Formalizing Constructive Real Analysis*

Max B. Forester

TR 93-1382
September 1993

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*This work supported by NSF grants CCR-9244739, CCR-9148222 and ONR grants N00014-92-J-1764, N00014-91-J-4123.

Formalizing Constructive Real Analysis*

Max B. Forester
Department of Computer Science
Cornell University

*This work supported by NSF grants CCR-9244739, CCR-9148222 and ONR grants N00014-92-J-1764, N00014-91-J-4123.

Formalizing Constructive Real Analysis

Max B. Forester
Department of Computer Science
Cornell University
forester@cs.cornell.edu

Abstract

This paper arises from a project with the Nuprl Proof Development System which involved formalizing parts of real analysis, up through the intermediate value theorem. Extensive development of the rational library was required as the real library was being built, resulting in the addition of about 125 rational theorems. The real library now contains about 150 theorems and includes enough basic results that further extensions of the library should be quite feasible. This paper aims to illustrate how higher mathematics can be implemented in a system like Nuprl, and also to introduce system users to the library.

1 Introduction

The first part of this paper is a guided tour through the real library with emphasis on the definitions, a handful of useful theorems, and the proof of the intermediate value theorem. Afterwards is a section giving detailed descriptions of the special-purpose tactics that were written in conjunction with the rationals and the reals. Users interested in developing theories in analysis should find both sections helpful.

2 Real Analysis in Nuprl

2.1 Definitions

In formalizing real analysis we follow the development of Bishop [1]. Real numbers are represented by infinite sequences of rationals, which are required to converge at a certain rate. We have

```
* ABS real
ℝ == {x:Seq(ℚ)| (∀i:ℕ⁺. ∀j:ℕ⁺. |x(i) - x(j)| ≤ 1/i + 1/j)}  ;;
```

and equality is given by

```
* ABS eq_real
r = s == ∀n:ℕ⁺. |r(n) - s(n)| ≤ 2/n ;;
```

Some operations on reals are defined as follows:

```
* ABS radd
a + b == λi.(a((2 * i))) + (b((2 * i))) ;;

* ABS rneg
-a == λi.-(a(i)) ;;

* ABS rsub
a - b == a + -b ;;
```

We identify the rationals with the subset of the reals represented by constant sequences.

```
* ABS real_rat
a == λi.a ;;
```

The definition of multiplication involves the *canonical bound* of a real number `a`, which is an integer `Ka` such that for all `n`, `Ka > |a(n)|`. We define `Ka` as the least integer greater than `|a(1)| + 2`.

```
* ABS canon_bound
Ka == |a(1).num ÷ a(1).den| + 3 ;;

* ABS rmul_gen
a *g b == λi.((a((2 * (Ka * i)))) * (b((2 * (Ka * i))))) ;;

* ABS rmul
a * b == if |a(1)| < |b(1)| then b *g a else a *g b  ;;
```

The purpose of the case split is to ensure that the maximum of the canonical bounds of `a` and `b` is used.

```
* ABS rmax
max{a,b} == λi.max{(a(i)),(b(i))} ;;

* ABS rabsval
|a| == max{a,-a} ;;

* ABS pos
a > 0 == ∃n:ℕ+. 1/n < a(n) ;;

* ABS lt_r
a < b == b - a > 0 ;;

* ABS gt_r
a > b == b < a ;;
```

```
* ABS non_neg
a ≥ 0 == (∀n:ℕ⁺. -1/n ≤ a(n))  ;;

* ABS le_r
a ≤ b == b - a ≥ 0 ;;

* ABS ge_r
a ≥ b == b ≤ a ;;
```

We also define the following subsets of $\mathbb{R}$. Notice that a positive real is a pair, consisting of a real and a positive integer. This integer is the witness that the real is positive. Nonzero reals are defined similarly.

```
* ABS real_pos
ℝ⁺ == r:ℝ × r > 0 ;;

* ABS real_non_neg
ℝ°⁺ == {r:ℝ| r ≥ 0}  ;;

* ABS real_nzero
ℝ⁻° == r:ℝ × {n:ℕ⁺| 1/n < |r(n)|}  ;;
```

It is often convenient to think of positive reals as real numbers rather than as pairs, and in Bishop this simplification is made with the understanding that witnesses are always available when needed. Nuprl's display mechanism easily adopts this style of presentation by having a positive number and its first projection look the same. This goes for nonzero reals as well.

```
* ABS rnum
a == a.1 ;;

* ABS rwit
a.wit == a.2 ;;
```

The following two operations are defined on $\mathbb{R}^{-\circ}$ and $\mathbb{R} \times \mathbb{R}^{-\circ}$.

```
* ABS rinv
1/a == λi.if i <z a.wit then 1/a(a.wit^3) else 1/a((i * a.wit^2))  ;;

* ABS rdiv
a/b == a * 1/b ;;
```

And finally,

```
* ABS interval
[a,b] == {r:ℝ| (a ≤ r ∧ r ≤ b)}  ;;

* ABS continuous
f cont on [a,b] == ∃w:ℝ → ℝ.  ∀ε:ℝ⁺. w(ε) > 0 ∧
                         (∀x:[a,b]. ∀y:[a,b]. |x - y| ≤ w(ε) ⇒
                                                  |(f(x)) - (f(y))| ≤ ε) ;;
```

2.2 Well-Formedness and Functionality

Since $\mathbb{R}$ is a proper subset of Seq($\mathbb{Q}$) it is necessary to prove that the operations above actually produce real numbers. In Nuprl such "well-formedness" proofs are generally required and usually trivial. With the constructive reals, though, these proofs are anything but routine, and in fact are the reason that the definitions are so complicated. Thus the following theorems deserve to be mentioned:

```
* THM radd_wf
∀a:ℝ. ∀b:ℝ. a + b ∈ ℝ

* THM rneg_wf
∀a:ℝ. -a ∈ ℝ

* THM rsub_wf
∀a:ℝ. ∀b:ℝ. a - b ∈ ℝ

* THM real_rat_wf
∀a:ℚ. a ∈ ℝ

* THM rmul_gen_wf
∀a:ℝ. ∀b:ℝ. |b(1)| ≤ |a(1)| ⇒ a *g b ∈ ℝ

* THM rmul_wf
∀a:ℝ. ∀b:ℝ. a * b ∈ ℝ

* THM rmax_wf
∀a:ℝ. ∀b:ℝ. max{a,b} ∈ ℝ

* THM rabsval_wf
∀a:ℝ. |a| ∈ ℝ

* THM rinv_wf
∀a:ℝ⁻°. 1/a ∈ ℝ

* THM rdiv_wf
∀a:ℝ. ∀b:ℝ⁻°. a/b ∈ ℝ
```

Another concern is that functions are well defined with respect to real equality. That is, they do not depend on the particular sequences representing their arguments. For example, the canonical bound of a number is *not* functional.

```
* THM radd_functionality
∀a1:ℝ. ∀a2:ℝ. ∀b1:ℝ. ∀b2:ℝ. a1 = a2 ⇒ b1 = b2 ⇒ a1 + b1 = a2 + b2

* THM rneg_functionality
∀a:ℝ. ∀b:ℝ. a = b ⇒ -a = -b
```

```
* THM rsub_functionality
∀a1:ℝ. ∀a2:ℝ. ∀b1:ℝ. ∀b2:ℝ. a1 = a2 ⇒ b1 = b2 ⇒ a1 - b1 = a2 - b2
```

```
* THM real_rat_functionality
∀a:ℚ. ∀b:ℚ. a = b ⇒ a = b
```

This theorem is not quite as trivial as it looks — the equality on the left is rational equality whereas the one on the right is real equality.

```
* THM rmul_functionality
∀a1:ℝ. ∀a2:ℝ. ∀b1:ℝ. ∀b2:ℝ. a1 = a2 ⇒ b1 = b2 ⇒ a1 * b1 = a2 * b2
```

```
* THM rmax_functionality
∀a1:ℝ. ∀a2:ℝ. ∀b1:ℝ.  ∀b2:ℝ. a1 = a2 ⇒ b1 = b2 ⇒ max{a1,b1} =
                                                        max{a2,b2}
```

```
* THM rabsval_functionality
∀a:ℝ. ∀b:ℝ. a = b ⇒ |a| = |b|
```

```
* THM rinv_functionality
∀a:ℝ⁻°. ∀b:ℝ⁻°. a = b ⇒ 1/a = 1/b
```

```
* THM rdiv_functionality
∀a1:ℝ. ∀a2:ℝ. ∀b1:ℝ⁻°. ∀b2:ℝ⁻°. a1 = a2 ⇒ b1 = b2 ⇒ a1/b1 = a2/b2
```

Similarly the order relations are well defined:

```
* THM pos_functionality
∀a:ℝ. ∀b:ℝ. a = b ⇒ (a > 0 ⇐⇒ b > 0)
```

```
* THM lt_r_functionality
∀a1:ℝ. ∀a2:ℝ. ∀b1:ℝ.  ∀b2:ℝ. a1 = a2 ⇒ b1 = b2 ⇒ (a1 < b1 ⇐⇒
                                                          a2 < b2)
```

```
* THM non_neg_functionality
∀a:ℝ. ∀b:ℝ. a = b ⇒ (a ≥ 0 ⇐⇒ b ≥ 0)
```

```
* THM le_r_functionality
∀a1:ℝ. ∀a2:ℝ. ∀b1:ℝ.  ∀b2:ℝ. a1 = a2 ⇒ b1 = b2 ⇒ (a1 ≤ b1 ⇐⇒
                                                         a2 ≤ b2)
```

2.3 Some Theorems

The surprising thing about the definitions is that they can be thought of in a very intuitive way. For instance, the following lemma says that two real numbers are equal if and only if they have the same limit.

```
* THM eq_real_lemma
∀x:ℝ. ∀y:ℝ. x = y ⟺ (∀j:ℕ⁺. ∃N:ℕ⁺. ∀n:{N...}. |x(n) - y(n)| ≤ 1/j)
```

This lemma is useful in showing that real equality is transitive, and thinking about real equality in terms of limits makes this fact obvious. However, from the definition alone it seems that transitivity does not hold — if |a(n) - b(n)| ≤ 2/n and |b(n) - c(n)| ≤ 2/n, surely |a(n) - c(n)| can be made to be greater than 2/n ! The solution is that the convergence rate in the definition of $\mathbb{R}$ prevents this from happening (the proposed arrangement disqualifies at least one of a, b, and c from being a real number).

Another characterization of real equality is

```
* THM eq_real_lemma2
∀a:ℝ. ∀b:ℝ. a = b ⟺ (∀i:ℕ⁺. ∀j:ℕ⁺. |a(i) - b(j)| ≤ 1/i + 1/j)
```

which says that if you confuse equal numbers a and b (that is, if you interchange a(i) and b(i) for various i), the resulting sequence is a valid real number.

One fact that was especially useful in proving arithmetic properties is that any subsequence of a real number is again a real number, equal to the first. Subsequences are defined by composing a monotone function on $\mathbb{N}^+$ with the real (which is a function $\mathbb{N}^+ \rightarrow \mathbb{Q}$).

```
* ABS monotone
mono(f) == ∀a:ℕ⁺. ∀b:ℕ⁺. a < b ⇒ f(a) < f(b) ;;

* THM real_subsequence
∀f:ℕ⁺ → ℕ⁺. mono(f) ⇒ (∀a:ℝ. λi.a(f(i)) = a)

* THM real_subsequence2
∀f:ℕ⁺ → ℕ⁺. mono(f) ⇒ (∀a:ℝ. (λi.a(f(i))) ∈ ℝ)
```

There are lemmas about the order relations which provide some more intuition. In particular, a number is nonnegative if and only if its limit is, and positive if and only if there is a 1/n that the sequence stays above from n on (which is a little stricter than saying that the limit is positive).

```
* THM real_pos_lemma
∀a:ℝ. a > 0 ⟺ (∃N:ℕ⁺. ∀m:{N...}. 1/N ≤ a(m))

* THM real_non_neg_lemma
∀a:ℝ. a ≥ 0 ⟺ (∀n:ℕ⁺. ∃N:ℕ⁺. ∀m:{N...}. -1/n ≤ a(m))
```

Now for some more interesting results. For example, the nth element of a real actually approximates the real to within 1/n.

```
* THM rat_approx
∀a:ℝ. ∀n:ℕ⁺. |a - a(n)| ≤ 1/n
```

Also, as one might expact, the rationals are dense in the reals.

```
* THM rat_dense
∀a:ℝ. ∀b:ℝ. a < b ⇒ (∃q:ℚ. a < q ∧ q < b)
```

The following fact is quite important and plays a significant role in the intermediate value theorem. It is not true in general that `a ≤ 0 ∨ a > 0` for real `a`, but given an arbitrarily small positive epsilon we can do almost as well.

```
* THM pos_quasi_decidable
∀a:ℝ. ∀ε:ℝ⁺. a < ε ∨ a > 0
```

2.4 The Intermediate Value Theorem

Our implementation of the intermediate value theorem uses the bisection method, and follows the proof outline given in an earlier system [3]. The theorem is:

```
* THM ivt
∀a:ℝ. ∀b:ℝ. a < b ⇒ (∀f:ℝ → ℝ. f cont on [a,b] ⇒
              f(b) > 0 ⇒ -(f(a)) > 0 ⇒ (∀ε:ℝ⁺. ∃c:[a,b]. |f(c)| < ε))
```

The procedure is to let `c` be the midpoint of `a` and `b`, and decide if `|f(c)| < ε`. If so then we are done, and otherwise `|f(c)| > 0` by `pos_quasi_decidable`. It follows easily that either `f(c) > 0` or `-(f(c)) > 0`. In the former case consider the interval `[c,b]`, and in the latter, `[a,c]`, and repeat the procedure on this new interval. To see that this terminates, note that the diameter of the intervals goes to zero, and by continuity the diameter of the range of `f` goes to zero as well. This range always includes both positive and negative values, so eventually it lies wholly inside `(-ε, ε)`.

The proof generating this algorithm is an induction proof with the following induction hypothesis:

```
* ABS bisects
bisects(a; b; f; ε; n) == (∃c:[a,b]. |f(c)| < ε) ∨
              (∃α:[a,b]. ∃β:[a,b].  β - α = 1/2^n * b - a ∧
                                          f(β) > 0 ∧ -(f(α)) > 0) ;;
```

The proof that this holds for all `n` is interesting in that the base case and the induction step are almost identical. The base case is the lemma `bisection_step`, and in the induction step we just instantiate this lemma on the appropriate sub-interval.

```
* THM bisection_step
∀a:ℝ. ∀b:ℝ. a < b ⇒ (∀f:ℝ → ℝ. f(b) > 0 ⇒ -(f(a)) > 0 ⇒
                                  (∀ε:ℝ⁺. bisects(a; b; f; ε; 1)))
```

```
* THM bisection_induction
∀a:ℝ. ∀b:ℝ. a < b ⇒ (∀f:ℝ → ℝ. f(b) > 0 ⇒ -(f(a)) > 0 ⇒
                                 (∀ε:ℝ⁺. ∀n:ℕ⁺. bisects(a; b; f; ε; n)))
```

The lemma that tells us how many steps to take is

```
* THM halving_to_zero
∀a:ℝ. ∀b:ℝ. a > 0 ⇒ (∃n:ℕ⁺. 1/2^n * b < a)
```

and together with continuity and the following lemma provide all that is necessary to complete the proof.

```
* THM rabsval_across_zero
∀a:ℝ. ∀b:ℝ. b > 0 ⇒ -a > 0 ⇒ |b| < |b - a|
```

Now we can look at the details, though perhaps some explanation is in order.

A Nuprl proof is a tree where each node has a numbered list of hypotheses and declarations, a conclusion (or *goal*), and a tactic name. If a tactic succeeds in proving the goal from the hypotheses then the node becomes a leaf. Otherwise the tactic generates one or more *subgoals* which, if proven, are sufficient to justify the claim of the current node. Thus the overall idea is to start with the statement of the theorem at the root and gradually reduce it to trivialities by applying tactics.

In reading this proof, the easiest way to understand what the tactics do is probably to just look past them and see what subgoals they generate.

```
* THM ivt
>> ∀a:ℝ. ∀b:ℝ. a < b ⇒
|   (∀f:ℝ → ℝ. f cont on [a,b] ⇒ f(b) > 0 ⇒ -(f(a)) > 0 ⇒
|                                        (∀ε:ℝ⁺. ∃c:[a,b]. |f(c)| < ε))
BY UnivCD THENA Auto
 \
  1. a: ℝ
  2. b: ℝ
  3. a < b
  4. f: ℝ → ℝ
  5. f cont on [a,b]
  6. f(b) > 0
  7. -(f(a)) > 0
  8. ε: ℝ⁺
  >> ∃c:[a,b]. |f(c)| < ε
  |
  BY Unfold 'continuous' 5
   \
    5. ∃w: ℝ → ℝ.  ∀ε:ℝ⁺. w(ε) > 0 ∧
    |     (∀x:[a,b]. ∀y:[a,b]. |x - y| ≤ w(ε) ⇒ |(f(x)) - (f(y))| ≤ ε)
    |
```

```
>> ∃c:[a,b]. |f(c)| < ε
|
BY D 5
THENM With 'ε' (D 6)
THENM D (-1)
THENA Auto
 \
  5. w: ℝ → ℝ
  9. w(ε) > 0
  10. ∀x:[a,b]. ∀y:[a,b]. |x - y| ≤ w(ε) ⇒ |(f(x)) - (f(y))| ≤ ε
  >> ∃c:[a,b]. |f(c)| < ε
  |
  BY Using ['b','b - a'] (FwdThruLemma 'halving_to_zero' [9])
  THENM D (-1)
  THENA Auto
   \
    11. n: ℕ+
    12. 1/2^n * b - a < w(ε)
    >> ∃c:[a,b]. |f(c)| < ε
    |
    BY InstLemma 'bisection_induction' ['a';'b';'f';'ε';'n']
    THENM Unfold 'bisects' (-1)
    THENA Auto
     \
      13.  (∃c:[a,b]. |f(c)| < ε) ∨
      |          (∃α:[a,b]. ∃β:[a,b].  β - α = 1/2^n * b - a ∧
      |                                    f(β) > 0 ∧ -(f(α)) > 0)
      >> ∃c:[a,b]. |f(c)| < ε
      |
      BY D 13
      THEN Auto % If the first disjunct holds then we're done. %
      THEN RepeatMFor 4 (D (-1))
      THENA Auto
       \
        13. α: [a,b]
        14. β: [a,b]
        15. β - α = 1/2^n * b - a
        16. f(β) > 0
        17. -(f(α)) > 0
        >> ∃c:[a,b]. |f(c)| < ε
        |
        BY With 'β' (D 10)
        THENM With 'α' (D (-1))
        THENA Auto
```

```
\
10. n: ℕ+
11. 1/2^n * b - a < w(ε)
12. α: [a,b]
13. β: [a,b]
14. β - α = 1/2^n * b - a
15. f(β) > 0
16. -(f(α)) > 0
17. |β - α| ≤ w(ε) ⇒ |(f(β)) - (f(α))| ≤ ε
>> ∃c:[a,b]. |f(c)| < ε
|
BY Rewrite (\e. HigherC (RevHypC 14 e) e) 11
THENM Assert 'β - α = |β - α|'
THENM Rewrite (\e. HigherC (HypC (-1) e) e) (-2)
THENM OnHyps [(-1);13] Thin
THENA Auto
|\
| 11. α: [a,b]
| 12. β: [a,b]
| 13. β - α = 1/2^n * b - a
| 14. f(β) > 0
| 15. -(f(α)) > 0
| 16. |β - α| ≤ w(ε) ⇒ |(f(β)) - (f(α))| ≤ ε
| 17. β - α < w(ε)
| >> β - α = |β - α|
| |
| BY BackThruLemma `eq_real_sym`
| THENM BackThruLemma `rabsval_zero`
| THENA Auto
|  \
|   >> β - α ≥ 0
|   |
|   BY Rewrite (\e. HigherC (HypC 13 e) e) 0
|   THENM BackThruLemma `rmul_preserves_non_neg`
|   THENM Try (Fold `le_r` 0)
|   THENM Try (BackThruLemma `le_r_weakening_lt_r`)
|   THEN Auto
|    \
|     >> 1/2^n ≥ 0
|     |
|     BY Rewrite (HigherC real_to_ratC) 0
|     THENM BackThruLemma `expon_preserves_non_neg`
|     THEN Auto
|     THEN le_q_expand 0
```

```
|     THENM Rewrite (HigherC qnd_evalC) 0
|     THEN Auto
\
 11. α: [a,b]
 12. β: [a,b]
 13. f(β) > 0
 14. -(f(α)) > 0
 15. |β - α| ≤ w(ε) ⇒ |(f(β)) - (f(α))| ≤ ε
 16. |β - α| < w(ε)
 >> ∃c:[a,b]. |f(c)| < ε
 |
 BY FwdThruLemma 'le_r_weakening_lt_r' [16]
 THENM FwdThruHyp 15 [(-1)]
 THENM OnHyps [(-2);16;15] Thin
 THENA Auto
  \
   15. |(f(β)) - (f(α))| ≤ ε
   >> ∃c:[a,b]. |f(c)| < ε
   |
   BY With 'β' (D 0)
   THENM Rewrite (\e. HigherC (RevHypC 15 e) e) 0
   THENM BackThruLemma 'rabsval_across_zero'
   THEN Auto
```

3 Tactics

The following is a listing of the user defined tactics in the rational and real libraries together with illustrations of their use. Some do a fair amount of computation or rewriting while others are basically macros for various sequences of tactics that show up repeatedly in the libraries. "Direct computation" tactics perform only term reduction and unfolding and folding of definitions.

`CaseSplit` This tactic decomposes an `ifthenelse` term and rewrites the Boolean condition to a proposition. From `qabsval_le_q`:

```
1. a: Q
>> a ≤ if (a.num * -a.den) <z (-a.num * a.den) then -a else a
|
BY CaseSplit 0
|\
| 2. a.num * -a.den < -a.num * a.den
| >> a ≤ -a
 \
  2. -a.num * a.den ≤ a.num * -a.den
```

```
  >> a ≤ a
```

`qnd_evalC` This is a direct computation tactic that reduces `a/b.num` to `a` and `a/b.den` to `b`. From `eq_q_refl`:

```
1. x1: ℤ
2. x2: ℕ+
>> x1/x2.num * x1/x2.den = x1/x2.num * x1/x2.den
|
BY Rewrite (HigherC qnd_evalC) 0
 \
  >> x1 * x2 = x1 * x2
```

`expon_unrollC` This unrolls the definition of `expon` using direct computation.

```
* ABS expon_gen
expon_gen == λf,a,n.if n =z 1 then a else (a * f(a)(n - 1))  ;;

* ABS expon
a^n == Y(expon_gen)(a)(n) ;;
```

The effect is to replace `a^n` with `if n =z 1 then a else (a * a^(n - 1))`. From `expon_preserves_pos`:

```
1. a: ℚ
2. 0. < a
3. n: ℤ
4. 1 < n
5. 0. < a^(n - 1)
>> 0. < a^n
|
BY Rewrite (HigherC expon_unrollC) 0
THENM CaseSplit 0
THEN Auto
 \
  6. ¬n = 1
  >> 0. < (a * a^(n - 1))
```

`lt_q_expand` Converts `a < b` to `a.num * b.den < b.num * a.den`. `le_q_expand` and `eq_q_expand` are similar. From `lt_q_qmax`:

```
1. a: ℚ
2. b: ℚ
3. a < b
>> max{a,b} = b ∈ ℚ
|
BY Unfold 'qmax' 0
THEN lt_q_expand 3
THENA Auto
 \
  3. a.num * b.den < b.num * a.den
  >> if (a.num * b.den) <z (b.num * a.den) then b else a  = b ∈ ℚ
```

`qabsval_evalC` This tactic rewrites absolute values of rational arithmetic expressions to expressions with absolute values in the numerators. From `rat_triangle`:

```
1. a: ℚ
2. b: ℚ
3. c: ℚ
>> |a - c| ≤ |a - b| + |b - c|
|
BY Rewrite (HigherC qabsval_evalC) 0
THENA Auto
THEN Try (BackThruLemma 'mul_zero_lt')
THEN Auto
 \
  >> |a.num * c.den + -c.num * a.den|/(a.den * c.den) ≤
            |a.num * b.den + -b.num * a.den|/(a.den * b.den) +
                  |b.num * c.den + -c.num * b.den|/(b.den * c.den)
```

`SAuto` This is a version of the autotactic that tries calling `AbSetMemEqTypeCD`. It is useful for handling membership goals of the form `a = b ∈ T` where `T` is a set type. From `real_pos_inc`:

```
1. r: ℝ
2. x1: ℕ+
3. 1/x1 < r(x1)
>> <r,x1> ∈ r:ℝ × {n:ℕ+| 1/n < |r(n)|}
|
BY Assert '1/x1 < |r(x1)|'
THEN SAuto
 \
  >> 1/x1 < |r(x1)|
```

rat_to_intC Rewrites expressions of rational integers (rationals of the form `a/1`) to integer expressions. In the example, from `canon_bound_property`, the conclusion is an equality between rationals.

```
1. a: ℝ
2. n: ℕ+
3. a(1).den ≠ 0
>> |a(1).num ÷ a(1).den| + 3 = |a(1).num ÷ a(1).den| + 1 + 2
|
BY Rewrite (HigherC rat_to_intC) 0
THEN Auto
```

int_to_ratC Rewrites integer expressions to rational ones.

RAuto Like `SAuto` except that if it gets stuck on a membership goal it will try to unfold the type subterm. It is useful for goals like `a = b ∈ ℝ` where the type needs unfolding. From `rat_nzero_to_real_nzero`:

```
1. a: ℚ
2. a.num ≠ 0
3. 0 < |a|
4. n: ℕ+
5. 1/n < |a|
>> <a,n> = <a,n> ∈ ℝ-o
|
BY RAuto
 \
  >> 1/n < |a(n)|
```

PosHD Decomposes a hypothesis of the form `a > 0` and obtains a witness, although it sometimes must be used with the tactic `Unhide`. From `real_pos_lemma`:

```
1. a: ℝ
2. a > 0
>> ∃N:ℕ+. ∀m:{N...}. 1/N ≤ a(m)
|
BY PosHD 2
THEN Unhide
THENA Auto
 \
  2. n: ℕ+
  3. 1/n < a(n)
  >> ∃N:ℕ+. ∀m:{N...}. 1/N ≤ a(m)
```

PosCD This takes a term as an argument to be used as a witness for a goal of the form `a > 0`. From `rneg_functionality_wrt_lt_r`:

```
1. a: ℝ
2. b: ℝ
3. n: ℕ+
4. 1/n < (b - a)(n)
>> (-a - -b) > 0
|
BY PosCD 'n'
THENA Auto
 \
  >> 1/n < (-a - -b)(n)
```

rnw_evalC This tactic is similar to `qnd_evalC` except that it takes pairs which are nonzero reals. In this example from `pos_quasi_decidable` the term ε in the conclusion is actually a display for `rnum(ε)`, which becomes `rnum(<ε1,n>)` after `(D 2)` is applied.

```
1. a: ℝ
2. ε: ℝ+
>> a < ε ∨ a > 0
|
BY New ['ε1';'n'] (D 2)
THEN Rewrite (HigherC rnw_evalC) 0
 \
  2. ε1: ℝ
  3. n: ε1 > 0
  >> a < ε1 ∨ a > 0
```

real_apply_evalC This direct computation tactic takes a real expression applied to a natural number, and "distributes" the application inside the expression according to the definitions of the operations involved. The operations it knows about are **radd**, **rneg**, **rsub**, **real_rat**, **rmul_gen**, **rmax** and **rabsval**. From `radd_preserves_pos`:

```
1. a: ℝ
2. b: ℝ
3. c: ℝ
4. N: ℕ+
5. ∀m:{N...}. 1/N ≤ (b - a)(m)
>> ∃N:ℕ+. ∀m:{N...}. 1/N ≤ (b + c - a + c)(m)
|
```

```
BY With 'N' (D 0)
THENM Rewrite (HigherC real_apply_evalC) 5
THENM Rewrite (HigherC real_apply_evalC) 0
THENA Auto
 \
  5. ∀m:{N...}. 1/N ≤ b(2 * m) - a(2 * m)
  >> ∀m:{N...}.
 1/N ≤ b(2 * (2 * m)) + c(2 * (2 * m)) - a(2 * (2 * m)) +
                                                 c(2 * ( 2 * m))
```

real_to_ratC Rewrites expressions of reals which are constant sequences to rational expressions.

rat_to_realC Rewrites rational expressions to real expressions, but does not recognize `qinv` or `qdiv`. The reason for this is that `rinv` and `rdiv` must be given nonzero reals, requiring witnesses. In this example from `halving_to_zero` the conclusion is a rational expression. It gets converted to a real expression and then is decomposed so that it matches hypothesis 7.

```
1. a: ℝ
2. b: ℝ
3. a > 0
4. 0 < a
5. q: ℚ
6. n: ℕ+
7. 1/n < (q - 0)(n)
8. q < a
9. r: ℚ
10. b < r
11. b > 0
>> 0 < q
|
BY Rewrite (HigherC rat_to_realC) 0
THENM Unfold 'lt_r' 0
THENM PosCD 'n'
THEN Auto
```

rat_to_real_witnessC Same as `rat_to_realC` except that it handles `qinv` and `qdiv`. It takes a natural number to be used as a witness for all instances of `rinv` and `rdiv`. This tactic does not get used in the library so we have a cooked up example:

```
1. a: ℚ
2. b: ℚ
3. c: ℚ
```

```
4. c.num ≠ 0
5. n: ℕ+
>> -|a| + b - (1/c * b) = -c - (b + c * a)
|
BY Rewrite (rat_to_real_witnessC '7') 0
THENA SAuto
|\
| >> <c,7> = <c,7> ∈ ℝ-o
|\
| >> 0 < 7
 \
  >> -|a| + b - (1/<c,7> * b) = -c - (b + c * a)
```

A Tactic Code

The tactics described above, together with some special-purpose lemmas to which they refer.

```
* ML case_split
let CaseSplit =
    \i.(if (i=0) then
         (IfThenElseCases 0
          THENM RW bool_to_propC (-1)
          THENA Auto)
        else
         (IfThenElseCases i
          THENM RW bool_to_propC i
          THENA Auto)) ;;

* ML qnd_eval
let qnd_evalC = UnfoldsTopC ``qnumer qdenom``
   ANDTHENC AddrC [1] (UnfoldTopC `qnum`)
   ANDTHENC (pi1_evalC ORELSEC pi2_evalC)
  ;;

* ML expon_unroll
let expon_unrollC =
    FwdMacroC `expon_unrollC`
      (UnfoldC `expon`
       ANDTHENC HigherC YUnrollC
       ANDTHENC UnfoldC `expon_gen`
       ANDTHENC ReduceC
       ANDTHENC FoldC `expon_gen`
       ANDTHENC FoldC `expon`
```

```
      )
     'a^n'
    ;;
```

```
* ML expand_defs
let lt_q_expand =
    \i. (Unfold 'lt_q' i
         THEN RWH (LemmaC 'assert_of_qlt_rw') i)
       ;;

  let le_q_expand =
    \i. (Unfold 'le_q' i
         THEN Unfold 'qle' i
         THEN RWH (LemmaC 'assert_of_bnot') i
         THENM RWH (LemmaC 'assert_of_qlt_rw') i)
       ;;

  let eq_q_expand =
    \i. (Unfold 'eq_q' i
         THEN Unfold 'eq_rat' i
         THEN RWH (LemmaC 'assert_of_eq_int') i)
       ;;
```

```
* ML qabsval_eval
let qabsval_evalC =
      RepeatC (UnfoldsC ''qdiv qinv qsub qneg qmul qadd'')
      ANDTHENC HigherC qnd_evalC
      ANDTHENC LemmaC 'qabsval_to_iabsval'
      ANDTHENC HigherC qnd_evalC
      ANDTHENC RepeatC (FoldsC ''qadd qmul qneg qsub qinv qdiv'')
    ;;
```

For readability, different notations have been used in the following theorems. a. is the rational number a/1.

* THM ri_eq_q
$\forall$a:$\mathbb{Z}$. $\forall$b:$\mathbb{Z}$. a. $=_Q$ b. $\Longleftrightarrow$ a = b

* THM ri_lt_q
$\forall$a:$\mathbb{Z}$. $\forall$b:$\mathbb{Z}$. a. $<_Q$ b. $\Longleftrightarrow$ a < b

* THM ri_le_q
$\forall$a:$\mathbb{Z}$. $\forall$b:$\mathbb{Z}$. a. $\leq_Q$ b. $\Longleftrightarrow$ a $\leq$ b

* THM ri_qneg_to_minus
$\forall$a:$\mathbb{Z}$. -(a.) = (-a). $\in \mathbb{Q}$

```
* THM ri_qadd_to_add
∀a:ℤ. ∀b:ℤ. a. +_Q b. = (a + b). ∈ ℚ

* THM ri_qsub_to_sub
∀a:ℤ. ∀b:ℤ. a. -_Q b. = (a - b). ∈ ℚ

* THM ri_qmul_to_mul
∀a:ℤ. ∀b:ℤ. a. *_Q b. = (a * b). ∈ ℚ

* THM ri_qabsval_to_iabsval
∀a:ℤ. |a.| = |a|. ∈ ℚ

* ML collect_numer
let collect_numerC =
      TryC
        (SweepUpC (FirstC (map LemmaC ``ri_qneg_to_minus ri_qadd_to_add
           ri_qsub_to_sub ri_qmul_to_mul ri_qabsval_to_iabsval``))
         )
    ;;

* ML convert_rel
let convert_relC =
      TryC
        (FirstC (map LemmaC ``ri_eq_q ri_lt_q ri_le_q``)
        )
    ;;

* ML rat_to_int
% rat_to_intC
    -----------
  If a term is of the form
      exp1 R exp2
  where exp1 and exp2 are arithmetic expressions of rational
  integers, and R is eq_q, lt_q, or le_q, rat_to_intC will
  convert it to an equivalent integer expression.
  %

  let rat_to_intC =
      HigherC collect_numerC
      ANDTHENC convert_relC
    ;;

* ML spread_numer
let spread_numerC =
      TryC
        (SweepDnC (FirstC (map RevLemmaC ``ri_qneg_to_minus ri_qadd_to_add
           ri_qsub_to_sub ri_qmul_to_mul ri_qabsval_to_iabsval``))
         )
    ;;
```

```
* ML rev_convert_rel
let rev_convert_relC =
      TryC
        (FirstC (map RevLemmaC ``ri_eq_q ri_lt_q ri_le_q``)
        )
    ;;

* ML int_to_rat
let int_to_ratC =
      rev_convert_relC
      ANDTHENC HigherC spread_numerC
    ;;

* ML pos_d
let RAuto = Auto THEN
        \p.((if is_equal_term (concl p) then
              (UnfoldAtAddr [1] 0 THEN SAuto)
          else Id) p)
        ;;

  let PosHD = \i.(Unfold `pos` i
          THEN Unfold `exists_sq` i
          THEN AbSetHD i)
        ;;

  let PosCD t = Unfold `pos` 0
          THEN Unfold `exists_sq` 0
          THEN (Refine `dependent_set_memberFormation`
                [mk_level_exp_arg(mk_level_exp[`i`,0]);
                 mk_term_arg t;
                 mk_var_arg `mephistopheles']
          THENL [AddHiddenLabel `aux`;AddHiddenLabel `main`;
                 Try (Reduce 0) THEN RAuto THEN SIAuto
                                          THEN Thin (-1) ])
        ;;

* ML rnw_eval
let rnw_evalC = UnfoldsTopC ``rnum rwit``
     ANDTHENC (pi1_evalC ORELSEC pi2_evalC)
    ;;

* ML real_apply_eval
let radd_evalC =
    SimpleMacroC `radd_evalC`
    'a +r b n' '(a (2 * n)) + (b (2 * n))' ``radd``
  ;;
```

```
let rneg_evalC =
  SimpleMacroC `rneg_evalC`
  '-a n' '-(a n)' ``rneg``
;;

let rmax_evalC =
  SimpleMacroC `rmax_evalC`
  'max{a,b} n' 'max{(a n),(b n)}' ``rmax``
;;

let rmul_gen_evalC =
  SimpleMacroC `rmul_gen_evalC`
  'a * b n' '((a (2 * (Ka * n))) * (b (2 * (Ka * n))))' ``rmul_gen``
;;

let rsub_evalC =
  MacroC `rsub_evalC`
  (AllC [UnfoldC `rsub`;radd_evalC;AddrC [2] rneg_evalC])
     'a -r b n'
  (UnfoldC `qsub`) 'a (2 * n) - b (2 * n)'
;;

let rabsval_evalC =
  MacroC `rabsval_evalC`
  (AllC [UnfoldC `rabsval`;rmax_evalC;AddrC [2] rneg_evalC])
     '|a| n'
  (UnfoldC `qabsval`) '|a n|'
;;

let real_apply_evalC =
  (SweepDnC
    (FirstC [rabsval_evalC;rmax_evalC;rmul_gen_evalC;
             rsub_evalC;rneg_evalC;radd_evalC])
   ANDTHENC TryC (UnfoldC `real_rat`
                  ANDTHENC ReduceC))
;;
```

Again we use some more visible notation. a• is the real number λi.a.

```
* THM rr_qadd_to_radd
∀a:ℚ. ∀b:ℚ. (a +_Q b)• = a• +_R b• ∈ ℝ

* THM rr_qneg_to_rneg
∀a:ℚ. (-a)• = -(a•) ∈ ℝ

* THM rr_qsub_to_rsub
∀a:ℚ. ∀b:ℚ. (a -_Q b)• = a• -_R b• ∈ ℝ
```

* THM rr_qmul_to_rmul
$\forall a{:}\mathbb{Q}.\ \forall b{:}\mathbb{Q}.\ (a *_Q b)\cdot = a\cdot *_R b\cdot \in \mathbb{R}$

* THM rr_qinv_to_rinv
$\forall a{:}\mathbb{Q}^{-o}.\ \forall n{:}\mathbb{N}^{+}.\ (1/a)\cdot = 1/\langle a\cdot,n\rangle \in \mathbb{R}$

* THM rr_qdiv_to_rdiv
$\forall a{:}\mathbb{Q}.\ \forall b{:}\mathbb{Q}^{-o}.\ \forall n{:}\mathbb{N}^{+}.\ (a \div b)\cdot = a\cdot/\langle b\cdot,n\rangle \in \mathbb{R}$

* THM rr_qabsval_to_rabsval
$\forall a{:}\mathbb{Q}.\ |a|\cdot = |a\cdot| \in \mathbb{R}$

* THM rr_eq_real
$\forall a{:}\mathbb{Q}.\ \forall b{:}\mathbb{Q}.\ a =_Q b \iff a\cdot =_R b\cdot$

* THM rr_lt_r
$\forall a{:}\mathbb{Q}.\ \forall b{:}\mathbb{Q}.\ a <_Q b \iff a\cdot <_R b\cdot$

* THM rr_le_r
$\forall a{:}\mathbb{Q}.\ \forall b{:}\mathbb{Q}.\ a \leq_Q b \iff a\cdot \leq_R b\cdot$

* THM rr_pos
$\forall a{:}\mathbb{Q}.\ 0 <_Q a \iff a\cdot > 0$

* THM rr_non_neg
$\forall a{:}\mathbb{Q}.\ 0 \leq_Q a \iff a\cdot \geq 0$

```
* ML collect
let collectC =
        TryC
           (SweepUpC (FirstC (map RevLemmaC ``rr_qneg_to_rneg
             rr_qadd_to_radd rr_qsub_to_rsub rr_qinv_to_rinv
             rr_qmul_to_rmul rr_qdiv_to_rdiv rr_qabsval_to_rabsval``))
            )
         ;;
```

```
* ML convert
let convertC =
        TryC
          (FirstC (map RevLemmaC ``rr_eq_real rr_lt_r rr_le_r rr_pos
             rr_non_neg``)
           )
        ;;
```

```
* ML real_to_rat
let real_to_ratC =
       HigherC collectC
       ANDTHENC convertC
     ;;

* ML disseminate
let disseminateC =
        TryC
          (SweepDnC (FirstC (map LemmaC ``rr_qneg_to_rneg rr_qadd_to_radd
             rr_qsub_to_rsub rr_qmul_to_rmul rr_qabsval_to_rabsval``))
           )
        ;;

  let disseminate_witnessC n =
        TryC
          (SweepDnC ((FirstC (map LemmaC ``rr_qneg_to_rneg rr_qadd_to_radd
                rr_qsub_to_rsub rr_qmul_to_rmul rr_qabsval_to_rabsval``))
             ORELSEC (LemmaWithC [`n',n] `rr_qinv_to_rinv`)
             ORELSEC (LemmaWithC [`n',n] `rr_qdiv_to_rdiv`))
           )
        ;;

* ML rev_convert
let rev_convertC =
        TryC
          (FirstC (map LemmaC ``rr_eq_real rr_pos rr_non_neg rr_lt_r
             rr_le_r``)
           )
        ;;

* ML rat_to_real
let rat_to_realC =
        rev_convertC
        ANDTHENC HigherC disseminateC
       ;;

  let rat_to_real_witnessC n =
        rev_convertC
        ANDTHENC HigherC (disseminate_witnessC n)
       ;;
```

References

[1] E. Bishop. *Foundations of Constructive Analysis.* McGraw-Hill, New York, 1967.

[2] D. S. Bridges. *Constructive Functional Analysis.* Pitman, London, 1979.

[3] F. Brown, J. Chirimar and D. J. Howe, unpublished Nuprl libraries.

[4] J. Chirimar and D. J. Howe. Implementing Constructive Real Analysis: A Preliminary Report. In *Symposium on Constructivity in Computer Science*, *Lecture Notes in Computer Science*, Springer Verlag, 1991.

[5] R. L. Constable, et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[6] W. M. Farmer and F. J. Thayer, Two Computer-Supported Proofs in Metric Space Topology, *Notices of the AMS*, Nov 1991, Vol. 38, number 9, pp. 1133–1138.

[7] J. Harrison, Constructing the real numbers in Higher Order Logic.

[8] J. Harrison and L. Théry, Extending the HOL theorem prover with a Computer Algebra System to Reason about the Reals.

[9] J. T. Udding, A Theory of Real Numbers and its Presentation in AUTOMATH, Vol. I–III, Master's Thesis, Eindhoven University of Technology, 1980.