Nuprl

Paul Jackson

Formalization and answers by Paul Jackson <pbj@dcs.ed.ac.uk>.

14.1 Statement

$\neg(\exists u{:}\mathbb{Q}.\ u *_q u = 2 / 1)$

14.2 Definitions

Definition of $\mathbb{Q}$

*D rat_df	Rat== rat
*A rat	Rat == $\mathbb{Z} \times \mathbb{Z}^{-0}$
*T rat_wf	Rat $\in \mathbb{U}_1$
*D qnumer_df	<q:Q:*>.num== qnumer(<q>)
*A qnumer	q.num == q.1
*T qnumer_wf	$\forall$q:Rat. q.num $\in \mathbb{Z}$
*D qdenom_df	<q:Q:*>.den== qdenom(<q>)
*A qdenom	q.den == q.2
*T qdenom_wf	$\forall$q:Rat. q.den $\in \mathbb{Z}^{-0}$
*D qequiv_df	<x:x:*> $\equiv$q <y:y:*>== qequiv(<x>; <y>)
*A qequiv	x $\equiv$q y == x.num * y.den = y.num * x.den
*T qequiv_wf	$\forall$x,y:Rat. x $\equiv$q y $\in \mathbb{U}_1$
*D qrat_df	$\mathbb{Q}$== qrat
*A qrat	$\mathbb{Q}$ == x,y:Rat//x $\equiv$q y
*T qrat_wf	$\mathbb{Q} \in \mathbb{U}_1$

14.3 Proof

%[

Theory Descriptions:

- rat_1
  Definition of rationals as pair of ints. Slight alteration of rat_1 theory from standard distribution.
- quot_2
  Extension of quot_1. Further support for use of quotient type, where all injections into type are explicitly tagged.
- rat_2
  Definition of rationals using quotient type, building on rat_1.

F. Wiedijk (Ed.): The Seventeen Provers of the World, LNAI 3600, pp. 116–126, 2006.

– num_thy_2
  Additions to num_thy_1.
– root_2_irrat
  The main proofs, over the integers and over the rationals.

```
]%

set_theory_filenames wdir ["rat_1"] ;;
set_theory_filenames wdir ["quot_2"] ;;
set_theory_filenames wdir ["rat_2"] ;;
set_theory_filenames wdir ["num_thy_2"] ;;
set_theory_filenames wdir ["root_2_irrat"] ;;

set_theory_ancestors "rat_1" ["STD"];;
set_theory_ancestors "quot_2" ["STD"];;
set_theory_ancestors "num_thy_2" ["STD"];;
set_theory_ancestors "rat_2" ["rat_1";"quot_2"];;
set_theory_ancestors "root_2_irrat" ["rat_2";"num_thy_2"];;
```

File root_2_irrat.prl*:*

```
*C root_2_irrat_begin          ************ ROOT_2_IRRAT ************
*T qrat_irreduc_witness        ∀u:ℚ. ↓(∃n:ℤ. ∃d:ℤ⁻⁰. u = n / d ∧ CoPrime(n,d))
*T exists_zero_one_elim        ∀P:ℕ2 ⟶ ℙ. (∃i:ℕ2. P[i]) ⟺ P[0] ∨ P[1]
*T two_div_square              ∀n:ℤ. 2 | n * n ⇒ 2 | n
*T root_2_irrat_over_int       ¬(∃m,n:ℤ. CoPrime(m,n) ∧ m * m = 2 * n * n)
*T root_2_irrat                ¬(∃u:ℚ. u *q u = 2 / 1)
*C root_2_irrat_end            ************************************
```

Proof of qrat_irreduc_witness*:*

```
⊢ ∀u:ℚ. ↓(∃n:ℤ. ∃d:ℤ⁻⁰. u = n / d ∈ ℚ ∧ CoPrime(n,d))
|
BY (D 0 THENM InstLemma ‘qrat_witness‘ [⌈u⌉]
|   THENM D 2 THENM D 0 THENM ExRepD ...a)
|
1. u: ℚ
2. n: ℤ
3. d: ℤ⁻⁰
4. u = n / d ∈ ℚ
⊢ ∃n:ℤ. ∃d:ℤ⁻⁰. u = n / d ∈ ℚ ∧ CoPrime(n,d)
|
BY % Facts to help type-checking %
|
|  (Assert ⌈gcd(n;d) ≠ 0⌉ THENA
|    (D 0 THENM Sel (-1) (FLemma ‘gcd_zero_result‘ [5]) ...))
|  THEN
|  (Assert ⌈d ÷ gcd(n;d) ≠ 0⌉ THENA
|    (Backchain ‘‘div_zero_a gcd_is_divisor_2‘‘ ...))
|
5. gcd(n;d) ≠ 0
6. d ÷ gcd(n;d) ≠ 0
|
BY (Inst [⌈n ÷ gcd(n;d)⌉;⌈d ÷ gcd(n;d)⌉] 0 ...a)
|\
```

```
| ⊢ u = (n ÷ gcd(n;d)) / (d ÷ gcd(n;d)) ∈ ℚ
| |
| BY (RWH (RevLemmaWithC [‘n’,⌈gcd(n;d)⌉] ‘numq_factor_cancel‘) 0 ...a)
| |
| ⊢ u = (gcd(n;d) * (n ÷ gcd(n;d))) / (gcd(n;d) * (d ÷ gcd(n;d))) ∈ ℚ
| |
| BY RWH (LemmaC ‘mul_com‘) 0
| |  THENM RWH (GenLemmaC 1 ‘divides_iff_div_exact‘) 0
| |  THENA Repeat
| |    (Progress Auto
| |     ORELSE (RemoveLabel THEN EqTypeCD)
| |     ORELSE Backchain ‘‘gcd_is_divisor_1 gcd_is_divisor_2 int_entire_a‘‘)
| |
| ⊢ u = n / d ∈ ℚ
| |
| BY Auto
 \
  ⊢ CoPrime(n ÷ gcd(n;d),d ÷ gcd(n;d))
  |
  BY Unfold ‘coprime‘ 0
  |
  ⊢ GCD(n ÷ gcd(n;d);d ÷ gcd(n;d);1)
  |
  BY (Using [‘n’,⌈gcd(n;d)⌉] (BLemma ‘gcd_p_mul_elim‘)
  |   THENM RWH IntSimpC 0  ...a)
  |
  ⊢ GCD((n ÷ gcd(n;d)) * gcd(n;d);(d ÷ gcd(n;d)) * gcd(n;d);gcd(n;d))
  |
  BY (RWH (GenLemmaC 1 ‘divides_iff_div_exact‘) 0
      THEN Backchain ‘‘gcd_is_divisor_1 gcd_is_divisor_2 gcd_sat_gcd_p‘‘ ...)
```

Proof of exists_zero_one_elim*:*

```
⊢ ∀P:ℕ2 ⟶ ℙ. (∃i:ℕ2. P[i]) ⟺ P[0] ∨ P[1]
|
BY GenExRepD THENA Auto
|\
| 1. P: ℕ2 ⟶ ℙ
| 2. i: ℕ2
| 3. P[i]
| ⊢ P[0] ∨ P[1]
| |
| BY (Assert ⌈i = 0 ∨ i = 1⌉ ...a)
| |\
| | ⊢ i = 0 ∨ i = 1
| | |
| | BY SIAuto
|  \
|   4. i = 0 ∨ i = 1
|   |
|   BY D 4
|   |\
|   | 4. i = 0
|   | |
|   | BY HypSubst 4 3 THENA Auto
|   | |
|   | 3. P[0]
```

```
|   | |
|   | BY ProveProp THENA Auto
|    \
|     4. i = 1
|     |
|     BY HypSubst 4 3 THENA Auto
|     |
|     3. P[1]
|     |
|     BY (ProveProp ...)
|\
| 1. P: ℕ2 ⟶ ℙ
| 2. P[0]
| ⊢ ∃i:ℕ2. P[i]
| |
| BY (Inst [⌜0⌝] 0 ...)
 \
  1. P: ℕ2 ⟶ ℙ
  2. P[1]
  ⊢ ∃i:ℕ2. P[i]
  |
  BY (Inst [⌜1⌝] 0 ...)
```

Proof of two_div_square*:*

```
⊢ ∀n:ℤ. 2 | n * n ⇒ 2 | n
|
BY (D 0 ...a)
|
1. n: ℤ
⊢ 2 | n * n ⇒ 2 | n
|
BY Assert ⌜n * n = 0 mod 2 ⇒ n = 0 mod 2⌝
|\
| ⊢ n * n = 0 mod 2 ⇒ n = 0 mod 2
| |
| BY (D 0 ...a)
| |
| 2. n * n = 0 mod 2
| ⊢ n = 0 mod 2
| |
| BY (InstLemma ‘eqmod_exists‘ [⌜2⌝;⌜n⌝] ...a)
| |
| 3. ∃b:ℕ2. n = b mod 2
| |
| BY (RWH (LemmaC ‘exists_zero_one_elim‘) 3 ...a)
| |
| 3. n = 0 mod 2 ∨ n = 1 mod 2
| |
| BY D 3 THEN Auto
| |
| 3. n = 1 mod 2
| |
| BY (FLemma ‘multiply_functionality_wrt_eqmod‘ [3;3]
| |   THENM ArithSimp 4 ...a)
| |
| 4. n * n = 1 mod 2
```

```
| |
| BY (FLemma ‘eqmod_unique‘ [2;4] ...a)
| |
| 5. 0 = 1
| |
| BY Auto
 \
  2. n * n = 0 mod 2 ⇒ n = 0 mod 2
  |
  BY (Unfold ‘eqmod‘ 2 THEN ArithSimp 2 ...)
```

Proof of root_2_irrat_over_int*:*

```
⊢ ¬(∃m:ℤ. ∃n:ℤ. CoPrime(m,n) ∧ m * m = 2 * n * n)
|
BY (D 0 THENM ExRepD ...a)
|
1. m: ℤ
2. n: ℤ
3. CoPrime(m,n)
4. m * m = 2 * n * n
⊢ False
|
BY Assert ⌜2 | m⌝
|\
| ⊢ 2 | m
| |
| BY (BLemma ‘two_div_square‘ THENM Unfold ‘divides‘ 0
|     THENM AutoInstConcl [] ...a)
 \
  5. 2 | m
  |
  BY Assert ⌜2 | n⌝
  |\
  | ⊢ 2 | n
  | |
  | BY (BLemma ‘two_div_square‘ THENM All (Unfold ‘divides‘)
  |     THENM ExRepD THENM Inst [⌜c * c⌝] 0
  |     THENM RWO "6" 4 ...)
   \
    6. 2 | n
    |
    BY (RWO "coprime_elim" 3 THENM FHyp 3 [5;6] ...a)
    |
    3. ∀c:ℤ. c | m ⇒ c | n ⇒ c ∼ 1
    7. 2 ∼ 1
    |
    BY (RWO "assoced_elim" 7 THENM D (-1) ...)
```

Proof of root_2_irrat*:*

```
⊢ ¬(∃u:ℚ. u *q u = 2 / 1 ∈ ℚ)
|
BY (D 0 THENM D (-1) ...a)
|
1. u: ℚ
2. u *q u = 2 / 1 ∈ ℚ
```

```
⊢ False
|
BY (InstLemma ‘qrat_irreduc_witness‘ [⌈u⌉]
|   THENM D (-1) THENM ExRepD
|   THENM RWW "5" 2 THENM On [4;1] Thin ...a)
|
1. n: ℤ
2. d: ℤ−0
3. CoPrime(n,d)
4. (n / d) *q (n / d) = 2 / 1 ∈ ℚ
|
BY (Unfolds ‘‘mulq numq‘‘ 4 THENM AbReduce 4
|   THENM RWO "eq_qrat_elim" 4
|   THENM Unfolds ‘‘qmul qequiv‘‘ 4 THENM AbReduce 4
|   THENM RWH IntSimpC 4 ...a)
|
4. n * n = 2 * d * d
|
BY AssertLemma ‘root_2_irrat_over_int‘ [] THENM D (-1)
|
⊢ ∃m:ℤ. ∃n:ℤ. CoPrime(m,n) ∧ m * m = 2 * n * n
|
BY (AutoInstConcl [] ...)
```

Script of root_2_irrat*:*

```
TACTIC  top :
  (D 0 THENM D (-1) ...a)
SUBTREES
  TACTIC  top 1 :
    (InstLemma ‘qrat_irreduc_witness‘ [⌈u⌉]
     THENM D (-1) THENM ExRepD
     THENM RWW "5" 2 THENM On [4;1] Thin ...a)
  SUBTREES
    TACTIC  top 1 1 :
      (Unfolds ‘‘mulq numq‘‘ 4 THENM AbReduce 4
       THENM RWO "eq_qrat_elim" 4
       THENM Unfolds ‘‘qmul qequiv‘‘ 4 THENM AbReduce 4
       THENM RWH IntSimpC 4 ...a)
    SUBTREES
      TACTIC  top 1 1 1 :
        AssertLemma ‘root_2_irrat_over_int‘ [] THENM D (-1)
      SUBTREES
        TACTIC  top 1 1 1 1 :
          (AutoInstConcl [] ...)
        SUBTREES
          "<no subterms>"
        END
      END
    END
  END
END
```

S-expression of root_2_irrat *in file* root_2_irrat.thy*:*

```
(|root_2_irrat| THM NIL ((|not|) (NIL (|exists|) (NIL (|qrat|)) (("u") (
|equal|) (NIL (|qrat|)) (NIL (|mulq|) (NIL . "u") (NIL . "u")) (NIL (|nu
```

```
mq|) (NIL (|natural_number| ("2" . |natural|))) (NIL (|natural_number| (
"1" . |natural|))))))) (COMPLETE ((((!ml_text_cons|) (NIL (|!text| ("" .
 |string|))) (NIL (|!ml_text_cons|) (NIL (|aux_auto|) (NIL (|!text| ("D
0 THENM D (-1)" . |string|)))) (NIL (|!text| ("" . |string|))))) ((((!ml
_text_cons|) (NIL (|!text| ("" . |string|))) (NIL (|!ml_text_cons|) (NIL
 (|aux_auto|) (NIL (|!text_cons|) (NIL (|!text| ("InstLemma `qrat_irredu
c_witness` [" . |string|))) (NIL (|!text_cons|) (NIL (|!prl_term|) (NIL
. "u")) (NIL (|!text_cons|) (NIL (|!text| ("]" . |string|))) (NIL (|!tex
t_cons|) (NIL (|!newline|)) (NIL (|!text_cons|) (NIL (|!text| ("THENM D
(-1) THENM ExRepD" . |string|))) (NIL (|!text_cons|) (NIL (|!newline|))
(NIL (|!text| ("THENM RWW \"5\" 2 THENM On [4;1] Thin" . |string|)))))))
))) (NIL (|!text| ("" . |string|))))) ((((!ml_text_cons|) (NIL (|!text|
("" . |string|))) (NIL (|!ml_text_cons|) (NIL (|aux_auto|) (NIL (|!text_
cons|) (NIL (|!text| ("Unfolds ``mulq numq`` 4 THENM AbReduce 4 " . |str
ing|))) (NIL (|!text_cons|) (NIL (|!newline|)) (NIL (|!text_cons|) (NIL
(|!text| ("THENM RWO \"eq_qrat_elim\" 4 " . |string|))) (NIL (|!text_con
s|) (NIL (|!newline|)) (NIL (|!text_cons|) (NIL (|!text| ("THENM Unfolds
 ``qmul qequiv`` 4 THENM AbReduce 4" . |string|))) (NIL (|!text_cons|) (
NIL (|!newline|)) (NIL (|!text| ("THENM RWH IntSimpC 4" . |string|))))))
)))) (NIL (|!text| ("" . |string|))))) ((((!text| ("AssertLemma `root_2_
irrat_over_int` [] THENM D (-1)" . |string|))) (((|!ml_text_cons|) (NIL
(|!text| ("" . |string|))) (NIL (|!ml_text_cons|) (NIL (|auto|) (NIL (|!
text| ("AutoInstConcl []" . |string|)))) (NIL (|!text| ("" . |string|)))
)))))) ((|lambda|) (("%") (|spread|) (NIL . "%") (("u" "%1") (|apply|)
(NIL (|lambda|) (("%2") (|apply|) (NIL (|lambda|) (("%2") (|axiom|))) (N
IL (|apply|) (NIL . "%2") (NIL . "u")))) (NIL (|qrat_irreduc_witness| (N
UPRL-PARAMETER-VARS::|\\v| . |level-expression|)))))) (REFS (|coprime_wf
| |auto| |root_2_irrat_over_int| |qnum_wf| |qmul_wf| |qnum| |pi2| |pi1|
|qdenom| |qnumer| |qequiv| |qmul| |eq_qrat_elim| |quot_inj| |qrat_inj| |
quot_let| |qrat_proj| |mulq| |numq| |true| |comb_for_mulq_wf| |rev_impli
es| |iff| |and| |squash| |qrat_irreduc_witness| |iprl_term| |inewline| |
itext_cons| |nequal_wf| |false| |nequal| |int_nzero| |subtype| |numq_wf|
 |mulq_wf| |all| |qrat_wf| |prop| |member| |exists| |implies| |not| |aux
_auto| |iml_text_cons|) (|root_2_irrat_over_int| |eq_qrat_elim| |comb_fo
r_mulq_wf| |qrat_irreduc_witness|))) NIL)
```

14.4 System

What is the home page of the system?

<http://www.cs.cornell.edu/Info/Projects/NuPrl/nuprl.html>

What are the books about the system? The Nuprl book:

> Robert L. Constable, et al., *Implementing Mathematics with the Nuprl Proof Development System*, Prentice Hall, Engelwood Cliffs, New Jersey, April 1986, x+299 pp, ISBN 0134518322.

is the standard reference. However the system has changed sufficiently that none of the tutorials given in the book will work in the current system. The manual of version 4.2:

> Paul B. Jackson, *The Nuprl Proof Development System, Version 4.2. Reference Manual and User's Guide*, July 1995.

and the manual of version 5:

Nuprl 5 Manual (Draft).

both are available from the Nuprl home page.

The Nuprl home page also provides access to the Nuprl *Math Library* which showcases a selection of recent developments in Nuprl.

What is the logic of the system? Nuprl uses a constructive type theory based on that of Martin-Löf as described in his paper *Constructive Mathematics and Computer Programming*, in proceedings of Sixth International Congress for Logic, Methodology, and Philosophy of Science, North Holland, 1982, pp. 153–175. Some particular features of the type theory are commented on below in the section on distinguishing features of Nuprl.

Witnesses of inhabitation of a type do not contain sufficient information to guide proofs of inhabitation, as they do in systems such as Coq and Lego. This is partly because the presence of subtypes in the type theory. These witnesses therefore do not function as 'proof objects'.

What is the implementation architecture of the system? Nuprl has an LCF architecture based on an abstract data type of proofs. This proof datatype maintains the structure of both incomplete and complete proofs, down to the level of primitive inferences if desired. The implementation of the datatype is not as small as in HOL or Isabelle. For example, it has primitive rules for arithmetic simplification and deciding linear arithmetic problems.

Nuprl has an extensive tactic collection covering operations such as typechecking, rewriting, forward and backward chaining, arithmetic reasoning and inequality reasoning. Tactics are written in Edinburgh ML.

Nuprl 4.2 is implemented in Common Lisp and the 1979 Edinburgh ML.

What does working with the system look like? Proofs are developed interactively by running tactics in a proof tree editor. With this editor, one focuses on one node of the proof tree at a time. The editor presents the user with the full sequent for the node, together with, if any, the tactic run on that node and the children proof nodes generated by the tactic. The Nuprl Math Library presents proofs using views similar to those generated by this proof tree editor.

All Nuprl expressions are entered using a structure editor and presented on the screen and in print using syntax chosen for brevity and clarity. There are no requirements that display syntax be parsable. The structure editor is also used for the ML tactics: for example, the ... notation is used in display forms for invocations of Nuprl's typechecking and basic-reasoning tactic.

With Nuprl 4.2 one typically has

- 1 scrollable library window, listing loaded theorems, definitions, display forms, comments, and ML objects.
- 1 ML top loop window

- For each proof (many proofs can be open simultaneously)
  1 proof tree editor window as described above.
  The proof refinement window is read only.
- To enter main goals of theorems, or enter tactics when developing proofs, one pops up temporary 'term editor' windows which permit structure editing.
  Term editor windows are also used for displaying and editing definitions and display forms, and for the ML top loop (one needs to be able to enter Nuprl term arguments to tactics).

Proofs are saved to files in the form of proof scripts, recording the tactics used to generate the proofs.

Definitions, theorems, ML code and comments are grouped together in *theories*. Each theory is saved in a single `.thy` file using Lisp S-expression syntax. This syntax is verbose and low-level, and users rarely view or edit these files directly with a text editor.

What is special about the system compared to other systems?

- The type theory includes less-common *subtype* and *quotient type* type constructors, and, since 1995, *very-dependent* function types, where the type of the value of the function on some argument can depend on its value on other arguments smaller in some well-founded order.
- Having subtypes defined by arbitrary predicates makes type checking undecidable, and all type checking is done by proof. This contrasts with PVS where a type-checking algorithm automates all simpler type-checking tasks. Caching of type-checking obligations improves the efficiency of type-checking by proof.
- The computation language is untyped, enabling the expression of polymorphic functions without dummy type arguments and the definition of general recursive functions with the Y combinator.
- Structure editors are used for all interaction with the system. This permits the use of rich syntaxes for terms, proofs and theories that don't have to be mechanically parsable. Any ambiguities in syntax are easily resolved by interaction with the editors.
- Proofs are edited and viewed as proof trees, rather than, for example, goal stacks.

What are other versions of the system? The proofs described here have been carried out in Nuprl 4.2 which dates from about 1995. There are more recent versions of Nuprl (Nuprl 5, MetaPrl) with significant developments in the user interface, type theory, and performance, but the level of automation in these versions is mostly not much different from that shown above. Further details can be found from the Nuprl home page.

The Nuprl Math Library pages accessible from the home page illustrate ongoing work in producing readable presentations of formal mathematics.

Who are the people behind the system? The principal people involved in the design and implementation of Nuprl 4 were Bob Constable, Stuart Allen, Richard Eaton, Jason Hickey, Doug Howe and Paul Jackson. Many others were involved in previous versions.

What are the main user communities of the system? The main users of Nuprl are virtually all existing or former members of the Nuprl group at Cornell.

What large mathematical formalizations have been done in the system? Specific developments, roughly ordered from more purely mathematical to more applied include:

- Constructive real analysis: the intermediate value theorem (Howe, Chirimar 1991, Forester 1993)
- Computational abstract algebra: multivariate polynomial arithmetic, unique factorisation domains (Jackson 1995)
- Girard's paradox (Howe 1987)
- Metatheory of propositional logics (Caldwell, Gent, Underwood 1990-2001)
- Computational metatheory (Howe 1988)
- Extracting constructive content from classical proofs: Higman's Lemma (Murthy 1991)
- Automata theory, Turing machines (Bickford, Constable, Kreitz, Nogin, Naumov, Jackson, Uribe 1986-2001)

See the Nuprl home pages (both publications and Math Library pages) for full citations and further details.

What representation of the formalization has been put in this paper? Proofs are shown above in one of the formats commonly used for full proofs. For brevity, each sequent in the tree is shown only with those hypotheses and conclusion formulae and types that have changed from ancestor sequents in the tree.

Proofs are saved to files in the form of proof scripts. As an example, the proof section shows the usual display of the proof script for `root_2_irrat`. This is how scripts are displayed interactively when one wants to edit directly an existing script. The '`top 1 2 3`' addresses are automatically filled in. These aren't saved when scripts are saved.

What needs to be explained about this specific proof? The proof follows the traditional approach that is ascribed to Pythagoras and that Lamport chose for his 1993 DEC SRC report *How to Write a Proof*.

This Nuprl V4.2 development relies on slight modifications and extensions of standard V4.2 theories concerning number theory and the rationals. In particular, whereas the standard theory stops with a definition of the rationals as pairs of integers, here we take here the extra step of using Nuprl's quotient type to construct a more abstract and elegant rational type:

```
rat:
  Rat ==  ℤ × ℤ⁻ⁿ⁰
qequiv:
  x ≡q y ==  x.num * y.den = y.num * x.den
qrat:
  ℚ ==  x,y:Rat//x ≡q y
```

For constructivity reasons, elements of quotient types in Nuprl are single representatives of equivalence classes rather than the equivalence classes themselves. When functions are typed with $\mathbb{Q}$ rather than `Rat`, the type theory requires the functions to respect the standard equality on rationals. For example, the typing lemma for the function $\mathbb{Q}$ is:

```
mulq_wf:
  ∀u:ℚ. ∀v:ℚ.  u *q v ∈ ℚ
```

Any proof of this typing lemma must not only show that the function returns an element of $\mathbb{Q}$ for arguments in $\mathbb{Q}$, but also that it returns equivalent rationals for equivalent arguments.

The proofs presented above are revisions of the proofs as originally entered: steps have been combined to reduce the lengths of the proofs and emphasize the main steps of the proofs. For comparison, the `two_div_square` proof has not been revised.

The proof of `qrat_irreduc_witness` is complicated by the fact that Nuprl V4.2's tactics don't handle subtypes as well as, for example, PVS. Integer division (the $\div$ operation) is only well-formed for non-zero second arguments, and the tactics need explicit help from the user to prove this.

The $\downarrow$ operator in

```
qrat_irreduc_witness:
  ∀u:ℚ. ↓(∃n:ℤ. ∃d:ℤ⁻ⁿ⁰. u = n / d ∈ ℚ ∧ CoPrime(n,d))
```

is called *squash*. The type $\downarrow T$ is a singleton type if T is inhabited, otherwise it is an empty type. The lemma is slightly simpler to prove with the squash operation included than without. Without, the proof must exhibit a function, implicitly or explicitly, that computes the same numerator `n` and denominator `d` for any pair of equal rational numbers `u`. In other words, the proof must involve computing canonical forms of rational numbers and showing these forms are unique.

The full Nuprl theory and ML files for this work are available online at `<http://www.dcs.ed.ac.uk/home/pbj/nuprl>`.

The Nuprl Math Library contains an alternate proof of the irrationality of $\sqrt{2}$ produced by Stuart Allen. This alternate proof does not involve the rationals, but does show the irrationality of square roots of arbitrary primes.