: (L9)
                  (COMMON-DIVISOR N M DC-255)) INT))
```

Step 6: While the previous five steps were essentially canonical, we shall now start the island approach to sketch the refutation argument. First, we need some calculations to infer $2n^2 = m^2$ from $\sqrt{2}n = m$. To do so, we use the tactic ISLAND-TACTIC, which allows us to insert arbitrarily large

steps into our proof. The correctness of these steps is checked later when ISLAND-TACTIC is expanded. Note that we specify the premises we want to employ. This is important information because if we specify premises that are too weak, this may have the effect that the island gap cannot be successfully closed later on.

```
OMEGA: island-tactic
CONC (NDLINE) Conclusion of step: nil
PREMS (NDLINE-LIST) Premises of step: (L11 L6 L8)
PARAM (TERM) Formula of Conclusion: (= (times 2 (power n 2)) (power m 2))

L13 (L4 L5) |- (= (TIMES 2 (POWER N 2)) (POWER M 2)) ISLAND-TACTIC: (L11 L6 L8)
```

Step 7: Next, we infer from $2n^2 = m^2$ that m^2 is even...

```
OMEGA: island-tactic nil (L13 L6 L8) (evenp (power m 2))

L14 (L4 L5) |- (EVENP (POWER M 2))                  ISLAND-TACTIC: (L13 L6 L8)
```

Step 8: ... and therefore m is even, too.

```
OMEGA: island-tactic nil (L14 L8) (evenp m)

L15 (L4 L5) |- (EVENP M)                              ISLAND-TACTIC: (L14 L8)
```

Step 9: Next, we unfold[3] the definition of EVENP.

```
OMEGA: defn-expand
LINE (NDLINE) Line to be rewritten: [RAT-CRITERION]L15
DEFINITION (THY-ASSUMPTION) Definition to be expanded: [EVENP]
POSITION (POSITION) Position of occurrence: [(0)]

L16 (L4 L5) |- (EXISTS-SORT ([DC-263].                              DefnE: (L15)
                (= M (TIMES 2 DC-263))) INT)
```

Step 10: As before, we eliminate the sorted existential quantifier by introducing the constant k. Again, the information that k is an integer is added automatically.

```
OMEGA: mexistse-sort*
CONCLINE (NDLINE) Conclusion Line.: [L10]
EXLINE (NDLINE) An existentially quanitified line: [L3]L16
SUBGOAL (NDLINE) Subgoal Line.: [()]
PARAMETER (TERMSYM-LIST) Termsym List.: [(dc-2631)](k)

L17 (L17)          |- (AND (INT K) (= M (TIMES 2 K)))                    HYP
L18 (L17)          |- (INT K)                                    ANDEL: (L17)
L19 (L17)          |- (= M (TIMES 2 K))                          ANDER: (L17)
L20 (L17 L4 L5 L1) |- FALSE                                          OPEN
L10 (L4 L5 L1)     |- FALSE                  Existse-Sort*-m: ((K)) (L16 L20)
```

[3]The folding and unfolding of definitions have historically been called definition contraction (**defn-contract**) and expansion (**defn-expand**) in ΩMEGA.

Step 11: Now, we can go on with our calculation using the ISLAND-TACTIC. By inserting $2k$ for m in $2n^2 = m^2$ we obtain $n^2 = 2k^2$.

```
OMEGA: island-tactic nil (L19 L13 L6 L8 L18)
                         (= (power n 2) (times 2 (power k 2)))

L21 (L4 L5 L17) |- (= (POWER N 2) (TIMES 2 (POWER K 2)))
                                        ISLAND-TACTIC: (L19 L13 L6 L8 L18)
```

Step 12: That means that n^2 is even...

```
OMEGA: island-tactic nil (L21 L6 L18) (evenp (power n 2))

L22 (L5 L4 L17) |- (EVENP (POWER N 2))              ISLAND-TACTIC: (L21 L6 L18)
```

Step 13: ... and so is n.

```
OMEGA: island-tactic nil (L22 L6) (evenp n)

L23 (L17 L5 L4) |- (EVENP N)                          ISLAND-TACTIC: (L22 L6)
```

Step 14: Since both n and m are even, they have a common divisor, namely 2.

```
OMEGA: island-tactic nil (L15 L23 L6 L8) (common-divisor n m 2)

L24 (L17 L4 L5) |- (COMMON-DIVISOR N M 2)        ISLAND-TACTIC: (L15 L23 L6 L8)
```

Step 15: This proves our contradiction and we are done.

```
OMEGA: island-tactic L20 (L12 L24) false

L20 (L17 L4 L5 L1) |- FALSE                           ISLAND-TACTIC: (L12 L24)
```

A verbal presentation of this proof is given in Fig. 2.

2.4 Closing the Gaps

The application of ISLAND-TACTIC does not necessarily result in an automatically verifiable logic-level proof, as filling the gaps can become a challenging task in its own right.

We shall now describe the (semi-automated) task of closing the gaps between the islands in our case study, which leads to a verifiable proof object at the logic level. ΩMEGA supports this process by providing interfaces to external systems. In particular, we use the theorem prover OTTER and the computer algebra system MAPLE. Although these external systems are essentially capable of closing the gaps in this case, the user still has to call them "in the right way" and to provide missing information. For instance, the user has to decide which system he wants to employ, he has to

load additional theorems from the database, and he has to manually unfold some definitions.

The first step when dealing with ISLAND-TACTIC is always to expand its application and we present this step only for the first application of ISLAND-TACTIC. By expanding the tactic its conclusion node becomes open (i.e., unjustified) again and all its premises are now support nodes, that is, the open node is supposed to be derivable from these support nodes. Moreover, theorems and axioms imported from the database automatically become support nodes as well.

Note that the expansion of a tactic application in ΩMEGA can result in proof segments that again contain tactic applications. Thus, expanding a tactic down to the ND level is a recursive process over several levels. We call the recursive process over all necessary levels until an ND-level proof is reached the *full expansion of a tactic*, whereas with *expansion of a tactic* we mean only the direct one-level expansion.

Proving L20 (n and m were supposed to have no common divisor but they actually do have 2 as a common divisor, hence contradiction): Line L20 is justified by an application of ISLAND-TACTIC to the premises L12 and L24. When we expand L20, it becomes open again and its support nodes specify that RAT-CRITERION, L12 and L24 can be used to close it (indeed, RAT-CRITERION is not necessary as we shall see later).

```
OMEGA: expand-node L20
Expanding the node L20 ...

L20 (L17 L4 L5 L1) |- FALSE                                              OPEN

OMEGA: show-supports L20
RAT-CRITERION L12 L24

L12 (L5)        |- (NOT (EXISTS-SORT ([DC-255].                    ANDE: (L9)
                         (COMMON-DIVISOR N M DC-255))
                         INT))
L24 (L17 L4 L5) |- (COMMON-DIVISOR N M 2)        ISLAND-TACTIC: (L15 L23 L6 L8)
```

Although the formulae involved are in first-order logic, OTTER fails to prove L20 with these supports.

```
OMEGA: call-otter-on-node L20
 Normalizing ...
 Calling otter process 27411 with time resource 10sec .
 otter Time Resource in seconds: 10sec
 Search stopped because sos empty.
 Parsing Otter Proof ...
 OTTER HAS FAILED TO FIND A PROOF
```

OTTER fails because one premise is missing, which is not inferable from the context, namely that 2 is an integer. Thus, we speculate this statement as a lemma for L20. This creates the new line L25, which is added as support for L20. We can prove L25 directly with the tactic WELLSORTED.

```
OMEGA: lemma L20 (INT 2)

L25 (L17 L4 L5 L1) |- (INT 2)                                          OPEN

OMEGA: wellsorted l25 ()

L25 (L17 L4 L5 L1) |- (INT 2)                                  WELLSORTED: ()
```

We apply OTTER again to L20, and this time it succeeds. TRAMP automatically translates the OTTER proof to an ND proof at the more abstract assertion level [Huang, 1994].

```
OMEGA: call-otter-on-node L20
 Normalizing ...
 Calling otter process 27554 with time resource 10sec .
 otter Time Resource in seconds: 10sec
 -------- PROOF --------
 Search stopped by max_proofs option.
 Parsing Otter Proof ...
 OTTER HAS FOUND A PROOF
 Creating Refutation-Graph ...
 Translating ...
 Translation finished!

L12 (L5)           |-  (NOT (EXISTS-SORT ([DC-255].                 ANDE: (L9)
                             (COMMON-DIVISOR N M DC-255))
                            INT))
L24 (L17 L4 L5)    |- (COMMON-DIVISOR N M 2)     ISLAND-TACTIC: (L15 L23 L6 L8)
L28 (L5)           |- (NOT (EXISTS [DC-97457]                     DEFNE: (L12)
                           (AND (INT DC-97457)
                                (COMMON-DIVISOR N M DC-97457))))
L30 (L17 L4 L5)    |- (NOT (INT 2))                        ASSERTION: (L28 L24)
L25 (L17 L4 L5 L1) |- (INT 2)                                WELLSORTED: ()
L31 (L1 L17 L4 L5) |- FALSE                                    NOTE: (L25 L30)
L29 (L17 L4 L5 L1) |- FALSE                                     WEAKEN: (L31)
L20 (L17 L4 L5 L1) |- FALSE                                     WEAKEN: (L29)
```

This proves that L20 is derivable from L12 and L24 at a lower level of abstraction. However, the nodes L30 and L25 are still not at the ND level, but justified by tactics. To verify these steps we have to fully expand them also, which works automatically and results in an ND-level subproof for L25 that consists of 13 steps and an ND-level subproof for L30 with 40 steps.

Proving L15 (m^2 is even implies m is even) and L23 (n^2 is even implies n is even): In order to close the gap between L15 and its premises L14 and L8 and between L23 and its premises L22 and L6 we need the theorem SQUARE-EVEN from the database. With this theorem OTTER manages to prove L15 and L23, respectively, and TRAMP outputs the corresponding assertion level proofs, which consist essentially of an application of the assertion SQUARE-EVEN. When fully expanded, the subproofs, that is, the ND-level proof deriving L15 from L14 and L8 and the ND-level proof deriving L23 from L22 and L6, consist of 6 steps each.

Proving L14 ($2n^2 = m^2$ implies m^2 is even) and L22 ($n^2 = 2k^2$ implies n^2 is even): The proof line L14 is justified by an application of

ISLAND-TACTIC to L13, L6, and L8. OTTER fails to prove L14 with respect to these supports. Indeed, using OTTER to obtain a proof for L14 requires some further steps: (1) we have to unfold the definition of EVENP in L14, and (2) we have to speculate that n^2 is an integer as a lemma for L14, which is added as node L41 to the proof. L41 can then be closed by applying the tactic WELLSORTED with L6 as premise such that afterwards the problem has the following form:

```
L13 (L4 L5) |- (= (TIMES 2 (POWER N 2)) (POWER M 2)) ISLAND-TACTIC: (L11 L6 L8)
L41 (L4 L5) |- (INT (POWER N 2))                          WELLSORTED: (L6)
L40 (L4 L5) |- (EXISTS-SORT ([DC-98774].                             OPEN
                 (= (POWER M 2) (TIMES 2 DC-98774))) INT)
L14 (L4 L5) |- (EVENP (POWER M 2))                          DEFNI: (L40)
```

After applying OTTER successfully to L40, TRAMP provides a proof that derives L40 in 5 steps from L13 and L41. A fully expanded subproof at the ND level consists of 11 steps. When the application of WELLSORTED is fully expanded, L41 is derived from L6 by a subproof consisting of 17 steps.

Closing the gap between L22 and its premises L21, L6, and L18 works similarly. The only difference is that instead of the lemma that n^2 is an integer, the lemma that k^2 is an integer has to be speculated. This lemma can be closed by an application of the tactic WELLSORTED to the node L18 with formula (INT K).

Proving L24 (since n and m are even they have 2 as a common divisor): Before we can exploit OTTER to obtain a proof for the gap between L24 and its supports L15, L23, L6, and L8 we have to unfold some defined concepts and speculate some lemmata. In L24, we have to unfold the defined concept COMMON-DIVISOR, which closes L24 and results in a new open node L59. In L59 we then have to unfold all occurrences of the defined concept DIVISOR, which closes L59 and creates a new open node L60. L60 inherits the supports of L24 via L59. Next, we have to unfold EVENP in the two support nodes L15 and L23, which creates the two new supports L61 and L62 for L60 that contain the formulae resulting from unfolding EVENP. Moreover, for L60 we have to speculate the two lemmata that $1 \neq 2$ and that 2 is an integer, which are introduced as nodes L63 and L64 in the proof.

The open node L60 and its supports can now be proved using OTTER. TRAMP translates OTTER's proof into an ND-level proof that consists of 16 steps. We already proved that 2 is an integer in L25. To prove the second lemma, $1 \neq 2$, is actually not as trivial as it may seem. The numbers 1 and 2 are defined concepts in ΩMEGA and they are more convenient representations of (S ZERO) and (S (S (ZERO))), where S is the successor function. After unfolding 1 and 2 we have to prove that (NOT (= (S ZERO) (S (S ZERO)))) holds. We can prove this statement with OTTER, but to do so we need to import the following axioms from the database into the proof: ZERO is a natural number, the successor of a natural number is again a natural

number, the successor function is injective, and ZERO has no predecessor. Then TRAMP provides as output an assertion-level proof that consists of 8 steps. When fully expanded, the subproof for L63 consists of 28 steps.

Proving L13 ($\sqrt{2}n = m$ implies $2n^2 = m^2$) and L21 ($m = 2k$ and $2n^2 = m^2$ imply $n^2 = 2k^2$): So far, we always used OTTER and TRAMP to expand applications of ISLAND-TACTIC. Indeed, automated theorem proving was the right choice for this task, since all subproofs essentially rely on some theorems or definitions, which — after being imported from the database — can be used by OTTER to derive a proof.

In contrast, the steps deriving L13 from L11, L6, and L8 as well as deriving L21 from L19, L13, L6, L8, and L18 represent algebraic computations, for which ATPs are not particularly well suited. In ΩMEGA, the tactic BYCOMPUTATION employs the computer algebra system MAPLE to check whether an equation containing arithmetic expressions follows from a set of other equations. To do so, BYCOMPUTATION passes all equations to MAPLE and calls MAPLE's is function to check whether the conclusion equation holds assuming the premise equations. This tactic succeeds when applied to L13 and L21. For example, it closes L13 as follows:

```
L11 (L5)    |- (= (TIMES (SQRT 2) N) M)                          ANDE: (L9)
L13 (L4 L5) |- (= (TIMES 2 (POWER N 2)) (POWER M 2))      BYCOMPUTATION: (L11)
```

Currently, the tactic BYCOMPUTATION can only be partially expanded into an ND-level proof, namely for computations with polynomials. We are working on extensions of SAPPER to cover more cases, such as the equation above. The two applications of BYCOMPUTATION that justify L13 and L21 are therefore only "verified" by MAPLE, but not automatically by an ND-level proof.[4]

Result: When fully expanded, the island proof consists of 282 nodes, where the nodes L13 and L21 are justified by the unexpanded tactic BYCOMPUTATION. Hence, ΩMEGA's checker verifies that the proof is correct modulo the correctness of these computations. Fully expanding and checking the proof takes about 300 seconds on a 1.8 GHz Pentium III machine with 512 MB RAM running LINUX.

2.5 Automation of Proof Tasks

As described in detail in Sections 2.3 and 2.4, the proof of the $\sqrt{2}$-problem is constructed in two phases: First an abstract proof is outlined, in which

[4] Applications of mathematical systems for simplification make their efficient computation available to the construction of the top-level proof plan. The time-consuming verification is left to the expansion mechanism.

islands are coupled by tactics without care for the detailed logical dependencies between them. Next, the gaps between the islands are closed in the second phase with detailed ND-proof segments. So far both tasks require fairly detailed user interaction: at the abstract level the user has to provide the islands (and to apply some tactics); and for the expansion into the logic level he has to speculate the *right* lemmata, import the *right* theorems, and unfold the *right* definitions up to the *right* level.

The main research in ΩMEGA is currently to better automate these tasks. To this end, we examine two approaches:

First, proof planning with MULTI [Meier, 2003] is the means to automatically create island proofs at a user-friendly level of abstraction. In Section 3 we shall show the knowledge-engineering process that generalizes the main ideas underlying the interactive proof into corresponding proof methods and control knowledge for MULTI. Using this knowledge MULTI is capable of automatically planning the proofs for theorems of the generalized $\sqrt[i]{l}$-problems, that is, whether $\sqrt[i]{l}$ is irrational. We also have expansion tactics for the employed methods such that the expansions of the proof plans for $\sqrt[i]{l}$-problems can be done automatically and we can in fact fully expand and verify them with our proof checker (again, modulo the computer algebra system computations).

Second, semi-automated agent-based reasoning with Ω-ANTS is a means to obtain logic-level proofs for the interactively generated island proof plans with fewer user interactions. Expanding and closing island gaps is often more challenging than in proof planning, since there is no knowledge immediately available. In general, the expansion of the tactic `ISLAND-TACTIC` corresponds to a completely new theorem, which may be solved by providing more specific islands. The idea is that after some hierarchical decomposition the gaps become so small that they can be filled in automatically. To obtain a better degree of automation for closing the island gaps, we are working on the following ideas:

- Ω-ANTS "agentifies" methods, tactics, calculus rules, and heterogeneous external reasoners in the sense that these search proactively for their respective applicability. It should be possible to link the `ISLAND-TACTIC` with appropriate Ω-ANTS agents such that these autonomously and cooperatively try to close the gaps in the background while the user works on the next island steps. In case Ω-ANTS cannot close a gap automatically, the user will be informed and he may rethink the island step or he may provide further knowledge that can be used by Ω-ANTS.

- We are currently also examining the Ω-ANTS mechanism as a mediator between a knowledge base and proof planning (first results are reported in [Benzmüller *et al.*, 2003]). The mediator supports the idea

of semantically guided retrieval of mathematical knowledge (theorems, definitions) from the database MBase.

- The speculation of suitable lemmata can be supported by a model generator. For instance, when applying the model generator Mace or Satchmo to the failing proof attempt with Otter for `L20` a counter-model is generated, in which 2 is not an integer. A general mechanism that employs model generation for the speculation of missing lemmata (such as the one asserting that 2 is an integer) is certainly possible and promising.

- A better support for unfolding definitions is to adapt Bishop and Andrew's selective unfolding mechanism [Bishop and Andrews, 1998] to our proof planning context.

3 PROOF PLANNING THE $\sqrt{2}$-PROBLEM

The user has to apply the island tactic eight times in the proof discussed in Section 2 (namely in steps 6, 7, 8, 11, 12, 13, 14, and 15). These are the crucial and creative steps that provide the essential idea of this proof.

Now, the question is: Can we find these creative steps automatically? The answer is yes, as we shall show in this section. However, while we can answer the question in the affirmative, not every reader may be convinced that this is really the final answer, as our solution touches upon a subtle point, which opens the Pandora Box of critical issues in the paradigm of proof planning [Bundy, 2002]. It is easy to write some specific methods, which perform just the steps in the interactively found proof and then call the proof planner Multi to fit the methods together into a proof plan for our problem. This, of course, shows nothing of substance: Just as we could write down all the definitions and theorems required for the problem in first-order predicate logic and hand them to a first-order prover such as Otter,[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, 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 such that we can solve, for example, $\sqrt[j]{l}$-problems for arbitrary natural numbers j and l. However, as we shall discuss in Section 3.4, this is certainly not the end of the story.

[5] As it was done when tackling the $\sqrt{2}$-problem with Otter; see [Wiedijk, 2002] for the original Otter case study and [Benzmüller *et al.*, 2002] for its replay with ΩMEGA.

3.1 The Knowledge Acquisition

In order to find a general approach for $\sqrt[j]{l}$-problems for arbitrary natural numbers j and l, we first 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, that together form one planner strategy for this kind of problems. We shall now discuss the acquired methods and control rules.

The essential ideas of the proof in Section 2 are as follows:

(1) Use the theorem `RAT-CRITERION` and construct an indirect proof.

(2) In order to derive the contradiction show that the two constants (existential variables) 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 $m = 2 \cdot k$).

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, which generalizes `SQUARE-EVEN`, is now available in ΩMEGA's database:

```
(th~deftheorem POWER-PRIME-DIVISOR (in integer)
  (conclusion (forall-sort (lam (n num)
                (forall-sort (lam (d num)
                  (forall-sort (lam (x num)
                    (implies (prime-divisor d (power x n))
                             (prime-divisor d x)))
                    INT))
                  INT))
                NAT)))
```

To realize idea (1) the planner MULTI has to decide to try 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 overfitting methods we decided to employ already existing methods from other domains: NotI-M (prove by contradiction), 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 by the method. 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 formula $\sqrt[j]{l}$.

Idea (2) is realized with the method ContradictionCommonDivisor-M. When MULTI tries to apply the method it searches first for a proof line that states that two terms t_1, t_2 have no common divisor, and second for two proof lines that state that t_1 and t_2, respectively, have a divisor d. This method is not guided by control rules, but MULTI tries to apply it to some derived proof lines in each planning cycle.

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 the 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 returns a proof line whose formula 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 proof lines that state 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 returns the proof line $t = p_{1'} \cdot \ldots \cdot p_{n'} \cdot c'$ 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 proof lines that state 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.

3.2 Applying MULTI *to the* $\sqrt{2}$*-Problem*

MULTI constructs now a proof plan as follows:

First, it applies the methods MAssertion-M, NotI-M, ExistsE-Sort-M, AndE-M, and TransFormEquation-M, in order to apply the theorem `RAT-CRITERION`, to establish an indirect proof, and to decompose existing existentially quantified formulae or conjunctions.[6]

```
L3 (L3)          |- (EXISTS-SORT ([DC-626].                                  HYP
                      (EXISTS-SORT ([DC-629].
                        (AND (= (TIMES (SQRT 2) DC-626) DC-629)
                             (NOT (EXISTS-SORT ([DC-633].
                                    (COMMON-DIVISOR DC-626 DC-629 DC-633))
                                    INT))))
                        INT))
                      INT)
L5 (L5)          |- (AND (INT CONST1)                                        HYP
                        (EXISTS-SORT ([DC-629].
                          (AND (= (TIMES (SQRT 2) CONST1) DC-629)
                               (NOT (EXISTS-SORT ([DC-633].
                                      (COMMON-DIVISOR CONST1 DC-629 DC-633))
                                      INT))))
                          INT))
L7 (L5)          |- (INT CONST1)                                  AndE-m: (L5)
L8 (L5)          |- (EXISTS-SORT ([DC-629].                       AndE-m: (L5)
                      (AND (= (TIMES (SQRT 2) CONST1) DC-629)
                           (NOT (EXISTS-SORT ([DC-633].
                                  (COMMON-DIVISOR CONST1 DC-629 DC-633))
                                  INT))))
                      INT)
L9 (L9)          |- (AND (INT CONST2)                                        HYP
                        (AND (= (TIMES (SQRT 2) CONST1) CONST2)
                             (NOT (EXISTS-SORT ([DC-633].
                                    (COMMON-DIVISOR CONST1 CONST2 DC-633))
                                    INT))))
L11 (L9)         |- (INT CONST2)                                  AndE-m: (L9)
L12 (L9)         |- (AND (= (TIMES (SQRT 2) CONST1) CONST2)       AndE-m: (L9)
                        (NOT (EXISTS-SORT ([DC-633].
                               (COMMON-DIVISOR CONST1 CONST2 DC-633))
                               INT)))
L13 (L9)         |- (= (TIMES (SQRT 2) CONST1) CONST2)           AndE-m: (L12)
L14 (L9)         |- (NOT (EXISTS-SORT ([DC-633].                 AndE-m: (L12)
                           (COMMON-DIVISOR CONST1 CONST2 DC-633))
                           INT)))
L15 (L9)         |- (= (TIMES 2 (POWER CONST1 2))    TRANSFORMEQUATION-M: (L13)
                      (POWER CONST2 2))
```

[6]Actually, MAssertion-M, which applies `RAT-CRITERION`, also introduces the open line `L1`, which is closed by the method Reflex-M. This line results from variable bindings internal to the theorem application process. It states that the variable `X1`, which corresponds to the universally quantified variable in `RAT-CRITERION`, has to be bound to the term `(SQRT 2)`. This binding has already been applied to the rest of the proof (e.g., in `L2`).

```
L10 (L9 L5 L3)   |- FALSE                                                    OPEN
L6 (L5 L3)       |- FALSE                                 Existse-Sort-m: (L8 L10)
L4 (L3)          |- FALSE                                  Existse-Sort-m: (L3 L6)
L2 ()            |- (NOT (EXISTS-SORT ([DC-626].                  NOTI-M: (L4)
                          (EXISTS-SORT ([DC-629].
                            (AND (= (TIMES (SQRT 2) DC-626) DC-629)
                                 (NOT (EXISTS-SORT ([DC-633].
                                        (COMMON-DIVISOR DC-626 DC-629 DC-633))
                                        INT))))
                                 INT))
                                INT))
L1 ()            |- (= X1 (SQRT 2))                                      REFLEX-M
SQRT2-NOT-RAT () |- (NOT (RAT (SQRT 2)))                  MAssertion-M: (L2)
```

Next, using the equation $2 \cdot const1^2 = const2^2$ in line L15[7] the methods PrimeFacsProduct-M and PrimeDivPower-M derive that $const2$ has the prime divisor 2.

```
L16 (L5 L9) |- (PRIME-DIVISOR 2 (POWER CONST2 2))    PRIMEFACS-PRODUCT-M: (L15)
L17 (L5 L9) |- (PRIME-DIVISOR 2 CONST2)                 PRIMEDIV-POWER-M: (L16)
```

Then, CollectDivs-M computes a representation for $const2$ with respect to line L17. Since CollectDivs-M introduces a new hypothesis in line L19 it reduces also the open line L10 to the new open line L20, which also contains the new hypothesis.

```
L19 (L19)          |- (AND (INT CONST3) (= CONST2 (TIMES 2 CONST3)))        HYP
L21 (L19)          |- (INT CONST3)                                    AndE-m: (L19)
L22 (L19)          |- (= CONST2 (TIMES 2 CONST3))                     AndE-m: (L19)


L20 (L19 L9 L5 L3) |- FALSE                                                OPEN
L10 (L9 L5 L3)     |- FALSE                                 COLLECTDIVS-M: (L20 L17)
```

Next, the methods =Subst-m and TransFormEquation-M derive with the new representation for $const2$ the equation in line L25.

```
L24 (L19 L9) |- (= (TIMES 2 (POWER CONST1 2))                  =subst-m: (L15 L22)
                  (POWER (TIMES 2 CONST3) 2))
L25 (L9 L19) |- (= (POWER CONST1 2)                    TRANSFORMEQUATION-M: (L24)
                  (TIMES 2 (POWER CONST3 2)))
```

Then, with respect to this equation the methods PrimeFacsProduct-M and PrimeDivPower-M derive that $const1$ has the prime divisor 2.

```
L26 (L5 L19 L9) |- (PRIME-DIVISOR 2 (POWER CONST1 2))
                                                    PRIMEFACS-PRODUCT-M: (L25)
L27 (L9 L19 L5) |- (PRIME-DIVISOR 2 CONST1)            PRIMEDIV-POWER-M: (L26)
```

Finally, ContradictionCommonDivisor-M closes the open line L20 and MULTI terminates with the final line:

```
L20 (L19 L9 L5 L3) |- FALSE         CONTRADICTIONCOMMONDIVISOR-M: (L14 L27 L17)
```

[7] Here, the automatically generated constants $const1$ and $const2$ replace the constants n and m, respectively, in the interactive proof given in Section 2.

This automatically generated proof plan required less than four seconds CPU time on a 1.8 GHz Pentium III machine with 512 MB RAM running LINUX. A considerable amount of time was required to call the external computer algebra system MAPLE twice within the applications of the method PrimeFacsProduct-M. The complete proof plan can then be passed to *P.rex*, which produces a natural language presentation of the proof as given in Fig. 3. This concludes the first phase, that is, the automated construction of a proof plan, and we shall now look at the second phase, that is, the expansion of this proof plan to an ND-level proof.

3.3 Expansion

All methods used in ΩMEGA have as part of their specification the knowledge on how to expand. A method contains a schematic proof segment that shows how to derive its conclusions from its premises using tactics and rules. When an application of the method is expanded, then the proof schema is instantiated and introduced into the proof plan justifying the conclusions of the method from its premises at a lower level of abstraction. Note that the methods only specify their direct expansion level, the recursive expansion is part of the overall expansion process.

As an example, consider the expansion of PrimeDivPower-M and PrimeFacsProduct-M, which demonstrate in particular how additional theorems and sort statements are found and treated in the expansion process.

The expansion schema for PrimeDivPower-M is as follows:

```
(DECL-CONTENT
    (S0 () (PRIME-DIVISOR N (POWER A M)))
    (S1 () (NAT M)                                          ("WELLSORTED" ()))
    (S2 () (INT N)                                          ("WELLSORTED" ()))
    (S3 () (INT A)                                        ("WELLSORTED" (IA)))
    (S4 () (IMPLIES (PRIME-DIVISOR N (POWER A M))
                    (PRIME-DIVISOR N A))     ("FORALLE-SORT*" (PPD S1 S2 S3)))
    (S5 () (PRIME-DIVISOR N A)                            ("IMPE" () (S4 S0))))
```

This proof segment represents a proof at tactic level. It shows how to derive line S5 (the conclusion of the method) from line S0 (the premise of the method). This proof segment uses the `Power-Prime-Divisor` theorem, whose incorporation into the proof plan during the expansion of an application of PrimeDivPower-M is specified in the method by a so-called expansion computation (see [Meier *et al.*, 2002a]). The `Power-Prime-Divisor` theorem is abbreviated in the proof segment as PPD and it is used to derive S4. To apply the theorem we have to establish some sort statements for m, n, and a in the lines S0, S1, and S2, respectively. Since m and n occur in the application part of the method, there is a concrete natural number and a concrete integer, respectively. We can prove these sort statements with

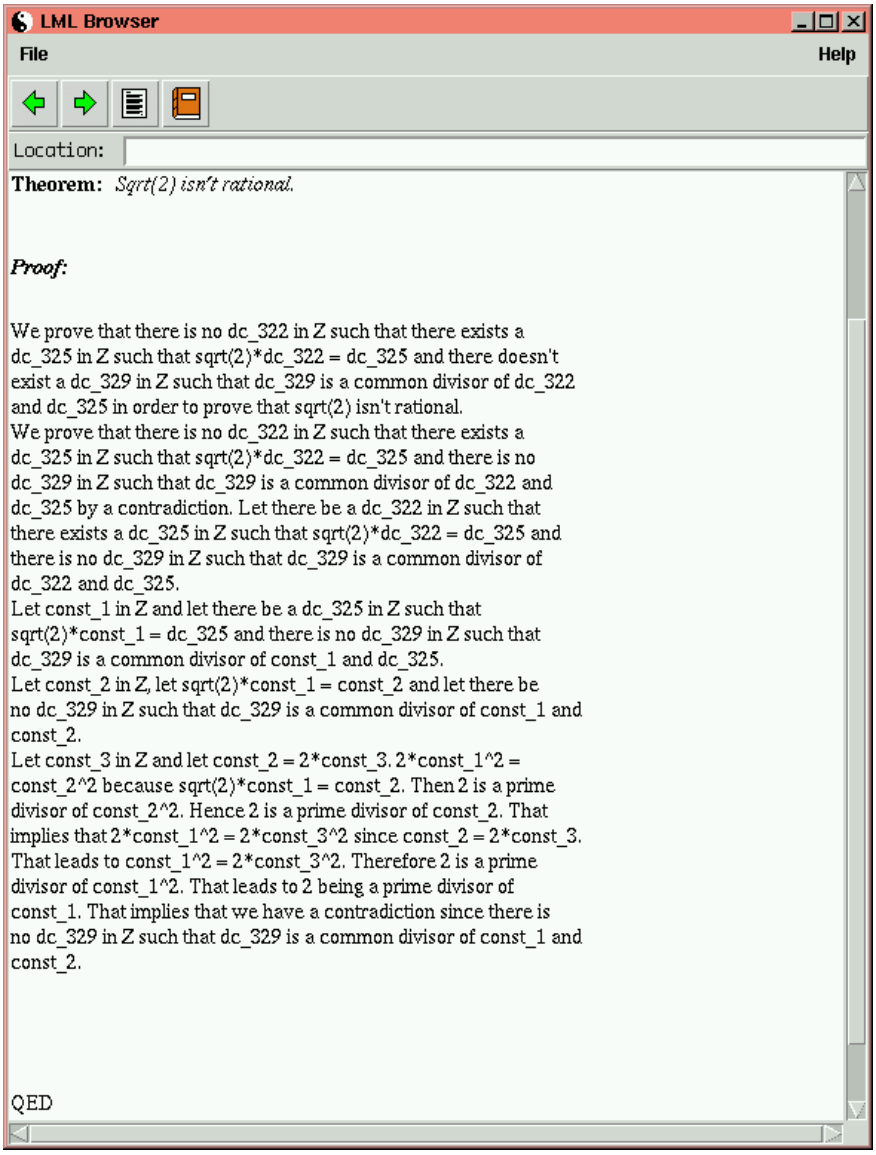

Figure 3. Natural language proof presentation of the planned proof by *P.rex*.

the tactic WELLSORTED. The variable a does not represent a concrete number, thus we have to derive from existing hypotheses that a is an integer. This derivation is part of the applicability check of the method and these hypotheses are represented above as IA in S3.

PrimeDivPower-M derives line L17. When it is expanded, the following proof segment is added to the proof plan:

```
L16 (L5 L9) |- (PRIME-DIVISOR 2 (POWER CONST2 2))    PRIMEFACS-PRODUCT-M: (L15)
L28 ()      |- (NAT 2)                                        WELLSORTED: ()
L29 ()      |- (INT 2)                                        WELLSORTED: ()
L30 (L9)    |- (INT CONST2)                                 WELLSORTED: (L11)
L31 (L9)    |- (IMPLIES                        FORALLE-SORT*: (PPD L28 L29 L30)
                 (PRIME-DIVISOR 2 (POWER CONST2 2))
                 (PRIME-DIVISOR 2 CONST2))
L17 (L5 L9) |- (PRIME-DIVISOR 2 CONST2)                        IMPE: (L31 L16)
```

Here, PPD stands for the Power-Prime-Divisor theorem. This theorem is now inserted into the proof plan as:

```
PPD (PPD) |-  (FORALL-SORT ([N].                                          THM
                (FORALL-SORT ([D].
                  (FORALL-SORT ([X]. (IMPLIES (PRIME-DIVISOR D (POWER X N))
                                              (PRIME-DIVISOR D X)))
                    INT))
                  INT))
                NAT)))
```

WELLSORTED and FORALLE-SORT* are tactics, whose applications can be expanded automatically. If all of this is done, the resulting logic-level proof segment derives L17 from L16 in 19 steps.

The proof schema of PrimeFacsProduct-M specifies that the conclusion of the formula, line S2, is derived by an application of the tactic EXPAND--PRIMEFACSPRODUCT to the premise of the method in line S1:

```
(DECL-CONTENT
    (S1 () (= A B))
    (S2 () CONJUNCTION        ("EXPAND-PRIMEFACSPRODUCT" () (S1 SORT-PREMS))))
```

Here, CONJUNCTION is a substitute for the actual conjunction of terms (PRIME-DIVISOR P X), which are computed during the application of the method. As opposed to the expansion of a method, which is stated declaratively, the expansion of a tactic is given by a LISP function.

The method PrimeFacsProduct-M derives line L16. When it is expanded, no new proof lines are added, but the justification of L16 is changed:

```
L11 (L9)    |- (INT CONST2)                                    AndE-m: (L9)
L7  (L5)    |- (INT CONST1)                                    AndE-m: (L5)
L15 (L9)    |- (= (TIMES 2 (POWER CONST1 2))         TRANSFORMEQUATION-M: (L13)
                 (POWER CONST2 2))
L16 (L5 L9) |- (PRIME-DIVISOR 2 (POWER CONST2 2))
                                  EXPAND-PRIMEFACS-PRODUCT: (L15 L11 L7)
```

In this case, lines L11 and L7 are mandatory premises for the tactic EXPAND-PRIMEFACSPRODUCT. They provide the necessary sort information.

The expansion of EXPAND-PRIMEFACSPRODUCT works as follows: The first premise of EXPAND-PRIMEFACSPRODUCT is an equation $x = l \cdot y$, and its conclusion is a conjunction of terms (PRIME-DIVISOR P X). Moreover, further premises of EXPAND-PRIMEFACSPRODUCT are necessary sort statements. First, for each conjunct (PRIME-DIVISOR P X) of the conclusion the definition of PRIME-DIVISOR is expanded.

```
(th~defdef prime-divisor (in integer)
  (definition (lam (x num) (lam (y num) (and (divisor x y) (prime x)))))
  (help "The predicate for prime divisors."))
```

This results in two new subgoals, respectively, namely that p is a prime number, formalized as (PRIME P), and that p is a divisor of x, formalized as (DIVISOR P X). By rewriting the term $l \cdot y$ to $p \cdot (r \cdot y)$ (r is computed from l and p by MAPLE), the expansion establishes that $x = p \cdot (r \cdot y)$ holds. Since this is essentially the definition of divisor in ΩMEGA's database, the expansion can derive (DIVISOR P X).

```
(th~defdef divisor (in integer)
  (definition
    (lam (x num)
         (lam (y num)
              (and (and (int x) (int y))
                   (exists-sort (lam (z num) (= y (times x z))) int)))))
  (help "The predicate for integer divisibility."))
```

The sort premises are needed, since the divisor definition requires that y, p, and $r \cdot x$ are of sort integer.

To rewrite $l \cdot y$ as stated above, the expansion has to establish that $l = p \cdot r$ holds, where l, p, and r are concrete numbers. This is currently justified by an application of the tactic BYCOMPUTATION, that is, it is verified by MAPLE. The same holds for the statements (PRIME P) for concrete numbers p. They are also justified by BYCOMPUTATION.

The expansion of EXPAND-PRIMEFACSPRODUCT in turn employs other tactics (e.g., tactics for definition expansion and equality substitution). The recursive expansion of all tactics that prove L16 results in a proof segment that derives L16 from L15, L11, and L7 in 75 steps at the logic level. The proof segment is verified except for the mentioned proof nodes justified by applications of BYCOMPUTATION.

The expansion of all other methods works similarly. Note that the methods CollectDivs-M and ContradictionCommonDivisor-M expand to proof segments that use the tactic OTTER. When this tactic is expanded, it employs OTTER in order to justify the conclusion of the tactic application from its premises and the resulting proof lines are inserted.

Completely expanded, the proof of the $\sqrt{2}$-problem consists of 753 steps. The proof is verified except for the nodes justified by the tactic

BYCOMPUTATION, which is used 6 times, that is, currently we trust MAPLE. Note that the expansion mechanism operates locally and schematically and therefore does not optimize the final proof for its size.

3.4 Discussion

In order to evaluate the appropriateness of our approach we suggest the following three 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?

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 proof plan any interesting problems. As the other extreme case, 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.

The amount of search and the variety of potential proof plans for a given problem are measures for the generality of the methods and also for the appropriateness for tackling the class of problems by planning. If tight control rules or highly specific methods restrict the search to just one branch in the search tree, then the resulting proof plans will merely instantiate a pattern. In this case, a single tactic or method that realizes the proof steps of the underlying pattern is more suitable than planning. The possibility of creating a variety of proof plans with the given methods and control rules is thus an important feature.

In the following, we shall discuss proof planning for $\sqrt[i]{l}$-problems with respect to these three criteria.

(1) The methods Notl-M, ExistsE-Sort-M, AndE-M, =Subst-m, and MAssertion-M of our approach encode logic-level steps (Notl-M, AndE-M, =Subst-m) or tactic steps very close to the logic-level (ExistsE-Sort-M, MAssertion-M). Thus, they are very general, but they do not encode specific domain knowledge, and they are in fact still in the spirit of Gerhard Gentzen's analysis of mathematical proofs [Gentzen, 1935].

PrimeFacsProduct-M and CollectDivs-M encode domain knowledge about integers. PrimeFacsProduct-M encodes the extraction of prime divisors for integers from equations on integers. CollectDivs-M computes different representations for integers, for which some divisors are known. The functionalities of these methods are currently rather restricted and focused on our actual problem domain (e.g., CollectDivs-M could deal also with divisors and not only with prime divisors, PrimeFacsProduct-M could handle a more general class of equations). However, suitably extended, these two methods will be useful for many problem classes dealing with integers.[8]

PrimeDivPower-M applies the theorem `POWER-PRIME-DIVISOR`. This could be done also by the MAssertion-M method. We encoded the application of this theorem into an extra method in order to hide the sort goals. Thus, PrimeDivPower-M is a very specific method just fitted to the particular needs of our problem class. The same holds for TransFormEquation-M, which performs exactly the equation transformations that we need to deal with our problem class.

ContradictionCommonDivisor-M is not specific to our domain and should also be useful for other problem classes dealing with divisors.

The control rules `apply-collectdivs`, `apply-=subst`, and `reject-loop` contain no particular domain knowledge, but force MULTI into the right search branch. Hence, they are useful for any search. Only `apply-ratcriterion` touches a subtle point: this control rule encodes the knowledge that the theorem `RAT-CRITERION` should be applied via the method MAssertion-M and is hence fitted to our particular problem domain. We are currently examining a mediator mechanism between our knowledge base and proof planning (first results are reported in [Benzmüller *et al.*, 2003]), which supports a semantically guided retrieval of theorems and definitions. This approach should replace particular control rules for theorem retrieval (such as `apply-ratcriterion`) by a more general mechanism.

(2) MULTI performs depth-first search, which typically involves backtracking from search branches with no solutions. In our domain and with the described methods and control rules, however, there is no backtracking. Rather, the search consists of all possible transformations and derivations for finding two prime divisors that yield the contradiction.

For instance, when tackling the $\sqrt{6}$-problem, MULTI derives from $6 \cdot n^2 = m^2$ that m has two prime divisors 2 and 3. With respect to these prime divisors it computes the following three representations of m: $m = 2 \cdot a$, $m = 3 \cdot b$, $m = 2 \cdot 3 \cdot c$, and each representation is used to substitute m in the equation $6 \cdot n^2 = m^2$.

[8] To extend the capabilities of the methods we would also have to extend the tactics for the expansion of the methods. Since this is not trivial for both methods, we instead decided for now to implement these "light" versions for which the expansion is fully specified and leave the generality to future work.

Not all proof plans we found are similar to the proof plan for the $\sqrt{2}$-problem. For instance, the proof plan for the $\sqrt{8}$-problem has the following steps: From $8 \cdot n^2 = m^2$ MULTI derives that m has the prime divisor 2. With respect to this prime divisor m can be written as $m = 2 \cdot a$. Substituting m in the initial equation and simplifying the equation yields $2 \cdot n^2 = a^2$. From this equation no prime divisors of n can be derived, which would then yield a contradiction with the prime divisors of m as in the proof plan for the $\sqrt{2}$-problem. Instead, MULTI derives that 2 is a prime divisor of a and computes a new representation for a: $a = 2 \cdot b$. Substituting a with respect to this equation then yields that $n^2 = 2 \cdot b^2$ from which MULTI can derive that n has the prime divisor 2 which yields the contradiction to the fact that m has also the prime divisor 2.

Although the proof plans for different problems can be different we found that they vary only very little. The main variations are with respect to the numbers of "detect prime divisors for c"—"represent c as $c = p \cdot c'$"—"substitute c in prior equations" cycles. In particular, the proof plans for every $\sqrt{p}$-problem, where p is a prime number, looks exactly as the proof plan for the $\sqrt{2}$-problem.

(3) The number of potential proof plans depends on the particular problem. For prime numbers p the proof plan is the same as for the $\sqrt{2}$-problem (modulo instantiations). For numbers l that have several prime divisors there are typically several proof plans that vary with respect to the common divisors.[9]

This concludes the actual descriptions of how the $\sqrt{2}$-problem (respectively its generalization, the $\sqrt[j]{l}$-problem) was solved with ΩMEGA.

What general lessons can we learn from small, albeit typical mathematical challenges of this kind?

1. The devil is in the detail, that is, it is always possible to hide the crucial creative step in some small pre-programmed step and to pretend a level of generality that has not actually been achieved. To evaluate a solution *all* tactics, methods, theorems and definitions have to made explicit.

   In this paper, we have tried to strike a balance and to provide enough information to judge the strength (and weakness) of the current state of the art in the new paradigm of proof planning without providing too many details.

[9]For instance, when tackling $\sqrt{6}$ MULTI derives from $6 \cdot n^2 = m^2$ that m has the prime divisors 2 and 3 and hence m can be represented as $m = 2 \cdot 3 \cdot a$. From this representation MULTI derives that $n^2 = 6 \cdot a^2$. This equation yields that n has the prime divisors 2 and 3. Now, MULTI can use both 2 and 3 to derive a contradiction.

2. The enormous distance between the well-known (top-level) proof of the Pythagorean School, which consists of about a dozen single proof steps in comparison to the final proof at the ND level with 753 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 influential Bourbaki group showed: only 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 [Bourbaki, 1968]: "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 point of the concrete contribution of this paper: Now that we can prove theorems in the $\sqrt[3]{1}$-problem class, the skeptical reader may still ask *So what?* Will this ever lead to a *general* system for mathematical assistance?

   We have demonstrated in [Melis and Siekmann, 1999; Melis, 1998] that the class of ϵ-δ-proofs for limit theorems can indeed be solved with a few dozen mathematically meaningful methods and control rules. Similarly, the domain of group theory with its class of residue theorems can be formalized with even less [Meier and Sorge, 2000; Meier *et al.*, 2001; Meier *et al.*, 2002b], and the crucial general observation is that these methods correspond to the kind of mathematical knowledge a freshman would have to learn to master this level of professionalism.

   Is the same true for $\sqrt[3]{1}$-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 $\sqrt[3]{1}$-problems? Probably not!

4 MATHEMATICAL KNOWLEDGE ENGINEERING

Mathematical knowledge is preserved in books and monographs, but the art of doing mathematics [Polya, 1973; Hadamard, 1944] 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", and they make this implicit and informal mathematical knowledge explicit and formal[10].

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 complex objects is to look at their characteristic properties, for example, their normal forms or some other uniform notation in the respective theory. If there is a characteristic property that distinguishes the two objects then they cannot be equal.

In order to tackle $\sqrt[j]{l}$-problems we have to find a property that distinguishes the integers $l \cdot n^j$ and m^j. Since each integer can be represented as a product of prime numbers, the normal forms in the integer theory are products of primes. If $P[i]$ denotes the prime product of i, then we can write $l \cdot n^j$ and m^j as $P[l] \cdot P[n^j]$ and $P[m^j]$. For instance, for $l = 2$, $j = 2$ this is $2 \cdot P[n^2]$ and $P[m^2]$. A characteristic property distinguishing these two integers is the quantity of prime numbers in the normal form. For $P[n^2]$ and $P[m^2]$ we do not know the exact quantity, but we do know for both that the quantity is even, since each occurrence in $P[x]$ is duplicated in $P[x^2]$. The quantity of prime numbers for 2 is 1, thus, we know that the quantity of prime numbers in $2 \cdot P[n^2]$ is odd, whereas the quantity of prime numbers in $P[m^2]$ is even. Thus, they cannot be equal. Note that for this argument the premise that n and m have no common divisor is not necessary.

The use of normal forms and characteristic properties as in the above argument is a common mathematical principle. For example, in polynomial rings over finite fields irreducible polynomials play the role of prime numbers. That is, each polynomial can be expressed as a product of irreducible polynomials. For instance, the irreducible polynomials of grade 1 in $\mathbb{F}_3[x]$ are $x+1$ and $x+2$. Can there be two polynomials $n[x]$ and $m[x]$ in $\mathbb{F}_3[x]$ such that $(x+1) \cdot n[x]^2 = (x+2) \cdot m[x]^2$? The answer is no, and the argument is essentially the same as for integers and their prime number representations. Whereas the quantity of occurrences of the irreducible polynomial $x + 1$ in a normal form representation of $(x + 2) \cdot n[x]^2$ has to be even, it is odd in a normal form representation of $(x + 1) \cdot m[x]^2$.

A similar argument is used in *set theory*, where the reasoning about two sets can sometimes be reduced to their cardinality, that is, a uniform representation of the two sets under scrutiny.

Hence, we have now a general principle and argument used in set theory, number theory and polynomial rings (and probably also other areas of mathematics), from which the argument for the irrationality of $\sqrt{2}$ is just a special instance. We are currently working on methods, tactics and con-

[10] In the sense of coding it into some representational formalism.

trol rules to mechanize this more general approach, and to demonstrate its feasibility in much larger and diverse fields of mathematics.

The general idea here is to represent objects via normal forms and to use projection in order to rewrite these representations until a distinguishing property becomes obvious. For our problem class at hand a distinguishing property can be computed as follows: Compute the prime product normal form of l. This can be done with a computer algebra system. Let $l = (p_1)^{o_1} \cdot \ldots \cdot (p_k)^{o_k}$, that is, o_i is the quantity of occurrences of the prime p_i. Then, there exists an $o \in \{o_1, \ldots, o_k\}$, which is not divisible by j, since otherwise $\sqrt[j]{l}$ would be rational (again the divisibility for each o_i can be checked with a computer algebra system). Let p be the prime number corresponding to o. For an integer x let $\delta_p(x)$ be the number of occurrences of p in the prime product of x. δ_p is a homomorphism with respect to multiplication and addition, that is, $\delta_p(x \cdot y) = \delta_p(x) + \delta_p(y)$. Moreover, we know that $\delta_p(x^y) = y \cdot \delta_p(x)$. The application of δ_p to both sides and repeated applications of these equations rewrite the initial equation as follows:

$$\begin{array}{lll} & l \cdot n^j & = & m^j \\ \Rightarrow & \delta_p(l \cdot n^j) & = & \delta_p(m^j) \\ \Rightarrow & \delta_p(l) + \delta_p(n^j) & = & \delta_p(m^j) \\ \Rightarrow & o + j \cdot \delta_p(n) & = & j \cdot \delta_p(m) \end{array}$$

Now, it is clearly visible that the left hand side of the equation is not divisible by j whereas the right hand side is.

This is a far more general approach and the corresponding methods are certainly more likely candidates for an entry into the international knowledge base on mathematics in the centuries to come.

We are now working on formalizing these methods in rather general terms and then instantiate them with appropriate parameters to the domain in question (number theory, set theory, or polynomial rings) — and the crucial creative step of the system MULTI is then to find the instantiation by some general heuristics.

Acknowledgments

We thank Claus-Peter Wirth and Volker Sorge for their generous support in writing this paper and many fruitful discussions on the proof planning topics raised in this paper. Moreover, we are grateful to the anonymous reviewers for their thought-provoking comments.

This work has been supported partially by the EU training network CALCULEMUS (HPRN-CT-2000-00102) and partially by the Sonderforschungsbereich 378 of the Deutsche Forschungsgemeinschaft DFG.

BIBLIOGRAPHY

[Allen *et al.*, 2000] S. Allen, R. Constable, R. Eaton, C. Kreitz, and L. Lorigo. The Nuprl open logical environment. In McAllester [2000].

[Andrews *et al.*, 1996] P. 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.

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

[Baumgartner and Furbach, 1994] P. Baumgartner and U. Furbach. PROTEIN, a PROver with a Theory INterface. In Bundy [1994], pages 769–773.

[Benzmüller and Kohlhase, 1998] C. Benzmüller and M. Kohlhase. LEO — a higher-order theorem prover. In Kirchner and Kirchner [1998].

[Benzmüller and Sorge, 1998] C. Benzmüller and V. Sorge. A blackboard architecture for guiding interactive proofs. In Giunchiglia [1998].

[Benzmüller and Sorge, 2000] C. Benzmüller and V. Sorge. Ω-ANTS – An open approach at combining Interactive and Automated Theorem Proving. In Kerber and Kohlhase [2000].

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

[Benzmüller *et al.*, 2002] 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.

[Benzmüller *et al.*, 2003] C. Benzmüller, A. Meier, and V. Sorge. Bridging theorem proving and mathematical knowledge retrieval. In *Festschrift in Honour of Jörg Siekmann's 60s Birthday*, LNAI. Springer, 2003. To appear.

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

[Bishop and Andrews, 1998] M. Bishop and P. Andrews. Selectively instantiating definitions. In Kirchner and Kirchner [1998].

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

[Bourbaki, 1968] N. Bourbaki. *Theory of sets.* His Elements of mathematics, 1. Paris, Hermann Reading Mass., Addison-Wesley, 1968.

[Bundy *et al.*, 1990] 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.

[Bundy, 1988] A. Bundy. The use of explicit plans to guide inductive proofs. In Lusk and Overbeek [1988], pages 111–120.

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

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

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

[Cheikhrouhou and Siekmann, 1998] L. Cheikhrouhou and J. Siekmann. Planning diagonalization proofs. In Giunchiglia [1998], pages 167–180.

[Cheikhrouhou and Sorge, 2000] L. Cheikhrouhou and V. Sorge. $\mathcal{PDS}$ — 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.

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

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

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

[Doris, 2001] The Doris system is available at `http://www.cogsci.ed.ac.uk/~jbos/doris/`.

[Drummond, 1994] M. Drummond. On precondition achievement and the computational economics of automatic planning. In *Current Trends in AI Planning*, pages 6–13. IOS Press, 1994.

[Fiedler, 2001a] A. Fiedler. *P.rex*: An interactive proof explainer. In Goré et al. [2001].

[Fiedler, 2001b] 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.

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

[Fikes and Nilsson, 1971] R.E. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Fikes *et al.*, 1972] R. R. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[Franke and Kohlhase, 2000] A. Franke and M. Kohlhase. System description: MBase, an open mathematical knowledge base. In McAllester [2000].

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

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

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

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

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

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

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

[Hillenbrand *et al.*, 1999] Th. Hillenbrand, A. Jaeger, and B. Löchner. System description: Waldmeister — improvements in performance and ease of use. In Ganzinger [1999], pages 232–236.

[Huang, 1994] X. Huang. Reconstructing Proofs at the Assertion Level. In Bundy [1994], pages 738–752.

[Ireland and Bundy, 1996] A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1-2):79–111, 1996.

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

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

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

[Kohlhase and Franke, 2001] 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.

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

[Manthey and Bry, 1988] R. Manthey and F. Bry. SATCHMO: A theorem prover implemented in Prolog. In Lusk and Overbeek [1988], pages 415–434.

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

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

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

[Meier and Sorge, 2000] A. Meier and V. Sorge. Exploring properties of residue classes. In Kerber and Kohlhase [2000].

[Meier *et al.*, 2001] 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.

[Meier *et al.*, 2002a] 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.

[Meier *et al.*, 2002b] 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.

[Meier, 2000] A. Meier. TRAMP: Transformation of Machine-Found Proofs into Natural Deduction Proofs at the Assertion Level. In McAllester [2000].

[Meier, 2003] A. Meier. *Proof Planning with Multiple Strategies.* PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2003.

[Melis and Meier, 2000] 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.

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

[Melis *et al.*, 2000] E. Melis, J. Zimmer, and T. Müller. Integrating constraint solving into proof planning. In Kirchner and Ringeissen [2000].

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

[Melis, 1998] Erica 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.

[Newell and Simon, 1963] A. Newell and H. Simon. GPS: A program that simulates human thought. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 279–296. McGraw-Hill, New York, NY, 1963.

[Nipkow *et al.*, 2002] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic.* Number 2283 in LNCS. Springer, 2002.

[Owre *et al.*, 1996] 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.

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

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

[Riazanov and Voronkov, 2001] A. Riazanov and A. Voronkov. Vampire 1.1 (system description). In Goré et al. [2001].

[Richardson *et al.*, 1998] J. Richardson, A. Smaill, and I. Green. System description: Proof planning in higher-order logic with λClam. In Kirchner and Kirchner [1998].

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

[Siekmann *et al.*, 1999] 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 ΩMEGA $\mathcal{U}$ser $\mathcal{I}$nterface. *Formal Aspects of Computing*, 11:326–342, 1999.

[Siekmann *et al.*, 2002] J. Siekmann, C. Benzmüller, V. Brezhnev, L. Cheikhrouhou, A. Fiedler, A. Franke, H. Horacek, M. Kohlhase, A. Meier, E. Melis, M. Moschner, I. Normann, M. Pollet, V. Sorge, C. Ullrich, C.-P. Wirth, and J. Zimmer. Proof development with ΩMEGA. In Voronkov [2002], pages 143–148.

[Sorge, 2000] V. Sorge. Non-Trivial Computations in Proof Planning. In Kirchner and Ringeissen [2000].

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

[Sutcliffe *et al.*, 1994] G. Sutcliffe, C. Suttner, and T. Yemenis. The TPTP problem library. In Bundy [1994].

[Vo *et al.*, 2003] B. Quoc Vo, C. Benzmüller, and S. Autexier. Assertion application in theorem proving and proof planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003. (poster description).

[Voronkov, 2002] A. Voronkov, editor. *Proceedings of the 18th International Conference on Automated Deduction*, number 2392 in LNAI. Springer, 2002.

[Weidenbach *et al.*, 1999] 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 [1999], pages 378–382.

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

[Wiedijk, 2002] F. Wiedijk. The fifteen provers of the world. Unpublished Draft, 2002.

[Zhang and Zhang, 1995] 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.

[Zimmer and Kohlhase, 2002] J. Zimmer and M. Kohlhase. System description: The Mathweb Software Bus for distributed mathematical reasoning. In Voronkov [2002], pages 138–142.