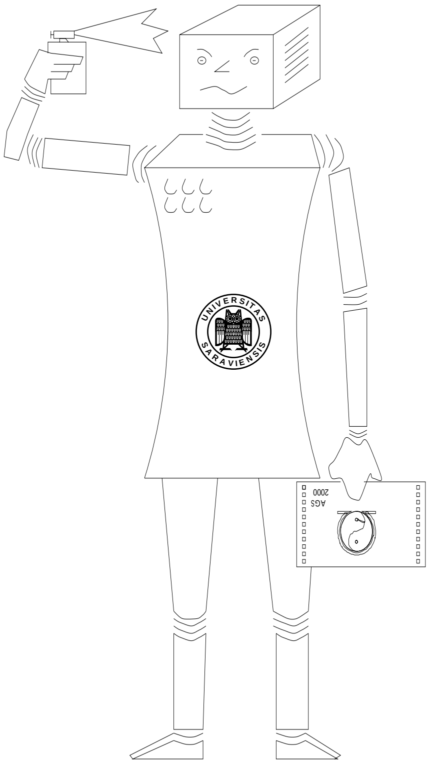
UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D-66041 SAARBRÜCKEN
GERMANY
WWW: `http://www.ags.uni-sb.de/`



SEKI Report

ISSN 1437-4447

Irrationality of $\sqrt{2}$ – A case study in ΩMEGA

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

`{chris,afiedler,ameier,pollet}@ags.uni-sb.de`

FR 6.2 Informatik, Universität des Saarlandes, 66041 Saarbrücken, Germany

SEKI Report SR-02-03

Irrationality of $\sqrt{2}$ – A case study in ΩMEGA

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

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 ΩMEGA 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 ΩMEGA is then described in Section 5 and the interactive proof is given in Section 6. The subsequent sections address the aspects 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 ΩMEGA 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 ΩMEGA

1. ***Where is the home page of the system?***

   The homepage of ΩMEGA 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 ΩMEGA 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 λ-calculus [19]. Higher levels of abstraction are defined in terms of steps at lower levels.

   ΩMEGA'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 ΩMEGA's core calculus. The ND proof can then be verified by ΩMEGA's proof checker.

4. ***What is the implementation architecture of the system?***

   Figure 1 illustrates the basic architecture of ΩMEGA: 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 ΩMEGA 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

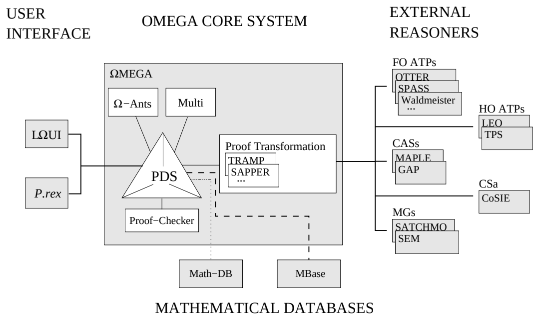


Figure 1: The architecture of the ΩMEGA proof assistant. Thin lines denote internal interfaces, thick lines denote communication via MATHWEB-SB. The thick dashed line indicates that MBASE is soon to be integrated via MATHWEB-SB. It will replace the current mathematical database (thin dotted line).

then accessible by other distant research groups as well. Thus, a very active user community could be established, which has ΩMEGA 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 ΩMEGA is the *proof plan data structure* $\mathcal{PDS}$ [17], in which proofs and *proof plans* are represented at various levels of granularity and abstraction. The proof plans are developed and then classified with respect to a taxonomy of mathematical theories, which is currently being replaced by the mathematical data base MBASE [26, 31]. The users of ΩMEGA, the proof planner MULTI [37], or the suggestion mechanism Ω-ANTS [11] modify the $\mathcal{PDS}$ 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 $\mathcal{PDS}$, 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 ΩMEGA, this is accomplished by transforming the solution into a subproof. CASs' 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 $\mathcal{PDS}$ can be checked by ΩMEGA'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 ΩMEGA is the guidance mechanism provided by the suggestion module Ω-ANTS.

Ω-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?***

ΩMEGA is a mathematical assistant tool that supports proof development in mathematical domains at a user-friendly level of abstraction. It is a modular system with a central data structure and several complementary subsystems. ΩMEGA has many characteristics in common with systems like NUPRL [1], COQ [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 ΩMEGA 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 ΩMEGA 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 ΩMEGA group at Saarland University currently consists of the following researchers (many of them work in research projects related to ΩMEGA and only a few directly on the kernel of the system): Jörg Siekmann, Christoph Benzmüller, Vladimir Brezhnev, Armin Fiedler, Andreas Franke, Helmut Horacek, Andreas Meier, Erica Melis, Markus Moschner, Immanuel Normann, Martin Pollet, Carsten Ullrich, Claus-Peter Wirth, and Jürgen Zimmer.

Former members of the ΩMEGA group, who contributed substantially to the current version, are: Lassaad Cheikhrouhou (now at the German Research Center for Artificial Intellence 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 ΩMEGA 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 ΩMEGA system has been used in several case studies, which illustrate in particular the interplay of the various components, such as proof planning supported by heterogeneous external reasoning systems.

A typical example for a class of problems that cannot be solved by traditional automated theorem provers is the class of ϵ–δ–proofs [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 δ depending on an ϵ). Crucial for the success of ΩMEGA's proof planning is the integration of suitable experts for these tasks: the arithmetic computations are done with the computer algebra system MAPLE, and an appropriate instantiation for δ is computed by the constraint solver $\mathcal{CoSIE}$. We have been able to solve all 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 $(\mathbb{Z}_5, \bar{+})$ is associative, it has a unit element, and other similar properties, where $\mathbb{Z}_5$ is the set of all congruence classes modulo 5 $\{\bar{0}_5, \bar{1}_5, \bar{2}_5, \bar{3}_5, \bar{4}_5\}$ and $\bar{+}$ is the addition on residue classes. We have also investigated whether two given structures are isomorphic or not, and 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 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 MAPLE is applied to check whether the equality actually holds. In this substantial case study, the computer algebra systems MAPLE and GAP are employed to compute witnesses for particular elements, for instance, to compute $\bar{0}_5$, the unit element of $(\mathbb{Z}_5, \bar{+})$.

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 ΩMEGA (see [27]).

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

The problem has been formalized in $\mathcal{POST}$ syntax. $\mathcal{POST}$ stands for ***p***artial functions ***o***rder ***s***orted ***t***ype theory. The formalization employs knowledge provided in ΩMEGAs 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 OTTER proofs, have been translated and integrated into the central ΩMEGA 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 ΩMEGA.

For comparison with the 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 $\mathcal{L}\Omega\mathcal{UI}$ [46], which is by far better suited for the human user of ΩMEGA 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 ΩMEGA, we explicitly keep the original proof plan in an expansion hierarchy. Thus the $\mathcal{PDS}$ makes the hierarchical structure of proof plans explicit and retains it for further applications such as proof explanation with *P.rex* or analogical transfer of 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[1] in ΩMEGA 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 ΩMEGA), 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, ΩMEGA allows for the simultaneous representation of different proofs of the same problem.

The final proof object generated by ΩMEGA in our case study is illustrated in the Appendices A resp. B, where the unexpanded and the expanded proof object are presented in LaTeX resp. $\mathcal{POST}$ format.

[1] To get an overview, type `help` to enter the HELP on OMEGA interpreter and then `commands rules` (to exit HELP type `exit`).

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

$$\sqrt{2} = p/q.$$

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 $\mathcal{POST}$ syntax, which is the logical input language for ΩMEGA.

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

The concepts `rat` and `sqrt` are defined in the knowledge base as well. 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 ι-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."))

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

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 ΩMEGA 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 MAPLE to justify the transformation of $q\sqrt{2} = p$ into $2q^2 = p^2$. To do so, ΩMEGA expressions, such as $\sqrt{2}$, are mapped to corresponding MAPLE representations and MAPLE uses its own built-in knowledge to manipulate them. Using and verifying these computation steps requires expansion of MAPLE's computation to the calculus layer in ΩMEGA. This is done by replaying MAPLE's computation by special computational tactics in ΩMEGA. These tactics and their expansions, which are part of the SAPPER system, correspond directly to the mathematical definitions available in ΩMEGA's knowledge base. The natural number 2 is defined in theory `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."))


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

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


(th~deftheorem 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."))
```

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 ΩMEGA 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."))


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

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

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

(th~defdef recursion
           (in natural)
           (definition
             (lam (h ((num num) num))
                  (lam (g num)
                       (lam (n num)
                            (that
                             (lam (m num)
```

```
                        (forall
                         (lam (U (o num num))
                              (implies (and (U zero g)
                                            (forall
                                             (lam (y num)
                                                  (forall
                                                   (lam (x num)
                                                        (implies (U x y)
                                                                 (U (s x) (h x y))))))))
                                       (U n m))))))))))
(help "The recursion operator."))
```

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 $\mathcal{L}\Omega\mathcal{U}\mathcal{I}$ [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-pds` commands.

We present ND proofs in a linearized style. A proof line is of the form '$L \quad (\Delta) \vdash F \quad \mathcal{R}$', where L is a unique label, $(\Delta) \vdash F$ denotes that the formula F can be derived from the formulae whose labels occur in the list Δ, and $\mathcal{R}$ 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.
***

OMEGA: load-problems
THEORY-NAME (EXISTING-THEORY) The name of a theory whose problems ar 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.
```

```
;;; 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.

*** 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

               ...

SQRT2-NOT-RAT ()         ! (NOT (RAT (SQRT 2)))                                            OPEN

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


*** Step: 2
*** We prove the goal indirectly.
OMEGA: noti
NEGATION (NDLINE) A negated line: [SQRT2-NOT-RAT]
FALSITY (NDLINE) A falsity line: [()]

OMEGA: show-pds


L1            (L1)       ! (RAT (SQRT 2))                                                   HYP
               ...

L2            (L1)       ! FALSE                                                            OPEN

SQRT2-NOT-RAT ()         ! (NOT (RAT (SQRT 2)))                                     NOTI: (L2)


*** 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


L1            (L1)       ! (RAT (SQRT 2))                                                   HYP
               ...

L2            (L1)       ! FALSE                                                            OPEN

RAT-CRITERION (RAT-CRITERION) ! (FORALL-SORT                                                 THM
                                 ([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)

SQRT2-NOT-RAT ()         ! (NOT (RAT (SQRT 2)))                             NOTI: (L2)


*** 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: foralle-sort
UNIV-LINE (NDLINE) Universal line: [RAT-CRITERION]
LINE (NDLINE) A line: [()]
TERM (TERM) Term to substitute: (sqrt 2)
SO-LINE (NDLINE) A line with sort: [L1]
;;;CSM Arbitrary [2]: 0 provers have to be killed

OMEGA: show-line* (rat-criterion l3)


RAT-CRITERION (RAT-CRITERION) ! (FORALL-SORT                                   THM
                                ([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                                    FORALLE-SORT:
                         ([DC-4248].                                         ((SQRT 2))
                          (EXISTS-SORT                                (RAT-CRITERION L1)
                           ([DC-4251].
                            (AND (= (TIMES (SQRT 2) DC-4248)
                                    DC-4251)
                                 (NOT (EXISTS-SORT ([DC-4255].
                                                    (COMMON-DIVISOR DC-4248 DC-4251 DC-4255))
                                                   INT))))
                           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: existse-sort
EX-LINE (NDLINE) An existential line: [L3]
LINE (NDLINE) A line to be proved: [L2]
PARAM (TERMSYM) A term: [dc-42481]n
PREM (NDLINE) The second premise line: [()]

OMEGA: show-line* (l2 l3 l4 l5)


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



L3         (L1)       ! (EXISTS-SORT                                    FORALLE-SORT:
```

```
                         ([DC-4248].                                        ((SQRT 2))
                          (EXISTS-SORT                              (RAT-CRITERION L1)
                           ([DC-4251].
                            (AND (= (TIMES (SQRT 2) DC-4248)
                                    DC-4251)
                                 (NOT (EXISTS-SORT ([DC-4255].
                                                   (COMMON-DIVISOR DC-4248 DC-4251 DC-4255))
                                                  INT))))
                           INT))
                         INT)


L4         (L4)       ! (AND (INT N)                                                   HYP
                             (EXISTS-SORT
                              ([DC-4260].
                               (AND (= (TIMES (SQRT 2) N) DC-4260)
                                    (NOT (EXISTS-SORT ([DC-4264].
                                                      (COMMON-DIVISOR N DC-4260 DC-4264))
                                                     INT))))
                              INT))


L5         (L4 L1)    ! FALSE                                                          OPEN

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

OMEGA: show-line* (l6 l7)


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


L7         (L4)       ! (EXISTS-SORT                                            ANDE: (L4)
                         ([DC-4260].
                          (AND (= (TIMES (SQRT 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 conjnction in line L8.
OMEGA: existse-sort
EX-LINE (NDLINE) An existential line: [L3]l7
LINE (NDLINE) A line to be proved: [L5]
PARAM (TERMSYM) A term: [dc-42601]m
PREM (NDLINE) The second premise line: [()]

OMEGA: show-line* (l7 l5 l8 l9)


L7         (L4)       ! (EXISTS-SORT                                            ANDE: (L4)
                         ([DC-4260].
                          (AND (= (TIMES (SQRT 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 (SQRT 2) N) M)
                                   (NOT (EXISTS-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* (l10 l11 l12)


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


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


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


*** Step: 9
*** We want to infer from (= (TIMES (SQRT 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 (NDPLANLINE) An open node: [L9]
FORMULA (FORMULA) Formula to be proved as lemma: (= (power m 2) (times 2
(power n 2)))

OMEGA: show-line* (l13)


L13         (L8 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 MathWeb, 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.: l13
LINE2 (NDLINE-LIST) A list containing premises to be used.: (l11)

OMEGA: show-line* (l11 l13)


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

```
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 (NDPLANLINE) An open node: [L9]
FORMULA (FORMULA) Formula to be proved as lemma: (evenp (power m 2))

OMEGA: show-line* (l14)


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: defn-contract
LINE (NDLINE) Line to be rewritten: [L14]
DEFINITION (THY-ASSUMPTION) Definition to be contracted: [EVENP]
POS (POSITION) Position of occurrence: [(0)]

OMEGA: show-line* (l15 l14)



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



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 (NDPLANLINE) An open node: [L15]l9
FORMULA (FORMULA) Formula to be proved as lemma: (int (power n 2))

OMEGA: show-line* (l16)


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 (NDLINE) A line with sort: [L16]
PREMISES (NDLINE-LIST) A list of premises: [(L10 L1 L6)]
```

```
OMEGA: show-line* (l16)


L16       (L8 L4 L1) ! (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)
                                                     ((S ZERO) NAT SUCC-NAT)
                                                     (ZERO NAT ZERO-NAT)))
                                                                                          (L6)


*** Step: 15
*** Now we can complete this part of the proof and successfully connect
*** L15 and L13.
OMEGA: existsi-sort
EX-LINE (NDLINE) Existential line to prove: [L15]
PARAM (TERM) Witness term: (power n 2)
LINE (NDLINE) A line: [()]l13
SO-LINE (NDLINE) A line with sort: [L16]
POS-LIST (POSITION-LIST) The position(s) of the witness term: [((2 2))]

OMEGA: show-line* (l13 l15 l16)


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


L15       (L8 L4 L1) ! (EXISTS-SORT                                             EXISTSI-SORT:
                         ([DC-4278]. (= (POWER M 2) (TIMES 2 DC-4278)))     ((POWER N 2) ((2 2)))
                         INT)                                                         (L13 L16)


L16       (L8 L4 L1) ! (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)
                                                     ((S ZERO) NAT SUCC-NAT)
                                                     (ZERO NAT ZERO-NAT)))
                                                                                          (L6)


*** 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 M) 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
theory: square-even

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


L14       (L8 L4 L1) ! (EVENP (POWER M 2))                                          DefnI:
                              (EVENP ([X]. (EXISTS-SORT ([Y]. (= X (TIMES 2 Y))) INT)) (0))
                                                                                         (L15)


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


*** 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 OTTER is
*** called.
OMEGA: assert
FORMULA (TERM) A formula: [false](evenp m)
PROOF-LINES (NDLINE-LIST) Depends on proof lines: [(SQUARE-EVEN RAT-CRITERION)]
(square-even l10 l14)
DEFIS (THY-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.


 OTTER HAS FOUND A PROOF

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



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


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



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



L14        (L8 L4 L1) ! (EVENP (POWER M 2))                                                DefnI:
                                      (EVENP ([X]. (EXISTS-SORT ([Y]. (= X (TIMES 2 Y))) INT)) (0))
                                                                                          (L15)



L17        (L8 L4 L1) ! (EVENP M)                    ASSERT: ((EVENP M) NIL) (SQUARE-EVEN L10 L14)


*** Step: 18
*** Next we expand the definition of EVENP in L17.
OMEGA: defn-expand
LINE (NDLINE) Line to be rewritten: [SQUARE-EVEN]l17
DEFINITION (THY-ASSUMPTION) Definition to be expanded: [EVENP]
POSITION (POSITION) Position of occurrence: [(0)]

OMEGA: show-line* (l17 l18)


L17        (L8 L4 L1) ! (EVENP M)                    ASSERT: ((EVENP M) NIL) (SQUARE-EVEN L10 L14)



L18        (L8 L4 L1) ! (EXISTS-SORT ([DC-4334].(= M (TIMES 2 DC-4334))) INT)
                                                                                         DefnE:
                                      (EVENP ([X]. (EXISTS-SORT ([Y]. (= X (TIMES 2 Y))) INT)) (0))
```

```
                                                                                       (L17)

*** Step: 19
*** Then we eliminate the (sorted) existential quantifier and introduce
*** a constant k.
OMEGA: existse-sort
EX-LINE (NDLINE) An existential line: [L18]
LINE (NDLINE) A line to be proved: [L9]
PARAM (TERMSYM) A term: [dc-43341]k
PREM (NDLINE) The second premise line: [()]

OMEGA: show-line* (l18 l19 l20)


L18        (L8 L4 L1) ! (EXISTS-SORT ([DC-4334].(= M (TIMES 2 DC-4334))) INT)
                                                                                      DefnE:
                                (EVENP ([X]. (EXISTS-SORT ([Y]. (= X (TIMES 2 Y))) INT)) (0))
                                                                                       (L17)


L19        (L19)      ! (AND (INT K) (= M (TIMES 2 K)))                                  HYP


L20        (L19 L8    ! FALSE                                                           OPEN
            L4 L1)


*** Step: 20
*** We immediately split the obtained conjunction in line L19.
OMEGA: ande
CONJUNCTION (NDLINE) Conjunction to split: [L4]l19
LCONJ (NDLINE) Left conjunct: [()]
RCONJ (NDLINE) Right conjunct: [()]

OMEGA: show-line* (l19 l21 l22)


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 (= (POWER M 2) (TIMES 2 (POWER N 2))) in L13 we now want
*** to infer that (= (POWER N 2) (TIMES 2 (POWER K 2))) holds.
OMEGA: lemma
NODE (NDPLANLINE) An open node: [L20]
FORMULA (FORMULA) Formula to be proved as lemma: (= (power n 2)
(times 2 (power k 2)))

OMEGA: show-line* (l23)


L23        (L19 L8    ! (= (POWER N 2) (TIMES 2 (POWER K 2)))                           OPEN
            L4 L1)


*** Step: 22
```

```
*** This can be done again by calling a computer algebra system.
OMEGA: by-computation
LINE1 (NDLINE) A line an arithmetic term to justify.: l23
LINE2 (NDLINE-LIST) A list containing premises to be used.: (l13 l22)

OMEGA: show-line* (l13 l22 l23)


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



L22       (L19)      ! (= M (TIMES 2 K))                                            ANDE: (L19)



L23       (L19 L8    ! (= (POWER N 2) (TIMES 2 (POWER K 2)))            BY-COMPUTATION: (L13 L22)
           L4 L1)


*** 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.
OMEGA: lemma
NODE (NDPLANLINE) An open node: [L20]
FORMULA (FORMULA) Formula to be proved as lemma: (evenp (power n 2))

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

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


*** Step: 26
OMEGA: wellsorted
LINE (NDLINE) A line with sort: [L26]
PREMISES (NDLINE-LIST) A list of premises: [(L16 L6 L1 L10 L21)](l21)

*** Step: 27
OMEGA: existsi-sort
EX-LINE (NDLINE) Existential line to prove: [L25]
PARAM (TERM) Witness term: (power k 2)
LINE (NDLINE) A line: [()]l23
SO-LINE (NDLINE) A line with sort: [L26]
POS-LIST (POSITION-LIST) The position(s) of the witness term: [((2 2))]

OMEGA: show-line* (l21 l24 l25 l26)


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



L24       (L19 L8    ! (EVENP (POWER N 2))                                                DefnI:
           L4 L1)                   (EVENP ([X]. (EXISTS-SORT ([Y]. (= X (TIMES 2 Y))) INT)) (0))
                                                                                          (L25)



L25       (L19 L8    ! (EXISTS-SORT                                                 EXISTSI-SORT:
```

```
           L4 L1)       ([DC-4344]. (= (POWER N 2) (TIMES 2 DC-4344)))        ((POWER K 2) ((2 2)))
                        INT)                                                           (L23 L26)


L26        (L19 L8    ! (INT (POWER K 2))                                              WELLSORTED:
           L4 L1)                                   ((((POWER K (S (S ZERO))) INT POWER-INT-CLOSED)
                                                      ((S (S ZERO)) INT NAT-INT)
                                                      ((S (S ZERO)) NAT SUCC-NAT)
                                                      ((S ZERO) NAT SUCC-NAT)
                                                      (ZERO NAT ZERO-NAT)))
                                                                                           (L21)


*** Step: 28
*** Similar to before (steps 16 and 17), where we derived (EVENP M)
*** from (EVENP (POWER M 2)) application of theorem SQUARE-EVEN, we now
*** derive (EVENP N) from (EVENP (POWER N 2)). The assertion is
*** immediately closed by the external ATP OTTER.
OMEGA: assert
FORMULA (TERM) A formula: [false](evenp n)
PROOF-LINES (NDLINE-LIST) Depends on proof lines: [(SQUARE-EVEN RAT-CRITERION)]
(square-even l6 l24)
DEFIS (THY-ASS-LIST) A list of definitions that should be expanded: [(EVENP)]()

 Normalizing ...
LINE: [1] 29808
 Calling otter process 29808 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* (l27)


L27        (L19 L8    ! (EVENP N)                      ASSERT: ((EVENP N) NIL) (SQUARE-EVEN L6 L24)
           L4 L1)


*** Step: 29
*** It remains to be shown that (EVEN N) and (EVEN M) contradict the
*** assumption in line L12 that N and M have now common divisor. For
*** this we first load the EVEN-COMMON-DIVISOR theorem from the
*** database.
OMEGA: import-ass
ASS-NAME (THY-ASSUMPTION) A name of an assumption to be imported from the problem
theory: even-common-divisor

OMEGA: show-line* (even-common-divisor)


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

```
*** Step: 30
*** We also have to ensure that 2 is an integer.
OMEGA: lemma
NODE (NDPLANLINE) An open node: [L20]
FORMULA (FORMULA) Formula to be proved as lemma: (int 2)

OMEGA: show-line* (l28)


L28        (L19 L8    ! (INT 2)                                                        OPEN
            L4 L1)


*** Step: 31
*** L28 can immediately be justified by tactic WELLSORTED.
OMEGA: wellsorted
LINE (NDLINE) A line with sort: [L28]
PREMISES (NDLINE-LIST) A list of premises: [(L16 L6 L1 L10 L21)]()

OMEGA: show-line* (l28)


L28        (L19 L8    ! (INT 2)                                                  WELLSORTED:
            L4 L1)                                             ((((S (S ZERO)) INT NAT-INT)
                                                                 ((S (S ZERO)) NAT SUCC-NAT)
                                                                 ((S ZERO) NAT SUCC-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 (TERM) A formula: [false]
PROOF-LINES (NDLINE-LIST) Depends on proof lines: [(EVEN-COMMON-DIVISOR
SQUARE-EVEN RAT-CRITERION)](even-common-divisor l10 l6 l12 l17 l27 l28)
DEFIS (THY-ASS-LIST) A list of definitions that should be expanded: [()]

 Normalizing ...
LINE: [1] 29890
 Calling otter process 29890 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 L28)
            L4 L1)
```

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

OMEGA: show-line* (l29 l20)


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 occuring in our
*** proof. Thus our proof can for the moment only be verified modulo
*** 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 ...
```

```
 Creating Refutation-Graph ...
 Translating ...
 Translation finished!
Expanding the node L26 ...
Expanding the node L27 ...
 Expanding line L27 justified by OTTER call
THE NODE: (evenp n) #<Justified by OTTER from (L24 L6 L59)>
 Normalizing ...
LINE: [1] 30090
 Calling otter process 30090 with time resource 10sec .
 otter Time Resource in seconds:
 9sec
  -------- 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 L65 ...
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 #:L168> has a correct justification.
Node #<pdsn+node #:L165> has a correct justification.
Node #<pdsn+node #:L162> 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 #:L135> has a correct justification.
Node #<pdsn+node #:L174> has a correct justification.
Node #<pdsn+node #:L173> has a correct justification.
Node #<pdsn+node #:L137> has a correct justification.
Node #<pdsn+node #:L134> has a correct justification.
Node #<pdsn+node #:L180> has a correct justification.
Node #<pdsn+node #:L179> has a correct justification.
Node #<pdsn+node #:L136> has a correct justification.
Node #<pdsn+node #:L19> has a correct justification.
Node #<pdsn+node #:L22> has a correct justification.
Node #<pdsn+node #:L21> has a correct justification.
Node #<pdsn+node #:L8> has a correct justification.
Node #<pdsn+node #:L30> has a correct justification.
Node #<pdsn+node #:L12> has a correct justification.
Node #<pdsn+node #:L11> has a correct justification.
Node #<pdsn+node #:L10> has a correct justification.
Node #<pdsn+node #:L4> has a correct justification.
Node #<pdsn+node #:L7> has a correct justification.
```

```
Node #<pdsn+node #:L6> has a correct justification.
Node #<pdsn+node #:L1> has a correct justification.
Node #<pdsn+node #:L33> has a correct justification.
Node #<pdsn+node #:L2> has a correct justification.
Node #<pdsn+node RAT-CRITERION> has a correct justification.
Node #<pdsn+node #:L36> has a correct justification.
Node #<pdsn+node #:L9> has a correct justification.
Node #<pdsn+node #:L39> has a correct justification.
Node #<pdsn+node #:L5> has a correct justification.
Node #<pdsn+node #:L3> has a correct justification.
;;;WARNING: The application of the wild-tactic #<INFER+WILD-TACTIC BY-COMPUTATION>
is not checked
;;;WARNING: Node #<pdsn+node #:L13> with the formula (= (power m 2)
(times 2 (power n 2))) has an uncorrect or unknown
justification!
Node #<pdsn+node #:L43> has a correct justification.
Node #<pdsn+node #:L42> has a correct justification.
Node #<pdsn+node #:L15> has a correct justification.
Node #<pdsn+node #:L14> has a correct justification.
Node #<pdsn+node #:L16> has a correct justification.
Node #<pdsn+node SQUARE-EVEN> has a correct justification.
Node #<pdsn+node #:L67> has a correct justification.
Node #<pdsn+node #:L66> has a correct justification.
Node #<pdsn+node #:L59> has a correct justification.
Node #<pdsn+node #:L88> has a correct justification.
Node #<pdsn+node #:L89> has a correct justification.
Node #<pdsn+node #:L90> has a correct justification.
Node #<pdsn+node #:L50> has a correct justification.
Node #<pdsn+node #:L92> has a correct justification.
Node #<pdsn+node #:L93> has a correct justification.
Node #<pdsn+node #:L94> has a correct justification.
Node #<pdsn+node #:L20> has a correct justification.
Node #<pdsn+node #:L17> has a correct justification.
Node #<pdsn+node #:L18> has a correct justification.
;;;WARNING: The application of the wild-tactic #<INFER+WILD-TACTIC
BY-COMPUTATION> is not checked
;;;WARNING: Node #<pdsn+node #:L23> with the formula (= (power n 2)
(times 2 (power k 2))) has an uncorrect or unknown
justification!
Node #<pdsn+node #:L26> has a correct justification.
Node #<pdsn+node #:L58> has a correct justification.
Node #<pdsn+node #:L57> has a correct justification.
Node #<pdsn+node #:L25> has a correct justification.
Node #<pdsn+node #:L24> has a correct justification.
Node #<pdsn+node #:L27> has a correct justification.
Node #<pdsn+node #:L115> has a correct justification.
Node #<pdsn+node EVEN-COMMON-DIVISOR> 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 #:L65> has a correct justification.
Node #<pdsn+node #:L130> has a correct justification.
Node #<pdsn+node #:L129> has a correct justification.
Node #<pdsn+node #:L127> has a correct justification.
Node #<pdsn+node #:L111> has a correct justification.
Node #<pdsn+node #:L112> has a correct justification.
Node #<pdsn+node #:L194> has a correct justification.
Node #<pdsn+node #:L193> has a correct justification.
Node #<pdsn+node #:L113> has a correct justification.
Node #<pdsn+node #:L64> has a correct justification.
Node #<pdsn+node #:L121> has a correct justification.
Node #<pdsn+node #:L63> has a correct justification.
Node #<pdsn+node #:L62> has a correct justification.
Node #<pdsn+node #:L61> has a correct justification.
Node #<pdsn+node #:L60> has a correct justification.
Node #<pdsn+node #:L56> has a correct justification.
```

```
Node #<pdsn+node #:L55> has a correct justification.
Node #<pdsn+node #:L54> has a correct justification.
Node #<pdsn+node #:L53> has a correct justification.
Node #<pdsn+node #:L52> has a correct justification.
Node #<pdsn+node #:L51> has a correct justification.
Node #<pdsn+node #:L49> has a correct justification.
Node #<pdsn+node #:L48> has a correct justification.
Node #<pdsn+node #:L47> has a correct justification.
Node #<pdsn+node #:L46> has a correct justification.
Node #<pdsn+node #:L45> has a correct justification.
Node #<pdsn+node #:L44> has a correct justification.
Node #<pdsn+node #:L40> has a correct justification.
Node #<pdsn+node #:L41> has a correct justification.
Node #<pdsn+node #:L37> has a correct justification.
Node #<pdsn+node #:L34> has a correct justification.
Node #<pdsn+node #:L31> has a correct justification.
Node #<pdsn+node #:L28> has a correct justification.
Node #<pdsn+node #:L140> has a correct justification.
Node #<pdsn+node #:L114> has a correct justification.
Node #<pdsn+node #:L29> has a correct justification.
Node #<pdsn+node ZERO-NAT> has a correct justification.
Node #<pdsn+node SUCC-NAT> has a correct justification.
Node #<pdsn+node #:L80> has a correct justification.
Node #<pdsn+node #:L106> has a correct justification.
Node #<pdsn+node #:L79> has a correct justification.
Node #<pdsn+node #:L105> has a correct justification.
Node #<pdsn+node #:L76> has a correct justification.
Node #<pdsn+node #:L100> has a correct justification.
Node #<pdsn+node #:L75> has a correct justification.
Node #<pdsn+node #:L99> has a correct justification.
Node #<pdsn+node #:L73> has a correct justification.
Node #<pdsn+node #:L97> has a correct justification.
Node #<pdsn+node #:L72> has a correct justification.
Node #<pdsn+node #:L96> has a correct justification.
Node #<pdsn+node NAT-INT> has a correct justification.
Node #<pdsn+node #:L81> has a correct justification.
Node #<pdsn+node #:L107> has a correct justification.
Node #<pdsn+node #:L77> has a correct justification.
Node #<pdsn+node #:L101> has a correct justification.
Node #<pdsn+node #:L74> has a correct justification.
Node #<pdsn+node #:L98> has a correct justification.
Node #<pdsn+node POWER-INT-CLOSED> has a correct justification.
Node #<pdsn+node #:L166> has a correct justification.
Node #<pdsn+node #:L164> has a correct justification.
Node #<pdsn+node #:L188> has a correct justification.
Node #<pdsn+node #:L186> has a correct justification.
Node #<pdsn+node #:L161> has a correct justification.
Node #<pdsn+node #:L158> has a correct justification.
Node #<pdsn+node #:L156> has a correct justification.
Node #<pdsn+node #:L160> has a correct justification.
Node #<pdsn+node #:L151> has a correct justification.
Node #<pdsn+node #:L146> has a correct justification.
Node #<pdsn+node #:L142> has a correct justification.
Node #<pdsn+node #:L145> has a correct justification.
Node #<pdsn+node #:L141> has a correct justification.
Node #<pdsn+node #:L139> has a correct justification.
Node #<pdsn+node #:L138> has a correct justification.
Node #<pdsn+node #:L132> has a correct justification.
Node #<pdsn+node #:L128> has a correct justification.
Node #<pdsn+node #:L126> has a correct justification.
Node #<pdsn+node #:L125> has a correct justification.
Node #<pdsn+node #:L124> has a correct justification.
Node #<pdsn+node #:L123> has a correct justification.
Node #<pdsn+node #:L95> has a correct justification.
Node #<pdsn+node #:L91> has a correct justification.
Node #<pdsn+node #:L87> has a correct justification.
Node #<pdsn+node #:L144> has a correct justification.
```

```
Node #<pdsn+node #:L143> has a correct justification.
Node #<pdsn+node #:L84> has a correct justification.
Node #<pdsn+node #:L116> has a correct justification.
Node #<pdsn+node #:L149> has a correct justification.
Node #<pdsn+node #:L148> has a correct justification.
Node #<pdsn+node #:L150> has a correct justification.
Node #<pdsn+node #:L147> has a correct justification.
Node #<pdsn+node #:L117> has a correct justification.
Node #<pdsn+node #:L118> has a correct justification.
Node #<pdsn+node #:L154> has a correct justification.
Node #<pdsn+node #:L153> has a correct justification.
Node #<pdsn+node #:L155> has a correct justification.
Node #<pdsn+node #:L152> has a correct justification.
Node #<pdsn+node #:L119> has a correct justification.
Node #<pdsn+node #:L120> has a correct justification.
Node #<pdsn+node #:L159> has a correct justification.
Node #<pdsn+node #:L157> has a correct justification.
Node #<pdsn+node #:L122> has a correct justification.
Node #<pdsn+node #:L85> has a correct justification.
Node #<pdsn+node #:L86> has a correct justification.
Node #<pdsn+node #:L83> has a correct justification.
Node #<pdsn+node #:L82> has a correct justification.
Node #<pdsn+node #:L108> has a correct justification.
Node #<pdsn+node #:L109> has a correct justification.
Node #<pdsn+node #:L110> has a correct justification.
Node #<pdsn+node #:L78> has a correct justification.
Node #<pdsn+node #:L102> has a correct justification.
Node #<pdsn+node #:L103> has a correct justification.
Node #<pdsn+node #:L104> has a correct justification.
Node #<pdsn+node TERTIUM-NON-DATUR> has a correct justification.
Node #<pdsn+node #:L192> has a correct justification.
Node #<pdsn+node #:L191> has a correct justification.
Node #<pdsn+node #:L190> has a correct justification.
Node #<pdsn+node #:L187> has a correct justification.
Node #<pdsn+node #:L189> has a correct justification.
Node #<pdsn+node #:L185> has a correct justification.
Node #<pdsn+node #:L184> has a correct justification.
Node #<pdsn+node #:L182> has a correct justification.
Node #<pdsn+node #:L195> has a correct justification.
Node #<pdsn+node #:L178> has a correct justification.
Node #<pdsn+node #:L177> has a correct justification.
Node #<pdsn+node #:L181> has a correct justification.
Node #<pdsn+node #:L176> has a correct justification.
Node #<pdsn+node #:L172> has a correct justification.
Node #<pdsn+node #:L171> has a correct justification.
Node #<pdsn+node #:L175> has a correct justification.
Node #<pdsn+node #:L170> has a correct justification.
Node #<pdsn+node SQRT2-NOT-RAT> has a correct justification.

OMEGA:
```

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 $\mathcal{L}\Omega\mathcal{UI}$. $\mathcal{L}\Omega\mathcal{UI}$ 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 $\mathcal{L}\Omega\mathcal{UI}$ presentation of the final proof is given next:

Figure 2 provides a screenshot of $\mathcal{L}\Omega\mathcal{UI}$ after completing but before expanding the proof, that

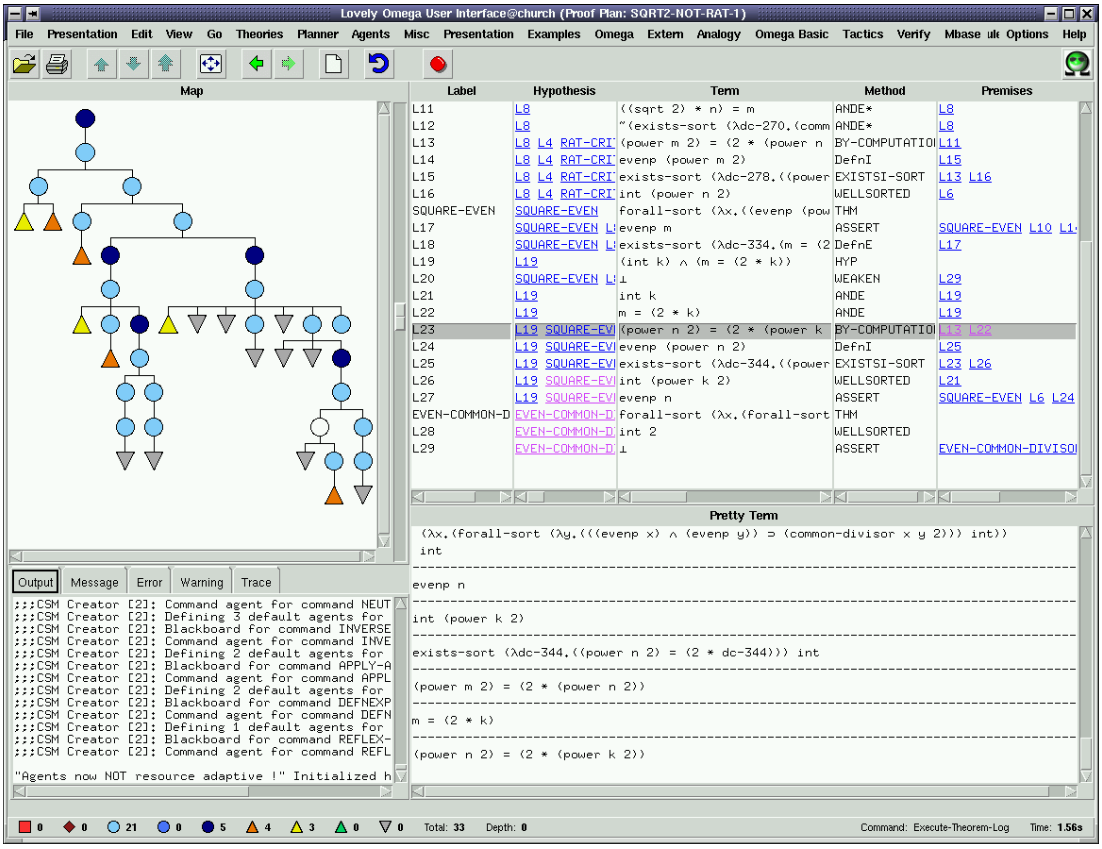


Figure 2: Presentation of the unexpanded proof in the graphical user interface $\mathcal{L}\Omega\mathcal{U}\mathcal{I}$.

is, after Step 33 in Section 6. Figure 3 provides a screenshot of $\mathcal{L}\Omega\mathcal{U}\mathcal{I}$ after expanding the proof, that is, after Step 34 in Section 6.

ΩMEGA 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.rex* [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.rex* anytime, if the current explanation is not satisfactory. *P.rex* 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.rex* presentation of the proof in an emacs interface and Figure 5 its presentation in $\mathcal{L}\Omega\mathcal{U}\mathcal{I}$. 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 $x \in \mathbb{Q}$.

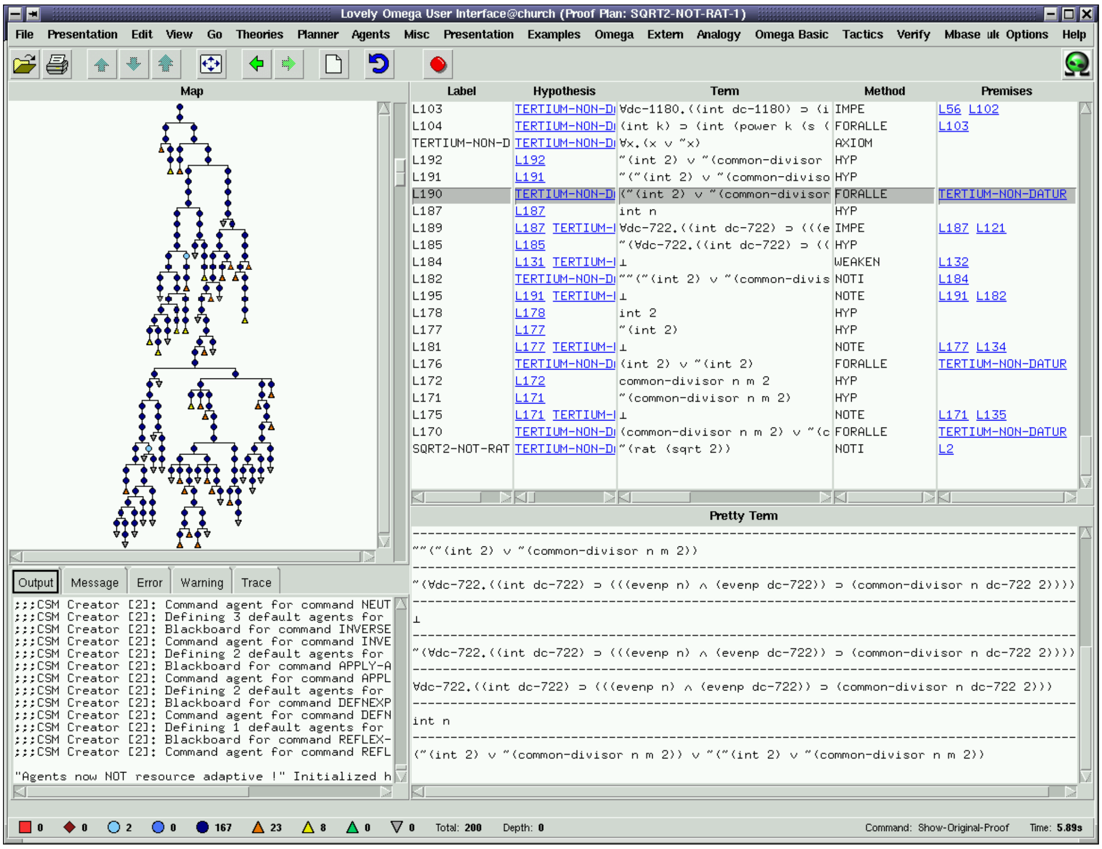


Figure 3: Presentation of the expanded proof in the graphical user interface $\mathcal{L}\Omega\mathcal{UI}$.

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 $dc_{269} \in \mathbb{Z}$ such that $\sqrt{2} \cdot n = dc_{269}$ and there doesn't exist a $dc_{273} \in \mathbb{Z}$ such that dc_{273} is a common divisor of n and dc_{269}.

Let $m \in \mathbb{Z}$, let $\sqrt{2} \cdot n = m$ and let there be no $dc_{279} \in \mathbb{Z}$ such that dc_{279} 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 $dc_{287} \in \mathbb{Z}$ such that $m^2 = 2 \cdot dc_{287}$. $m^2 = 2 \cdot n^2$ since $\sqrt{2} \cdot n = m$.

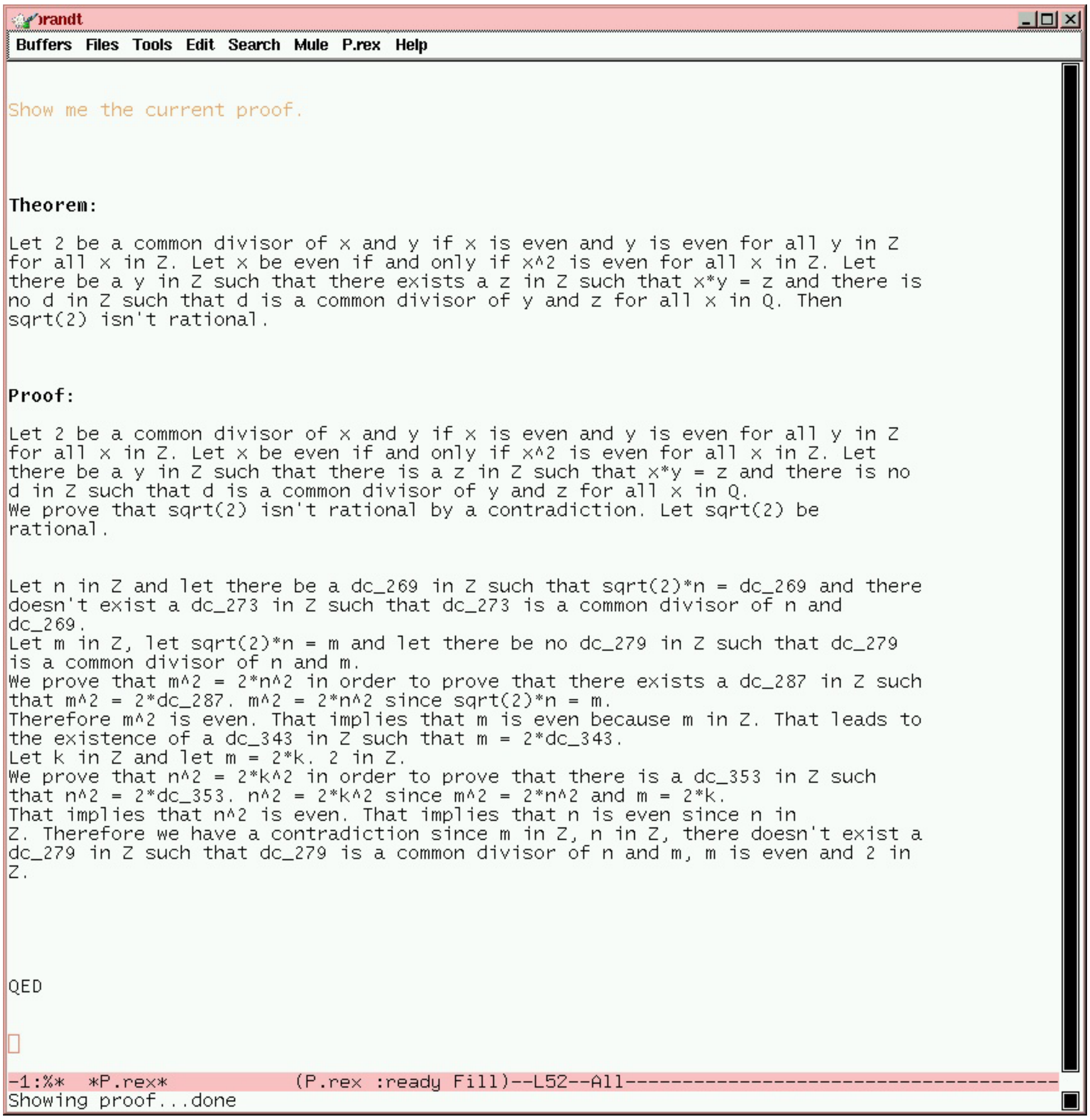
Therefore m^2 is even. That implies that m is even because $m \in \mathbb{Z}$. That leads to the existence of a $dc_{343} \in \mathbb{Z}$ such that $m = 2 \cdot dc_{343}$.

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

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

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 $dc_{279} \in \mathbb{Z}$ such that dc_{279} is a common divisor of n and m, m is even and $2 \in \mathbb{Z}$.

■



```
brandt
Buffers Files Tools Edit Search Mule P.rex Help

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

Let n in Z and let there be a dc_269 in Z such that sqrt(2)*n = dc_269 and there
doesn't exist a dc_273 in Z such that dc_273 is a common divisor of n and
dc_269.
Let m in Z, let sqrt(2)*n = m and let there be no dc_279 in Z such that dc_279
is a common divisor of n and m.
We prove that m^2 = 2*n^2 in order to prove that there exists a dc_287 in Z such
that m^2 = 2*dc_287. m^2 = 2*n^2 since sqrt(2)*n = m.
Therefore m^2 is even. That implies that m is even because m in Z. That leads to
the existence of a dc_343 in Z such that m = 2*dc_343.
Let k in Z and let m = 2*k. 2 in Z.
We prove that n^2 = 2*k^2 in order to prove that there is a dc_353 in Z such
that n^2 = 2*dc_353. n^2 = 2*k^2 since m^2 = 2*n^2 and m = 2*k.
That implies that n^2 is even. That implies that n is even since n in
Z. Therefore we have a contradiction since m in Z, n in Z, there doesn't exist a
dc_279 in Z such that dc_279 is a common divisor of n and m, m is even and 2 in
Z.

QED

-1:%*  *P.rex*        (P.rex :ready Fill)--L52--All-----------------------------
Showing proof...done
```

Figure 4: *P.rex* presentation of the proof in emacs.

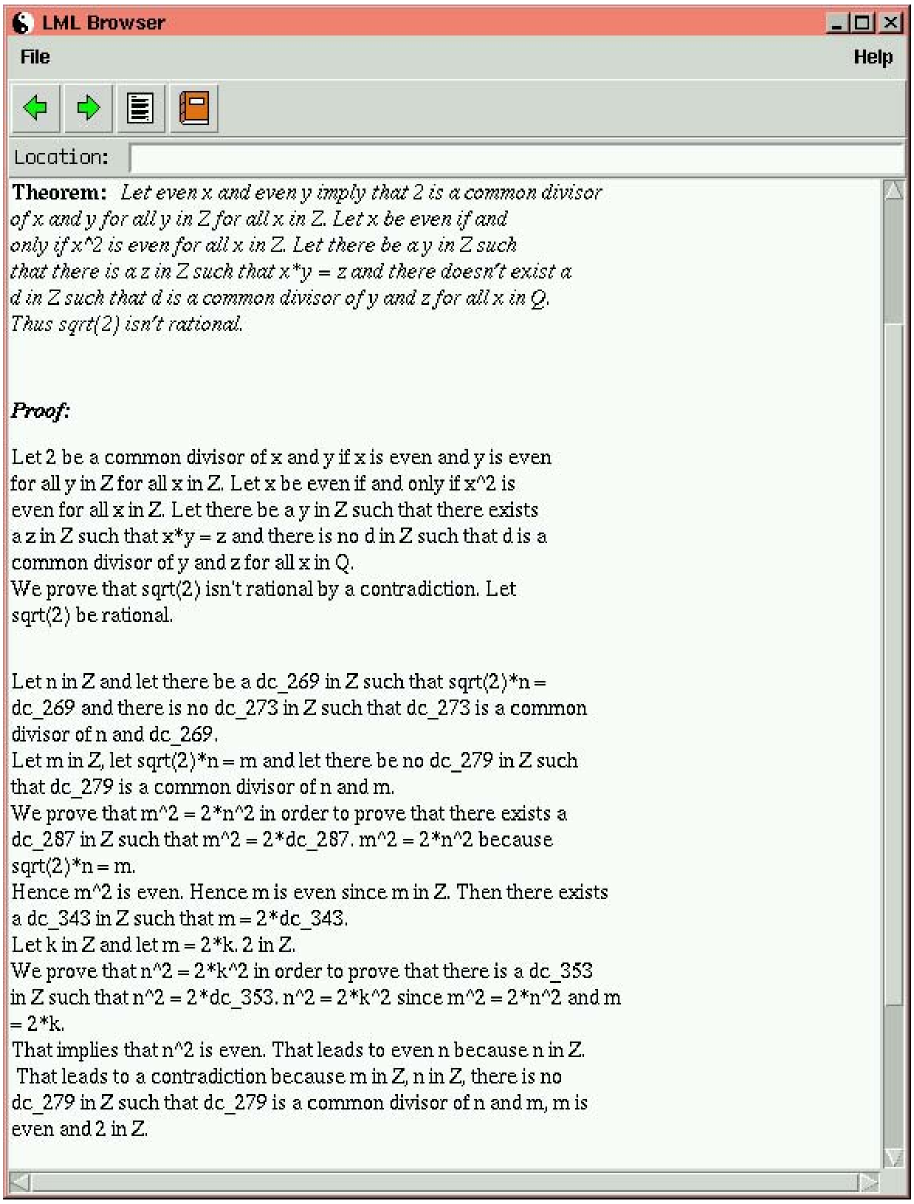


Figure 5: *P.rex* presentation of the proof in $\mathcal{L}\Omega\mathcal{UI}$.

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. ΩMEGA interfaces heterogeneous external systems such as *computer algebra systems (CASs)*, higher- and first-order *automated theorem proving systems (ATPs)*, *constraint solvers (CSs)*, and *model generation systems (MGs)*. Their use is twofold: they may 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 $\mathcal{PDS}$, which maintains the overall proof plan. This is beneficial for interfacing systems that operate at different levels of abstraction, as well as for a human-oriented display and inspection of a partial proof. When integrating partial results, it is important to check the soundness of each contribution. This is accomplished by translating the external solution into a subproof in ΩMEGA, which is then refined to a logic-level proof to be examined by ΩMEGA's proof checker.

The integrated external systems in ΩMEGA 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 ΩMEGA. CASs are integrated via the transformation and translation module SAPPER [47]. Currently, ΩMEGA uses the computer algebra systems MAPLE and GAP.

ATPs are employed to solve subgoals. Currently ΩMEGA uses the first-order automated theorem proving systems BLIKSEM, EQP, OTTER, PROTEIN, SPASS, WALDMEISTER, and the higher-order systems TPS [2, 3], and $\mathcal{LEO}$ [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 ΩMEGA's $\mathcal{PDS}$. 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, ΩMEGA 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, ΩMEGA employs $\mathcal{C}o\mathcal{SIE}$ [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 ΩMEGA, 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 ΩMEGA environment is that resolution proofs generated by ATPs like OTTER can be tranformed via another mathematical service called TRAMP [33] into natural-deduction–style proofs in ΩMEGA. 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 (o i i)))

 (assumption TWONOTONE                 ;; 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 x (mult y z)) (mult (mult x y) z)))))))))

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

 (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 TWOPRIME                  ;; 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 THM
             (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 x a)
                                                         (divides x b))
                                                    (= x one))))))))))
)
```

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

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

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

```
OMEGA: read-problem
"~/omega/tex/omega/sqrt-2-omega/sqrt2-otter-case-study-formalization.post"

;;; 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.
Redefining problem OTTER-CASE-STUDY

Changing to proof plan OTTER-CASE-STUDY-1

OMEGA: show-pds


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

ONE-IDENTITY-LEFT-MULT   (ONE-IDENTITY-LEFT-MULT) ! (FORALL [X:I] (= (MULT ONE X) X))       HYP

ONE-IDENTITY-RIGHT-MULT  (ONE-IDENTITY-RIGHT-MULT) ! (FORALL [X:I] (= (MULT X ONE) X))      HYP

ASSOCIATIVITY-MULT       (ASSOCIATIVITY-MULT) ! (FORALL [X:I,Y:I,Z:I]                       HYP
                                                (= (MULT X (MULT Y Z))
                                                   (MULT (MULT X Y) Z)))

COMMUTATIVITY-M          (COMMUTATIVITY-M) ! (FORALL [X:I,Y:I]                              HYP
                                             (= (MULT X Y) (MULT Y X)))

CANCELLATION             (CANCELLATION) ! (FORALL [X:I,Y:I,Z:I]                             HYP
                                          (IMPLIES (= (MULT X Y) (MULT X Z))
                                                   (= Y Z)))

DIVIDES                  (DIVIDES)  ! (FORALL [X:I,Y:I]                                     HYP
                                      (EQUIV (DIVIDES X Y)
                                             (EXISTS [Z:I] (= (MULT X Z) Y))))

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

THM                      (TWONOTONE ! (NOT (EXISTS [A:I,B:I]                               OPEN
                          ONE-IDENTITY-LEFT-MULT   (AND (= (MULT A A) (MULT TWO (MULT B B)))
                          ONE-IDENTITY-RIGHT-MULT      (FORALL [X:I] (IMPLIES (AND (DIVIDES X A)
                          ASSOCIATIVITY-MULT                                       (DIVIDES X B))
                          COMMUTATIVITY-M                                      (= X ONE))))))
                          CANCELLATION
                          DIVIDES
                          TWOPRIME)


OMEGA: call-otter-on-node
NODE (NDLINE) Node to prove with OTTER: [THM]
DIR (CREATING-DIRECTORY) The (writable!) directory for depositing OTTER auxiliary
files: [/tmp/chris-atp-dir/]
MODE (SYMBOL) Mode for calling OTTER (auto/user/combined): [AUTO]
EXPAND (SYMBOL) A proof found by OTTER is used to (test/parse/expand): [EXPAND]
PROOF-OBJECT (BOOLEAN) Use build_proof_object: [T]
USER-FLAG-STRING (STRING) A string of user flag-settings or a
file-name: [] "set(ur_res).assign(max_distinct_vars,1)."
USER-WEIGHT-STRING (STRING) A string of user weight-settings or a file-name: []
RESSOURCE (INTEGER) A time ressource in seconds (integer).: [10]
SSPU-STYLE (SYMBOL) The SSPU-style (direct/compact/auto): [AUTO]
INDIRECT-PROOF (BOOLEAN) Indirect proof: [()]
```

```
INTEGRAL-FORMULAS (BOOLEAN) Integral formulas: [()]
MAXIMAL-DEPTH (INTEGER) Maximal depth of searching integral-formulas: [2]
THN (BOOLEAN) Prefer tertium non datur case analyses: [T]
AVOID-DOUBELING (BOOLEAN) Avoid doubling: [T]
LEMMAS (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.
 Normalizing ...
 No File found for USER-FLAGS, USER-FLAGS interpreted as direct user input.
 Calling otter process 31851 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 ...
 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:I,B:I]                                         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 C1 C1) (MULT TWO (MULT B B)))
                                         (FORALL [X:I] (IMPLIES (AND (DIVIDES X C1) (DIVIDES X B))
                                                                (= 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]                                      ANDE: (L9)
                                           (IMPLIES (AND (DIVIDES X C1) (DIVIDES X C2))
                                                    (= X ONE)))


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)))           ANDE: (L24)

L28                       (L24)      ! (FORALL [X:I]                                       ANDE: (L24)
                                               (IMPLIES (AND (DIVIDES X C1) (DIVIDES X C3))
                                                        (= X ONE)))

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

L43                       (L43)      ! (= (MULT TWO C5) (MULT C4 C1))                              HYP

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

ONE-IDENTITY-LEFT-MULT  (ONE-IDENTITY-LEFT-MULT) ! (FORALL [X:I] (= (MULT ONE X) X))              HYP

ONE-IDENTITY-RIGHT-MULT (ONE-IDENTITY-RIGHT-MULT) ! (FORALL [X:I]  (= (MULT X ONE) X))            HYP

ASSOCIATIVITY-MULT        (ASSOCIATIVITY-MULT) ! (FORALL [X:I,Y:I,Z:I]                             HYP
                                                  (= (MULT X (MULT Y Z)) (MULT (MULT X Y) Z)))

L18                       (ASSOCIATIVITY-MULT) ! (FORALL [Y:I,Z:I]                      FORALLE: (TWO)
                                                  (=                         (ASSOCIATIVITY-MULT)
                                                   (MULT TWO (MULT Y Z))
                                                   (MULT (MULT TWO Y) Z)))

L34                       (ASSOCIATIVITY-MULT) ! (FORALL [Z:I]                   FORALLE: (C4) (L18)
                                                  (= (MULT TWO (MULT C4 Z))
                                                     (MULT (MULT TWO C4) Z)))

L35                       (ASSOCIATIVITY-MULT) ! (=  (MULT TWO (MULT C4 C1))       FORALLE: (C1) (L34)
                                                (MULT (MULT TWO C4) C1))

L45                       (ASSOCIATIVITY-MULT ! (= (MULT TWO (MULT C4 C1))   =SUBST: ((2 1)) (L35 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))      =SUBST: ((1)) (L46 L27)
                           ASSOCIATIVITY-MULT     (MULT TWO (MULT C4 C1)))
                           L24)

L36                       (ASSOCIATIVITY-MULT) ! (=  (MULT TWO (MULT C4 C4))       FORALLE: (C4) (L34)
                                                (MULT (MULT TWO C4) C4))

L55                       (ASSOCIATIVITY-MULT ! (= (MULT TWO (MULT C4 C4))   =SUBST: ((2 1)) (L36 L31)
                           L31)                   (MULT C1 C4))

COMMUTATIVITY-M           (COMMUTATIVITY-M) ! (FORALL [X:I,Y:I]                                    HYP
                                                (= (MULT X Y) (MULT Y X)))
```

```
L19                  (COMMUTATIVITY-M) ! (FORALL [Y:I]          FORALLE: (C1) (COMMUTATIVITY-M)
                                          (= (MULT C1 Y) (MULT Y C1)))


L33                  (COMMUTATIVITY-M) ! (=  (MULT C1 C4)                  FORALLE: (C4) (L19)
                                             (MULT C4 C1))


CANCELLATION         (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 C4 C1))
                      ASSOCIATIVITY-MULT
                      L24)


L49                  (CANCELLATION ! (= (MULT C4 C1) (MULT C3 C3))                   =SYM: (L48)
                      L31
                      ASSOCIATIVITY-MULT
                      L24)


L53                  (L24       ! (= (MULT TWO C5) (MULT C3 C3))         =SUBST: ((1)) (L49 L43)
                      ASSOCIATIVITY-MULT
                      L31
                      CANCELLATION
                      L43)

DIVIDES              (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 L58)
                      L9)


L20                  (DIVIDES)  ! (FORALL [Y:I]                         FORALLE: (TWO) (DIVIDES)
                                    (EQUIV (DIVIDES TWO Y)
                                           (EXISTS [Z:I] (= (MULT TWO Z) Y))))


L21                  (DIVIDES)  ! (EQUIV (DIVIDES TWO C1)                  FORALLE: (C1) (L20)
                                   (EXISTS [Z:I] (= (MULT TWO Z) C1)))


L22                  (DIVIDES)  ! (IMPLIES (DIVIDES TWO C1)                      EQUIVE: (L21)
                                    (EXISTS [Z:I] (= (MULT TWO Z) C1)))


L23                  (DIVIDES)  ! (IMPLIES (EXISTS [Z:I] (= (MULT TWO Z) C1))    EQUIVE: (L21)
                                    (DIVIDES TWO C1))


L26                  (DIVIDES)  ! (EQUIV (DIVIDES TWO (MULT C3 C3)) FORALLE: ((MULT C3 C3)) (L20)
                                   (EXISTS [Z:I] (= (MULT TWO Z) (MULT C3 C3))))


L37                  (DIVIDES)  ! (EQUIV (DIVIDES TWO (MULT C4 C1)) FORALLE: ((MULT C4 C1)) (L20)
                                   (EXISTS [Z:I] (= (MULT TWO Z) (MULT C4 C1))))
```

```
L39                    (DIVIDES)  ! (IMPLIES (DIVIDES TWO (MULT C4 C1))            EQUIVE: (L37)
                                          (EXISTS [Z:I] (= (MULT TWO Z) (MULT C4 C1))))


L40                    (DIVIDES)  ! (IMPLIES (EXISTS [Z:I] (= (MULT TWO Z)          EQUIVE: (L37)
                                                          (MULT C4 C1)))
                                          (DIVIDES TWO (MULT C4 C1)))


L38                    (DIVIDES)  ! (EQUIV (DIVIDES TWO (MULT C1 C4)) FORALLE: ((MULT C1 C4)) (L20)
                                        (EXISTS [Z:I] (= (MULT TWO Z) (MULT C1 C4))))


L56                    (DIVIDES   ! (DIVIDES TWO (MULT C1 C4))               ASSERTION: (L38 L55)
                        ASSOCIATIVITY-MULT
                        L31)


L57                    (L31       ! (DIVIDES TWO (MULT C4 C1))            =SUBST: ((2)) (L56 L33)
                        ASSOCIATIVITY-MULT
                        DIVIDES
                        COMMUTATIVITY-M)


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


L30                    (L24 L5    ! (EXISTS [Z:I] (= (MULT TWO Z) C1))            IMPE: (L1 L22)
                        L2
                        TWOPRIME
                        DIVIDES
                        CANCELLATION
                        COMMUTATIVITY-M
                        ASSOCIATIVITY-MULT
                        ONE-IDENTITY-RIGHT-MULT
                        ONE-IDENTITY-LEFT-MULT
                        TWONOTONE)


L42                    (L31 L24   ! (EXISTS [Z:I] (= (MULT TWO Z) (MULT C4 C1)))   IMPE: (L57 L39)
                        L5 L2
                        TWOPRIME
                        DIVIDES
                        CANCELLATION
                        COMMUTATIVITY-M
                        ASSOCIATIVITY-MULT
                        ONE-IDENTITY-RIGHT-MULT
                        ONE-IDENTITY-LEFT-MULT
                        TWONOTONE)


L32                    (L31 L24   ! FALSE                                 EXISTSE: (C5) (L42 L54)
                        L5 L2
                        TWOPRIME
                        DIVIDES
                        CANCELLATION
                        COMMUTATIVITY-M
                        ASSOCIATIVITY-MULT
                        ONE-IDENTITY-RIGHT-MULT
                        ONE-IDENTITY-LEFT-MULT
                        TWONOTONE)


L25                    (L24 L5 L2 ! FALSE                                 EXISTSE: (C4) (L30 L32)
                        TWOPRIME
```

```
                   DIVIDES
                   CANCELLATION
                   COMMUTATIVITY-M
                   ASSOCIATIVITY-MULT
                   ONE-IDENTITY-RIGHT-MULT
                   ONE-IDENTITY-LEFT-MULT
                   TWONOTONE)

L8                (L5 L2     ! FALSE                               EXISTSE: (C3) (L5 L25)
                   TWOPRIME
                   DIVIDES
                   CANCELLATION
                   COMMUTATIVITY-M
                   ASSOCIATIVITY-MULT
                   ONE-IDENTITY-RIGHT-MULT
                   ONE-IDENTITY-LEFT-MULT
                   TWONOTONE)

L3                (L2        ! FALSE                                EXISTSE: (C1) (L2 L8)
                   TWOPRIME
                   DIVIDES
                   CANCELLATION
                   COMMUTATIVITY-M
                   ASSOCIATIVITY-MULT
                   ONE-IDENTITY-RIGHT-MULT
                   ONE-IDENTITY-LEFT-MULT
                   TWONOTONE)

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

L14               (TWOPRIME) ! (IMPLIES  (DIVIDES TWO (MULT C1 C1))       FORALLE: (C1) (L13)
                                         (OR (DIVIDES TWO C1) (DIVIDES TWO C1)))

L16               (L9 L5 L2  ! (OR (DIVIDES TWO C1) (DIVIDES TWO C1))          IMPE: (L59 L14)
                   DIVIDES
                   TWOPRIME)

L17               (L9 L5 L2  ! (DIVIDES TWO C1)                                   IDEMOR: (L16)
                   DIVIDES
                   TWOPRIME)

L1                (L5 L2     ! (DIVIDES TWO C1)                    EXISTSE: (C2) (L5 L17)
                   DIVIDES
                   TWOPRIME)

L50               (L24 L5 L2 ! (NOT (DIVIDES TWO C3))           ASSERTION: (L28 L1 TWONOTONE)
                   DIVIDES
                   TWOPRIME
                   TWONOTONE)

L51               (L24 L5 L2 ! (NOT (DIVIDES TWO (MULT C3 C3)))  ASSERTION: (TWOPRIME L50 L50)
                   DIVIDES
                   TWOPRIME
                   TWONOTONE)
```

```
L52                   (L24 L5 L2 ! (NOT (= (MULT TWO C5) (MULT C3 C3)))      ASSERTION: (L26 L51)
                       DIVIDES
                       TWOPRIME
                       TWONOTONE)


L54                   (L43       ! FALSE                                          NOTE: (L53 L52)
                       CANCELLATION
                       L31
                       ASSOCIATIVITY-MULT
                       L24 L5 L2
                       DIVIDES
                       TWOPRIME
                       TWONOTONE)


THM                   (TWONOTONE ! (NOT (EXISTS [A:I,B:I]                            NOTI: (L3)
                       ONE-IDENTITY-LEFT-MULT   (AND (= (MULT A A) (MULT TWO (MULT B B)))
                       ONE-IDENTITY-RIGHT-MULT       (FORALL [X:I]
                       ASSOCIATIVITY-MULT                   (IMPLIES (AND (DIVIDES X A)
                       COMMUTATIVITY-M                                    (DIVIDES X B))
                       CANCELLATION                                   (= X ONE))))))
                       DIVIDES
                       TWOPRIME)

OMEGA:
```

10 Discussion

In [53], it was suggested to discuss different proof assistants according to the following criteria, where + and − indicate whether the criteron 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 $\surd$:	+

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 $\mathcal{L}\Omega\mathcal{U}\mathcal{I}$, 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 of why they are not used as widely as, say, computer algebra systems.

So this observation is the crucial issue in the ΩMEGA 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

[1] S. Allen, R. Constable, R. Eaton, C. Kreitz, and L. Lorigo. The Nuprl open logical environment. In McAllester [32].

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

[3] P. B. Andrews, S. Issar, D. Nesmith, and F. Pfenning. The TPS theorem proving system. In Stickel [49].

[4] Serge Autexier. *Theory and Architecture of an Hierarchical Contextual Reasoning Framework*. PhD thesis, Saarland University, Saarbrücken, Germany, forthcoming.

[5] Robert G. Bartle and Donald Sherbert. *Introduction to Real Analysis*. Wiley, 2 edition, 1982.

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

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

[8] C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMEGA: Towards a mathematical assistant. In W. McCune, editor, *Proc. of the 14th Conference on Automated Deduction*, number 1249 in LNAI. Springer, 1997.

[9] C. Benzmüller and V. Sorge. A blackboard architecture for guiding interactive proofs. In *Proceedings of 8th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA'98)*, LNAI, Sozopol, Bulgaria, 1998.

[10] C. Benzmüller and V. Sorge. Critical Agents Supporting Interactive Theorem Proving. In *Proc. of 9th Portuguese Conference on Artificial Intelligence (EPIA-99)*, volume 1695 of *LNAI*. Springer, 1999.

[11] C. Benzmüller and V. Sorge. Ω-ANTS – An open approach at combining Interactive and Automated Theorem Proving. In M. Kerber and M. Kohlhase, editors, *Proc. of the 8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus-2000)*. AK Peters, 2001.

[12] Christoph Benzmüller and Michael Kohlhase. Leo – a higher-order theorem prover. In *Proceedings of the 15th International Conference on Automated Deduction (CADE-15)*, LNAI, LINDAU, Germany, 1998.

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

[14] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam System. In Stickel [49], pages 647 – 648.

[15] Alan Bundy. The use of explicit plans to guide inductive proofs. In Ewing L. Lusk and Ross A. Overbeek, editors, *Proc. of the 9th Conference on Automated Deduction*, number 310 in LNCS, pages 111–120, Argonne, Illinois, USA, 1988. Springer Verlag.

[16] Alan Bundy, editor. *Proc. of the 12th Conference on Automated Deduction*, number 814 in LNAI, Nancy, France, 1994. Springer Verlag.

[17] L. Cheikhrouhou and V. Sorge. $\mathcal{PDS}$ — A Three-Dimensional Data Structure for Proof Plans. In *Proc. of the International Conference on Artificial and Computational Intelligence (ACIDCA'2000)*, 2000.

[18] Lassaad Cheikhrouhou and Jörg H. Siekmann. Planning Diagonalization Proofs. In Fausto Giunchiglia, editor, *Proc. of 8th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA'98)*, pages 167–180, Sozopol, Bulgaria, September 1998. Springer Verlag, Berlin Germany, LNAI 1480.

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

[20] Coq Development Team. *The Coq Proof Assistant Reference Manual*. INRIA. see `http://coq.inria.fr/doc/main.html`.

[21] A. Fiedler. *P.rex*: An interactive proof explainer. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Automated Reasoning — 1st International Joint Conference, IJCAR 2001*, number 2083 in LNAI. Springer, 2001.

[22] Armin Fiedler. Using a cognitive architecture to plan dialogs for the adaptive explanation of proofs. In Thomas Dean, editor, *Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 358–363, Stockholm, Sweden, 1999. Morgan Kaufmann.

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

[24] Armin Fiedler. *User-adaptive proof explanation*. PhD thesis, Naturwissenschaftlich-Technische Fakultät I, Universität des Saarlandes, Saarbrücken, Germany, 2001.

[25] A. Franke and M. Kohlhase. System description: MathWeb, an agent-based communication layer for distributed automated theorem proving. In H. Ganzinger, editor, *Proc. of the 16th Conference on Automated Deduction*, number 1632 in LNAI. Springer, 1999.

[26] A. Franke and M. Kohlhase. System description: MBase, an open mathematical knowledge base. In McAllester [32].

[27] Holger Gebhard. Beweisplanung für die beweise der vollständigkeit verschiedener resolutionskalküle in Ωmega. Master's thesis, Saarlandes University, Saarbrücken, Germany, 1999.

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

[29] X. Huang. Reconstructing Proofs at the Assertion Level. In Bundy [16], pages 738–752.

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

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

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

[33] A. Meier. TRAMP: Transformation of Machine-Found Proofs into Natural Deduction Proofs at the Assertion Level. In McAllester [32].

[34] Andreas Meier, Martin Pollet, and Volker Sorge. Classifying Isomorphic Residue Classes. In R. Moreno-Diaz, B. Buchberger, and J.-L. Freire, editors, *A Selection of Papers from the 8th International Workshop on Computer Aided Systems Theory (EuroCAST 2001)*, volume 2178 of *LNCS*, pages 494 – 508. Springer Verlag, February 19–23 2001.

[35] Andreas Meier, Martin Pollet, and Volker Sorge. Comparing approaches to the exploration of the domain of residue classes. *Journal of Symbolic Computation*, forthcoming.

[36] Andreas Meier and Volker Sorge. Exploring properties of residue classes. In M. Kerber and M. Kohlhase, editors, *8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus-2000)*, 2000.

[37] E. Melis and A. Meier. Proof planning with multiple strategies. In J. Loyd, V. Dahl, U. Furbach, M. Kerber, K. Lau, C. Palamidessi, L.M. Pereira, and Y. Sagivand P. Stuckey, editors, *Proc. of the First International Conference on Computational Logic*, volume 1861 of *LNAI*, pages 644–659. Springer-Verlag, 2000.

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

[39] E. Melis, J. Zimmer, and T. Müller. Integrating constraint solving into proof planning. In Kirchner and Ringeissen [30].

[40] Lenard J. Monk. Inference rules using local contexts. *Journal of Automated Reasoning*, 4:445–462, 1988.

[41] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: Combining specification, proof checking, and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, volume 1102 of *LNCS*, pages 411–414, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[42] Julian D.C. Richardson, Alan Smaill, and Ian M. Green. System description: Proof planning in higher-order logic with λ*clam*. In Claude Kirchner and Hélène Kirchner, editors, *Proc. of the 15th Conference on Automated Deduction*, number 1421 in LNAI. Springer Verlag, 1998.

[43] Peter J. Robinson and John Staples. Formalizing a hierarchical structure of practical mathematical reasoning. *Journal of Logic and Computation*, 3(1):47–61, 1993.

[44] J. Siekmann, C. Benzmüller, V. Brezhnev, L. Cheikhrouhou, A. Fiedler, A. Franke, H. Horacek, M. Kohlhase, A. Meier, E. Melis, M. Moschner, I. Normann, M. Pollet, V. Sorge, C. Ullrich, C.-P. Wirth, and J. Zimmer. Proof development with omega. In Voronkov [51], pages 143–148.

[45] J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, and M. Pollet. Proof development with ΩMEGA: $\sqrt{2}$ is not rational. *Submitted to Journal of Automated Reasoning*, 2002.

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

[47] V. Sorge. Non-Trivial Computations in Proof Planning. In Kirchner and Ringeissen [30].

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

[49] Mark Stickel, editor. *Proc. of the 10th Conference on Automated Deduction*, number 449 in LNCS, Kaiserslautern, Germany, 1990. Springer Verlag.

[50] Geoff Stucliffe, Christian Suttner, and Theodor Yemenis. The TPTP problem library. In Bundy [16].

[51] Andrei Voronkov, editor. *Proc. of the 18th International Conference on Automated Deduction*, number 2392 in LNAI. Springer Verlag, 2002.

[52] Lincoln A. Wallen. *Automated Proof Search in Non-Classical Logics. Efficient Matrix Proof Methods for Modal and Intuitionistic Logics.* MIT Press, Cambridge, Massachusetts;London, England, 1990.

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

[54] J. Zimmer and M. Kohlhase. System description: The mathweb software bus for distributed mathematical reasoning. In Voronkov [51], pages 138–142.

A LaTeX Presentations of the Proof

A.1 The Unexpanded Proof

L19.	L19	$\vdash [\mathbb{Z}_{[\nu\to o]}(K_{[\nu]}) \wedge M_{[\nu]} = (2\cdot_{[(\nu,\nu)\to\nu]} K)]$	(Hyp)
L22.	$\mathcal{H}_0$	$\vdash M = (2\cdot K)$	($\wedge E$ L19)
L21.	$\mathcal{H}_0$	$\vdash \mathbb{Z}(K)$	($\wedge E$ L19)
L8.	L8	$\vdash [\mathbb{Z}(M) \wedge [(\sqrt{{}_{[\nu\to\nu]}2}\cdot N_{[\nu]}) = M \wedge \neg\exists X_{1[\nu]} : \mathbb{Z}_{[\nu\to o]}\blacksquare\text{Common-Divisor}_{[(\nu,\nu,\nu)\to o]}(N, M, X_1)]]$	(Hyp)
L12.	$\mathcal{H}_1$	$\vdash \neg\exists X_{1[\nu]} : \mathbb{Z}\blacksquare\text{Common-Divisor}(N, M, X_1)$	($\wedge E*$ L8)
L11.	$\mathcal{H}_1$	$\vdash (\sqrt{2}\cdot N) = M$	($\wedge E*$ L8)
L10.	$\mathcal{H}_1$	$\vdash \mathbb{Z}(M)$	($\wedge E*$ L8)
L4.	L4	$\vdash [\mathbb{Z}(N) \wedge \exists X_{2[\nu]} : \mathbb{Z}\blacksquare[(\sqrt{2}\cdot N) = X_2 \wedge \neg\exists X_{3[\nu]} : \mathbb{Z}\blacksquare\text{Common-Divisor}(N, X_2, X_3)]]$	(Hyp)
L7.	$\mathcal{H}_2$	$\vdash \exists X_{2[\nu]} : \mathbb{Z}\blacksquare[(\sqrt{2}\cdot N) = X_2 \wedge \neg\exists X_{3[\nu]} : \mathbb{Z}\blacksquare\text{Common-Divisor}(N, X_2, X_3)]$	($\wedge E$ L4)
L6.	$\mathcal{H}_2$	$\vdash \mathbb{Z}(N)$	($\wedge E$ L4)
L1.	L1	$\vdash \mathbb{Q}_{[\nu\to o]}(\sqrt{2})$	(Hyp)
L2.	$\mathcal{H}_3$	$\vdash \bot_{[o]}$	(Existse-Sort L3,L5)
RC.	RC	$\vdash \forall X_{4[\nu]} : \mathbb{Q}_{[\nu\to o]}\blacksquare\exists X_{5[\nu]} : \mathbb{Z}, X_{6[\nu]} : \mathbb{Z}\blacksquare[(X_4\cdot X_5) = X_6 \wedge \neg\exists X_{7[\nu]} : \mathbb{Z}\blacksquare\text{Common-Divisor}(X_5, X_6, X_7)]$	(Thm)
L9.	$\mathcal{H}_4$	$\vdash \bot$	(Existse-Sort L18,L20)
L5.	$\mathcal{H}_5$	$\vdash \bot$	(Existse-Sort L7,L9)
L3.	$\mathcal{H}_3$	$\vdash \exists X_{8[\nu]} : \mathbb{Z}, X_{9[\nu]} : \mathbb{Z}\blacksquare[(\sqrt{2}\cdot X_8) = X_9 \wedge \neg\exists X_{10[\nu]} : \mathbb{Z}\blacksquare\text{Common-Divisor}(X_8, X_9, X_{10})]$	(Foralle-Sort RC,L1)
L13.	$\mathcal{H}_4$	$\vdash (M\hat{}_{[(\nu,\nu)\to\nu]}2) = (2\cdot(N\hat{}2))$	(By-Computation L11)
L15.	$\mathcal{H}_4$	$\vdash \exists X_{11[\nu]} : \mathbb{Z}\blacksquare(M\hat{}2) = (2\cdot X_{11})$	(Existsi-Sort L13,L16)
L14.	$\mathcal{H}_4$	$\vdash \text{Evenp}_{[\nu\to o]}((M\hat{}2))$	(Defni L15)
L16.	$\mathcal{H}_4$	$\vdash \mathbb{Z}((N\hat{}2))$	(Wellsorted L6)
SE.	SE	$\vdash \forall X_{12[\nu]} : \mathbb{Z}\blacksquare[\text{Evenp}((X_{12}\hat{}2)) \Leftrightarrow \text{Evenp}(X_{12})]$	(Thm)
L20.	$\mathcal{H}_6$	$\vdash \bot$	(Weaken L29)
L17.	$\mathcal{H}_4$	$\vdash \text{Evenp}(M)$	(Assert SE,L10,L14)
L18.	$\mathcal{H}_4$	$\vdash \exists X_{13[\nu]} : \mathbb{Z}\blacksquare M = (2\cdot X_{13})$	(Defne L17)
L23.	$\mathcal{H}_6$	$\vdash (N\hat{}2) = (2\cdot(K\hat{}2))$	(By-Computation L13,L22)
L25.	$\mathcal{H}_6$	$\vdash \exists X_{14[\nu]} : \mathbb{Z}\blacksquare(N\hat{}2) = (2\cdot X_{14})$	(Existsi-Sort L23,L26)
L24.	$\mathcal{H}_6$	$\vdash \text{Evenp}((N\hat{}2))$	(Defni L25)
L27.	$\mathcal{H}_6$	$\vdash \text{Evenp}(N)$	(Assert SE,L6,L24)
L26.	$\mathcal{H}_6$	$\vdash \mathbb{Z}((K\hat{}2))$	(Wellsorted L21)
ECD.	ECD	$\vdash \forall X_{15[\nu]} : \mathbb{Z}, X_{16[\nu]} : \mathbb{Z}\blacksquare[[\text{Evenp}(X_{15}) \wedge \text{Evenp}(X_{16})] \Rightarrow \text{Common-Divisor}(X_{15}, X_{16}, 2)]$	(Thm)
L28.	$\mathcal{H}_6$	$\vdash \mathbb{Z}(2)$	(Wellsorted)
L29.	$\mathcal{H}_6$	$\vdash \bot$	(Assert ECD,L10,L6, L12,L17,L27,L28)
S2NR.	$\mathcal{H}_7$	$\vdash \neg\mathbb{Q}(\sqrt{2})$	($\neg I$ L2)

S2NR = Sqrt2-Not-Rat; ECD = Even-Common-Divisor; SE = Square-Even; RC = Rat-Criterion;

$\mathcal{H}_0$ = ECD, L19; $\mathcal{H}_1$ = ECD, SE, L8; $\mathcal{H}_2$ = ECD, SE, L4; $\mathcal{H}_3$ = ECD, RC, SE, L1; $\mathcal{H}_4$ = ECD, RC, SE, L1, L4, L8; $\mathcal{H}_5$ = ECD, RC, SE, L1, L4; $\mathcal{H}_6$ = ECD, RC, SE, L1, L4, L8, L19; $\mathcal{H}_7$ = ECD, RC, SE

A.2 The Expanded Proof

L185.	L185	$\vdash \neg\forall X_{70[\nu]}\blacksquare[\mathbb{Z}_{[\nu\to o]}(X_{70}) \Rightarrow [[\text{Evenp}_{[\nu\to o]}(N_{[\nu]}) \wedge \text{Evenp}(X_{70})] \Rightarrow \text{Common-Divisor}_{[(\nu,\nu,\nu)\to o]}(N, X_{70}, 2)]]$	(Hyp)
ECD.	ECD	$\vdash \forall X_{[\nu]} : \mathbb{Z}_{[\nu\to o]}, Y_{[\nu]} : \mathbb{Z}\blacksquare[[\text{Evenp}(X) \wedge \text{Evenp}(Y)] \Rightarrow \text{Common-Divisor}(X, Y, 2)]$	(Thm)
L87.	ECD, TND	$\vdash \forall X_{434[\nu]} : \mathbb{Z}\forall X_{435[\nu]}\blacksquare[\mathbb{Z}(X_{435}) \Rightarrow [[\text{Evenp}(X_{434}) \wedge \text{Evenp}(X_{435})] \Rightarrow \text{Common-Divisor}(X_{434}, X_{435}, 2)]]$	(Defne ECD)
L71.	$\mathcal{H}_1$	$\vdash \forall X_{940[\nu]}\blacksquare[\mathbb{Z}(X_{940}) \Rightarrow \forall X_{958[\nu]}\blacksquare[\mathbb{Z}(X_{958}) \Rightarrow [[\text{Evenp}(X_{940}) \wedge \text{Evenp}(X_{958})] \Rightarrow \text{Common-Divisor}(X_{940}, X_{958}, 2)]]]$	(Defne L87)
L64.	$\mathcal{H}_1$	$\vdash \forall X_{684[\nu]}\blacksquare[\mathbb{Z}(X_{684}) \Rightarrow \forall X_{70[\nu]}\blacksquare[\mathbb{Z}(X_{70}) \Rightarrow [[\text{Evenp}(X_{684}) \wedge \text{Evenp}(X_{70})] \Rightarrow \text{Common-Divisor}(X_{684}, X_{70}, 2)]]]$	(Lambda L71)

L121.	$\mathcal{H}_1$	$\vdash [\mathbb{Z}(N) \Rightarrow \forall X_{70[\nu]}\blacksquare[\mathbb{Z}(X_{70}) \Rightarrow [[\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(X_{70})] \Rightarrow \mathrm{Common\text{-}Divisor}(N, X_{70}, 2)]]]$	($\forall E$ L64)
L187.	L187	$\vdash \mathbb{Z}(N)$	(Hyp)
L189.	$\mathcal{H}_2$	$\vdash \forall X_{70[\nu]}\blacksquare[\mathbb{Z}(X_{70}) \Rightarrow [[\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(X_{70})] \Rightarrow \mathrm{Common\text{-}Divisor}(N, X_{70}, 2)]]$	($\Rightarrow E$ L187,L121)
L188.	$\mathcal{H}_3$	$\vdash \bot_{[o]}$	($\neg E$ L189,L185)
L186.	$\mathcal{H}_4$	$\vdash \neg\mathbb{Z}(N)$	($\neg I$ L188)
L161.	$\mathcal{H}_1$	$\vdash [\neg\forall X_{70[\nu]}\blacksquare[\mathbb{Z}(X_{70}) \Rightarrow [[\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(X_{70})] \Rightarrow \mathrm{Common\text{-}Divisor}(N, X_{70}, 2)]] \Rightarrow \neg\mathbb{Z}(N)]$	($\Rightarrow I$ L186)
L158.	L158	$\vdash \neg[\mathbb{Z}(X_{7021[\nu]}) \Rightarrow [[\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(X_{7021})] \Rightarrow \mathrm{Common\text{-}Divisor}(N, X_{7021}, 2)]]$	(Hyp)
L156.	L156	$\vdash \forall X_{70[\nu]}\blacksquare[\mathbb{Z}(X_{70}) \Rightarrow [[\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(X_{70})] \Rightarrow \mathrm{Common\text{-}Divisor}(N, X_{70}, 2)]]$	(Hyp)
L160.	TND, L156	$\vdash [\mathbb{Z}(X_{7021}) \Rightarrow [[\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(X_{7021})] \Rightarrow \mathrm{Common\text{-}Divisor}(N, X_{7021}, 2)]]$	($\forall E$ L156)
L159.	$\mathcal{H}_5$	$\vdash \bot$	($\neg E$ L160,L158)
L141.	L141	$\vdash \neg\mathrm{Common\text{-}Divisor}(N, M_{[\nu]}, 2)$	(Hyp)
L144.	$\mathcal{H}_6$	$\vdash \neg\mathrm{Common\text{-}Divisor}(N, M, 2)$	(Weaken L141)
L139.	L139	$\vdash \neg\mathbb{Z}(2)$	(Hyp)
NI.	NI	$\vdash \forall X_{[\nu]} : \mathbb{N}_{[\nu\to o]}\blacksquare\mathbb{Z}(X)$	(Axiom)
L74.	$\mathcal{H}_7$	$\vdash \forall X_{38[\nu]}\blacksquare[\mathbb{N}(X_{38}) \Rightarrow \mathbb{Z}(X_{38})]$	(Defne NI)
L98.	$\mathcal{H}_7$	$\vdash [\mathbb{N}(\mathrm{s}_{[\nu\to\nu]}(\mathrm{s}(0_{[\nu]}))) \Rightarrow \mathbb{Z}(\mathrm{s}(\mathrm{s}(0)))]$	($\forall E$ L74)
SN.	SN	$\vdash \forall X_{[\nu]} : \mathbb{N}\blacksquare\mathbb{N}(\mathrm{s}(X))$	(Axiom)
L73.	$\mathcal{H}_7$	$\vdash \forall X_{401[\nu]}\blacksquare[\mathbb{N}(X_{401}) \Rightarrow \mathbb{N}(\mathrm{s}(X_{401}))]$	(Defne SN)
L97.	$\mathcal{H}_7$	$\vdash [\mathbb{N}(\mathrm{s}(0)) \Rightarrow \mathbb{N}(\mathrm{s}(\mathrm{s}(0)))]$	($\forall E$ L73)
L72.	$\mathcal{H}_7$	$\vdash \forall X_{994[\nu]}\blacksquare[\mathbb{N}(X_{994}) \Rightarrow \mathbb{N}(\mathrm{s}(X_{994}))]$	(Defne SN)
L96.	$\mathcal{H}_7$	$\vdash [\mathbb{N}(0) \Rightarrow \mathbb{N}(\mathrm{s}(0))]$	($\forall E$ L72)
0N.	0N	$\vdash \mathbb{N}(0)$	(Axiom)
L63.	$\mathcal{H}_8$	$\vdash \mathbb{N}(0)$	(Weaken 0N)
L62.	$\mathcal{H}_8$	$\vdash \mathbb{N}(\mathrm{s}(0))$	($\Rightarrow E$ L63,L96)
L61.	$\mathcal{H}_8$	$\vdash \mathbb{N}(\mathrm{s}(\mathrm{s}(0)))$	($\Rightarrow E$ L62,L97)
L60.	$\mathcal{H}_8$	$\vdash \mathbb{Z}(\mathrm{s}(\mathrm{s}(0)))$	($\Rightarrow E$ L61,L98)
L28.	$\mathcal{H}_8$	$\vdash \mathbb{Z}(2)$	(Defni L60)
L140.	$\mathcal{H}_9$	$\vdash \bot$	($\neg E$ L28,L139)
L114.	$\mathcal{H}_8$	$\vdash \neg\neg\mathbb{Z}(2)$	($\neg I$ L140)
L142.	L142	$\vdash \neg\mathbb{Z}(2)$	(Hyp)
L145.	$\mathcal{H}_{10}$	$\vdash \bot$	($\neg E$ L142,L114)
L143.	$\mathcal{H}_{10}$	$\vdash \neg\mathrm{Common\text{-}Divisor}(N, M, 2)$	(Falsee L145)
L8.	L8	$\vdash [\mathbb{Z}(M) \wedge [(\sqrt{}_{[\nu\to\nu]}2\cdot_{[(\nu,\nu)\to\nu]}N) = M \wedge \neg\exists X_{50[\nu]} : \mathbb{Z}\blacksquare\mathrm{Common\text{-}Divisor}(N, M, X_{50})]]$	(Hyp)
L30.	$\mathcal{H}_{11}$	$\vdash [(\sqrt{2}\cdot N) = M \wedge \neg\exists X_{50[\nu]} : \mathbb{Z}\blacksquare\mathrm{Common\text{-}Divisor}(N, M, X_{50})]$	($\wedge E_R$ L8)
L12.	$\mathcal{H}_{12}$	$\vdash \neg\exists X_{50[\nu]} : \mathbb{Z}\blacksquare\mathrm{Common\text{-}Divisor}(N, M, X_{50})$	($\wedge E_R$ L30)
L70.	$\mathcal{H}_{12}$	$\vdash \neg\exists X_{857[\nu]}\blacksquare[\mathbb{Z}(X_{857}) \wedge \mathrm{Common\text{-}Divisor}(N, M, X_{857})]$	(Defne L12)
L65.	$\mathcal{H}_{12}$	$\vdash \neg\exists X_{25[\nu]}\blacksquare[\mathbb{Z}(X_{25}) \wedge \mathrm{Common\text{-}Divisor}(N, M, X_{25})]$	(Lambda L70)
L128.	L128	$\vdash [\mathbb{Z}(X_{251[\nu]}) \wedge \mathrm{Common\text{-}Divisor}(N, M, X_{251})]$	(Hyp)
L130.	$\mathcal{H}_{13}$	$\vdash \exists X_{25[\nu]}\blacksquare[\mathbb{Z}(X_{25}) \wedge \mathrm{Common\text{-}Divisor}(N, M, X_{25})]$	($\exists I$ L128)
L129.	$\mathcal{H}_{13}$	$\vdash \bot$	($\neg E$ L130,L65)
L127.	$\mathcal{H}_{12}$	$\vdash \neg[\mathbb{Z}(X_{251}) \wedge \mathrm{Common\text{-}Divisor}(N, M, X_{251})]$	($\neg I$ L129)
L111.	$\mathcal{H}_{12}$	$\vdash \forall X_{25[\nu]}\blacksquare\neg[\mathbb{Z}(X_{25}) \wedge \mathrm{Common\text{-}Divisor}(N, M, X_{25})]$	($\forall I$ L127)
L112.	$\mathcal{H}_{12}$	$\vdash \neg[\mathbb{Z}(2) \wedge \mathrm{Common\text{-}Divisor}(N, M, 2)]$	($\forall E$ L111)
L131.	L131	$\vdash \neg[\neg\mathbb{Z}(2) \vee \neg\mathrm{Common\text{-}Divisor}(N, M, 2)]$	(Hyp)
L166.	L166	$\vdash \neg\mathrm{Common\text{-}Divisor}(N, M, 2)$	(Hyp)
L169.	$\mathcal{H}_{14}$	$\vdash [\neg\mathbb{Z}(2) \vee \neg\mathrm{Common\text{-}Divisor}(N, M, 2)]$	($\vee I_R$ L166)
L167.	$\mathcal{H}_{14}$	$\vdash \bot$	($\neg E$ L169,L131)
L163.	TND, L131	$\vdash \neg\neg\mathrm{Common\text{-}Divisor}(N, M, 2)$	($\neg I$ L167)
L164.	L164	$\vdash \neg\mathbb{Z}(2)$	(Hyp)
L168.	$\mathcal{H}_{15}$	$\vdash [\neg\mathbb{Z}(2) \vee \neg\mathrm{Common\text{-}Divisor}(N, M, 2)]$	($\vee I_L$ L164)
L165.	$\mathcal{H}_{15}$	$\vdash \bot$	($\neg E$ L168,L131)
L162.	TND, L131	$\vdash \neg\neg\mathbb{Z}(2)$	($\neg I$ L165)
L133.	TND, L131	$\vdash [\neg\neg\mathbb{Z}(2) \wedge \neg\neg\mathrm{Common\text{-}Divisor}(N, M, 2)]$	($\wedge I$ L162,L163)
L135.	TND, L131	$\vdash \neg\neg\mathrm{Common\text{-}Divisor}(N, M, 2)$	($\wedge E_R$ L133)
L171.	L171	$\vdash \neg\mathrm{Common\text{-}Divisor}(N, M, 2)$	(Hyp)
L175.	$\mathcal{H}_{16}$	$\vdash \bot$	($\neg E$ L171,L135)

L174.	$\mathcal{H}_{16}$	$\vdash \text{Common-Divisor}(N, M, 2)$	(Falsee L175)
L172.	L172	$\vdash \text{Common-Divisor}(N, M, 2)$	(Hyp)
L173.	$\mathcal{H}_{17}$	$\vdash \text{Common-Divisor}(N, M, 2)$	(Weaken L172)
TND.	TND	$\vdash \forall X_{[o]\blacksquare}[X \vee \neg X]$	(Axiom)
L170.	TND	$\vdash [\text{Common-Divisor}(N, M, 2) \vee \neg\text{Common-Divisor}(N, M, 2)]$	($\forall E$ TND)
L137.	TND, L131	$\vdash \text{Common-Divisor}(N, M, 2)$	($\vee E$ L170,L173,L174)
L134.	TND, L131	$\vdash \neg\neg\mathbb{Z}(2)$	($\wedge E_L$ L133)
L177.	L177	$\vdash \neg\mathbb{Z}(2)$	(Hyp)
L181.	$\mathcal{H}_{18}$	$\vdash \bot$	($\neg E$ L177,L134)
L180.	$\mathcal{H}_{18}$	$\vdash \mathbb{Z}(2)$	(Falsee L181)
L178.	L178	$\vdash \mathbb{Z}(2)$	(Hyp)
L179.	$\mathcal{H}_{19}$	$\vdash \mathbb{Z}(2)$	(Weaken L178)
L176.	TND	$\vdash [\mathbb{Z}(2) \vee \neg\mathbb{Z}(2)]$	($\forall E$ TND)
L136.	TND, L131	$\vdash \mathbb{Z}(2)$	($\vee E$ L176,L179,L180)
L138.	TND, L131	$\vdash [\mathbb{Z}(2) \wedge \text{Common-Divisor}(N, M, 2)]$	($\wedge I$ L136,L137)
L132.	$\mathcal{H}_{20}$	$\vdash \bot$	($\neg E$ L138,L112)
L184.	$\mathcal{H}_{20}$	$\vdash \bot$	(Weaken L132)
L182.	$\mathcal{H}_{12}$	$\vdash \neg\neg[\neg\mathbb{Z}(2) \vee \neg\text{Common-Divisor}(N, M, 2)]$	($\neg I$ L184)
L191.	L191	$\vdash \neg[\neg\mathbb{Z}(2) \vee \neg\text{Common-Divisor}(N, M, 2)]$	(Hyp)
L195.	$\mathcal{H}_{21}$	$\vdash \bot$	($\neg E$ L191,L182)
L194.	$\mathcal{H}_{21}$	$\vdash [\neg\mathbb{Z}(2) \vee \neg\text{Common-Divisor}(N, M, 2)]$	(Falsee L195)
L192.	L192	$\vdash [\neg\mathbb{Z}(2) \vee \neg\text{Common-Divisor}(N, M, 2)]$	(Hyp)
L193.	$\mathcal{H}_{22}$	$\vdash [\neg\mathbb{Z}(2) \vee \neg\text{Common-Divisor}(N, M, 2)]$	(Weaken L192)
L190.	TND	$\vdash [[\neg\mathbb{Z}(2) \vee \neg\text{Common-Divisor}(N, M, 2)] \vee \neg[\neg\mathbb{Z}(2) \vee \neg\text{Common-Divisor}(N, M, 2)]]$	($\forall E$ TND)
L113.	$\mathcal{H}_{12}$	$\vdash [\neg\mathbb{Z}(2) \vee \neg\text{Common-Divisor}(N, M, 2)]$	($\vee E$ L190,L193,L194)
L84.	$\mathcal{H}_{8}$	$\vdash \neg\text{Common-Divisor}(N, M, 2)$	($\vee E$ L113,L143,L144)
SE.	SE	$\vdash \forall X_{[\nu]} : \mathbb{Z}_\blacksquare[\text{Evenp}((X\hat{}_{[(\nu,\nu)\to\nu]}2)) \Leftrightarrow \text{Evenp}(X)]$	(Thm)
L67.	$\mathcal{H}_{23}$	$\vdash \forall X_{803[\nu]\blacksquare}[\mathbb{Z}(X_{803}) \Rightarrow [\text{Evenp}((X_{803}\hat{}2)) \Leftrightarrow \text{Evenp}(X_{803})]]$	(Defne SE)
L50.	$\mathcal{H}_{23}$	$\vdash \forall X_{10[\nu]\blacksquare}[\mathbb{Z}(X_{10}) \Rightarrow [\text{Evenp}((X_{10}\hat{}2)) \Leftrightarrow \text{Evenp}(X_{10})]]$	(Lambda L67)
L92.	$\mathcal{H}_{23}$	$\vdash [\mathbb{Z}(M) \Rightarrow [\text{Evenp}((M\hat{}2)) \Leftrightarrow \text{Evenp}(M)]]$	($\forall E$ L50)
L10.	$\mathcal{H}_{12}$	$\vdash \mathbb{Z}(M)$	($\wedge E_L$ L8)
L93.	$\mathcal{H}_{12}$	$\vdash [\text{Evenp}((M\hat{}2)) \Leftrightarrow \text{Evenp}(M)]$	($\Rightarrow E$ L10,L92)
L126.	$\mathcal{H}_{12}$	$\vdash [[\text{Evenp}((M\hat{}2)) \Rightarrow \text{Evenp}(M)] \wedge [\text{Evenp}(M) \Rightarrow \text{Evenp}((M\hat{}2))]]$	(Defne L93)
L94.	$\mathcal{H}_{12}$	$\vdash [\text{Evenp}((M\hat{}2)) \Rightarrow \text{Evenp}(M)]$	($\wedge E_L$ L126)
L11.	$\mathcal{H}_{12}$	$\vdash (\sqrt{2}\cdot N) = M$	($\wedge E_L$ L30)
L13.	$\mathcal{H}_{24}$	$\vdash (M\hat{}2) = (2\cdot(N\hat{}2))$	(By-Computation L11)
PIC.	PIC	$\vdash \forall X_{[\nu]} : \mathbb{Z}, Y_{[\nu]} : \mathbb{Z}_\blacksquare\mathbb{Z}((Y\hat{}X))$	(Thm)
L123.	PIC, TND	$\vdash \forall X_{95[\nu]} : \mathbb{Z}\forall X_{66[\nu]\blacksquare}[\mathbb{Z}(X_{66}) \Rightarrow \mathbb{Z}((X_{66}\hat{}X_{95}))]$	(Defne PIC)
L82.	PIC, TND	$\vdash \forall X_{264[\nu]\blacksquare}[\mathbb{Z}(X_{264}) \Rightarrow \forall X_{282[\nu]\blacksquare}[\mathbb{Z}(X_{282}) \Rightarrow \mathbb{Z}((X_{282}\hat{}X_{264}))]]$	(Defne L123)
L108.	PIC, TND	$\vdash [\mathbb{Z}(\text{s}(\text{s}(0))) \Rightarrow \forall X_{282[\nu]\blacksquare}[\mathbb{Z}(X_{282}) \Rightarrow \mathbb{Z}((X_{282}\hat{}\text{s}(\text{s}(0))))]]$	($\forall E$ L82)
L81.	$\mathcal{H}_{25}$	$\vdash \forall X_{229[\nu]\blacksquare}[\mathbb{N}(X_{229}) \Rightarrow \mathbb{Z}(X_{229})]$	(Defne NI)
L107.	$\mathcal{H}_{25}$	$\vdash [\mathbb{N}(\text{s}(\text{s}(0))) \Rightarrow \mathbb{Z}(\text{s}(\text{s}(0)))]$	($\forall E$ L81)
L80.	$\mathcal{H}_{26}$	$\vdash \forall X_{207[\nu]\blacksquare}[\mathbb{N}(X_{207}) \Rightarrow \mathbb{N}(\text{s}(X_{207}))]$	(Defne SN)
L106.	$\mathcal{H}_{26}$	$\vdash [\mathbb{N}(\text{s}(0)) \Rightarrow \mathbb{N}(\text{s}(\text{s}(0)))]$	($\forall E$ L80)
L79.	$\mathcal{H}_{26}$	$\vdash \forall X_{88[\nu]\blacksquare}[\mathbb{N}(X_{88}) \Rightarrow \mathbb{N}(\text{s}(X_{88}))]$	(Defne SN)
L105.	$\mathcal{H}_{26}$	$\vdash [\mathbb{N}(0) \Rightarrow \mathbb{N}(\text{s}(0))]$	($\forall E$ L79)
L49.	$\mathcal{H}_{24}$	$\vdash \mathbb{N}(0)$	(Weaken 0N)
L48.	$\mathcal{H}_{24}$	$\vdash \mathbb{N}(\text{s}(0))$	($\Rightarrow E$ L49,L105)
L47.	$\mathcal{H}_{24}$	$\vdash \mathbb{N}(\text{s}(\text{s}(0)))$	($\Rightarrow E$ L48,L106)
L46.	$\mathcal{H}_{24}$	$\vdash \mathbb{Z}(\text{s}(\text{s}(0)))$	($\Rightarrow E$ L47,L107)
L109.	$\mathcal{H}_{24}$	$\vdash \forall X_{282[\nu]\blacksquare}[\mathbb{Z}(X_{282}) \Rightarrow \mathbb{Z}((X_{282}\hat{}\text{s}(\text{s}(0))))]$	($\Rightarrow E$ L46,L108)
L110.	$\mathcal{H}_{24}$	$\vdash [\mathbb{Z}(N) \Rightarrow \mathbb{Z}((N\hat{}\text{s}(\text{s}(0))))]$	($\forall E$ L109)
L4.	L4	$\vdash [\mathbb{Z}(N) \wedge \exists X_{40[\nu]} : \mathbb{Z}_\blacksquare[(\sqrt{2}\cdot N) = X_{40} \wedge \neg\exists X_{244[\nu]} : \mathbb{Z}_\blacksquare\text{Common-Divisor}(N, X_{40}, X_{244})]]$	(Hyp)
L6.	$\mathcal{H}_{27}$	$\vdash \mathbb{Z}(N)$	($\wedge E_L$ L4)
L45.	$\mathcal{H}_{24}$	$\vdash \mathbb{Z}(N)$	(Weaken L6)
L44.	$\mathcal{H}_{24}$	$\vdash \mathbb{Z}((N\hat{}\text{s}(\text{s}(0))))$	($\Rightarrow E$ L45,L110)
L16.	$\mathcal{H}_{24}$	$\vdash \mathbb{Z}((N\hat{}2))$	(Defni L44)
L43.	$\mathcal{H}_{24}$	$\vdash [\mathbb{Z}((N\hat{}2)) \wedge (M\hat{}2) = (2\cdot(N\hat{}2))]$	($\wedge I$ L16,L13)

L42.	$\mathcal{H}_{24}$	$\vdash \exists X_{591[\nu]}{}_{\blacksquare}[\mathbb{Z}(X_{591}) \wedge (M\hat{}2) = (2{\cdot}X_{591})]$	$(\exists I$ L43)
L15.	$\mathcal{H}_{24}$	$\vdash \exists X_{258[\nu]} : \mathbb{Z}{}_{\blacksquare}(M\hat{}2) = (2{\cdot}X_{258})$	(Defni L42)
L14.	$\mathcal{H}_{24}$	$\vdash \mathrm{Evenp}((M\hat{}2))$	(Defni L15)
L68.	$\mathcal{H}_{24}$	$\vdash \mathrm{Evenp}(M)$	$(\Rightarrow E$ L14,L94)
L17.	$\mathcal{H}_{24}$	$\vdash \mathrm{Evenp}(M)$	(Weaken L68)
L66.	$\mathcal{H}_{23}$	$\vdash \forall X_{69[\nu]}{}_{\blacksquare}[\mathbb{Z}(X_{69}) \Rightarrow [\mathrm{Evenp}((X_{69}\hat{}2)) \Leftrightarrow \mathrm{Evenp}(X_{69})]]$	(Defne SE)
L53.	$\mathcal{H}_{23}$	$\vdash \forall X_{52[\nu]}{}_{\blacksquare}[\mathbb{Z}(X_{52}) \Rightarrow [\mathrm{Evenp}((X_{52}\hat{}2)) \Leftrightarrow \mathrm{Evenp}(X_{52})]]$	(Lambda L66)
L88.	$\mathcal{H}_{23}$	$\vdash [\mathbb{Z}(N) \Rightarrow [\mathrm{Evenp}((N\hat{}2)) \Leftrightarrow \mathrm{Evenp}(N)]]$	$(\forall E$ L53)
L89.	$\mathcal{H}_{27}$	$\vdash [\mathrm{Evenp}((N\hat{}2)) \Leftrightarrow \mathrm{Evenp}(N)]$	$(\Rightarrow E$ L6,L88)
L125.	$\mathcal{H}_{27}$	$\vdash [[\mathrm{Evenp}((N\hat{}2)) \Rightarrow \mathrm{Evenp}(N)] \wedge [\mathrm{Evenp}(N) \Rightarrow \mathrm{Evenp}((N\hat{}2))]]$	(Defne L89)
L90.	$\mathcal{H}_{27}$	$\vdash [\mathrm{Evenp}((N\hat{}2)) \Rightarrow \mathrm{Evenp}(N)]$	$(\wedge E_L$ L125)
L19.	L19	$\vdash [\mathbb{Z}(K_{[\nu]}) \wedge M = (2{\cdot}K)]$	(Hyp)
L22.	$\mathcal{H}_{28}$	$\vdash M = (2{\cdot}K)$	$(\wedge E_R$ L19)
L23.	$\mathcal{H}_{8}$	$\vdash (N\hat{}2) = (2{\cdot}(K\hat{}2))$	(By-Computation L13,L22)
L124.	PIC, TND	$\vdash \forall X_{44[\nu]} : \mathbb{Z}\forall X_{55[\nu]}{}_{\blacksquare}[\mathbb{Z}(X_{55}) \Rightarrow \mathbb{Z}((X_{55}\hat{}X_{44}))]$	(Defne PIC)
L78.	$\mathcal{H}_{7}$	$\vdash \forall X_{42[\nu]}{}_{\blacksquare}[\mathbb{Z}(X_{42}) \Rightarrow \forall X_{60[\nu]}{}_{\blacksquare}[\mathbb{Z}(X_{60}) \Rightarrow \mathbb{Z}((X_{60}\hat{}X_{42}))]]$	(Defne L124)
L102.	$\mathcal{H}_{7}$	$\vdash [\mathbb{Z}(\mathrm{s}(\mathrm{s}(0))) \Rightarrow \forall X_{60[\nu]}{}_{\blacksquare}[\mathbb{Z}(X_{60}) \Rightarrow \mathbb{Z}((X_{60}\hat{}\mathrm{s}(\mathrm{s}(0))))]]$	$(\forall E$ L78)
L77.	$\mathcal{H}_{7}$	$\vdash \forall X_{41[\nu]}{}_{\blacksquare}[\mathbb{N}(X_{41}) \Rightarrow \mathbb{Z}(X_{41})]$	(Defne NI)
L101.	$\mathcal{H}_{7}$	$\vdash [\mathbb{N}(\mathrm{s}(\mathrm{s}(0))) \Rightarrow \mathbb{Z}(\mathrm{s}(\mathrm{s}(0)))]$	$(\forall E$ L77)
L76.	$\mathcal{H}_{7}$	$\vdash \forall X_{85[\nu]}{}_{\blacksquare}[\mathbb{N}(X_{85}) \Rightarrow \mathbb{N}(\mathrm{s}(X_{85}))]$	(Defne SN)
L100.	$\mathcal{H}_{7}$	$\vdash [\mathbb{N}(\mathrm{s}(0)) \Rightarrow \mathbb{N}(\mathrm{s}(\mathrm{s}(0)))]$	$(\forall E$ L76)
L75.	$\mathcal{H}_{7}$	$\vdash \forall X_{63[\nu]}{}_{\blacksquare}[\mathbb{N}(X_{63}) \Rightarrow \mathbb{N}(\mathrm{s}(X_{63}))]$	(Defne SN)
L99.	$\mathcal{H}_{7}$	$\vdash [\mathbb{N}(0) \Rightarrow \mathbb{N}(\mathrm{s}(0))]$	$(\forall E$ L75)
L59.	$\mathcal{H}_{8}$	$\vdash \mathbb{N}(0)$	(Weaken 0N)
L58.	$\mathcal{H}_{8}$	$\vdash \mathbb{N}(\mathrm{s}(0))$	$(\Rightarrow E$ L59,L99)
L57.	$\mathcal{H}_{8}$	$\vdash \mathbb{N}(\mathrm{s}(\mathrm{s}(0)))$	$(\Rightarrow E$ L58,L100)
L56.	$\mathcal{H}_{8}$	$\vdash \mathbb{Z}(\mathrm{s}(\mathrm{s}(0)))$	$(\Rightarrow E$ L57,L101)
L103.	$\mathcal{H}_{8}$	$\vdash \forall X_{60[\nu]}{}_{\blacksquare}[\mathbb{Z}(X_{60}) \Rightarrow \mathbb{Z}((X_{60}\hat{}\mathrm{s}(\mathrm{s}(0))))]$	$(\Rightarrow E$ L56,L102)
L104.	$\mathcal{H}_{8}$	$\vdash [\mathbb{Z}(K) \Rightarrow \mathbb{Z}((K\hat{}\mathrm{s}(\mathrm{s}(0))))]$	$(\forall E$ L103)
L21.	$\mathcal{H}_{28}$	$\vdash \mathbb{Z}(K)$	$(\wedge E_L$ L19)
L55.	$\mathcal{H}_{8}$	$\vdash \mathbb{Z}(K)$	(Weaken L21)
L54.	$\mathcal{H}_{8}$	$\vdash \mathbb{Z}((K\hat{}\mathrm{s}(\mathrm{s}(0))))$	$(\Rightarrow E$ L55,L104)
L26.	$\mathcal{H}_{8}$	$\vdash \mathbb{Z}((K\hat{}2))$	(Defni L54)
L52.	$\mathcal{H}_{8}$	$\vdash [\mathbb{Z}((K\hat{}2)) \wedge (N\hat{}2) = (2{\cdot}(K\hat{}2))]$	$(\wedge I$ L26,L23)
L51.	$\mathcal{H}_{8}$	$\vdash \exists X_{33[\nu]}{}_{\blacksquare}[\mathbb{Z}(X_{33}) \wedge (N\hat{}2) = (2{\cdot}X_{33})]$	$(\exists I$ L52)
L25.	$\mathcal{H}_{8}$	$\vdash \exists X_{24[\nu]} : \mathbb{Z}{}_{\blacksquare}(N\hat{}2) = (2{\cdot}X_{24})$	(Defni L51)
L24.	$\mathcal{H}_{8}$	$\vdash \mathrm{Evenp}((N\hat{}2))$	(Defni L25)
L69.	$\mathcal{H}_{8}$	$\vdash \mathrm{Evenp}(N)$	$(\Rightarrow E$ L24,L90)
L27.	$\mathcal{H}_{8}$	$\vdash \mathrm{Evenp}(N)$	(Weaken L69)
L115.	$\mathcal{H}_{8}$	$\vdash [\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(M)]$	$(\wedge I$ L27,L17)
L116.	$\mathcal{H}_{8}$	$\vdash [[\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(M)] \wedge \neg\mathrm{Common\text{-}Divisor}(N, M, 2)]$	$(\wedge I$ L115,L84)
L149.	$\mathcal{H}_{8}$	$\vdash \neg\mathrm{Common\text{-}Divisor}(N, M, 2)$	$(\wedge E_R$ L116)
L146.	L146	$\vdash [[\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(M)] \Rightarrow \mathrm{Common\text{-}Divisor}(N, M, 2)]$	(Hyp)
L148.	$\mathcal{H}_{8}$	$\vdash [\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(M)]$	$(\wedge E_L$ L116)
L150.	$\mathcal{H}_{29}$	$\vdash \mathrm{Common\text{-}Divisor}(N, M, 2)$	$(\Rightarrow E$ L148,L146)
L147.	$\mathcal{H}_{29}$	$\vdash \bot$	$(\neg E$ L150,L149)
L117.	$\mathcal{H}_{8}$	$\vdash \neg[[\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(M)] \Rightarrow \mathrm{Common\text{-}Divisor}(N, M, 2)]$	$(\neg I$ L147)
L118.	$\mathcal{H}_{8}$	$\vdash [\mathbb{Z}(M) \wedge \neg[[\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(M)] \Rightarrow \mathrm{Common\text{-}Divisor}(N, M, 2)]]$	$(\wedge I$ L10,L117)
L154.	$\mathcal{H}_{8}$	$\vdash \neg[[\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(M)] \Rightarrow \mathrm{Common\text{-}Divisor}(N, M, 2)]$	$(\wedge E_R$ L118)
L151.	L151	$\vdash [\mathbb{Z}(M) \Rightarrow [[\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(M)] \Rightarrow \mathrm{Common\text{-}Divisor}(N, M, 2)]]$	(Hyp)
L153.	$\mathcal{H}_{8}$	$\vdash \mathbb{Z}(M)$	$(\wedge E_L$ L118)
L155.	$\mathcal{H}_{30}$	$\vdash [[\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(M)] \Rightarrow \mathrm{Common\text{-}Divisor}(N, M, 2)]$	$(\Rightarrow E$ L153,L151)
L152.	$\mathcal{H}_{30}$	$\vdash \bot$	$(\neg E$ L155,L154)
L119.	$\mathcal{H}_{8}$	$\vdash \neg[\mathbb{Z}(M) \Rightarrow [[\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(M)] \Rightarrow \mathrm{Common\text{-}Divisor}(N, M, 2)]]$	$(\neg I$ L152)
L120.	$\mathcal{H}_{8}$	$\vdash \exists X_{70[\nu]}{}_{\blacksquare}\neg[\mathbb{Z}(X_{70}) \Rightarrow [[\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(X_{70})] \Rightarrow \mathrm{Common\text{-}Divisor}(N, X_{70}, 2)]]$	$(\exists I$ L119)
L157.	$\mathcal{H}_{31}$	$\vdash \bot$	$(\exists E$ L120,L159)
L122.	$\mathcal{H}_{8}$	$\vdash \neg\forall X_{70[\nu]}{}_{\blacksquare}[\mathbb{Z}(X_{70}) \Rightarrow [[\mathrm{Evenp}(N) \wedge \mathrm{Evenp}(X_{70})] \Rightarrow \mathrm{Common\text{-}Divisor}(N, X_{70}, 2)]]$	$(\neg I$ L157)
L85.	$\mathcal{H}_{8}$	$\vdash \neg\mathbb{Z}(N)$	$(\Rightarrow E$ L122,L161)
L86.	$\mathcal{H}_{8}$	$\vdash \bot$	$(\neg E$ L6,L85)
L83.	$\mathcal{H}_{8}$	$\vdash \bot$	(Weaken L86)
L29.	$\mathcal{H}_{8}$	$\vdash \bot$	(Weaken L83)
L20.	$\mathcal{H}_{8}$	$\vdash \bot$	(Weaken L29)
L36.	$\mathcal{H}_{8}$	$\vdash \bot$	(Weaken L20)
L18.	$\mathcal{H}_{24}$	$\vdash \exists X_{14[\nu]} : \mathbb{Z}{}_{\blacksquare}M = (2{\cdot}X_{14})$	(Defne L17)

L34.	$\mathcal{H}_{24}$	$\vdash \exists X_{99[\nu]}\boldsymbol{.}[\mathbb{Z}(X_{99}) \land M = (2\cdot X_{99})]$	(Defne L18)
L9.	$\mathcal{H}_{24}$	$\vdash \bot$	($\exists E$ L34,L36)
L39.	$\mathcal{H}_{24}$	$\vdash \bot$	(Weaken L9)
L7.	$\mathcal{H}_{27}$	$\vdash \exists X_{40[\nu]} : \mathbb{Z}\boldsymbol{.}[(\sqrt{2}\cdot N) = X_{40} \land \neg\exists X_{244[\nu]} : \mathbb{Z}\boldsymbol{.}\text{Common-Divisor}(N, X_{40}, X_{244})]$	($\land E_R$ L4)
L37.	$\mathcal{H}_{27}$	$\vdash \exists X_{23[\nu]}\boldsymbol{.}[\mathbb{Z}(X_{23}) \land [(\sqrt{2}\cdot N) = X_{23} \land \neg\exists X_{31[\nu]} : \mathbb{Z}\boldsymbol{.}\text{Common-Divisor}(N, X_{23}, X_{31})]]$	(Defne L7)
L5.	$\mathcal{H}_{32}$	$\vdash \bot$	($\exists E$ L37,L39)
L33.	$\mathcal{H}_{32}$	$\vdash \bot$	(Weaken L5)
RC.	RC	$\vdash \forall X_{[\nu]} : \mathbb{Q}_{[\nu\to o]}\boldsymbol{.}\exists Y_{[\nu]} : \mathbb{Z}, Z_{[\nu]} : \mathbb{Z}\boldsymbol{.}[(X\cdot Y) = Z \land \neg\exists D_{[\nu]} : \mathbb{Z}\boldsymbol{.}\text{Common-Divisor}(Y, Z, D)]$	(Thm)
L40.	$\mathcal{H}_{33}$	$\vdash \forall X_{54[\nu]}\boldsymbol{.}[\mathbb{Q}(X_{54}) \Rightarrow \exists X_{64[\nu]} : \mathbb{Z}, X_{67[\nu]} : \mathbb{Z}\boldsymbol{.}[(X_{54}\cdot X_{64}) = X_{67} \land \neg\exists X_{71[\nu]} : \mathbb{Z}\boldsymbol{.}\text{Common-Divisor}(X_{64}, X_{67}, X_{71})]]$	(Defne RC)
L41.	$\mathcal{H}_{33}$	$\vdash [\mathbb{Q}(\sqrt{2}) \Rightarrow \exists X_{64[\nu]} : \mathbb{Z}, X_{67[\nu]} : \mathbb{Z}\boldsymbol{.}[(\sqrt{2}\cdot X_{64}) = X_{67} \land \neg\exists X_{71[\nu]} : \mathbb{Z}\boldsymbol{.}\text{Common-Divisor}(X_{64}, X_{67}, X_{71})]]$	($\forall E$ L40)
L1.	L1	$\vdash \mathbb{Q}(\sqrt{2})$	(Hyp)
L3.	$\mathcal{H}_{34}$	$\vdash \exists X_{8[\nu]} : \mathbb{Z}, X_{1[\nu]} : \mathbb{Z}\boldsymbol{.}[(\sqrt{2}\cdot X_8) = X_1 \land \neg\exists X_{2[\nu]} : \mathbb{Z}\boldsymbol{.}\text{Common-Divisor}(X_8, X_1, X_2)]$	($\Rightarrow E$ L1,L41)
L31.	$\mathcal{H}_{34}$	$\vdash \exists X_{7[\nu]}\boldsymbol{.}[\mathbb{Z}(X_7) \land \exists X_{5[\nu]} : \mathbb{Z}\boldsymbol{.}[(\sqrt{2}\cdot X_7) = X_5 \land \neg\exists X_{9[\nu]} : \mathbb{Z}\boldsymbol{.}\text{Common-Divisor}(X_7, X_5, X_9)]]$	(Defne L3)
L2.	$\mathcal{H}_{34}$	$\vdash \bot$	($\exists E$ L31,L33)
S2NR.	$\mathcal{H}_{35}$	$\vdash \neg\mathbb{Q}(\sqrt{2})$	($\neg I$ L2)

S2NR = Sqrt2-Not-Rat
ECD = Even-Common-Divisor
SE = Square-Even
RC = Rat-Criterion
TND = Tertium-Non-Datur
NI = Nat-Int
PIC = Power-Int-Closed
SN = Succ-Nat
ZN = Zero-Nat

$\mathcal{H}_1$ = ECD, NI, PIC, SN, TND, 0N
$\mathcal{H}_2$ = ECD, NI, PIC, SN, TND, 0N, L187
$\mathcal{H}_3$ = ECD, NI, PIC, SN, TND, 0N, L185, L187
$\mathcal{H}_4$ = ECD, NI, PIC, SN, TND, 0N, L185
$\mathcal{H}_5$ = ECD, NI, PIC, RC, SE, SN, TND, 0N, L1, L4, L8, L19, L156, L158
$\mathcal{H}_6$ = ECD, NI, PIC, RC, SE, SN, TND, 0N, L1, L4, L8, L19, L141
$\mathcal{H}_7$ = NI, PIC, SN, TND, 0N
$\mathcal{H}_8$ = ECD, NI, PIC, RC, SE, SN, TND, 0N, L1, L4, L8, L19
$\mathcal{H}_9$ = ECD, NI, PIC, RC, SE, SN, TND, 0N, L1, L4, L8, L19, L139
$\mathcal{H}_{10}$ = ECD, NI, PIC, RC, SE, SN, TND, 0N, L1, L4, L8, L19, L142
$\mathcal{H}_{11}$ = NI, PIC, SN, TND, 0N, L8
$\mathcal{H}_{12}$ = ECD, NI, PIC, SE, SN, TND, 0N, L8
$\mathcal{H}_{13}$ = ECD, NI, PIC, SE, SN, TND, 0N, L8, L128
$\mathcal{H}_{14}$ = TND, L131, L166
$\mathcal{H}_{15}$ = TND, L131, L164
$\mathcal{H}_{16}$ = TND, L131, L171
$\mathcal{H}_{17}$ = TND, L131, L172
$\mathcal{H}_{18}$ = TND, L131, L177
$\mathcal{H}_{19}$ = TND, L131, L178
$\mathcal{H}_{20}$ = ECD, NI, PIC, SE, SN, TND, 0N, L8, L131
$\mathcal{H}_{21}$ = ECD, NI, PIC, SE, SN, TND, 0N, L8, L191
$\mathcal{H}_{22}$ = ECD, NI, PIC, SE, SN, TND, 0N, L8, L192
$\mathcal{H}_{23}$ = NI, PIC, SE, SN, TND, 0N
$\mathcal{H}_{24}$ = ECD, NI, PIC, RC, SE, SN, TND, 0N, L1, L4, L8
$\mathcal{H}_{25}$ = NI, PIC, TND
$\mathcal{H}_{26}$ = NI, PIC, SN, TND
$\mathcal{H}_{27}$ = ECD, NI, PIC, SE, SN, TND, 0N, L4
$\mathcal{H}_{28}$ = ECD, NI, PIC, SN, TND, 0N, L19
$\mathcal{H}_{29}$ = ECD, NI, PIC, RC, SE, SN, TND, 0N, L1, L4, L8, L19, L146
$\mathcal{H}_{30}$ = ECD, NI, PIC, RC, SE, SN, TND, 0N, L1, L4, L8, L19, L151
$\mathcal{H}_{31}$ = ECD, NI, PIC, RC, SE, SN, TND, 0N, L1, L4, L8, L19, L156
$\mathcal{H}_{32}$ = ECD, NI, PIC, RC, SE, SN, TND, 0N, L1, L4
$\mathcal{H}_{33}$ = NI, PIC, RC, SN, TND, 0N
$\mathcal{H}_{34}$ = ECD, NI, PIC, RC, SE, SN, TND, 0N, L1
$\mathcal{H}_{35}$ = ECD, NI, PIC, RC, SE, SN, TND, 0N

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 a idea to add its definition to this file.
%
% The definitions here will overdefine/overwrite the standard print style
% generated by post2tex.

% Numbers (amsfonts!)
\def\Nat{\mathbb N}
\def\Int{\mathbb Z}
\def\Rat{\mathbb Q}
```

```
\def\Real{\mathbb R}

% Types
\def\ptotomicron{o}
\def\ptotnu{\nu}

% Sig Base
\def\ptotFALSEa{{\bot}}
\def\ptotFALSEoa{{\bot}_{[\ptotomicron]}}


% Sig Set
\def\ptotINc#1#2{#1\in #2}
\def\ptotINLBRnumKOMLBRnumRBRARoRBRARoc#1#2{#1\in_{[(\ptotnu,\ptotnu \rightarrow
\ptotomicron)\rightarrow \ptotomicron]}#2}

% Sig Struct
\def\ptotASSOCIATIVELBRLBRLBRnumRBRARoRBRARoKOMLBRLBRnumRBRARoKOMLBRnumRBRARoKOMnumRBRARoRBRARoc#1#2
{\mbox{associative}(#1,#2)}

% Sig Natural
\def\ptotNATa{{\Nat}}
\def\ptotNATLBRnumRBRARoa{{\Nat}_{[\ptotnu \rightarrow \ptotomicron]}}
\def\ptotNATb#1{{\Nat}(#1)}
\def\ptotZEROa{{0}}
\def\ptotZEROnuma{{0}_{[\ptotnu]}}
\def\ptotSb#1{\mbox{s}(#1)}
\def\ptotSLBRnumRBRARnumb#1{\mbox{s}_{[\ptotnu \rightarrow \ptotnu]}(#1)}

% Sig Integer
\def\ptotINTa{{\Int}}
\def\ptotINTLBRnumRBRARoa{{\Int}_{[\ptotnu \rightarrow \ptotomicron]}}
\def\ptotINTb#1{{\Int}(#1)}
\def\ptotINTLBRnumRBRARob#1{{\Int}_{[\ptotnu \rightarrow \ptotomicron]}(#1)}
\def\ptotPLUSc#1#2{(#1{+}#2)}
\def\ptotPLUSLBRnumKOMnumRBRARnumc#1#2{(#1{+}_{[(\ptotnu,\ptotnu)\rightarrow \ptotnu]}#2)}
\def\ptotMINUSc#1#2{(#1{+}#2)}
\def\ptotMINUSLBRnumKOMnumRBRARnumc#1#2{(#1{+}_{[(\ptotnu,\ptotnu)\rightarrow \ptotnu]}#2)}
\def\ptotTIMESc#1#2{(#1{\cdot}#2)}
\def\ptotTIMESLBRnumKOMnumRBRARnumc#1#2{(#1{\cdot}_{[(\ptotnu,\ptotnu)\rightarrow \ptotnu]}#2)}
\def\ptotMODc#1#2{(#1\bmod #2)}
\def\ptotMODLBRnumKOMnumRBRARnumc#1#2{(#1\bmod_{[(\ptotnu,\ptotnu)\rightarrow \ptotnu]}#2)}
\def\ptotEVENPb#1{\mbox{Evenp}(#1)}
\def\ptotEVENPLBRnumRBRARob#1{\mbox{Evenp}_{[\ptotnu \rightarrow \ptotomicron]}(#1)}
\def\ptotPOWERc#1#2{(#1\hat{\ }#2)}
\def\ptotPOWERLBRnumKOMnumRBRARnumc#1#2{(#1{\hat{\ }}_{[(\ptotnu,\ptotnu)\rightarrow \ptotnu]}#2)}
\def\ptotCOMMONminusDIVISORd#1#2#3{\mbox{Common-Divisor}(#1,#2,#3)}
\def\ptotCOMMONminusDIVISORLBRnumKOMnumKOMnumRBRARod#1#2#3{\mbox{Common-Divisor}_{[(\ptotnu,\ptotnu,
\ptotnu)\rightarrow \ptotomicron]}(#1,#2,#3)}

% Sig Rational
\def\ptotRATb#1{{\Rat}(#1)}
\def\ptotRATLBRnumRBRARob#1{{\Rat}_{[\ptotnu \rightarrow \ptotomicron]}(#1)}
\def\ptotRATLBRnumRBRARoa{{\Rat}_{[\ptotnu \rightarrow \ptotomicron]}}
\def\ptotSQRTb#1{{\sqrt{#1}}}
\def\ptotSQRTLBRnumRBRARnumb#1{\sqrt{_{[\ptotnu \rightarrow \ptotnu]}#1}}

% Sig ZMZ
\def\ptotRESCLASSminusSETb#1{\Int_#1}
\def\ptotRESCLASSminusSETLBRnumKOMLBRnumRBRARoRBRARob#1{\Int_#1}
\def\ptotRESCLASSc#1#2{\mbox{Resclass}(#2,#1)}
\def\ptotRESCLASSLBRnumKOMnumKOMnumRBRARoc#1#2{\mbox{Resclass}_{[(\ptotnu,\ptotnu,\ptotnu)\rightarrow
\ptotomicron]}(#2,#1)}
\def\ptotPLUSminusRESCLASSc#1#2{(#1\bar{+}#2)}
\def\ptotPLUSminusRESCLASSLBRLBRnumRBRARoKOMLBRnumRBRARoKOMnumRBRARoa{\bar{+}}
\def\ptotPLUSminusRESCLASSLBRLBRnumRBRARoKOMLBRnumRBRARoKOMnumRBRARoc#1#2{(#1\bar{+}_{[(\ptotnu
\rightarrow \ptotomicron,\ptotnu \rightarrow \ptotomicron,\ptotnu)\rightarrow \ptotomicron]}#2)}
```

```
% Standard Variables
\def\ptotVARDGDa{{X_{3}}}
\def\ptotVARDGDnuma{{X_{3}}_{[\ptotnu]}}
\def\ptotVARDGCa{{X_{2}}}
\def\ptotVARDGCnuma{{X_{2}}_{[\ptotnu]}}
\def\ptotVARDFJa{{X_{1}}}
\def\ptotVARDFJnuma{{X_{1}}_{[\ptotnu]}}
```

B ΩMEGA Proof Objects

In this section, we give the proof objects in $\mathcal{POST}$ 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 ($\mathcal{PDS}$) 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.

```
(PDS (problem SQRT2-NOT-RAT)
  (in REAL)
 (declarations (type-variables )(type-constants )
   (constants  (K NUM) (N NUM) (M NUM))(meta-variables )(variables ))
   (conclusion SQRT2-NOT-RAT)
   (assumptions)
   (open-nodes)
   (support-nodes EVEN-COMMON-DIVISOR SQUARE-EVEN RAT-CRITERION)
   (nodes
     (L19 (L19) (AND (INT K) (= M (TIMES 2 K)))
      (0 ("HYP" () () "grounded" () ()))
      )
     (L22 (EVEN-COMMON-DIVISOR L19) (= M (TIMES 2 K))
      (0 ("ANDE" () (L19) "unexpanded" ()
          ("L21" "NONEXISTENT" "EXISTENT")))
      )
     (L21 (EVEN-COMMON-DIVISOR L19) (INT K)
      (0 ("ANDE" () (L19) "unexpanded" ()
          ("NONEXISTENT" "L22" "EXISTENT")))
      )
     (L8 (L8) (AND (INT M) (AND (= (TIMES (SQRT 2) N) M)
                               (NOT (EXISTS-SORT (lam (VAR76 NUM) (COMMON-DIVISOR N M VAR76)) INT))))
      (0 ("HYP" () () "grounded" () ()))
      )
     (L12 (EVEN-COMMON-DIVISOR SQUARE-EVEN L8) (NOT (EXISTS-SORT (lam (VAR76 NUM) (COMMON-DIVISOR N M VAR76)) INT))
      (0 ("ANDE*" () (L8) "unexpanded" ()
          ("L10" "L11" "NONEXISTENT" "EXISTENT")))
      )
     (L11 (EVEN-COMMON-DIVISOR SQUARE-EVEN L8) (= (TIMES (SQRT 2) N) M)
      (0 ("ANDE*" () (L8) "unexpanded" ()
          ("L10" "NONEXISTENT" "L12" "EXISTENT")))
      )
     (L10 (EVEN-COMMON-DIVISOR SQUARE-EVEN L8) (INT M)
      (0 ("ANDE*" () (L8) "unexpanded" ()
          ("NONEXISTENT" "L11" "L12" "EXISTENT")))
      )
     (L4 (L4) (AND (INT N) (EXISTS-SORT (lam (VAR79 NUM) (AND (= (TIMES (SQRT 2) N) VAR79)
      (NOT (EXISTS-SORT (lam (VAR80 NUM) (COMMON-DIVISOR N VAR79 VAR80)) INT)))) INT))
      (0 ("HYP" () () "grounded" () ()))
      )
     (L7 (EVEN-COMMON-DIVISOR SQUARE-EVEN L4) (EXISTS-SORT (lam (VAR79 NUM) (AND (= (TIMES
      (SQRT 2) N) VAR79) (NOT (EXISTS-SORT (lam (VAR80 NUM) (COMMON-DIVISOR N VAR79 VAR80)) INT)))) INT)
      (0 ("ANDE" () (L4) "unexpanded" ()
          ("L6" "NONEXISTENT" "EXISTENT")))
      )
     (L6 (EVEN-COMMON-DIVISOR SQUARE-EVEN L4) (INT N)
      (0 ("ANDE" () (L4) "unexpanded" ()
          ("NONEXISTENT" "L7" "EXISTENT")))
      )
     (L1 (L1) (RAT (SQRT 2))
      (0 ("HYP" () () "grounded" () ()))
      )
     (L2 (EVEN-COMMON-DIVISOR SQUARE-EVEN RAT-CRITERION L1) FALSE
      (0 ("EXISTSE-SORT" ((:pds-term N)) (L3 L5) "unexpanded" ()
          ("EXISTENT" "EXISTENT" "NONEXISTENT")))
      )
     (RAT-CRITERION (RAT-CRITERION) (FORALL-SORT (lam (VAR81 NUM) (EXISTS-SORT (lam (VAR82 NUM)
      (EXISTS-SORT (lam (VAR83 NUM) (AND (= (TIMES VAR81 VAR82) VAR83) (NOT (EXISTS-SORT
      (lam (VAR84 NUM) (COMMON-DIVISOR VAR82 VAR83 VAR84)) INT)))) INT)) INT)) RAT)
      (0 ("THM" () () "grounded" () ()))
      )
     (L9 (EVEN-COMMON-DIVISOR SQUARE-EVEN L8 L4 RAT-CRITERION L1) FALSE
```

```
 (0 ("EXISTSE-SORT" ((:pds-term K)) (L18 L20) "unexpanded" ()
     ("EXISTENT" "EXISTENT" "NONEXISTENT")))
 )
(L5 (EVEN-COMMON-DIVISOR SQUARE-EVEN L4 RAT-CRITERION L1) FALSE
 (0 ("EXISTSE-SORT" ((:pds-term M)) (L7 L9) "unexpanded" ()
     ("EXISTENT" "EXISTENT" "NONEXISTENT")))
 )
(L3 (EVEN-COMMON-DIVISOR SQUARE-EVEN RAT-CRITERION L1) (EXISTS-SORT (lam (VAR85 NUM)
 (EXISTS-SORT (lam (VAR86 NUM) (AND (= (TIMES (SQRT 2) VAR85) VAR86) (NOT (EXISTS-SORT
 (lam (VAR87 NUM) (COMMON-DIVISOR VAR85 VAR86 VAR87)) INT)))) INT)) INT)
 (0 ("FORALLE-SORT" ((:pds-term (SQRT 2))) (RAT-CRITERION L1) "unexpanded"
     () ("NONEXISTENT" "EXISTENT" "EXISTENT")))
 )
(L13 (EVEN-COMMON-DIVISOR SQUARE-EVEN L8 L4 RAT-CRITERION L1) (= (POWER M 2)
 (TIMES 2 (POWER N 2)))
 (0 ("BY-COMPUTATION" () (L11) "unexpanded" ()
     ("EXISTENT" "EXISTENT")))
 )
(L15 (EVEN-COMMON-DIVISOR SQUARE-EVEN L8 L4 RAT-CRITERION L1) (EXISTS-SORT (lam (VAR88 NUM)
 (= (POWER M 2) (TIMES 2 VAR88))) INT)
 (0 ("EXISTSI-SORT" ((:pds-term (POWER N 2))((:pds-post-obj (position 2 2)))) (L13 L16)
 "unexpanded"
     () ("EXISTENT" "EXISTENT" "EXISTENT")))
 )
(L14 (EVEN-COMMON-DIVISOR SQUARE-EVEN L8 L4 RAT-CRITERION L1) (EVENP (POWER M 2))
 (0 ("DefnI" ((:pds-term EVENP)(:pds-term (lam (X NUM) (EXISTS-SORT (lam (Y NUM)
 (= X (TIMES 2 Y))) INT)))(:pds-post-obj (position 0))) (L15) "grounded"
     () ("EXISTENT" "NONEXISTENT")))
 )
(L16 (EVEN-COMMON-DIVISOR SQUARE-EVEN L8 L4 RAT-CRITERION L1) (INT (POWER N 2))
 (0 ("WELLSORTED" ((((:pds-term (POWER N (S (S ZERO))))(:pds-sort INT)(:pds-symbol
  POWER-INT-CLOSED))((:pds-term (S (S ZERO)))(:pds-sort INT)(:pds-symbol NAT-INT))
  ((:pds-term (S (S ZERO)))(:pds-sort NAT)(:pds-symbol SUCC-NAT))((:pds-term (S ZERO))
  (:pds-sort NAT)(:pds-symbol SUCC-NAT))((:pds-term ZERO)(:pds-sort NAT)
  (:pds-symbol ZERO-NAT)))) (L6) "unexpanded"
     () ("EXISTENT" "EXISTENT")))
 )
(SQUARE-EVEN (SQUARE-EVEN) (FORALL-SORT (lam (VAR89 NUM) (EQUIV (EVENP (POWER VAR89 2))
 (EVENP VAR89))) INT)
 (0 ("THM" () () "grounded" () ()))
 )
(L20 (EVEN-COMMON-DIVISOR L19 SQUARE-EVEN L8 L4 RAT-CRITERION L1) FALSE
 (0 ("WEAKEN" () (L29) "grounded" () ("EXISTENT" "EXISTENT")))
 )
(L17 (EVEN-COMMON-DIVISOR SQUARE-EVEN L8 L4 RAT-CRITERION L1) (EVENP M)
 (0 ("ASSERT" ((:pds-term (EVENP M))(:pds-nil)) (SQUARE-EVEN L10 L14) "unexpanded"
     () ("NONEXISTENT" "EXISTENT" "EXISTENT" "EXISTENT")))
 )
(L18 (EVEN-COMMON-DIVISOR SQUARE-EVEN L8 L4 RAT-CRITERION L1) (EXISTS-SORT (lam (VAR90 NUM)
 (= M (TIMES 2 VAR90))) INT)
 (0 ("DefnE" ((:pds-term EVENP)(:pds-term (lam (X NUM) (EXISTS-SORT (lam (Y NUM)
 (= X (TIMES 2 Y))) INT)))(:pds-post-obj (position 0))) (L17) "grounded"
     () ("NONEXISTENT" "EXISTENT")))
 )
(L23 (EVEN-COMMON-DIVISOR L19 SQUARE-EVEN L8 L4 RAT-CRITERION L1) (= (POWER N 2) (TIMES 2 (POWER K 2)))
 (0 ("BY-COMPUTATION" () (L13 L22) "unexpanded" ()
     ("EXISTENT" "EXISTENT" "EXISTENT")))
 )
(L25 (EVEN-COMMON-DIVISOR L19 SQUARE-EVEN L8 L4 RAT-CRITERION L1) (EXISTS-SORT (lam
 (VAR91 NUM) (= (POWER N 2) (TIMES 2 VAR91))) INT)
 (0 ("EXISTSI-SORT" ((:pds-term (POWER K 2))((:pds-post-obj (position 2 2)))) (L23 L26)
 "unexpanded"
     () ("EXISTENT" "EXISTENT" "EXISTENT")))
 )
(L24 (EVEN-COMMON-DIVISOR L19 SQUARE-EVEN L8 L4 RAT-CRITERION L1) (EVENP (POWER N 2))
 (0 ("DefnI" ((:pds-term EVENP)(:pds-term (lam (X NUM) (EXISTS-SORT (lam (Y NUM)
 (= X (TIMES 2 Y))) INT)))(:pds-post-obj (position 0))) (L25) "grounded"
     () ("EXISTENT" "NONEXISTENT")))
 )
(L27 (EVEN-COMMON-DIVISOR L19 SQUARE-EVEN L8 L4 RAT-CRITERION L1) (EVENP N)
 (0 ("ASSERT" ((:pds-term (EVENP N))(:pds-nil)) (SQUARE-EVEN L6 L24) "unexpanded"
     () ("NONEXISTENT" "EXISTENT" "EXISTENT" "EXISTENT")))
 )
(L26 (EVEN-COMMON-DIVISOR L19 SQUARE-EVEN L8 L4 RAT-CRITERION L1) (INT (POWER K 2))
 (0 ("WELLSORTED" ((((:pds-term (POWER K (S (S ZERO))))(:pds-sort INT)(:pds-symbol POWER-INT-CLOSED))
 ((:pds-term (S (S ZERO)))(:pds-sort INT)(:pds-symbol NAT-INT))((:pds-term (S (S ZERO)))
 (:pds-sort NAT)(:pds-symbol SUCC-NAT))((:pds-term (S ZERO))(:pds-sort NAT)(:pds-symbol SUCC-NAT))
 ((:pds-term ZERO)(:pds-sort NAT)(:pds-symbol ZERO-NAT)))) (L21) "unexpanded"
     () ("EXISTENT" "EXISTENT")))
 )
```

```
    (EVEN-COMMON-DIVISOR (EVEN-COMMON-DIVISOR) (FORALL-SORT (lam (VAR92 NUM) (FORALL-SORT
     (lam (VAR93 NUM) (IMPLIES (AND (EVENP VAR92) (EVENP VAR93)) (COMMON-DIVISOR VAR92 VAR93 2)))
     INT)) INT)
     (0 ("THM" () () "grounded" () ()))
    )
   (L28 (EVEN-COMMON-DIVISOR L19 SQUARE-EVEN L8 L4 RAT-CRITERION L1) (INT 2)
    (0 ("WELLSORTED" ((((:pds-term (S (S ZERO)))(:pds-sort INT)(:pds-symbol NAT-INT))
    ((:pds-term (S (S ZERO)))(:pds-sort NAT)(:pds-symbol SUCC-NAT))((:pds-term (S ZERO))
    (:pds-sort NAT)(:pds-symbol SUCC-NAT))((:pds-term ZERO)(:pds-sort NAT)(:pds-symbol ZERO-NAT))))
    () "unexpanded"
        () ("EXISTENT")))
    )
   (L29 (EVEN-COMMON-DIVISOR L19 SQUARE-EVEN L8 L4 RAT-CRITERION L1) FALSE
    (0 ("ASSERT" ((:pds-term FALSE)(:pds-nil)) (EVEN-COMMON-DIVISOR L10 L6 L12 L17 L27 L28)
     "unexpanded"
        ()
        ("NONEXISTENT" "EXISTENT" "EXISTENT" "EXISTENT" "EXISTENT" "EXISTENT" "EXISTENT"
         "EXISTENT")))
    )
   (SQRT2-NOT-RAT (EVEN-COMMON-DIVISOR SQUARE-EVEN RAT-CRITERION) (NOT (RAT (SQRT 2)))
    (0 ("NOTI" () (L2) "grounded" () ("EXISTENT" "NONEXISTENT")))
    ))
 (lemmata)
 (agenda)
 (controls
   (L19 (() () () ()))
   (L22 (() () () ()))
   (L21 (() () () ()))
   (L8 (() () () ()))
   (L12 (() () () ()))
   (L11 (() () () ()))
   (L10 (() () () ()))
   (L4 (() () () ()))
   (L7 (() () () ()))
   (L6 (() () () ()))
   (L1 (() () () ()))
   (L2 ((L3 L1) () () ()))
   (RAT-CRITERION (() () () ()))
   (L9 ((L16 L6 L7 L4 L1 L3 L8 L12 L11 L10 L13 L14 L17 L18) (L5 L2 L17) () ()))
   (L5 ((L6 L7 L4 L1 L3) (L2) () ()))
   (L3 (() () () ()))
   (L13 ((L10 L11 L12 L8 L3 L1 L4 L7 L6) () () ()))
   (L15 ((L13 L10 L11 L12 L8 L3 L1 L4 L7 L6) () () ()))
   (L14 ((L13 L10 L11 L12 L8 L3 L1 L4 L7 L6) () () ()))
   (L16 ((L6 L7 L4 L1 L3 L8 L12 L11 L10 L13 L14) () () ()))
   (SQUARE-EVEN (() () () ()))
   (L20 ((L29 L28 L27 L24 L23 L21 L22 L19 L18 L14 L13 L10 L11 L12 L8 L3 L1 L4 L7 L6 L16 L26 L17)
   (L9 L5 L2) () ()))
   (L17 ((L26 L16 L6 L7 L4 L1 L3 L8 L12 L11 L10 L13 L14 L17 L18 L19 L22 L21 L23 L24 L27 L28)
   (L17 L2 L5 L9) () ()))
   (L18 (() () () ()))
   (L23 ((L21 L22 L19 L18 L14 L13 L10 L11 L12 L8 L3 L1 L4 L7 L6 L16) () () ()))
   (L25 ((L23 L21 L22 L19 L18 L14 L13 L10 L11 L12 L8 L3 L1 L4 L7 L6 L16) () () ()))
   (L24 ((L23 L21 L22 L19 L18 L14 L13 L10 L11 L12 L8 L3 L1 L4 L7 L6 L16) () () ()))
   (L27 (() () () ()))
   (L26 ((L16 L6 L7 L4 L1 L3 L8 L12 L11 L10 L13 L14 L18 L19 L22 L21 L23 L24) () () ()))
   (EVEN-COMMON-DIVISOR (() () () ()))
   (L28 ((L27 L24 L23 L21 L22 L19 L18 L14 L13 L10 L11 L12 L8 L3 L1 L4 L7 L6 L16 L26) () () ()))
   (L29 (() () () ()))
   (SQRT2-NOT-RAT (() () () ())))
 (plan-steps (SQRT2-NOT-RAT 0 L1 0 L2 0) (L3 0 RAT-CRITERION 0 L1 0)
   (L2 0 L4 0 L3 0 L5 0) (L6 0 L4 0) (L7 0 L4 0)
   (L5 0 L8 0 L7 0 L9 0) (L10 0 L8 0) (L11 0 L8 0) (L12 0 L8 0)
   (L13 0 L11 0) (L14 0 L15 0) (L16 0 L6 0) (L15 0 L13 0 L16 0)
   (L17 0 SQUARE-EVEN 0 L10 0 L14 0) (L18 0 L17 0)
   (L9 0 L19 0 L18 0 L20 0) (L21 0 L19 0) (L22 0 L19 0)
   (L23 0 L13 0 L22 0) (L24 0 L25 0) (L26 0 L21 0)
   (L25 0 L23 0 L26 0) (L27 0 SQUARE-EVEN 0 L6 0 L24 0) (L28 0)
   (L29 0 EVEN-COMMON-DIVISOR 0 L10 0 L6 0 L12 0 L17 0 L27 0 L28 0)
   (L20 0 L29 0) ))
```

B.2 The Expanded Proof

The expansion of the $\mathcal{PDS}$ from Section B.1 leads to the insertion of more detailed proof objects for each proof node that is not justified by a base-level calculus rule. The only inferences that cannot be fully expanded yet in our proof are the computation steps justified with `by-computation`. By recursively expanding the $\mathcal{PDS}$ fully in this way we obtain a proof object that consists of about

200 nodes in total. This $\mathcal{PDS}$ is unfortunately too large to be presented in this report. The base-level view on this $\mathcal{PDS}$ corresponds to the LaTeX representation of the fully expanded proof in Appendix A.

For an illustration of the way a $\mathcal{PDS}$ 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 ΩMEGA 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 $\mathcal{PDS}$ via TRAMP. There, it can be checked after expansion to ΩMEGA'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 l10  (int m))
         (assumption l6   (int n))
         (assumption l12  (not (exists-sort (lam (x num) (common-divisor n m x)) int)))
         (assumption l17  (evenp m))
         (assumption l27  (evenp n))
         (assumption l28  (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 l29  false)
         )
```

C.2 Interactive Session with a Call of OTTER

```
OMEGA: read-problem "~/omega/tex/omega/sqrt-2-omega/sqrt2-tramp-formalization.post"
;;; Rules loaded for theory REAL.
;;; Theorems loaded for theory REAL.
;;; Tactics loaded for theory REAL.
;;; Methods loaded for theory REAL.
Redefining problem SQRT-PART

Changing to proof plan SQRT-PART-1

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

L29                 (L10 L6    ! FALSE                                                     OPEN
                     L12 L17
                     L27 L28
                     EVEN-COMMON-DIVISOR)
```

```
OMEGA: call-otter-on-node
NODE (NDLINE) Node to prove with OTTER: [L29]
DIR (CREATING-DIRECTORY) The (writable!) directory for depositing OTTER auxiliary
files: [/tmp/chris-atp-dir/]
MODE (SYMBOL) Mode for calling OTTER (auto/user/combined): [AUTO]
EXPAND (SYMBOL) A proof found by OTTER is used to (test/parse/expand): [EXPAND]
PROOF-OBJECT (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: []
RESSOURCE (INTEGER) A time ressource in seconds (integer).: [10]
SSPU-STYLE (SYMBOL) The SSPU-style (direct/compact/auto): [AUTO]
INDIRECT-PROOF (BOOLEAN) Indirect proof: [()]
INTEGRAL-FORMULAS (BOOLEAN) Integral formulas: [()]
MAXIMAL-DEPTH (INTEGER) Maximal depth of searching integral-formulas: [2]
THN (BOOLEAN) Prefer tertium non datur case analyses: [T]
AVOID-DOUBELING (BOOLEAN) Avoid doubling: [T]
LEMMAS (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 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!

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


L1                (L12)      ! (NOT (EXISTS [DC-4424:NUM] (AND (INT DC-4424)
                                                    (COMMON-DIVISOR N M DC-4424))))
  DEFNE*: (KAP[DC-7]. EXISTS-SORT KAP[AA]. ([T, S].(EXISTS [X:AA] (AND (S X) (T X)))) ((1 0))) (L12)


L17               (L17)      ! (EVENP M)                                                 HYP


L27               (L27)      ! (EVENP N)                                                 HYP


L28               (L28)      ! (INT 2)                                                   HYP
```

```
L30                (L12 L28)  ! (NOT (COMMON-DIVISOR N M 2))                  ASSERTION: (L1 L28)


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))))))
                                                                                         DEFNE*:
     (KAP[DC-6]. FORALL-SORT KAP[AA]. ([V, U].(FORALL [X:AA] (IMPLIES (U X) (V X)))) ((0) (1 0 0)))
                                                                           (EVEN-COMMON-DIVISOR)


L31                (EVEN-COMMON-DIVISOR ! (NOT (INT N))           ASSERTION: (L2 L10 L27 L17 L30)
                    L10 L27 L17 L12 L28)


L32                (L6                 ! FALSE                                     NOTE: (L6 L31)
                    EVEN-COMMON-DIVISOR
                    L10 L27 L17 L12 L28)


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


OMEGA: 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 L35 ...
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 ...
```

```
Expanding the node L54 ...
Expanding the node L56 ...
Expanding the node L55 ...
Expanding the node L35 ...
Expanding the node L68 ...
Expanding the node L73 ...
Expanding the node L80 ...
Expanding the node L35 ...
Checking nodes.
Node #<pdsn+node #:L50> has a correct justification.
Node #<pdsn+node #:L87> has a correct justification.
Node #<pdsn+node #:L84> has a correct justification.
Node #<pdsn+node #:L81> has a correct justification.
Node #<pdsn+node #:L88> has a correct justification.
Node #<pdsn+node #:L86> has a correct justification.
Node #<pdsn+node #:L82> has a correct justification.
Node #<pdsn+node #:L52> has a correct justification.
Node #<pdsn+node #:L54> has a correct justification.
Node #<pdsn+node #:L93> has a correct justification.
Node #<pdsn+node #:L92> has a correct justification.
Node #<pdsn+node #:L56> has a correct justification.
Node #<pdsn+node #:L53> has a correct justification.
Node #<pdsn+node #:L99> has a correct justification.
Node #<pdsn+node #:L98> has a correct justification.
Node #<pdsn+node #:L55> has a correct justification.
Node #<pdsn+node L10> has a correct justification.
Node #<pdsn+node L6> has a correct justification.
Node #<pdsn+node L12> has a correct justification.
Node #<pdsn+node #:L1> has a correct justification.
Node #<pdsn+node #:L49> has a correct justification.
Node #<pdsn+node #:L48> has a correct justification.
Node #<pdsn+node #:L46> has a correct justification.
Node #<pdsn+node #:L33> has a correct justification.
Node #<pdsn+node #:L34> has a correct justification.
Node #<pdsn+node #:L113> has a correct justification.
Node #<pdsn+node #:L112> has a correct justification.
Node #<pdsn+node #:L35> has a correct justification.
Node #<pdsn+node L17> has a correct justification.
Node #<pdsn+node L27> has a correct justification.
Node #<pdsn+node #:L38> has a correct justification.
Node #<pdsn+node L28> has a correct justification.
Node #<pdsn+node #:L59> has a correct justification.
Node #<pdsn+node #:L36> has a correct justification.
Node #<pdsn+node #:L63> has a correct justification.
Node #<pdsn+node #:L62> has a correct justification.
Node #<pdsn+node #:L30> has a correct justification.
Node #<pdsn+node #:L39> 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 #<pdsn+node #:L40> has a correct justification.
Node #<pdsn+node #:L41> 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 #:L42> has a correct justification.
Node #<pdsn+node #:L43> has a correct justification.
Node #<pdsn+node #:L78> has a correct justification.
Node #<pdsn+node #:L76> has a correct justification.
Node #<pdsn+node #:L45> has a correct justification.
Node #<pdsn+node EVEN-COMMON-DIVISOR> has a correct justification.
Node #<pdsn+node #:L85> has a correct justification.
Node #<pdsn+node #:L83> has a correct justification.
Node #<pdsn+node #:L77> has a correct justification.
Node #<pdsn+node #:L75> has a correct justification.
Node #<pdsn+node #:L79> has a correct justification.
Node #<pdsn+node #:L70> has a correct justification.
```

```
Node #<pdsn+node #:L74> has a correct justification.
Node #<pdsn+node #:L65> has a correct justification.
Node #<pdsn+node #:L69> has a correct justification.
Node #<pdsn+node #:L61> has a correct justification.
Node #<pdsn+node #:L64> has a correct justification.
Node #<pdsn+node #:L60> has a correct justification.
Node #<pdsn+node #:L58> has a correct justification.
Node #<pdsn+node #:L57> has a correct justification.
Node #<pdsn+node #:L51> has a correct justification.
Node #<pdsn+node #:L47> has a correct justification.
Node #<pdsn+node #:L37> has a correct justification.
Node #<pdsn+node #:L3> has a correct justification.
Node #<pdsn+node #:L2> has a correct justification.
Node #<pdsn+node #:L44> has a correct justification.
Node #<pdsn+node #:L107> has a correct justification.
Node #<pdsn+node #:L105> has a correct justification.
Node #<pdsn+node #:L80> has a correct justification.
Node #<pdsn+node #:L31> has a correct justification.
Node #<pdsn+node #:L32> has a correct justification.
Node #<pdsn+node TERTIUM-NON-DATUR> has a correct justification.
Node #<pdsn+node #:L111> has a correct justification.
Node #<pdsn+node #:L110> has a correct justification.
Node #<pdsn+node #:L109> has a correct justification.
Node #<pdsn+node #:L106> has a correct justification.
Node #<pdsn+node #:L108> has a correct justification.
Node #<pdsn+node #:L104> has a correct justification.
Node #<pdsn+node #:L103> has a correct justification.
Node #<pdsn+node #:L101> has a correct justification.
Node #<pdsn+node #:L114> has a correct justification.
Node #<pdsn+node #:L97> has a correct justification.
Node #<pdsn+node #:L96> has a correct justification.
Node #<pdsn+node #:L100> has a correct justification.
Node #<pdsn+node #:L95> has a correct justification.
Node #<pdsn+node #:L91> has a correct justification.
Node #<pdsn+node #:L90> has a correct justification.
Node #<pdsn+node #:L94> has a correct justification.
Node #<pdsn+node #:L89> has a correct justification.
Node #<pdsn+node L29> 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 $\mathcal{POST}$ 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.

```
(PDS (problem SQRT-PART)
  (in REAL)
 (declarations (type-variables )(type-constants )(constants ) (meta-variables )(variables ))
   (conclusion L29)
   (assumptions L10 L6 L12 L17 L27 L28 EVEN-COMMON-DIVISOR)
   (open-nodes)
   (support-nodes L10 L6 L12 L17 L27 L28 EVEN-COMMON-DIVISOR)
   (nodes
     (L10 (L10) (INT M)
      (0 ("HYP" () () "grounded" () ()))
      )
     (L6 (L6) (INT N)
      (0 ("HYP" () () "grounded" () ()))
      )
     (L12 (L12) (NOT (EXISTS-SORT (lam (VAR111 NUM) (COMMON-DIVISOR N M VAR111)) INT))
      (0 ("HYP" () () "grounded" () ()))
      )
     (L1 (L12) (NOT (EXISTS (lam (VAR112 NUM) (AND (INT VAR112) (COMMON-DIVISOR N M VAR112)))))
      (0 ("DEFNE*" ((:pds-term (all-types DC-7 EXISTS-SORT))(:pds-term (all-types AA (lam (T (O AA)) (S (O AA))
           (EXISTS (lam (X AA) (AND (S X) (T X)))))))((:pds-post-obj (position 1 0)))) (L12) "unexpanded"
```

```
        () ("NONEXISTENT" "EXISTENT")))
    )
   (L17 (L17) (EVENP M)
    (0 ("HYP" () () "grounded" () ()))
    )
   (L27 (L27) (EVENP N)
    (0 ("HYP" () () "grounded" () ()))
    )
   (L28 (L28) (INT 2)
    (0 ("HYP" () () "grounded" () ()))
    )
   (L30 (L12 L28) (NOT (COMMON-DIVISOR N M 2))
    (0 ("ASSERTION" () (L1 L28) "unexpanded" () ()))
    )
   (EVEN-COMMON-DIVISOR (EVEN-COMMON-DIVISOR) (FORALL-SORT (lam (VAR113 NUM) (FORALL-SORT (lam (VAR114 NUM)
        (IMPLIES (AND (EVENP VAR113) (EVENP VAR114)) (COMMON-DIVISOR VAR113 VAR114 2))) INT)) INT)
    (0 ("HYP" () () "grounded" () ()))
    )
   (L3 (L10 L6 L12 L17 L27 L28 EVEN-COMMON-DIVISOR) FALSE
    (1 ("OTTER" ((:pds-nil)) (L2 L28 L27 L17 L1 L6 L10) "expanded" ()
        ())
     ("WEAKEN" () (L32) "grounded" () ("EXISTENT" "EXISTENT")))
    )
   (L2 (EVEN-COMMON-DIVISOR) (FORALL (lam (VAR115 NUM) (IMPLIES (INT VAR115) (FORALL (lam (VAR116 NUM)
        (IMPLIES (INT VAR116) (IMPLIES (AND (EVENP VAR115) (EVENP VAR116)) (COMMON-DIVISOR VAR115 VAR116
          2))))))))
    (0 ("DEFNE*" ((:pds-term (all-types DC-6 FORALL-SORT))(:pds-term (all-types AA (lam (V (O AA)) (U (O AA))
        (FORALL (lam (X AA) (IMPLIES (U X) (V X)))))))((:pds-post-obj (position 0))(:pds-post-obj
        (position 1 0 0))))
        (EVEN-COMMON-DIVISOR) "unexpanded"
        () ("NONEXISTENT" "EXISTENT")))
    )
   (L31 (EVEN-COMMON-DIVISOR L10 L27 L17 L12 L28) (NOT (INT N))
    (0 ("ASSERTION" () (L2 L10 L27 L17 L30) "unexpanded" () ()))
    )
   (L32 (L6 EVEN-COMMON-DIVISOR L10 L27 L17 L12 L28) FALSE
    (0 ("NOTE" () (L6 L31) "grounded" ()
        ("NONEXISTENT" "EXISTENT" "EXISTENT")))
    )
   (L29 (L10 L6 L12 L17 L27 L28 EVEN-COMMON-DIVISOR) FALSE
    (2 ("OTTER" ((:pds-nil)) (L10 L6 L12 L17 L27 L28 EVEN-COMMON-DIVISOR) "expanded"
        ()
        ("EXISTENT" "CLOSED" "CLOSED" "CLOSED" "CLOSED" "CLOSED" "CLOSED" "CLOSED"))
     ("OTTER" ((:pds-nil)) (L10 L6 L1 L17 L27 L28 L2) "expanded" ()
      ())
     ("simplify-goal" () (L3) "unexpanded" ()
      ("EXISTENT" "NONEXISTENT")))
    ))
 (lemmata)
 (agenda)
 (controls
   (L10 (() () () ()))
   (L6 (() () () ()))
   (L12 (() () () ()))
   (L1 (() () () ()))
   (L17 (() () () ()))
   (L27 (() () () ()))
   (L28 (() () () ()))
   (L30 (() () () ()))
   (EVEN-COMMON-DIVISOR (() () () ()))
   (L3 (() () () ()) ())
   (L2 (() () () ()))
   (L31 (() () () ()))
   (L32 (() () () ()))
   (L29 (() () () ()) () (() () () ())))
 (plan-steps
   (L29 0 L3 0 L32 0 L31 0 L30 0 L2 0 L1 0 L10 0 L6 0 L12 0 L17 0 L27 0 L28 0 EVEN-COMMON-DIVISOR 0) ))
```

C.4 The Expanded Proof Object

In this section, we give the proof objects in $\mathcal{POST}$ 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.

```
(PDS (problem SQRT-PART)
  (in REAL)
 (declarations (type-variables ) (type-constants ) (constants  (VAR1121 NUM) (VAR1161 NUM)) (meta-variables )
```

```
(variables ))
(conclusion L29)
(assumptions L10 L6 L12 L17 L27 L28 EVEN-COMMON-DIVISOR)
(open-nodes)
(support-nodes TERTIUM-NON-DATUR L10 L6 L12 L17 L27 L28 EVEN-COMMON-DIVISOR)
(nodes
  (L67 (L67) (NOT (OR (NOT (INT 2)) (NOT (COMMON-DIVISOR N M 2))))
   (0 ("HYP" () () "grounded" () ()))
   )
  (L107 (L103 TERTIUM-NON-DATUR L67) (OR (NOT (INT 2)) (NOT (COMMON-DIVISOR N M 2)))
   (0 ("ORIL" ((:pds-term (NOT (COMMON-DIVISOR N M 2)))) (L103) "grounded"
       () ("EXISTENT" "EXISTENT")))
   )
  (L104 (L103 TERTIUM-NON-DATUR L67) FALSE
   (0 ("NOTE" () (L107 L67) "grounded" ()
       ("EXISTENT" "NONEXISTENT" "EXISTENT")))
   )
  (L101 (TERTIUM-NON-DATUR L67) (NOT (NOT (INT 2)))
   (0 ("NOTI" () (L104) "grounded" () ("EXISTENT" "NONEXISTENT")))
   )
  (L108 (L105 TERTIUM-NON-DATUR L67) (OR (NOT (INT 2)) (NOT (COMMON-DIVISOR N M 2)))
   (0 ("ORIR" ((:pds-term (NOT (INT 2)))) (L105) "grounded" ()
       ("EXISTENT" "EXISTENT")))
   )
  (L106 (L105 TERTIUM-NON-DATUR L67) FALSE
   (0 ("NOTE" () (L108 L67) "grounded" ()
       ("EXISTENT" "NONEXISTENT" "EXISTENT")))
   )
  (L102 (TERTIUM-NON-DATUR L67) (NOT (NOT (COMMON-DIVISOR N M 2)))
   (0 ("NOTI" () (L106) "grounded" () ("EXISTENT" "NONEXISTENT")))
   )
  (L69 (TERTIUM-NON-DATUR L67) (AND (NOT (NOT (INT 2))) (NOT (NOT (COMMON-DIVISOR N M 2))))
   (1 ("PUSHNEG" () (L67) "expanded" () ("NONEXISTENT" "EXISTENT"))
    ("ANDI" () (L101 L102) "grounded" ()
     ("EXISTENT" "NONEXISTENT" "NONEXISTENT")))
   )
  (L71 (TERTIUM-NON-DATUR L67) (NOT (NOT (COMMON-DIVISOR N M 2)))
   (1 ("ANDE" () (L69) "unexpanded" ()
       ("L70" "NONEXISTENT" "EXISTENT"))
    ("ANDER" () (L69) "grounded" () ("EXISTENT" "EXISTENT")))
   )
  (L99 (L96 TERTIUM-NON-DATUR L67) (COMMON-DIVISOR N M 2)
   (0 ("FALSEE" () (L100) "grounded" () ("EXISTENT" "EXISTENT")))
   )
  (L98 (L97 TERTIUM-NON-DATUR L67) (COMMON-DIVISOR N M 2)
   (0 ("WEAKEN" () (L97) "grounded" () ("EXISTENT" "EXISTENT")))
   )
  (L73 (TERTIUM-NON-DATUR L67) (COMMON-DIVISOR N M 2)
   (1 ("NOTNOTE" () (L71) "expanded" () ("NONEXISTENT" "EXISTENT"))
    ("ORE" () (L95 L98 L99) "grounded" ()
     ("EXISTENT" "EXISTENT" "NONEXISTENT" "NONEXISTENT")))
   )
  (L70 (TERTIUM-NON-DATUR L67) (NOT (NOT (INT 2)))
   (1 ("ANDE" () (L69) "expanded" ()
       ("NONEXISTENT" "L71" "EXISTENT"))
    ("ANDEL" () (L69) "grounded" () ("EXISTENT" "EXISTENT")))
   )
  (L93 (L90 TERTIUM-NON-DATUR L67) (INT 2)
   (0 ("FALSEE" () (L94) "grounded" () ("EXISTENT" "EXISTENT")))
   )
  (L92 (L91 TERTIUM-NON-DATUR L67) (INT 2)
   (0 ("WEAKEN" () (L91) "grounded" () ("EXISTENT" "EXISTENT")))
   )
  (L72 (TERTIUM-NON-DATUR L67) (INT 2)
   (1 ("NOTNOTE" () (L70) "expanded" () ("NONEXISTENT" "EXISTENT"))
    ("ORE" () (L89 L92 L93) "grounded" ()
     ("EXISTENT" "EXISTENT" "NONEXISTENT" "NONEXISTENT")))
   )
  (L10 (L10) (INT M)
   (0 ("HYP" () () "grounded" () ()))
   )
  (L6 (L6) (INT N)
   (0 ("HYP" () () "grounded" () ()))
   )
  (L12 (L12) (NOT (EXISTS-SORT (lam (VAR181 NUM) (COMMON-DIVISOR N M VAR181)) INT))
   (0 ("HYP" () () "grounded" () ()))
   )
  (L1 (TERTIUM-NON-DATUR L12) (NOT (EXISTS (lam (VAR184 NUM) (AND (INT VAR184) (COMMON-DIVISOR N M VAR184)))))
   (1 ("DEFNE*" ((:pds-term (all-types DC-7 EXISTS-SORT))(:pds-term (all-types AA (lam (T (O AA)) (S (O AA))
        (EXISTS (lam (X AA) (AND (S X) (T X)))))))((:pds-post-obj (position 1 0)))) (L12) "expanded"
```

```
    () ("NONEXISTENT" "EXISTENT"))
  ("DefnE" ((:pds-term (all-types DC-7 EXISTS-SORT))(:pds-term (all-types AA (lam (T (O AA)) (S (O AA))
    (EXISTS (lam (X AA) (AND (S X) (T X)))))))(:pds-post-obj (position 1 0))) (L12) "grounded"
   () ("EXISTENT" "EXISTENT")))
 )
(L80 (TERTIUM-NON-DATUR L78 L12) (EXISTS (lam (VAR184 NUM) (AND (INT VAR184) (COMMON-DIVISOR N M VAR184))))
 (0 ("EXISTSI" ((:pds-term VAR1121)((:pds-post-obj (position 1 1))(:pds-post-obj (position 2 3)))) (L78)
    "grounded" () ("EXISTENT" "EXISTENT")))
 )
(L79 (TERTIUM-NON-DATUR L78 L12) FALSE
 (0 ("NOTE" () (L80 L1) "grounded" ()
    ("EXISTENT" "NONEXISTENT" "EXISTENT")))
 )
(L77 (TERTIUM-NON-DATUR L12) (NOT (AND (INT VAR1121) (COMMON-DIVISOR N M VAR1121)))
 (0 ("NOTI" () (L79) "grounded" () ("EXISTENT" "NONEXISTENT")))
 )
(L42 (TERTIUM-NON-DATUR L12) (FORALL (lam (VAR184 NUM) (NOT (AND (INT VAR184) (COMMON-DIVISOR N M VAR184)))))
 (1 ("PUSHNEG" () (L1) "expanded" () ("EXISTENT" "EXISTENT"))
  ("FORALLI" ((:pds-term VAR1121)) (L77) "grounded" ()
   ("EXISTENT" "NONEXISTENT")))
 )
(L43 (TERTIUM-NON-DATUR L12) (NOT (AND (INT 2) (COMMON-DIVISOR N M 2)))
 (0 ("FORALLE" ((:pds-term 2)) (L42) "grounded" ()
    ("EXISTENT" "EXISTENT")))
 )
(L113 (L110 TERTIUM-NON-DATUR L12) (OR (NOT (INT 2)) (NOT (COMMON-DIVISOR N M 2)))
 (0 ("FALSEE" () (L114) "grounded" () ("EXISTENT" "EXISTENT")))
 )
(L112 (L111 TERTIUM-NON-DATUR L12) (OR (NOT (INT 2)) (NOT (COMMON-DIVISOR N M 2)))
 (0 ("WEAKEN" () (L111) "grounded" () ("EXISTENT" "EXISTENT")))
 )
(L44 (TERTIUM-NON-DATUR L12) (OR (NOT (INT 2)) (NOT (COMMON-DIVISOR N M 2)))
 (3 ("PUSHNEG" () (L43) "expanded" () ("EXISTENT" "EXISTENT"))
  ("INDIRECT" () (L68) "expanded" () ("EXISTENT" "NONEXISTENT"))
  ("NOTNOTE" () (L86) "expanded" () ("EXISTENT" "NONEXISTENT"))
  ("ORE" () (L109 L112 L113) "grounded" ()
   ("EXISTENT" "EXISTENT" "NONEXISTENT" "NONEXISTENT")))
 )
(L17 (L17) (EVENP M)
 (0 ("HYP" () () "grounded" () ()))
 )
(L27 (L27) (EVENP N)
 (0 ("HYP" () () "grounded" () ()))
 )
(L33 (TERTIUM-NON-DATUR L17 L27) (AND (EVENP N) (EVENP M))
 (0 ("ANDI" () (L27 L17) "grounded" ()
    ("EXISTENT" "EXISTENT" "EXISTENT")))
 )
(L28 (L28) (INT 2)
 (0 ("HYP" () () "grounded" () ()))
 )
(L76 (TERTIUM-NON-DATUR L75 L28) FALSE
 (0 ("NOTE" () (L28 L75) "grounded" ()
    ("EXISTENT" "EXISTENT" "EXISTENT")))
 )
(L45 (TERTIUM-NON-DATUR L28) (NOT (NOT (INT 2)))
 (1 ("NOTNOTI" () (L28) "expanded" () ("EXISTENT" "EXISTENT"))
  ("NOTI" () (L76) "grounded" () ("EXISTENT" "NONEXISTENT")))
 )
(L65 (TERTIUM-NON-DATUR L62 L12 L28) (NOT (COMMON-DIVISOR N M 2))
 (0 ("WEAKEN" () (L62) "grounded" () ("EXISTENT" "EXISTENT")))
 )
(L64 (TERTIUM-NON-DATUR L63 L12 L28) (NOT (COMMON-DIVISOR N M 2))
 (0 ("FALSEE" () (L66) "grounded" () ("EXISTENT" "EXISTENT")))
 )
(L30 (TERTIUM-NON-DATUR L12 L28) (NOT (COMMON-DIVISOR N M 2))
 (2 ("ASSERTION" () (L1 L28) "expanded" () ())
  ("ORMP" () (L44 L45) "expanded" ()
   ("EXISTENT" "EXISTENT" "EXISTENT"))
  ("ORE" () (L44 L64 L65) "grounded" ()
   ("EXISTENT" "EXISTENT" "NONEXISTENT" "NONEXISTENT")))
 )
(L34 (TERTIUM-NON-DATUR L28 L12 L17 L27) (AND (AND (EVENP N) (EVENP M)) (NOT (COMMON-DIVISOR N M 2)))
 (0 ("ANDI" () (L33 L30) "grounded" ()
    ("EXISTENT" "EXISTENT" "EXISTENT")))
 )
(L60 (TERTIUM-NON-DATUR L28 L12 L17 L27) (NOT (COMMON-DIVISOR N M 2))
 (1 ("ANDE" () (L34) "unexpanded" ()
    ("L59" "NONEXISTENT" "EXISTENT"))
  ("ANDER" () (L34) "grounded" () ("EXISTENT" "EXISTENT")))
```

```
 )
(L59 (TERTIUM-NON-DATUR L28 L12 L17 L27) (AND (EVENP N) (EVENP M))
 (1 ("ANDE" () (L34) "expanded" ()
     ("NONEXISTENT" "L60" "EXISTENT"))
  ("ANDEL" () (L34) "grounded" () ("EXISTENT" "EXISTENT")))
 )
(L58 (TERTIUM-NON-DATUR L57 L28 L12 L17 L27) FALSE
 (0 ("NOTE" () (L61 L60) "grounded" ()
     ("EXISTENT" "EXISTENT" "EXISTENT")))
 )
(L35 (TERTIUM-NON-DATUR L28 L12 L17 L27) (NOT (IMPLIES (AND (EVENP N) (EVENP M)) (COMMON-DIVISOR N M 2)))
 (1 ("PULLNEG" () (L34) "expanded" () ("EXISTENT" "EXISTENT"))
  ("NOTI" () (L58) "grounded" () ("EXISTENT" "NONEXISTENT")))
 )
(L36 (TERTIUM-NON-DATUR L10 L28 L12 L17 L27) (AND (INT M) (NOT (IMPLIES (AND (EVENP N) (EVENP M))
 (COMMON-DIVISOR N M 2))))
 (0 ("ANDI" () (L10 L35) "grounded" ()
     ("EXISTENT" "EXISTENT" "EXISTENT")))
 )
(L55 (TERTIUM-NON-DATUR L10 L28 L12 L17 L27) (NOT (IMPLIES (AND (EVENP N) (EVENP M)) (COMMON-DIVISOR N M 2)))
 (1 ("ANDE" () (L36) "unexpanded" ()
     ("L54" "NONEXISTENT" "EXISTENT"))
  ("ANDER" () (L36) "grounded" () ("EXISTENT" "EXISTENT")))
 )
(L54 (TERTIUM-NON-DATUR L10 L28 L12 L17 L27) (INT M)
 (1 ("ANDE" () (L36) "expanded" ()
     ("NONEXISTENT" "L55" "EXISTENT"))
  ("ANDEL" () (L36) "grounded" () ("EXISTENT" "EXISTENT")))
 )
(L53 (TERTIUM-NON-DATUR L52 L10 L28 L12 L17 L27) FALSE
 (0 ("NOTE" () (L56 L55) "grounded" ()
     ("EXISTENT" "EXISTENT" "EXISTENT")))
 )
(L37 (TERTIUM-NON-DATUR L10 L28 L12 L17 L27) (NOT (IMPLIES (INT M) (IMPLIES (AND (EVENP N) (EVENP M))
 (COMMON-DIVISOR N M 2))))
 (1 ("PULLNEG" () (L36) "expanded" () ("EXISTENT" "EXISTENT"))
  ("NOTI" () (L53) "grounded" () ("EXISTENT" "NONEXISTENT")))
 )
(L38 (TERTIUM-NON-DATUR L10 L28 L12 L17 L27) (EXISTS (lam (VAR197 NUM) (NOT (IMPLIES (INT VAR197)
      (IMPLIES (AND (EVENP N) (EVENP VAR197)) (COMMON-DIVISOR N VAR197 2))))))
 (0 ("EXISTSI" ((:pds-term M)((:pds-post-obj (position 1 1 1))(:pds-post-obj (position 1 2 1 2 1))
      (:pds-post-obj (position 1 2 2 2)))) (L37) "grounded"
     () ("EXISTENT" "EXISTENT")))
 )
(L50 (TERTIUM-NON-DATUR L49 L47 L10 L28 L12 L17 L27) FALSE
 (0 ("NOTE" () (L51 L49) "grounded" ()
     ("EXISTENT" "EXISTENT" "EXISTENT")))
 )
(L48 (TERTIUM-NON-DATUR L47 L10 L28 L12 L17 L27) FALSE
 (0 ("EXISTSE" ((:pds-term VAR1161)) (L38 L50) "grounded" ()
     ("EXISTENT" "EXISTENT" "NONEXISTENT")))
 )
(L40 (TERTIUM-NON-DATUR L10 L28 L12 L17 L27) (NOT (FORALL (lam (VAR197 NUM) (IMPLIES (INT VAR197)
      (IMPLIES (AND (EVENP N) (EVENP VAR197)) (COMMON-DIVISOR N VAR197 2))))))
 (1 ("PULLNEG" () (L38) "expanded" () ("EXISTENT" "EXISTENT"))
  ("NOTI" () (L48) "grounded" () ("EXISTENT" "NONEXISTENT")))
 )
(EVEN-COMMON-DIVISOR (EVEN-COMMON-DIVISOR) (FORALL-SORT (lam (VAR187 NUM) (FORALL-SORT (lam (VAR188 NUM)
  (IMPLIES (AND (EVENP VAR187) (EVENP VAR188)) (COMMON-DIVISOR VAR187 VAR188 2))) INT)) INT)
 (0 ("HYP" () () "grounded" () ()))
 )
(L88 (TERTIUM-NON-DATUR L67 L12) FALSE
 (0 ("WEAKEN" () (L68) "grounded" () ("EXISTENT" "EXISTENT")))
 )
(L86 (TERTIUM-NON-DATUR L12) (NOT (NOT (OR (NOT (INT 2)) (NOT (COMMON-DIVISOR N M 2)))))
 (0 ("NOTI" () (L88) "grounded" () ("EXISTENT" "NONEXISTENT")))
 )
(L83 (L83) (INT N)
 (0 ("HYP" () () "grounded" () ()))
 )
(L81 (L81) (NOT (FORALL (lam (VAR197 NUM) (IMPLIES (INT VAR197) (IMPLIES (AND (EVENP N) (EVENP VAR197))
           (COMMON-DIVISOR N VAR197 2))))))
 (0 ("HYP" () () "grounded" () ()))
 )
(L78 (L78) (AND (INT VAR1121) (COMMON-DIVISOR N M VAR1121))
 (0 ("HYP" () () "grounded" () ()))
 )
(L75 (L75) (NOT (INT 2))
 (0 ("HYP" () () "grounded" () ()))
 )
```

```
(L74 (TERTIUM-NON-DATUR L67) (AND (INT 2) (COMMON-DIVISOR N M 2))
 (0 ("ANDI" () (L72 L73) "grounded" ()
     ("NONEXISTENT" "EXISTENT" "EXISTENT")))
 )
(L68 (TERTIUM-NON-DATUR L67 L12) FALSE
 (0 ("NOTE" () (L74 L43) "grounded" ()
     ("EXISTENT" "EXISTENT" "EXISTENT")))
 )
(L63 (L63) (NOT (INT 2))
 (0 ("HYP" () () "grounded" () ()))
 )
(L66 (TERTIUM-NON-DATUR L28 L63) FALSE
 (0 ("NOTE" () (L63 L45) "grounded" ()
     ("NONEXISTENT" "EXISTENT" "EXISTENT")))
 )
(L62 (L62) (NOT (COMMON-DIVISOR N M 2))
 (0 ("HYP" () () "grounded" () ()))
 )
(L57 (L57) (IMPLIES (AND (EVENP N) (EVENP M)) (COMMON-DIVISOR N M 2))
 (0 ("HYP" () () "grounded" () ()))
 )
(L61 (TERTIUM-NON-DATUR L57 L28 L12 L17 L27) (COMMON-DIVISOR N M 2)
 (0 ("IMPE" () (L59 L57) "grounded" ()
     ("NONEXISTENT" "EXISTENT" "EXISTENT")))
 )
(L52 (L52) (IMPLIES (INT M) (IMPLIES (AND (EVENP N) (EVENP M)) (COMMON-DIVISOR N M 2)))
 (0 ("HYP" () () "grounded" () ()))
 )
(L56 (TERTIUM-NON-DATUR L52 L10 L28 L12 L17 L27) (IMPLIES (AND (EVENP N) (EVENP M)) (COMMON-DIVISOR N M 2))
 (0 ("IMPE" () (L54 L52) "grounded" ()
     ("NONEXISTENT" "EXISTENT" "EXISTENT")))
 )
(L49 (L49) (NOT (IMPLIES (INT VAR1161) (IMPLIES (AND (EVENP N) (EVENP VAR1161)) (COMMON-DIVISOR N VAR1161
      2))))
 (0 ("HYP" () () "grounded" () ()))
 )
(L47 (L47) (FORALL (lam (VAR197 NUM) (IMPLIES (INT VAR197) (IMPLIES (AND (EVENP N) (EVENP VAR197))
           (COMMON-DIVISOR N VAR197 2)))))
 (0 ("HYP" () () "grounded" () ()))
 )
(L51 (TERTIUM-NON-DATUR L47) (IMPLIES (INT VAR1161) (IMPLIES (AND (EVENP N) (EVENP VAR1161))
                             (COMMON-DIVISOR N VAR1161 2)))
 (0 ("FORALLE" ((:pds-term VAR1161)) (L47) "grounded" ()
     ("NONEXISTENT" "EXISTENT")))
 )
(L41 (TERTIUM-NON-DATUR EVEN-COMMON-DIVISOR) (FORALL-SORT (lam (VAR191 NUM) (FORALL (lam (VAR192 NUM)
     (IMPLIES (INT VAR192) (IMPLIES (AND (EVENP VAR191) (EVENP VAR192)) (COMMON-DIVISOR VAR191 VAR192 2))))))
      INT)
 (0 ("DefnE" ((:pds-term (all-types DC-6 FORALL-SORT))(:pds-term (all-types AA (lam (V (O AA)) (U (O AA))
      (FORALL (lam (X AA) (IMPLIES (U X) (V X)))))))(:pds-post-obj (position 1 0 0)))
       (EVEN-COMMON-DIVISOR) "grounded"
     () ("NONEXISTENT" "EXISTENT")))
 )
(L3 (TERTIUM-NON-DATUR L10 L6 L12 L17 L27 L28 EVEN-COMMON-DIVISOR) FALSE
 (1 ("OTTER" ((:pds-nil)) (L2 L28 L27 L17 L1 L6 L10) "expanded" ()
     ())
  ("WEAKEN" () (L32) "grounded" () ("EXISTENT" "EXISTENT")))
 )
(L2 (TERTIUM-NON-DATUR EVEN-COMMON-DIVISOR) (FORALL (lam (VAR193 NUM) (IMPLIES (INT VAR193)
     (FORALL (lam (VAR197 NUM) (IMPLIES (INT VAR197) (IMPLIES (AND (EVENP VAR193) (EVENP VAR197))
     (COMMON-DIVISOR VAR193 VAR197 2))))))))
 (1 ("DEFNE*" ((:pds-term (all-types DC-6 FORALL-SORT))(:pds-term (all-types AA (lam (V (O AA)) (U (O AA))
     (FORALL (lam (X AA) (IMPLIES (U X) (V X)))))))((:pds-post-obj (position 0))(:pds-post-obj (position 1 0
0)))) (EVEN-COMMON-DIVISOR) "expanded"
     () ("NONEXISTENT" "EXISTENT"))
  ("DefnE" ((:pds-term (all-types DC-6 FORALL-SORT))(:pds-term (all-types AA (lam (V (O AA)) (U (O AA))
            (FORALL (lam (X AA) (IMPLIES (U X) (V X)))))))(:pds-post-obj (position 0))) (L41) "grounded"
   () ("EXISTENT" "EXISTENT")))
 )
(L39 (TERTIUM-NON-DATUR EVEN-COMMON-DIVISOR) (IMPLIES (INT N) (FORALL (lam (VAR197 NUM) (IMPLIES (INT VAR197)
      (IMPLIES (AND (EVENP N) (EVENP VAR197)) (COMMON-DIVISOR N VAR197 2))))))
 (0 ("FORALLE" ((:pds-term N)) (L2) "grounded" ()
     ("EXISTENT" "EXISTENT")))
 )
(L85 (TERTIUM-NON-DATUR EVEN-COMMON-DIVISOR L83) (FORALL (lam (VAR197 NUM) (IMPLIES (INT VAR197)
      (IMPLIES (AND (EVENP N) (EVENP VAR197)) (COMMON-DIVISOR N VAR197 2)))))
 (0 ("IMPE" () (L83 L39) "grounded" ()
     ("NONEXISTENT" "EXISTENT" "EXISTENT")))
 )
(L84 (TERTIUM-NON-DATUR L83 L81 EVEN-COMMON-DIVISOR) FALSE
```

```
 (0 ("NOTE" () (L85 L81) "grounded" ()
     ("EXISTENT" "EXISTENT" "EXISTENT")))
 )
(L82 (TERTIUM-NON-DATUR L81 EVEN-COMMON-DIVISOR) (NOT (INT N))
 (0 ("NOTI" () (L84) "grounded" () ("EXISTENT" "NONEXISTENT")))
 )
(L46 (TERTIUM-NON-DATUR EVEN-COMMON-DIVISOR) (IMPLIES (NOT (FORALL (lam (VAR197 NUM) (IMPLIES (INT VAR197)
      (IMPLIES (AND (EVENP N) (EVENP VAR197)) (COMMON-DIVISOR N VAR197 2)))))) (NOT (INT N)))
 (1 ("CONTRAPOS" () (L39) "expanded" () ("NONEXISTENT" "EXISTENT"))
  ("IMPI" () (L82) "grounded" () ("EXISTENT" "NONEXISTENT")))
 )
(L31 (TERTIUM-NON-DATUR EVEN-COMMON-DIVISOR L10 L27 L17 L12 L28) (NOT (INT N))
 (2 ("ASSERTION" () (L2 L10 L27 L17 L30) "expanded" () ())
  ("MODTOLL" () (L39 L40) "expanded" ()
   ("EXISTENT" "EXISTENT" "EXISTENT"))
  ("IMPE" () (L40 L46) "grounded" ()
   ("EXISTENT" "EXISTENT" "EXISTENT")))
 )
(L32 (TERTIUM-NON-DATUR L6 EVEN-COMMON-DIVISOR L10 L27 L17 L12 L28) FALSE
 (0 ("NOTE" () (L6 L31) "grounded" ()
     ("NONEXISTENT" "EXISTENT" "EXISTENT")))
 )
(TERTIUM-NON-DATUR (TERTIUM-NON-DATUR) (FORALL (lam (VAR198 O) (OR VAR198 (NOT VAR198))))
 (0 ("AXIOM" () () "grounded" () ()))
 )
(L111 (L111) (OR (NOT (INT 2)) (NOT (COMMON-DIVISOR N M 2)))
 (0 ("HYP" () () "grounded" () ()))
 )
(L110 (L110) (NOT (OR (NOT (INT 2)) (NOT (COMMON-DIVISOR N M 2))))
 (0 ("HYP" () () "grounded" () ()))
 )
(L114 (L110 TERTIUM-NON-DATUR L12) FALSE
 (0 ("NOTE" () (L110 L86) "grounded" ()
     ("NONEXISTENT" "EXISTENT" "EXISTENT")))
 )
(L109 (TERTIUM-NON-DATUR) (OR (OR (NOT (INT 2)) (NOT (COMMON-DIVISOR N M 2))) (NOT (OR (NOT (INT 2))
       (NOT (COMMON-DIVISOR N M 2)))))
 (0 ("FORALLE" ((:pds-term (OR (NOT (INT 2)) (NOT (COMMON-DIVISOR N M 2))))) (TERTIUM-NON-DATUR) "grounded"
     () ("NONEXISTENT" "EXISTENT")))
 )
(L105 (L105) (NOT (COMMON-DIVISOR N M 2))
 (0 ("HYP" () () "grounded" () ()))
 )
(L103 (L103) (NOT (INT 2))
 (0 ("HYP" () () "grounded" () ()))
 )
(L97 (L97) (COMMON-DIVISOR N M 2)
 (0 ("HYP" () () "grounded" () ()))
 )
(L96 (L96) (NOT (COMMON-DIVISOR N M 2))
 (0 ("HYP" () () "grounded" () ()))
 )
(L100 (L96 TERTIUM-NON-DATUR L67) FALSE
 (0 ("NOTE" () (L96 L71) "grounded" ()
     ("NONEXISTENT" "EXISTENT" "EXISTENT")))
 )
(L95 (TERTIUM-NON-DATUR) (OR (COMMON-DIVISOR N M 2) (NOT (COMMON-DIVISOR N M 2)))
 (0 ("FORALLE" ((:pds-term (COMMON-DIVISOR N M 2))) (TERTIUM-NON-DATUR) "grounded"
     () ("NONEXISTENT" "EXISTENT")))
 )
(L91 (L91) (INT 2)
 (0 ("HYP" () () "grounded" () ()))
 )
(L90 (L90) (NOT (INT 2))
 (0 ("HYP" () () "grounded" () ()))
 )
(L94 (L90 TERTIUM-NON-DATUR L67) FALSE
 (0 ("NOTE" () (L90 L70) "grounded" ()
     ("NONEXISTENT" "EXISTENT" "EXISTENT")))
 )
(L89 (TERTIUM-NON-DATUR) (OR (INT 2) (NOT (INT 2)))
 (0 ("FORALLE" ((:pds-term (INT 2))) (TERTIUM-NON-DATUR) "grounded"
     () ("NONEXISTENT" "EXISTENT")))
 )
(L29 (TERTIUM-NON-DATUR L10 L6 L12 L17 L27 L28 EVEN-COMMON-DIVISOR) FALSE
 (3 ("OTTER" ((:pds-nil)) (L10 L6 L12 L17 L27 L28 EVEN-COMMON-DIVISOR) "expanded"
     ()
     ("EXISTENT" "CLOSED" "CLOSED" "CLOSED" "CLOSED" "CLOSED" "CLOSED" "CLOSED"))
  ("OTTER" ((:pds-nil)) (L10 L6 L1 L17 L27 L28 L2) "expanded" ()
   ())
```

```
    ("simplify-goal" () (L3) "expanded" ()
     ("EXISTENT" "NONEXISTENT"))
    ("WEAKEN" () (L3) "grounded" () ("EXISTENT" "EXISTENT")))
   ))
(lemmata)
(agenda)
(controls
  (L67 (() () () ()))
  (L107 (() () () ()))
  (L104 (() () () ()))
  (L101 (() () () ()))
  (L108 (() () () ()))
  (L106 (() () () ()))
  (L102 (() () () ()))
  (L69 (() () () ()) ())
  (L71 (() () () ()) ())
  (L99 (() () () ()))
  (L98 (() () () ()))
  (L73 (() () () ()) ())
  (L70 (() () () ()) ())
  (L93 (() () () ()))
  (L92 (() () () ()))
  (L72 (() () () ()) ())
  (L10 (() () () ()))
  (L6 (() () () ()))
  (L12 (() () () ()))
  (L1 (() () () ()) ())
  (L80 (() () () ()))
  (L79 (() () () ()))
  (L77 (() () () ()))
  (L42 (() () () ()) ())
  (L43 (() () () ()))
  (L113 (() () () ()))
  (L112 (() () () ()))
  (L44 (() () () ()) (() () () ()) (() () () ()) ())
  (L17 (() () () ()))
  (L27 (() () () ()))
  (L33 (() () () ()))
  (L28 (() () () ()))
  (L76 (() () () ()))
  (L45 (() () () ()) ())
  (L65 (() () () ()))
  (L64 (() () () ()))
  (L30 (() () () ()) (() () () ()) ())
  (L34 (() () () ()))
  (L60 (() () () ()) ())
  (L59 (() () () ()) ())
  (L58 (() () () ()))
  (L35 (() () () ()) ())
  (L36 (() () () ()))
  (L55 (() () () ()) ())
  (L54 (() () () ()) ())
  (L53 (() () () ()))
  (L37 (() () () ()) ())
  (L38 (() () () ()))
  (L50 (() () () ()))
  (L48 (() () () ()))
  (L40 (() () () ()) ())
  (EVEN-COMMON-DIVISOR (() () () ()))
  (L88 (() () () ()))
  (L86 (() () () ()))
  (L83 (() () () ()))
  (L81 (() () () ()))
  (L78 (() () () ()))
  (L75 (() () () ()))
  (L74 (() () () ()))
  (L68 (() () () ()))
  (L63 (() () () ()))
  (L66 (() () () ()))
  (L62 (() () () ()))
  (L57 (() () () ()))
  (L61 (() () () ()))
  (L52 (() () () ()))
  (L56 (() () () ()))
  (L49 (() () () ()))
  (L47 (() () () ()))
  (L51 (() () () ()))
  (L41 (() () () ()))
  (L3 (() () () ()) ())
  (L2 (() () () ()) ())
```

```
  (L39 (() () () ()))
  (L85 (() () () ()))
  (L84 (() () () ()))
  (L82 (() () () ()))
  (L46 (() () () ()) ())
  (L31 (() () () ()) (() () () ()) ())
  (L32 (() () () ()))
  (TERTIUM-NON-DATUR (() () () ()))
  (L111 (() () () ()))
  (L110 (() () () ()))
  (L114 (() () () ()))
  (L109 (() () () ()))
  (L105 (() () () ()))
  (L103 (() () () ()))
  (L97 (() () () ()))
  (L96 (() () () ()))
  (L100 (() () () ()))
  (L95 (() () () ()))
  (L91 (() () () ()))
  (L90 (() () () ()))
  (L94 (() () () ()))
  (L89 (() () () ()))
  (L29 (() () () ()) () (() () () ()) ()))
(plan-steps
  (L29 0 L3 0 L32 0 L31 0 L46 0 L40 0 L82 0 L48 0 L84 0 L50 0 L38 0 L81 0 L85 0 L49 0 L51 0 L37 0 L39 0
   L83 0 L47 0 L53 0 L2 0 L55 0 L56 0 L41 0 L36 0 L52 0 L54 0 L35 0 L36 0 L58 0 L35 0 L60 0 L61 0 L58 0
   L34 0 L57 0 L59 0 L60 0 L61 0 L30 0 L33 0 L34 0 L57 0 L59 0 L65 0 L64 0 L44 0 L30 0 L33 0 L34 0 L62 0
   L66 0 L113 0 L112 0 L109 0 L65 0 L64 0 L44 0 L30 0 L33 0 L45 0 L63 0 L114 0 L111 0 TERTIUM-NON-DATUR 0
   L62 0 L66 0 L113 0 L112 0 L109 0 L65 0 L64 0 L44 0 L76 0 L86 0 L110 0 L45 0 L63 0 L114 0 L111 0
   TERTIUM-NON-DATUR 0 L62 0 L66 0 L113 0 L112 0 L109 0 L75 0 L88 0 L76 0 L86 0 L110 0 L45 0 L63 0 L114 0
   L111 0 TERTIUM-NON-DATUR 0 L68 0 L75 0 L88 0 L76 0 L86 0 L110 0 L43 0 L74 0 L68 0 L75 0 L88 0 L42 0
   L73 0 L72 0 L43 0 L74 0 L68 0 L77 0 L99 0 L98 0 L95 0 L93 0 L92 0 L89 0 L42 0 L73 0 L72 0 L43 0 L74 0
   L79 0 L100 0 L97 0 L94 0 L91 0 TERTIUM-NON-DATUR 0 L77 0 L99 0 L98 0 L95 0 L93 0 L92 0 L89 0 L42 0
   L73 0 L72 0 L1 0 L80 0 L71 0 L96 0 L70 0 L90 0 L79 0 L100 0 L97 0 L94 0 L91 0 TERTIUM-NON-DATUR 0
   L77 0 L99 0 L98 0 L95 0 L93 0 L92 0 L89 0 L78 0 L69 0 L1 0 L80 0 L71 0 L96 0 L70 0 L90 0 L79 0 L100 0
   L97 0 L94 0 L91 0 TERTIUM-NON-DATUR 0 L102 0 L101 0 L78 0 L69 0 L1 0 L80 0 L71 0 L96 0 L70 0 L90 0
   L106 0 L104 0 L102 0 L101 0 L78 0 L69 0 L108 0 L67 0 L107 0 L106 0 L104 0 L102 0 L101 0 L105 0 L103 0
   L108 0 L67 0 L107 0 L106 0 L104 0 L105 0 L103 0 L108 0 L67 0 L107 0 L105 0 L103 0 L10 0 L6 0 L12 0
   L17 0 L27 0 L28 0 EVEN-COMMON-DIVISOR 0) ))
```

D ΩMEGA's Knowledge Base

ΩMEGA's knowledge base is hierachically structured. A theory with name ⟨*theory*⟩ comprises definitions (given in a file ⟨*theory*⟩`.thy`), lemmata and theorems (file ⟨*theory*⟩`-theorems.thy`), proof problems (file ⟨*theory*⟩`-problems.thy`), inference rules (file ⟨*theory*⟩`-rules.thy`), proof tactics (file ⟨*theory*⟩`-tactics.thy`), proof methods (file ⟨*theory*⟩`-methods.thy`), and Ω-ANTS agents (file ⟨*theory*⟩`-agents.thy`).

Here, we present only the files `real.thy`, `real-theorems.thy`, `rational.thy`, `rational-theorems.thy`, `integer.thy`, `integer-theorems.thy`, `natural.thy`, and `natural-theorems.thy`.

D.1 Theory `Real`

D.1.1 `real.thy`

```
(th~deftheory REAL
              (uses rational sequences)
              (constants (completion ((struct num) (struct num))))
              (help "Peano Arithmetic for real numbers."))

(th~defconstant real
               (in real)
               (type (o num))
               (sort))

(th~defdef real\0
           (in real)
           (sort)
           (definition (setminus real (singleton zero)))
           (help "The set of reals without 0."))


(th~defdef real-struct
           (in real)
           (definition (completion rat-struct))
           (help "The real numbers, defined as the completion of the rational numbers."))

(th~defaxiom real-plus-closed
             (in real)
             (formula (closed-under real plus))
             (help "Plus is closed."))

(th~defaxiom real-times-closed
             (in real)
             (formula (closed-under real times))
             (help "Times is closed."))

(th~defaxiom real-plus-assoc
             (in real)
             (formula (associative real plus))
             (help "Plus is assoc."))

(th~defaxiom real-times-assoc
             (in real)
             (formula (associative real times))
             (help "Times is assoc."))

(th~defaxiom real-plus-commu
             (in real)
             (formula (commutative real plus))
             (help "Plus is commu."))

(th~defaxiom real-times-commu
             (in real)
             (formula (commutative real times))
             (help "Times is commu."))

(th~defaxiom real-plus-times-distrib
             (in real)
             (formula (distributive real plus times))
             (help "Distributivity for plus and times."))

(th~defaxiom real-plus-unit
             (in real)
```

```
             (formula (unit real plus zero))
             (help "Zero is additive unit element."))

(th~defaxiom real-times-unit
             (in real)
             (formula (unit real times one))
             (help "One is multiplicative unit element."))

(th~defaxiom real-plus-inv
             (in real)
             (formula (inverse-exist real plus zero))
             (help "Existence of inverse elements."))

(th~defaxiom real-times-inv
             (in real)
             (formula (inverse-exist real\0 times one))
             (help "Existence of inverse elements."))

(th~defaxiom real-trichotomy
             (in real)
             (formula (trichotomy real less))
             (help "Trichotomy for reals."))

(th~defaxiom real-less-trichotomy
             (in real)
             (formula (trichotomy real less))
             (help "Trichotomy for reals."))

(th~defaxiom real-less-transitive
             (in real)
             (formula (transitivity real less))
             (help "Less is transitive for reals."))

(th~defaxiom real-less-times-mono
             (in real)
             (formula (monotone less times real\0))
             (help "Less is monotone for times."))

(th~defaxiom real-less-plus-mono
             (in real)
             (formula (monotone less plus real))
             (help "Less is monotone for plus."))

(th~defaxiom real-complete
             (in real)
             (formula
              (forall (lam (xx (o num))
                           (forall (lam (yy (o num))
                                        (implies (and (and (not (empty xx))(not (empty yy)))
                                                      (and (= real (union xx yy))
                                                           (forall-sort (lam (x num)
                                                                             (forall-sort (lam (y num)
                                                                                               (less x y))
                                                                                          yy))
                                                                        xx)))
                                                 (exists-sort (lam (t num)
                                                                   (forall-sort (lam (x num)
                                                                                     (forall-sort
                                                                                      (lam (y num)
                                                                                           (and (leq x t)
                                                                                                (leq t y)))
                                                                                      yy))
                                                                                xx))
                                                              real)))))))
             (help "Completeness of the reals."))


(th~defaxiom rat-real
             (in real)
             (formula
              (forall-sort (lam (x num) (real x)) rat))
             (termdecl)
             (help "All rational numbers are reals."))

(th~defdef closed-interval-with-bounds
           (in real)
           (definition
             (lam (G (o num))
                  (lam (l num)
                       (lam (r num)
```

```
                         (forall (lam (x num)
                                        (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
             (lam (G (o num))
                  (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
             (lam (G (o num))
                  (lam (l num)
                       (lam (r num)
                            (forall (lam (x num)
                                          (forall (lam (x num)
                                                       (equiv (and (less x r) (greater x l))
                                                              (in x G))))))))))
           (help "Predicate for open intervals of real numbers."))

(th~defdef open-interval
           (in real)
           (definition
             (lam (G (o num))
                  (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
             (lam (x num)
                  (lam (y num)
                       (lam (z num)
                            (and (leq z x) (geq x y))))))
           (help "The closed interval for given bounds."))


(th~defdef open-interval-bounds
           (in real)
           (definition
             (lam (x num)
                  (lam (y num)
                       (lam (z num)
                            (and (less z x) (greater x y))))))
           (help "The open interval for given bounds."))



(th~defdef interval-center
           (in real)
           (definition
             (lam (I1 (o num))
                  (divide (minus (supremum real-struct I1)
                                 (infimum real-struct I1))
                          (s one))))
           (help "The center of an interval."))


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

D.1.2 real-theorems.thy

```
(th~deftheorem plus-real-step2
               (in real)
               (conclusion
                (forall (lam (x num)
                             (forall (lam (y num)
                                          (implies (and (in x real) (in y real))
                                                   (= (plus x (s y)) (s (plus x y)))))))))
               (help "The recursive definition of plus on the right term."))

(th~deftheorem plus-real-closed
               (in real)
               (conclusion
                (forall (lam (x num)
                             (forall (lam (y num)
                                          (implies (and (in x real) (in y real))
                                                   (in (plus x y) real))))))))


(th~deftheorem a-plus-real
               (in real)
               (conclusion
                (forall (lam (x num)
                             (forall (lam (y num)
                                          (forall (lam (z num)
                                                       (implies (and (in x real) (and (in y real) (in z real)))
                                                                (= (plus (plus x y) z)
                                                                   (plus x (plus y z))))))))))))

(th~deftheorem c-plus-real
               (in real)
               (conclusion
                (forall (lam (x num)
                             (forall (lam (y num)
                                          (implies (and (in x real) (in y real))
                                                   (= (plus x y) (plus y x)))))))))

(th~deftheorem 0-plus-real
               (in real)
               (conclusion (forall (lam (x num) (implies (in x real) (= (plus 0 x) x))))))

(th~deftheorem 1-times-real
               (in real)
               (conclusion (forall (lam (x num) (implies (in x real) (= (times 1 x) x))))))

(th~deftheorem 0-times-real
               (in real)
               (conclusion (forall (lam (x num) (implies (in x real) (= (times 0 x) 0))))))

(th~deftheorem a-times-real
               (in real)
               (conclusion
                (forall (lam (x num)
                             (forall (lam (y num)
                                          (forall (lam (z num)
                                                       (implies (and (in x real) (and (in y real) (in z real)))
                                                                (= (times (times x y) z)
                                                                   (times x (times y z))))))))))))

(th~deftheorem c-times-real
               (in real)
               (conclusion
                (forall (lam (x num)
                             (forall (lam (y num)
                                          (implies (and (in x real) (in y real))
                                                   (= (times x y) (times y x)))))))))


(th~deftheorem Dist-Right-real
               (in real)
               (conclusion
                (forall (lam (x num)
                             (forall (lam (y num)
                                          (forall (lam (z num)
                                                       (implies (and (in x real) (and (in y real) (in z real)))
                                                                (= (times (plus x y) z)
                                                                   (plus (times x z) (times y z))))))))))))

(th~deftheorem Dist-Left-real
```

```
               (in real)
               (conclusion
                (forall (lam (x num)
                             (forall (lam (y num)
                                          (forall (lam (z num)
                                                       (implies (and (in x real) (and (in y real) (in z real)))
                                                                (= (times x (plus y z))
                                                                   (plus (times x y) (times x z)))))))))))

(th~deftheorem minus2plus
               (in real)
               (conclusion
                (forall (lam (x num)
                             (forall (lam (y num)
                                          (implies (and (in x real) (in y real))
                                                   (= (minus x y) (plus x (times -1 y))))))))))

(th~deftheorem power-1-real
               (in real)
               (conclusion (forall (lam (x num) (implies (in x real) (= (power x 1) x))))))

(th~deftheorem 1-power-real
               (in real)
               (conclusion (forall (lam (x num) (implies (in x real) (= (power 1 x) 1))))))

(th~deftheorem 0-power-real
               (in real)
               (conclusion (forall (lam (x num) (implies (in x real) (= (power 0 x) 0))))))

(th~deftheorem power-0-real
               (in real)
               (conclusion (forall (lam (x num) (implies (in x real) (= (power x 0) 1))))))

(th~deftheorem times-power-real
               (in real)
               (conclusion
                (forall (lam (x num)
                             (forall (lam (y num)
                                          (forall (lam (z num)
                                                       (implies (and (in x real) (and (in y real) (in z real)))
                                                                (= (power x (plus y z))
                                                                   (times (power x y) (power x z)))))))))))

(th~deftheorem times-power-2-real
               (in real)
               (conclusion
                (forall (lam (x num)
                             (forall (lam (y num)
                                          (forall (lam (z num)
                                                       (implies (and (in x real) (and (in y real) (in z real)))
                                                                (= (power (times x y) z)
                                                                   (times (power x z) (power y z)))))))))))

(th~deftheorem power-power-real
               (in real)
               (conclusion
                (forall (lam (x num)
                             (forall (lam (y num)
                                          (forall (lam (z num)
                                                       (implies (and (in x real) (and (in y real) (in z real)))
                                                                (= (power (power x y) z)
                                                                   (power x (times y z)))))))))))

(th~deftheorem rat-criterion
               (in rational)
               (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 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))
                               int\0)))
           (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))
                           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)
```

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


(th~defdef one-over
           (in rational)
           (definition
             (lam (x num)
                  (frac (denominator x)
                        (numerator x))))
           (help "The reciprocal value of a fraction."))

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


(th~defaxiom plus-frac
             (in rational)
             (formula
              (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 "The axiom for plus on the rationals."))



(th~defaxiom times-frac
             (in rational)
             (formula
              (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 "The axiom for times on the rationals."))


(th~defaxiom rat-mul-struct
             (in rational)
             (formula
              (and (= (struct-set rat-mul-struct) rat)
                   (= (struct-op rat-mul-struct) times)))
             (help "The group of Rationals with operation times."))



(th~defaxiom rat-struct
             (in rational)
             (formula
              (and (and (= (struct-set rat-struct) rat)
                        (= (struct-op rat-struct) plus))
                   (and (= (struct-mul-sgroup rat-struct) rat-mul-struct)
                        (= (struct-ordering rat-struct) leq))))
             (help "The ordered field of rationals with operation plus."))

(th~defdef pdivide
           (in rational)
           (definition
            (apply-pointwise-2 divide))
           (help "The definition of pointwise addition of functions."))


(th~defaxiom int-rat
             (in rational)
             (formula (forall-sort (lam (x num) (rat x)) int))
             (help "An integer is rational."))
```

D.2.2 rational-theorems.thy

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

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

(th~deftheorem numerator-equals-zero
              (in rational)
              (conclusion
               (forall-sort (lam (x num)
                                 (implies (less zero x)
                                          (= (frac zero x) zero)))
                            nat))
              (help "A rational number with numerator zero is the integer zero."))

(th~deftheorem int-to-rat
              (in rational)
              (conclusion
               (forall-sort (lam (x num)
                                 (= x (frac x (s zero))))
                             int))
              (help "Conversion of integers to fractions."))

(th~deftheorem rat-to-int
              (in rational)
              (conclusion
               (forall-sort (lam (x num)
                                 (= (frac x (s zero)) x))
                             int))
              (help "Conversion of fractions with denominator one to integers."))

(th~deftheorem numerator-of-frac
              (in rational)
              (conclusion
               (forall-sort (lam (x num)
                                 (forall-sort (lam (y num)
                                                   (implies (rat (frac x y))
                                                            (= (numerator (frac x y)) x)))
                                              nat))
                            int))
              (help "Extraction of the numerator of a rational number."))

(th~deftheorem denominator-of-frac
              (in rational)
              (conclusion
               (forall-sort (lam (x num)
                                 (forall-sort (lam (y num)
                                                   (implies (rat (frac x y))
                                                            (= (denominator (frac x y)) y)))
                                              nat))
                            int))
              (help "Extraction of the denominator of a rational number."))

(th~deftheorem 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."))

(th~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."))

(th~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."))

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

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

(th~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."))

(th~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."))

(th~deftheorem power-rat-nat
                (in rational)
                (conclusion
                 (forall-sort (lam (x num)
                                  (forall-sort (lam (y num)
```

```
                                                (= (power x y)
                                                   (frac (power (numerator x) y)
                                                         (power (denominator x) y))))
                                           nat))
                         rat))
              (help "Natural powers of rational numbers."))

(th~deftheorem power-rat-nnat
              (in rational)
              (conclusion
               (forall-sort (lam (x num)
                                 (forall-sort (lam (y num)
                                                   (= (power x y)
                                                      (one-over (power x (change-sign y)))))
                                              nnat))
                            rat))
              (help "Negative whole-numbered powers of rational numbers."))

(th~deftheorem power-rat-base-one
              (in rational)
              (conclusion
               (forall-sort (lam (x num)
                                 (= (power (s zero) x)
                                    (s zero)))
                            int))
              (help "Powers with base one."))

(th~deftheorem less-rat
              (in rational)
              (conclusion
               (forall-sort (lam (x num)
                                 (forall-sort (lam (y num)
                                                   (implies (less (times (numerator x) (denominator y))
                                                                  (times (numerator y) (denominator x)))
                                                            (less x y)))
                                              rat))
                            rat))
              (help "Less on rational numbers."))

(th~deftheorem less-rat-neg-and-pos
              (in rational)
              (conclusion
               (forall-sort (lam (x num)
                                 (forall-sort (lam (y num)
                                                   (implies (and (less (numerator x) zero)
                                                                 (less zero (numerator y)))
                                                            (less x y)))
                                              rat))
                            rat))
              (help "A negative rational number is smaller than a positive rational number."))

(th~deftheorem less-implies-leq-rat
              (in rational)
              (conclusion
               (forall-sort (lam (x num)
                                 (forall-sort (lam (y num)
                                                   (implies (less x y) (leq x y)))
                                              rat))
                            rat))
              (help "Less implies less or equal."))

(th~deftheorem equal-implies-leq-rat
              (in rational)
              (conclusion
               (forall-sort (lam (x num)
                                 (forall-sort (lam (y num)
                                                   (implies (= x y) (leq x y)))
                                              rat))
                            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."))
```

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

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

(th~defdef int
           (in integer)
           (sort)
           (definition (union nat nnat))
           (help "The set of integers, constructed as the naturals and their negatives."))

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

#+sorts(th~deftheorem int-sortlike
    (in integer)
    (conclusion conc (forall-sort (lam (x num) (or (int x) (not (int x)))) defined)))
#+sorts(th~defsort int
    (in integer)
    (by int-sortlike))

;(th~defaxiom pred-succ
;             (in integer)
;             (formula (forall-sort (lam (x num) (= x (p (s x))))  int))
;             (help "The predecessor of the successor of x is x."))
;
;(th~defaxiom succ-pred
;             (in integer)
;             (formula (forall-sort (lam (x num) (= x (s (p x))))  int))
;             (help "The predecessor of the successor of x is x."))


(th~defaxiom nat-int
             (in integer)
             (formula
              (forall-sort (lam (x num) (int x))
                           nat))
             (termdecl)
             (help "A natural number is whole-numbered."))

(th~defaxiom nnat-int
             (in integer)
             (formula
              (forall-sort (lam (x num) (int x))
                           nnat))
             (help "A nonpositive integer is whole-numbered."))

(th~defaxiom sp-int
             (in integer)
             (formula
              (forall-sort (lam (x num) (= (s (p x)) x))
                           int))
             (help "The successor of the predecessor of a number equals the number itself."))

(th~defaxiom ps-int
             (in integer)
             (formula
              (forall-sort (lam (x num) (= (p (s x)) x))
                           int))
             (help "The successor of the predecessor of a number equals the number itself."))


(th~defdef minus
           (in integer)
           (definition
             (lam (x num)
                  (lam (y num)
                       (plus x (change-sign y)))))
           (help "The difference operators on natural numbers."))
```

```
(th~defaxiom plus-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))))))))
                                          nnat))
                         nnat))
            (help "Extension of plus to the integers."))

(th~defaxiom times-int
            (in integer)
            (formula
             (forall-sort (lam (x num)
                              (forall-sort (lam (y num)
                                               (= (times (change-sign x) y)
                                                  (change-sign (times x y))))
                                          int))
                         int))
            (help "Extension of multiplication to the integers."))



(th~defaxiom int-mul-struct
            (in integer)
            (formula
             (and (= (struct-set int-mul-struct) int)
                  (= (struct-op int-mul-struct) times)))
            (help "The monoid of integers with operation times."))



(th~defaxiom int-struct
            (in integer)
            (formula
             (and (and (= (struct-set int-struct) int)
                       (= (struct-op int-struct) plus))
                  (and (= (struct-mul-sgroup int-struct) int-mul-struct)
                       (= (struct-ordering int-struct) leq))))
            (help "The ordered ring of integers with operation plus."))

(th~defdef div
          (in integer)
          (definition
            (lam (x num)
                 (lam (y num)
                      (that (lam (z num)
                                 (and (leq (times z y) x)
                                      (greater (times (s z) y) x)))))))
          (help "The div operator for natural numbers."))

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


(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."))



(th~defdef common-multiple
           (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 (divisor x z)
                                           (divisor y z))))))))
           (help "The predicate for common integer divisibility."))

(th~defdef gcd
           (in integer)
           (definition
             (lam (x num)
                  (lam (y num)
                       (maximum int-struct (lam (z num)
                                                 (common-divisor z x y ))))))
           (help "The predicate for common integer divisibility."))

(th~defdef lcm
           (in integer)
           (definition
             (lam (x num)
                  (lam (y num)
                       (minimum int-struct (lam (z num)
                                                 (common-multiple z x y ))))))
           (help "The predicate for common integer divisibility."))


(th~defdef mod
           (in integer)
           (definition
             (lam (x num)
                  (lam (y num)
                       (that (lam (z num)
                                  (= x (plus z (times y (div x y)))))))))
           (help "The mod operator for natural numbers."))


(th~defdef pminus
           (in integer)
           (definition
             (apply-pointwise-2 minus))
           (help "The definition of pointwise subtraction of functions."))


(th~defdef integer-intervall
           (in integer)
           (definition
             (lam (x num)
                  (lam (y num)
                       (lam (elem num)
                            (and (int elem)
                                 (and (leq x elem)
                                      (leq elem y)))))))
           (help "The set of all integers in the closed intervall from x to y."))

(th~defdef common-divisor-p
           (in integer)
           (definition
             (lam (x  num)
                  (lam (y num)
                       (exists-sort (lam (z num)
                                         (common-divisor x y z))
                                    int))))
           (help "Definition of the property of having a common divisor."))

(th~defdef evenp
           (in integer)
           (definition
             (lam (x  num)
                  (exists-sort (lam (y num)
                                    (= x (times 2 y)))
```

```
                              int)))
           (help "Definition of even."))
```

D.3.2 integer-theorems.thy

```
(th~deftheorem neutral-of-Nat
               (in integer)
               (conclusion
                (= zero (struct-neut nat-plus-struct)))
  (help "The constant zero, is the neutral element of NatPlus."))

(th~defsimplifier neutral-of-Nat-simp
                  (in integer)
                  (status global)
                  (equation neutral-of-Nat)
                  (direction rl)
  (help "Simplify the neutral of the nat-struct."))

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

(th~deftheorem assoc-plus-Nat
               (in integer)
               (conclusion THM (associative nat plus)))

(th~deftheorem assoc-times-Nat
               (in integer)
               (conclusion THM (associative nat times)))

(th~deftheorem commutative-plus-Nat
               (in integer)
               (conclusion THM (commutative nat plus)))

(th~deftheorem commutative-times-Nat
               (in integer)
               (conclusion THM (commutative nat times)))

(th~deftheorem closed-plus-Nat
               (in integer)
               (conclusion THM (closed-under-2 nat plus)))

(th~deftheorem closed-times-Nat
               (in integer)
               (conclusion THM (closed-under-2 nat times)))


;(th~deftheorem semigroup-NatPlus
;               (in integer)
;               (conclusion THM (semigroup Nat-Plus-struct)))
;
;(th~deftheorem semigroup-times-Nat
;               (in integer)
;               (conclusion THM (semigroup Nat-Times-struct)))
;
;(th~deftheorem monoid-NatPlus
;               (in integer)
;               (conclusion THM (monoid Nat-Plus-struct)))
;
;(th~deftheorem monoid-times-Nat
;               (in integer)
;               (conclusion THM (monoid Nat-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-times-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 (unit int plus zero)))

(th~deftheorem neutral-times-Int
               (in integer)
               (conclusion THM (unit int times (s zero))))

;(th~deftheorem monoid-plus-Int
;               (in integer)
;               (conclusion THM (monoid int-struct)))
;
;(th~deftheorem monoid-times-Int
;               (in integer)
;               (conclusion THM (monoid int-mul-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 (= (change-sign zero) zero))
  (help "Zero is the negative of zero."))


(th~defsimplifier neg-zero
                  (in integer)
                  (status global)
                  (equation neg-zero)
                  (direction lr)
  (help "Simplify  - 0 to 0."))

(th~deftheorem neg-nilpotent
               (in integer)
               (conclusion (nilpotent int change-sign))
  (help "The function for changing signs of integers is nilpotent."))



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

(th~deftheorem plus-int-base2
               (in integer)
               (conclusion
                (forall-sort (lam (x num)
                                  (= (plus zero x) x))
                             int))
```

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

(th~deftheorem plus-int-step-s
              (in integer)
              (conclusion
               (forall-sort (lam (x num)
                                 (forall-sort (lam (y num)
                                                   (= (plus x (s y)) (s (plus x y))))
                                              nat))
                            int))
  (help "The step case for recursive definition of addition."))

(th~deftheorem plus-int-step-p
              (in integer)
              (conclusion
               (forall-sort (lam (x num)
                     (forall-sort (lam (y num)
                                       (= (plus x (p y)) (p (plus x y))))
                                  nnat))
                            int))
  (help "Another step case for recursive definition of addition."))

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

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

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

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

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

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

(th~deftheorem times-int-base
              (in integer)
              (conclusion
               (forall-sort (lam (x num)
                                 (= (times x zero) zero))
                            int))
```

```
  (help "The base case for recursive definition of multiplication."))

(th~deftheorem times-int-base2
               (in integer)
               (conclusion
                (forall-sort (lam (x num)
                                  (= (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 (lam (x num)
                                  (forall-sort (lam (y num)
                                                    (= (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 (lam (x num)
                                  (forall-sort (lam (y num)
                                                    (= (times x (p y)) (plus (change-sign x) (times x y))))
                                               nnat))
                             int))
  (help "Another step case for recursive definition of multiplication."))

(th~deftheorem times-int-step2-s
               (in integer)
               (conclusion
                (forall-sort (lam (y num)
                                  (forall-sort (lam (x num)
                                                    (= (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 (lam (y num)
                                  (forall-sort (lam (x num)
                                                    (= (times (p x) y) (plus (change-sign y) (times x y))))
                                               nnat))
                             int))
  (help "Another step case for recursive definition of multiplication."))

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

(th~deftheorem mod-int
               (in integer)
               (conclusion
                (forall-sort (lam (x num)
                                  (forall-sort (lam (y num)
                                                    (implies (less zero y)
                                                             (= (mod x y)
                                                                (minus x (times y (div x y))))))
                                               nat))
                             int))
  (help "Residue of whole-numbered division."))
```

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

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

(th~deftheorem gcd-left-arg-zero
              (in integer)
              (conclusion
               (forall-sort (lam (x num)
                                 (= (gcd zero x)
                                    x))
                            nat))
  (help "A base case of the Euclidian algorithm."))

(th~deftheorem gcd-right-arg-zero
              (in integer)
              (conclusion
               (forall-sort (lam (x num)
                                 (= (gcd x zero)
                                    x))
                            nat))
  (help "Another base case of the Euclidian algorithm."))

(th~deftheorem gcd-equal-args
              (in integer)
              (conclusion
               (forall-sort (lam (x num)
                                 (= (gcd x x)
                                    x))
                            nat))
  (help "Another base case of the Euclidian algorithm."))

(th~deftheorem gcd-neg-left-arg
              (in integer)
              (conclusion
               (forall-sort (lam (x num)
                                 (forall-sort (lam (y num)
                                                   (= (gcd x y)
                                                      (gcd (change-sign x) y)))
                                              int))
                            nnat))
  (help "Greatest common divisor with negative arguments."))

(th~deftheorem gcd-neg-right-arg
              (in integer)
              (conclusion
               (forall-sort (lam (x num)
                                 (forall-sort (lam (y num)
                                                   (= (gcd x y)
                                                      (gcd x (change-sign y))))
                                              nnat))
                            int))
  (help "Greatest common divisor with negative arguments."))

(th~deftheorem gcd-diff-1
              (in integer)
              (conclusion
               (forall-sort (lam (x num)
                                 (forall-sort (lam (y num)
                                                   (= (gcd x y)
                                                      (gcd (minus x y) y)))
                                              nat))
                            nat))
  (help "The step case of the Euclidian algorithm."))

(th~deftheorem gcd-diff-2
```

```
               (in integer)
               (conclusion
                (forall-sort (lam (x num)
                                  (forall-sort (lam (y num)
                                                    (= (gcd x y)
                                                       (gcd x (minus y x))))
                                               nat))
                             nat))
  (help "Another step case of the Euclidian algorithm."))

(th~deftheorem lcm-left-arg-zero
               (in integer)
               (conclusion
                (forall-sort (lam (x num)
                                  (= (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 num)
                                  (= (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 num)
                                  (= (lcm x x)
                                     x))
                             nat))
  (help "Another base case for the least common multiple."))

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

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

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

(th~deftheorem less-nnat-base
               (in integer)
               (conclusion
                (forall-sort (lam (x num)
                                  (less (p x) zero))
                             nnat))
  (help "The base case for recursive definition of less."))
```

```
(th~deftheorem less-nnat-step
               (in integer)
               (conclusion
                (forall-sort (lam (x num)
                                  (forall-sort (lam (y num)
                                                    (implies (less x y) (less (p x) (p y))))
                                               nnat))
                             nnat))
  (help "The step case for recursive definition of less."))

(th~deftheorem less-implies-leq-int
               (in integer)
               (conclusion
                (forall-sort (lam (x num)
                                  (forall-sort (lam (y num)
                                                    (implies (less x y) (leq x y)))
                                               int))
                             int))
  (help "Less implies less or equal."))

(th~deftheorem equal-implies-leq-int
               (in integer)
               (conclusion
                (forall-sort (lam (x num)
                                  (forall-sort (lam (y num)
                                                    (implies (= x y) (leq x y)))
                                               int))
                             int))
  (help "Equal implies less or equal."))

(th~deftheorem neg-less-pos-int
               (in integer)
               (conclusion
                (forall-sort (lam (x num)
                                  (forall-sort (lam (y num)
                                                    (less (p x) (s y)))
                                               nat))
                             nnat))
  (help "A negative integer is smaller than a positive integer."))

(th~deftheorem power-closed-int
               (in integer)
               (conclusion
                (forall-sort (lam (y num)
                                  (forall-sort (lam (x num)
                                                    (int (power x y)))
                                               int))
                             nat))
  (help "The power is closed on integers."))

(th~deftheorem even-on-integers
               (in integer)
               (conclusion
                (forall-sort (lam (x num)
                                  (equiv (evenp x)
                                         (exists-sort (lam (y num)
                                                           (= x (times 2 y))) int))) int))
  (help "An integer x is even, iff an integer y exists so that x=2*x."))

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



(th~deftheorem 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."))
```

```
(th~deftheorem power-int-closed
               (in integer)
               (conclusion
                (forall-sort (lam (x num)
                                 (forall-sort (lam (y num)
                                                   (int (power y x)))  int))  int))
  (help "The set of integers is closed under power."))
```

D.4 Theory `Natural`

D.4.1 `natural.thy`

```
(th~deftheory natural
              (uses poset function struct)
              (help "Peano Arithmetic for naturals."))

(th~deftype num
            (in natural)
            (arguments 0)
            (help "The type of number objects, like natural nubmers, 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 num))
                (sort)
                (help "The set of natural numbers"))

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

(th~defconstant pos-Nat
                (in natural)
                (type (o num))
                (sort)
                (help "The set of natural numbers"))

(th~defconstant NNat
                (in natural)
                (type (o num))
                (sort)
                (help "The set of negative natural numbers"))

(th~defconstant neg-NNat
                (in natural)
                (type (o num))
                (sort)
                (help "The set of negative natural numbers"))


(th~defconstant iterate-warg
                (in natural)
                (type (all-types bb (bb bb num (bb bb num))))
                (help "An interatiton Combinator for the natural numbers"))

(th~defconstant change-sign
                (in natural)
                (type (num num))
                (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 num))
                (help "The structure of natural numbers with times as an operation"))

;(th~defdef nat-iterate
;           (in natural)
;           (definition
;             (lam (n num)
;                  ((that (lam (P (o num))
;                    (and (and (and (and (P zero)
;                                        (forall-sort (lam (x num) (P (s X))) P ))
;                                   (injective P s))
;                              (forall-sort (lam (x num) (not (= zero (s x)))) P))
;                         (forall (lam (Q (o num))
;                          (implies (and (Q zero)
;                                        (forall-sort (lam (n num)
;                                                          (implies (Q n)
;                                                                   (Q (s n))))
;                                                     P))
;                                   (forall-sort (lam (n num) (Q n))
;                                                P)))))))
;                   n))))
;
;
; Peano's axioms for natural numbers
;


(th~defaxiom total-nat         ;;; from: (sort nat)
             (in natural)
             (formula (forall-sort (lam (x num) (defined (nat x))) defined))
             (termdecl)
             (help "The predicate Nat is defined everywhere."))

(th~deftheorem subsort-nat-defined         ;;;from sort-defined
             (in natural)
             (conclusion (forall-sort (lam (x num) (defined x)) nat))
             (termdecl)
             (help "The predicate Nat is defined everywhere."))


(th~defaxiom zero-nat
             (in natural)
             (formula
              (nat zero))
             (termdecl)
             (help "Zero is a natural number."))

(th~defaxiom succ-nat
             (in natural)
             (formula
              (forall-sort
               (lam (x num)
                    (nat (s x)))
               nat))
             (termdecl)
             (help "The successor of a natural number is natural."))

(th~defaxiom nat-inj-succ
                (in natural)
             (formula (injective nat s))
             (help "The successor function is injective."))

(th~defaxiom nat-no-pred-zero
             (in natural)
             (formula (forall-sort
                       (lam (X num)
                            (not (= (s X) zero)))
                      Nat))
             (help "Zero has no predecessor."))

(th~defaxiom nat-induction
             (in natural)
             (formula
              (forall (lam (Q (o num))
                           (implies (and (subset Q Nat)
                                         (and (in zero Q)
                                              (closed-under-1 Q s)))
```

```
                                    (= Nat Q)))))
             (help "The induction axiom for natural numbers."))

(th~defaxiom nat-induct
             (in natural)
             (formula
              (forall (lam (Q (o num))
                           (implies (and (Q zero)
                                         (forall-sort (lam (n num)
                                                           (implies (Q n)
                                                                    (Q (s n))))
                                                      Nat))
                                    (forall-sort (lam (n num) (Q n))
                                                 Nat)))))
             (help "The induction axiom."))

;
; Miscellaneous
;

(th~defdef one
           (in natural)
           (definition (s zero))
           (help "The number 1 defined as the successor of 0."))

#-struct-3(th~defsimplifier one-simp
                  (in natural)
                  (status global)
                  (equation one)
                  (direction lr)
                  (help "Simplify the number one."))


(th~defdef two
           (in natural)
           (definition (s one))
           (help "The number 2 defined as the successor of 1."))

(th~defdef three
           (in natural)
           (definition (s two))
           (help "The number 2 defined as the successor of 1."))

(th~defdef four
           (in natural)
           (definition (s three))
           (help "The number 2 defined as the successor of 1."))

(th~defdef five
           (in natural)
           (definition (s four))
           (help "The number 2 defined as the successor of 1."))

(th~defdef six
           (in natural)
           (definition (s five))
           (help "The number 2 defined as the successor of 1."))

(th~defdef seven
           (in natural)
           (definition (s six))
           (help "The number 2 defined as the successor of 1."))

(th~defdef eight
           (in natural)
           (definition (s seven))
           (help "The number 2 defined as the successor of 1."))

(th~defdef nine
           (in natural)
           (definition (s eight))
           (help "The number 2 defined as the successor of 1."))

(th~defdef ten
           (in natural)
           (definition (s nine))
           (help "The number 2 defined as the successor of 1."))


(th~defdef p
```

```
           (in natural)
           (definition
              (lam (n num) (that (lam (m num) (= (s m) n)))))
           (help "Predecessor function."))

;
; Definitions of the order relations
;

(th~defdef leq
           (in natural)
           (definition
             (lam (m num)
                  (lam (n num)
                       (forall (lam (Q (o num))
                                    (implies (and (in m Q)
                                                  (forall (lam (l num)
                                                               (implies (in l Q)
                                                                        (in (s l) Q)))))
                                             (in n Q)))))))
                 (help "The classical less-or-equal operator on the natural numbers."))

(th~defdef less
           (in natural)
           (definition
             (lam (x num)
                  (lam (y num)
                       (and (leq x y)
                            (not (= x y))))))
           (help "The less predicate."))

(th~defdef greater
           (in natural)
           (definition
             (lam (x num)
                  (lam (y num)
                       (less  y x))))
           (help "The greater predicate."))

(th~defdef geq
           (in natural)
           (definition
             (lam (x num)
                  (lam (y num)
                       (leq y x))))
           (help "The greater-or-equal predicate."))

;
; Arithmetic operations defined through the recursion operator
;

(th~defdef recursion-poly
           (in natural)
           (type-variables cc)
           (definition
             (lam (h (cc cc num))
                  (lam (g cc)
                       (lam (n num)
                            (that (lam (m cc)
                                       (forall (lam (U (o cc num))
                                                    (implies
                                                     (and (U zero g)
                                                          (forall (lam (y cc)
                                                                       (forall (lam (x num)
                                                                                    (implies (U x y)
                                                                                             (U (s x)
                                                                                                (h x y))))))))
                                                     (U n m))))))))))
           (help "A polymorphic version of the recursion operator."))

(th~defdef recursion
           (in natural)
           (definition
             (lam (h ((num num) num))
                  (lam (g num)
                       (lam (n num)
                            (that (lam (m num)
                                       (forall (lam (U (o num num))
                                                    (implies
                                                     (and (U zero g)
```

```
                                                        (forall (lam (y num)
                                                                     (forall (lam (x num)
                                                                                  (implies
                                                                                   (U x y)
                                                                                   (U (s x) (h x y))))))))
                                                   (U n m)))))))))
           (help "The recursion operator."))

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

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

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

(th~defdef iterate
           (in natural)
           (type-variables bb)
           (definition
             (lam (F (bb bb))
                  (lam (n num)
                       (recursion-poly (lam (n num) (compose-functions F)) (lam (x bb) x) n))))
           (help "The iteration operator."))



(th~defdef pos-nat
           (in natural)
           (definition
             (lam (x num) (and (in x nat) (not (= x zero)))))
           (help "The set of positive natural numbers."))


(th~deftheorem total-pos-nat
               (in natural)
               (conclusion conc (forall (lam (x num) (defined (pos-nat x)))))
               (help "The predicate Nat is defined everywhere."))


(th~defdef NNat
           (in natural)
           (definition
             (lam (x num) (or (= x zero) (exists-sort (lam (y num) (= (s x) y))  NNat))))
           (help "The set of positive natural numbers."))

(th~deftheorem total-nnat
               (in natural)
               (conclusion conc (forall (lam (x num) (defined (nnat x)))))
               (help "The predicate Nat is defined everywhere."))

(th~defaxiom zero-nnat
             (in natural)
             (formula
              (nnat zero))
             (termdecl)
             (help "Zero is a negative natural number."))

(th~defaxiom pred-nnat
             (in natural)
             (formula
              (forall-sort (lam (x num)
                                (nnat (p x)))
                           nnat))
             (termdecl)
             (help "The predecessor of a negative natural number is a negative natural number."))

(th~defaxiom nnat-closed
```

```
            (in natural)
            (formula (closed-under-1 nnat p))
            (help "The set of neg-natural numbers is closed under successors."))


(th~defaxiom nnat-inj-pred
            (in natural)
            (formula (injective nnat p))
            (help "The successor function is injective."))

(th~defaxiom nnat-no-suc-zero
            (in natural)
            (formula
             (forall-sort (lam (X num)
                               (not (= (s X) zero)))
                          Nnat))
            (help "Zero has no successor in NNat."))

(th~defaxiom nnat-induction
            (in natural)
            (formula
             (forall (lam (Q (o num))
                          (implies (and (in zero Q)
                                        (closed-under-1 Q p))
                                   (subset nnat Q)))))
            (help "The induction axiom for neg-natural numbers."))


(th~defdef neg-NNat
          (in natural)
          (definition (lam (x num) (and (in x NNat) (not (= x zero)))))
          (help "The set of negative Nnats."))


(th~defaxiom change-sign
            (in natural)
            (formula
             (and (= (change-sign zero) zero)
                  (and (forall-sort (lam (x num)
                                         (= (change-sign (s x)) (p (change-sign x))))
                                    Nat)
                       (forall-sort (lam (x num)
                                         (= (change-sign (p x)) (s (change-sign x))))
                                    NNat))))
            (help "The negative operator on Natural numbers."))

;;#{Ordering properties of the natural numbers}#

(th~defaxiom leq-nat
            (in natural)
            (formula
             (and (and (forall-sort (lam (y num) (leq zero y))
                                    Nat)
                       (forall-sort (lam (x num)
                                         (implies (in x pos-nat) (not (leq x zero))))
                                    Nat ))
                  (forall-sort (lam (x num)
                                    (forall-sort (lam (y num)
                                                      (implies (leq (s x) (s y))
                                                               (leq x y)))
                                                 pos-nat))
                               pos-nat)))
            (help "The classical less-or-equal operator on the natural numbers."))


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


(th~defdef cardinality
          (in natural)
          (type-variables bb)
          (definition
            (lam (G (o bb))
                 (that (lam (x num)
```

```
                             (and (in x Nat)
                                  (exists (lam (F (num bb))
                                               (bijective G (first-n-nats x) F))))))))
           (help "Definition of the finite cardinalities of sets."))

(th~defdef finite-cardinality
           (in natural)
           (type-variables bb)
           (definition
             (lam (G (o bb))
                  (exists-sort (lam (x num)
                                    (exists (lam (F (num bb))
                                                 (bijective G (first-n-nats x) F))))
                               Nat)))
           (help "Definition of the finite cardinalities of sets."))


(th~defdef finite
           (in natural)
           (type-variables bb)
           (definition
             (lam (G (o bb))
                  (exists-sort  (lam (n num)
                                     (= n (cardinality G)))
                                Nat        )))
           (help "Predicate for finiteness of sets."))


(th~defdef finite-subset
           (in natural)
           (type-variables bb)
           (definition
             (lam (G (o bb))
                  (lam (H (o bb))
                       (and (subset G H)
                            (finite G)))))
           (help "Predicates for finite subsets."))


(th~defaxiom nat-plus-struct
             (in natural)
             (formula
              (and (and (= (struct-set nat-plus-struct) Nat)
                        (= (struct-op nat-plus-struct) plus))
                   (= (struct-ordering nat-plus-struct) leq)))
             (help "The structure of the natural numbers with plus."))


(th~defaxiom nat-times-struct
             (in natural)
             (formula
              (and (= (struct-set nat-times-struct) Nat)
                   (= (struct-op nat-times-struct) times)))
             (help "The structure of the natural numbers with times."))


(th~defdef absval
           (in natural)
           (definition
             (lam (x num) (ifthen (less x zero) (change-sign x) x)))
           (help "The absolute value on numbers."))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Some Axioms for first-n-nats necessary to derive explicit sets
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(th~defaxiom first-n-nats-base
             (in natural)
             (formula
              (= (first-n-nats (s 0))
                 (lam (x num) (= x 0))))
             (help ""))

(th~defaxiom first-n-nats-step
             (in natural)
             (formula
              (forall-sort (lam (n num)
                                (= (first-n-nats (s n))
                                   (lam (x num) (or (= x n) (in x (first-n-nats n))))))
                           nat))
```

```
           (help ""))
```

D.4.2 natural-theorems.thy

```
(th~deftheorem times-nat-closed
               (in natural)
               (category theorem)
               (conclusion
                (forall-sort (lam (x num)
                                  (forall-sort (lam (y num)
                                                    (nat (times y x)))
                                               nat))
                             nat))
               (termdecl)
               (help "The set of natural numbers is closed under plus."))

(th~deftheorem power-nat-closed
               (in natural)
               (category theorem)
               (conclusion
                (forall-sort (lam (x num)
                                  (forall-sort (lam (y num)
                                                    (nat (power y x)))
                                               nat))
                             nat))
               (termdecl)
               (help "The set of natural numbers is closed under plus."))


(th~deftheorem nat-closed-s
               (in natural)
               (conclusion (closed-under-1 nat s))
               (help "The set of natural numbers is closed under successors."))

(th~deftheorem nat-plus-struct-set
               (in natural)
               (conclusion (= (struct-set nat-plus-struct) Nat))
               (help "The set of the nat-plus-struct is nat."))

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

(th~deftheorem nat-plus-struct-op
               (in natural)
               (conclusion (= (struct-op nat-plus-struct) plus))
               (help "The operation of nat-plus-struct is plus."))

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

(th~deftheorem nat-plus-struct-ord
               (in natural)
               (conclusion (= (struct-ordering nat-plus-struct) leq))
               (help "The ordering of nat-plus-struct is leq."))

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

(th~deftheorem nat-times-struct-set
               (in natural)
               (conclusion (= (struct-set nat-times-struct) Nat))
               (help "The set of the nat-times-struct is nat."))

(th~defsimplifier nat-times-struct-set
                  (in natural)
                  (status global)
                  (equation nat-times-struct-set)
```

```
                  (direction lr)
                  (help "Simplify the set of nat-times-struct."))

(th~deftheorem nat-times-struct-op
               (in natural)
               (conclusion (= (struct-op nat-times-struct) times))
               (help "The operation of nat-times-struct is times."))

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

(th~deftheorem nat-times-struct-ord
               (in natural)
               (conclusion (= (struct-ordering nat-times-struct) leq))
               (help "The ordering of nat-times-struct is leq."))

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



(th~deftheorem recursion-exists
               (in natural)
               (conclusion
                (forall (lam (h (num num num))
                             (forall (lam (g num)
                                          (and (= (recursion h g zero) g)
                                               (forall-sort (lam (n num)
                                                                 (= (recursion h g (s n))
                                                                    (h n (recursion h g n))))
                                                            Nat)))))))
               (help "Existence of the recursion operator."))

(th~deftheorem recursion-uniq
               (in natural)
               (category theorem)
               (conclusion
                (all-types cc
                           (forall
                            (lam (h (cc cc num))
                                 (forall
                                  (lam (g cc)
                                       (forall
                                        (lam (r1 (cc num cc (cc cc num)))
                                             (forall
                                              (lam (r2 (cc num cc (cc cc num)))
                                                   (implies (and (and (= (r1 h g zero) g)
                                                                      (forall-sort
                                                                       (lam (n num)
                                                                            (= (r1 h g (s n)) (h n (r1 h g n))))
                                                                       nat))
                                                                 (and (= (r2 h g zero) g)
                                                                      (forall-sort
                                                                       (lam (n num)
                                                                            (= (r2 h g (s n)) (h n (r2 h g n))))
                                                                       nat)))
                                                            (forall-sort
                                                             (lam (n num)
                                                                  (= (r1 h g n) (r2 h g n)))
                                                             nat))))))))))))
               (help "Uniqueness of the recursion operator."))



(th~deftheorem leq-refl
               (in natural)
               (category theorem)
               (conclusion (forall (lam (x num) (leq x x))))
               (help "Reflexivity of less-equal."))
```

```
(th~deftheorem leq-trans
               (in natural)
               (category theorem)
               (conclusion
                (forall (lam (x num)
                             (forall (lam (y num)
                                          (forall (lam (z num)
                                                       (implies (and (leq x y)
                                                                     (leq y z))
                                                                (leq x z)))))))))
               (help "Transitivity of less-equal."))


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



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


;(th~deftheorem plus-nat-base
;               (in natural)
;               (category theorem)
;               (conclusion (forall-sort (lam (x num) (= (plus x zero) x)) nat))
;               (help "Base case of the recursive definition of plus."))
;
;(th~deftheorem plus-nat-base2
;               (in natural)
;               (category theorem)
;               (conclusion (forall-sort (lam (x num) (= (plus zero x) x)) nat))
;               (help "Base case of the recursive definition of plus."))
;
;(th~deftheorem 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))))  nat))  nat))
;               (help "Step case of the recursive definition of plus."))



(th~deftheorem c-plus-nat-base
               (in natural)
               (category theorem)
               (conclusion
                (forall-sort (lam (x num) (= (plus x zero) (plus zero x)))
                             nat))
               (help "Base case of the commutative property of plus for natural numbers."))

;(th~deftheorem 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))))  nat))  nat))
;               (help "Another step case of the recursive definition of plus."))

(th~deftheorem c-plus-nat
               (in natural)
               (category theorem)
               (conclusion
                (forall-sort (lam (x num)
                                  (forall-sort (lam (y num)
                                                    (= (plus x y) (plus y x)))
                                               nat))
                             nat))
               (help "Commutative property of addtition."))
```

```
(th~deftheorem plus-nat-closed
               (in natural)
               (category theorem)
               (conclusion
                (forall-sort (lam (x num)
                                  (forall-sort (lam (y num)
                                                    (nat (plus y x)))
                                               nat))
                             nat))
               (termdecl)
               (help "The set of natural numbers is closed under plus."))

(th~deftheorem a-plus-nat
               (in natural)
               (category theorem)
               (conclusion
                (forall-sort (lam (x num)
                                  (forall-sort (lam (y num)
                                                    (forall-sort (lam (z num)
                                                                      (= (plus z (plus y x)) (plus (plus z y) x)))
                                                                 nat))
                                               nat))
                             nat))
               (help "Associative property of addition."))


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

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

(th~deftheorem plus-nat-step
               (in natural)
               (conclusion
                (forall-sort (lam (x num)
                                  (forall-sort (lam (y num)
                                                    (= (plus x (s y)) (s (plus x y))))
                                               nat))
                             nat))
               (help "The step case for recursive definition of addition."))

(th~deftheorem plus-nat-step2
               (in natural)
               (conclusion
                (forall-sort (lam (y num)
                                  (forall-sort (lam (x num)
                                                    (= (plus (s x) y) (s (plus x y))))
                                               nat))
                             nat))
               (help "Another step case for recursive definition of addition."))

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

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

(th~deftheorem times-nat-step
               (in natural)
```

```
               (conclusion
                (forall-sort (lam (x num)
                                  (forall-sort (lam (y num)
                                                    (= (times x (s y)) (plus x (times x y))))
                                               nat))
                             nat))
               (help "The step case for recursive definition of multiplication."))

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

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

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

(th~deftheorem power-nat-base-zero
               (in natural)
               (conclusion
                (forall-sort (lam (x num)
                                  (implies (less zero x)
                                           (= (power zero x)
                                              zero)))
                             nat))
               (help "Powers with base zero."))

(th~deftheorem power-nat-base-one
               (in natural)
               (conclusion
                (forall-sort (lam (x num)
                                  (= (power (s zero) x)
                                     (s zero)))
                             nat))
               (help "Powers with base one."))

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

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

(th~deftheorem less-implies-leq-nat
               (in natural)
               (conclusion
                (forall-sort (lam (x num)
                                  (forall-sort (lam (y num)
                                                    (implies (less x y) (leq x y)))
                                               nat))
```

```
                            nat))
               (help "Less implies less or equal."))

(th~deftheorem equal-implies-leq-nat
               (in natural)
               (conclusion
                (forall-sort (lam (x num)
                                   (forall-sort (lam (y num)
                                                     (implies (= x y) (leq x y)))
                                                nat))
                             nat))
               (help "Equal implies less or equal."))


(th~deftheorem a-plus-num
               (in natural)
               (category theorem)
               (conclusion
                (forall (lam (x num)
                             (forall (lam (y num)
                                          (forall (lam (z num)
                                                       (= (plus z (plus y x)) (plus (plus z y) x)))))))))
               (help "Associative property of addition."))

(th~deftheorem c-plus-num
               (in natural)
               (category theorem)
               (conclusion
                (forall (lam (x num)
                             (forall (lam (y num)
                                          (= (plus x y) (plus y x)))))))
               (help "Associative property of addition."))


;;;Something about even numbers

(th~deftheorem subsort-even-nat
               (in natural)
               (conclusion
                (forall-sort (lam (x num) (nat x))
                             even))
               (termdecl)
               (help "Even numbers are a subset of natural numbers."))

(th~deftheorem even-from-nat
               (in natural)
               (conclusion
                (forall-sort (lam (x num) (even (plus x x)))
                             nat))
               (termdecl)
               (help "The sum of the same two natural numbers is even."))

(th~deftheorem even-plus-closed
               (in natural)
               (conclusion
                (forall-sort (lam (y num)
                                   (forall-sort (lam (x num)
                                                     (even (plus y x)))
                                                even))
                             even))
               (termdecl)
               (help "The sum of the same two natural numbers is even."))
```