Proof Development with ΩMEGA: $\sqrt{2}$ Is Irrational

J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, and M. Pollet

FR 6.2 Informatik, Universität des Saarlandes, 66041 Saarbrücken, Germany
{siekmann|chris|afiedler|ameier|pollet}@ags.uni-sb.de

Abstract. Freek Wiedijk proposed the well-known theorem about the irrationality of $\sqrt{2}$ as a case study and used this theorem for a comparison of fifteen (interactive) theorem proving systems, which were asked to present their solution (see [48]).
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 [45] back to real mathematical challenges.
In this paper we present an overview of the ΩMEGA system as far as it is relevant for the purpose of this paper and show the development of a proof for this theorem.

1 ΩMEGA

The ΩMEGA proof development system [40] 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 data structure and several complementary subsystems. ΩMEGA has many characteristics in common with systems like NuPrL [1], CoQ [19], Hol [24], and PVS [37]. 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 systems at Edinburgh [38]. 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 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. The ΩMEGA system is a representative of the systems in the new paradigm of *proof planning* and combines

M. Baaz and A. Voronkov (Eds.): LPAR 2002, LNAI 2514, pp. 367–387, 2002.

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 version of Church's simply typed λ-calculus [18]. The logical language also supports partial functions and is called $\mathcal{POST}$ [46], 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* which is called *proof planning* (see Section 1.3). Proof construction is also supported by already proved assertions and theorems and by calls to external systems to simplify or solve subproblems, as shown in Section 1.2.

At the core of ΩMEGA is the *proof plan data structure* $\mathcal{PDS}$ [17] in which proofs and *proof plans* are represented at various levels of granularity and abstraction. The proof plans are developed and then classified with respect to a taxonomy of mathematical theories, which is currently being replaced by the mathematical data base MBASE [22,28]. The user of ΩMEGA, the proof planner MULTI [34], or the suggestion mechanism Ω-ANTS [10] 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 is to be expanded by submethods and subtactics 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.

User interaction is supported by the graphical user interface $\mathcal{L}\Omega\mathcal{UI}$ [42] and the interactive proof explanation system *P.rex* [20].

The previously monolithic system was split up and separated into several independent modules and these modules are connected via the mathematical software bus MATHWEB-SB [49]. An important benefit is that MATHWEB modules can be distributed over the Internet and are then accessible by other distant research groups as well and there is a very active user community with sometimes several thousand theorems and lemmata being proved per day (most theorems are generated automatically as subproblems in natural language processing, proof planning and verification tasks).

1.2 External Systems

Proof problems require many different skills for their solutions and 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. The

output of an incorporated reasoning system is translated and inserted as a subproof into the $\mathcal{PDS}$, which maintains the overall proof plan. This is beneficial for interfacing systems that operate at different levels of abstraction, as well as for a human oriented display and inspection of a partial proof. When integrating partial results, it is important to check the soundness of each contribution. This is accomplished by translating the external solution into a subproof in ΩMEGA, which is then refined to a logic-level proof to be examined by ΩMEGA's proof checker.

Currently, the following external systems are integrated in ΩMEGA:

CASs, i.e. *computer algebra systems*, provide symbolic computation, which can be used in two ways: to compute hints to guide the proof search (e.g., witnesses for existential variables) and secondly 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 [43]. Currently, ΩMEGA uses the systems MAPLE [15] and GAP [39].

ATPs are employed to solve subgoals. Currently ΩMEGA uses the first-order *automated theorem proving systems* BLIKSEM, EQP, OTTER, PROTEIN, SPASS, WALDMEISTER, and the higher-order systems TPS [2], and $\mathcal{LEO}$ [7,4]. The first-order ATPs are connected via TRAMP [30], 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 [5].

MGs guide the proof search. A *model generator* provides witnesses for free (existential) variables or counter-models that show that some subgoal is not a theorem. Currently, ΩMEGA uses the MGs SATCHMO and SEM.

CSs construct mathematical objects with theory-specific properties as witnesses for free (existential) variables. Moreover, a *constraint solver* can help to reduce the proof search by checking for inconsistencies of constraints. Currently, ΩMEGA employs $\mathcal{CoSIE}$ [36], a constraint solver for inequalities and equations over the field of real numbers.

1.3 Proof Planning

ΩMEGA's main focus is on knowledge-based proof planning [13,35], 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 (ATP) 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 and meta-level reasoning. It differs from traditional search-based techniques in ATP 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, i.e. the abstract proof plan, can be recursively expanded

and it will thus construct a proof within a logical calculus. The plan operators represent mathematical techniques familiar to working mathematicians. 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 [35] used the limit theorems proposed by Woody Bledsoe [12] 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.

1.4 Interface and System Support via Ω-ANTS

ΩMEGA's graphical user interface $\mathcal{LOUI}$ [42] 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* [20,21], 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, e.g. by detailing some ill understood subproof.

Another system support feature of ΩMEGA is the guidance mechanism provided by the suggestion module Ω-ANTS which searches pro-actively for a set of possible actions that may be helpful in finding a proof and orders them in a preference list. These actions can be 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 (see [9,10,44] for more details). Ω-ANTS is based on a hierarchical blackboard upon which data about the current proof state is collected, which is computed by concurrent computational threads that communicate via this blackboard.

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 (LCF-style) tactic applications. Conceptually, each such justification represents a proof plan (the expansion of

the justification) at a lower level of abstraction that is computed when the tactic is executed. In ΩMEGA, we explicitly keep the original proof plan in an expansion hierarchy. Thus the $\mathcal{PDS}$ makes the hierarchical structure of proof plans explicit and retains it for further applications such as proof explanation with *P.rex* or analogical transfer of plans.

The lowest level of abstraction of a $\mathcal{PDS}$ is the level of Gentzen's Natural Deduction Calculus. A $\mathcal{PDS}$ could be constructed manually at this level, however for each ND rule, there is a command in ΩMEGA that applies the rule. The application direction of an ND rule is determined according to the given arguments of the associated command, for instance, a rule is applied backwards when an existing open node is entered for the conclusion and a `NIL` is given for every premise. All ND rules are "agentified", i.e., the agent of this rule searches pro-actively for its application condition and when it succeeds it suggests itself.

The proof object generated by ΩMEGA in this case study of the irrationality of $\sqrt{2}$ is recorded in a technical report [6] where the unexpanded and the expanded proof object is presented.

1.6 Case Studies

The ΩMEGA system has been used in several 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 [35]. This class was originally proposed by W. Bledsoe [12] 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 a similar 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 computations are done with 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 W. Bledsoe and many more theorems in this class taken from a standard textbook [3].

Another class of problems we tackled with proof planning is concerned with residue classes [32,31]. In this domain we show theorems such as: the residue class structure $(\mathbb{Z}_5, \bar{+})$ is associative, it has a unit element, and other 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 all in all we have shown about 10,000 theorems of this kind (see [44]). Although the problems in this domain are mostly 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 sometimes generated substantially different proofs based on entirely different proof ideas.

The essence of proof planning is to capture and represent standard mathematical techniques; for example, the early developments of proof planning in Alan Bundy's group at Edinburgh used proofs by induction as their favorite case studies [13].

Another important proof technique is Cantor's diagonalization technique and we also developed methods and strategies for this class [16]. 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 $[0, 1]$ 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 lifting this to the general level. We have done this for many refinements of resolution with ΩMEGA (see [23]).

2 A Case Study: $\sqrt{2}$ Is Not Rational

ΩMEGA's main aim is to become a proof assistant tool for the working mathematician and 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 Pythagoras, provide an excellent challenge to evaluate whether this ambitious goal has been reached, and in [48] fifteen systems that have solved this problem show their respective results. The protocols of their respective sessions have been compared on a multidimensional scale in order to assess the "naturalness" by which real mathematical problems of this kind can be proved within the respective system.

We contributed to this case study essentially in three different forms. Our initial contribution was an interactive proof in ΩMEGA without adding special domain knowledge represented in tactics, methods, or control rules 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 [6]. The most important 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 it is already an abstraction from the calculus level typical for most ATPs (it is called the *assertion level* in [25]), 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 ATP about fifteen years ago.

Our second contribution to the case study of the irrationality of $\sqrt{2}$ is based on interactive island planning [33]. Interactive island planning expects an outline of the proof and the user provides 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 irrationality of $\sqrt{2}$ is a fully automatically planned and expanded proof of the theorem to be presented in a forthcoming paper [41].

In the remainder of this paper, we present the interactive island approach for the irrationality of $\sqrt{2}$ example in detail. The actual challenge, attributed to the Pythagorean school, is as follows:

Theorem 1. $\sqrt{2}$ *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.

q.e.d.

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 `rat` and `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.

To prove the given problem, further mathematical knowledge is required. Our proof employs the definition of `evenp` and some theorems about the concepts `rat`, `common-divisor`, and `evenp`. These theorems are also proved with ΩMEGA which requires the definition of concepts such as `rat`, `sqrt` and `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, ΩMEGA expressions such as $\sqrt{2}$ are mapped to respective MAPLE representations, and MAPLE uses its own built-in knowledge to manipulate them. Using and verifying these computation steps requires expansion of MAPLE's computation to the calculus layer in ΩMEGA. This is done by replaying

MAPLE's computation by special computational tactics in ΩMEGA which may also unfold some definitions like `sqrt`. These tactics and their expansions are part of the SAPPER system and they 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.

```
(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 Interactive Proof Development in ΩMEGA

The theorem can be shown interactively in ΩMEGA along the lines of the previously given textbook style proof. However, due to space restrictions we will not show the proof development using ΩMEGA's graphical user interface $\mathcal{LOUI}$, but the more cumbersome command line interface of the emacs editor in order to avoid $\mathcal{LOUI}$ screenshots 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 lines and the previously open proof lines 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. A proof line is of the form '$L \quad (\Delta)\ !\ \varphi \quad \mathcal{R}$', where L is a unique label, $(\Delta)\ !\ \varphi$ 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. Apologies to the user that we use the symbol `!` here for the syntactic derivation $\vdash$, since the latter symbol is not available in the emacs interface.

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.
```

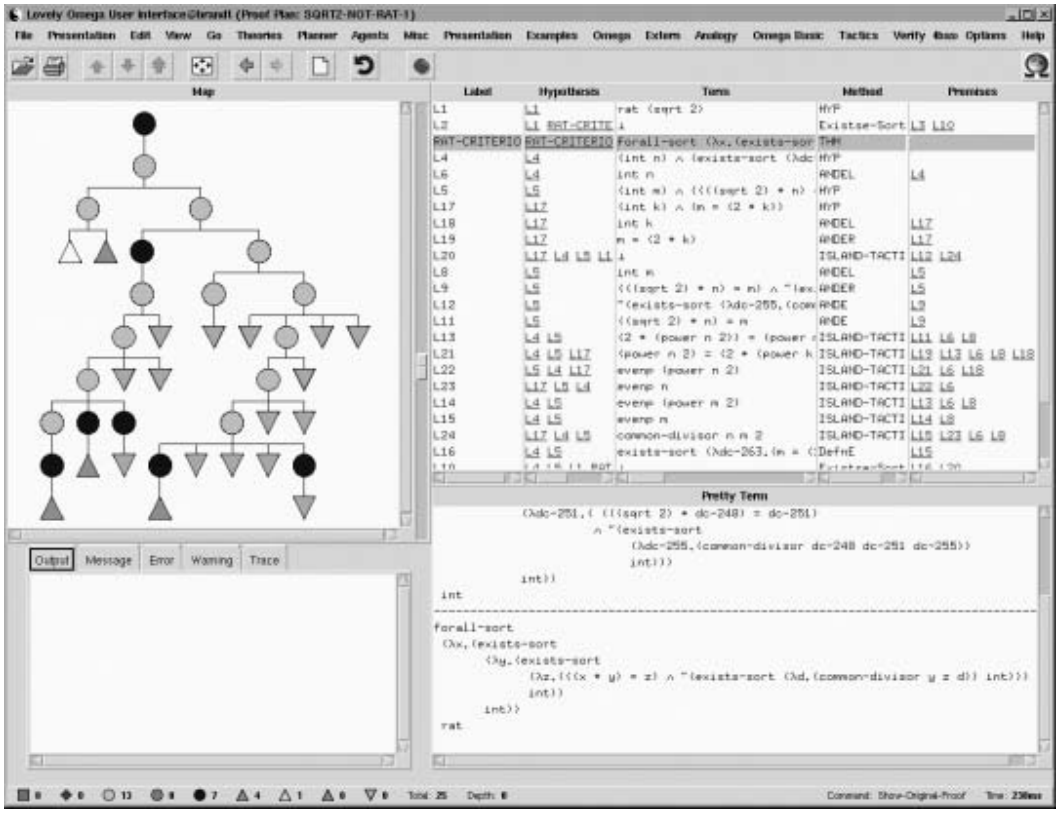

Fig. 1. Multi-modal proof presentation in the graphical user interface $\mathcal{LOUI}$.

```
;;; Tactics loaded for theory REAL.
;;; Methods loaded for theory REAL.
;;; Strategies loaded for theory REAL.
[...]
```

First, 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 the theorem `RAT-CRITERION` from the database. As a side effect, the newly introduced proof line containing that theorem is implicitly added to the hypotheses lists of all other proof lines. Note that the retrieval of this theorem and its application in the context is non-trivial, since no

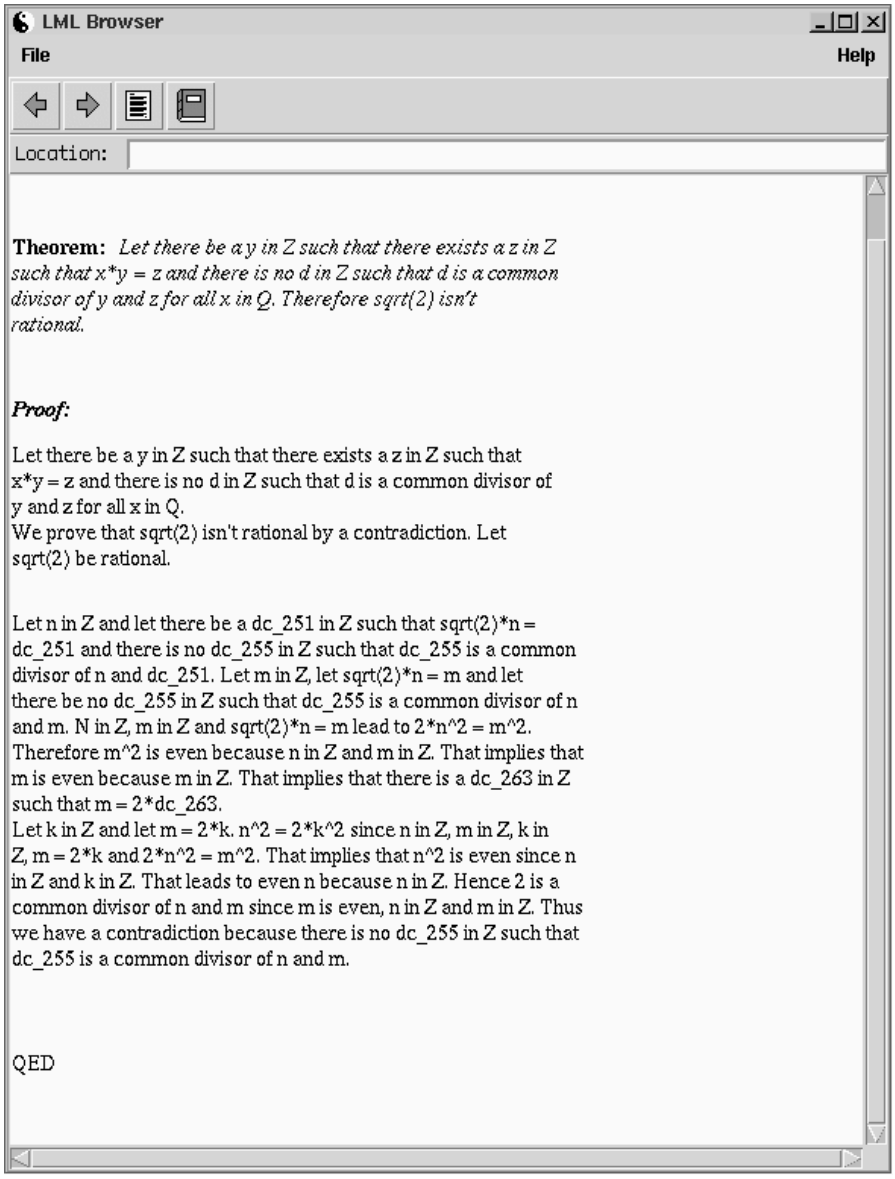

Fig. 2. Natural language proof presentation by *P.rex* in $\mathcal{L}\Omega\mathcal{UI}$.

syntactical criteria can be exploited, such as subterms in the theorem that match expressions in the proof context. Automatic acquisition of this theorem is hence very challenging and requires semantical tools.

```
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 quantifiers 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)) (RAT-CRITERION L1)
            (EXISTS-SORT ([DC-251].
              (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 quantifier 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 their sort information, are decomposed right away.

```
OMEGA: mexistse-sort*
CONCLINE (NDLINE) Conclusion Line.: [L2]
EXLINE (NDLINE) An existentially quanitified 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]. (COMMON-DIVISOR N M DC-255)) INT))    ANDE: (L9)
```

Step 6: While these five steps are more or less canonical, we shall now start the island approach to sketch the refutation argument. First, we do 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 do arbitrarily large steps in our proof. The correctness of these steps is then checked when ISLAND-TACTIC is expanded. Note that we specify the premises that we want to employ, which is important information as specifying too few premises can 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[1] 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]. (= M (TIMES 2 DC-263))) INT)         DefnE: (L15)
```

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)
```

Step 11: Now, we can go on with our calculations using 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)
```

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

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)
```

2.3 Closing the Gaps

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

We shall now describe the (semi-automated) task of closing the gaps between the islands, 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 automated theorem prover OTTER, which is called via the TRAMP system, and the computer algebra system MAPLE. Although these external systems are essentially capable of closing the gaps, 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 to manually unfold some definitions. All of this will eventually be done automatically as discussed in Sect. 3.

The first step dealing with `ISLAND-TACTIC` is always to expand its application. For lack of space, 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. The recursive process over all necessary levels until an ND-level proof is reached is called *full expansion of a tactic* whereas *expansion of a tactic* means only the direct one-level expansion of the tactic application.

Proving L20**:** Proof node 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 and L24 are:

```
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 further 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 then prove L25 using 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.

```
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, which works automatically and results in an ND-level subproof for L25 that consists of 13 steps and an ND-level subproof with 40 steps for L30.

Proving L15 and L23: 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 succeeds to prove L15 and L23, respectively, and TRAMP provides as output 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 and L22: 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 and TRAMP to obtain an ND-level 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]. (= (POWER M 2) (TIMES 2 DC-98774))) INT)    OPEN
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 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: Before we can exploit OTTER and TRAMP 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 the unfolding of EVENP, respectively. 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 provides 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 the statement suggests. 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 and L21: 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 automatically 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. In order to do so, BYCOMPUTATION passes all equations to MAPLE and calls MAPLE's is function to check whether the conclusion equation holds when assuming the premise equations. This tactic succeeds, for instance, when applied to L13 and L21, for instance, closing 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)
```

The tactic BYCOMPUTATION can currently not be expanded to an ND-level proof, but we are working on extensions to expand such equations also into ND proof lines. The two applications of BYCOMPUTATION that justify L13 and L21 are therefore only 'verified' by MAPLE, but not automatically by an ND-level proof.

Result: When fully expanded, the proof consists of 282 nodes where two nodes `L13` and `L21` are justified by the tactic `BYCOMPUTATION`, which can currently not be further expanded automatically. 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 `PENTIUM III` 1.8 GHz machine with 512 MB RAM running `LINUX`.

3 Proof Automation: Results and Discussion

The proof for the theorem that $\sqrt{2}$ is irrational is constructed in two steps: First an abstract proof is outlined, in which islands are coupled by tactics without care for the detailed logical dependencies between them, and the gaps between the islands are filled 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: Proof planning with MULTI is the means to create island proofs at a user-friendly abstract level and in order to obtain verifiable logic-level proofs we are working on extensions of Ω-ANTS, as we shall now discuss.

3.1 Proof Planning the $\sqrt{2}$ Problem

Proof planning depends on appropriate domain specific methods and control knowledge. To tackle the $\sqrt{2}$ problem with proof planning we represented some of the respective mathematical knowledge and added it to ΩMEGA, where the goal of the game is to find *general* methods for a whole class of theorems, which solve not only a particular problem, but also other theorems in that class. Thus, we first analyzed proofs for similar statements such as $\sqrt{8}$, $\sqrt{(3 \cdot 3) - 1}$, or $\sqrt[3]{2}$ in order to find a general approach for the problem that $\sqrt[k]{l}$ is irrational for integers k and l. The proof technique we encoded into methods and control rules essentially follows the proof outline of the abstract island proof for $\sqrt{2}$, but it generalizes some concepts. After deriving the equation $l \cdot n^k = m^k$ for two integers n and m, which are supposed to have no common divisor, MULTI employs MAPLE to compute the prime divisors of l, which are thus also prime divisors of m^k (i.e., the concept "even" is generalized to the concept "prime divisor"). Then MULTI employs a generalized version of the `SQUARE-EVEN` theorem. This generalized theorem states that i and i^k have the same prime divisors where i is an integer. Therefore, m has the same prime divisors as m^k. By representing m as a product of its prime divisors we can rewrite the equation $l \cdot n^k = m^k$ to an equation $n^k = k' \cdot c'^k$ from which prime divisors of n can be computed (in general several rewrite cycles are necessary, until MULTI finds such a representation for n). If p is a prime divisor of n and m, MULTI constructs the contradiction that n and m should have no common divisor.

The result is that ΩMEGA can automatically generate abstract proof plans for irrationality statements such as $\sqrt{2}$, $\sqrt{8}$, $\sqrt{(3 \cdot 3) - 1}$ or $\sqrt[3]{2}$ in a uniform way.

The required CPU time is less than 4 sec. on a 1.8 GHz machine with 512 MB RAM running LINUX.

However, we still feel that these methods are too specific and we are looking for a more general mathematical fact of technique or which these methods are just instances.

3.2 Closing the Gaps Automatically

We have now expansion tactics for the previously mentioned methods such that the expansion of the irrationality of $\sqrt[k]{l}$ proof plans can be done automatically and we can in fact fully expand and verify them (modulo the computer algebra system computations) with our proof checker.

In contrast, expanding and closing island gaps is more challenging, 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.

In order 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 pro-actively 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 [8]). 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 appears to be attainable.
- A better support for unfolding definitions is to adapt Bishop and Andrew's selective unfolding mechanism [11] to our proof planning context.

Acknowledgments. We thank Claus-Peter Wirth for his 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 useful comments.

References

1. S. Allen, R. Constable, R. Eaton, C. Kreitz, and L. Lorigo. The Nuprl open logical environment. In McAllester [29].
2. 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.
3. R. Bartle and D. Sherbert. *Introduction to Real Analysis*. Wiley, 2 edition, 1982.
4. C. Benzmüller. *Equality and Extensionality in Higher-Order Theorem Proving*. PhD thesis, Department of Computer Science, Saarland University, 1999.
5. C. Benzmüller, M. Bishop, and V. Sorge. Integrating TPS and ΩMEGA. *Journal of Universal Computer Science*, 5:188–207, 1999.
6. 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, 2002.
7. C. Benzmüller and M. Kohlhase. LEO — a higher-order theorem prover. In *Proceedings of the 15th International Conference on Automated Deduction (CADE-15)*, LNAI, LINDAU, Germany, 1998.
8. 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, 2002.
9. C. Benzmüller and V. Sorge. A blackboard architecture for guiding interactive proofs. In *Proceedings of 8th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA'98)*, LNAI, Sozopol, Bulgaria, 1998.
10. C. Benzmüller and V. Sorge. Ω-ANTS – An open approach at combining Interactive and Automated Theorem Proving. In M. Kerber and M. Kohlhase, editors, *Proceedings of the 8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus-2000)*. AK Peters, 2001.
11. M. Bishop and P. Andrews. Selectively instantiating definitions. In Kirchner and Kirchner [26].
12. W. Bledsoe. Challenge problems in elementary calculus. *Journal of Automated Reasoning*, 6:341–359, 1990.
13. A. Bundy. The use of explicit plans to guide inductive proofs. In E. Lusk and R. Overbeek, editors, *Proceedings of the 9th Conference on Automated Deduction*, number 310 in LNCS, pages 111–120, Argonne, Illinois, USA, 1988. Springer Verlag.
14. A. Bundy, editor. *Proceedings of the 12th Conference on Automated Deduction*, number 814 in LNAI, Nancy, France, 1994. Springer Verlag.
15. B. Char, K. Geddes, G. Gonnet, B. Leong, M. Monagan, and S. Watt. *First leaves: a tutorial introduction to Maple V*. Springer Verlag, Berlin, 1992.
16. L. Cheikhrouhou and J. Siekmann. Planning diagonalization proofs. In F. Giunchiglia, editor, *Proceedings of 8th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA'98)*, pages 167–180, Sozopol, Bulgaria, 1998. Springer Verlag, Berlin, Germany, LNAI 1480.
17. 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 (ACIDCA'2000)*, 2000.
18. A. Church. A Formulation of the Simple Theory of Types. *The Journal of Symbolic Logic*, 5:56–68, 1940.
19. Coq Development Team. *The Coq Proof Assistant Reference Manual*. INRIA. see `http://coq.inria.fr/doc/main.html`.

20. A. Fiedler. *P.rex*: An interactive proof explainer. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Automated Reasoning — 1st International Joint Conference, IJCAR 2001*, number 2083 in LNAI. Springer, 2001.
21. A. Fiedler. *User-adaptive proof explanation*. PhD thesis, Naturwissenschaftlich-Technische Fakultät I, Saarland University, Saarbrücken, Germany, 2001.
22. A. Franke and M. Kohlhase. System description: MBase, an open mathematical knowledge base. In McAllester [29].
23. H. Gebhard. Beweisplanung für die beweise der vollständigkeit verschiedener resolutionskalküle in ΩMEGA. Master's thesis, Saarland University, Saarbrücken, Germany, 1999.
24. M. Gordon and T. Melham. *Introduction to HOL – A theorem proving environment for higher order logic*. Cambridge University Press, 1993.
25. X. Huang. Reconstructing Proofs at the Assertion Level. In Bundy [14], pages 738–752.
26. C. Kirchner and H. Kirchner, editors. *Proceedings of the 15th Conference on Automated Deduction*, number 1421 in LNAI. Springer Verlag, 1998.
27. H. Kirchner and C. Ringeissen, editors. *Frontiers of combining systems: Third International Workshop, FroCoS 2000*, volume 1794 of *LNAI*. Springer, 2000.
28. 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.
29. D. McAllester, editor. *Proceedings of the 17th Conference on Automated Deduction*, number 1831 in LNAI. Springer, 2000.
30. A. Meier. TRAMP: Transformation of Machine-Found Proofs into Natural Deduction Proofs at the Assertion Level. In McAllester [29].
31. 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)*, volume 2178 of *LNCS*, pages 494 – 508. Springer Verlag, 2001.
32. A. Meier, M. Pollet, and V. Sorge. Comparing approaches to the exploration of the domain of residue classes. *Journal of Symbolic Computation*, forthcoming.
33. E. Melis. Island planning and refinement. Seki-Report SR-96-10, Department of Computer Science, Saarland University, 1996.
34. 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, *Proceedings of the First International Conference on Computational Logic*, volume 1861 of *LNAI*, pages 644–659. Springer-Verlag, 2000.
35. E. Melis and J. Siekmann. Knowledge-based proof planning. *Artificial Intelligence*, 115(1):65–105, 1999.
36. E. Melis, J. Zimmer, and T. Müller. Integrating constraint solving into proof planning. In Kirchner and Ringeissen [27].
37. 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*, volume 1102 of *LNCS*, pages 411–414, New Brunswick, NJ, 1996. Springer-Verlag.
38. J. Richardson, A. Smaill, and I. Green. System description: Proof planning in higher-order logic with λ*clam*. In Kirchner and Kirchner [26].
39. M. Schönert et al. *GAP – Groups, Algorithms, and Programming*. Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 1995.

40. 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 [47], pages 143–148. See `http://www.ags.uni-sb.de/~omega/`.
41. J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, and M. Pollet. Proof development with ΩMEGA: $\sqrt{2}$ is not rational. *Special Issue of Journal of Automated Reasoning*, 2002. Submitted.
42. 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.
43. V. Sorge. Non-Trivial Computations in Proof Planning. In Kirchner and Ringeissen [27].
44. V. Sorge. Ω-ANTS — *A Blackboard Architecture for the Integration of Reasoning Techniques into Proof Planning.* PhD thesis, Saarland University, Saarbrücken, Germany, 2001.
45. G. Sutcliffe, C. Suttner, and T. Yemenis. The TPTP problem library. In Bundy [14].
46. The ΩMEGA group. $\mathcal{POST}$. See at `http://www.ags.uni-sb.de/~omega/primer/post.html`.
47. A. Voronkov, editor. *Proceedings of the 18th International Conference on Automated Deduction*, number 2392 in LNAI. Springer Verlag, 2002.
48. F. Wiedijk. The fifteen provers of the world. Unpublished Draft, 2002.
49. J. Zimmer and M. Kohlhase. System description: The mathweb software bus for distributed mathematical reasoning. In Voronkov [47], pages 138–142.