Irrationality of $\sqrt{2}$

A case study in OMEGA

Christoph Benzmüller, Armin Fiedler, Andreas Meier, and Martin Pollet

SEKI Report SR-02-03

SEKI Report
ISSN 1437-4447

UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D-66041 SAARBRÜCKEN
GERMANY
WWW: http://www.ags.uni-sb.de/
Irrationality of $\sqrt{2} - A$ case study in $\Omega$MEGA

Christoph Benzmüller, Armin Fiedler, Andreas Meier, Martin Pollet

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

Contents

1 Introduction 3
2 Questionnaire on $\Omega$MEGA 3
3 Proof Objects 7
4 Our Proof of Choice 8
5 Problem Formalization 8
   5.1 The Proof Problem 8
   5.2 Further Definitions and Lemmata 9
6 Interactive Theorem Proving in $\Omega$MEGA 11
7 Proof Presentation 27
8 External Reasoning Systems 32
9 A Related Case Study 32
10 Discussion 40

A $\LaTeX$ Presentations of the Proof 45
   A.1 The Unexpanded Proof 45
   A.2 The Expanded Proof 45
   A.3 Customizing $\LaTeX$ Presentations 49

B $\Omega$MEGA Proof Objects 52
   B.1 The Unexpanded Proof 52
   B.2 The Expanded Proof 54

C Proof Transformation with Tramp 56
   C.1 An Isolated Subproblem from the Case Study 56
   C.2 Interactive Session with a Call of Otter 56
   C.3 The Unexpanded Proof Object Generated by Tramp 60
   C.4 The Expanded Proof Object 61
### D OMEGA’s Knowledge Base

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>D.1</td>
<td>Theory Real</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>D.1.1 real.thy</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>D.1.2 real-theorems.thy</td>
<td>72</td>
</tr>
<tr>
<td>D.2</td>
<td>Theory Rational</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>D.2.1 rational.thy</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>D.2.2 rational-theorems.thy</td>
<td>76</td>
</tr>
<tr>
<td>D.3</td>
<td>Theory Integer</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td>D.3.1 Integer.thy</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td>D.3.2 integer-theorems.thy</td>
<td>82</td>
</tr>
<tr>
<td>D.4</td>
<td>Theory Natural</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td>D.4.1 natural.thy</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td>D.4.2 natural-theorems.thy</td>
<td>96</td>
</tr>
</tbody>
</table>
1 Introduction

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 [53]).

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 [50] back to real mathematical challenges.

The structure of this report is as follows: We first present an overview of the $\Omega$mega system as far as it is relevant for the purpose of this report in Section 2 and describe the central data structure for proof objects in Section 3. Section 4 presents our proof of choice for the irrationality of $\sqrt{2}$ problem. The formalization of the problem in $\Omega$mega is then described in Section 5 and the interactive proof is given in Section 6. The subsequent sections address the aspects of proof presentation (Section 7) and external reasoning systems (Section 8). Finally, Section 9 briefly sketches a related case study before a summarizing discussion of the features of $\Omega$mega in Section 10 concludes the report. The appendix contains several detailed protocols and documents that illustrate various aspects that have been addressed in the main part of the report.

2 Questionnaire on $\Omega$MEGA

1. Where is the home page of the system?

The home page of $\Omega$mega can be accessed at http://www.ags.uni-sb.de/~omega. There, the system and its components are described in some detail. Moreover, the current implementation can be accessed and literature about the system can be retrieved.

2. Are there any books about the system?

There is no book available yet, but there are several journal and conference publications. An overview on recent publications is provided by the $\Omega$mega system description at CADE 2002 [44] and in [45] as well as on the home page (see 1).

3. What is the logic of the system?

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 $\lambda$-calculus [19]. Higher levels of abstraction are defined in terms of steps at lower levels.

$\Omega$mega's main focus is on knowledge-based proof planning [15, 38], where proofs are not conceived in terms of low-level calculus rules but at a higher level of abstraction that highlights the main ideas and de-emphasizes minor logical or mathematical manipulations on formulae. This viewpoint is realized in the system by proof tactics and abstract proof methods. In contrast to, for instance, the LCF philosophy, our tactics and methods are not necessarily always correct as they have heuristic elements incorporated that account for their strength, such that an informed use of these methods is unlikely to run into failures too often. Since an abstract proof plan may be incorrect for a specific case, its correctness has to be tested by refining it into a logical ND proof in $\Omega$mega's core calculus. The ND proof can then be verified by $\Omega$mega's proof checker.

4. What is the implementation architecture of the system?

Figure 1 illustrates the basic architecture of $\Omega$mega: the previously monolithic system, as it was described in [8], has been split up and separated into several independent modules. These modules are connected via the mathematical software bus MathWeb-SB [54]. Different modules are written in different programming languages (e.g., the $\Omega$mega kernel and the proof planner are written in Lisp, the graphical user interface is written in Oz). An important benefit is that MathWeb modules can be distributed over the Internet and are
then accessible by other distant research groups as well. Thus, a very active user community could be established, which has OMEGA prove sometimes several thousand theorems and lemmata per day. Most theorems are generated automatically as subproblems in natural language processing, proof planning and verification tasks.

At the core of OMEGA is the proof plan data structure 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 [26, 31]. The users of OMEGA, the proof planner Multi [37], or the suggestion mechanism O-Ants [11] modify the PDS during proof development until a complete proof plan has been found. They can also invoke heterogeneous external reasoning systems such as computer algebra systems (CASs), higher- and first-order automated theorem proving systems (ATPs), constraint solvers (CSs), and model generators (MGs). Their use is twofold: they may provide a solution to a subproblem, or they may give hints for the control of the proof search. The output of an incorporated reasoning system is translated and inserted as a subproof into the PDS, which maintains the 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 correctness of each contribution. In OMEGA, this is accomplished by transforming the solution into a subproof. CAS’s results, for instance, are integrated via the transformation module Sapper [47] and first-order ATPs’ results by the proof transformation module TRAMP [33].

Once a complete proof plan at the most appropriate level of abstraction has been found, this plan is to be expanded incrementally into increasingly lower levels of abstraction until finally a proof at the lowest level, that is, the logical calculus, is established. After full expansion, the PDS can be checked by OMEGA’s proof checker.

5. What does working with the system look like?

The interactive search for a proof (plan) is system-supported in the following sense:

(a) The mathematician user may want to construct the proof essentially on his own. In this case, he will just call the appropriate inference rules and tactics, while the system checks for correctness. This usage of the system compares essentially to tactical theorem proving.

(b) In addition to 5a, the user may want to call external reasoning systems that generate subproofs or intermediate calculations.
(c) The user may rely on the proof planner Multi to find a proof plan. Only if this goes astray he will try to provide some top-level guidance.

(d) Another system support feature of Omega is the guidance mechanism provided by the suggestion module Omega-Ants.

Omega-Ants searches proactively for a set of possible actions that may be helpful in finding a proof and orders them in a preference list. Such an action can be an application of a particular calculus rule, a 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 mathematical 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, 11, 48] for more details).

6. What is special about the system compared to other systems?
Omega 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. Omega has many characteristics in common with systems like NuPRL [1], Coq [20], HOL [28], and PVS [41]. However, it differs from these systems with respect to its focus on proof planning and in that respect it is similar to the systems at Edinburgh [14, 42]. Special features of Omega include (1) facilities to access a considerable number of different reasoning systems and to integrate their results into a single proof structure, (2) support for interactive proof development through some non-standard inspection facilities and guidance in the search for a proof, and (3) methods to develop proofs at a knowledge-based level.

7. What are other versions of the system?
The most recent version is Omega 3.6. We are planning to exchange the natural deduction core of the system in the future by the prototypical theorem prover in [4], which combines and significantly extends ideas from [40, 43, 52].

8. Who are the people behind the system?
The Omega group at Saarland University currently consists of the following researchers (many of them work in research projects related to Omega and only a few directly on the kernel of the system): Jörg Siekmann, Christoph Benzmüller, Vladimir Breslin, Armin Fiedler, Andreas Franke, Helmut Horacek, Andreas Meier, Erica Melis, Markus Moschner, Immanuel Normann, Martin Pollet, Carsten Ulrich, Claus-Peter Wirth, and Jürgen Zimmer.

Former members of the Omega group, who contributed substantially to the current version, are: Lassaad Cheikhrouhou (now at the German Research Center for Artificial Intelligence DFKI, Saarbrücken, Germany), Michael Kohlhase (now at Carnegie Mellon University, Pittsburgh, PA), and Volker Sorge (now at University of Birmingham, Birmingham, UK).

9. Which are the main user communities of the system?
The Omega system has been employed at:

- Saarland University, Saarbrücken, Germany (AG Siekmann)
- University of Birmingham, Birmingham, England (Manfred Kerber and Volker Sorge)
- Carnegie Mellon University, Pittsburgh, USA (Michael Kohlhase)
- Cambridge University, Cambridge, England (Mateja Jamnik)

In addition, the MathWeb system has been employed at the University of Edinburgh, Edinburgh, Scotland (Alan Bundy).
10. Which large mathematical formalizations have been done in the system?

The \textsc{Omega} 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 $\epsilon$-$\delta$-proofs [38]. This class was originally proposed by W. Bledsoe [13] and comprises theorems about limits such as the theorem that the limit of the sum of two functions equals the sum of their limits, and 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 $\delta$ depending on an $\epsilon$). Crucial for the success of \textsc{Omega}'s proof planning is the integration of suitable experts for these tasks: the arithmetic computations are done with the computer algebra system \textsc{Maple}, and an appropriate instantiation for $\delta$ is computed by the constraint solver \textsc{Coq}. We have been able to solve all open problems suggested by W. Bledsoe and many more theorems in this class taken from a standard textbook on real analysis [5].

Another class of problems we tackled with proof planning is concerned with residue classes [35, 34, 36]. In this domain we show theorems such as: the residue class structure $(\mathbb{Z}_5, \oplus)$ is associative, it has a unit element, and other similar properties, where $\mathbb{Z}_5$ is the set of all congruence classes modulo 5 (0, 1, 2, 3, 4) and $\oplus$ is the addition on residue classes. We have also investigated whether two given structures are isomorphic or not, and in total we have shown about 10,000 theorems of this kind (see [48]). 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. For instance, one strategy we realized in \textsc{Multi} converts statements on residue classes into statements on numbers and then applies an exhaustive case analysis. Another strategy tries to reduce the original goal into sets of equations to which \textsc{Maple} is applied to check whether the equality actually holds. In this substantial case study, the computer algebra systems \textsc{Maple} and \textsc{GAP} are employed to compute witnesses for particular elements, for instance, to compute $0_5$, the unit element of $(\mathbb{Z}_5, \oplus)$.

Another important proof technique is Cantor's diagonalization technique. We developed methods and strategies for this class [18] and have been able to prove important theorems such as the undecidability of the halting problem, 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 lifted this to the general level. We have done this for many refinements of resolutions with \textsc{Omega} (see [27]).

11. What representation of the formalization has been put in this paper?

The problem has been formalized in \textsc{PVS} syntax. \textsc{PVS} stands for partial functions order sorted type theory. The formalization employs knowledge provided in \textsc{Omega} hierarchically structured knowledge base. See Section 5 for further details.

12. What needs to be explained about the specific proof presented in this paper?

Our aim was to follow the proof sketch shown in Section 4 as closely as possible within the system. We replayed the proof idea in the system by partly employing interactive theorem proving in an island style, that is, we anticipated some islands (some intermediate proof goals) and closed the gaps with the help of tactics and external reasoning systems. Results of external system applications, such as \textsc{Otter} proofs, have been translated and integrated into the central \textsc{Omega} proof object. This proof object has been verified by an independent
proof checker after expansion to the base ND calculus level. The only tactic that could not yet be fully expanded and checked is by-computation, which encodes the computations contributed by MAPLE. We are currently working on the expansions of MAPLE calculations in $\Omega$EQA.

For comparison with the other systems in [53] we used $\Omega$EQA's emacs interface and simply stored the generated output afterwards in a file. This presentation is useful for tracing the interaction between user and $\Omega$EQA in great detail and, hence, allows you an easy comparison with other systems. However, the trace neglects $\Omega$EQA's graphical user interface $\LaTeX$ [46], which is by far better suited for the human user of $\Omega$EQA than the emacs interface.

3 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, which is computed when the tactic is executed. This proof plan can be recursively expanded, until we have reached a proof plan, which is in fact a fully explicit proof, since all nodes are justified by the inference rules of a higher-order variant of Gentzen's calculus of natural deduction (ND). In $\Omega$EQA, 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 $\mathcal{P}rex$ or analogical transfer of proof plans.

The lowest level of abstraction of a $\mathcal{PDS}$ is the level of Gentzen's ND calculus. A $\mathcal{PDS}$ can be constructed manually on this level. Several ND rules can be applied in many directions, for instance, backwards to close an existing open node (the conclusion) by generating new open nodes (the premises), or forwards to deduce a new node from some other existing nodes. For each ND rule, there is a command\footnote{To get an overview, type help to enter the HELP on $\Omega$EQA interpreter and then commands rules (to exit HELP type exit).} in $\Omega$EQA that allows the user or the planner to apply the rule in different directions. 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 NIL is given for every premise. All ND rules are "agentified", that is, the agent for this rule searches proactively for a formula that fulfills the rule's application condition, and when the agent succeeds it suggests its rule.

Tactics and methods are generalizations of rules that are also agentified and applied similarly. But they have a somewhat different ontological status: Just as ND rules, they construct a $\mathcal{PDS}$ node with a justification that cites the name of the tactic or the method, but these are not elementary, but represent a sub-$\mathcal{PDS}$ consisting of nodes with justifications on a lower level of abstraction. In particular, tactic justifications can be expanded by the command expand-node to the $\mathcal{PDS}$ they represent. In contrast to the set of rules (which is pre-defined in $\Omega$EQA), the set of tactics and methods can be arbitrarily extended by the user.

Moreover, it is worth mentioning that there can be more than one proof object for a given problem. Thus, $\Omega$EQA allows for the simultaneous representation of different proofs of the same problem.

The final proof object generated by $\Omega$EQA in our case study is illustrated in the Appendices A resp. B, where the unexpanded and the expanded proof object are presented in $\text{MPLX}$ resp. $\text{POST}$ format.
4 Our Proof of Choice

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 \( p, q \) with no common divisor such that

\[
\sqrt{2} = \frac{p}{q}.
\]

Then

\[
q \sqrt{2} = p,
\]

and thus

\[
2q^2 = p^2.
\]

Hence \( p^2 \) is even and, since odd numbers square to odds, \( p \) is even; say

\[
p = 2m.
\]

Then

\[
2q^2 = (2m)^2 = 4m^2,
\]

that is,

\[
q^2 = 2m^2.
\]

Hence, \( q^2 \) is even, too, and so is \( q \). Then, however, both \( q \) and \( p \) are even, contradicting the fact that they have no common divisor.

q.e.d.

5 Problem Formalization

5.1 The Proof Problem

We begin with formulating the theorem in Ωmega’s knowledge base as an open problem in the theory real. The problem is encoded in 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. These definitions are not needed in the interactive session as illustrated below. We nevertheless present the definitions of rat and sqrt here, to show how the knowledge base is built up, as these two concepts refer in turn to other defined concepts, such as frac and power in Ωmega’s structured knowledge base. While it is not necessary to understand all the details of the actual low-level code of the knowledge base, we give the following hints for the technically interested reader: that is the ε-operator and exists-sort takes two arguments: The first argument, for example (lam (z num) ...) in the following definition of rat, is a λ-expression that defines and binds a variable z of [hard] type num. The second argument (pos-nat in the example) encodes the (soft) sort of variable z bound in the λ-expression in the first argument.
(th`defdef rat
   (in rational)
   (definition
     (lam (x num)
       (exists-sort (lam (y num)
         (exists-sort (lam (z num)
           (and (not (= (mod x y) zero))
             (= x (frac y z)))
           pos-nat))
         int)))
     (help "The set of rationals, constructed as reduced fractions a/b of integers.\n"))

(th`defdef sqrt
   (in real)
   (definition
     (lam (x num)
       (that (lam (y num) (= (power y 2) x)))
     )
     (help "Definition of square root.\n"))

5.2 Further Definitions and Lemmata

To prove the stated problem the system needs further mathematical knowledge. Our proof employs the definition of evenp and some lemmata about the concepts rat, common-divisor, and evenp. These lemmata are also proved with \Omega\textsc{mega} and require the definitions 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 \textsc{maple} to justify the transformation of $q\sqrt{2} = p$ into $2q^2 = p^2$. To do so, \Omega\textsc{mega} expressions, such as $\sqrt{2}$, are mapped to corresponding \textsc{maple} representations and \textsc{maple} uses its own built-in knowledge to manipulate them. Using and verifying these computation steps requires expansion of \textsc{maple}'s computation to the calculus layer in \Omega\textsc{mega}. This is done by replaying \textsc{maple}'s computation by special computational tactics in \Omega\textsc{mega}. These tactics and their expansions, which are part of the \textsc{sapper} system, correspond directly to the mathematical definitions available in \Omega\textsc{mega}'s knowledge base. The natural number 2 is defined in theory \texttt{natural} as $s(s(0))$, where $s$ stands for the successor function. Again, this knowledge is only required when expanding the abstract proof to the basic calculus layer.

All the knowledge required at the interaction layer, however, is given in the definitions that follow.

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

(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.\n"))
(th'defthm 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.\n"))

(th'defthm even-common-divisor
   (in integer)
   (conclusion
     (forall-sort (lam (x num)
                   (forall-sort (lam (y num)
                                 (implies (and (evenp x) (evenp y))
                                           (common-divisor x y 2)))
                   int))
      int))
   (help "If x and y are even, then they have a common divisor.\n")

   These definitions depend in turn on the theories real, rational, integer and natural, which
are given as hierarchical theories in MBase. Here, we present only some further definitions from
these theories. Appendix D gives the complete Omega theories real, rational, integer and
natural.

(th'defdef common-divisor
   (in integer)
   (definition
     (lam (x num)
       (lam (y num)
         (lam (z num)
           (and (and (in x int) (in y int))
                (and (in z int)
                     (and (not (= 1 z))
                          (and (divisor z x) (divisor z y)))))))
       (help "The predicate for non-trivial common integer divisibility.\n"))

(th'defdef power
   (in natural)
   (definition
     (lam (n num)
       (recursion (lam (x num) (times m) one)))
       (help "Exponentiation defined as iterated multiplication.\n"))

(th'defdef times
   (in natural)
   (definition
     (lam (m num)
       (recursion (lam (x num) (plus m) zero)))
       (help "Multiplication defined as iterated addition.\n"))

(th'defdef plus
   (in natural)
   (definition
     (recursion (lam (x num) s)))
     (help "Addition defined as iterated application of successor.\n"))

(th'defdef recursion
   (in natural)
   (definition
     (lam (h ((num num) num))
       (lam (g num)
         (lam (n num)
           (that
             (lam (n num)
               \}))
             \}))

10
(foralll
  (lam (U (x: num num))
    (implies (and (U x zero) g)
      (foralll
        (lam (y: num)
          (foralll
            (lam (x: num)
              (implies (U x y)
                (implies (U (s x) (h x)))))))))
    (U n))))))))

6 Interactive Theorem Proving in ΩMEGA

We shall now present a detailed protocol of the interactive session with ΩMEGA. To allow for comparison with other systems in [53] we used ΩMEGA’s emacs interface and simply stored the generated output afterwards in a file. This presentation is useful for tracing the interaction between user and ΩMEGA in great detail and, hence, allows you an easy comparison with other systems. However, the trace neglects ΩMEGA’s graphical user interface (GUI) [46], which is by far better suited for the human user of ΩMEGA than the emacs interface.

The following is the actual run of the session with some comments added manually in the protocol. These additional comments are indicated in the protocol by the prefixes "***". In order to shorten the presentation here we removed some less interesting system output. The proof construction is a successive process of inference rule applications, that is, tactics and ND rules. Each application refines the proof under construction by either justifying existing proof lines or adding new (open or justified) proof lines. In the trace, we show all affected proof lines after each inference step, respectively, using show-line or show-ps commands.

We present ND proofs in a linearized style. A proof line is of the form ‘L (Δ) ![F : R]’, where L is a unique label, ![Δ] ! F denotes that the formula F can be derived from the formulae whose labels occur in the list Δ, and R is a justification expressing how the line was derived in a proof. For instance, the proof line

L2 (L1) ![FALSE] EXISTSE-SORT:(N) (L3 L5)

is a proof line with label L2 and denotes that the formula FALSE (the primitive falsity of our logic) can be derived assuming the hypothesis represented by proof line L1. In the proof under construction L2 was derived from the proof nodes L3 and L5 by an application of the inference rule EXISTSE-SORT (with some further parameter N).

*** We will now load the theory Real, in which the problem is defined.

***

ÔMEGA: load-problems
THEORY-NAME (EXISTING-THEORY) The name of a theory whose problems are to be loaded: [REAL]real

| Rules loaded for theory REAL |
| Theorems loaded for theory REAL |
| Tactics loaded for theory REAL |
| Methods loaded for theory REAL |
| Rules loaded for theory REAL |
| Theorems loaded for theory REAL |
| Tactics loaded for theory REAL |
| Methods loaded for theory REAL |
| Rules loaded for theory REAL |
| Theorems loaded for theory REAL |
| Tactics loaded for theory REAL |
| Methods loaded for theory REAL |
| Rules loaded for theory REAL |

11
::: Theorems loaded for theory REAL.
::: Tactics loaded for theory REAL.
::: Methods loaded for theory REAL.
::: Rules loaded for theory REAL.
::: Theorems loaded for theory REAL.
::: Tactics loaded for theory REAL.
::: Methods loaded for theory REAL.
::: Rules loaded for theory REAL.
::: Theorems loaded for theory REAL.
::: Tactics loaded for theory REAL.
::: Methods loaded for theory REAL.
::: Rules loaded for theory REAL.

*** Step: 1
*** First we load the problem from the OMEGA database and declare some
*** constant symbols which we shall use later on.

OMEGA: prove sqrt2-not-rat
Changing to proof plan SQRT2-NOT-RAT-21

OMEGA: show-pds

\[
\text{SQRT2-NOT-RAT}() \quad \text{! (\text{RAT} (\text{SQRT} 2))} \\
\text{OPEN}
\]

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

*** Step: 2
*** We prove the goal indirectly.

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

OMEGA: show-pds

\[
L_1 \quad (L_1) \quad \text{! (RAT (SQRT 2))} \\
\text{HYP}
\]

\[
L_2 \quad (L_1) \quad \text{! FALSE} \\
\text{OPEN}
\]

\[
\text{SQRT2-NOT-RAT}() \quad \text{! (RAT (SQRT 2))} \\
\text{NUTI: (L_2)}
\]

*** Step: 3
*** We load the theorem RAT-CRITERION from the database.
*** (This has the side effect that the newly introduced proof line
*** containing the theorem is added to the hypotheses lists of all
*** other proof lines.)

OMEGA: import-ass rat-criterion

OMEGA: show-pds

\[
L_1 \quad (L_1) \quad \text{! (RAT (SQRT 2))} \\
\text{HYP}
\]

\[
L_2 \quad (L_1) \quad \text{! FALSE} \\
\text{OPEN}
\]

\[
\text{RAT-CRITERION (RAT-CRITERION)}! \quad \text{(FORALL-SORT (X)) (EXISTS-SORT (Y)) (EXISTS-SORT (Z)) (AND (TIMES X Y Z) (NUT (EXISTS-SORT (D) (COMMON-DIVISOR Y Z D))))} \\
\text{TIM}
\]

12
*** Step: 4
*** We instantiate now the (sorted) universal quantifier of
*** RAT-CRITERION with term (sqrt 2). Thereby we employ the information
*** in L1 saying that (sqrt 2) is of sort RAT.

OMEGA: forallle-sort
UNIV-LINE (NDLINE) Universal line: [RAT-CRITERION]
LINE (NDLINE) A line: [(]
TERM (TERM) Term to substitute: (sqrt 2)
SL-LINE (NDLINE) A line with sort: [L1]
:::CSM Arbitrary [2]: O provers have to be killed

OMEGA: show-line* (rat-criterion 13)

RAT-CRITERION (RAT-CRITERION) ! (FORALL-SORT TERT
  (X).
  (EXISTS-SORT (Y).
    (EXISTS-SORT (Z).
      (AND (= (TIMES X Y) Z)
        (NOT (EXISTS-SORT (D) (COMMON-DIVISOR Y Z D)))
        INT))
    INT))
  INT))
RAT)

L3 (L1) ! (EXISTS-SORT FURALLE-SORT:
  (DC-4246)
  (EXISTE-SORT (RAT-CRITERION L1)
    (AND (= (TIMES (SORT 2) DC-4246)
      DC-4261)
    (NOT (EXISTS-SORT (DC-4265)
      (COMMON-DIVISOR DC-4248 DC-4261 DC-4265))
      INT))
  INT))

*** Step: 5
*** We eliminate the first (sorted) existential quantifier by introducing
*** constant n. This generates the additional information that n is of sort
*** integer in line L4.

OMEGA: existe-sort
EX-LINE (NDLINE) An existential line: [L5]
LINE (NDLINE) A line to be proved: [L2]
PARAM (TERM) A term: [dc-42481
PREM (NDLINE) The second premise line: [(]

OMEGA: show-line* (12 13 14 15)

L2 (L1) ! FALSE

L3 (L1) ! (EXISTS-SORT FURALLE-SORT:
(DC-4248).
(EXISTS-SORT
  ([DC-4261]
    (AND (~ DC-4261)
     (NOT (EXISTS-SORT ([DC-4265]
                        (COMMON-DIVISOR DC-4248 DC-4261 DC-4265))
                          INT))))
  INT))

L4  (LA)  ! (AND (INT N)
     (EXISTS-SORT
      ([DC-4260]
       (AND (~ (TIMES (SURT 2) N) DC-4260)
        (NOT (EXISTS-SORT ([DC-4264]
                           (COMMON-DIVISOR N DC-4260 DC-4264))
                          INT))))
     INT))

L5  (LA L1)  ! FALSE

*** Step: 6
*** We split the obtained conjunction in L4 in its conjuncts.
OMEGA: ande
CONJUNCTION (NOLINE) Conjunction to split: [L4]
LCLNJ (NOLINE) Left conjunct: []
RCNJ (NOLINE) Right conjunct: []

OMEGA: show-line* (16 17)

L6  (LA)  ! (INT N)
     ANDE: (L4)

L7  (LA)  ! (EXISTS-SORT
     ([DC-4260]
      (AND (~ (TIMES (SURT 2) N) DC-4260)
       (NOT (EXISTS-SORT ([DC-4264]
                          (COMMON-DIVISOR N DC-4260 DC-4264))
                          INT))))
     INT)

*** Step: 7
*** We eliminate the second (sorted) existential quantifier by introducing
*** constant m. This introduces the conjunction in line L8.
OMEGA: exists-sort
EX-LINE (NOLINE) An existential line: [L8] 17
LINE (NOLINE) A line to be proved: [L5]
PARAM (TERMDEF) A term: [dc-42601]m
PREM (NOLINE) The second premise line: []

OMEGA: show-line* (17 15 18 19)

L7  (LA)  ! (EXISTS-SORT
     ([DC-4260]
      (AND (~ (TIMES (SURT 2) N) DC-4260)
       (NOT (EXISTS-SORT ([DC-4264]
                          (COMMON-DIVISOR N DC-4260 DC-4264))
                          INT))))
     INT)
L5  (L4 L1)  ! FALSE  EXISTSE-SORT: (M) (L7 L9)

L8  (L8)  ! (AND (INT M) HYP
       (AND (= (TIMES (SURT 2) N) M)
       (NOT (EXISTSE-SORT ([DC-4270]. (COMMON-DIVISOR N M DC-4270))
            INT)))))

L9  (L8 L4 L1)  ! FALSE  OPEN

*** Step: 8
*** We split the conjunction in line L8 into its conjuncts.
OMEGA: ande:
CONJUNCT-LIST (NDLINE) Premises to split: L8
CONJUNCTION (NDLINE-LIST) List of conjuncts: ()

OMEGA: show-line* (110 111 112)

L10 (L8)  ! (INT M)  ANDE*: (L8)

L11 (L8)  ! (= (TIMES (SURT 2) N) M)  ANDE*: (L8)

L12 (L8)  ! (NOT (EXISTSE-SORT ([DC-4270]. (COMMON-DIVISOR N M DC-4270))
            INT))  ANDE*: (L8)

*** Step: 9
*** We want to infer from (= (TIMES (SURT 2) N) M) in L11 that
*** (= (POWER M 2) (TIMES 2 (POWER N 2))). To do so, we anticipate the
*** later formula by introducing it as a lemma for the current open
*** subgoal L9. Thereby, the new lemma is supposed to be derivable from the
*** same proof lines as L9.
OMEGA: lemma
NODE (NDPLAN-LINE) An open node: [L9]
FORMULA (FORMULA) Formula to be proved as lemma: (= (power m 2) (times 2
               (power n 2)))

OMEGA: show-line* (113)

L13 (L6 L4 L1)  ! (= (POWER M 2) (TIMES 2 (POWER N 2)))  OPEN

*** Step: 10
*** The lemma is proven by calling the computer algebra system Maple;
*** the command for this is BY-COMPUTATION. The computation problem is
*** passed from OMEGA to the mathematical software bus MapleWeb, which
*** in turn passes the problem to an available instance of MAPLE
*** somewhere on the Internet.
OMEGA: by-computation
LINE1 (NDLINE) A line an arithmetic term to justify.: 113
LINE2 (NDLINE-LIST) A list containing premises to be used.: (111)

OMEGA: show-line* (111 113)

L11 (L8)  ! (= (TIMES (SURT 2) N) M)  ANDE*: (L8)

15
L13  (L8 L4 L1) ! (= (POWER M 2) (TIMES 2 (POWER N 2)))  BY-COMPUTATION: (L11)

*** Step: 11
*** L13 already shows the criterion for (POWER M 2) to be even. We
*** anticipate this result and introduce (EVENP (POWER M 2)) as a
*** lemma.
OMEGA: lemma
NODE (NDFPLANLINE) An open node: [L9]
FORMULA (FORMULA) Formula to be proved as lemma: (evenp (power m 2))

OMEGA: show-line* (114)

L14  (L8 L4 L1) ! (EVENP (POWER M 2))  OPEN

*** Step: 12
*** The lemma can now be justified by the definition of evenp.
*** Unfortunately we cannot immediately use this definition in L13.
*** Further steps are required to ensure that (POWER N 2) is indeed an
*** integer.
OMEGA: defin-contract
LINE (NDFLINE) Line to be rewritten: [L14]
DEFINITION (THY-ASSUMPTION) Definition to be contracted: [EVENP]
POS (POSITION) Position of occurrence: [(0)]

OMEGA: show-line* (115 114)

L15  (L8 L4 L1) ! (EXISTS-SORT ([DC-4278]. (= (POWER M 2) (TIMES 2 DC-4278)))
     INT)  OPEN

L14  (L8 L4 L1) ! (EVENP (POWER M 2))

DefnI:
  (EVENP ([X]. (EXISTS-SORT ([Y]. (= X (TIMES 2 Y))) INT)) (0))
  (L15)

*** Step: 13
*** We now show that (POWER N 2) is an integer. This is the term we
*** want to instantiate for the existential variable in line L15 in
*** order to justify line L13.
OMEGA: lemma
NODE (NDFPLANLINE) An open node: [L15]19
FORMULA (FORMULA) Formula to be proved as lemma: (int (power n 2))

OMEGA: show-line* (116)

L16  (L8 L4 L1) ! (INT (POWER N 2))  OPEN

*** Step: 14
*** (POWER N 2) is indeed an integer and this can be verified by
*** application of the tactic WELLSORTED. WELLSORTED thereby employs
*** for instance, the information that n is an integer.
OMEGA: wellsorted
LINE (NDFLINE) A line with sort: [L16]
PREMISES (NDFLINE-LIST) A list of premises: [(L10 L1 L6)]
Omega: show-line* (l16)

l16 (la la li) ! (int (power n 2))

Wellsorted:
((power n (s (s zero))) int power-int-closed)
((s (s zero)) int nat-int)
((s (s zero)) nat succ-nat)
((zero nat zero-nat))

*** Step: 15
*** Now we can complete this part of the proof and successfully connect
*** l15 and l13.
Omega: existential-sort
Ex-line (nd-line) Existential line to prove: [l16]
Param (term?) Witness term: (power n 2)
Line (nd-line) A line: [[]]113
Su-line (nd-line) A line with sort: [l16]
Pos-list (position-list) The position(s) of the witness term: [[[2 2]]]
Omega: show-line* (113 115 l16)

l13 (la la li) ! (¬ (power m 2) (times 2 (power n 2)))

By-computation: (l11)

l15 (la la li) ! (exists-sort

[(dc-4278). (¬ (power m 2) (times 2 dc-4278))] (power n 2) ((2 2))

exist-si-sort: (l13 l16)

l16 (la la li) ! (int (power n 2))

Wellsorted:
((power n (s (s zero))) int power-int-closed)
((s (s zero)) int nat-int)
((s (s zero)) nat succ-nat)
((zero nat zero-nat))

*** Step: 16
*** Now we come back to our now fully justified intermediate result in
*** l14 saying that (evenp (power m 2)). From this we want to conclude
*** that (evenp n) holds by application of a respective theorem in the
*** database. First we load the theorem.
Omega: import-ass
Ass-name (thy-assumption) A name of an assumption to be imported from the problem
type: square-even
Omega: show-line* (l14 square-even)

l14 (la la li) ! (evenp (power m 2))

Defn:
(evenp (dc). (exists-sort ([y]. (¬ x (times 2 y))) int)) (0)
(l15)

square-even (square-even) ! (for-all-sort ([x]. (equiv (evenp (power x 2)) (evenp x))))

Termination

*** Step: 17
*** Next we assert that (EVENP M) holds. By application of the assert
*** tactic the introduced goal is automatically tackled by provers
*** connected via MathWeb to OMEGA. In this case the system UTTER is
*** called.

OMEGA: assert

FORMULA (THEM) A formula: [false](evenp n)
PROOF-LINES (NDLINE-LIST) Depends on proof lines: [(SQUARE-EVEN RAT-CRITERION)]
(square-even l10 l14)
DEFIS (THN-ASS-LIST) A list of definitions that should be expanded: [(EVENP)]()

Normalizing ...

OMEGA: LINE: [1] 29680
Calling otter process 29680 with time resource 10sec .

otter Time Resource in seconds:
10sec

--------- PROOF --------

Search stopped by max proofs option.

UTTER HAS FOUND A PROOF

;; CSM Arbitrary [2]: O provers have to be killed

OMEGA: show-line* (l10 square-even l14 l17)

L10 (L1) ! (INT M) ANDE*: (L1)

SQUARE-EVEN (SQUARE-EVEN) ! (FORALL-SORT ([X]. (EQUIV (EVENP (POWER X 2)) (EVENP X)))) TYP
INT)

L14 (L1 LA L1) ! (EVENP (POWER M 2)) DefnI:
  (EVENP ([X]. (EXISTS-SORT ([Y]. (- X (TIMES 2 Y)))) INT)) (O)) (L16)

L17 (L1 LA L1) ! (EVENP M) ASSERT: ((EVENP M) NIL) (SQUARE-EVEN L10 L14)

*** Step: 18
*** Next we expand the definition of EVENP in L17.

OMEGA: defin-expand

LINE (NDLINE) Line to be rewritten: [SQUARE-EVEN] l17
DEFINITION (THN-ASSUMPTION) Definition to be expanded: [EVENP]
POSITION (POSITION) Position of occurrence: [(O)]

OMEGA: show-line* (l17 l18)

L17 (L1 LA L1) ! (EVENP M) ASSERT: ((EVENP M) NIL) (SQUARE-EVEN L10 L14)

L18 (L1 LA L1) ! (EXISTS-SORT ([DC=4334]. (~ M (TIMES 2 DC=4334))) INT) DefnE:
  (EVENP ([X]. (EXISTS-SORT ([Y]. (~ X (TIMES 2 Y))) INT)) (O))
*** Step: 19
*** Then we eliminate the (sorted) existential quantifier and introduce
*** a constant k.

(SORT EXIST) An existential line: [L18]
(LINE 1) A line to be proved: [L9]
(PARAME (TERM_SYM) A term: [id~43341]k

(SORT PREM) The second premise line: [()]

**SMEX: show-line* (118 119 120)**

L18 (L8 L4 L1) ! (EXISTS-SORT (DC=4334). (~ M (TIMES 2 DC=4334))) INT

**DefnE:**

(EVENP (Xn). (EXISTS-SORT (Ym: ~ (X (TIMES 2 Y))). INT)) (G)
(L17)

L19 (L19) ! (AND (INT K) (~ M (TIMES 2 K)))

**HYP**

L20 (L19 L8 ! FALSE L4 L1)

**OPEN**

*** Step: 20
*** We immediately split the obtained conjunction in line L19.

**SMEX: ande**

(CONJUNCTION (TERM_SYM) Conjunction to split: [L4] L19

(LHSJ (TERM_SYM) Left conjunct: [()]

(RHSJ (TERM_SYM) Right conjunct: [()]

**SMEX: show-line* (119 121 122)**

L19 (L19) ! (AND (INT K) (~ M (TIMES 2 K)))

**HYP**

L21 (L19) ! (INT K)

**ANDE: (L19)**

L22 (L19) ! (~ M (TIMES 2 K))

**ANDE: (L19)**

*** Step: 21
*** With the help of the equation (~ M (TIMES 2 K)) in L22 and the
*** equation (~ (POW 2 N) (TIMES 2 (POW 2 K))) in L13 we now want
*** to infer that (~ (POW 2 N) (TIMES 2 (POW 2 K))) holds.

**SMEX: lemma**

(OPEN OPEN) An open node: [L20]

(FORMULA (FORMULA) Formula to be proved as lemma: (~ (POW 2 N) (TIMES 2 (POW 2 K)))

**SMEX: show-line* (123)**

L23 (L19 L8 ! (~ (POW 2 N) (TIMES 2 (POW 2 K))) L4 L1)

**OPEN**

*** Step: 22
*** This can be done again by calling a computer algebra system.
SOMEGA: by-computation
LINE1: (NDLINE) A line an arithmetic term to justify: 123
LINE2 (NDLINE-LIST) A list containing premises to be used: (L13 L122)
SOMEGA: show-line* (L13 L122 L123)

L13  (L6 L4 L1) ! (~ (POWER M 2) (TIMES 2 (POWER N 2))) BY-COMPUTATION: (L11)

L122 (L19) ! (~ M (TIMES 2 K)) ANDE: (L19)

L123 (L19 L8) ! (~ (POWER N 2) (TIMES 2 (POWER K 2))) BY-COMPUTATION: (L13 L22)

*** Step: 23
*** Similarly as before, where we derived (EVENP (POWER M 2)), we can
*** now infer that (EVENP (POWER N 2)) holds. We present the proof
*** steps here without further comments.
SOMEGA: lemma
NOIDE (NDPLANLINE) An open node: [L20]
FORMULA (FORMULA) Formula to be proved as lemma: (euenp (power n 2))

*** Step: 24
SOMEGA: defin-contract
LINE (NDLINE) Line to be rewritten: [L24]
DEFINITION (THY-ASSUMPTION) Definition to be contracted: [EVENP]
POS (POSITION) Position of occurrence: [[0]]

*** Step: 25
SOMEGA: lemma
NOIDE (NDPLANLINE) An open node: [L25]
FORMULA (FORMULA) Formula to be proved as lemma: (int (power k 2))

*** Step: 26
SOMEGA: wellsorted
LINE (NDLINE) A line with sort: [L26]
PREMISES (NDLINE-LIST) A list of premises: {(L16 L6 L1 L10 L21)}(121)

*** Step: 27
SOMEGA: existusi-sort
EX-LINE (NDLINE) Existential line to prove: [L28]
PARAM (TERM) Witness term: (power k 2)
LINE (NDLINE) A line: [(])123
SU-LINE (NDLINE) A line with sort: [L26]
POS-LIST (POSITION-LIST) The position(s) of the witness term: [[(2 2)]]
SOMEGA: show-line* (121 124 125 126)

L21 (L19) ! (INT K) ANDE: (L19)

L24 (L19 L8) ! (EVENP (POWER N 2))

L25 (L19 L8) ! (EVENP (POWER N 2))

L26 (L19 L8) ! (EXISTS-SORT EXISTS1-SORT:
14 L1) \[(\text{DC-4344}). \ ((- (\text{POWER N 2}) \ (T\text{IMES 2 DC-4344}))) \]
\[\ ((\text{POWER K 2}) \ ((2 2))) \]
\[\text{INT} \]
\[\text{(L23 L26)} \]

L26 \[(L19 \text{ L8}) \ (\text{INT (POWER K 2)}) \]
\[\ (((((\text{POWER K (S (S ZERO))} \ \text{INT POWER-INT-CLOSED})) \ ((S (S ZERO)) \ \text{INT NAT-INT})) \ ((S (S ZERO)) \ \text{NAT SUCC-NAT})) \ ((\text{ZERO NAT ZERO-NAT})))) \]
\[\text{(L21)} \]

*** Step: 28
*** Similar to before (steps 16 and 17), where we derived \((\text{EVENP N})\)
*** from \((\text{EVENP (POWER M 2)})\) application of theorem \text{SQUARE-EVEN}, we now
*** derive \((\text{EVENP N})\) from \((\text{EVENP (POWER N 2)})\). The assertion is
*** immediately closed by the external ATP UTTER.

\text{SHLEX: assert}

\text{FORMULA (TERM) A formula: [false](evenp n)}
\text{PROOF-LINES (HDLINE-LIST) Depends on proof lines: [(SQUARE-EVEN NAT-CRITERION)]}
\text{(square-even 16 124)}
\text{DEFIN (THY-ASSI-LIST) A list of definitions that should be expanded: [(EVENP)]}()

Normalizing ...
LINE: [1] 298008
Calling otter process 298008 with time resource 10sec .

otter Time Resource in seconds:
9sec

---------- PROOF ----------

Search stopped by max_proofs option.

UTTER HAS FOUND A PROOF

:::CSN! Arbitrary \([2]: 0\) provers have to be killed

\text{OMEGA: show-line* (L27)

L27 \[(L19 \text{ L8}) \ (\text{EVENP N}) \]
\[\text{ASSERT: ((EVENP N) \text{ NIL}) (SQUARE-EVEN L6 L24) \text{L4 L1)} \]

*** Step: 29
*** It remains to be shown that \((\text{EVENP N})\) and \((\text{EVENP M})\) contradict the
*** assumption in line L12 that \(N\) and \(M\) have no common divisor. For
*** this we first load the \text{EVEN-COMMON-DIVISOR} theorem from the
*** database.

\text{OMEGA: import-ax}

\text{ASS-NAME (THY-ASSUMPTION) A name of an assumption to be imported from the problem}
\text{theory: even-common-divisor}

\text{OMEGA: show-line* (even-common-divisor)

\text{EVEN-COMMON-DIVISOR \text{EVEN-COMMON-DIVISOR}) ! (FORALL-SORT \text{TBM \)
\[\text{[X].}
\text{\text{(FORALL-SORT ([Y]. (IMPLIES (AND (EVENP X) (EVENP Y))
\text{(COMMON-DIVISOR X Y 2)))
\text{INT)))
\text{INT}) \}

21
*** Step: 30
*** We also have to ensure that 2 is an integer.
OMEGA: lemma
NODE (NDPLANLINE) An open node: [L26]
FORMULA (FORMULA) Formula to be proved as lemma: (int 2)
OMEGA: show-line* (L26)

L26 (L19 L8 ! (INT 2) OPEN
L4 L1)

*** Step: 31
*** L26 can immediately be justified by tactic WELLSORTED.
OMEGA: wellsorted
LINE (NDLINE) A line with sort: [L26]
PREMISES (NDLINE-LIST) A list of premises: [(L16 L6 L1 L10 L21)]()
OMEGA: show-line* (L26)

L26 (L19 L8 ! (INT 2)
L4 L1) WELLSORTED:
(((S (S ZERO)) INT NAT-INT)
((S (S ZERO)) NAT SOKC-NAT)
((S ZERO) NAT SOKC-NAT)
(ZERO NAT ZERO-NAT))

*** Step: 32
*** The final contradiction is now easily established by any ATP
*** available via MathWeb (OTTER in this example).
OMEGA: assert
FORMULA (TER(M)) A formula: [false]
DEFIS (THY-ASS-LIST) A list of definitions that should be expanded: []

Normalizing ...
LINE: [1] 290900
Calling otter process 290900 with time resource 10sec .

.otter Time Resource in seconds: 9sec

-------- PROOF --------

Search stopped by max.proofs option.

OTTER HAS FOUND A PROOF

;;;CSM Arbitrary [2]: 0 provers have to be killed

OMEGA: show-line*

LINES (NDLINE-LIST) A list of lines to be shown: (L29)

L29 (L19 L8 ! FALSE ASSERT: (FALSE NIL) (EVEN-COMMON-DIVISOR L10 L6 L12 L17 L27 L26) L4 L1)

22
*** Step: 33
*** We complete the proof by connecting the contradiction in L29 with
*** L20.
OMEGA: weaken
LOWERNELINE (NDLINE) Line to justify: [L29]
UPPERLINE (NDLINE) Already-derived line: [L29]

OMEGA: show-line* (L29 120)

L29  (L19 L8  ! FALSE  ASSERT: (FALSE NIL) (EVEN-COMMON-DIVISOR L10 L6 L12 L17 L27 L28)
     L4 L1)

L20  (L19 L8  ! FALSE  WEAKEN: (L29)
     L4 L1)

*** Step: 34
*** The proof is complete now. We might be interested to check whether the proof
*** is logically correct. This can be done by applying the command CHECK-PROOF.
*** First the proof will be fully expanded to base calculus level and then an
*** independent proof checker investigates logical correctness of the single
*** base calculus inference steps.
*** Note that some expansion steps require the application of external ATPs.
*** Unfortunately we cannot fully expand yet the computations occurring in our
*** proof. Thus our proof can for the moment only be verified modularly.
*** correctness of these computations steps. The expansion of the by-computation
*** wild-tactics is work in progress (remark: An OMEGA wild-tactic is a tactic
*** whose outline pattern, i.e. the pattern of premises and conclusions, is not
*** statically determined).

OMEGA: check-proof
TACTIC-LIST (SYMBOL-LIST) The tactics that should not be expanded: [(])
(by-computation)
Expanding nodes......
  Expanding the node L22 ...
  Expanding the node L12 ...
  Expanding the node L11 ...
  Expanding the node L10 ...
  Expanding the node L7 ...
  Expanding the node L2 ...
  Expanding the node L9 ...
  Expanding the node L5 ...

***
*** Here we did cut out some OMEGA output of the form 'Expanding the node Lxy ...'
***

Expanding the node L50 ...
Expanding the node L17 ...
Expanding line L17 justified by OTTER call
THE NODE: (evenp m) #<Justified by OTTER from (L14 L10 L50)>
Normalizing ...
LINE: [1] 30008
Calling otter process 30008 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
OMEGA-CURRENT-RESOLUTION-PROOF IS SET TO THE FOUND RESOLUTION PROOF
Searching for lemmata ...

23
Creating Refutation-Graph ...
Translating ...
Translation finished!
Expanding the node L26 ...
Expanding the node L27 ...
Expanding line L27 justified by OTTER call
THE NODE: (even n) #<Justified by OTTER from (L24 L6 L59)>

Normalizing ...
LINE: [1] 00090
Calling otter process 00090 with time resource 10 sec.

otter Time Resource in seconds: 9 sec
------- PROOF -------
Search stopped by max_proofs option.
Parsing Otter Proof ...
OTTER HAS FOUND A PROOF
OMEGA+CURRENT-RESOLUTION-PROOF IS SET TO THE FOUND RESOLUTION PROOF
Searching for lemmata ...
Creating Refutation-Graph ...
Translating ...
Translation finished!
Expanding the node L66 ...
Expanding the node L64 ...

*** Here we did cut out some OMEGA output of the form 'Expanding the node Lxy ...
***

Expanding the node L137 ...
Expanding the node L136 ...
Expanding the node L113 ...
Expanding the node L161 ...
Expanding the node L149 ...
Expanding the node L154 ...
Expanding the node L113 ...

*** The proof is now fully expanded and the system applies the proof checker.
***

Checking nodes.
Node #<pdsn> node #L131> has a correct justification.
Node #<pdsn> node #L165> has a correct justification.
Node #<pdsn> node #L165> has a correct justification.
Node #<pdsn> node #L166> has a correct justification.
Node #<pdsn> node #L169> has a correct justification.
Node #<pdsn> node #L170> has a correct justification.
Node #<pdsn> node #L169> has a correct justification.
Node #<pdsn> node #L167> has a correct justification.
Node #<pdsn> node #L163> has a correct justification.
Node #<pdsn> node #L133> has a correct justification.
Node #<pdsn> node #L136> has a correct justification.
Node #<pdsn> node #L174> has a correct justification.
Node #<pdsn> node #L170> has a correct justification.
Node #<pdsn> node #L170> has a correct justification.
Node #<pdsn> node #L137> has a correct justification.
Node #<pdsn> node #L134> has a correct justification.
Node #<pdsn> node #L189> has a correct justification.
Node #<pdsn> node #L179> has a correct justification.
Node #<pdsn> node #L195> has a correct justification.
Node #<pdsn> node #L195> has a correct justification.
Node #<pdsn> node #L22> has a correct justification.
Node #<pdsn> node #L121> has a correct justification.
Node #<pdsn> node #L139> has a correct justification.
Node #<pdsn> node #L230> has a correct justification.
Node #<pdsn> node #L12> has a correct justification.
Node #<pdsn> node #L111> has a correct justification.
Node #<pdsn> node #L110> has a correct justification.
Node #<pdsn> node #L14> has a correct justification.
Node #<pdsn> node #L7> has a correct justification.

24
Node #<pdsn+node #:L69> has a correct justification.
Node #<pdsn+node #:L11> has a correct justification.
Node #<pdsn+node #:L133> has a correct justification.
Node #<pdsn+node #:L129> has a correct justification.
Node #<pdsn+node #:L1290> has a correct justification.
Node #<pdsn+node #:L166> has a correct justification.
Node #<pdsn+node #:L169> has a correct justification.
Node #<pdsn+node #:L159> has a correct justification.
Node #<pdsn+node #:L1500> has a correct justification.
Node #<pdsn+node #:L126> has a correct justification.
Node #<pdsn+node #:L1269> has a correct justification.
Node #<pdsn+node #:L165> has a correct justification.
Node #<pdsn+node #:L1111> has a correct justification.
Node #<pdsn+node #:L1127> has a correct justification.
Node #<pdsn+node #:L1129> has a correct justification.
Node #<pdsn+node #:L1130> has a correct justification.
Node #<pdsn+node #:L1139> has a correct justification.
Node #<pdsn+node #:L1194> has a correct justification.
Node #<pdsn+node #:L1196> has a correct justification.
Node #<pdsn+node #:L1197> has a correct justification.
Node #<pdsn+node #:L1198> has a correct justification.
Node #<pdsn+node #:L1199> has a correct justification.
Node #<pdsn+node #:L1191> has a correct justification.
Node #<pdsn+node #:L1192> has a correct justification.
Node #<pdsn+node #:L1193> has a correct justification.
Node #<pdsn+node #:L11960> has a correct justification.
Node #<pdsn+node #:L11971> has a correct justification.
Node #<pdsn+node #:L11982> has a correct justification.
Node #<pdsn+node #:L11993> has a correct justification.

:::WARNING: The application of the wild-tactic #<INFER+WILD-TACTIC BY-COMPUTATION> is not checked
:::WARNING: Node #<pdsn+node #:L130> with the formula (= (power n 2) (times 2 (power n 2))) has an uncorrect or unknown justification!
Node #<pdsn+node #:L143> has a correct justification.
Node #<pdsn+node #:L142> has a correct justification.
Node #<pdsn+node #:L141> has a correct justification.
Node #<pdsn+node #:L140> has a correct justification.
Node #<pdsn+node #:L139> has a correct justification.
Node #<pdsn+node #:L138> has a correct justification.
Node #<pdsn+node #:L137> has a correct justification.
Node #<pdsn+node #:L136> has a correct justification.
Node #<pdsn+node #:L135> has a correct justification.
Node #<pdsn+node #:L134> has a correct justification.
Node #<pdsn+node #:L133> has a correct justification.
Node #<pdsn+node #:L132> has a correct justification.
Node #<pdsn+node #:L115> has a correct justification.
Node #<pdsn+node #:L114> has a correct justification.
Node #<pdsn+node #:L113> has a correct justification.
Node #<pdsn+node #:L112> has a correct justification.
Node #<pdsn+node #:L111> has a correct justification.
Node #<pdsn+node #:L110> has a correct justification.
Node #<pdsn+node #:L1100> has a correct justification.
Node #<pdsn+node #:L109> has a correct justification.
Node #<pdsn+node #:L108> has a correct justification.
Node #<pdsn+node #:L107> has a correct justification.
Node #<pdsn+node #:L106> has a correct justification.
Node #<pdsn+node #:L105> has a correct justification.
Node #<pdsn+node #:L104> has a correct justification.
Node #<pdsn+node #:L103> has a correct justification.
Node #<pdsn+node #:L102> has a correct justification.
Node #<pdsn+node #:L101> has a correct justification.
Node #<pdsn+node #:L100> has a correct justification.
Node #<pdsn+node #:L99> has a correct justification.
Node #<pdsn+node #:L98> has a correct justification.
Node #<pdsn+node #:L97> has a correct justification.
Node #<pdsn+node #:L96> has a correct justification.
Node #<pdsn+node #:L95> has a correct justification.
Node #<pdsn+node #:L94> has a correct justification.
Node #<pdsn+node #:L93> has a correct justification.
Node #<pdsn+node #:L92> has a correct justification.
Node #<pdsn+node #:L91> has a correct justification.
Node #<pdsn+node #:L90> has a correct justification.
Node #<pdsn+node #:L89> has a correct justification.
Node #<pdsn+node #:L88> has a correct justification.
Node #<pdsn+node #:L87> has a correct justification.
Node #<pdsn+node #:L86> has a correct justification.
Node #<pdsn+node #:L85> has a correct justification.
Node #<pdsn+node #:L84> has a correct justification.
Node #<pdsn+node #:L83> has a correct justification.
Node #<pdsn+node #:L82> has a correct justification.
Node #<pdsn+node #:L81> has a correct justification.
Node #<pdsn+node #:L80> has a correct justification.
Node #<pdsn+node #:L79> has a correct justification.
Node #<pdsn+node #:L78> has a correct justification.
Node #<pdsn+node #:L77> has a correct justification.
Node #<pdsn+node #:L76> has a correct justification.
Node #<pdsn+node #:L75> has a correct justification.
Node #<pdsn+node #:L74> has a correct justification.
Node #<pdsn+node #:L73> has a correct justification.
Node #<pdsn+node #:L72> has a correct justification.
Node #<pdsn+node #:L71> has a correct justification.
Node #<pdsn+node #:L70> has a correct justification.
Node #<pdsn+node #:L69> has a correct justification.
Node #<pdsn+node #:L68> has a correct justification.
Node #<pdsn+node #:L67> has a correct justification.
Node #<pdsn+node #:L66> has a correct justification.
Node #<dns-node #:124> has a correct justification.
Node #<dns-node #:125> has a correct justification.
Node #<dns-node #:126> has a correct justification.
Node #<dns-node #:127> has a correct justification.
Node #<dns-node #:128> has a correct justification.
Node #<dns-node #:129> has a correct justification.
Node #<dns-node #:130> has a correct justification.
Node #<dns-node #:131> has a correct justification.
Node #<dns-node #:132> has a correct justification.
Node #<dns-node #:133> has a correct justification.
Node #<dns-node #:134> has a correct justification.
Node #<dns-node #:135> has a correct justification.
Node #<dns-node #:136> has a correct justification.
Node #<dns-node #:137> has a correct justification.
Node #<dns-node #:138> has a correct justification.
Node #<dns-node #:139> has a correct justification.
Node #<dns-node #:140> has a correct justification.
Node #<dns-node #:141> has a correct justification.
Node #<dns-node #:142> has a correct justification.
Node #<dns-node #:143> has a correct justification.
Node #<dns-node #:144> has a correct justification.
Node #<dns-node #:145> has a correct justification.
Node #<dns-node #:146> has a correct justification.
Node #<dns-node #:147> has a correct justification.
Node #<dns-node #:148> has a correct justification.
Node #<dns-node #:149> has a correct justification.
Node #<dns-node #:150> has a correct justification.
Node #<dns-node #:151> has a correct justification.
Node #<dns-node #:152> has a correct justification.
Node #<dns-node #:153> has a correct justification.
Node #<dns-node #:154> has a correct justification.
Node #<dns-node #:155> has a correct justification.
Node #<dns-node #:156> has a correct justification.
Node #<dns-node #:157> has a correct justification.
Node #<dns-node #:158> has a correct justification.
Node #<dns-node #:159> has a correct justification.
Node #<dns-node #:160> has a correct justification.
Node #<dns-node #:161> has a correct justification.
Node #<dns-node #:162> has a correct justification.
Node #<dns-node #:163> has a correct justification.
Node #<dns-node #:164> has a correct justification.
Node #<dns-node #:165> has a correct justification.
Node #<dns-node #:166> has a correct justification.
Node #<dns-node #:167> has a correct justification.
Node #<dns-node #:168> has a correct justification.
Node #<dns-node #:169> has a correct justification.
Node #<dns-node #:170> has a correct justification.
Node #<dns-node #:171> has a correct justification.
Node #<dns-node #:172> has a correct justification.
Node #<dns-node #:173> has a correct justification.
Node #<dns-node #:174> has a correct justification.
Node #<dns-node #:175> has a correct justification.
Node #<dns-node #:176> has a correct justification.
Node #<dns-node #:177> has a correct justification.
Node #<dns-node #:178> has a correct justification.
Node #<dns-node #:179> has a correct justification.
Node #<dns-node #:180> has a correct justification.
Node #<dns-node #:181> has a correct justification.
Node #<dns-node #:182> has a correct justification.
Node #<dns-node #:183> has a correct justification.
Node #<dns-node #:184> has a correct justification.
Node #<dns-node #:185> has a correct justification.
Node #<dns-node #:186> has a correct justification.
Node #<dns-node #:187> has a correct justification.
Node #<dns-node #:188> has a correct justification.
Node #<dns-node #:189> has a correct justification.
Node #<dns-node #:190> has a correct justification.
Node #<dns-node #:191> has a correct justification.
Node #<dns-node #:192> has a correct justification.
Node #<dns-node #:193> has a correct justification.
Node #<dns-node #:194> has a correct justification.
Node #<dns-node #:195> has a correct justification.
Node #<dns-node #:196> has a correct justification.
Node #<dns-node #:197> has a correct justification.
Node #<dns-node #:198> has a correct justification.
Node #<dns-node #:199> has a correct justification.
Node #<dns-node #:200> has a correct justification.
Node #<dns-node #:201> has a correct justification.
Node #<dns-node #:202> has a correct justification.
Node #<dns-node #:203> has a correct justification.
Node #<dns-node #:204> has a correct justification.
Node #<dns-node #:205> has a correct justification.
Node #<dns-node #:206> has a correct justification.
Node #<dns-node #:207> has a correct justification.
Node #<dns-node #:208> has a correct justification.
Node #<dns-node #:209> has a correct justification.
Node #<dns-node #:210> has a correct justification.
Node #<dns-node #:211> has a correct justification.
Node #<dns-node #:212> has a correct justification.
Node #<dns-node #:213> has a correct justification.
Node #<dns-node #:214> has a correct justification.
Node #<dns-node #:215> has a correct justification.
Node #<dns-node #:216> has a correct justification.
Node #<dns-node #:217> has a correct justification.
Node #<dns-node #:218> has a correct justification.
Node #<dns-node #:219> has a correct justification.
Node #<dns-node #:220> has a correct justification.
Node #<dns-node #:221> has a correct justification.
Node #<dns-node #:222> has a correct justification.
Node #<dns-node #:223> has a correct justification.
Node #<dns-node #:224> has a correct justification.
Node #<dns-node #:225> has a correct justification.
Node #<dns-node #:226> has a correct justification.
Node #<dns-node #:227> has a correct justification.
Node #<dns-node #:228> has a correct justification.
Node #<dns-node #:229> has a correct justification.
Node #<dns-node #:230> has a correct justification.
Node #<dns-node #:231> has a correct justification.
Node #<dns-node #:232> has a correct justification.
Node #<dns-node #:233> has a correct justification.
Node #<dns-node #:234> has a correct justification.
Node #<dns-node #:235> has a correct justification.
Node #<dns-node #:236> has a correct justification.
Node #<dns-node #:237> has a correct justification.
Node #<dns-node #:238> has a correct justification.
Node #<dns-node #:239> has a correct justification.
Node #<dns-node #:240> has a correct justification.
Node #<dns-node #:241> has a correct justification.
Node #<dns-node #:242> has a correct justification.
Node #<dns-node #:243> has a correct justification.
Node #<dns-node #:244> has a correct justification.
Node #<dns-node #:245> has a correct justification.
Node #<dns-node #:246> has a correct justification.
Node #<dns-node #:247> has a correct justification.
Node #<dns-node #:248> has a correct justification.
Node #<dns-node #:249> has a correct justification.
Node #<dns-node #:250> has a correct justification.
Node #<dns-node #:251> has a correct justification.
Node #<dns-node #:252> has a correct justification.
Node #<dns-node #:253> has a correct justification.
Node #<dns-node #:254> has a correct justification.
Node #<dns-node #:255> has a correct justification.
Node #<dns-node #:256> has a correct justification.
Node #<dns-node #:257> has a correct justification.
Node #<dns-node #:258> has a correct justification.
Node #<dns-node #:259> has a correct justification.
Node #<dns-node #:260> has a correct justification.
Node #<dns-node #:261> has a correct justification.
Node #<dns-node #:262> has a correct justification.
Node #<dns-node #:263> has a correct justification.
Node #<dns-node #:264> has a correct justification.
Node #<dns-node #:265> has a correct justification.
Node #<dns-node #:266> has a correct justification.
Node #<dns-node #:267> has a correct justification.
Node #<dns-node #:268> has a correct justification.
Node #<dns-node #:269> has a correct justification.
Node #<dns-node #:270> has a correct justification.
Node #<dns-node #:271> has a correct justification.
Node #<dns-node #:272> has a correct justification.
Node #<dns-node #:273> has a correct justification.
Node #<dns-node #:274> has a correct justification.
Node #<dns-node #:275> has a correct justification.
Node #<ndsnode :#1143> has a correct justification.
Node #<ndsnode :#1144> has a correct justification.
Node #<ndsnode :#1145> has a correct justification.
Node #<ndsnode :#1146> has a correct justification.
Node #<ndsnode :#1147> has a correct justification.
Node #<ndsnode :#1148> has a correct justification.
Node #<ndsnode :#1150> has a correct justification.
Node #<ndsnode :#1151> has a correct justification.
Node #<ndsnode :#1152> has a correct justification.
Node #<ndsnode :#1153> has a correct justification.
Node #<ndsnode :#1154> has a correct justification.
Node #<ndsnode :#1155> has a correct justification.
Node #<ndsnode :#1156> has a correct justification.
Node #<ndsnode :#1157> has a correct justification.
Node #<ndsnode :#1158> has a correct justification.
Node #<ndsnode :#1159> has a correct justification.
Node #<ndsnode :#1160> has a correct justification.
Node #<ndsnode :#1161> has a correct justification.
Node #<ndsnode :#1162> has a correct justification.
Node #<ndsnode :#1163> has a correct justification.
Node #<ndsnode :#1164> has a correct justification.
Node #<ndsnode :#1165> has a correct justification.
Node #<ndsnode :#1166> has a correct justification.
Node #<ndsnode :#1167> has a correct justification.
Node #<ndsnode :#1168> has a correct justification.
Node #<ndsnode :#1169> has a correct justification.
Node #<ndsnode :#1170> has a correct justification.
Node #<ndsnode :#1171> has a correct justification.
Node #<ndsnode :#1172> has a correct justification.
Node #<ndsnode :#1173> has a correct justification.
Node #<ndsnode :#1174> has a correct justification.
Node #<ndsnode :#1175> has a correct justification.
Node #<ndsnode :#1176> has a correct justification.
Node #<ndsnode :#1177> has a correct justification.
Node #<ndsnode :#1178> has a correct justification.
Node #<ndsnode :#1179> has a correct justification.
Node #<ndsnode :#1180> has a correct justification.
Node #<ndsnode :#1181> has a correct justification.
Node #<ndsnode :#1182> has a correct justification.
Node #<ndsnode :#1183> has a correct justification.
Node #<ndsnode :#1184> has a correct justification.
Node #<ndsnode :#1185> has a correct justification.
Node #<ndsnode :#1186> has a correct justification.
Node #<ndsnode :#1187> has a correct justification.
Node #<ndsnode :#1188> has a correct justification.
Node #<ndsnode :#1189> has a correct justification.
Node #<ndsnode :#1190> has a correct justification.
Node #<ndsnode :#1191> has a correct justification.
Node #<ndsnode :#1192> has a correct justification.
Node #<ndsnode :#1193> has a correct justification.
Node #<ndsnode :#1194> has a correct justification.
Node #<ndsnode :#1195> has a correct justification.
Node #<ndsnode :#1196> has a correct justification.
Node #<ndsnode :#1197> has a correct justification.
Node #<ndsnode :#1198> has a correct justification.
Node #<ndsnode :#1199> has a correct justification.
Node #<ndsnode :#1200> has a correct justification.

7 Proof Presentation

The emacs-based interface of Ωmega is usually only needed by the developers of Ωmega. who sometimes need additional information or access to internal data. The user, in contrast, controls the system only via the graphical user interface ΩMEGA. ΩMEGA displays information on the current proof state in multiple (cross-linked) modalities: a graphical map of the proof tree, a linearized presentation of the proof nodes with their formulae and justifications, and a term browser. When inspecting portions of a proof using 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. The ΩMEGA presentation of the final proof is given next:

Figure 2 provides a screenshot of ΩMEGA after completing but before expanding the proof, that
Figure 2: Presentation of the unexpanded proof in the graphical user interface $\Omega UI$. 

is, after Step 33 in Section 6. Figure 3 provides a screenshot of $\Omega UI$ after expanding the proof, that is, after Step 34 in Section 6.

$\Omega$ is also capable of translating its internal proof data structure into a representation in $\LaTeX$. A presentation of the unexpanded and expanded proof generated with this option is given in Appendix A.

Moreover, a natural language explanation of the proof is provided by the system $P$ [21, 24], which is interactive and adaptive. The system explains a proof step at the most abstract level that the user is assumed to know, and it reacts flexibly to questions and requests [22]. While the explanation is in progress, the user can interrupt $P$ anytime, if the current explanation is not satisfactory. $P$ analyzes the user’s interaction and enters into a clarification dialog when needed to identify the reason why the explanation was not satisfactory and re-plans a better explanation, for example, by switching to another level of abstraction [23]. Figure 4 displays the $P$ presentation of the proof in an $\LaTeX$ interface and Figure 5 its presentation in $\Omega UI$. The presentation in $\LaTeX$ is as follows:

**Theorem 1** Let 2 be a common divisor of $x$ and $y$ if $x$ is even and $y$ is even for all $y \in \mathbb{Z}$, for all $x \in \mathbb{Z}$. Let $x$ be even if and only if $x^2$ is even for all $x \in \mathbb{Z}$. Let there be a $y \in \mathbb{Z}$ such that there exists a $z \in \mathbb{Z}$ such that $x \cdot y = z$ and there is no $d \in \mathbb{Z}$ such that $d$ is a common divisor of $y$ and $z$ for all $x \in \mathbb{Q}$. Then $\sqrt{2}$ isn’t rational.

**Proof:**

Let 2 be a common divisor of $x$ and $y$ if $x$ is even and $y$ is even for all $y \in \mathbb{Z}$ for all $x \in \mathbb{Z}$. Let $x$ be even if and only if $x^2$ is even for all $x \in \mathbb{Z}$. Let there be a $y \in \mathbb{Z}$ such that there is a $z \in \mathbb{Z}$ such that $x \cdot y = z$ and there is no $d \in \mathbb{Z}$ such that $d$ is a common divisor of $y$ and $z$ for all $x \in \mathbb{Q}$. 

28
Figure 3: Presentation of the expanded proof in the graphical user interface \( \Omega UI \).

We prove that \( \sqrt{2} \) isn’t rational by a contradiction. Let \( \sqrt{2} \) be rational.

Let \( n \in \mathbb{Z} \) and let there be a \( d_{2350} \in \mathbb{Z} \) such that \( \sqrt{2} \cdot n = d_{2350} \) and there doesn’t exist a \( d_{2370} \in \mathbb{Z} \) such that \( d_{2370} \) is a common divisor of \( n \) and \( d_{2350} \).

Let \( m \in \mathbb{Z} \), let \( \sqrt{2} \cdot n = m \) and let there be no \( d_{2370} \in \mathbb{Z} \) such that \( d_{2370} \) is a common divisor of \( n \) and \( m \).

We prove that \( m^2 = 2 \cdot n^2 \) in order to prove that there exists a \( d_{2387} \in \mathbb{Z} \) such that \( m^2 = 2 \cdot d_{2387} \).

Therefore \( m^2 \) is even. That implies that \( m \) is even because \( m \in \mathbb{Z} \). That leads to the existence of a \( d_{2343} \in \mathbb{Z} \) such that \( m = 2 \cdot d_{2343} \).

Let \( k \in \mathbb{Z} \) and let \( m = 2 \cdot k \). \( k \in \mathbb{Z} \).

We prove that \( n^2 = 2 \cdot k^2 \) in order to prove that there is a \( d_{2333} \in \mathbb{Z} \) such that \( n^2 = 2 \cdot d_{2333} \).

That implies that \( n^2 \) is even. That implies that \( n \) is even since \( n \in \mathbb{Z} \). Therefore we have a contradiction since \( m \in \mathbb{Z} \), \( n \in \mathbb{Z} \), there doesn’t exist a \( d_{2370} \in \mathbb{Z} \) such that \( d_{2370} \) is a common divisor of \( n \) and \( m \), \( m \) is even and \( 2 \in \mathbb{Z} \).
Show me the current proof.

Theorem:
Let $2$ be a common divisor of $x$ and $y$ if $x$ is even and $y$ is even for all $y$ in $2$ for all $x$ in $2$. Let $x$ be even if and only if $x/2$ is even for all $x$ in $2$. Let there be a $y$ in $2$ such that there exists a $z$ in $2$ such that $x/y = z$ and there is no $d$ in $2$ such that $d$ is a common divisor of $y$ and $2$ for all $x$ in $2$. Then $\sqrt{2}$ isn't rational.

Proof:
Let $2$ be a common divisor of $x$ and $y$ if $x$ is even and $y$ is even for all $y$ in $2$ for all $x$ in $2$. Let $x$ be even if and only if $x/2$ is even for all $x$ in $2$. Let there be a $y$ in $2$ such that there exists a $z$ in $2$ such that $x/y = z$ and there is no $d$ in $2$ such that $d$ is a common divisor of $y$ and $2$ for all $x$ in $2$. We prove that $\sqrt{2}$ isn't rational by contradiction. Let $\sqrt{2}$ be rational.

Let $n$ in $2$ and let there be a $dc_{269}$ in $2$ such that $\sqrt{2}/n = dc_{269}$ and there doesn't exist a $dc_{273}$ in $2$ such that $dc_{273}$ is a common divisor of $n$ and $dc_{269}$.

Let $m$ in $2$, let $\sqrt{2}/m = m$ and let there be no $dc_{279}$ in $2$ such that $dc_{279}$ is a common divisor of $n$ and $m$.

We prove that $m^2 = 2m^2$ in order to prove that there exists a $dc_{283}$ in $2$ such that $m^2 = 2^*dc_{287}$. $m^2 = 2^*m^2$ since $\sqrt{2}/m = m$.

Therefore $m^2$ is even. That implies that $m$ is even because $m$ in $2$. That leads to the existence of a $dc_{343}$ in $2$ such that $m = 2^*dc_{343}$.

Let $k$ in $2$ and let $m = 2^k$. $k$ in $2$.

We prove that $m^2 = 2^km^2$ in order to prove that there is a $dc_{383}$ in $2$ such that $m^2 = 2^*dc_{383}$. $m^2 = 2^km^2$ since $m^2 = 2^km^2$ and $m = 2^k$.

That implies that $n^2$ is even. That implies that $n$ is even since $n$ in $2$. Therefore we have a contradiction since $m$ in $2$, $n$ in $2$, there doesn't exist a $dc_{279}$ in $2$ such that $dc_{279}$ is a common divisor of $n$ and $m$, $m$ is even and $2$ in $2$.

QEO

Figure 4: P.rex presentation of the proof in emacs.
Theorem: Let even $x$ and even $y$ imply that 2 is a common divisor of $x$ and $y$, for all $x, y \in \mathbb{Z}$. Let $x$ be even if and only if $x^2$ is even for all $x \in \mathbb{Z}$. Let there be a $y \in \mathbb{Z}$ such that there is a $z \in \mathbb{Z}$ such that $x^2 = z$ and there doesn't exist a $d \in \mathbb{Z}$ such that $d$ is a common divisor of $y$ and $z$ for all $x \in \mathbb{Q}$. Thus $\sqrt{2}$ isn't rational.

Proof:

Let $z$ be a common divisor of $x$ and $y$ if $x$ is even and $y$ is even for all $y \in \mathbb{Z}$ for all $x \in \mathbb{Z}$. Let $x$ be even if and only if $x^2$ is even for all $x \in \mathbb{Z}$. Let there be a $y \in \mathbb{Z}$ such that there exists a $z \in \mathbb{Z}$ such that $x^2 = z$ and there is no $d \in \mathbb{Z}$ such that $d$ is a common divisor of $y$ and $z$ for all $x \in \mathbb{Q}$. We prove that $\sqrt{2}$ isn't rational by a contradiction. Let $\sqrt{2}$ be rational.

Let $n \in \mathbb{Z}$ and let there be a $d_2, 269$ in $\mathbb{Z}$ such that $\sqrt{2} \times n = d_2, 269$ and there is no $d_2, 279$ in $\mathbb{Z}$ such that $d_2, 279$ is a common divisor of $n$ and $d_2, 269$.

Let $m \in \mathbb{Z}$, let $\sqrt{2} \times m = m$ and let there be no $d_2, 279$ in $\mathbb{Z}$ such that $d_2, 279$ is a common divisor of $n$ and $m$.

We prove that $m^2 = 2 \times n^2$ in order to prove that there exists a $d_2, 287$ in $\mathbb{Z}$ such that $m^2 = 2 \times d_2, 287$. $m^2 = 2 \times n^2$ because $\sqrt{2} \times m = m$.

Hence $m^2$ is even. Hence $m$ is even since $m \in \mathbb{Z}$. Then there exists a $d_2, 343$ in $\mathbb{Z}$ such that $m^2 = 2 \times d_2, 343$.

Let $x \in \mathbb{Z}$ and let $m = 2 \times k \in \mathbb{Z}$.

We prove that $n^2 = 2 \times m^2$ in order to prove that there is a $d_2, 353$ in $\mathbb{Z}$ such that $n^2 = 2 \times d_2, 353$. $n^2 = 2 \times m^2$ since $m^2 = 2 \times n^2$ and $m = 2 \times k$.

That implies that $n^2$ is even. That leads to even $n$ because $n \in \mathbb{Z}$.

That leads to a contradiction because $n \in \mathbb{Z}$, $n \in \mathbb{Z}$, there is no $d_2, 279$ in $\mathbb{Z}$ such that $d_2, 279$ is a common divisor of $n$ and $m$, $m$ is even and $2 \times m \in \mathbb{Z}$.

Figure 5: Prew presentation of the proof in LOGIC.

31
8 External Reasoning Systems

Proof problems require in general many different skills to find the solutions. Therefore, it is desirable to have access to several systems with complementary capabilities, to orchestrate their use, and to integrate their results. \OMEGA 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 either provide a solution to a subproblem or 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 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 \OMEGA, which is then refined to a logic-level proof to be examined by \OMEGA's proof checker.

The integrated external systems in \OMEGA are currently the following:

**CASs** provide symbolic computation, which can be used in two ways: firstly, to compute hints to guide the proof search (e.g., witnesses for free (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 \OMEGA. CASs are integrated via the transformation and translation module SAPPER [47]. Currently, \OMEGA uses the computer algebra systems MAPLE and GAP.

**ATPs** are employed to solve subgoals. Currently \OMEGA uses the first-order automated theorem proving systems BLIKSEM, EQP, OTTER, ProTeIn, SPASS, WaldM
test, and the higher-order systems TPS [2, 3], and LEO [12, 6]. The first-order ATPs are connected via Tramp [33], a proof transformation system that transforms resolution-style proofs into assertion level ND proofs to be integrated into \OMEGA's PDS. TPS already provides ND proofs, which can be further processed and checked with little transformational effort [7].

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

**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, \OMEGA employs CoSIE [39], a constraint solver for inequalities and equations over the field of real numbers.

In the interactive proof given in Section 6 we employed tactics that make calls to external reasoning systems. Concretely, the CAS MAPLE is employed for simple computations (see Steps 10 and 22) and the ATP OTTER is employed for simple logical derivations (see Steps 17, 28, and 32). Instead of calling them directly from \OMEGA, respective service requests are send to the mathematical software bus MathWeb-SB [25], which then passes the requests to available systems.

A peculiarity of the \OMEGA environment is that resolution proofs generated by ATPs like OTTER can be transformed via another mathematical service called Tramp [33] into natural-deduction-style proofs in \OMEGA. There we can then check the translated proof by an independent proof checker after expanding it to base calculus level.

9 A Related Case Study

In addition to the proof presented in Section 6, we encoded the problem also in the same way as in the Otter case study in [53].
(problem otter-case-study
  (in base)
  (constants (one i)
    (two i)
    (mult (i i i))
    (divides (i i i)))

(assumption TWIN-NOTONE ;; two is unequal one
  (not (= two one)))

(assumption ONE-IDENTITY-LEFT-MULT ;; ONE is left-unit element of multiplication
  (forall (lam (x i)) (= (mult one x) x)))

(assumption ONE-IDENTITY-RIGHT-MULT ;; ONE is right-unit element of multiplication
  (forall (lam (x i)) (= (mult x one) x)))

(assumption ASSOCIATIVITY-MULT ;; Multiplication is associative
  (forall (lam (x i)
    (forall (lam (y i)
      (forall (lam (z i)
        (= (mult (mult (mult y z)) (mult (mult x y) z))) ))))))

(assumption COMMUTATIVITY-M
 ;; Multiplication is commutative
 (forall (lam (x i)
   (forall (lam (y i)
     (forall (lam (z i)
       (implies (= (mult x y) (mult x z)) (= y z))))))))

(assumption CANCELLATION ;; Cancellation in multiplication
  (forall (lam (x i)
    (forall (lam (y i)
      (forall (lam (z i)
        (implies (= (mult x y) (mult x z)) (= y z))))))))

(assumption DIVIDES ;; Definition of divides
  (forall (lam (x i)
    (forall (lam (y i)
      (equiv (divides x y)
        (exists (lam (z i)) (= (mult x z) y))))))

(assumption TWIN-PRIME ;; Two is prime-number
  (forall (lam (x i)
    (forall (lam (y i)
      (implies (divides two (mult x y))
        (or (divides two x) (divides two y))))))

(conclusion T)
  (not (exists (lam (a i))
    (exists (lam (b i))
      (and (= (mult a a) (mult two (mult b b))))
      (forall (lam (x i))
        (implies (and (divides a x)
          (divides b x))
          (= x one))))))
)

After loading this problem in Omega, we employed our connection to MathWeb-SB to pass
the problem from Omega to Otter. Tramp [33] translates the Otter proof into a proof in
Omega. The benefit of calling Otter or other traditional ATPs via Omega instead of working
with them directly are:

- The results of different ATPs are translated into a uniform representation in the Omega
  system. Hence the user only has to understand Omega proof presentations.
- The proof can be expanded and independently checked in Omega.
- Pex can be applied to generate a natural-language presentation of the proof.

We present a part of the interactive session for this case study.

33
INTEGRAL-FORMULAS (BOOLEAN) Integral formulas: [][]
MAXIMAL-DEPTH (INTEGER) Maximal depth of searching integral-formulas: [2]
TNN (BOOLEAN) Prefer tertium non datur case analyzers: [?] AVOID-DOUBLING (BOOLEAN) Avoid doubling: [?] LEMMAS (SYMBOL) Lemmas over (nil/free/constant/full): [CONSTANTS]

;;; Rules loaded for theory BASE.
;;; Theorems loaded for theory BASE.
;;; Tactics loaded for theory BASE.
;;; Methods loaded for theory BASE.
;;; Control-rules loaded for theory BASE.
;;; Meta-predicates loaded for theory BASE.
;;; Strategies loaded for theory BASE.
;;; Agents loaded for theory BASE.

Normalizing ...
No File found for USER-FLAGS, USER-FLAGS interpreted as direct user input.
Calling otter process 31585 with time resource 10 sec.

otter Time Resource in seconds:
10sec

------- PROOF -------
Search stopped by max_proofs option.
Parsing otter Proof ...
OTTER HAS FOUND A PROOF
OMEGA>CURRENT-RESOLUTION-PROOF IS SET TO THE FOUND RESOLUTION PROOF
Searching for lemmata ...
Creating Refutation-Graph ...
Creating Refutation-Graph ...
Translating ...
PREPARING DECOMPOSE UNIT FOR SPLITTING ...
PREPARING DECOMPOSE UNIT FOR SPLITTING ...
PREPARING DECOMPOSE UNIT FOR SPLITTING ...
PREPARING DECOMPOSE UNIT FOR SPLITTING ...
Translation finished!

OMEGA: show-pds

L2 (L2) ! (EXISTS [A:1:B:1]) HYP
      (AND (= (MULT A A) (MULT TWO (MULT B B))))
      (FORALL [X:I] (IMPLIES (AND (DIVIDES X A))
                       (DIVIDES X B))
               (= X ONE)))

L5 (L5) ! (EXISTS [B:I]) HYP
      (AND (= (MULT (MULT C1 C1) (MULT TWO (MULT C2 C2))))
           (FORALL [X:I] (IMPLIES (AND (DIVIDES X C1) (DIVIDES X C2))
                               (= X ONE))))

L9 (L9) ! (AND (= (MULT C1 C1) (MULT TWO (MULT C2 C2)))) HYP
      (FORALL [X:I] (IMPLIES (AND (DIVIDES X C1) (DIVIDES X C2))
                             (= X ONE))))

L11 (L9) ! (= (MULT C1 C1) (MULT TWO (MULT C2 C2))) ANDE: (L9)

L58 (L9) ! (= (MULT TWO (MULT C2 C2)) (MULT C1 C1)) =SYM: (L11)

L12 (L9) ! (FORALL [X:I] (IMPLIES (AND (DIVIDES X C1) (DIVIDES X C2))
                             (= X ONE))) ANDE: (L9)

L24 (L24) ! (AND (= (MULT C1 C1) (MULT TWO (MULT C3 C3)))) HYP
      (FORALL [X:I] (IMPLIES (AND (DIVIDES X C1) (DIVIDES X C3))
                             (= X ONE))))
L27  (L24) ! (- (MULT C1 C1) (MULT TWO (MULT C3 C3))) AND: (L24)

L28  (L24) ! (FürALL [X:1] 
(IMP:IL (AND (DIVIDES X C1) (DIVIDES X C3)) 
(- X ONE))) AND: (L24)

L31  (L31) ! (- (MULT TWO C4) C1) HYP

L43  (L43) ! (- (MULT TWO C6) (MULT C4 C1)) HYP

TWONOTONE (TWONOTONE) ! (NOT (- TWO ONE)) HYP

ONE-IDENTITY-LEFT-MULT (ONE-IDENTITY-LEFT-MULT) ! (FürALL [X:1] (- (MULT ONE X) X)) HYP

ONE-IDENTITY-RIGHT-MULT (ONE-IDENTITY-RIGHT-MULT) ! (FürALL [X:1] (- (MULT X ONE) X)) HYP

ASSOCIATIVITY-MULT (ASSOCIATIVITY-MULT) ! (FürALL [X:Y:Z:I] 
(- (MULT X (MULT Y Z)) (MULT (MULT X Y) Z))) HYP

L18  (ASSOCIATIVITY-MULT) ! (FürALL [Y:Z:I]) 
(IMP:LL (ASSOCIATIVITY-MULT) 
(- (MULT TWO (MULT Y Z)) 
(MULT (MULT TWO Y) Z)))

L34  (ASSOCIATIVITY-MULT) ! (FürALL [Z:I]) 
(FURALLE: (C4) (L18) 
(- (MULT TWO (MULT C4 Z)) 
(MULT (MULT TWO C4 Z) Z)))

L36  (ASSOCIATIVITY-MULT) ! (- (MULT TWO (MULT C4 C1)) 
(IMP:LL (C1) (L34) 
(MULT (MULT TWO C4 C1))))

L45  (ASSOCIATIVITY-MULT) ! (- (MULT TWO (MULT C4 C1)) 
(-SUBST: ((2 1)) (L26 L31) L31) 
(MULT C1 C1))

L46  (ASSOCIATIVITY-MULT) ! (- (MULT C1 C1) 
(-SYM: (L45) 
L31) 
(MULT TWO (MULT C4 C1)))

L47  (L31) ! (- (MULT TWO (MULT C3 C3)) 
(IMP:LL (ASSOCIATIVITY-MULT) 
(MULT TWO (MULT C4 C1))) 
(L24))

L56  (ASSOCIATIVITY-MULT) ! (- (MULT TWO (MULT C4 C4)) 
(FURALLE: (C4) (L34) 
(MULT (MULT TWO C4 C4))

L55  (ASSOCIATIVITY-MULT) ! (- (MULT TWO (MULT C4 C4)) 
(-SUBST: ((2 1)) (L26 L31) L31) 
(MULT C1 C4))

COMPUTATIVITY-M (COMPUTATIVITY-M) ! (FürALL [X:Y:I] 
(- (MULT X Y) (MULT Y X))) HYP
(COMMUTATIVITY-M) ! (FORALL [y:1] 
FORALLE: (C1) (COMMUTATIVITY-M) 
(= (MULT C1 Y) (MULT Y C1)))

(L91) 

(COMMUTATIVITY-M) ! (= (MULT C1 C0) 
FORALLE: (C0) (L91) 
(MULT C0 C1))

(CANCELLATION) ! (FORALL [x:i,y:i,z:i] 
HYP 
(IMPLIES (= (MULT X Y) (MULT X Z)) 
(= Y Z)))

(L48) 

(CANCELLATION ! (= (MULT C3 C3) 
ASSERTION: (CANCELLATION L47) 
L31 
(MULT C0 C1))

(L49) 

(CANCELLATION ! (= (MULT C0 C1) (MULT C3 C3)) 
-SYM: (L48) 
L31 
ASSOCIATIVITY-MULT 
L24)

(L53) 

(L24 ! (= (MULT TWO C5) (MULT C3 C3)) 
-SUBST: ((1)) (L49 L33) 
L31 
ASSOCIATIVITY-MULT 
L24)

(DIVIDES) ! (FORALL [x:i,y:i] 
HYP 
(EQUIV (DIVIDES X Y) 
(EXISTS [z:i] (= (MULT X Z) Y])))

(L59) 

(DIVIDES ! (DIVIDES TWO (MULT C1 C1)) 
ASSERTION: (DIVIDES L59) 
L9)

(L60) 

(DIVIDES ! (FORALL [y:1] 
FORALLE: (TWO) (DIVIDES) 
(EQUIV (DIVIDES TWO Y) 
(EXISTS [z:i] (= (MULT TWO Z) Y))))

(L61) 

(DIVIDES ! (EQUIV (DIVIDES TWO C1) 
FORALLE: (C1) (L60) 
(EXISTS [z:i] (= (MULT TWO Z) C1)))

(L62) 

(DIVIDES ! (IMPLIED (DIVIDES TWO C1) 
(EQUIV [z:i] (= (MULT TWO Z) C1)))

(L63) 

(DIVIDES ! (IMPLIED (EXISTS [z:i] (= (MULT TWO Z) C1)) 
(EQUIV: (L62))

(DIVIDES TWO C1))

(L66) 

(DIVIDES ! (EQUIV (DIVIDES TWO (MULT C3 C3)) 
FORALLE: ((MULT C3 C3)) (L60) 
(EXISTS [z:i] (= (MULT TWO Z) (MULT C3 C3))))

(L68) 

(DIVIDES ! (EQUIV (DIVIDES TWO (MULT C0 C1)) 
FORALLE: ((MULT C0 C1)) (L60) 
(EXISTS [z:i] (= (MULT TWO Z) (MULT C0 C1))))
(DIVIDES)  ! (IMPLIES (DIVIDES TWO (MULT C1 C1))
   (EXISTS [Z:1] (= (MULT TWO Z) (MULT C1 C1))))
   EQUIVE: (L37)

(DIVIDES)  ! (IMPLIES (EXISTS [Z:1] (= (MULT TWO Z)
   (MULT C1 C1)))
   (DIVIDES TWO (MULT C1 C1)))
   EQUIVE: (L37)

(DIVIDES)  ! (EQUIV (DIVIDES TWO (MULT C1 C1))
   (EXISTS [Z:1] (= (MULT TWO Z) (MULT C1 C1))))
   EQUIVE: (L28)
   EQUIVE: (L37)

(DIVIDES)  ! (DIVIDES TWO (MULT C1 C1))
   ASSETION: (L38 L66)
   ASOCIATIVITY-MULT
   L31)

(L36:  ! (DIVIDES TWO (MULT C1 C1))
   ASSETION: (L38 L66)
   DIVIDES
   ASOCIATIVITY-MULT
   COMUTATIVITY-M)

(TWOPRIME)  ! (FORALL [X:1,Y:1]
   (IMPLIES (DIVIDES TWO (MULT X Y))
   (OR (DIVIDES TWO X) (DIVIDES TWO Y))))
   HYP

(L30:  ! (EXISTS [Z:1] (= (MULT TWO Z) C1))
   IMPE: (L1 L22)
   L2
   TWOPRIME
   DIVIDES
   CANCELLATION
   COMUTATIVITY-M
   ASOCIATIVITY-MULT
   ONE-IDENTITY-RIGHT-MULT
   ONE-IDENTITY-LEFT-MULT
   TWOPRIME)

(L31 L24:  ! (EXISTS [Z:1] (= (MULT TWO Z) (MULT C1 C1)))
   IMPE: (L57 L39)
   L5 L2
   TWOPRIME
   DIVIDES
   CANCELLATION
   COMUTATIVITY-M
   ASOCIATIVITY-MULT
   ONE-IDENTITY-RIGHT-MULT
   ONE-IDENTITY-LEFT-MULT
   TWOPRIME)

(L31 L24:  ! FALSE
   EXISTSE: (C5) (L42 L54)
   L5 L2
   TWOPRIME
   DIVIDES
   CANCELLATION
   COMUTATIVITY-M
   ASOCIATIVITY-MULT
   ONE-IDENTITY-RIGHT-MULT
   ONE-IDENTITY-LEFT-MULT
   TWOPRIME)

(L24 L5 L2:  ! FALSE
   EXISTSE: (C4) (L30 L32)
   TWOPRIME
L8
(L5 L2  ! FALSE
TWOPRIME
DIVIDES
CANCELLATION
COMMUTATIVITY-M
ASSOCIATIVITY-MULT
ONE-IDENTITY-RIGHT-MULT
ONE-IDENTITY-LEFT-MULT
TWOUNITONE)

EXISTSE: (C3) (L5 L25)

L9
(L2  ! FALSE
TWOPRIME
DIVIDES
CANCELLATION
COMMUTATIVITY-M
ASSOCIATIVITY-MULT
ONE-IDENTITY-RIGHT-MULT
ONE-IDENTITY-LEFT-MULT
TWOUNITONE)

EXISTSE: (C1) (L2 L8)

L13
(TWOPRIME) ! (FORALL [Y: I]
(FORALL: (C1) (TWOPRIME)
(IMPRES (DIVIDES TWO (MULT C1 Y))
(OR (DIVIDES TWO C1) (DIVIDES TWO Y))))

L14
(TWOPRIME) ! (IMPRES (DIVIDES TWO (MULT C1 C1))
FORALL: (C1) (L13)
(OR (DIVIDES TWO C1) (DIVIDES TWO C1)))

L16
(L9 L5 L2  ! (OR (DIVIDES TWO C1) (DIVIDES TWO C1))
DIVIDES
TWOPRIME)

IMPE: (L59 L14)

L17
(L9 L5 L2  ! (DIVIDES TWO C1)
DIVIDES
TWOPRIME)

IDEMOR: (L16)

L1
(L5 L2  ! (DIVIDES TWO C1)
DIVIDES
TWOPRIME)

EXISTSE: (C2) (L5 L17)

L50
(L24 L5 L2  ! (NOT (DIVIDES TWO C3))
ASSERTION: (L26 L1 TWOUNITONE)
DIVIDES
TWOPRIME
TWONEUNONE)

L51
(L24 L5 L2  ! (NOT (DIVIDES TWO (MULT C3 C3)))
ASSERTION: (TWOPRIME L50 L50)
DIVIDES
TWOPRIME
TWONEUNONE)
10 Discussion

In [53], it was suggested to discuss different proof assistants according to the following criteria, where + and − indicate whether the criterion is fulfilled by ΩMEGA:

- Constructive logic supported: −
- Small proof kernel (proof objects): +
- Calculations can be proved automatically: +
- Extensible/programmable by the user: +
- Powerful automation: +
- Readable proof input files: arguable
- Based on higher order logic: +
- Based on ZFC set theory: −
- Statement about R: +
- Statement about √: +

While this is a good start for a comparison of proof assistants, there may be further criteria that should be considered as well. To our view, further criteria should be, for instance, the quality of the user interface and the availability of different proof presentations tools. In this report, we tried to illustrate some of the tools that are provided in ΩMEGA in this respect. However, it seems to be quite impossible to provide a representative flavor of user interaction in ΩMEGA on paper format. Features such as the hypertext mechanism in LATEX, which maintains the mappings between the nodes in the tree presentation of the proof and the proof lines in the linearized proof presentation, cannot be sufficiently illustrated on paper.

The most important lesson to be learned from this case study is to show the wrong level of abstraction still common in most automated and tactical theorem proving environments. While this is already an abstraction from the calculus level (called the assertion level in [29]), it is nevertheless clear that as long as a system does not hide all these 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 ATP systems and one of the reasons why they are not used as widely as, say, computer algebra systems.
So this observation is the crucial issue in the $\Omega_\text{mega}$ project and our motivation for departing from the classical paradigm of ATP about fifteen years ago. A main aim of the project is to further improve and optimally integrate the illustrated features of the system. It thereby should become possible to reduce the interaction steps required in our case study to a few steps only. These steps then ideally address the main mathematical arguments that would also appear in a textbook proof.

References


A \TeX\ Presentations of the Proof

A.1 The Unexpanded Proof

\begin{itemize}
  \item L.19. \quad l_{10} \vdash [Z_{\psi \rightarrow \theta}(K_{\psi}) \land M_{\psi}] = (2 \land [\psi \rightarrow \theta]) K_{\psi}\quad \text{(Hyp)}
  \item L.22. \quad w_0 \vdash M = (2 \land K)\quad \text{(And L.19)}
  \item L.21. \quad w_0 \vdash Z(K)\quad \text{(And L.19)}
  \item L.8. \quad l_8 \vdash [Z(M) \land \{\sqrt{\phi - \theta}[\phi \rightarrow \theta] N_{\psi}] = M \land -\exists X_{\psi}] : (Hyp)
  \item L.11. \quad w_1 \vdash -\exists X{\psi} : Z_{\psi} \text{Common-Divisor}(N, M, X_{\psi})\quad \text{(And L.18)}
  \item L.11. \quad w_1 \vdash (\sqrt{\phi - \theta} N) = M\quad \text{(And L.18)}
  \item L.10. \quad w_1 \vdash Z(M)\quad \text{(And L.18)}
  \item L.4. \quad l_4 \vdash [Z(N) \land \exists X_{\psi}] : Z_{\psi}[(\sqrt{\phi - \theta} N) = X_{\psi} \land -\exists X_{\psi}] : (Hyp)
  \item L.7. \quad w_2 \vdash + \exists X_{\psi} : Z_{\psi}[(\sqrt{\phi - \theta} N) = X_{\psi} \land -\exists X_{\psi}] : Z_{\psi} \text{Common-Divisor}(N, X_{\psi}, X_{\psi})\quad \text{(And E L.4)}
  \item L.6. \quad w_2 \vdash Z(N)\quad \text{(And E L.4)}
  \item L.1. \quad l_1 \vdash Q_{\psi \rightarrow \theta}(\sqrt{\phi - \theta})\quad \text{(Hyp)}
  \item L.2. \quad r_c \vdash \exists X_{\psi} : Q_{\psi \rightarrow \theta}(\sqrt{\phi - \theta} N) = X_{\psi} \land -\exists X_{\psi} : Z_{\psi} \text{Common-Divisor}(X_{\psi}, X_{\psi}, X_{\psi})\quad \text{(Thm)}
  \item L.9. \quad w_4 \vdash \bot\quad \text{(Existential L1, L20)}
  \item L.5. \quad w_5 \vdash \bot\quad \text{(Existential L1, L20)}
  \item L.3. \quad l_3 \vdash + \exists X_{\psi} : Z_{\psi} X_{\psi} : Z_{\psi}[(\sqrt{\phi - \theta} N) = X_{\psi} \land -\exists X_{\psi}] : Z_{\psi} \text{Common-Divisor}(X_{\psi}, X_{\psi}, X_{\psi})\quad \text{(Fornala-Sort RC L.11)}
  \item L.13. \quad w_4 \vdash (M_{\psi \rightarrow \theta}(\sqrt{\phi - \theta})) = (2 \land \{\psi \rightarrow \theta\})\quad \text{(By-Computation L.11)}
  \item L.15. \quad w_4 \vdash + \exists X\psi : Z_{\psi}(M_{\psi \rightarrow \theta}) = (2 \land \{\psi \rightarrow \theta\})\quad \text{(Existential L.13, L.16)}
  \item SE. \quad s_r \vdash + \exists X_{\psi} : Z_{\psi}[\text{Even}(\{\psi \rightarrow \theta\})] \Rightarrow \text{Even}(X_{\psi})\quad \text{(Thm)}
  \item L.20. \quad w_6 \vdash \bot\quad \text{(Thm)}
  \item L.17. \quad w_4 \vdash + \exists X : Z_{\psi} M\quad \text{(Defini L.15)}
  \item L.18. \quad w_4 \vdash + \exists X_{\psi} : Z_{\psi} M = (2 \land \{\psi \rightarrow \theta\})\quad \text{(Wellsorted L.21)}
  \item L.23. \quad w_6 \vdash (N_{\psi}) = (2 \land \{\psi \rightarrow \theta\})\quad \text{(By-Computation L.13, L.22)}
  \item L.25. \quad w_6 \vdash + \exists X_{\psi} : Z_{\psi}(N) = (2 \land \{\psi \rightarrow \theta\})\quad \text{(Existential L.23, L.26)}
  \item L.24. \quad w_6 \vdash + \exists X_{\psi} : Z_{\psi}(N)\quad \text{(Defini L.25)}
  \item L.27. \quad w_6 \vdash + \exists X_{\psi} : Z_{\psi}(N)\quad \text{(Wellsorted L.21)}
  \item ECD. \quad r_c \vdash + \exists X_{\psi} : Z_{\psi}(N)\quad \text{(Thm)}
  \item L.28. \quad w_6 \vdash Z(2)\quad \text{(Wellsorted)}
  \item L.29. \quad w_6 \vdash \bot\quad \text{(Wellsorted)}
  \item S2NDR. \quad w_7 \vdash -Q(\sqrt{\phi - \theta})\quad \text{(~ I L.2)\quad}
\end{itemize}

A.2 The Expanded Proof

\begin{itemize}
  \item L.185. \quad l_{50} \vdash + \exists X_{\gamma} : Z_{\psi \rightarrow \theta}(X_{\gamma}) \Rightarrow \text{Even}(\{\psi \rightarrow \theta\}) \Rightarrow \text{Even}(X_{\gamma})\quad \text{(Hyp)}
  \item ECD. \quad r_c \vdash + \exists X_{\psi} : Z_{\psi}[\text{Even}(X) \land \text{Even}(\psi)] \Rightarrow \text{Even}(X_{\psi})\quad \text{(Thm)}
  \item L.87. \quad r_c \vdash + \exists X_{\psi} : \phi(\psi) = (2 \land \{\psi \rightarrow \theta\})\quad \text{(Define ECD)}
  \item L.71. \quad w_1 \vdash + \exists X_{\psi} : Z_{\psi}(X_{\psi}) \Rightarrow \text{Even}(X_{\psi})\Rightarrow (2 \land \{\psi \rightarrow \theta\})\quad \text{(Define L.71)}
  \item L.64. \quad l_{35} \vdash + \exists X_{\psi} : Z_{\psi}(X_{\psi}) \Rightarrow \text{Even}(X_{\psi})\Rightarrow (2 \land \{\psi \rightarrow \theta\})\quad \text{(Lambda L.71)}
\end{itemize}
L121. $\frac{121}{W} \frac{1}{1} \frac{121}{\therefore} \frac{121}{Z(N) \Rightarrow \forall X_{301} \mid Z(X_{301}) \Rightarrow (\text{Event}(N) \land \text{Event}(X_{301}) \Rightarrow \text{Common-Divisor}(N, X_{301}, 2))]} {\forall E \ 164}$

L187. $\frac{121}{\frac{121}{W} \frac{121}{Z(N)}} {\text{(Hyp)}}$

L189. $\frac{121}{\frac{121}{W} \frac{121}{\forall X_{301} \mid Z(X_{301}) \Rightarrow (\text{Event}(N) \land \text{Event}(X_{301}) \Rightarrow \text{Common-Divisor}(N, X_{301}, 2))]} {\Rightarrow E \ 187, \ 121}$

L188. $\frac{121}{\frac{121}{W} \frac{121}{\perp}} {\text{(-}E \ 188)}$

L186. $\frac{121}{\frac{121}{W} \frac{121}{\neg Z(N)}} {\text{(-}I \ 188)}$

L160. $\frac{121}{\frac{121}{W} \frac{121}{\forall X_{301} \mid Z(X_{301}) \Rightarrow (\text{Event}(N) \land \text{Event}(X_{301}) \Rightarrow \text{Common-Divisor}(N, X_{301}, 2))]} {\forall E \ 156}$

L159. $\frac{121}{\frac{121}{W} \frac{121}{\perp}} {\text{(Hyp)}}$

L141. $\frac{121}{\frac{121}{W} \frac{121}{\neg \text{Common-Divisor}(N, M, 2)}} {\text{(Weak H 141)}}$

L144. $\frac{121}{\frac{121}{W} \frac{121}{\neg \text{Common-Divisor}(N, M, 2)}} {\text{(Hyp)}}$

L139. $\frac{121}{\frac{121}{W} \frac{121}{\neg Z(2)}} {\text{(Axiom)}}$

L124. $\frac{121}{\frac{121}{W} \frac{121}{\perp}} {\text{(Hyp)}}$

L119. $\frac{121}{\frac{121}{W} \frac{121}{\neg \text{Common-Divisor}(N, M, 2)}} {\text{(Hyp)}}$

L8. $\frac{121}{\frac{121}{W} \frac{121}{\neg Z(M) \land (\sqrt[21]{X_{301}} \Rightarrow Z(M)) \Rightarrow \neg \exists X_{301} \mid Z(M, X_{301})]} {\text{(Fals H 145)}}$

L30. $\frac{121}{\frac{121}{W} \frac{121}{\forall X_{301} \mid Z(M) \Rightarrow \neg \exists X_{301} \mid Z(M, X_{301})]} {\text{(Hyp)}}$

L121. $\frac{121}{\frac{121}{W} \frac{121}{\perp}} {\text{(Hyp)}}$

L143. $\frac{121}{\frac{121}{W} \frac{121}{\neg \text{Common-Divisor}(N, M, 2)}} {\text{(Hyp)}}$

L142. $\frac{121}{\frac{121}{W} \frac{121}{\perp}} {\text{(Hyp)}}$

L145. $\frac{121}{\frac{121}{W} \frac{121}{\perp}} {\text{(Hyp)}}$

L134. $\frac{121}{\frac{121}{W} \frac{121}{\neg Z(2) \land \neg \text{Common-Divisor}(N, M, 2)}} {\text{(Hyp)}}$

L127. $\frac{121}{\frac{121}{W} \frac{121}{\neg Z(2) \land \neg \text{Common-Divisor}(N, M, 2)}} {\text{(Hyp)}}$

L125. $\frac{121}{\frac{121}{W} \frac{121}{\neg Z(2) \land \neg \text{Common-Divisor}(N, M, 2)}} {\text{(Hyp)}}$

L130. $\frac{121}{\frac{121}{W} \frac{121}{\neg Z(2) \land \neg \text{Common-Divisor}(N, M, 2)}} {\text{(Hyp)}}$

L132. $\frac{121}{\frac{121}{W} \frac{121}{\neg Z(2) \land \neg \text{Common-Divisor}(N, M, 2)}} {\text{(Hyp)}}$

L131. $\frac{121}{\frac{121}{W} \frac{121}{\neg Z(2) \land \neg \text{Common-Divisor}(N, M, 2)}} {\text{(Hyp)}}$

L136. $\frac{121}{\frac{121}{W} \frac{121}{\neg Z(2) \land \neg \text{Common-Divisor}(N, M, 2)}} {\text{(Hyp)}}$
\[ L_{174}. \ \vdash \text{Common-Divisor}(N, M, 2) \quad \text{(False: } L_{175}) \]
\[ L_{172}. \ \vdash \text{Common-Divisor}(N, M, 2) \quad \text{(Hyp)} \]
\[ L_{173}. \ \vdash \text{Common-Divisor}(N, M, 2) \quad \text{(Weaken: } L_{172}) \]
\[ \text{TND. TND} \vdash \forall x \in \mathbb{N} : [x \in \mathbb{N} \land x \div N] \quad \text{(Axiom)} \]
\[ L_{170}. \ \text{TND} \vdash \text{Common-Divisor}(N, M, 2) \lor \neg \text{Common-Divisor}(N, M, 2) \quad \text{(}\forall E \ \text{TND)} \]
\[ L_{173}. \ \text{TND. TND} \vdash \text{Common-Divisor}(N, M, 2) \quad \text{(}\forall E \text{ } L_{173}) \]
\[ L_{174}. \ \text{TND. TND} \vdash x \div N \quad \text{Hyp} \]
\[ L_{175}. \ \vdash \neg x \div N \quad \text{(}\forall E \text{ } L_{174}) \]
\[ L_{176}. \ \vdash \forall x \in \mathbb{N} : [x \div N \lor \neg x \div N] \quad \text{(}\forall E \text{ } L_{173}) \]
\[ L_{176}. \ \text{TND} \vdash \exists x \in \mathbb{N} : x \div N \quad \text{(}\forall E \text{ } L_{173}) \]
\[ L_{177}. \ \vdash \exists x \in \mathbb{N} : x \div N \quad \text{(}\forall E \text{ } L_{173}) \]
\[ L_{178}. \ \vdash \exists x \in \mathbb{N} : x \div N \quad \text{(}\forall E \text{ } L_{173}) \]
\[ L_{179}. \ \vdash \exists x \in \mathbb{N} : x \div N \quad \text{(}\forall E \text{ } L_{173}) \]
\[ L_{176}. \ \text{TND} \vdash \exists x \in \mathbb{N} : x \div N \quad \text{(}\forall E \text{ } L_{173}) \]
\[ L_{177}. \ \vdash \exists x \in \mathbb{N} : x \div N \quad \text{(}\forall E \text{ } L_{173}) \]
\[ L_{178}. \ \vdash \exists x \in \mathbb{N} : x \div N \quad \text{(}\forall E \text{ } L_{173}) \]
\[ L_{179}. \ \vdash \exists x \in \mathbb{N} : x \div N \quad \text{(}\forall E \text{ } L_{173}) \]
\[ \exists X_{\text{odd}}, \exists X_{\text{even}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(3F L43)

\[ \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Defn L42)

\[ \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} = \{ M \cdot 2 \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Defn L15)

\[ \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ M \cdot 2 \} \]

(Defn L14)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)

\[ \forall \exists X_{\text{odd}} \mid \{ X_{\text{odd}} \} \wedge \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Weakens L68)

\[ \forall \exists X_{\text{even}} \mid \{ X_{\text{even}} \} = \{ 2 \cdot X_{\text{odd}} \} \]

(Define SE)
\[ L_{34} \vdash \exists X_{99}[\forall N \exists X_{90} \wedge M = [2 \cdot X_{90}]] \] (Define L.18)

\[ L_{35} \vdash \bot \] (Weakens L.9)

\[ L_{36} \vdash \bot \] (Weakens L.5)

\[ L_{37} \vdash \exists X_{23}[\forall (X_{28}) \wedge [(\exists N) = X_{28} \wedge \exists X_{31}[\forall X]] \wedge \exists X_{23}[\forall X]] \] (Define L.7)

\[ L_{38} \vdash \bot \] (Weakens L.37, L.38)

\[ L_{39} \vdash \bot \] (Weakens L.5)

\[ L_{40} \vdash \forall X_{44}[\forall (X_{45}) \Rightarrow \exists X_{64}[\forall X_{64} = X_{67} \wedge \exists X_{71}[\forall X_{64} = X_{67} \wedge \exists X_{71}[\forall X]] \wedge Z [\exists X_{64}[\forall X_{64} = X_{67} \wedge \exists X_{71}[\forall X]] \] (Define RC)

\[ L_{41} \vdash \forall \sqrt{2} \Rightarrow \exists X_{64}[\forall X_{64} = X_{67} \wedge \exists X_{71}[\forall X_{64} = X_{67} \wedge \exists X_{71}[\forall X]] \wedge Z [\exists X_{64}[\forall X_{64} = X_{67} \wedge \exists X_{71}[\forall X]] \] (Def. L.40)

\[ L_{42} \vdash \bot \] (Hyp)

\[ L_{43} \vdash \forall \sqrt{2} \Rightarrow \exists X_{64}[\forall X_{64} = X_{67} \wedge \exists X_{71}[\forall X_{64} = X_{67} \wedge \exists X_{71}[\forall X]] \wedge Z [\exists X_{64}[\forall X_{64} = X_{67} \wedge \exists X_{71}[\forall X]] \] (\( \Rightarrow E \, L.11, L.41 \))

\[ L_{44} \vdash \forall \sqrt{2} \Rightarrow \exists X_{64}[\forall X_{64} = X_{67} \wedge \exists X_{71}[\forall X_{64} = X_{67} \wedge \exists X_{71}[\forall X]] \wedge Z [\exists X_{64}[\forall X_{64} = X_{67} \wedge \exists X_{71}[\forall X]] \] (Define L.3)

\[ S2NR \vdash \bot \] (\( \neg I \, L.2 \))

\[ S2NR \rightarrow \neg \sqrt{2} \rightarrow \text{Nat} \]

\[ \text{ECG} \rightarrow \text{Even, Odd, Real-Gl}
\]

\[ \text{SE} \rightarrow \text{Square-Even} \]

\[ \text{RC} \rightarrow \text{Ratio-Criterion} \]

\[ \text{TND} \rightarrow \text{Terrestrial-Non-Diatomic} \]

\[ W_{1} = \text{ECD, NI, PIC, SN, TND, ON} \]

\[ W_{2} = \text{ECD, NI, PIC, SN, TND, ON, L.187} \]

\[ W_{3} = \text{ECD, NI, PIC, SN, TND, ON, L.188} \]

\[ W_{4} = \text{ECD, NI, PIC, SN, TND, ON, L.188} \]

\[ W_{5} = \text{ECD, NI, PIC, RC, SN, TND, ON, L.14} \]

\[ W_{6} = \text{ECD, NI, PIC, RC, SN, TND, ON, L.14} \]

\[ W_{7} = \text{ECD, NI, PIC, RC, SN, TND, ON, L.14} \]

\[ W_{8} = \text{ECD, NI, PIC, RC, SN, TND, ON, L.14} \]

\[ W_{9} = \text{ECD, NI, PIC, RC, SN, TND, ON, L.14} \]

\[ W_{10} = \text{ECD, NI, PIC, RC, SN, TND, ON, L.14} \]

\[ W_{11} = \text{ECD, NI, PIC, SN, TND, ON, L.14} \]

\[ W_{12} = \text{ECD, NI, PIC, SN, TND, ON, L.14} \]

\[ W_{13} = \text{ECD, NI, PIC, SN, TND, ON, L.14} \]

\[ W_{14} = \text{TND, L.131, L.160} \]

\[ W_{15} = \text{TND, L.131, L.164} \]

\[ W_{16} = \text{TND, L.131, L.171} \]

\[ W_{17} = \text{TND, L.131, L.172} \]

\[ W_{18} = \text{TND, L.131, L.177} \]

A.3 Customizing \LaTeX\ Presentations

The user may customize the \LaTeX\ presentation by providing respective definitions in a \LaTeX\ declarations file. In our case we did employ the following declarations file:

\% Whenever you have manually modified the print style for a theory constant
\% it could be a good idea to add its definition to this file.
\%
\% The definitions here will override/overwrite the standard print style
\% generated by \texttt{post2tex}.
\%
\% Numbers (andfloats)
\%\def\Kat{mathbb{M}}
\%\def\Kint{\mathbb{Z}}
\%\def\Kat{\mathbb{Q}}

49
% Types
\def\ptomicron{s}
\def\ptomnu{\mu}

% Sig Base
\def\ptomBASE\{}\def\ptomBASE{[\ptomicron]}
\def\ptomBASE\{}\def\ptomBASE{[\ptomnu \rightarrow \ptomicron]}

% Sig Set
\def\ptomSET\{}\def\ptomSET{[\ntu \rightarrow \ptomicron]}
\def\ptomSET\{}\def\ptomSET{[\ntu \rightarrow \ptomicron]}\def\ptomSET{[\ntu \rightarrow \ptomicron]}
\def\ptomSET\{}\def\ptomSET{[\ntu \rightarrow \ptomicron]}\def\ptomSET{[\ntu \rightarrow \ptomicron]}

% Sig Struct
\def\ptomSTRUCT\{}\def\ptomSTRUCT{[\ntu \rightarrow \ptomicron]}\def\ptomSTRUCT{[\ntu \rightarrow \ptomicron]}
\def\ptomSTRUCT\{}\def\ptomSTRUCT{[\ntu \rightarrow \ptomicron]}\def\ptomSTRUCT{[\ntu \rightarrow \ptomicron]}
\def\ptomSTRUCT\{}\def\ptomSTRUCT{[\ntu \rightarrow \ptomicron]}\def\ptomSTRUCT{[\ntu \rightarrow \ptomicron]}

% Sig Natural
\def\ptomNAT\{}\def\ptomNAT{[\ntu \rightarrow \ptomicron]}
\def\ptomNAT\{}\def\ptomNAT{[\ntu \rightarrow \ptomicron]}
\def\ptomNAT\{}\def\ptomNAT{[\ntu \rightarrow \ptomicron]}

% Sig Integer
\def\ptomINT\{}\def\ptomINT{[\ntu \rightarrow \ptomicron]}
\def\ptomINT\{}\def\ptomINT{[\ntu \rightarrow \ptomicron]}
\def\ptomINT\{}\def\ptomINT{[\ntu \rightarrow \ptomicron]}

% Sig Rational
\def\ptomRAT\{}\def\ptomRAT{[\ntu \rightarrow \ptomicron]}
\def\ptomRAT\{}\def\ptomRAT{[\ntu \rightarrow \ptomicron]}
\def\ptomRAT\{}\def\ptomRAT{[\ntu \rightarrow \ptomicron]}

% Sig ZZ
\def\ptomZZ\{}\def\ptomZZ{[\ntu \rightarrow \ptomicron]}
\def\ptomZZ\{}\def\ptomZZ{[\ntu \rightarrow \ptomicron]}
\def\ptomZZ\{}\def\ptomZZ{[\ntu \rightarrow \ptomicron]}

50
% Standard Variables
\def\ptotVARX{{\textit{X}_{\textit{x}.3}}}
\def\ptotVARXnum{{\textit{X}_{\textit{x}.0}}_{\{\textit{ptotnu}\}}}
\def\ptotVARD{{\textit{X}_{\textit{x}.2}}}
\def\ptotVARDOCnum{{\textit{X}_{\textit{x}.2}}_{\{\textit{ptotnu}\}}}
\def\ptotVARDFJ{{\textit{X}_{\textit{x}.1}}}
\def\ptotVARDFJnum{{\textit{X}_{\textit{x}.1}}_{\{\textit{ptotnu}\}}}

B \textbf{ΩMEGA Proof Objects}

In this section, we give the proof objects in \textit{POST} syntax. We shall not go into detail of the syntax here. The only interesting part for our purposes is titled nodes and gives the proof nodes. The remaining parts store information necessary for the complete reconstruction of the proof object as it was built during the planning process.

B.1 The Unexpanded Proof

The proof object (PDS) was stored after Step 33 of the interactive session in Section 6 into a file. It corresponds to the LATEX presentation of the unexpanded proof as given in Appendix A and contains 33 proof nodes in total.

```latex
\textbf{QDS (problem SUBTRACT-RAT)}
\textbf{(in REAL)}
\textbf{(declarations (type-variables |(type-constants )}
\textbf{(constants (R SUM) (R SUM) (R SUM)) (meta-variables |(variables ))}
\textbf{(conclusion (SUBTRACT-RAT))}
\textbf{(assumptions)}
\textbf{(open-nodes)}
\textbf{(support-nodes \textit{EVEN-DIVISOR-SQUARE-EVEN-RAT-CRITERION})}
\textbf{Nodes}
\textbf{(L19 (L19) (AND (INT X) (= X (TIMES 2 R)))
\textbf{))}
\textbf{(HYP ("\textit{grounded}" (\textit{false}))
\textbf{))}
\textbf{(L22 (EVEN-DIVISOR-L19) (= X (TIMES 2 R)))
\textbf{))}
\textbf{(\"\textit{assertion}" (L19) \"\textit{expanded}" (}
\textbf{\"\textit{HYPER-EXIST" \"\textit{EXISTS}" \"\textit{EXISTENT}}} ())))
\textbf{(L21 (EVEN-DIVISOR-L19) (INT X)) (\"\textit{assertion}" (L19) \"\textit{expanded}" (}
\textbf{\"\textit{HYPER-EXIST" \"\textit{L24" \"\textit{EXISTENT}}} ())))
\textbf{(L20 (L20) (AND (INT X) (= (TIMES (SHORT 2) X) R)))
\textbf{))}
\textbf{(HYP ("\textit{grounded}" (\textit{false}))
\textbf{))}
\textbf{(L12 (EVEN-DIVISOR-SQUARE-EVEN-LR) \textit{(NOT (EXISTS-SHORT (Lam (YAR76 SUM) \textit{COMMON-DIVISOR H YAR76)}) \textit{INT}))}
\textbf{))}
\textbf{(\"\textit{assertion}" (L20) \"\textit{expanded}" (}
\textbf{\"\textit{HYPER-EXIST" \"\textit{L11" \"\textit{EXISTENT}}} ())))
\textbf{(L11 (EVEN-DIVISOR-SQUARE-EVEN-LR) (= (TIMES (SHORT 2) X) R)))
\textbf{))}
\textbf{(\"\textit{assertion}" (L20) \"\textit{expanded}" (}
\textbf{\"\textit{HYPER-EXIST" \"\textit{L12" \"\textit{EXISTENT}}} ())))
\textbf{(L10 (EVEN-DIVISOR-SQUARE-EVEN-LR) (INT X)) (\"\textit{assertion}" (L20) \"\textit{expanded}" (}
\textbf{\"\textit{HYPER-EXIST" \"\textit{L13" \"\textit{EXISTENT}}} ())))
\textbf{(L4.04) (AND (INT X) (EXISTS-SHORT Lam (YAR79 SUM) (AND (= (TIMES (SHORT 2) X) YAR79)
\textit{NOT (EXISTS-SHORT (Lam (YAR79 SUM) \textit{COMMON-DIVISOR H YAR79)}) \textit{INT})))
\textbf{))}
\textbf{(HYP ("\textit{grounded}" (\textit{false}))
\textbf{))}
\textbf{(L7 (EVEN-DIVISOR-SQUARE-EVEN-L4) \textit{EXISTS-SHORT (Lam (YAR79 SUM) (AND (= (TIMES (SHORT 2) X) YAR79)
\textit{NOT (EXISTS-SHORT (Lam (YAR79 SUM) \textit{COMMON-DIVISOR H YAR79)}) \textit{INT})) \textit{INT}))}
\textbf{))}
\textbf{(\"\textit{assertion}" (L4) \"\textit{expanded}" (}
\textbf{\"\textit{HYPER-EXIST" \"\textit{L6" \"\textit{EXISTENT}}} ())))
\textbf{(L6) (EVEN-DIVISOR-SQUARE-EVEN-L4) (INT X)) (\"\textit{assertion}" (L4) \"\textit{expanded}" (}
\textbf{\"\textit{HYPER-EXIST" \"\textit{L7" \"\textit{EXISTENT}}} ())))
\textbf{(L1 (L1) \textit{RAT (SHORT 2)})
\textbf{))}
\textbf{(HYP ("\textit{grounded}" (\textit{false}))
\textbf{))}
\textbf{(L2 (EVEN-DIVISOR-SQUARE-EVEN-RAT-CRITERION-L1) FALSE)
\textbf{))}
\textbf{(\"\textit{EXISTS-SHORT" ((\textit{RAT-TERM}) (L3 L5) \"\textit{expanded}" (}
\textbf{\"\textit{EXISTENT" \"\textit{EXISTENT" \"\textit{HYPER-EXIST}}} ())))
\textbf{(RAT-CRITERION (RAT-CRITERION-L1) \textit{FORALL-SHORT Lam (YAR81 SUM) (EXISTS-SHORT Lam (YAR82 SUM)
\textit{EXISTS-SHORT (Lam (YAR83 SUM) (AND (= (TIMES (SHORT YAR82) YAR83) \textit{NOT (EXISTS-SHORT (Lam (YAR83 SUM) \textit{COMMON-DIVISOR H YAR82 YAR83)}) \textit{INT})) \textit{INT})) \textit{INT})) \textit{RAT})
\textbf{))}
\textbf{(\\textit{THM} ("\textit{grounded}" (\textit{false}))
\textbf{))}
\textbf{(L9 (EVEN-DIVISOR-SQUARE-EVEN-L8 L4 RAT-CRITERION-L1) FALSE)
\textbf{))}
\textbf{)}
```
200 nodes in total. This $\mathcal{P}DS$ is unfortunately too large to be presented in this report. The base-level view on this $\mathcal{P}DS$ corresponds to the $\LaTeX$ representation of the fully expanded proof in Appendix A.

For an illustration of the way a $\mathcal{P}DS$ is modified by proof expansion we refer to Appendix C. There, we isolated one of the subproblems tackled by Otter in our case study and investigate how the proof generated by Otter and transformed into $\Omega$MEGA by Tramp can be expanded and verified.
C Proof Transformation with TRAMP

C.1 An Isolated Subproblem from the Case Study

In Step 29 of the interactive session in Section 6, for instance, we employed an external ATP, Otter, to close a small gap automatically. The proof generated by Otter is translated into the PDS via TRAMP. There, it can be checked after expansion to Omega's basic calculus layer. We briefly illustrate this here and present the unexpanded and fully expanded proof objects.

(problem sqrt-part
  (in real)
  (constants (m num)(n num))
  (assumption 110 (int m))
  (assumption 116 (int n))
  (assumption 112 (not (exists-sort (lam (x num) (common-divisor n m x)) int)))
  (assumption 117 (evenp m))
  (assumption 127 (evenp n))
  (assumption 128 (int 2))
  (assumption EVEN-COMMON-DIVISOR
    (forall-sort (lam (x num)
      (forall-sort (lam (y num)
        (implies (and (evenp x) (evenp y))
          (common-divisor x y 2)))
        int))
    int))
  (conclusion 129 false)
)

C.2 Interactive Session with a Call of Otter

Omega: read-problem "~/omega/ot/omega/sqrt2-omega/sqrt2-tramp-formalization.post"

Omega: show-pds

L10   (L10) ! (INT M) HYP
L6    (L6)  ! (INT N) HYP
L12   (L12) ! (NOT (EXISTS-SORT (X) (COMMON-DIVISOR N M X)) INT)) HYP
L17   (L17) ! (EVENP M) HYP
L27   (L27) ! (EVENP N) HYP
L28   (L28) ! (INT 2) HYP

EVEN-COMMON-DIVISOR (EVEN-COMMON-DIVISOR) ! (FORALL-SORT
  (X). (FORALL-SORT (Y). (IMPLIES (AND (EVENP X) (EVENP Y))
    (COMMON-DIVISOR X Y 2)))
  INT)) HYP

L29   (L10 L6 ! FALSE OPEN
L12 L17
L27 L28
EVEN-COMMON-DIVISOR)
OMEGA: call-otter-on-node
NODE (ONLINE) Node to prove with OTTER: [L25]
DIR (CREATING-DIRECTORY) The (writeable) directory for depositing OTTER auxiliary
files: [/tmp/chris-atp-dir/]
NODE (SYMBOL) Mode for calling OTTER (auto/user/combined): [AUTO]
EXPAND (SYMBOL) A proof found by OTTER is used to (test/parse/expand): [EXPAND]
PROOF-SUBJECT (BOOLEAN) Use build-proof-object: [T]
USER-FLAG-STRING (STRING) A string of user flag-settings or a file-name: []
USER-WEIGHT-STRING (STRING) A string of user weight-settings or a file-name: []
RESOURCE (INTEGER) A time resource in seconds (integer): [10]
SSSPU-STYLE (SYMBOL) The SSSPU-style (direct/compact/auto): [AUTO]
INDIRECT-PROOF (BOOLEAN) Indirect proof: [T]
INTEGRAL-FORMULAS (BOOLEAN) Integral formulas: [T]
MAXIMAL-DEPTH (INTEGER) Maximal depth of searching integral-formulas: [2]
TND (BOOLEAN) Prefer tertium non datur case analyses: [T]
AVOID-DUPELING (BOOLEAN) Avoid doubling: [T]
LEMMA (SYMBOL) Lemmas over (nil/free/constants/full): [CONSTANTS]

::: Rules loaded for theory BASE.
::: Theorems loaded for theory BASE.
::: Tactics loaded for theory BASE.
::: Methods loaded for theory BASE.
::: Control-rules loaded for theory BASE.
::: Meta-predicates loaded for theory BASE.
::: Strategies loaded for theory BASE.
::: Agents loaded for theory BASE.

OMEGA: Normalizing ...
Calling otter process 1341 with time resource 10sec .
          otter Time Resource in seconds:
          10sec
---------- PROOF ----------
Search stepped by max_proofs option.
Parsing Otter Proof ...
OMEGA*CURRENT-RESOLUTION-PROOF IS SET TO THE FOUND RESOLUTION PROOF
Searching for lemmata ...
Creating Refutation-Graph ...
Translating ...
Translation finished!

OMEGA: show-pds

L10  (L10) ! (INT N)  HYP
L6   (L6)   ! (INT N)  HYP
L12  (L12) ! (NOT (EXISTS-SORT ([X].(COMMON-DIVISOR N M X)) INT)) HYP
L1   (L12) ! (NOT (EXISTS [DC-4424;N4N4] (AND (INT DC-4424)
(COMMON-DIVISOR N M DC-4424)))) HYP
      DEFINE: (KAP[DC-7] . EXIST-SORT KAP[AA]. ([T, S].(EXISTS [X:AA] (AND (S X) (T X)))) ((1 O)) (L12))
L17  (L17) ! (EVENP M)  HYP
L27  (L27) ! (EVENP N)  HYP
L28  (L28) ! (INT 2)   HYP
L30 (L12 L26) ! (NOT (COMMON-DIVISOR N M 2))  ASSERTION: (L1 L26)

EVEN-COMMON-DIVISOR (EVEN-COMMON-DIVISOR) ! (FORALL-SORT HYP
  ([X].
    (FORALL-SORT ([Y]. (IMPLIES (AND (EVENP X) (EVENP Y))
                        (COMMON-DIVISOR X Y 2))))
    (INT))
  (INT))

L3 (L10 L6) ! FALSE  WEAKEN: (L32)
L12 L17
L27 L28
EVEN-COMMON-DIVISOR

L2 (EVEN-COMMON-DIVISOR) ! (FORALL-
  [DC-4456:NUM]
  (IMPLIES (INT DC-4456)
    (FORALL
      [DC-4474:NUM]
      (IMPLIES
        (INT DC-4474)
        (IMPLIES
          (AND (EVENP DC-4456) (EVENP DC-4474))
          (COMMON-DIVISOR DC-4456 DC-4474 2))))))
  (KAP[DC-6]). FORALL-SORT KAP[AA]. [(V, U).(FORALL [X:AA] (IMPLIES (U X) (V X))) ((O) (O O))]
  (EVEN-COMMON-DIVISOR)

L31 (EVEN-COMMON-DIVISOR ! (NOT (INT N))  ASSERTION: (L2 L10 L27 L17 L30)
L10 L27 L17 L12 L26)

L32 (L6) ! FALSE  NOTE: (L6 L31)
EVEN-COMMON-DIVISOR
L10 L27 L17 L12 L26

L29 (L10 L6 L12 L17) ! FALSE  simplify-goal: (L3)
L27 L28
EVEN-COMMON-DIVISOR

SMEGA: check-proof
TACTIC-LIST (SYMBOL-LIST) The tactics that should not be expanded: [][]

Expanding nodes......
Expanding the node L1 ....
Expanding the node L30 ....
Creating rule tree #:L1 .......
Expanding the node L2 ....
Expanding the node L31 ....
Creating rule tree #:L2 _____________________________
Expanding the node L29 ....
Expanding the node L33 ....
Expanding the node L36 ....
Expanding the node L36 ....
Expanding the node L30 ....
Expanding the node L40 ....
Expanding the node L42 ....
Expanding the node L45 ....
Expanding the node L31 ....
Expanding the node L52 ....

58
Expanding the node 1.54 ...
Expanding the node 1.56 ...
Expanding the node 1.55 ...
Expanding the node 1.56 ...
Expanding the node 1.58 ...
Expanding the node 1.73 ...
Expanding the node 1.50 ...
Expanding the node 1.26 ...

Checking nodes.
Node #<pdsn+ node #1.50> has a correct justification.
Node #<pdsn+ node #1.57> has a correct justification.
Node #<pdsn+ node #1.54> has a correct justification.
Node #<pdsn+ node #1.51> has a correct justification.
Node #<pdsn+ node #1.56> has a correct justification.
Node #<pdsn+ node #1.57> has a correct justification.
Node #<pdsn+ node #1.52> has a correct justification.
Node #<pdsn+ node #1.56> has a correct justification.
Node #<pdsn+ node #1.54> has a correct justification.
Node #<pdsn+ node #1.53> has a correct justification.
Node #<pdsn+ node #1.56> has a correct justification.
Node #<pdsn+ node #1.55> has a correct justification.
Node #<pdsn+ node #1.56> has a correct justification.
Node #<pdsn+ node #1.56> has a correct justification.
Node #<pdsn+ node #1.149> has a correct justification.
Node #<pdsn+ node #1.168> has a correct justification.
Node #<pdsn+ node #1.169> has a correct justification.
Node #<pdsn+ node #1.168> has a correct justification.
Node #<pdsn+ node #1.149> has a correct justification.
Node #<pdsn+ node #1.168> has a correct justification.
Node #<pdsn+ node #1.169> has a correct justification.
Node #<pdsn+ node #1.168> has a correct justification.
Node #<pdsn+ node #1.149> has a correct justification.
Node #<pdsn+ node #1.168> has a correct justification.
Node #<pdsn+ node #1.169> has a correct justification.
Node #<pdsn+ node #1.168> has a correct justification.
Node #<pdsn+ node #1.149> has a correct justification.
Node #<pdsn+ node #1.168> has a correct justification.
Node #<pdsn+ node #1.169> has a correct justification.
Node #<pdsn+ node #1.168> has a correct justification.
Node #<pdsn+ node #1.149> has a correct justification.
Node #<pdsn+ node #1.168> has a correct justification.
Node #<pdsn+ node #1.169> has a correct justification.
Node #<pdsn+ node #1.168> has a correct justification.
Node #<pdsn+ node #1.149> has a correct justification.
Node #<pdsn+ node #1.168> has a correct justification.
Node #<pdsn+ node #1.169> has a correct justification.
Node #<pdsn+ node #1.168> has a correct justification.
Node #<pdn+node #:1.174> has a correct justification.
Node #<pdn+node #:1.168> has a correct justification.
Node #<pdn+node #:1.169> has a correct justification.
Node #<pdn+node #:1.161> has a correct justification.
Node #<pdn+node #:1.164> has a correct justification.
Node #<pdn+node #:1.160> has a correct justification.
Node #<pdn+node #:1.158> has a correct justification.
Node #<pdn+node #:1.157> has a correct justification.
Node #<pdn+node #:1.151> has a correct justification.
Node #<pdn+node #:1.147> has a correct justification.
Node #<pdn+node #:1.137> has a correct justification.
Node #<pdn+node #:1.13> has a correct justification.
Node #<pdn+node #:1.129> has a correct justification.
Node #<pdn+node #:1.125> has a correct justification.
Node #<pdn+node #:1.144> has a correct justification.
Node #<pdn+node #:1.1077> has a correct justification.
Node #<pdn+node #:1.1065> has a correct justification.
Node #<pdn+node #:1.1080> has a correct justification.
Node #<pdn+node #:1.131> has a correct justification.
Node #<pdn+node #:1.122> has a correct justification.
Node #<pdn+node #:1.11> has a correct justification.
Node #<pdn+node #:1.1069> has a correct justification.
Node #<pdn+node #:1.1068> has a correct justification.
Node #<pdn+node #:1.1067> has a correct justification.
Node #<pdn+node #:1.1066> has a correct justification.
Node #<pdn+node #:1.1065> has a correct justification.
Node #<pdn+node #:1.1064> has a correct justification.
Node #<pdn+node #:1.1063> has a correct justification.
Node #<pdn+node #:1.1062> has a correct justification.
Node #<pdn+node #:1.144> has a correct justification.
Node #<pdn+node #:1.197> has a correct justification.
Node #<pdn+node #:1.196> has a correct justification.
Node #<pdn+node #:1.190> has a correct justification.
Node #<pdn+node #:1.189> has a correct justification.
Node #<pdn+node #:1.189> has a correct justification.
Node #<pdn+node #:1.191> has a correct justification.
Node #<pdn+node #:1.190> has a correct justification.
Node #<pdn+node #:1.194> has a correct justification.
Node #<pdn+node #:1.194> has a correct justification.
Node #<pdn+node #:1.189> has a correct justification.
Node #<pdn+node #:1.188> has a correct justification.
Node #<pdn+node #:1.189> has a correct justification.
Node #<pdn+node #:1.189> has a correct justification.

Unexpanding nodes......

Well done, the proof is correct.

C.3 The Unexpanded Proof Object Generated by TRAMP

In this section, we give the proof objects in POST syntax. We shall not go into the details of the syntax here. The only interesting part for our purposes is titled nodes and gives the proof nodes. The remaining parts store information necessary for the complete reconstruction of the proof object as it was built during the planning process.
C.4 The Expanded Proof Object

In this section, we give the proof objects in POST syntax. We shall not go into the details of the syntax here. The only interesting part for our purposes is titled nodes and gives the proof nodes. The remaining parts store information necessary for the complete reconstruction of the proof object as it was built during the planning process.

```plaintext
(type-variables ) (type-constants ) (constants (VARS121 SUM) (VARS116 SUM)) (meta-variables )
```
D \textbf{\textit{\OMEGA}’s Knowledge Base}

\textit{\OMEGA}’s knowledge base is hierarchically structured. A theory with name \textit{(theory)} comprises definitions (given in a file \textit{(theory).thy}), lemmata and theorems (file \textit{(theory)-theorems.thy}), proof problems (file \textit{(theory)-problems.thy}), inference rules (file \textit{(theory)-rules.thy}), proof tactics (file \textit{(theory)-tactics.thy}), proof methods (file \textit{(theory)-methods.thy}), and \textsc{\textit{\OMEGA}-ants} agents (file \textit{(theory)-agents.thy}).

Here, we present only the files \textit{real.thy}, \textit{real-theorems.thy}, \textit{rational.thy}, \textit{rational-theorems.thy}, \textit{integer.thy}, \textit{integer-theorems.thy}, \textit{natural.thy}, and \textit{natural-theorems.thy}.

D.1 Theory Real

D.1.1 \textit{real.thy}

(\textbf{\texttt{th-deftheory REAL}}
 (uses rational sequences)
 (constants \texttt{(completion \{\texttt{struct sum} \texttt{(struct sum)}\}))
 (help \texttt{Peano arithmetic for real numbers.}))

(\textbf{\texttt{th-def constant real}}
 (is real)
 (type \texttt{(\textbf{\texttt{\_\_}\_})})
 (sort))

(\textbf{\texttt{th-def real\_0}}
 (in real)
 (sort)
 (definition \texttt{(setminus real \{\texttt{singleton zero}\})))
 (help \texttt{The set of reals without 0.}))

(\textbf{\texttt{th-def real\_struct}}
 (in real)
 (definition \texttt{(completion rat-struct)})
 (help \texttt{The real numbers, defined as the completion of the rational numbers.}))

(\textbf{\texttt{th-defaxion real\_plus\_closed}}
 (in real)
 (formula \texttt{(closed-under real plus)})
 (help \texttt{Plus is closed.})

(\textbf{\texttt{th-defaxion real\_times\_closed}}
 (in real)
 (formula \texttt{(closed-under real times)})
 (help \texttt{Times is closed.})

(\textbf{\texttt{th-defaxion real\_plus\_assoc}}
 (in real)
 (formula \texttt{(associative real plus)})
 (help \texttt{Plus is associative.})

(\textbf{\texttt{th-defaxion real\_times\_assoc}}
 (in real)
 (formula \texttt{(associative real times)})
 (help \texttt{Times is associative.})

(\textbf{\texttt{th-defaxion real\_plus\_comm}}
 (in real)
 (formula \texttt{(commutative real plus)})
 (help \texttt{Plus is commutative.})

(\textbf{\texttt{th-defaxion real\_times\_comm}}
 (in real)
 (formula \texttt{(commutative real times)})
 (help \texttt{Times is commutative.})

(\textbf{\texttt{th-defaxion real\_plus\_times\_distrib}}
 (in real)
 (formula \texttt{(distributive real plus times)})
 (help \texttt{Distributivity for plus and times.})

(\textbf{\texttt{th-defaxion real\_plus\_unit}}
 (in real)
(forall (lan (x sum))
  (equiv (and (leq x r) (geq x l))
    (in x g))))

(help 'Predicate for closed intervals of real numbers. '))

(th 'defdef closed-interval
  (in real)
  (definition
    (lan (G (o sum)))
    (closed-interval-with-bounds G
      (supremum real-struct G)
      (infimum real-struct G))))

(help 'Predicate for closed intervals of real numbers. '))

(th 'defdef open-interval-with-bounds
  (in real)
  (definition
    (lan (G (o sum)))
    (lan (x sum))
    (lan (r sum))
    (forall (lan (x sum))
      (forall (lan (z sum))
        (equiv (and (less x r) (greater z l))
          (in x g))))

(help 'Predicate for open intervals of real numbers. '))

(th 'defdef open-interval
  (in real)
  (definition
    (lan (G (o sum)))
    (open-interval-with-bounds G
      (supremum real-struct G)
      (infimum real-struct G))))

(help 'Predicate for open intervals of real numbers. '))

(th 'defdef closed-interval-bounds
  (in real)
  (definition
    (lan (x sum))
    (lan (y sum))
    (lan (z sum))
    (and (leq x z) (geq x y))))

(help 'The closed interval for given bounds. '))

(th 'defdef open-interval-bounds
  (in real)
  (definition
    (lan (x sum))
    (lan (y sum))
    (lan (z sum))
    (and (less x z) (greater y)))

(help 'The open interval for given bounds. '))

(th 'defdef interval-center
  (in real)
  (definition
    (lan (Ii (o sum)))
    (divide (cmix (supremum real-struct Ii)
      (infimum real-struct Ii))
      (is one))))

(help 'The center of an interval. '))

(th 'defdef sqrt
  (in real)
  (definition
    (lan (x sum))
    (that (lan (y sum) (= (power y 2) z))))

(help 'Definition of square root. '))
D.1.2 real-theorems.thy

(th`deftheorem plus-real-step2
  (in real)
  (conclusion
    (forall (lan (x y z))
      (forall (lan (y z))
        (implies (and (in x real) (in y real))
          (in (plus x y) real))))
    (forall (lan (x y z))
      (forall (lan (x y))
        (implies (and (in x real) (in y real))
          (in (plus x y) real)))))
  (help "The recursive definition of plus on the right term.")
)

(th`deftheorem plus-real-closed
  (in real)
  (conclusion
    (forall (lan (x y z))
      (forall (lan (x y z))
        (implies (and (in x real) (in y real))
          (in (plus x y) real)))
    (forall (lan (x y z))
      (forall (lan (x y z))
        (implies (and (in x real) (in y real))
          (in (plus x y) real))))))

(th`deftheorem nat-plus-real
  (in real)
  (conclusion
    (forall (lan (x y z))
      (forall (lan (x y z))
        (implies (and (nat x real) (nat y real) (nat z real))
          (nat (plus x y) real)))
    (forall (lan (x y z))
      (forall (lan (x y z))
        (implies (and (nat x real) (nat y real) (nat z real))
          (nat (plus x y) real))))))

(th`deftheorem c-plus-real
  (in real)
  (conclusion
    (forall (lan (x y z))
      (forall (lan (x y z))
        (implies (and (in x real) (in y real))
          (in (plus x y) real))))
    (forall (lan (x y z))
      (forall (lan (x y z))
        (implies (and (in x real) (in y real))
          (in (plus x y) real))))))

(th`deftheorem 0-plus-real
  (in real)
  (conclusion (forall (lan (x y z)) (implies (in x real) (= (plus 0 x) x))))

(th`deftheorem int-times-real
  (in real)
  (conclusion (forall (lan (x y z)) (implies (in x real) (= (int times 1 x) x))))

(th`deftheorem 0-times-real
  (in real)
  (conclusion (forall (lan (x y z)) (implies (in x real) (= (times 0 x) 0))))

(th`deftheorem nat-times-real
  (in real)
  (conclusion (forall (lan (x y z)) (implies (in x real) (= (nat times y z) y))))
    (forall (lan (x y z))
      (forall (lan (x y z))
        (implies (and (int x real) (int y real) (int z real))
          (int (times x y) z))
      (forall (lan (x y z))
        (forall (lan (x y z))
          (implies (and (nat x real) (nat y real) (nat z real))
            (nat (times x y) (times y z)))
      (forall (lan (x y z))
        (forall (lan (x y z))
          (implies (and (nat x real) (nat y real) (nat z real))
            (nat (times x y) (times y z)))
  (forall (lan (x y z))
    (forall (lan (x y z))
      (implies (and (nat x real) (nat y real) (nat z real))
        (nat (times x y) (times y z))))))

(th`deftheorem Dist-Right-real
  (in real)
  (conclusion
    (forall (lan (x y z))
      (forall (lan (x y z))
        (implies (and (int x real) (int y real) (int z real))
          (int (times y z) x))
      (forall (lan (x y z))
        (forall (lan (x y z))
          (implies (and (nat x real) (nat y real) (nat z real))
            (nat (times y z) x))
      (forall (lan (x y z))
        (forall (lan (x y z))
          (implies (and (nat x real) (nat y real) (nat z real))
            (nat (times y z) x))))))

(th`deftheorem Dist-Left-real
  (in real)
  (conclusion
    (forall (lan (x y z))
      (forall (lan (x y z))
        (implies (and (int x real) (int y real) (int z real))
          (int (times x y) z))
      (forall (lan (x y z))
        (forall (lan (x y z))
          (implies (and (nat x real) (nat y real) (nat z real))
            (nat (times x y) z))
      (forall (lan (x y z))
        (forall (lan (x y z))
          (implies (and (nat x real) (nat y real) (nat z real))
            (nat (times x y) z))))))

72
(in real) (forall (lam (x sum)) (forall (lam (y sum)) (forall (lam (z sum)) (implies (and (in x real) (and (in y real) (in z real))) (x (times x (plus y z)) (plus (times x y) (times x z))))))

(th 'deftheorem nisux2plus (in real) (forall (lam (x sum)) (forall (lam (y sum)) (implies (and (in x real) (in y real)) (x (nise x y) (plus x (times -1 y))))))

(th 'deftheorem power-1-real (in real) (forall (lam (x sum)) (implies (in x real) (equal (power x 1) x))))

(th 'deftheorem 1-power-real (in real) (forall (lam (x sum)) (implies (in x real) (equal (power 1 x) 1))))

(th 'deftheorem 0-power-real (in real) (forall (lam (x sum)) (implies (in x real) (equal (power 0 x) 0))))

(th 'deftheorem power-0-real (in real) (forall (lam (x sum)) (implies (in x real) (equal (power x 0) 1))))

(th 'deftheorem times-power-real (in real) (forall (lam (x sum)) (forall (lam (y sum)) (forall (lam (z sum)) (implies (and (in x real) (and (in y real) (in z real))) (x (times x y z) (times (power x y) (power x z))))))

(th 'deftheorem times-power-2-real (in real) (forall (lam (x sum)) (forall (lam (y sum)) (forall (lam (z sum)) (implies (and (in x real) (and (in y real) (in z real))) (x (times (times x y) z) (times (power x z) (power y z))))))

(th 'deftheorem power-power-real (in real) (forall (lam (x sum)) (forall (lam (y sum)) (forall (lam (z sum)) (implies (and (in x real) (and (in y real) (in z real))) (x (power (power x y) z) (power (times x y) z))))))

(th 'deftheorem rat-criterion (in rational) (forall-sort (lam (x sum)) (exists-sort (lam (y sum)) (exists-sort (lam (z sum)) (and (in x sum) (in y sum) (in z sum) (and (in y real) (in z real)) (x (times x y z) (times (expression y z) (expression x z)) (int)))) int)))

'rat))

'Help"'x rational implies there exist integers y,z which have no common divisor with x*y*z."')
D.2 Theory Rational

D.2.1 rational.thy

(th:deftheory RATIONAL
  (uses integer)
  (help "Peano Arithmetic for rationals."))

(th:defconstant frac
  (in rational)
  (type (num num num))
  (help "The fraction constructor for rational numbers"))

(th:defconstant rat-struct
  (in rational)
  (type (struct num))
  (help "The structure of rational numbers with addition as operation"))

(th:defconstant rat-mul-struct
  (in rational)
  (type (struct num))
  (help "The structure of non-zero rational numbers with multiplication as operation"))

(th:defdef rat
  (in rational)
  (definition
   (lam (x num)
     (exists-sort (lam (y num)
        (exists-sort (lam (z num)
           (= x (frac y z)))
           (int)))
           (sort)
           (help "The set of rationals, constructed as fractions a/b of integers."))
    )
  )

(th:defaxiom reduce-frac
  (in rational)
  (formula
   (forall-sort (lam (x num)
     (forall-sort (lam (y num)
        (forall-sort (lam (z num)
           (implies (not (z zero))
           (= (frac (times x z) (times y z))
                (frac x y)))
           (rat))
           (rat)
           (help "Reducing fractions by cancellation."))
        )
     )
   )
  )

(th:defdef numerator
  (in rational)
  (definition
   (lam (x num)
     (that (lam (y num)
        (exists (lam (z num)
           (= x (frac y z)))))
     )
     (help "The numerator of a fraction x/y is x."))
  )

(th:defdef denominator
  (in rational)
  (definition
   (lam (x num)
     (that (lam (y num)
        (exists (lam (z num)
           (= x (frac z y))))))
     )
     (help "The numerator of a fraction x/y is y."))
  )

(th:defdef divide
  (in rational)
  (definition
   (lam (x num)
     (lam (y num)
       (frac (times (numerator x) (denominator y))
            (times (denominator x) (numerator y)))))
     (help "The division operator of the rationals."))
  )

(th:defdef one-over
  (in rational)
  (definition
   (lam (x num)
     (lam (y num)
       (frac (times (numerator x) (denominator y))
            (times (denominator x) (numerator y)))))
     (help "The division operator of the rationals."))
  )

74
; (divide one x))
; (help "The multiplicative inversion operator of the rationals.")

(th'defn one-over
  (in rational)
  (definition
    (lam (x y)
      (frac (denominator x) (numerator y)))))
  (help "The reciprocal value of a fraction.")

(th'defn divide
  (in rational)
  (definition
    (lam (x y)
      (times x (one-over y))))
  (help "The division operator of the rational numbers.")

(th'defn plus-frac
  (in rational)
  (formula
    (forall-sort (lam (x y z)
      (forall-sort (lam (y z)
        (frac (plus x y)
          (frac (plus (times (numerator x) (denominator y))
            (times (numerator y) (denominator x)))
            (times (denominator y) (denominator x)))))))
      (rat)))
  (help "The axiom for plus on the rationals.")

(th'defn time-frac
  (in rational)
  (formula
    (forall-sort (lam (x y z)
      (forall-sort (lam (y z)
        (frac (times x y)
          (times (numerator x) (denominator y)))
            (times (denominator y) (denominator x)))))))
      (rat))
  (help "The axiom for times on the rationals.")

(th'defn rat-mul-struct
  (in rational)
  (formula
    (and (= (struct-set rat-mul-struct) rat)
      (times (struct-op rat-mul-struct) times))))
  (help "The group of rationals with operation times.")

(th'defn rat-struct
  (in rational)
  (formula
    (and (= (struct-set rat-struct) rat)
      (= (struct-op rat-struct) plus)
      (and (= (struct-nil-group rat-struct) rat-nil-struct)
        (= (struct-ordering rat-struct) leq)))
  (help "The ordered field of rationals with operation plus.")

(th'defn divide
  (in rational)
  (definition
    (apply-pointwise-2 divide)))
  (help "The definition of pointwise division of functions.")

(th'defn int-rat
  (in rational)
  (formula (forall-sort (lam (x y z) (rat z)) int))
  (help "An integer is rational.")

75
D.2.2 rational-theorems.thy

(th`defthm rat-crit
  (in rational)
  (conclusion
    (forall-sort (lam (x zm))
      (forall-sort (lam (y zm))
        (implies (and (less (z zero) y)
                      (= (gcd x y) (z zero)))
          (rat (frac x y))))
      nat))
  (help "A criterion for a number being rational.")
)

(th`defthm cancel-fraction
  (in rational)
  (conclusion
    (forall-sort (lam (x zm))
      (forall-sort (lam (y zm))
        (forall-sort (lam (z zm))
          (implies (and (or (less zero x)
                          (less zero y))
            (= (frac (times x z) (times y z))
                   (frac x y))))
        nat))
  (int)
  (help "A theorem for cancelling fractions.")
)

(th`defthm numerator-equal-zero
  (in rational)
  (conclusion
    (forall-sort (lam (z zm))
      (implies (less zero x)
        (= (frac x (z zero))
            (int))
    (help "A rational number with numerator zero is the integer zero.")
)

(th`defthm int-to-rat
  (in rational)
  (conclusion
    (forall-sort (lam (z zm))
      (= x (frac x (z zero)))
      int))
  (help "Conversion of integers to fractions.")
)

(th`defthm rat-to-int
  (in rational)
  (conclusion
    (forall-sort (lam (x zm))
      (= (frac x (z zero)) z)
    int))
  (help "Conversion of fractions with denominator one to integers.")
)

(th`defthm numerator-of-frac
  (in rational)
  (conclusion
    (forall-sort (lam (x zm))
      (forall-sort (lam (y zm))
        (implies (rat (frac x y))
          (= (numerator (frac x y)) z))
    int))
  (help "Extraction of the numerator of a rational number.")
)

(th`defthm denominator-of-frac
  (in rational)
  (conclusion
    (forall-sort (lam (x zm))
      (forall-sort (lam (y zm))
        (implies (rat (frac x y))
          (= (denominator (frac x y)) y)))
    int))
  (help "Extraction of the denominator of a rational number.")
)

(th`defthm numerator-of-int
  (in rational)
  (conclusion)
(forall-sort (lam (x num)
  (= (numerator x) x))
  int))
  (help "The numerator of an whole-numbered number is the number itself.")

(deftheorem denominator-of-int
  (in rational)
  (conclusion
    (forall-sort (lam (x num)
                  (= (denominator x) (s zero)))
                 int))
  (help "The denominator of an whole-numbered number is one.")

(deftheorem plus-rat
  (in rational)
  (conclusion
    (forall-sort (lam (x num)
                     (forall-sort (lam (y num)
                                   (= (plus x y)
                                       (frac (plus (times (numerator x) (denominator y))
                                               (times (numerator y) (denominator x)))
                                             (times (denominator x) (denominator y))))
                                   rat)))
    rat))
  (help "Brute force addition on rational numbers.")

(deftheorem plus-rat-equal-denoms
  (in rational)
  (conclusion
    (forall-sort (lam (x num)
                     (forall-sort (lam (y num)
                                   (= (plus x y)
                                       (frac (plus (times (numerator x) (denominator y))
                                               (times (numerator y) (denominator x)))
                                             (denominator x))))
                                   rat)))
    rat))
  (help "Addition on rational numbers with equal denominators")

(deftheorem plus-rat-expanded-fracs
  (in rational)
  (conclusion
    (forall-sort (lam (x num)
                     (forall-sort (lam (y num)
                                   (implies (= (less zero x)
                                               (= (plus (frac x y) (frac y x))
                                               (frac (plus x y) x))))
                                   rat))
                     int))
    int))
  (help "Addition of expanded fractions.")

(deftheorem change-sign-rat
  (in rational)
  (conclusion
    (forall-sort (lam (x num)
                     (= (change-sign x)
                         (frac (change-sign (numerator x))
                               (denominator x))))
                     rat))
  (help "Unary minus on rational numbers.")

(deftheorem times-rat
  (in rational)
  (conclusion
    (forall-sort (lam (x num)
                     (forall-sort (lam (y num)
                                   (= (times x y)
                                       (frac (times (numerator x) (numerator y))
                                             (times (denominator x) (denominator y))))
                                   rat)))
    rat))
  (help "Multiplication on rational numbers.")

(deftheorem power-rat-rat
  (in rational)
  (conclusion
    (forall-sort (lam (x num)
                     (forall-sort (lam (y num)
                                   (= (power x y)
                                       (frac (power (numerator x) (numerator y))
                                             (power (denominator x) (denominator y))))
                                   rat)))
    rat))
(\text{rat})\)
(help "Natural powers of rational numbers.")

(th\'deftheorem power-rat-smaller-one
(in rational)
(conclusion
(forall-sort \(\text{int}\) (\(x\) \text{mm} (\(y\) \text{mm})

(| (\(x\) \text{zero})

| (\(y\) \text{zero}))

| (\(z\) \text{zero}))

| (\text{int}))

(help "Powers with base one.")

(th\'deftheorem less-rat
(in rational)
(conclusion
(forall-sort \(\text{int}\) \(x\) \(y\) \text{mm} \(z\) \text{mm}

| (\text{less \(x\) \(y\) \(z\) \text{mm}})

| (\text{less \(y\) \(z\) \text{mm}})

| (\text{less \(z\) \(y\) \text{mm}})

| (\text{less \(z\) \(x\) \text{mm}})

| (\text{rat}))

(help "Less on rational numbers.")

(th\'deftheorem less-rat-seg-and-pos
(in rational)
(conclusion
(forall-sort \(\text{int}\) \(x\) \(y\) \text{mm} \(z\) \text{mm}

| (\text{less \(x\) \(y\) \(z\) \text{mm}})

| (\text{less \(y\) \(z\) \text{mm}})

| (\text{less \(z\) \(y\) \text{mm}})

| (\text{less \(z\) \(x\) \text{mm}})

| (\text{rat}))

(help "A negative rational number is smaller than a positive rational number.")

(th\'deftheorem less-implies-leq-rat
(in rational)
(conclusion
(forall-sort \(\text{int}\) \(x\) \(y\) \text{mm} \(z\) \text{mm}

| (\text{less \(y\) \(x\) \text{mm}})

| (\text{leq \(x\) \(y\) \text{mm}})

| (\text{rat}))

(help "Less implies less or equal.")

(th\'deftheorem equal-implies-leq-rat
(in rational)
(conclusion
(forall-sort \(\text{int}\) \(x\) \(y\) \text{mm} \(z\) \text{mm}

| (\text{equal \(x\) \(y\) \text{mm}})

| (\text{leq \(y\) \(x\) \text{mm}})

| (\text{rat}))

(help "Equal implies less or equal.")

D.3 Theory Integer

D.3.1 Integer.thy

(th\'deftheory INTEGER
(uses natural)
(help "Peano Arithmetic for integers.")

78
(th `defconstant int-struct
  (in integer)
  (type (struct sum))
  (help "The structure of integers together with addition as an operation"))

(th `defconstant int-mul-struct
  (in integer)
  (type (struct sum))
  (help "The structure of integers together with multiplication as an operation"))

(th `defdef int
  (in integer)
  (sort)
  (definition (union nat zmat))
  (help "The set of integers, constructed as the natural numbers and their negatives."))

(th `defdef int\()
  (in integer)
  (sort)
  (definition (setminus int (singleton zero)))
  (help "The set of integers without 0."))

; (th `defsort int
  (in integer)
  (by int-sort-like)

; (th `defthorem int-sort-like
  (in integer)
  (conclusion conc (forall-sort (lan (x sum) (or (int x) (not (int x)))) defined)))

; (th `defthorem pred-succ
  (in integer)
  (formula (forall-sort (lan (x sum) (= x (p (s x)))) int))
  (help "The predecessor of the successor of x is x."))

; (th `defthorem succ-pred
  (in integer)
  (formula (forall-sort (lan (x sum) (= x (s (p x)))) int))
  (help "The predecessor of the successor of x is x."))

(th `defaxiom nat-int
  (in integer)
  (formula (forall-sort (lan (x sum) (int x)) nat))
  (termdecl)
  (help "A natural number is whole-numbered."))

(th `defaxiom nat-int
  (in integer)
  (formula (forall-sort (lan (x sum) (int x)) nat))
  (help "A positive integer is whole-numbered."))

(th `defaxiom gp-int
  (in integer)
  (formula (forall-sort (lan (x sum) (= (s (p x)) x)) int))
  (help "The successor of the predecessor of a number equals the number itself."))

(th `defaxiom gp-int
  (in integer)
  (formula (forall-sort (lan (x sum) (= (p (s x)) x)) int))
  (help "The successor of the predecessor of a number equals the number itself."))

(th `defdef minus
  (in integer)
  (definition (lan (y sum)
                   (plus x (change-sign y))))
  (help "The difference operators on natural numbers."))
(th'defaxion plus-\operator{int}
 (in integer)
 (formula
   (forall-sort (lam (x num))
     (forall-sort (lam (y num))
       (and (= (plus (change-sign x) (change-sign y))
           (change-sign (plus x y)))
       (and (= (plus x y)
           (that (lam (z num))
             (= x (plus y z))))
       (= (plus x y)
           (that (lam (z num))
             (= y (plus x z))))))))

  (help 'Extension of plus to the integers.'))

(th'defaxion time-\operator{int}
 (in integer)
 (formula
   (forall-sort (lam (x num))
     (forall-sort (lam (y num))
       (= (times (change-signs x) y)
           (change-sign (times x y)))))
     (help 'Extension of multiplication to the integers.'))

(th'defaxion \operator{ist-mul-struct}
 (in integer)
 (formula
   (and (= (struct-set ist-mul-struct) ist)
     (= (struct-op ist-mul-struct) times))
     (help 'The monoid of integers with operation times.'))

(th'defaxion \operator{ist-struct}
 (in integer)
 (formula
   (and (= (struct-set ist-struct) ist)
     (= (struct-op ist-struct) plus))
     (help 'The ordered ring of integers with operation plus.'))

(th'defaxion div
 (in integer)
 (definition
   (lam (x num))
   (lam (y num))
   (that (lam (z num))
     (and (= (seq (times y z) x)
         (greater (times (s x) y) z))))))
     (help 'The div operator for natural numbers.'))

(th'defaxion divisor
 (in integer)
 (definition
   (lam (x num))
   (lam (y num))
   (exists (lam (x num))
     (and (in z int)
       (= y (times x z)))))
     (help 'The predicate for integer divisibility.'))

(th'defaxion common-divisor
 (in integer)
 (definition
   (lam (x num))
   (lam (y num))
   (lam (z num))
   (and (in x int)
     (in y int)
     (divisor z x)
     (divisor z y)))))

80
(defn common-multiple
  (in integer)
  (definition
    (lan (x num)
      (lan (y num)
        (and (in x int) (in y int))
        (and (in y int)
          (and (divisor x y)
            (divisor y x)))))))
  (help "The predicate for non-trivial common integer divisibility.\")

(defn gcd
  (in integer)
  (definition
    (lan (x num)
      (lan (y num)
        (maximum int-struct (lan (z num) (common-divisor z x y))))))
  (help "The predicate for common integer divisibility.\")

(defn lcm
  (in integer)
  (definition
    (lan (x num)
      (lan (y num)
        (minimum int-struct (lan (z num) (common-multiple x y))))))
  (help "The predicate for common integer divisibility.\")

(defn mod
  (in integer)
  (definition
    (lan (x num)
      (lan (y num)
        (lambda (lan (z num)
          (= x (plus z (times (div x y)))))))
      (help "The mod operator for natural numbers.\")

(defn minus
  (in integer)
  (definition
    (apply-pointwise=- minus))
  (help "The definition of pointwise subtraction of functions.\")

(defn integer-interval
  (in integer)
  (definition
    (lan (x num)
      (lan (y num)
        (lan (elem num)
          (and (int elem)
            (and (leq x elem)
              (leq elem y)))))))
  (help "The set of all integers in the closed interval from x to y.\")

(defn common-divisor-p
  (in integer)
  (definition
    (lan (x num)
      (lan (y num)
        (exists-sort (lan (z num) (common-divisor x y z)) int)))))
  (help "Definition of the property of having a common divisor.\")

(defn empty
  (in integer)
  (definition
    (lan (x num)
      (exists-sort (lan (y num)
        (= x (times 2 y))))
D.3.2 integer-theorems.thy

(th 'deftheorem neutral-of-Sat
  (in integer)
  (conclusion (= zero (struct-ent sat-plus-struct)))
  (help "The constant zero, is the neutral element of SatPlus."))

(th 'defsimplifier neutral-of-Sat-simp
  (in integer)
  (status global)
  (equation neutral-of-Sat)
  (direction rl)
  (help "Simplify the neutral of the sat-struct."))

(th 'deftheorem no-fix-succ
  (in integer)
  (conclusion THM
    (forall (lam (X sum)
      (implies (sat x)
        (not (= (s x) X)))))
  )
  (help "The successor function has no fixed point."))

(th 'deftheorem assoc-plus-Sat
  (in integer)
  (conclusion THM (associative sat plus)))

(th 'deftheorem assoc-times-Sat
  (in integer)
  (conclusion THM (associative sat times)))

(th 'deftheorem commutative-plus-Sat
  (in integer)
  (conclusion THM (commutative sat plus)))

(th 'deftheorem commutative-times-Sat
  (in integer)
  (conclusion THM (commutative sat times)))

(th 'deftheorem closed-plus-Sat
  (in integer)
  (conclusion THM (closed-under-2 sat plus)))

(th 'deftheorem closed-times-Sat
  (in integer)
  (conclusion THM (closed-under-2 sat times)))

; (th 'deftheorem semigroup-SatPlus
;   (in integer)
;   (conclusion THM (semigroup Sat-Plus-struct)))
;
; (th 'deftheorem semigroup-times-Sat
;   (in integer)
;   (conclusion THM (semigroup Sat-Times-struct)))
;
; (th 'deftheorem monoid-SatPlus
;   (in integer)
;   (conclusion THM (monoid Sat-Plus-struct)))
;
; (th 'deftheorem monoid-times-Sat
;   (in integer)
;   (conclusion THM (monoid Sat-Times-struct)))

(th 'deftheorem assoc-plus-Int
  (in integer)
  (conclusion THM (associative int plus)))

(th 'deftheorem assoc-times-Int
  (in integer)
  (conclusion THM (associative int times)))

(th 'deftheorem commutative-plus-Int
  (in integer)
  (conclusion THM (commutative int plus)))
(th `deftheorem commutative-time-int
  (in integer)
  (conclusion THM (commutative int times)))

(th `deftheorem closed-plus-int
  (in integer)
  (conclusion THM (closed-under-2 int plus)))

(th `deftheorem closed-times-int
  (in integer)
  (conclusion THM (closed-under-2 int times)))

;(th `deftheorem semigroup-plus-int
  (in integer)
  (conclusion THM (semigroup int-struct)))

;(th `deftheorem semigroup-times-int
  (in integer)
  (conclusion THM (semigroup int-struct)))

(th `deftheorem neutral-plus-int
  (in integer)
  (conclusion THM (sunit int plus zero)))

(th `deftheorem neutral-times-int
  (in integer)
  (conclusion THM (sunit int times (s zero))))

;(th `deftheorem monoid-plus-int
  (in integer)
  (conclusion THM (smonoid int-struct)))

;(th `deftheorem monoid-times-int
  (in integer)
  (conclusion THM (smonoid int-mult-struct)))

;(th `deftheorem inverse-plus-int
  (in integer)
  (conclusion THM (inverse-in nat plus zero change-sign)))

(th `deftheorem group-plus-int
  (in integer)
  (conclusion THM (group nat-plus-struct)))

(th `deftheorem neg-zero
  (in integer)
  (conclusion (s (change-sign zero) zero)
    (help "Zero is the negative of zero.\n"))

(th `deftheorem neg-zero
  (in integer)
  (status global)
  (equation neg-zero)
  (direction lr)
  (help "Simplify \(-\) 0 to 0.\n")

(th `deftheorem neg-silpotent
  (in integer)
  (conclusion (silpotent int change-sign))
  (help "The function for changing signs of integers is silpotent.\n")

(th `deftheorem plus-int-base
  (in integer)
  (conclusion
    (forall-sort (lam (x sum)
      (= (plus x zero) x))
      int))
  (help "The base case for recursive definition of addition.\n")

(th `deftheorem plus-int-base2
  (in integer)
  (conclusion
    (forall-sort (lam (x sum)
      (= (plus zero x) x))
      int))
  (help "The base case for recursive definition of addition.\n")

83
(help "Another base case for recursive definition of addition.")

(th 'defthm plus-int-step-s
  (in integer)
  (conclusion
   (forall-sort (lam (x y z))
    (forall-sort (lam (y x z))
     (= (plus x (s y)) (s (plus x y)))))
  int))
  (help "The step case for recursive definition of addition.")

(th 'defthm plus-int-step-p
  (in integer)
  (conclusion
   (forall-sort (lam (x y z))
    (forall-sort (lam (y x z))
     (= (plus x (p y)) (p (plus x y))))
  int))
  (help "Another step case for recursive definition of addition.")

(th 'defthm plus-int-step2-s
  (in integer)
  (conclusion
   (forall-sort (lam (x y z))
    (forall-sort (lam (y x z))
     (= (plus (s x) y) (s (plus y x))))
  int))
  (help "Another step case for recursive definition of addition.")

(th 'defthm plus-int-step2-p
  (in integer)
  (conclusion
   (forall-sort (lam (x y z))
    (forall-sort (lam (y x z))
     (= (plus (p y) x) (p (plus x y))))
  int))
  (help "Another step case for recursive definition of addition.")

(th 'defthm change-sign-base
  (in integer)
  (conclusion
   (= (change-sign zero) zero))
  (help "The base case for recursive definition of unary minus.")

(th 'defthm change-sign-s
  (in integer)
  (conclusion
   (forall-sort (lam (x y))
    (= (change-sign (s x)) (p (change-sign x))))
  int))
  (help "The step case for recursive definition of unary minus.")

(th 'defthm change-sign-p
  (in integer)
  (conclusion
   (forall-sort (lam (x y))
    (= (change-sign (p x)) (s (change-sign x))))
  int))
  (help "Another step case for recursive definition of unary minus.")

(th 'defthm change-sign-reverse
  (in integer)
  (conclusion
   (forall-sort (lam (x y))
    (forall-sort (lam (y x))
     (implies (= (change-sign x) y) (= (change-sign y) x)))
  int))
  (help "Simplification of unary minus.")

(th 'defthm times-int-base
  (in integer)
  (conclusion
   (forall-sort (lam (x y))
    (= (times x zero) zero))
  int))
  (help "Another base case for recursive definition of multiplication.")
(help "The base case for recursive definition of multiplication.")

(th 'deftheorem times-int-base2
  (in integer)
  (conclusion
   (forall-sort (lan (x mn)
       (= (times zero x) zero))
   int))
  (help "Another base case for recursive definition of multiplication.")

(th 'deftheorem times-int-step-s
  (in integer)
  (conclusion
   (forall-sort (lan (x mn)
       (forall-sort (lan (y mn)
         (= (times x (s y)) (plus x (times x y))))
       nat))
   int))
  (help "The step case for recursive definition of multiplication.")

(th 'deftheorem times-int-step-p
  (in integer)
  (conclusion
   (forall-sort (lan (x mn)
       (forall-sort (lan (y mn)
         (= (times x (p y)) (plus (change-sign x) (times x y))))
       nat))
   int))
  (help "Another step case for recursive definition of multiplication.")

(th 'deftheorem times-int-step2-s
  (in integer)
  (conclusion
   (forall-sort (lan (y mn)
       (forall-sort (lan (x mn)
         (= (times (s x) y) (plus y (times x y))))
       nat))
   int))
  (help "Another step case for recursive definition of multiplication.")

(th 'deftheorem times-int-step2-p
  (in integer)
  (conclusion
   (forall-sort (lan (y mn)
       (forall-sort (lan (x mn)
         (= (times (p x) y) (plus (change-sign y) (times x y))))
       nat))
   int))
  (help "Another step case for recursive definition of multiplication.")

(th 'deftheorem div-int
  (in integer)
  (conclusion
   (forall-sort
    (lan (x mn)
     (forall-sort
      (lan (y mn)
       (forall-sort
        (lan (z mn)
         (implies (less zero y)
          (implies (and (leq (times z y) x)
            (greater (times (plus z (times x y z y)) x)
            (= (div x y) z)))))
         nat))
        int))
    int))
  (help "Whole-numbered division.")

(th 'deftheorem mod-int
  (in integer)
  (conclusion
   (forall-sort (lan (x mn)
       (forall-sort (lan (y mn)
         (implies (less zero y)
          (implies (and (mod x y)
            (times x (times y (div x y))))
           nat))
        int))
    int))
  (help "Residue of whole-numbered division.")

85
(th `defthm `power-int-base
  `(in integer)
  (conclusion
   `(forall-sort `(lam (x n)
               (= (power x zero) (s zero))))
   `int)
  `(help "The base case for recursive definition of exponentiation."))

(th `defthm `power-int-step
  `(in integer)
  (conclusion
   `(forall-sort `(lam (x n)
               `(forall-sort `(lam (y n)
                               (= (power x (s y)) (times x (power x y))))
               `nat))
   `int)
  `(help "The step case for recursive definition of exponentiation."))

(th `defthm `gcd-left-arg-zero
  `(in integer)
  (conclusion
   `(forall-sort `(lam (x n)
                   (= (gcd zero x) x))
    `nat))
  `(help "A base case of the Euclidean algorithm."))

(th `defthm `gcd-right-arg-zero
  `(in integer)
  (conclusion
   `(forall-sort `(lam (x n)
                   (= (gcd x zero) x))
    `nat))
  `(help "Another base case of the Euclidean algorithm."))

(th `defthm `gcd-equal-args
  `(in integer)
  (conclusion
   `(forall-sort `(lam (x n)
                   (= (gcd x x) x))
    `nat))
  `(help "Another base case of the Euclidean algorithm."))

(th `defthm `gcd-neg-left-arg
  `(in integer)
  (conclusion
   `(forall-sort `(lam (x n)
                   `(forall-sort `(lam (y n)
                                   (= (gcd x y) (gcd (change-sign x) y)))
                   `int))
    `nat))
  `(help "Greatest common divisor with negative arguments."))

(th `defthm `gcd-neg-right-arg
  `(in integer)
  (conclusion
   `(forall-sort `(lam (x n)
                   `(forall-sort `(lam (y n)
                                   (= (gcd x y) (gcd x (change-sign y))))
                   `nat)))
    `int))
  `(help "Greatest common divisor with negative arguments."))

(th `defthm `gcd-diff-1
  `(in integer)
  (conclusion
   `(forall-sort `(lam (x n)
                   `(forall-sort `(lam (y n)
                                   (= (gcd x y) (gcd (minus x y) y)))
                   `nat)))
    `nat))
  `(help "The step case of the Euclidean algorithm."))

(th `defthm `gcd-diff-2

86
(in integer)
(conclusion
 (forall-sort (lam (x nn)
             (forall-sort (lam (y nn)
                    (= (gcd x y)
                        (gcd x (minus y x)))
                 mat))
         mat)
       )
     )
   )
   (help "Another step case of the Euclidean algorithm.")
)
)
)
)
)
)

(th'deftheorem lcm-left-arg-zero
 (in integer)
(conclusion
 (forall-sort (lam (x nn)
             (= (lcm zero x)
                zero))
         int)
   )
   (help "A base case for the least common multiple.")
)
)
)
)
)
)

(th'deftheorem lcm-right-arg-zero
 (in integer)
(conclusion
 (forall-sort (lam (x nn)
             (= (lcm x zero)
                zero))
         int)
   )
   (help "Another base case for the least common multiple.")
)
)
)
)
)
)

(th'deftheorem lcm-equal-args
 (in integer)
(conclusion
 (forall-sort (lam (x nn)
             (= (lcm x x)
                 x))
         mat)
   )
   (help "Another base case for the least common multiple.")
)
)
)
)
)
)

(th'deftheorem lcm-neg-left-arg
 (in integer)
(conclusion
 (forall-sort (lam (x nn)
             (forall-sort (lam (y nn)
                    (= (lcm x y)
                        (lcm (change-sign x) y)))
                 int))
         mat)
   )
   (help "Least common multiple with negative arguments.")
)
)
)
)
)
)

(th'deftheorem lcm-neg-right-arg
 (in integer)
(conclusion
 (forall-sort (lam (x nn)
             (forall-sort (lam (y nn)
                    (= (lcm x y)
                        (lcm x (change-sign y))))
                 mat))
         int)
   )
   (help "Least common multiple with negative arguments.")
)
)
)
)
)
)

(th'deftheorem lcm-by-gcd
 (in integer)
(conclusion
 (forall-sort (lam (x nn)
             (forall-sort (lam (y nn)
                    (implies (or (less zero x) (less zero y))
                        (or (lcm x y)
                            (div (times x y)
                                (gcd x y))))
                 mat))
         mat)
   )
   (help "The least common multiple by the greatest common divisor.")
)
)
)
)
)
)

(th'deftheorem less-mat-base
 (in integer)
(conclusion
 (forall-sort (lam (x nn)
             (less (p x) zero))
         mat)
   )
   (help "The base case for recursive definition of less.")
)
)
)
)
)
)

87
(th'theorem power-int-closed
  (in integer)
  (conclusion
   (forall-sort (int (x mm))
    (forall-sort (int (y mm))
     (int (power y x)))
    (int x))
  (help "The set of integers is closed under power.")
)

D.4 Theory Natural
D.4.1 natural.thy

(th'theory
  (uses poset function struct)
  (help "Peano Arithmetic for naturals.")
)

(th'deftype
  (in natural)
  (arguments 0)
  (help "The type of number objects, like natural numbers, rationals, reals, complex,...")
)

(th'defconstant zero
  (in natural)
  (type num)
  (help "The zero of natural numbers")
)

(th'defconstant s
  (in natural)
  (type (num num))
  (help "The successor function of the natural numbers")
)

(th'defconstant Nat
  (in natural)
  (type (o mm))
  (sort)
  (help "The set of natural numbers")
)

(th'defconstant Even
  (in natural)
  (type (o mm))
  (sort)
  (help "The set of even natural numbers")
)

(th'defconstant pos-Nat
  (in natural)
  (type (o mm))
  (sort)
  (help "The set of positive natural numbers")
)

(th'defconstant Nat
  (in natural)
  (type (o mm))
  (sort)
  (help "The set of negative natural numbers")
)

(th'defconstant neg-Nat
  (in natural)
  (type (o mm))
  (sort)
  (help "The set of negative numbers")
)

(th'defconstant iterate-varg
  (in natural)
  (type (all-types b (bb bb mm (bb bb num)))))
  (help "An iterator Combinator for the natural numbers")
)

(th'defconstant change-sign
  (in natural)
  (type (num mm))
  (help "The unary minus operator of Integers")
)

(th'defconstant nat-plus-struct
  (in natural)
  (type (struct num))
  (help "The structure of the natural numbers with plus as an operation.")
)
(th’defconstant nat-times-struct
  (in natural)
  (type (struct sum))
  (help "The structure of natural numbers with times as an operation"))

(th’defnate iterate
  (in natural)
  (definition
   (lan (x n))
   ; (that (lan (P (o sum)))
   (and (and (and (P zero))
   (forall-sort (lan (x n) (P (s x))) (P n))
   ( injective P n)
   (forall-sort (lan (x n) (not (= zero (s x)))) (P n))
   (forall (lan (Q (o sum))))
   (implies (and (Q zero)
   (forall-sort (lan (x n)
   (implies (Q n)
   (Q (s n))))
   ( P))))
   (forall-sort (lan (x n) (Q n))
   ( P)))))))
)
)

Peano’s axioms for natural numbers

(th’defnax total-nat
  ;; from: (sort nat)
  (in natural)
  (formula (forall-sort (lan (x n)) (defined (nat x)) defined))
  (termdef)
  (help "The predicate Nat is defined everywhere."))

(th’defntheorem subset-nat-defined
  ;; from sort-defined
  (in natural)
  (conclusion (forall-sort (lan (x n)) (defined x) nat))
  (termdef)
  (help "The predicate Nat is defined everywhere."))

(th’defnax zero-nat
  (in natural)
  (formula
   (nat zero))
  (termdef)
  (help "Zero is a natural number."))

(th’defnax succ-nat
  (in natural)
  (formula
   (forall-sort
   (lan (x n))
   (nat (s x)))
   nat))
  (termdef)
  (help "The successor of a natural number is natural."))

(th’defnax inj-succ
  (in injective nat s))
  (formula
   (forall (injective nat x))
   (help "The successor function is injective."))

(th’defnax no-pred-zero
  (in natural)
  (formula
   (forall-sort
   (lan (x n))
   (not (s (s x) zero))
   (help "Zero has no predecessor."))

(th’defnax induction
  (in natural)
  (formula
   (forall (lan (Q (o sum)))
   (implies (and (subset Q Nat)
   (and (in zero Q)
   (closed-innder-1 Q s)))
   (help "Induction for natural numbers")

90
(th"`defaxion sat-induct
  (in natural)
  (formula
    (forall (lan Q (o sum))
      (implies (and (Q zero)
        (forall-sort (lan (s sum))
          (implies (Q s)
            (Q (s s))))
        (Sat)))
      (forall-sort (lan (s sum) (Q s))
        Sat))))
  (help "The induction axiom for natural numbers.")")

; ; Miscellaneous
;
(th\`defone
  (in natural)
  (definition (s zero))
  (help "The number 1 defined as the successor of 0.")")

\#-struct=N(th\`defsimplifier one-simp
  (in natural)
  (status global)
  (equation one)
  (direction lr)
  (help "Simplify the number one.")")

(th\`deftwo
  (in natural)
  (definition (s one))
  (help "The number 2 defined as the successor of 1.")")

(th\`defthree
  (in natural)
  (definition (s two))
  (help "The number 2 defined as the successor of 1.")")

(th\`deffour
  (in natural)
  (definition (s three))
  (help "The number 2 defined as the successor of 1.")")

(th\`deffive
  (in natural)
  (definition (s four))
  (help "The number 2 defined as the successor of 1.")")

(th\`defsix
  (in natural)
  (definition (s five))
  (help "The number 2 defined as the successor of 1.")")

(th\`defseven
  (in natural)
  (definition (s six))
  (help "The number 2 defined as the successor of 1.")")

(th\`defeight
  (in natural)
  (definition (s seven))
  (help "The number 2 defined as the successor of 1.")")

(th\`defnine
  (in natural)
  (definition (s eight))
  (help "The number 2 defined as the successor of 1.")")

(th\`deften
  (in natural)
  (definition (s nine))
  (help "The number 2 defined as the successor of 1.")")

(th\`defp
(in natural)
definition
(lam (x n) (that (lam (x n)) (= (s n) s)))))
(help "Predecessor function."
)
;
; Definitions of the order relations
;
(th'defeq leq
(in natural)
definition
(lam (n sum)
(lam (n sum)
(forall (lam (Q (o sum)))
(implies (and (in n Q)
(forall (lam (I sum))
(implies (in I Q)
(in (s I) Q)))))
(is n Q)))))
(help "The classical less-or-equal operator on the natural numbers."
)
(th'defeq less
(in natural)
definition
(lam (x n sum)
(lam (y n sum)
(lam (seq x y)
(not (= x y)))))))
(help "The less predicate."
)
(th'defeq greater
(in natural)
definition
(lam (x n sum)
(lam (y n sum)
(lam (eq x y)))
(help "The greater predicate."
)
(th'defeq geq
(in natural)
definition
(lam (x n sum)
(lam (y n sum)
(lam (eq y x))))
(help "The greater-or-equal predicate."
)
;
; Arithmetic operations defined through the recursion operator
;
(th'defeq recursion-poly
(in natural)
type-variables cc)
definition
(lam (G cc) cc sum)
(lam (g cc)
(lam (n sum)
(lam (o cc)
(forall (lam (U o cc sum))
(implies
(and (U zero g)
(forall (lam (y cc)
(forall (lam (x cc) (implies (U x y) (O x y))))))))))
(0 ≤ n)))))
)
(help "A polymorphic version of the recursion operator."
)
(th'defeq recursion
(in natural)
definition
(lam (G ((sum sum) sum))
(lam (g sum)
(lam (n sum)
(lam (o sum)
(forall (lam (U o sum))
(implies
(and (U zero g)

92
(forall (lan (y x sum))
  (forall (lan (x sum))
    (implies
     (and (lan (y x))
          (lan (x sum))
          (lan (y x sum)))
     (forall (n sum) (lan (y x n)))))))

(defun plus
  (in natural)
  (definition
   (recursion (lan (x sum) n))
   (help "Addition defined as iterated application of successor."))

(defun times
  (in natural)
  (definition
   (lan (n sum))
   (recursion (lan (x sum) n))
   (help "Multiplication defined as iterated addition."))

(defun power
  (in natural)
  (definition
   (lan (n sum))
   (recursion (lan (x sum) n))
   (help "Exponentiation defined as iterated multiplication."))

(defun iterate
  (in natural)
  (type-variables bb)
  (definition
   (lan (F (bb bb)))
   (recursion (lan (x sum) n))
   (help "The iteration operator."))

(defun pos-sat
  (in natural)
  (definition
   (lan (x sum))
   (help "The set of positive natural numbers."))

(defun total-pos-sat
  (in natural)
  (definition
   (lan (x sum))
   (help "The predicate Nat is defined everywhere."))

(defun nat
  (in natural)
  (definition
   (lan (x sum))
   (help "The set of positive natural numbers."))

(defun total-nat
  (in natural)
  (definition
   (lan (x sum))
   (help "The predicate Nat is defined everywhere."))

(defun zero-sat
  (in natural)
  (definition
   (lan (x sum))
   (help "Zero is a negative natural number."))

(defun pred-sat
  (in natural)
  (definition
   (lan (x sum))
   (help "The predecessor of a negative natural number is a negative natural number."))

(defun nat-closed
  (in natural)
  (definition
   (lan (x sum))
   (help "The natural numbers are closed under zero, sum, product, and power."))

93
(in natural)
(formula (closed-under-i nzat p))
(help "The set of neg-natural numbers is closed under successors.")

(th\'defaxion nzat-inj-pred
(in natural)
(formula (injective nzat p))
(help "The successor function is injective.")

(th\'defaxion nzat-no-su-zero
(in natural)
(formula
  (forall-sort (lam (x sum)
    (not (s (s x) zero)))
  nzat)
(help "Zero has no successor in NZat.")

(th\'defaxion nzat-induction
(in natural)
(formula
  (forall (lam (q o sum))
    (implies (and (is zero q)
      (closed-under-i q p))
      (subset nzat q))))
(help "The induction axiom for neg-natural numbers.")

(th\'defdef neg-nNat
(in natural)
(definition (lam (x sum) (and (is x NZat) (not (s x zero))))))
(help "The set of negative Nats.")

(th\'defaxion change-sign
(in natural)
(formula
  (and (= (change-sign zero) zero)
    (forall-sort (lam (x sum)
      (s (change-sign (s x)))
    nzat)
    (forall-sort (lam (x sum)
      (s (change-sign (p x)))
    nzat)))
(help "The negative operator on Natural numbers.")

;;;;Ordering properties of the natural numbers

(th\'defaxion leq-nat
(in natural)
(formula
  (and (forall-sort (lam (y sum) (leq zero y))
    nat)
    (forall-sort (lam (x sum)
      (leq (s x) (s y)))
    nat)
    (forall-sort (lam (x sum)
      (leq (s x) (s y)))
    pos-nat)
(help "The classical less-or-equal operator on the natural numbers.")

(th\'defdef first-nats
(in natural)
(definition
  (lam (x sum)
    (lam (y sum)
      (less x y)))
(help "The set of the first n natural numbers, i.e. the set \{0,...,n-1\}.")

(th\'defdef minimality
(in natural)
(type-variables \bb)
(definition
  (lam (G (p \bb))
    (that (lam (x sum)
D.4.2 natural-theorems.thy

(th`defthm times-nat-closed
  (in natural)
  (category theorem)
  (conclusion
    (forall-sort (lam (x n:nat)
      (forall-sort (lam (y n:nat)
        (mat (times x y)))
      mat))
  )
) (termi)
(help "The set of natural numbers is closed under times."))

(th`defthm power-nat-closed
  (in natural)
  (category theorem)
  (conclusion
    (forall-sort (lam (x n:nat)
      (forall-sort (lam (y n:nat)
        (mat (power x y)))
      mat))
  )
) (termi)
(help "The set of natural numbers is closed under power."))

(th`defthm nat-closed-s
  (in natural)
  (conclusion (closed-under-i nat s))
) (help "The set of natural numbers is closed under successors."))

(th`defthm nat-plus-struct-set
  (in natural)
  (conclusion (s (struct-set nat-plus-struct) Nat))
) (help "The set of the nat-plus-struct is nat."))

(th`defthmsimplifier nat-plus-struct-set
  (in natural)
  (status global)
  (equation nat-plus-struct-set)
  (direction lr)
) (help "Simplify the set of nat-plus-struct."))

(th`defthm nat-plus-struct-op
  (in natural)
  (conclusion (s (struct-op nat-plus-struct) plus))
) (help "The operation of nat-plus-struct is plus."))

(th`defthmsimplifier nat-plus-struct-op
  (in natural)
  (status global)
  (equation nat-plus-struct-op)
  (direction lr)
) (help "Simplify the operation of nat-plus-struct."))

(th`defthm nat-plus-struct-ord
  (in natural)
  (conclusion (s (struct-ordering nat-plus-struct) leq))
) (help "The ordering of nat-plus-struct is leq."))

(th`defthmsimplifier nat-plus-struct-ord
  (in natural)
  (status global)
  (equation nat-plus-struct-ord)
  (direction lr)
) (help "Simplify the ordering of nat-plus-struct."))

(th`defthm nat-times-struct-set
  (in natural)
  (conclusion (s (struct-set nat-times-struct) Nat))
) (help "The set of the nat-times-struct is nat."))

(th`defthmsimplifier nat-times-struct-set
  (in natural)
  (status global)
  (equation nat-times-struct-set)
)
(help "Simplify the set of nat-times-struct.")

(th `deftheorems nat-times-struct-op
  (in natural)
  (conclusion (x (struct-op nat-times-struct) times))
  (help "The operation of nat-times-struct is times.")
)

(th `defthesimplifier nat-times-struct-op
  (in natural)
  (status global)
  (equation nat-times-struct-op)
  (direction lr)
  (help "Simplify the operation of nat-times-struct.")
)

(th `deftheorems nat-times-struct-ord
  (in natural)
  (conclusion (x (struct-ordering nat-times-struct) leq))
  (help "The ordering of nat-times-struct is leq.")
)

(th `defthesimplifier nat-times-struct-ord
  (in natural)
  (status global)
  (equation nat-times-struct-ord)
  (direction lr)
  (help "Simplify the ordering of nat-times-struct.")
)

(th `deftheorems recursion-exists
  (in natural)
  (conclusion
    (forall (lan h (mm mm mm mm)))
    (forall (lan g mm))
    (forall (lan (x (recursion h g zero) g))
      (forall-sort (lan (s mm))
        (= (recursion h g (s z))
          (lan (x (recursion h g z))))
        Nat))))
  (help "Existence of the recursion operator.")
)

(th `deftheorems recursion-uniq
  (in natural)
  (category theorem)
  (conclusion
    (all-types cc)
    (forall (lan h (cc cc mm))
      (forall (lan g cc)
        (forall (lan (r1 (cc mm cc (cc cc mm))))
          (forall-sort (lan (r2 (cc mm cc (cc cc mm)))
            (implies (and (lan (x (r1 h g zero) g))
              (forall-sort (lan (s mm))
                (= (r2 h g (s z)) (lan (x (r2 h g z))))
              Nat))
            (lan (s mm))
            (= (r1 h g (s z)) (lan (x (r1 h g z))))
            Nat)))))
      (forall-sort (lan (s mm))
        (forall (lan (x (r1 h g z) (r2 h g z))))
        Nat)))))))
  (help "Uniqueness of the recursion operator.")
)

(th `deftheorems leq-refl
  (in natural)
  (category theorem)
  (conclusion (forall (lan (s mm) (leq x x))))
  (help "Reflexivity of less-equal.")
)
(th 'defthm leq-trans
  (in natural)
  (category theorem)
  (conclusion
    (forall (lam (x y))
      (forall (lam (y z))
        (forall (lam (x z))
          (implies (and (leq x y)
                          (leq y z))
            (leq x z))))))
  (help "Transitivity of less-equal.")
)

(th 'defthm leq-zero-x
  (in natural)
  (category theorem)
  (conclusion
    (forall-sort (lam (x num)) (leq zero x))
  )
  (help 'Zero is less-equal than any natural number.")
)

(th 'defthm leq-x-sx
  (in natural)
  (category theorem)
  (conclusion (forall (lam (x num)) (leq x (s x))))
  (help 'x is less-equal than the successor of x.")
)

;(th 'defthm plus-nat-base
 ; (in natural)
 ; (category theorem)
 ; (conclusion (forall-sort (lam (x num)) (= (plus x zero) x)) sat)
 ; (help "Base case of the recursive definition of plus.")
 ;
 ;(th 'defthm plus-nat-base2
 ; (in natural)
 ; (category theorem)
 ; (conclusion (forall-sort (lam (x num)) (= (plus zero x) x)) sat)
 ; (help "Base case of the recursive definition of plus.")
 ;
 ;(th 'defthm plus-nat-step
 ; (in natural)
 ; (category theorem)
 ; (conclusion
 ;   (forall-sort (lam (x num))
 ;     (forall-sort (lam (y num))
 ;       (= (plus x (s y)) (s (plus x y))) sat)) sat)
 ; (help "Step case of the recursive definition of plus.")
 ;
 ;(th 'defthm c-plus-nat-base
 ; (in natural)
 ; (category theorem)
 ; (conclusion (forall-sort (lam (x num)) (= (plus x zero) (plus zero x))) sat)
 ; (help "Base case of the commutative property of plus for natural numbers.")
 ;
 ;(th 'defthm plus-nat-step2
 ; (in natural)
 ; (category theorem)
 ; (conclusion
 ;   (forall-sort (lam (x num))
 ;     (forall-sort (lam (y num))
 ;       (= (plus (s x) y) (s (plus x y))) sat)) sat)
 ; (help "Another step case of the recursive definition of plus.")
 ;
 ;(th 'defthm c-plus-nat
 ; (in natural)
 ; (category theorem)
 ; (conclusion
 ;   (forall-sort (lam (x num))
 ;     (forall-sort (lam (y num))
 ;       (= (plus x y) (plus y x))) sat)
 ; )
 ; (help "Commutative property of addition.")
 ;)

98
(th'deftheorem plus-sat-closed
  (in natural)
  (category theorem)
  (conclusion
    (forall-sort (lam (x y n))
      (forall-sort (lam (y z n))
        (mat (plus y x))
        mat)
      mat)
  )
  (termiel)
  (help "The set of natural numbers is closed under plus."))

(th'deftheorem a-plus-sat
  (in natural)
  (category theorem)
  (conclusion
    (forall-sort (lam (x y n))
      (forall-sort (lam (z y n))
        (forall-sort (lam (x z n))
          (mat (plus z (plus y x))
           (plus (plus x y) z))
        mat)
      mat)
  )
  (help "Associative property of addition.")

(th'deftheorem plus-sat-base
  (in natural)
  (conclusion
    (forall-sort (lam (x zero))
      (= (plus x zero) x))
  )
  (help "The base case for recursive definition of addition.")

(th'deftheorem plus-sat-base2
  (in natural)
  (conclusion
    (forall-sort (lam (x zero))
      (= (plus zero x) x))
  )
  (help "Another base case for recursive definition of addition.")

(th'deftheorem plus-sat-step
  (in natural)
  (conclusion
    (forall-sort (lam (x y z n))
      (forall-sort (lam (y z n))
        (mat (plus x (plus y z)))
      (plus (plus x y) z))
  )
  (help "The step case for recursive definition of addition.")

(th'deftheorem plus-sat-step2
  (in natural)
  (conclusion
    (forall-sort (lam (x y z n))
      (forall-sort (lam (x y z))
        (mat (plus (plus x y) z)))
  )
  (help "Another step case for recursive definition of addition.")

(th'deftheorem times-sat-base
  (in natural)
  (conclusion
    (forall-sort (lam (x zero))
      (times x zero) zero))
  (help "The base case for recursive definition of multiplication.")

(th'deftheorem times-sat-base2
  (in natural)
  (conclusion
    (forall-sort (lam (x zero))
      (times zero x) zero))
  (help "Another base case for recursive definition of multiplication.")

(th'deftheorem times-sat-step
  (in natural)
(conclusion
  (forall-sort (lam (x m))
    (forall-sort (lam (y m))
      (= (times x (y m)) (plus x (times y m)))
      nat))
  (help "The step case for recursive definition of multiplication."))
)

(th 'deftheorem time-sat-step2
  (in natural)
  (conclusion
    (forall-sort (lam (y m))
      (forall-sort (lam (x m))
        (= (times (s x) y) (plus y (times x y)))
        nat))
    (help "Another step case for recursive definition of multiplication."))
)

(th 'deftheorem power-sat-base
  (in natural)
  (conclusion
    (forall-sort (lam (x m))
      (= (power x zero) (s zero)))
    (help "The base case for recursive definition of exponentiation."))
)

(th 'deftheorem power-sat-step
  (in natural)
  (conclusion
    (forall-sort (lam (y m))
      (forall-sort (lam (x m))
        (= (power x (s y)) (times x (power y)))
        nat))
    (help "The step case for recursive definition of exponentiation."))
)

(th 'deftheorem power-sat-base-zero
  (in natural)
  (conclusion
    (forall-sort (lam (x m))
      (implies (less zero x)
        (= (power zero x) zero)))
    (help "Powers with base zero.")
)

(th 'deftheorem power-sat-base-one
  (in natural)
  (conclusion
    (forall-sort (lam (x m))
      (= (power (s zero) x) (s zero)))
    (help "Powers with base one.")
)

(th 'deftheorem less-sat-base
  (in natural)
  (conclusion
    (forall-sort (lam (x m))
      (less zero (s x)))
    (help "The base case for recursive definition of less.")
)

(th 'deftheorem less-sat-step
  (in natural)
  (conclusion
    (forall-sort (lam (y m))
      (forall-sort (lam (x m))
        (implies (less x y) (less (s x) (s y))))
      nat))
    (help "The step case for recursive definition of less.")
)

(th 'deftheorem less-implies-leq-sat
  (in natural)
  (conclusion
    (forall-sort (lam (x m))
      (forall-sort (lam (y m))
        (implies (less x y) (leq x y)))
      nat))
    (help "The base case for recursive definition of leq.")
)

(th 'deftheorem leq-sat-step
  (in natural)
  (conclusion
    (forall-sort (lam (y m))
      (forall-sort (lam (x m))
        (implies (leq x y) (leq (s x) (s y))))
      nat))
    (help "The step case for recursive definition of leq.")
)
(th'deftheorem equal-imply-leq-sat
  (in natural)
  (conclusion
    (forall-sort (lam (x sum))
      (forall-sort (lam (y sum))
        (implies (s x y) (leq x y)))
      sat))
  (help "Equal implies less or equal.")
)

(th'deftheorem a-plus-som
  (in natural)
  (category theorem)
  (conclusion
    (forall (lam (x sum))
      (forall (lam (y sum))
        (forall (lam (z sum))
          (implies (plus x (plus y z)) (plus (plus x y) z))))))
  (help "Associative property of addition.")
)

(th'deftheorem e-plus-som
  (in natural)
  (category theorem)
  (conclusion
    (forall (lam (x sum))
      (forall (lam (y sum))
        (forall (lam (z sum))
          (implies (plus x y (plus y z)) (plus (plus x y) z))))))
  (help "Associative property of addition.")
)

;;;Something about even numbers

(th'deftheorem e-som-eat
  (in natural)
  (conclusion
    (forall-sort (lam (x sum)) (nat x)
      (even (plus x y)))
  (termiocl)
  (help "Even numbers are a subset of natural numbers.")
)

(th'deftheorem e-rom-eat
  (in natural)
  (conclusion
    (forall-sort (lam (x sum)) (even (plus x y))
      (nat x))
  (termiocl)
  (help "The sum of two natural numbers is even.")
)

(th'deftheorem e-rom-eat
  (in natural)
  (conclusion
    (forall-sort (lam (y sum))
      (implies (forall-sort (lam (x sum)) (even (plus x y)))
        (even (plus y x))
        (even (plus x y))
        (termiocl)
        (help "The sum of the same two natural numbers is even.")
      (nat x))
  (termiocl)
  (help "The sum of two natural numbers is even.")
)

101