$\neg\sqrt{2} \in \mathbb{Q}$

—

3 Proofs in ProofPower-HOL

R.D. Arthan
Lemma 1 Ltd.
rda@lemma-one.com

20th March 2005 with subsequent revisions*

Some while ago Freek Wiedijk proposed the irrationality of $\sqrt{2}$ as an interesting example to use in a comparative study of different proof assistants. Freek received formalisations using many systems and has prepared a report presenting the results. He was recently kind enough to draw my attention to a draft of his very interesting introduction to the report. This kind thought led to a small flurry of comments and then to the present contribution to the study using the ProofPower system. My apologies to Freek for creating extra work at the last minute. Despite my efforts to delay him, Freek's report was published in book form in 2006 with a very nice foreword by Dana Scott [*Seventeen Provers of the World*, LNAI 3600, Springer Verlag].

The formal material is based on the mathematical case studies in ProofPower-HOL that have been under evolutionary development over the last few years. The present (March 2005) version of this document does not yet form part of the official case studies, but parts of it are likely to be included in them when one or two obvious gaps in the breadth of coverage have been filled (most conspicuously the fundamental theorem of arithmetic). This note follows the mathematical case studies in including theory listings for all the theories developed (in sections 6 to 10). We use a slightly different approach in the discussion (giving the statements of the main results as ML quotations).

In retrospect, one of my main comments on Freek's report amounted to my surprise that the theorem-proving community only seemed to know one proof! The irrationality of $\sqrt{2}$ has several proofs and so, in the interests of variety, three proofs are presented here. Proof 1 is an ancient, but seemingly not so widely known, "geometrical" proof that requires no number theory. This proof is amenable to some interesting generalisations, but we do not look into that here (see, for example, *The Book of Numbers* by John H. Conway and Richard K. Guy). Proof 2 and proof 3 are the well-known proofs based on divisibility. Perhaps the least well-known thing about the well-known proofs is that there are two of them! We present them in reasonably full generality (showing in one case that the square root of any prime number is irrational, and in the other, that, if the square root of an integer is rational than the square root is actually an integer). Following Freek's rules, we are careful to derive the specific conclusion that $\sqrt{2}$ is irrational from the general proofs.

*Revised 11th November 2013 to be compatible with the latest version of the ProofPower mathemical case studies. Revised 23rd November 2022 to include the nice depiction of proof 1 as a proof without words mentioned in the foreword to Freek Wiedijk's book and to avoid inconvenient redeclaration of ML names.

1 Common Definitions

The problem clearly needs to be formalised in terms of the 5 symbols forming the title of this document. Of these, logical negation, the number 2 and the membership sign are supplied for free. The square root function is defined in the theory of analysis from the mathematical case studies with the following defining property:

$$\vdash \forall\, x \bullet\ 0. \leq x \Rightarrow 0. \leq Sqrt\ x \wedge Sqrt\ x \hat{}\ 2 = x$$

The proofs use no facts of analysis other than this definition. We need to define the set of rational numbers. The definition is common to all three proofs: after all, we want the three proofs all to prove exactly the same thing.

The following red tape sets up a theory *sqrt_defs* to hold this material.

SML

```
set_pc "basic_hol1";
open_theory "analysis";
force_delete_theory "sqrt2_defs" handle Fail _ => ();
new_theory "sqrt2_defs";
```

To state what is proved by the third proof, we need the set of integers as well as the set of rationals. The definitions follow (in the usual ProofPower-HOL constant specification boxes where we give the signature of the new constant and its desired defining property separated by a horizontal bar). The consistency of these equational definitions is proved automatically.

HOL Constant

$$\mathbb{Z} : \mathbb{R}\ SET$$

$$\mathbb{Z} = \{x \mid \exists m : \mathbb{N} \bullet x = \mathbb{NR}\ m \vee x = \sim(\mathbb{NR}\ m)\}$$

Here $\mathbb{NR}$ is the function that injects the type of natural numbers into the type of reals.

HOL Constant

$$\mathbb{Q} : \mathbb{R}\ SET$$

$$\mathbb{Q} = \{x \mid \exists a\ b : \mathbb{N} \bullet \neg b = 0 \wedge (x = a/b \vee x = \sim(a/b))\}$$

A handful of basic theorems about the square root function are developed in this theory, see the listing for details.

2 Proof 1

Our first proof is very simple. It makes no reference to prime divisors or the like. It is inspired by the construction depicted in figure 1. If we let $x = BD$ and $y = AB$ be respectively the diagonal and side of the larger square, so that $x/y = \sqrt{2}$, $DE = 2y - x$ and $DF = x - y$ form the diagonal and side of the smaller square. But then, if x and y were both integers, so also would be $x - y$ and $2y - x$ and we would have a contradiction, since we could repeat the construction to produce arbitrarily small squares with integer sides.

In his foreword to Freek's book, Dana Scott added a note giving a beautiful depiction of this proofs as a "proof without words" due to Stanley Tennenbaum. See figure 2 for Tennenbaum's diagram together with some pictorial equations showing the reasoning: the big square is presumed to have area equal to the sum of the areas of the two congruent hatched squares, which have been placed inside the big square at diagonally opposite corners. But then the intersection of the hatched squares, the doubly-hatched square in the figure must have its area equal to the sum of the areas of the two small white squares. But if the big square and the two hatched squares have integer side-lengths, then so do all the squares, showing that their can be no minimal pair of positive integers x and y with $x^2 = 2y^2$.

This proof translates very simply into algebra. To present the formalisation, we give the statements of a selection of the lemmas proved. The theory listings towards the end of the document give the output from ProofPower showing that these results have indeed been proved. The actual proof scripts are included in the master source text of this document, but not in the printed form.

The main work in this proof is given in a series of 5 lemmas. The first lemma gives the key algebraic facts about the geometrical construction. It also includes what amounts to the estimate that $1 < \sqrt{2} < 3/2$, which is needed to show that when the inputs to the construction are positive integers, then so are the outputs.

SML

$$
\begin{array}{l}
val\ \mathbf{proof1\_lemma1} = \ulcorner \\
\forall x\ y \bullet \\
\quad \mathbb{NR}\ 0 \leq x \wedge \mathbb{NR}\ 0 < y \wedge x \hat{}\ 2 = \mathbb{NR}\ 2 * y \hat{}\ 2 \\
\Rightarrow \quad y < x \wedge \mathbb{NR}\ 2 * x \leq \mathbb{NR}\ 3 * y \\
\wedge \quad (\mathbb{NR}\ 2 * y - x) \hat{}\ 2 = \mathbb{NR}\ 2 * (x - y) \hat{}\ 2 \\
\urcorner;
\end{array}
$$

The second and third lemmas (see the theory listing in section 7) essentially just specialise the above to the case where x and y are natural numbers.

All three proofs proceed by Fermat's "method of infinite descent". I.e., one shows that the existence of a positive integer counter-example to a conjecture implies the existence of a smaller counter-example. Thus in each case we have an "inductive step" that produces smaller counter-examples from larger ones. The following lemma gives this step for the present proof:

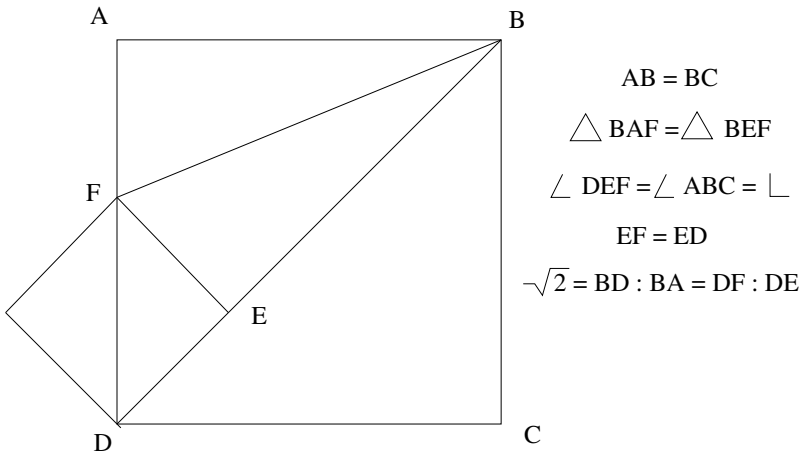

Figure 1: The Geometrical Construction

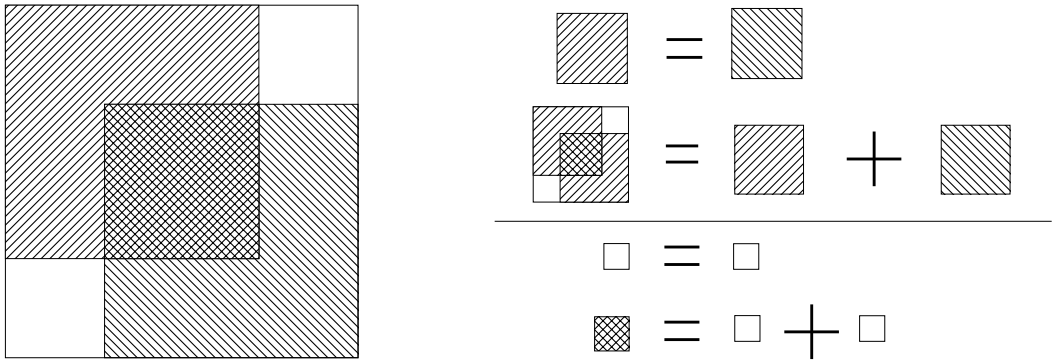

Figure 2: The Proof Without Words

```
SML
|val proof1_lemma4 = ⌜
|∀m n•
|        ℕℝ m ^ 2 = ℕℝ 2 * ℕℝ n ^ 2 ∧ 0 < n
|⇒       ∃m1 n1•0 < n1 ∧ n1 < n ∧ ℕℝ m1 ^ 2 = ℕℝ 2 * ℕℝ n1 ^ 2
|⌝;
```

From this we conclude that the only natural number solution to $m^2 = 2n^2$ has $n = 0$.

```
SML
|val proof1_lemma5 = ⌜
|        ∀n m• ℕℝ m ^ 2 = ℕℝ 2 * ℕℝ n ^ 2 ⇒ n = 0
|⌝;
```

The desired result follows easily from the above. We formalise it in two guises, the first guise is explicit:

```
SML
|val proof1_thm1 = ⌜
|        ∀a b• ¬b = 0 ⇒ ¬(a/b)^2 = ℕℝ 2
|⌝;
```

The second guise gives the result much as it is stated in the title of this note:

```
SML
|val proof1_thm2 = ⌜
|        ¬Sqrt (ℕℝ 2) ∈ ℚ
|⌝;
```

3 Divisibility

The other two proofs we give are the better known ones based on divisibility. They share common material from the theory of divisibility. In this section we define the common notions. First we set up a theory to hold the definitions. See section 8 for the listing of this theory.

SML

```
open_theory "fin_set";
force_delete_theory "divisibility" handle Fail _ => ();
new_theory "divisibility";
```

We have a choice about whether to develop the theory for the natural numbers or for the integers. On the one hand, it is more pleasant to work in a ring rather than a semi-ring, on the other hand negative numbers are not very relevant to the main results, even when they are useful in the proofs. We vote in favour of the natural numbers, and proceed to define the greatest common divisor function. This is an implicit definition: the first of the two conjuncts in the defining property say that the greatest common divisor is a common divisor and the second says that it is the greatest one (i.e., it is maximal with respect to the divisibility ordering of the natural numbers). We have several choices about how to capture formally the notion *"m is divisible by n"*. We opt to state it as $m\ Mod\ n = 0$.

HOL Constant

$$\boldsymbol{Gcd} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$$

$$
\begin{array}{lll}
(\forall m\ n\bullet & & 0 < m \wedge 0 < n \\
\quad \Rightarrow & & 0 < Gcd\ m\ n \\
\quad \wedge & & m\ Mod\ Gcd\ m\ n = 0 \\
\quad \wedge & & n\ Mod\ Gcd\ m\ n = 0) \\
\wedge\ (\forall m\ n\ d\bullet & & 0 < d \\
\quad \wedge & & m\ Mod\ d = 0 \\
\quad \wedge & & n\ Mod\ d = 0 \\
\quad \Rightarrow & & Gcd\ m\ n\ Mod\ d = 0)
\end{array}
$$

Now we define the set of prime numbers:

HOL Constant

$$\boldsymbol{Prime} : \mathbb{N}\ SET$$

$$Prime = \{p \mid 1 < p \wedge \forall m\ n\bullet\ p = m{*}n \Rightarrow m = 1 \vee n = 1\}$$

The important fact we need about prime numbers is the fact that a number is prime iff. it is greater than 1 and whenever it divides a product it divides one of the factors. The right-to-left direction of this is simple. It is for this the other direction that we need to develop the theory of the g.c.d.

SML

$$
\begin{array}{lll}
val\ \boldsymbol{prime\_thm} = \ulcorner & & \\
\quad \forall p\bullet & & p \in Prime \\
\quad \Leftrightarrow & & 1 < p \\
\wedge & & (\forall m\ n\bullet\ (m{*}n)\ Mod\ p = 0 \Rightarrow m\ Mod\ p = 0 \vee n\ Mod\ p = 0) \\
\urcorner; & &
\end{array}
$$

The bulk of the theory then comprises the supporting lemmas and theorems we need to prove that the definition of greatest common divisor is consistent and to reason about it (see the theory listing in section 8 for full details). The most common textbook account exhibits the g.c.d. of m and n as the smallest positive value of the form $am+bn$. This requires a and b to range over negative integers. A slightly less symmetrical alternative is to take the g.c.d. to be the smallest positive value of the

form $(am)\ Mod\ n$. This works over the natural numbers. The main result (after the consistency theorem) is the following:

SML
```
val gcd_eq_mod_thm = ⌜
        ∀m n• 0 < m ∧ 0 < n ∧ 0 < m Mod n
        ⇒       ∃a• 0 < (a*m) Mod n
        ∧       (∀b• 0 < (b*m) Mod n ⇒ (a*m) Mod n ≤ (b*m) Mod n)
        ∧       Gcd m n = (a*m) Mod n
⌝;
```

The final theorem in this theory for the current version of this document says that any integer greater than 1 has a prime divisor.

SML
```
val prime_divisor_thm = ⌜
        ∀m• 1 < m ⇒ ∃p n•p ∈ Prime ∧ m = p*n
⌝;
```

From this, it is a very short step to the fundamental theorem of arithmetic, but that is not needed for present purposes.

4 Proof 2

Our second proof is the most widely-known one: if $m^2 = 2n^2$, then m is even, but then so is n, so we can divide m and n by 2 to get a solution with smaller n. This gives a contradiction, because if there is a solution for n positive, we get an infinite descending sequence of positive integers. This proof generalises to show that $\sqrt{p}$ is irrational for any prime p.

To quote the formal steps in the proof we need to construct a theory in which the common definitions and the material on divisibility is available. The lemmas and theorems making up the proof are later stored in this theory. See section 9 for the listing.

SML
```
open_theory "divisibility";
force_delete_theory "sqrt2_proof2" handle Fail _ => ();
new_theory "sqrt2_proof2";
new_parent"sqrt2_defs";
```

The proof starts with the following lemma, the proof of which is easy given the results on divisibility. This is almost identical to lemma 4 in the 1st proof, but with an arbitrary prime p in place of the specific number 2. Of course, the method of proof is quite different: one observes that under the stated conditions, p must divide both m and n and so dividing through by it gives a smaller solution.

SML
```
val proof2_lemma1 = ⌜
∀p m n•                 p ∈ Prime ∧ m * m = p * n * n ∧ 0 < n
⇒       ∃m1 n1•0 < n1 ∧ n1 < n ∧ m1 * m1 = p * n1 * n1
⌝;
```

From this it follows that the only solution to $m^2 = pn^2$ in natural numbers m and n has $n = m = 0$. Whence:

```
SML
|val proof2_thm2 = ⌜
|        ∀p•p ∈ Prime ⇒ ¬Sqrt (ℕℝ p) ∈ ℚ
|⌝;
```

Freek quite rightly insists that we actually prove that $\sqrt{2}$ is irrational, so we need to prove that it is prime.

```
SML
|val proof2_lemma3 = ⌜
|        2 ∈ Prime
|⌝;
```

Whence we draw the usual conclusion:

```
SML
|val proof2_thm3 = ⌜
|        ¬Sqrt (ℕℝ 2) ∈ ℚ
|⌝;
```

5 Proof 3

This is the most general of the three proofs we give: if $m^2 = kn^2$ for any natural numbers k, m and n with k positive and $n > 1$, then any prime divisor of n is also a prime divisor of m. Thus by the usual infinite descent the only solutions of this equation with k and n positive have $n = 1$, i.e., the only solutions are when $k = m^2$ is a square.

As in the previous section we need to create a theory in which the right vocabulary is available to state the results: The lemmas and theorems making up the proof are later stored in this theory. See section 10 for the listing.

```
SML
|open_theory "divisibility";
|force_delete_theory "sqrt2_proof3" handle Fail _ => ();
|new_theory "sqrt2_proof3";
|new_parent"sqrt2_defs";
```

The first lemma is the following. It justifies the steps in the infinite descent.

```
SML
|val proof3_lemma1 = ⌜
|∀k m n•        0 < k ∧ m * m = k * n * n ∧ 1 < n
|⇒      ∃m1 n1•0 < n1 ∧ n1 < n ∧ m1 * m1 = k * n1 * n1
|⌝;
```

The next step is rather different. The infinite descent bottoms out at 1, from which we we conclude that if $m^2 = kn^2$ has a natural number solution with n positive, then k is a perfect square.

```
SML
|val proof3_lemma2 = ⌜
|∀k n m•        ℕℝ 0 < ℕℝ k ∧ ℕℝ 0 < ℕℝ n ∧ ℕℝ m ^ 2 = ℕℝ k * (ℕℝ n ^ 2)
|⇒       ∃i• ℕℝ i ^ 2 = ℕℝ k
|⌝;
```

With intermediate steps similar to the previous proofs, we arrive at the following:

```
SML
|val proof3_thm2 = ⌜
|∀i• ℕℝ 0 ≤ i ∧ i ∈ ℤ ∧ Sqrt i ∈ ℚ ⇒ Sqrt i ∈ ℤ
|⌝;
```

Yet again, we must exercise our skills on the specific number 2, which requires the following lemma (easily proved using the numerical estimate that $1 < \sqrt{2} < 2$).

```
SML
|val proof3_lemma3 = ⌜
|        ¬Sqrt (ℕℝ 2) ∈ ℤ
|⌝;
```

From which we conclude for the third and final time our old friend:

```
SML
|val proof3_thm3 = ⌜
|        ¬Sqrt (ℕℝ 2) ∈ ℚ
|⌝;
```

6 THE THEORY sqrt2_defs

6.1 Parents

analysis

6.2 Children

sqrt2_proof1 *sqrt2_proof3* *sqrt2_proof2*

6.3 Constants

$\mathbb{Z}$ $\mathbb{R}$ $\mathbb{P}$
$\mathbb{Q}$ $\mathbb{R}$ $\mathbb{P}$

6.4 Definitions

$\mathbb{Z}$ $\vdash \mathbb{Z} = \{x | \exists\ m \bullet\ x = \mathbb{NR}\ m \vee x = \sim (\mathbb{NR}\ m)\}$
$\mathbb{Q}$ $\vdash \mathbb{Q} = \{x | \exists\ a\ b \bullet \neg\ b = 0 \wedge (x = a\ /\ b \vee x = \sim (a\ /\ b))\}$

6.5 Theorems

sqrt_thm $\vdash \forall\ x \bullet\ 0. \leq x \Rightarrow Sqrt\ x\ \hat{}\ 2 = x$
square_even_thm
$\vdash \forall\ x \bullet \sim x\ \hat{}\ 2 = x\ \hat{}\ 2$
sqrt_eq_thm $\vdash \forall\ x\ y \bullet\ 0. \leq x \wedge x\ \hat{}\ 2 = y \Rightarrow x = Sqrt\ y$
sqrt_egs_thm $\vdash Sqrt\ 0. = 0.$
$\wedge\ Sqrt\ 1. = 1.$
$\wedge\ Sqrt\ 4. = 2.$
$\wedge\ Sqrt\ 9. = 3.$
square_square_root_mono_thm1
$\vdash \forall\ x\ y \bullet\ 0. \leq x \wedge 0. \leq y \Rightarrow (x\ \hat{}\ 2 < y\ \hat{}\ 2 \Leftrightarrow x < y)$
sqrt_less_thm
$\vdash \forall\ x\ y \bullet\ 0. \leq x \wedge 0. \leq y \Rightarrow (Sqrt\ x < Sqrt\ y \Leftrightarrow x < y)$

7 THE THEORY sqrt2_proof1

7.1 Parents

sqrt2_defs

7.2 Theorems

proof1_lemma1_thm
$$\vdash \forall\ x\ y \bullet\ 0. \leq x \wedge 0. < y \wedge x \hat{}\ 2 = 2. * y \hat{}\ 2 \Rightarrow y < x \wedge 2. * x \leq 3. * y \wedge (2. * y - x) \hat{}\ 2 = 2. * (x - y) \hat{}\ 2$$

proof1_lemma2_thm
$$\vdash \forall\ i\ j \bullet\ j \leq i \Rightarrow \mathbb{NR}\ (i - j) = \mathbb{NR}\ i - \mathbb{NR}\ j$$

proof1_lemma3_thm
$$\vdash \forall\ m\ n \bullet\ \mathbb{NR}\ m \hat{}\ 2 = 2. * \mathbb{NR}\ n \hat{}\ 2 \wedge 0 < n \Rightarrow n < m \wedge 2 * m \leq 3 * n \wedge \mathbb{NR}\ (2 * n - m) \hat{}\ 2 = 2. * \mathbb{NR}\ (m - n) \hat{}\ 2$$

proof1_lemma4_thm
$$\vdash \forall\ m\ n \bullet\ \mathbb{NR}\ m \hat{}\ 2 = 2. * \mathbb{NR}\ n \hat{}\ 2 \wedge 0 < n \Rightarrow (\exists\ m1\ n1 \bullet\ 0 < n1 \wedge n1 < n \wedge \mathbb{NR}\ m1 \hat{}\ 2 = 2. * \mathbb{NR}\ n1 \hat{}\ 2)$$

proof1_lemma5_thm
$$\vdash \forall\ n\ m \bullet\ \mathbb{NR}\ m \hat{}\ 2 = 2. * \mathbb{NR}\ n \hat{}\ 2 \Rightarrow n = 0$$

proof1_thm1_thm
$$\vdash \forall\ a\ b \bullet\ \neg\ b = 0 \Rightarrow \neg\ (a\ /\ b) \hat{}\ 2 = 2.$$

proof1_thm1a
$$\vdash \forall\ a\ b \bullet\ \neg\ b = 0 \Rightarrow \neg\ (a\ /\ b) \hat{}\ 2 = 2. \wedge \neg \sim (a\ /\ b) \hat{}\ 2 = 2.$$

proof1_thm2
$$\vdash \neg\ Sqrt\ 2. \in \mathbb{Q}$$

8 THE THEORY divisibility

8.1 Parents

fin_set

8.2 Children

sqrt2_proof3 *sqrt2_proof2*

8.3 Constants

Gcd	$\mathbb{N} \to \mathbb{N} \to \mathbb{N}$
Prime	$\mathbb{N}\ \mathbb{P}$

8.4 Definitions

Gcd $\vdash$ *ConstSpec*
$(\lambda\ Gcd'$
$\bullet\ (\forall\ m\ n$
$\bullet\ 0 < m \wedge 0 < n$
$\Rightarrow\ 0 < Gcd'\ m\ n$
$\wedge\ m\ Mod\ Gcd'\ m\ n = 0$
$\wedge\ n\ Mod\ Gcd'\ m\ n = 0)$
$\wedge\ (\forall\ m\ n\ d$
$\bullet\ 0 < d \wedge m\ Mod\ d = 0 \wedge n\ Mod\ d = 0$
$\Rightarrow\ Gcd'\ m\ n\ Mod\ d = 0))$
Gcd

Prime $\vdash$ *Prime*
$= \{p | 1 < p \wedge (\forall\ m\ n \bullet\ p = m * n \Rightarrow m = 1 \vee n = 1)\}$

8.5 Theorems

min_∈_thm $\vdash \forall\ n\ a \bullet\ n \in a \Rightarrow Min\ a \in a$

min_≤_thm $\vdash \forall\ n\ a \bullet\ n \in a \Rightarrow Min\ a \leq n$

times_eq_0_thm
$\vdash \forall\ m\ n \bullet\ m * n = 0 \Rightarrow m = 0 \vee n = 0$

times_cancel_thm
$\vdash \forall\ k\ m\ n \bullet\ 0 < k \wedge k * m = k * n \Rightarrow m = n$

times_eq_eq_1_thm
$\vdash \forall\ m\ n \bullet\ 0 < n \wedge m * n = n \Rightarrow m = 1$

times_eq_1_thm
$\vdash \forall\ m\ n \bullet\ m * n = 1 \Rightarrow m = 1 \wedge n = 1$

div_mod_1_thm
$\vdash \forall\ m \bullet\ m\ Div\ 1 = m \wedge m\ Mod\ 1 = 0$

m_div_mod_m_thm
$\vdash \forall\ m \bullet\ 0 < m \Rightarrow m\ Div\ m = 1 \wedge m\ Mod\ m = 0$

zero_div_mod_thm
$\vdash \forall\ m \bullet\ 0 < m \Rightarrow 0\ Div\ m = 0 \wedge 0\ Mod\ m = 0$

less_div_mod_thm

$\vdash \forall\ m\ n \bullet\ n < m \Rightarrow n\ Div\ m = 0 \wedge n\ Mod\ m = n$

div_mod_times_cancel_thm

$\vdash \forall\ k\ m\ n$
$\bullet\ 0 < k$
$\Rightarrow (m * k + n)\ Div\ k = m + n\ Div\ k$
$\wedge (m * k + n)\ Mod\ k = n\ Mod\ k$

mod_clauses $\vdash \forall\ k\ m\ n$
$\bullet\ 0 < k$
$\Rightarrow (m * k)\ Mod\ k = 0$
$\wedge (k * m)\ Mod\ k = 0$
$\wedge (k * m + n)\ Mod\ k = n\ Mod\ k$
$\wedge (m * k + n)\ Mod\ k = n\ Mod\ k$
$\wedge (k + n)\ Mod\ k = n\ Mod\ k$
$\wedge (n + k)\ Mod\ k = n\ Mod\ k$
$\wedge 0\ Mod\ k = 0$
$\wedge k\ Mod\ k = 0$
$\wedge m\ Mod\ k\ Mod\ k = m\ Mod\ k$

mod_eq_0_thm $\vdash \forall\ m\ n \bullet\ 0 < n \Rightarrow (m\ Mod\ n = 0 \Leftrightarrow (\exists\ k \bullet\ m = k * n))$

mod_eq_0_mod_eq_0_thm

$\vdash \forall\ m\ n$
$\bullet\ 0 < m \wedge 0 < n \wedge m\ Mod\ n = 0 \wedge n\ Mod\ m = 0 \Rightarrow m = n$

mod_plus_homomorphism_thm

$\vdash \forall\ m\ n\ k$
$\bullet\ 0 < k \Rightarrow (m + n)\ Mod\ k = (m\ Mod\ k + n\ Mod\ k)\ Mod\ k$

mod_times_homomorphism_thm

$\vdash \forall\ m\ n\ k$
$\bullet\ 0 < k \Rightarrow (m * n)\ Mod\ k = (m\ Mod\ k * n\ Mod\ k)\ Mod\ k$

gcd_consistent_lemma1

$\vdash \forall\ m\ n$
$\bullet\ 0 < m \wedge 0 < n \wedge 0 < m\ Mod\ n$
$\Rightarrow (\exists\ a$
$\bullet\ 0 < (a * m)\ Mod\ n$
$\wedge (\forall\ b$
$\bullet\ 0 < (b * m)\ Mod\ n$
$\Rightarrow (a * m)\ Mod\ n \leq (b * m)\ Mod\ n))$

gcd_consistent_lemma2

$\vdash \forall\ m\ n\ a\ d$
$\bullet\ 0 < m \wedge 0 < n \wedge 0 < d \wedge m\ Mod\ d = 0 \wedge n\ Mod\ d = 0$
$\Rightarrow (a * m)\ Mod\ n\ Mod\ d = 0$

gcd_consistent_lemma3

$\vdash \forall\ a\ b\ m\ n\ p\ r\ s$
$\bullet\ a * m = b * (n + 1) + r \wedge m = p * r + s$
$\Rightarrow (\exists\ q \bullet\ (q * m)\ Mod\ (n + 1) = s\ Mod\ (n + 1))$

gcd_consistent_lemma4

$\vdash \forall\ a\ b\ m\ n\ p\ r\ s$
$\bullet\ a * m = b * (n + 1) + r \wedge n + 1 = p * r + s$
$\Rightarrow (\exists\ q \bullet\ (q * m)\ Mod\ (n + 1) = s\ Mod\ (n + 1))$

gcd_consistent_lemma5

$\vdash \forall\ m\ n\ k\ a$
$\bullet\ 0 < m$
$\wedge\ 0 < n$

$\wedge\ 0 < m\ Mod\ n$
$\wedge\ 0 < (a * m)\ Mod\ n$
$\wedge\ (\forall\ b$
$\bullet\ 0 < (b * m)\ Mod\ n$
$\Rightarrow (a * m)\ Mod\ n \leq (b * m)\ Mod\ n)$
$\Rightarrow m\ Mod\ ((a * m)\ Mod\ n) = 0$
$\wedge\ n\ Mod\ ((a * m)\ Mod\ n) = 0$

Gcd_consistent

$\vdash Consistent$
$(\lambda\ Gcd'$
$\bullet\ (\forall\ m\ n$
$\bullet\ 0 < m \wedge 0 < n$
$\Rightarrow 0 < Gcd'\ m\ n$
$\wedge\ m\ Mod\ Gcd'\ m\ n = 0$
$\wedge\ n\ Mod\ Gcd'\ m\ n = 0)$
$\wedge\ (\forall\ m\ n\ d$
$\bullet\ 0 < d \wedge m\ Mod\ d = 0 \wedge n\ Mod\ d = 0$
$\Rightarrow Gcd'\ m\ n\ Mod\ d = 0))$

gcd_eq_mod_thm

$\vdash \forall\ m\ n$
$\bullet\ 0 < m \wedge 0 < n \wedge 0 < m\ Mod\ n$
$\Rightarrow (\exists\ a$
$\bullet\ 0 < (a * m)\ Mod\ n$
$\wedge\ (\forall\ b$
$\bullet\ 0 < (b * m)\ Mod\ n$
$\Rightarrow (a * m)\ Mod\ n \leq (b * m)\ Mod\ n)$
$\wedge\ Gcd\ m\ n = (a * m)\ Mod\ n)$

prime_0_less_thm

$\vdash \forall\ p\bullet\ p \in Prime \Rightarrow 0 < p$

gcd_prime_thm

$\vdash \forall\ m\ p\bullet\ 0 < m \wedge p \in Prime \Rightarrow Gcd\ m\ p = 1 \vee Gcd\ m\ p = p$

prime_thm

$\vdash \forall\ p$
$\bullet\ p \in Prime$
$\Leftrightarrow 1 < p$
$\wedge\ (\forall\ m\ n$
$\bullet\ (m * n)\ Mod\ p = 0$
$\Rightarrow m\ Mod\ p = 0 \vee n\ Mod\ p = 0)$

prime_divisor_thm

$\vdash \forall\ m\bullet\ 1 < m \Rightarrow (\exists\ p\ n\bullet\ p \in Prime \wedge m = p * n)$

9 THE THEORY sqrt2_proof2

9.1 Parents

sqrt2_defs *divisibility*

9.2 Theorems

proof2_lemma1_thm
$\vdash \forall\ p\ m\ n$
$\bullet\ p \in Prime \wedge m * m = p * n * n \wedge 0 < n$
$\Rightarrow (\exists\ m1\ n1$
$\bullet\ 0 < n1 \wedge n1 < n \wedge m1 * m1 = p * n1 * n1)$

proof2_lemma2_thm
$\vdash \forall\ p\ n\ m$
$\bullet\ p \in Prime \wedge \mathbb{NR}\ m \hat{}\ 2 = \mathbb{NR}\ p * \mathbb{NR}\ n \hat{}\ 2 \Rightarrow n = 0$

proof2_thm1 $\vdash \forall\ p\ a\ b \bullet\ p \in Prime \wedge \neg\ b = 0 \Rightarrow \neg\ (a\ /\ b) \hat{}\ 2 = \mathbb{NR}\ p$

proof2_thm1a $\vdash \forall\ a\ b$
$\bullet\ p \in Prime \wedge \neg\ b = 0$
$\Rightarrow \neg\ (a\ /\ b) \hat{}\ 2 = \mathbb{NR}\ p \wedge \neg \sim (a\ /\ b) \hat{}\ 2 = \mathbb{NR}\ p$

proof1_thm2_thm
$\vdash \forall\ p \bullet\ p \in Prime \Rightarrow \neg\ Sqrt\ (\mathbb{NR}\ p) \in \mathbb{Q}$

proof2_lemma3_thm
$\vdash 2 \in Prime$

proof2_thm3_thm
$\vdash \neg\ Sqrt\ 2. \in \mathbb{Q}$

10 THE THEORY sqrt2_proof3

10.1 Parents

sqrt2_defs *divisibility*

10.2 Theorems

proof3_lemma1_thm

$\vdash \forall\ k\ m\ n$
$\bullet\ 0 < k \wedge m * m = k * n * n \wedge 1 < n$
$\Rightarrow (\exists\ m1\ n1$
$\bullet\ 0 < n1 \wedge n1 < n \wedge m1 * m1 = k * n1 * n1)$

proof3_lemma2_thm

$\vdash \forall\ k\ n\ m$
$\bullet\ 0. < \mathbb{NR}\ k \wedge 0. < \mathbb{NR}\ n \wedge \mathbb{NR}\ m\ \hat{}\ 2 = \mathbb{NR}\ k * \mathbb{NR}\ n\ \hat{}\ 2$
$\Rightarrow (\exists\ i\bullet\ \mathbb{NR}\ i\ \hat{}\ 2 = \mathbb{NR}\ k)$

proof3_lemma3_thm

$\vdash \neg\ Sqrt\ 2. \in \mathbb{Z}$

proof3_thm1_thm

$\vdash \forall\ k\ a\ b$
$\bullet\ \neg\ b = 0 \wedge (a\ /\ b)\ \hat{}\ 2 = \mathbb{NR}\ k$
$\Rightarrow (\exists\ i\bullet\ \mathbb{NR}\ i\ \hat{}\ 2 = \mathbb{NR}\ k)$

proof3_thm2 $\vdash \forall\ i\bullet\ 0. \leq i \wedge i \in \mathbb{Z} \wedge Sqrt\ i \in \mathbb{Q} \Rightarrow Sqrt\ i \in \mathbb{Z}$

proof3_thm3 $\vdash \neg\ Sqrt\ 2. \in \mathbb{Q}$

Index

A Common Definitions — Proofs

Prove the consistency of the definition of the square root (using the exponential and logarithm functions for the witness):

SML
```
open_theory "sqrt2_defs";
set_merge_pcs["'ℤ", "'ℝ", "'sets_alg", "basic_hol1" ];
```

The existence of square roots has already been proved in the the theory of analysis. We just have to use the existence theorem to provide a witness.

SML
```
(*
push_consistency_goal ⌜Sqrt⌝;
a(prove_∃_tac THEN REPEAT strip_tac);
a(cases_tac⌜ℕℝ 0 ≤ x'⌝ THEN asm_rewrite_tac[]);
a(bc_thm_tac square_root_thm1 THEN REPEAT strip_tac);
save_consistency_thm ⌜Sqrt⌝ (pop_thm());
*)
```

SML
```
val sqrt_def = get_spec⌜Sqrt⌝;
val sqrt_thm = tac_proof(([],
        ⌜∀x• ℕℝ 0 ≤ x ⇒ Sqrt x ^ 2 = x⌝),
        REPEAT strip_tac THEN all_fc_tac[sqrt_def]);
val ℤ_def = get_spec⌜ℤ⌝;
val rats_def = get_spec⌜ℚ⌝;
```

SML
```
set_goal([], ⌜
        ∀x• ℕℝ 0 ≤ x ⇒ Sqrt x ^ 2 = x
⌝);
a(REPEAT strip_tac THEN all_fc_tac [sqrt_def]);
val sqrt_thm = save_pop_thm "sqrt_thm";
```

SML
```
set_goal([], ⌜
        ∀x : ℝ• (∼x) ^ 2 = x ^ 2
⌝);
a(rewrite_tac [ℝ_ℕ_exp_square_thm]
        THEN PC_T1 "ℝ_lin_arith" prove_tac[]);
val square_even_thm = save_pop_thm "square_even_thm";
```

SML

```
set_goal([], ⌜
        ∀x y• ℕℝ 0 ≤ x ∧ x^2 = y ⇒ x = Sqrt y
⌝);
a(REPEAT strip_tac);
a(lemma_tac⌜ℕℝ 0 ≤ y⌝);
(* *** Goal "1" *** *)
a(all_var_elim_asm_tac1 THEN rewrite_tac[ℝ_ℕ_exp_square_thm]);
a(bc_thm_tac ℝ_0_≤_0_≤_times_thm THEN REPEAT strip_tac);
(* *** Goal "2" *** *)
a(lemma_tac⌜x^2 = Sqrt y^2 ∧ ℕℝ 0 ≤ Sqrt y⌝ THEN1
        (all_fc_tac [sqrt_def] THEN asm_rewrite_tac[]));
a(fc_tac[square_root_unique_thm]);
a(PC_T1 "ℝ_lin_arith" asm_prove_tac[]);
val sqrt_eq_thm = save_pop_thm "sqrt_eq_thm";
```

SML

```
set_goal([], ⌜
        Sqrt (ℕℝ 0) = ℕℝ 0
∧       Sqrt (ℕℝ 1) = ℕℝ 1
∧       Sqrt (ℕℝ 4) = ℕℝ 2
∧       Sqrt (ℕℝ 9) = ℕℝ 3
⌝);
a(REPEAT strip_tac THEN conv_tac eq_sym_conv
        THEN bc_thm_tac sqrt_eq_thm THEN rewrite_tac[ℝ_ℕ_exp_square_thm]);
val sqrt_egs_thm = save_pop_thm "sqrt_egs_thm";
```

SML

```
set_goal([], ⌜∀ x y•
        ℕℝ 0 ≤ x ∧ ℕℝ 0 ≤ y
⇒       (x ^ 2 < y ^ 2 ⇔ x < y)⌝);
a(rewrite_tac[ℝ_≤_def] THEN REPEAT ∀_tac THEN ⇒_tac);
(* *** Goal "1" *** *)
a(ALL_FC_T1 fc_⇔_canon rewrite_tac[square_square_root_mono_thm]);
(* *** Goal "2" *** *)
a(all_var_elim_asm_tac1 THEN rewrite_tac[ℝ_ℕ_exp_square_thm]);
a(lemma_tac⌜ℕℝ 0 < x * x⌝ THEN1
        (bc_thm_tac ℝ_0_less_0_less_times_thm THEN REPEAT strip_tac));
a(PC_T1 "ℝ_lin_arith" asm_prove_tac[]);
(* *** Goal "3" *** *)
a(all_var_elim_asm_tac1 THEN rewrite_tac[ℝ_ℕ_exp_square_thm]);
a(lemma_tac⌜ℕℝ 0 < y * y⌝ THEN1
        (bc_thm_tac ℝ_0_less_0_less_times_thm THEN REPEAT strip_tac));
a(PC_T1 "ℝ_lin_arith" asm_prove_tac[]);
(* *** Goal "4" *** *)
```

```
a(all_var_elim_asm_tac1 THEN rewrite_tac[ℝ_ℕ_exp_square_thm]);
val square_square_root_mono_thm1 = save_pop_thm "square_square_root_mono_thm1";
```

SML

```
set_goal([], ⌜∀x y•ℕℝ 0 ≤ x ∧ ℕℝ 0 ≤ y
⇒       (Sqrt x < Sqrt y ⇔ x < y)
⌝);
a(REPEAT ∀_tac THEN ⇒_tac);
a(all_fc_tac[sqrt_def]);
a(LEMMA_T⌜x < y ⇔Sqrt x^2 < Sqrt y^2⌝ rewrite_thm_tac
        THEN1 asm_rewrite_tac[]);
a(ALL_FC_T1 fc_⇔_canon rewrite_tac[square_square_root_mono_thm1]);
val sqrt_less_thm = save_pop_thm "sqrt_less_thm";
```

B Proof 1 — Proofs

SML

```
|
```

The first proof is based on a geometrical construction which viewed algebraically goes a follows: if $x^2 = 2y^2$, then $y \leq x \leq (3/2)y$ and $(2y - x)^2 = 2(x - y)^2$. Whence, if x and y are positive integers with $x^2 = 2y^2$, then so also are $x' = 2y - x$ and $y' = x - y$. But $y' < y$ which leads to a contradiction, since if there is some solution to $x^2 = 2y^2$, there must be one for which y is minimal.

SML

```
|open_theory "sqrt2_defs";
|force_delete_theory "sqrt2_proof1" handle Fail _ => ();
|new_theory "sqrt2_proof1";
|set_merge_pcs["'ℤ", "'ℝ", "'sets_alg", "basic_hol1" ];
```

Now sneak up on the result in a series of lemmas.

Step 1: if $x^2 = 2y^2$, then $y < x \leq (3/2)y$, and $(2y - x)^2 = 2(x - y)^2$:

SML

```
|set_goal([], proof1_lemma1);
|a(rewrite_tac[ℝ_ℕ_exp_square_thm] THEN contr_tac);
|(* *** Goal "1" *** *)
|a(cases_tac⌜y = x⌝ THEN1 all_var_elim_asm_tac1);
|(* *** Goal "1.1" *** *)
|a(LEMMA_T⌜x*x = ℕℝ 0⌝ ante_tac THEN1 PC_T1 "ℝ_lin_arith" asm_prove_tac[]);
|a(rewrite_tac[ℝ_times_eq_0_thm] THEN  PC_T1 "ℝ_lin_arith" asm_prove_tac[]);
|(* *** Goal "1.2" *** *)
|a(lemma_tac⌜x*y < y*y⌝ THEN1
|        once_rewrite_tac[ℝ_times_comm_thm] THEN1
|        bc_thm_tac ℝ_times_mono_thm THEN1
|                PC_T1 "ℝ_lin_arith" asm_prove_tac[]);
|a(lemma_tac⌜x*x ≤ x*y⌝ THEN1
|        bc_thm_tac ℝ_≤_times_mono_thm THEN1
|                PC_T1"ℝ_lin_arith" asm_prove_tac[]);
|a(LEMMA_T⌜y * ℕℝ 0 < y*y⌝ (strip_asm_tac o rewrite_rule[]) THEN1
|        bc_thm_tac ℝ_times_mono_thm THEN1
|                PC_T1"ℝ_lin_arith" asm_prove_tac[]);
|a(all_fc_tac[ℝ_≤_less_trans_thm]
|        THEN PC_T1"ℝ_lin_arith" asm_prove_tac[]);
|(* *** Goal "2" *** *)
|a(lemma_tac⌜(ℕℝ 3*y)*(ℕℝ 2*x) < (ℕℝ 2*x)*(ℕℝ 2*x)⌝ THEN1
|        conv_tac(RANDS_C (eq_match_conv ℝ_times_comm_thm)) THEN1
|        bc_thm_tac ℝ_times_mono_thm THEN1
|                PC_T1 "ℝ_lin_arith" asm_prove_tac[]);
|a(lemma_tac⌜(ℕℝ 3*y)*(ℕℝ 3*y) ≤ (ℕℝ 3*y)*(ℕℝ 2*x)⌝ THEN1
|        bc_thm_tac ℝ_≤_times_mono_thm THEN1
|                PC_T1"ℝ_lin_arith" asm_prove_tac[]);
|
```

```
a(LEMMA_T⌜x * ℕℝ 0 < x*x⌝ (strip_asm_tac o rewrite_rule[]) THEN1
        bc_thm_tac ℝ_times_mono_thm THEN1
                PC_T1"ℝ_lin_arith" asm_prove_tac[]);
a(all_fc_tac[ℝ_≤_less_trans_thm]
        THEN PC_T1"ℝ_lin_arith" asm_prove_tac[]);
(* *** Goal "3" *** *)
a(PC_T1"ℝ_lin_arith" asm_prove_tac[]);
val proof1_lemma1_thm = save_pop_thm "proof1_lemma1_thm";
```

Step 1: tiny fact about the conversion of naturals to reals

```
SML
set_goal([], ⌜∀i j•j ≤ i ⇒ ℕℝ(i − j) = ℕℝ i − ℕℝ j⌝);
a(rewrite_tac[≤_def] THEN REPEAT strip_tac THEN
        all_var_elim_asm_tac1);
a(rewrite_tac[∀_elim⌜i'⌝ plus_order_thm,
        ℕℝ_plus_homomorphism_thm]
                THEN PC_T1 "ℝ_lin_arith" prove_tac[]);
val proof1_lemma2_thm = save_pop_thm "proof1_lemma2_thm";
```

Step 3: this is step 1 pulled back to the natural numbers:

```
SML
set_goal([], ⌜∀m n•
        ℕℝ m ^ 2 = ℕℝ 2 * ℕℝ n ^ 2 ∧ 0 < n
⇒       n < m ∧ 2 * m ≤ 3 * n
∧       ℕℝ (2 * n − m) ^ 2 = ℕℝ 2 * ℕℝ (m − n) ^ 2
⌝);
a(REPEAT ∀_tac THEN ⇒_tac);
a(lemma_tac ⌜ℕℝ 0 ≤ ℕℝ m ∧ ℕℝ 0 < ℕℝ n⌝ THEN1
        asm_rewrite_tac[ℕℝ_≤_thm, ℕℝ_less_thm]);
a(ALL_FC_T (MAP_EVERY ante_tac) [proof1_lemma1_thm]);
a(rewrite_tac[ℕℝ_≤_thm, ℕℝ_less_thm,
        conv_rule (ONCE_MAP_C eq_sym_conv)
        ℕℝ_times_homomorphism_thm] THEN REPEAT strip_tac);
a(lemma_tac⌜m ≤ 2*n ∧n ≤ m⌝ THEN1 PC_T1  "lin_arith" asm_prove_tac[]);
a(ALL_FC_T asm_rewrite_tac[proof1_lemma2_thm]);
val proof1_lemma3_thm = save_pop_thm "proof1_lemma3_thm";
```

Step 4: if m and n are positive integer solutions to $m^2 = 2n^2$, then there is a solution with smaller n:

SML

```
set_goal([], proof1_lemma4);
a(REPEAT strip_tac THEN all_fc_tac[proof1_lemma3_thm]);
a(∃_tac⌜2*n − m⌝ THEN ∃_tac⌜m − n⌝ THEN asm_rewrite_tac[]);
a(LEMMA_T ⌜n ≤ m⌝ (strip_asm_tac o rewrite_rule[≤_def])
        THEN1 PC_T1 "lin_arith" asm_prove_tac[]);
a(all_var_elim_asm_tac1);
a(rewrite_tac[∀_elim⌜i⌝ plus_order_thm]);
a(PC_T1 "lin_arith" asm_prove_tac[]);
val proof1_lemma4_thm = save_pop_thm "proof1_lemma4_thm";
```

Step 5: the induction that shows the only natural number solutions to $m^2 = 2n^2$ has $m = 0$:

SML

```
set_goal([], proof1_lemma5);
a(∀_tac THEN cov_induction_tac⌜n:ℕ⌝ THEN REPEAT strip_tac);
a(contr_tac THEN lemma_tac ⌜0 < n⌝ THEN1
        PC_T1 "lin_arith" asm_prove_tac[]);
a(all_fc_tac[proof1_lemma4_thm]);
a(all_asm_fc_tac[] THEN all_var_elim_asm_tac1);
val proof1_lemma5_thm = save_pop_thm "proof1_lemma5_thm";
```

. . . which gives what we wanted, expressed explicitly:

SML

```
set_goal([], proof1_thm1);
a(REPEAT strip_tac);
a(lemma_tac⌜¬ℕℝ b = ℕℝ 0⌝ THEN1
        asm_rewrite_tac[ℕℝ_one_one_thm]);
a(rewrite_tac[ℝ_frac_def] THEN ALL_FC_T rewrite_tac[ℝ_over_times_recip_thm]);
a(contr_tac THEN LEMMA_T⌜
        (ℕℝ a * ℕℝ b ^ -1) ^ 2 * ℕℝ b ^ 2 = ℕℝ 2 * ℕℝ b ^ 2⌝ ante_tac
        THEN1 asm_rewrite_tac[]);
a(rewrite_tac[]);
a(LEMMA_T⌜∀x y z:ℝ•(x*y)^2*z^2 = (x*z*y)^2⌝ rewrite_thm_tac THEN1
        (rewrite_tac[ℝ_ℕ_exp_square_thm]
                THEN PC_T1"ℝ_lin_arith" prove_tac[]));
a(ALL_FC_T rewrite_tac[ℝ_times_recip_thm]);
a(contr_tac THEN all_fc_tac[proof1_lemma5_thm]);
val proof1_thm1_thm = save_pop_thm "proof1_thm1_thm";
```

A lemma extending the above to the case when $a/b \leq 0$.

SML

```
set_goal([], ⌜
        ∀ a b• ¬ b = 0 ⇒ ¬(a/b)^2 = ℕℝ 2 ∧ ¬(∼(a/b))^2 = ℕℝ 2
⌝);
a(rewrite_tac[proof1_thm1_thm, square_even_thm]);
val proof1_thm1a = save_pop_thm "proof1_thm1a";
```

The final result in terms of the square root function and the set $\mathbb{Q}$ (which requires us also to deal with the possibility that a/b is negative).

SML

```
set_goal([], ⌜ ¬Sqrt (ℕℝ 2) ∈ ℚ ⌝);
a(rewrite_tac[get_spec⌜ℚ⌝] THEN REPEAT_UNTIL is_∨ strip_tac);
a(cases_tac ⌜b = 0⌝ THEN asm_rewrite_tac[]);
a(contr_tac THEN
        (LEMMA_T ⌜Sqrt(ℕℝ 2)^2 = ℕℝ 2⌝ ante_tac THEN1
                bc_tac(map (rewrite_rule[]) (fc_canon (get_spec⌜Sqrt⌝))))
                THEN ALL_FC_T asm_rewrite_tac[proof1_thm1a]);
val proof1_thm2 = save_pop_thm "proof1_thm2";
```

C Divisibility — Proofs

The other two proofs we give are the better known ones based on divisibility. They share a great deal of common material about primes, divisibility etc.

SML
```
open_theory "divisibility";
set_merge_pcs["'sets_alg", "basic_hol1" ];
```

SML
```
val prime_def = get_spec⌜Prime⌝;
```

SML
```

set_goal([], ⌜∀n a• n ∈ a ⇒ Min a ∈ a⌝);
a(∀_tac THEN cov_induction_tac ⌜n:ℕ⌝ THEN REPEAT strip_tac);
a(cases_tac⌜∃m• m < n ∧ m ∈ a⌝);
(* *** Goal "1" *** *)
a(all_asm_fc_tac[]);
(* *** Goal "2" *** *)
a(LEMMA_T ⌜Min a = n⌝ asm_rewrite_thm_tac);
a(bc_thm_tac(get_spec⌜Min⌝) THEN REPEAT strip_tac);
a(spec_nth_asm_tac 2 ⌜i⌝);
a(PC_T1 "lin_arith" asm_prove_tac[]);
val min_∈_thm = save_pop_thm"min_∈_thm";

```

SML
```

set_goal([], ⌜∀n a• n ∈ a ⇒ Min a ≤ n⌝);
a(∀_tac THEN cov_induction_tac ⌜n:ℕ⌝ THEN REPEAT strip_tac);
a(cases_tac⌜∃m• m < n ∧ m ∈ a⌝);
(* *** Goal "1" *** *)
a(all_asm_fc_tac[]);
a(PC_T1 "lin_arith" asm_prove_tac[]);
(* *** Goal "2" *** *)
a(LEMMA_T ⌜Min a = n⌝ rewrite_thm_tac);
a(bc_thm_tac(get_spec⌜Min⌝) THEN REPEAT strip_tac);
a(spec_nth_asm_tac 2 ⌜i⌝);
a(PC_T1 "lin_arith" asm_prove_tac[]);
val min_≤_thm = save_pop_thm"min_≤_thm";

```

SML
```
val div_mod_unique_thm1 =
        rewrite_rule[taut_rule
                ⌜∀p1 p2 p3• (p1 ⇒ p2 ⇒ p3) ⇔ (p1 ∧ p2 ⇒ p3)⌝]
        (conv_rule (ONCE_MAP_C(RAND_C(RAND_C (RANDS_C eq_sym_conv))))
                div_mod_unique_thm);
```

SML

```
set_goal([], ⌜∀m n•m * n = 0 ⇒ m = 0 ∨ n = 0⌝);
a(contr_tac);
a(LEMMA_T⌜1 ≤ m ∧ 1 ≤ n⌝ (strip_asm_tac o rewrite_rule[≤_def])
        THEN1 PC_T1 "lin_arith" asm_prove_tac[]);
a(all_var_elim_asm_tac1
        THEN PC_T1 "lin_arith" asm_prove_tac[]);
val times_eq_0_thm = save_pop_thm "times_eq_0_thm";
```

SML

```
set_goal([], ⌜∀k m n•0 < k ∧ k * m = k * n ⇒ m = n⌝);
a(contr_tac);
a((LEMMA_T⌜m ≤ n ∨ n ≤ m⌝ (strip_asm_tac o rewrite_rule[≤_def])
        THEN1 rewrite_tac[≤_cases_thm])
                THEN all_var_elim_asm_tac1);
(* *** Goal "1" *** *)
a(lemma_tac⌜k*i = 0⌝ THEN1 PC_T1 "lin_arith" asm_prove_tac[]);
a(fc_tac[times_eq_0_thm] THEN all_var_elim_asm_tac1
        THEN1 PC_T1 "lin_arith" asm_prove_tac[]);
(* *** Goal "2" *** *)
a(lemma_tac⌜k*i = 0⌝ THEN1 PC_T1 "lin_arith" asm_prove_tac[]);
a(fc_tac[times_eq_0_thm] THEN all_var_elim_asm_tac1
        THEN1 PC_T1 "lin_arith" asm_prove_tac[]);
val times_cancel_thm = save_pop_thm "times_cancel_thm";
```

SML

```
set_goal([], ⌜∀m n•0 < n ∧ m * n = n ⇒ m = 1⌝);
a(REPEAT strip_tac THEN
        bc_thm_tac (once_rewrite_rule[times_comm_thm] times_cancel_thm));
a(∃_tac⌜n⌝ THEN asm_rewrite_tac[]);
val times_eq_eq_1_thm = save_pop_thm "times_eq_eq_1_thm";
```

SML

```
set_goal([], ⌜∀m n•m * n = 1 ⇒ m = 1 ∧ n = 1⌝);
a(REPEAT ∀_tac);
a(cases_tac⌜m = 0 ∨ n = 0 ∨ m = 1 ∨ n = 1⌝ THEN_TRY
        (all_var_elim_asm_tac1 THEN PC_T1 "lin_arith" asm_prove_tac[]));
a(LEMMA_T⌜2 ≤ m ∧ 2 ≤ n⌝ (strip_asm_tac o rewrite_rule[≤_def])
        THEN1 PC_T1 "lin_arith" asm_prove_tac[]);
a(LIST_DROP_NTH_ASM_T [3, 4, 5, 6] discard_tac);
a(all_var_elim_asm_tac1 THEN PC_T1 "lin_arith" asm_prove_tac[]);
val times_eq_1_thm = save_pop_thm "times_eq_1_thm";
```

SML

```
set_goal([], ⌜∀m•m Div 1 = m ∧ m Mod 1 = 0⌝);
a(∀_tac);
a(bc_thm_tac div_mod_unique_thm1 THEN rewrite_tac[]);
val div_mod_1_thm = save_pop_thm "div_mod_1_thm";
```

SML

```
set_goal([], ⌜∀m• 0 < m ⇒ m Div m = 1 ∧ m Mod m = 0⌝);
a(∀_tac THEN ⇒_tac);
a(bc_thm_tac div_mod_unique_thm1 THEN asm_rewrite_tac[]);
val m_div_mod_m_thm = save_pop_thm "m_div_mod_m_thm";
```

SML

```
set_goal([], ⌜∀m•0 < m ⇒ 0 Div m = 0 ∧ 0 Mod m = 0⌝);
a(∀_tac THEN ⇒_tac);
a(bc_thm_tac div_mod_unique_thm1 THEN asm_rewrite_tac[]);
val zero_div_mod_thm = save_pop_thm "zero_div_mod_thm";
```

SML

```
set_goal([], ⌜∀m n•n < m ⇒ n Div m = 0 ∧ n Mod m = n⌝);
a(REPEAT ∀_tac THEN ⇒_tac);
a(bc_thm_tac div_mod_unique_thm1 THEN asm_rewrite_tac[]);
val less_div_mod_thm = save_pop_thm "less_div_mod_thm";
```

SML

```
set_goal([], ⌜∀k m n•0 < k ⇒ (m*k + n) Div k = m + n Div k ∧ (m*k + n) Mod k = n Mod k⌝);
a(REPEAT ∀_tac THEN ⇒_tac);
a(bc_thm_tac div_mod_unique_thm1
        THEN ALL_FC_T rewrite_tac[mod_less_thm]);
a(rewrite_tac[times_plus_distrib_thm, plus_assoc_thm]);
a(bc_thm_tac div_mod_thm THEN REPEAT strip_tac);
val div_mod_times_cancel_thm = save_pop_thm "div_mod_times_cancel_thm";
```

SML

```
set_goal([], ⌜∀k m n•
        0 < k
⇒       (m*k) Mod k = 0
∧       (k*m) Mod k = 0
∧       (k*m + n) Mod k = n Mod k
∧       (m*k + n) Mod k = n Mod k
∧       (k + n) Mod k = n Mod k
∧       (n + k) Mod k = n Mod k
∧       0 Mod k = 0
∧       k Mod k = 0
∧       (m Mod k) Mod k = m Mod k
```

```
⌝);
a(REPEAT ∀_tac THEN ⇒_tac);
a(rewrite_tac[∀_elim⌜k⌝ times_comm_thm]);
a(ALL_FC_T rewrite_tac[
        div_mod_times_cancel_thm, m_div_mod_m_thm, zero_div_mod_thm]);
a(pure_once_rewrite_tac[prove_rule[]
        ⌜m*k = m*k + 0 ∧ k + n = 1*k + n ∧ n + k = 1*k + n⌝]);
a(ALL_FC_T pure_rewrite_tac[div_mod_times_cancel_thm]
        THEN ALL_FC_T rewrite_tac[zero_div_mod_thm]);
a(lemma_tac⌜m Mod k < k⌝ THEN1 ALL_FC_T rewrite_tac[mod_less_thm]);
a(ALL_FC_T rewrite_tac[less_div_mod_thm]);
val mod_clauses = save_pop_thm "mod_clauses";
```

SML

```
set_goal([], ⌜∀m n•
        0 < n
⇒       (m Mod n = 0 ⇔ ∃k•m = k*n)
⌝);
a(REPEAT strip_tac);
(* *** Goal "1" *** *)
a(∃_tac⌜m Div n⌝);
a(ALL_FC_T (conv_tac o LEFT_C o once_rewrite_conv)[div_mod_thm]);
a(asm_rewrite_tac[]);
(* *** Goal "2" *** *)
a(all_var_elim_asm_tac1 THEN ALL_FC_T rewrite_tac[mod_clauses]);
val mod_eq_0_thm = save_pop_thm "mod_eq_0_thm";
```

SML

```
set_goal([], ⌜∀m n•
        0 < m ∧ 0 < n ∧ m Mod n = 0 ∧ n Mod m = 0
⇒       m = n
⌝);
a(REPEAT strip_tac THEN all_fc_tac[mod_eq_0_thm]);
a(lemma_tac⌜(k'*k)*m = m⌝ THEN1
        (POP_ASM_T (fn th => conv_tac (RIGHT_C(rewrite_conv[th])))
                THEN asm_rewrite_tac[times_assoc_thm]));
a(all_fc_tac[times_eq_eq_1_thm]);
a(all_fc_tac[times_eq_1_thm]);
a(all_var_elim_asm_tac1 THEN rewrite_tac[]);
val mod_eq_0_mod_eq_0_thm = save_pop_thm "mod_eq_0_mod_eq_0_thm";
```

SML

```
set_goal([], ⌜∀m n k•0 < k ⇒ (m + n) Mod k = (m Mod k + n Mod k) Mod k⌝);
a(REPEAT strip_tac);
a(all_fc_tac[div_mod_thm]);
```

```
a(TOP_ASM_T (ante_tac o ∀_elim⌜m⌝));
a(POP_ASM_T (ante_tac o ∀_elim⌜n⌝) THEN REPEAT strip_tac);
a(REPEAT (POP_ASM_T (fn th => conv_tac(LEFT_C(once_rewrite_conv[th])))));
a(rewrite_tac[pc_rule1 "lin_arith" prove_rule[]
        ⌜∀a b c d•(a*k + b) + (c*k + d) = (a+c)*k + b + d⌝]);
a(ALL_FC_T rewrite_tac[div_mod_times_cancel_thm]);
val mod_plus_homomorphism_thm = save_pop_thm "mod_plus_homomorphism_thm";
```

SML

```
set_goal([], ⌜∀m n k•0 < k ⇒ (m * n) Mod k = ((m Mod k) * (n Mod k)) Mod k⌝);
a(REPEAT strip_tac);
a(all_fc_tac[div_mod_thm]);
a(TOP_ASM_T (ante_tac o ∀_elim⌜m⌝));
a(POP_ASM_T (ante_tac o ∀_elim⌜n⌝) THEN REPEAT strip_tac);
a(REPEAT (POP_ASM_T (fn th => conv_tac(LEFT_C(once_rewrite_conv[th])))));
a(rewrite_tac[pc_rule1 "lin_arith" prove_rule[]
        ⌜∀a b c d•(a*k + b) * (c*k + d) = (a*c*k + a*d +b*c)*k + b*d⌝]);
a(ALL_FC_T rewrite_tac[div_mod_times_cancel_thm]);
val mod_times_homomorphism_thm = save_pop_thm "mod_times_homomorphism_thm";
```

SML

```
set_goal([], ⌜∀m n•
        0 < m ∧ 0 < n ∧ 0 < m Mod n
⇒       ∃a• 0 < (a*m) Mod n
∧       ∀b• 0 < (b*m) Mod n ⇒ (a*m) Mod n ≤ (b*m) Mod n⌝);
a(REPEAT strip_tac);
a(PC_T1 "predicates" lemma_tac
        ⌜ ∃i• i ∈ { i | ∃a•0 < (a*m) Mod n ∧ i = (a*m) Mod n } ⌝);
(* *** Goal "1" *** *)
a(∃_tac⌜m Mod n⌝ THEN REPEAT strip_tac);
a(∃_tac⌜1⌝ THEN asm_rewrite_tac[]);
(* *** Goal "2" *** *)
a(all_fc_tac[min_∈_thm]);
a(∃_tac⌜a⌝ THEN REPEAT strip_tac);
a(DROP_NTH_ASM_T 4 discard_tac);
a(PC_T1 "predicates" lemma_tac
        ⌜ (b*m) Mod n ∈ { i | ∃a•0 < (a*m) Mod n ∧ i = (a*m) Mod n } ⌝
        THEN1 (REPEAT strip_tac THEN asm_prove_tac[]));
a(all_fc_tac[min_≤_thm]);
a(POP_ASM_T ante_tac THEN asm_rewrite_tac[]);
val gcd_consistent_lemma1 = save_pop_thm"gcd_consistent_lemma1";
```

SML

```
set_goal([], ⌜∀m n a d•
        0 < m ∧ 0 < n ∧ 0 < d
∧       m Mod d = 0 ∧ n Mod d = 0
⇒       ((a*m) Mod n) Mod d = 0
⌝);
a(REPEAT strip_tac);
a(all_fc_tac[mod_eq_0_thm]);
a(ante_tac(list_∀_elim[⌜a*m⌝, ⌜n⌝] div_mod_thm));
a(strip_tac THEN LEMMA_T
        ⌜(((a*m) Div n)*n + (a*m) Mod n) Mod d = (a*m) Mod d⌝ ante_tac
        THEN1 POP_ASM_T (rewrite_thm_tac o eq_sym_rule));
a(ALL_FC_T once_rewrite_tac[mod_plus_homomorphism_thm]);
a(all_var_elim_asm_tac1
        THEN rewrite_tac[conv_rule (ONCE_MAP_C eq_sym_conv) times_assoc_thm]
        THEN ALL_FC_T rewrite_tac[mod_clauses]);
val gcd_consistent_lemma2 = save_pop_thm"gcd_consistent_lemma2";
```

SML

```
set_goal([], ⌜∀a b m n p r s•
        a*m = b*(n+1) + r
∧       m = p*r + s
⇒       ∃q•(q*m) Mod (n+1) = s Mod (n+1)⌝);
a(REPEAT strip_tac);
a(∃_tac⌜n*p*a+1⌝);
a(scale_nth_asm_tac⌜n*p⌝ 2);
a(rewrite_tac[times_assoc_thm, times_plus_distrib_thm] THEN
        POP_ASM_T (asm_rewrite_thm_tac o rewrite_rule[times_assoc_thm]));
a(LEMMA_T ⌜
        n*p*(b*(n + 1) + r) + p * r + s =
        (n+1)*(p*b*n + p*r) + s⌝ rewrite_thm_tac THEN1
        PC_T1 "lin_arith" prove_tac[]);
a(rewrite_tac[rewrite_rule[](∀_elim⌜n+1⌝mod_clauses)]);
val gcd_consistent_lemma3 = save_pop_thm"gcd_consistent_lemma3";
```

SML

```
set_goal([], ⌜∀a b m n p r s•
        a*m = b*(n+1) + r
∧       n + 1 = p*r + s
⇒       ∃q•(q*m) Mod (n+1) = s Mod (n+1)⌝);
a(REPEAT strip_tac);
a(LEMMA_T⌜∀x•(x*m) Mod (n + 1) = (x*m + n + 1) Mod (n + 1)⌝
        rewrite_thm_tac THEN1
                rewrite_tac[rewrite_rule[](∀_elim⌜n+1⌝mod_clauses)]);
a(∃_tac⌜n*p*a⌝);
```

```
a(scale_nth_asm_tac⌜n*p⌝ 2);
a(rewrite_tac[times_assoc_thm, times_plus_distrib_thm] THEN
        POP_ASM_T (rewrite_thm_tac o rewrite_rule[times_assoc_thm]));
a(LEMMA_T⌜∀x•x + n + 1 = x + p*r + s⌝ rewrite_thm_tac THEN1
        asm_rewrite_tac[]);
a(LEMMA_T ⌜
        n*p*(b*(n+1) + r) + p * r + s =
        (n+1)*(p*b*n + p*r) + s⌝ rewrite_thm_tac THEN1
        PC_T1 "lin_arith" prove_tac[]);
a(rewrite_tac[rewrite_rule[](∀_elim⌜n+1⌝mod_clauses)]);
val gcd_consistent_lemma4 = save_pop_thm"gcd_consistent_lemma4";
```

SML

```
set_goal([], ⌜∀m n k a•
        0 < m ∧ 0 < n ∧ 0 < m Mod n
∧       0 < (a*m) Mod n
∧       (∀b• 0 < (b*m) Mod n ⇒ (a*m) Mod n ≤ (b*m) Mod n)
⇒       m Mod ((a*m) Mod n) = 0 ∧ n Mod ((a*m) Mod n) = 0⌝);
a(REPEAT ∀_tac THEN ⇒_tac THEN
        LEMMA_T ⌜1 ≤ n⌝
                (strip_asm_tac o once_rewrite_rule[plus_comm_thm]
                        o rewrite_rule[≤_def]) THEN1
        PC_T1 "lin_arith" asm_prove_tac[]);
a(all_var_elim_asm_tac1 THEN contr_tac);
(* *** Goal "1" *** *)
a(ante_tac(list_∀_elim[ ⌜a⌝, ⌜(a*m) Div (i+1)⌝,
        ⌜m⌝, ⌜i⌝, ⌜m Div ((a*m) Mod (i+1))⌝,
        ⌜(a*m) Mod (i+1)⌝,
        ⌜m Mod ((a*m) Mod (i+1))⌝ ]gcd_consistent_lemma3));
a(asm_tac (prove_rule[]⌜0 < i + 1⌝));
a(ALL_FC_T rewrite_tac[
        conv_rule(ONCE_MAP_C eq_sym_conv) div_mod_thm]);
a(lemma_tac⌜m Mod ((a*m) Mod (i + 1)) < (a*m) Mod (i+1)⌝
        THEN1 EXTEND_PC_T1 "'mmp1" all_fc_tac[mod_less_thm]);
a(lemma_tac⌜(a*m) Mod (i+1) < (i+1)⌝
        THEN1 (bc_thm_tac mod_less_thm THEN REPEAT strip_tac));
a(lemma_tac⌜m Mod ((a*m) Mod (i + 1)) < (i+1)⌝
        THEN1 EXTEND_PC_T1 "'mmp1" all_fc_tac[less_trans_thm]);
a(ALL_FC_T rewrite_tac [less_div_mod_thm]);
a(contr_tac);
a(DROP_NTH_ASM_T 7 (ante_tac o ∀_elim⌜q⌝));
a(asm_rewrite_tac[]);
a(PC_T1 "lin_arith" asm_prove_tac[]);
(* *** Goal "2" *** *)
a(ante_tac(list_∀_elim[ ⌜a⌝, ⌜(a*m) Div (i+1)⌝,
```

```
        ⌜m⌝, ⌜i⌝, ⌜(i+1) Div ((a*m) Mod (i+1))⌝,
        ⌜(a*m) Mod (i+1)⌝,
        ⌜(i+1) Mod ((a*m) Mod (i+1))⌝ ]gcd_consistent_lemma4));
a(asm_tac (prove_rule[]⌜0 < i + 1⌝));
a(ALL_FC_T rewrite_tac[
        conv_rule(ONCE_MAP_C eq_sym_conv) (div_mod_thm)]);
a(lemma_tac⌜(i+1) Mod ((a*m) Mod (i+1)) < (a*m) Mod (i+1)⌝
        THEN1 (bc_thm_tac mod_less_thm THEN REPEAT strip_tac));
a(lemma_tac⌜(a*m) Mod (i+1) < (i+1)⌝
        THEN1 (bc_thm_tac mod_less_thm THEN REPEAT strip_tac));
a(lemma_tac⌜(i+1) Mod ((a*m) Mod (i + 1)) < (i+1)⌝
        THEN1 all_fc_tac[less_trans_thm]);
a(ALL_FC_T rewrite_tac [less_div_mod_thm]);
a(contr_tac);
a(DROP_NTH_ASM_T 7 (ante_tac o ∀_elim⌜q⌝));
a(asm_rewrite_tac[]);
a(PC_T1 "lin_arith" asm_prove_tac[]);
val gcd_consistent_lemma5 = save_pop_thm"gcd_consistent_lemma5";
```

SML

```
push_consistency_goal ⌜Gcd⌝;
a(prove_∃_tac THEN REPEAT strip_tac);
a(cases_tac⌜m' = 0⌝ THEN1 all_var_elim_asm_tac1);
(* *** Goal "1" *** *)
a(∃_tac⌜0⌝ THEN REPEAT strip_tac THEN
        ALL_FC_T rewrite_tac[mod_clauses]);
(* *** Goal "2" *** *)
a(cases_tac⌜n' = 0⌝ THEN1 all_var_elim_asm_tac1);
(* *** Goal "2.1" *** *)
a(∃_tac⌜0⌝ THEN REPEAT strip_tac THEN
        ALL_FC_T rewrite_tac[mod_clauses]);
(* *** Goal "2.2" *** *)
a(cases_tac⌜m' Mod n' = 0⌝);
(* *** Goal "2.2.1" *** *)
a(∃_tac⌜n'⌝ THEN REPEAT strip_tac THEN
        ALL_FC_T rewrite_tac[mod_clauses]);
(* *** Goal "2.2.2" *** *)
a(lemma_tac⌜0 < m' ∧ 0 < n' ∧ 0 < m' Mod n'⌝
        THEN1 PC_T1 "lin_arith" asm_prove_tac[]
        THEN LIST_DROP_NTH_ASM_T [4, 5, 6] discard_tac);
a(all_fc_tac [gcd_consistent_lemma1]);
a(all_fc_tac [gcd_consistent_lemma5]);
a(∃_tac⌜(a*m') Mod n'⌝ THEN asm_rewrite_tac[]);
a(REPEAT strip_tac);
a(bc_thm_tac gcd_consistent_lemma2
```

```
        THEN REPEAT strip_tac);
val _ = save_consistency_thm ⌜Gcd⌝ (pop_thm());
```

SML

```
val gcd_def = get_spec⌜Gcd⌝;
```

SML

```
set_goal([], gcd_eq_mod_thm);
a(REPEAT strip_tac);
a(all_fc_tac[gcd_consistent_lemma1]);
a(∃_tac⌜a⌝ THEN asm_rewrite_tac[]);
a(lemma_tac⌜Gcd m n Mod ((a * m) Mod n) = 0⌝ THEN1
        (all_fc_tac[gcd_consistent_lemma5]
                THEN all_fc_tac[∧_right_elim gcd_def]));
a(lemma_tac⌜((a * m) Mod n) Mod Gcd m n = 0⌝
        THEN1 (bc_thm_tac gcd_consistent_lemma2
                THEN all_fc_tac[∧_left_elim gcd_def]
                        THEN REPEAT strip_tac));
a(lemma_tac⌜0 < Gcd m n⌝
        THEN1 (all_fc_tac[∧_left_elim gcd_def]));
a(all_fc_tac[mod_eq_0_mod_eq_0_thm]);
val gcd_eq_mod_thm = save_pop_thm"gcd_eq_mod_thm";
```

SML

```
set_goal([], ⌜∀p• p ∈ Prime ⇒ 0 < p⌝);
a(rewrite_tac[prime_def] THEN PC_T1 "lin_arith" asm_prove_tac[]);
val prime_0_less_thm = save_pop_thm"prime_0_less_thm";
```

SML

```
set_goal([], ⌜∀m p•
        0 < m ∧ p ∈ Prime
⇒       Gcd m p = 1 ∨ Gcd m p = p⌝);
a(REPEAT strip_tac
        THEN all_fc_tac[prime_0_less_thm]
                THEN all_asm_ante_tac);
a(rewrite_tac[prime_def] THEN REPEAT strip_tac);
a(all_fc_tac[prime_0_less_thm]);
a(lemma_tac ⌜0 < Gcd m p⌝ THEN1 all_fc_tac[gcd_def]);
a(LEMMA_T ⌜p Mod Gcd m p = 0⌝ ante_tac THEN1 all_fc_tac[gcd_def]);
a(ALL_FC_T1 fc_⇔_canon rewrite_tac[mod_eq_0_thm]);
a(REPEAT strip_tac);
a(LIST_DROP_NTH_ASM_T [5] fc_tac);
a(all_var_elim_asm_tac1);
a(POP_ASM_T (ante_tac o eq_sym_rule) THEN rewrite_tac[]);
val gcd_prime_thm = save_pop_thm"gcd_prime_thm";
```

SML

```
set_goal([], ⌜∀p•
        (∀m n•          (m*n) Mod p = 0
        ⇒       m Mod p = 0 ∨ n Mod p = 0)
∧       1 < p
⇒       p ∈ Prime
⌝);
a(rewrite_tac[prime_def] THEN contr_tac);
a(all_var_elim_asm_tac1);
a(DROP_NTH_ASM_T 4 (ante_tac o list_∀_elim[⌜m⌝, ⌜n⌝]));
a(lemma_tac ⌜0 < m ∧ 0 < n⌝ THEN1
        (contr_tac THEN lemma_tac⌜m = 0 ∨ n = 0⌝
                THEN_TRY all_var_elim_asm_tac1 THEN
                        PC_T1"lin_arith" asm_prove_tac[]));
a(lemma_tac ⌜¬m*n = 0⌝ THEN1
        (contr_tac THEN all_fc_tac[times_eq_0_thm]
                THEN PC_T1 "lin_arith" asm_prove_tac[]));
a(lemma_tac ⌜0 < m*n ⌝ THEN1
        PC_T1"lin_arith" asm_prove_tac[]);
a(ALL_FC_T rewrite_tac[m_div_mod_m_thm]);
a(cases_tac⌜m < m*n⌝ THEN1
        ALL_FC_T asm_rewrite_tac[less_div_mod_thm]);
(* *** Goal "1" *** *)
a(cases_tac⌜n < m*n⌝ THEN1
        (ALL_FC_T asm_rewrite_tac[less_div_mod_thm] THEN
                PC_T1 "lin_arith" asm_prove_tac[]));
a(LEMMA_T⌜2 ≤ m⌝ (strip_asm_tac o rewrite_rule[≤_def])
        THEN1 PC_T1"lin_arith" asm_prove_tac[]);
a(all_var_elim_asm_tac1 THEN PC_T1"lin_arith" asm_prove_tac[]);
(* *** Goal "2" *** *)
a(LEMMA_T⌜2 ≤ n⌝ (strip_asm_tac o rewrite_rule[≤_def])
        THEN1 PC_T1"lin_arith" asm_prove_tac[]);
a(all_var_elim_asm_tac1 THEN PC_T1"lin_arith" asm_prove_tac[]);
val prime_lemma1 = pop_thm ();
```

SML

```
set_goal([], ⌜∀p m n•
        p ∈ Prime ∧ (m*n) Mod p = 0
⇒       m Mod p = 0 ∨ n Mod p = 0
⌝);
a(REPEAT ∀_tac THEN ⇒_tac);
a(all_fc_tac[prime_0_less_thm]);
a(cases_tac⌜m = 0 ∨ n = 0⌝ THEN_TRY
        (all_var_elim_asm_tac1 THEN
                ALL_FC_T rewrite_tac[mod_clauses]));
```

```
a(lemma_tac⌜0 < m ∧ 0 < n⌝ THEN1 PC_T1 "lin_arith" asm_prove_tac[]);
a(cases_tac⌜Gcd m p = p⌝ THEN1
        (POP_ASM_T (once_rewrite_thm_tac o eq_sym_rule) THEN
                all_fc_tac[gcd_def] THEN REPEAT strip_tac));
a(cases_tac⌜Gcd n p = p⌝ THEN1
        (POP_ASM_T (once_rewrite_thm_tac o eq_sym_rule) THEN
                all_fc_tac[(∧_left_elim gcd_def)]
                        THEN REPEAT strip_tac));
a(lemma_tac⌜Gcd m p = 1 ∧ Gcd n p = 1⌝ THEN1
        (fc_tac[gcd_prime_thm]
                THEN LIST_DROP_NTH_ASM_T [1, 2] fc_tac
                        THEN REPEAT strip_tac));
a(rewrite_tac[pc_rule1 "lin_arith" prove_rule[]
        ⌜∀a•a = 0 ⇔ ¬0 < a⌝]);
a(contr_tac);
a(all_fc_tac[gcd_eq_mod_thm]);
a(LIST_DROP_NTH_ASM_T [1, 4] (MAP_EVERY ante_tac));
a(asm_rewrite_tac[]);
a(LIST_DROP_NTH_ASM_T (interval 1 14) discard_tac);
a(conv_tac (ONCE_MAP_C eq_sym_conv) THEN contr_tac);
a(LEMMA_T⌜((a' * m)*(a * n)) Mod p = 1⌝ ante_tac THEN1
        ALL_FC_T once_asm_rewrite_tac[mod_times_homomorphism_thm]);
(* *** Goal "1" *** *)
a(DROP_NTH_ASM_T 5 (strip_asm_tac o rewrite_rule[prime_def]));
a(ALL_FC_T asm_rewrite_tac[less_div_mod_thm]);
(* *** Goal "2" *** *)
a(LEMMA_T⌜((a' * a)*(m * n)) Mod p = 0⌝ ante_tac THEN1
        (ALL_FC_T once_asm_rewrite_tac[mod_times_homomorphism_thm]
                THEN ALL_FC_T asm_rewrite_tac[mod_clauses]));
a(conv_tac(ONCE_MAP_C anf_conv) THEN PC_T1 "lin_arith" prove_tac[]);
val prime_lemma2 = pop_thm ();
```

SML

```
set_goal([], prime_thm);
a(REPEAT strip_tac THEN_LIST [
        POP_ASM_T (strip_asm_tac o rewrite_rule[prime_def])
                THEN asm_rewrite_tac[],
        all_fc_tac[prime_lemma2],
        all_fc_tac[prime_lemma1]]);
val prime_thm = save_pop_thm "prime_thm";
```

SML

```
set_goal([], prime_divisor_thm);
a(∀_tac THEN cov_induction_tac⌜m:ℕ⌝);
a(REPEAT strip_tac);
```

```
a(cases_tac⌜m ∈ Prime⌝ THEN1
        (∃_tac⌜m⌝ THEN ∃_tac⌜1⌝ THEN asm_rewrite_tac[]));
a(POP_ASM_T (strip_asm_tac o rewrite_rule[prime_def]));
a(cases_tac⌜m' = 0⌝ THEN1
        (all_var_elim_asm_tac1 THEN PC_T1 "lin_arith" asm_prove_tac[]));
a(lemma_tac⌜1 < m'⌝ THEN1 PC_T1 "lin_arith" asm_prove_tac[]);
a(cases_tac⌜¬m'< m⌝);
(* *** Goal "1" *** *)
a(cases_tac⌜n = 0⌝ THEN1
        (all_var_elim_asm_tac1 THEN PC_T1 "lin_arith" asm_prove_tac[]));
a(LEMMA_T⌜2 ≤ n⌝ (strip_asm_tac o rewrite_rule[≤_def])
        THEN1 PC_T1 "lin_arith" asm_prove_tac[]);
a(all_var_elim_asm_tac1 THEN  PC_T1 "lin_arith" asm_prove_tac[]);
(* *** Goal "2" *** *)
a(LIST_DROP_NTH_ASM_T [8] all_fc_tac);
a(∃_tac⌜p⌝ THEN ∃_tac⌜n'*n⌝ THEN asm_rewrite_tac[times_assoc_thm]);
val prime_divisor_thm = save_pop_thm "prime_divisor_thm";
```

D Proof 2 — Proofs

SML

```
open_theory "sqrt2_proof2";
set_merge_pcs["'ℤ", "'ℝ", "'sets_alg", "basic_hol1" ];
```

This is the traditional proof intended to impress non-mathematicians.

The main lemma for this proof is that if p is prime and m and n are positive solutions to $m^2 = pn^2$, then there is a solution with smaller n (obtained by dividing m and n by p):

SML

```
set_goal([], proof2_lemma1);
a(REPEAT strip_tac);
a(all_fc_tac[prime_0_less_thm]);
a(lemma_tac⌜(m*m) Mod p = 0⌝ THEN1
        (ALL_FC_T asm_rewrite_tac[mod_clauses]));
a(GET_NTH_ASM_T 5 (strip_asm_tac o rewrite_rule[prime_thm]));
a(LIST_GET_NTH_ASM_T [1] fc_tac);
(* *** Goal "1" which is the same as "2" *** *)
a(all_fc_tac[mod_eq_0_thm]);
a(lemma_tac⌜(n*n) Mod p = 0⌝);
(* *** Goal "1.1" *** *)
a(ALL_FC_T1 fc_⇔_canon rewrite_tac [mod_eq_0_thm]);
a(∃_tac⌜k*k⌝);
a(bc_thm_tac times_cancel_thm);
a(∃_tac⌜p⌝ THEN REPEAT strip_tac);
a(DROP_NTH_ASM_T 9 (rewrite_thm_tac o eq_sym_rule));
a(DROP_NTH_ASM_T 2 rewrite_thm_tac THEN PC_T1 "lin_arith" prove_tac[]);
(* *** Goal "1.2" *** *)
a(LIST_DROP_NTH_ASM_T [5, 7] fc_tac);
(* *** Goal "1.2.1" which is the same as all the 3 other outstanding goals! *** *)
a(POP_ASM_T ante_tac THEN DROP_NTH_ASM_T 3 ante_tac);
a(LIST_DROP_NTH_ASM_T [1, 2, 3, 8] discard_tac);
a(ALL_FC_T1 fc_⇔_canon rewrite_tac [mod_eq_0_thm]);
a(REPEAT strip_tac THEN all_var_elim_asm_tac1);
a(∃_tac⌜k⌝ THEN ∃_tac⌜k'⌝ THEN REPEAT strip_tac);
(* *** Goal "1.2.1.1" *** *)
a(contr_tac THEN lemma_tac⌜k' = 0⌝
        THEN_TRY all_var_elim_asm_tac1
        THEN PC_T1 "lin_arith" asm_prove_tac[]);
(* *** Goal "1.2.1.2" *** *)
a(LEMMA_T⌜2 ≤ p⌝ (strip_asm_tac o rewrite_rule[≤_def])
        THEN1 PC_T1 "lin_arith" asm_prove_tac[]);
a(all_var_elim_asm_tac1 THEN1 PC_T1 "lin_arith" asm_prove_tac[]);
(* *** Goal "1.2.1.3" *** *)
a(bc_thm_tac times_cancel_thm);
```

```
a(∃_tac⌜p⌝ THEN REPEAT strip_tac);
a(bc_thm_tac times_cancel_thm);
a(∃_tac⌜p⌝ THEN REPEAT strip_tac);
a(PC_T1 "lin_arith" asm_prove_tac[]);
val proof2_lemma1_thm = save_pop_thm "proof2_lemma1_thm";
```

SML

```
set_goal([], ⌜∀p n m •
        p ∈ Prime ∧ ℕℝ m ^ 2 = ℕℝ p * (ℕℝ n ^ 2)
⇒       n = 0
⌝);
a(rewrite_tac[ℝ_ℕ_exp_square_thm,
        ℕℝ_times_homomorphism_thm1, ℕℝ_one_one_thm]);
a(∀_tac THEN ∀_tac THEN cov_induction_tac⌜n:ℕ⌝ THEN REPEAT strip_tac);
a(contr_tac THEN lemma_tac ⌜0 < n⌝ THEN1
        PC_T1 "lin_arith" asm_prove_tac[]);
a(all_fc_tac[proof2_lemma1_thm]);
a(all_asm_fc_tac[] THEN all_var_elim_asm_tac1);
val proof2_lemma2_thm = save_pop_thm "proof2_lemma2_thm";
```

...giving us that the square roots of prime numbers are irrational, expressed explicitly:

SML

```
set_goal([], ⌜∀p a b•
        p ∈ Prime ∧ ¬b = 0 ⇒ ¬(a/b)^2 = ℕℝ p
⌝);
a(REPEAT strip_tac);
a(lemma_tac⌜¬ℕℝ b = ℕℝ 0⌝ THEN1
        asm_rewrite_tac[ℕℝ_one_one_thm]);
a(rewrite_tac[ℝ_frac_def] THEN ALL_FC_T rewrite_tac[ℝ_over_times_recip_thm]);
a(contr_tac THEN LEMMA_T⌜
        (ℕℝ a * ℕℝ b ^ -1) ^ 2 * ℕℝ b ^ 2 = ℕℝ p * ℕℝ b ^ 2⌝ ante_tac
        THEN1 asm_rewrite_tac[]);
a(LEMMA_T⌜∀x y z:ℝ•(x*y)^2*z^2 = (x*z*y)^2⌝ rewrite_thm_tac THEN1
        (rewrite_tac[ℝ_ℕ_exp_square_thm]
                THEN PC_T1"ℝ_lin_arith" prove_tac[]));
a(ALL_FC_T rewrite_tac[ℝ_times_recip_thm]);
a(contr_tac THEN all_fc_tac[proof2_lemma2_thm]);
val proof2_thm1 = save_pop_thm"proof2_thm1";
```

A lemma extending the above to the case when $a/b \leq 0$.

SML

```
set_goal([], ⌜
        ∀a b• p ∈ Prime ∧ ¬b = 0 ⇒ ¬(a/b)^2 = ℕℝ p ∧ ¬(∼(a/b))^2 = ℕℝ p
⌝);
a(rewrite_tac[proof2_thm1, square_even_thm]);
val proof2_thm1a = save_pop_thm "proof2_thm1a";
```

The final result in terms of the square root function and the set $\mathbb{Q}$ (which requires us also to deal with the possibility that a/b is negative).

SML

```
set_goal([], proof2_thm2);
a(rewrite_tac[get_spec⌜ℚ⌝] THEN REPEAT_UNTIL is_∨ strip_tac);
a(cases_tac ⌜b = 0⌝ THEN asm_rewrite_tac[]);
a(lemma_tac⌜ℕℝ 0 ≤ ℕℝ p⌝ THEN1
        (all_fc_tac[prime_0_less_thm] THEN
                asm_rewrite_tac[ℕℝ_≤_thm, ≤_def]));
a(contr_tac THEN
        (LEMMA_T ⌜Sqrt(ℕℝ p)^2 = ℕℝ p⌝ ante_tac THEN1
                bc_tac(map (rewrite_rule[]) (fc_canon (get_spec⌜Sqrt⌝))))
                THEN ALL_FC_T asm_rewrite_tac[proof2_thm1a]);
val proof2_thm2_thm = save_pop_thm "proof1_thm2_thm";
```

As Freek Wiedijk rightly insists, we must show that we can specialise this to $\sqrt{2}$, which means we need to prove that 2 is prime. The ProofPower demo suite actually includes an automatic conversion for testing for primality. Rather than reproduce that material here, we just prove that 2 is prime by hand.

SML

```
set_goal([], proof2_lemma3);
a(rewrite_tac[prime_def] THEN contr_tac);
a(cases_tac⌜m = 0⌝ THEN1
        (all_var_elim_asm_tac1 THEN PC_T1 "lin_arith" asm_prove_tac[]));
a(cases_tac⌜n = 0⌝ THEN1
        (all_var_elim_asm_tac1 THEN PC_T1 "lin_arith" asm_prove_tac[]));
a(LEMMA_T⌜2 ≤ m ∧ 2 ≤ n⌝ (strip_asm_tac o rewrite_rule[≤_def])
        THEN1 PC_T1 "lin_arith" asm_prove_tac[]);
a(all_var_elim_asm_tac1 THEN  PC_T1 "lin_arith" asm_prove_tac[]);
val proof2_lemma3_thm = save_pop_thm "proof2_lemma3_thm";
```

SML

```
set_goal([], proof2_thm3);
a(bc_thm_tac proof2_thm2_thm THEN accept_tac proof2_lemma3_thm);
val proof2_thm3_thm = save_pop_thm"proof2_thm3_thm";
```

E Proof 3 — Proofs

SML

```
open_theory "sqrt2_proof3";
set_merge_pcs["'ℤ", "'ℝ", "'sets_alg", "basic_hol1" ];
```

This is the most general proof (about square roots of integers) and the one that generalises further (to the statement that the integers are a integrally closed: i.e., that any rational solution to a monic polynomial with integer coefficients is actually an integer).

The main lemma for this proof is that if k is any positive number and m and n are solutions to $m^2 = kn^2$ with $n > 1$, then there is a solution with smaller n (obtained by dividing m and n by any prime factor of n):

SML

```
set_goal([], proof3_lemma1);
a(REPEAT strip_tac);
a(all_fc_tac[prime_divisor_thm]);
a(all_fc_tac[prime_0_less_thm]);
a(lemma_tac ⌜(m*m) Mod p = 0⌝ THEN1
        (asm_rewrite_tac[pc_rule1"lin_arith" prove_rule[]
                ⌜∀x y z•x*(p*y)*z = p*x*y*z⌝] THEN
                        ALL_FC_T rewrite_tac[mod_clauses]));
a(fc_tac[prime_thm]);
a(LIST_DROP_NTH_ASM_T [2] fc_tac);
(* *** Goal "1" same as "2" *** *)
a(DROP_NTH_ASM_T 3 discard_tac THEN all_fc_tac [mod_eq_0_thm]);
a(∃_tac⌜k'⌝ THEN ∃_tac⌜n'⌝ THEN all_var_elim_asm_tac1
        THEN REPEAT strip_tac);
(* *** Goal "1.1" *** *)
a(contr_tac THEN1
        (lemma_tac⌜n' = 0⌝ THEN_TRY all_var_elim_asm_tac1 THEN PC_T1 "lin_arith" asm_prove
(* *** Goal "1.2" *** *)
a(LEMMA_T ⌜2 ≤ p⌝ (strip_asm_tac o rewrite_rule[≤_def])
        THEN1 PC_T1 "lin_arith" asm_prove_tac[]);
a(cases_tac⌜n' = 0⌝ THEN1
        (all_var_elim_asm_tac1 THEN PC_T1 "lin_arith" asm_prove_tac[]));
a(all_var_elim_asm_tac1 THEN PC_T1 "lin_arith" asm_prove_tac[]);
(* *** Goal "1.3" *** *)
a(bc_thm_tac times_cancel_thm);
a(∃_tac⌜p⌝ THEN REPEAT strip_tac);
a(bc_thm_tac times_cancel_thm);
a(∃_tac⌜p⌝ THEN REPEAT strip_tac);
a(PC_T1 "lin_arith" asm_prove_tac[]);
val proof3_lemma1_thm = save_pop_thm "proof3_lemma1_thm";
```

```
SML
set_goal([], proof3_lemma2);
a(rewrite_tac[ℝ_ℕ_exp_square_thm,
        ℕℝ_times_homomorphism_thm1, ℕℝ_one_one_thm, ℕℝ_less_thm]);
a(REPEAT strip_tac);
a(PC_T1 "predicates" lemma_tac
        ⌜∃y•y ∈ { y | 0 < y ∧ ∃x• x * x = k * y * y }⌝
                THEN1 (∃_tac⌜n⌝ THEN REPEAT strip_tac
                        THEN ∃_tac⌜m⌝ THEN REPEAT strip_tac));
a(all_fc_tac[min_∈_thm]);
a(cases_tac⌜1 < Min { y | 0 < y ∧ ∃x• x * x = k * y * y }⌝
        THEN1 all_fc_tac[proof3_lemma1_thm]);
(* *** Goal "1" *** *)
a(PC_T1 "predicates" lemma_tac
        ⌜n1 ∈ { y | 0 < y ∧ ∃x• x * x = k * y * y }⌝
                THEN1 (REPEAT strip_tac THEN ∃_tac⌜m1⌝ THEN REPEAT strip_tac));
a(all_fc_tac[min_≤_thm] THEN PC_T1 "lin_arith" asm_prove_tac[]);
(* *** Goal "2" *** *)
a(lemma_tac⌜Min { y | 0 < y ∧ ∃x• x * x = k * y * y } = 1⌝
        THEN1 PC_T1 "lin_arith" asm_prove_tac[]);
a(∃_tac⌜x⌝ THEN asm_rewrite_tac[]);
val proof3_lemma2_thm = save_pop_thm "proof3_lemma2_thm";
```

In this proof, to show that $\sqrt{2}$ is irrational, we must show that it is not an integer:

```
SML
set_goal([], proof3_lemma3);
a(rewrite_tac[ℤ_def] THEN contr_tac);
(* *** Goal "1" *** *)
a(ante_tac (rewrite_rule[sqrt_egs_thm]
        (list_∀_elim[⌜ℕℝ 1⌝, ⌜ℕℝ 2⌝] sqrt_less_thm)));
a(asm_rewrite_tac [ℕℝ_less_thm]);
a(ante_tac (rewrite_rule[sqrt_egs_thm]
        (list_∀_elim[⌜ℕℝ 2⌝, ⌜ℕℝ 4⌝] sqrt_less_thm)));
a(asm_rewrite_tac [ℕℝ_less_thm]);
a(PC_T1 "lin_arith" prove_tac[]);
(* *** Goal "2" *** *)
a(asm_tac (rewrite_rule[sqrt_egs_thm]
        (list_∀_elim[⌜ℕℝ 1⌝, ⌜ℕℝ 2⌝] sqrt_less_thm)));
a(lemma_tac ⌜Sqrt (ℕℝ 2) ≤ ℕℝ 0⌝ THEN_LIST
        [asm_rewrite_tac[], PC_T1 "ℝ_lin_arith" asm_prove_tac[]]);
val proof3_lemma3_thm = save_pop_thm"proof3_lemma3_thm";
```

We now have the general result that rational square roots of prime numbers are integers, expressed explicitly:

SML

```
set_goal([], ⌜∀k a b•
        ¬b = 0 ∧ (a/b)^2 = ℕℝ k
⇒       ∃i• ℕℝ i ^ 2 = ℕℝ k
⌝);
a(REPEAT strip_tac);
a(cases_tac⌜k = 0⌝ THEN1
        (∃_tac⌜0⌝ THEN asm_rewrite_tac[ℝ_ℕ_exp_square_thm]));
a(lemma_tac⌜¬ℕℝ b = ℕℝ 0⌝ THEN1
        asm_rewrite_tac[ℕℝ_one_one_thm]);
a(swap_nth_asm_concl_tac 3);
a(rewrite_tac[ℝ_frac_def] THEN ALL_FC_T rewrite_tac[ℝ_over_times_recip_thm]);
a(contr_tac THEN LEMMA_T⌜
        (ℕℝ a * ℕℝ b ^-1) ^ 2 * ℕℝ b ^ 2 = ℕℝ k * ℕℝ b ^ 2⌝ ante_tac
        THEN1 asm_rewrite_tac[]);
a(LEMMA_T⌜∀x y z:ℝ•(x*y)^2*z^2 = (x*z*y)^2⌝ rewrite_thm_tac THEN1
        (rewrite_tac[ℝ_ℕ_exp_square_thm]
                THEN PC_T1"ℝ_lin_arith" prove_tac[]));
a(ALL_FC_T rewrite_tac[ℝ_times_recip_thm]);
a(lemma_tac⌜ℕℝ 0 < ℕℝ k ∧ ℕℝ 0 < ℕℝ b⌝ THEN1
        (rewrite_tac[ℕℝ_less_thm] THEN PC_T1 "lin_arith" asm_prove_tac[]));
a(contr_tac THEN all_fc_tac[proof3_lemma2_thm]);
a(all_asm_fc_tac[]);
val proof3_thm1_thm = save_pop_thm"proof3_thm1_thm";
```

The final result in terms of the square root function and the set $\mathbb{Q}$ (which requires us also to deal with the possibility that a/b is negative).

SML

```
set_goal([], proof3_thm2);
a(rewrite_tac[rats_def, ℤ_def] THEN contr_tac
        THEN all_var_elim_asm_tac1);
(* *** Goal "1" *** *)
a(LEMMA_T⌜Sqrt (ℕℝ m)^2 = ℕℝ m⌝ ante_tac THEN1
        ALL_FC_T rewrite_tac[sqrt_thm]);
a(asm_rewrite_tac[] THEN contr_tac THEN
        all_fc_tac[proof3_thm1_thm]);
a(lemma_tac⌜ℕℝ 0 ≤ ℕℝ i⌝ THEN1
        rewrite_tac[ℕℝ_≤_thm]);
a(all_fc_tac[sqrt_eq_thm]);
a(DROP_NTH_ASM_T 5 (ante_tac o ∀_elim⌜i⌝));
a(asm_rewrite_tac[]);
(* *** Goal "2" *** *)
a(LEMMA_T⌜Sqrt (ℕℝ m)^2 = ℕℝ m⌝ ante_tac THEN1
        ALL_FC_T rewrite_tac[sqrt_thm]);
a(LEMMA_T⌜∀x:ℝ•(∼x)^2 = x^2⌝ asm_rewrite_thm_tac THEN1
```

```
        (rewrite_tac[ℝ_ℕ_exp_square_thm] THEN
                PC_T1 "ℝ_lin_arith" prove_tac[]));
a(contr_tac THEN all_fc_tac[proof3_thm1_thm]);
a(lemma_tac⌜ℕℝ 0 ≤ ℕℝ i⌝ THEN1
        rewrite_tac[ℕℝ_≤_thm]);
a(all_fc_tac[sqrt_eq_thm]);
a(DROP_NTH_ASM_T 5 (ante_tac o ∀_elim⌜i⌝));
a(asm_rewrite_tac[]);
(* *** Goal "3" *** *)
a(DROP_NTH_ASM_T 4 ante_tac);
a(pure_once_rewrite_tac[ℝ_≤_≤_0_thm]);
a(rewrite_tac[ℕℝ_≤_thm]);
a(swap_nth_asm_concl_tac 1 THEN REPEAT strip_tac);
a(∃_tac⌜0⌝ THEN all_var_elim_asm_tac1 THEN asm_rewrite_tac[]);
a(POP_ASM_T (rewrite_thm_tac o eq_sym_rule));
a(rewrite_tac[sqrt_egs_thm]);
(* *** Goal "4" *** *)
a(DROP_NTH_ASM_T 4 ante_tac);
a(pure_once_rewrite_tac[ℝ_≤_≤_0_thm]);
a(rewrite_tac[ℕℝ_≤_thm]);
a(swap_nth_asm_concl_tac 1 THEN REPEAT strip_tac);
a(∃_tac⌜0⌝ THEN all_var_elim_asm_tac1 THEN asm_rewrite_tac[]);
a(POP_ASM_T (rewrite_thm_tac o eq_sym_rule));
a(rewrite_tac[sqrt_egs_thm]);
val proof3_thm2 = save_pop_thm"proof3_thm2";
```

And, using the fact that $\sqrt{2}$ is not an integer, we get the specific conclusion of the third proof:

SML

```
set_goal([], proof3_thm3);
a(contr_tac);
a(LEMMA_T ⌜ℕℝ 0 ≤ ℕℝ 2⌝ asm_tac THEN1 rewrite_tac[]);
a(lemma_tac ⌜ℕℝ 2 ∈ ℤ⌝ THEN1
        (rewrite_tac[ℤ_def] THEN ∃_tac⌜2⌝ THEN REPEAT strip_tac));
a(all_fc_tac[proof3_thm2]);
a(all_fc_tac[proof3_lemma3_thm]);
val proof3_thm3 = save_pop_thm"proof3_thm3";
```

Contents

SML

```
open_theory"sqrt2_defs";
output_theory{out_file="57.th0.doc", theory="sqrt2_defs"};
open_theory"sqrt2_proof1";
output_theory{out_file="57.th1.doc", theory="sqrt2_proof1"};
open_theory"divisibility";
output_theory{out_file="57.th2.doc", theory="divisibility"};
open_theory"sqrt2_proof2";
output_theory{out_file="57.th3.doc", theory="sqrt2_proof2"};
open_theory"sqrt2_proof3";
output_theory{out_file="57.th4.doc", theory="sqrt2_proof3"};
```