Computing with Classical Real Numbers

CEZARY KALISZYK
RUSSELL O'CONNOR
ICIS, Radboud University Nijmegen, The Netherlands
{cek,roconnor}@cs.ru.nl

There are two incompatible Coq libraries that have a theory of the real numbers; the Coq standard library gives an axiomatic treatment of classical real numbers, while the CoRN library from Nijmegen defines constructively valid real numbers. Unfortunately, this means results about one structure cannot easily be used in the other structure. We present a way interfacing these two libraries by showing that their real number structures are isomorphic assuming the classical axioms already present in the standard library reals. This allows us to use O'Connor's decision procedure for solving ground inequalities present in CoRN to solve inequalities about the reals from the Coq standard library, and it allows theorems from the Coq standard library to apply to problem about the CoRN reals.

1. INTRODUCTION

Coq is a proof assistant based on dependent type theory developed at INRIA [CDT08]. By default, it uses constructive logic via the Curry-Howard isomorphism. This isomorphism associates propositions with types and proofs of propositions with programs of the associated type. This makes Coq a functional programming language as well as a deduction system. The identification of a programming language with a deduction system allows Coq to reason about programs and allows Coq to use computation to prove theorems.

Coq can support classical reasoning by the declaration of additional axioms; however, these additional axioms will not have any corresponding computational component. This limits the use of computation to prove theorems, since Coq cannot compute the normal form of an expression where the head is an axiom. There are theories that allow program extraction from classical proofs, like A-translation, but this has not been done for proofs involving real numbers.

At least two different developments of the real numbers have been created for Coq. The Coq standard library declares the existence of the real numbers axiomatically. This library also requires the axioms for classical logic. It gives users the familiar, classical, real numbers as a complete ordered Archimedian field.

The other formalization of the real numbers is done constructively in the CoRN library [CFGW04]. This library specifies what a *constructive real number structure* is, and proves that all such structures are isomorphic. These real numbers are constructive and there is one efficient implementation where real numbers can be evaluated to arbitrary precision within Coq.

In this paper we show how to connect these two developments of the theory of the real numbers by showing that Coq's real numbers form a real number structure in CoRN. We do this by:

—Deriving some logical consequences of the classical real numbers (Section 2).

Specifically, we formally prove the well-known result that sentences in Π_1^0 are decidable. Bishop and Bridges [BB85] call it the *principle of omniscience* and consider it the root of nonconstructivity in classical mathematics.

—Using these logical consequences to prove that the classical real numbers form a constructive real number structure (Section 3).

—Using the resulting isomorphism between classical and constructive real numbers to prove some classical real number inequalities by evaluating constructive real number expressions (Section 4).

1.1 The two universes of Coq

Coq has a mechanism for program extraction [Let02]. Programs developed in Coq can be translated into Ocaml, Haskell, or Scheme. If these programs are proved correct in Coq, then the extracted programs have high assurance of correctness.

To facilitate extraction, Coq has two separate universes: the `Set` universe, and the `Prop` universe (plus an infinite series of `Type` universes on top of these two). The `Prop` universe is intended to contain only logical propositions and its values are discarded during extraction. The types in the `Set` universe are computationally relevant; the values of these types make up the extracted code. In order to maintain the soundness of extraction, the type system prevents information from flowing from the `Prop` universe to the `Set` universe. Otherwise, vital information could be thrown away during extraction, and the extracted programs would not run.

The `Prop`/`Set` distinction will play an important role in our work. The logical operators occur in both universes. The following table lists some logical operations and their corresponding syntax for both the `Prop` and `Set` universes.

Math Notation	`Prop` Universe	`Set` Universe
$A \wedge B$	`A /\ B`	`A * B`
$A \vee B$	`A \/ B`	`A + B`
$A \to B$	`A -> B`	`A -> B`
$\neg A$	`~A`	*not used*
$\forall x : X.P(x)$	`forall x:X, P x`	`forall x:X, P x`
$\exists x : X.P(x)$	`exists x:X, P x`	`{ x : X | P x }`

One might think that proving that classically defined real numbers satisfy the requirements of a constructive real number structure would be trivial. It seems that the constructive requirements be no stronger than the classical requirement for a real number structure when we use classical reasoning. However, Coq's `Prop`/`Set` distinction prevents a naive attempt at creating such an isomorphism between the classical and constructive real numbers.

The difficulty is that classical reasoning is only allowed in the `Prop` universe. A constructive real number structure requires a `Set`-level existence in the proof that a sequence converges to its limit (see Section 3.1), but the theory provided by the Coq standard library only proves a classical `Prop`-level existence. It is not allowed to use the x given by the `Prop` existential:

```
exists x:X, P x
```

to fulfill the requirement of a set existential:

```
{ x : X | P x }.
```

There may be alternative ways to prove that the Coq's classical reals form a constructive real number structure by completely ignoring the classical existence and extracting a witness from CoRN, but it still remains to be seen if this is feasible. We present our original solution that transforms the classical existentials provided by the Coq standard library into a constructive existential. This solution uses Coq's classical real number axioms to create constructive existentials from classical existentials for any Π_1^0 sentence (Section 2).

2. LOGICAL CONSEQUENCES OF COQ'S REAL NUMBERS

The Coq standard library defines the classical real numbers axiomatically. This axiomatic definition has some general logical consequences. In this section we present some of the axioms used to define the real numbers and then show how they imply the decidability of Π_1^0 sentences. The axioms of the real numbers cannot be effectively realized, so a decision procedure for Π_1^0 sentences is not implied by this decidability result.

2.1 The axiomatic definition of the real numbers

The definition for the reals in the Coq standard library asserts a set $\mathbb{R}$, the constants 0, 1, and the basic arithmetic operations:

```
Parameter R : Set.
Parameter R0 : R.
Parameter R1 : R.
Parameter Rplus : R -> R -> R.
Parameter Rmult : R -> R -> R.
...
```

A numeric literal is simply a notation for an expression, for example `20` is a notation for the binary encoding of 20 in terms of `R0` and `R1`:

```
(R1+R1)*((R1+R1)*(R1+(R1+R1)*(R1+R1)))
```

In addition to the arithmetic operations, an order relation is asserted.

```
Parameter Rlt : R -> R -> Prop.
```

Axioms for these operations and relations define their semantics. There are 17 axioms. We show only some relevant ones; the entire list of axioms can be found in the Coq standard library. The properties described by the axioms include associativity and commutativity of addition and multiplication, distributivity, and neutrality of zero and one.

```
Axiom Rplus_comm : forall r1 r2:R, r1 + r2 = r2 + r1.
...
```

There are also several axioms that state that the order relation for the real numbers form a total order. The most important axiom for our purposes will be the law of trichotomy. We describe the reasons for its shape (in particular why it is Set-based) in next subsection:

```
Axiom total_order_T :
  forall r1 r2:R, {r1 < r2} + {r1 = r2} + {r1 > r2}.
```

Finally, there is an Archimedian axiom (where `IZR` is the obvious injection $\mathbb{Z} \to \mathbb{R}$) and an axiom stating the least upper bound property.

```
Parameter up : R -> Z.
Axiom archimed : forall r:R, IZR (up r) > r /\ IZR (up r) - r <= 1.

Axiom completeness :
    forall E:R -> Prop,
      bound E -> (exists x : R, E x) -> {m : R | is_lub E m}.
```

2.2 Decidability of Π_1^0 sentences

It is important to notice that the trichotomy axiom uses `Set`-style disjunctions. This means that users are allowed to write functions that make decisions by comparing real numbers. This axiom might look surprising to a constructivist since real numbers are infinite structures and therefore comparing them is impossible in finite time in general. The motivation for this definition comes from classical mathematics where mathematicians regularly create functions based on real number trichotomy. It allows one to define a step function, which is not definable in constructive mathematics.

This trichotomy axiom can be used to decide any Π_1^0 property. For any decidable predicate over natural numbers P we first define a sequence of terms that take values when the property is true:

$$a_n =_{\text{def}} \begin{cases} \frac{1}{2^n} & \text{if } P(n) \text{ holds} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

We can then define the sum of this infinite sequence, which is guaranteed to converge:

$$S =_{\text{def}} \sum_{n=0}^{\infty} a_n \tag{2}$$

The trichotomy axiom allows us to compare S with 2. It follows that if $S = 2$ then the predicate P holds for every natural number, and if $S < 2$ then it is not the case (the case of $S > 2$ is easily ruled out). Furthermore, this distinction can be made in `Set` universe.

We formalized the above reasoning in Coq and we obtained the following logical theorem.

```
forall_dec
   : forall P : nat -> Prop,
     (forall n : nat, {P n} + {~ P n}) ->
     {(forall n : nat, P n)} + {~ (forall n : nat, P n)}
```

This statement means that:

$$\text{dec } \Sigma_n^0 \longrightarrow \text{dec } \Pi_{n+1}^0,$$

and since Σ_0^0 is decidable this implies that Π_1^0 is decidable. To see why this implies the decidability of particular Π_1^0 sentence, consider an arbitrary Π_1^0 sentence φ. If φ is Π_0^0, then it is decidable by the basic properties of Π_0^0 sentences. Otherwise, φ is of the form $\forall n : \mathbb{N}.\psi(n)$ where $\psi(n)$ is decidable. The above lemma allows us to conclude that φ is decidable from the fact that $\psi(n)$ is decidable.

2.2.1 *Constructive indefinite description.* We can extend the previous result by using a general logical lemma of Coq. The constructive indefinite description lemma states that if we have a decidable predicate over the natural numbers, then we can convert a `Prop` based existential into a `Set` based one. Its formal statement can be found in the standard library:

```
constructive_indefinite_description_nat
   : forall P : nat -> Prop,
     (forall x : nat, {P x} + {~ P x}) ->
     (exists n : nat, P n) -> {n : nat | P n}
```

This lemma can be seen as a form of Markov's principle in Coq. The lemma works by doing a bounded search for a new witness satisfying the predicate. The witness from the `Prop` based existential is only used to prove termination of the search. No information flows from the `Prop` universe to the `Set` universe because the witness found for the `Set` based existential is independent of the witness from the `Prop` based one.

Classical logic (included by `Reals`) allows us to convert a negated universal statement into an existential statement in `Prop`:

```
not_all_ex_not
   : forall (U : Type) (P : U -> Prop),
     ~ (forall n : U, P n) -> exists n : U, ~ P n
```

By combining these theorems with our previous result, we get a theorem whose conclusion is either a constructive existential or a universal statement:

```
sig_forall_dec
   : forall P : nat -> Prop,
     (forall n : nat, {P n} + {~ P n}) ->
     {n : nat | ~ P n} + {(forall n : nat, P n)}
```

3. THE CONSTRUCTION OF THE ISOMORPHISM

In this section we briefly present the algebraic hierarchy present in CoRN (it is described in detail in [GPWZ02] and [CF04]). We show that Coq's reals fulfill the requirements of a constructive real number structure, and hence they are isomorphic to any other real number structure.

3.1 Building a constructive reals structure based on Coq's reals

The collection of properties making up a real number structure in CoRN is broken down to form a hierarchy of different structures. The first level, `CSetoid`, defines the properties for equivalence and apartness. The next level is `CSemigroup` which defines some properties for addition. More structures are defined on top of each other all the way up to a constructive ordered field structure — `COrdField`. Up to

this point trichotomy is not required. Finally, the `CReals` structure is defined on top of the `COrderedField` structure. The full list of structures is given below.

CSetoid	–	constructive setoid
CSemiGroup	–	semi group
CMonoid	–	monoid
CGroup	–	group
CAbGroup	–	Abelian group
CRing	–	ring
CField	–	field
COrdField	–	ordered field
CReals	–	real number structure

To prove that classical reals form a constructive real number structure, we created instances of all these structures for the classical real numbers (called `RSetoid`, `RSemigroup`, etc.). For example, `RSetoid` is the constructive setoid based on Coq's real numbers. The carrier is `R`, while standard Coq equality (equivalent to Leibnitz equality) and its negation are used as the equality and apartness relations. The proofs of the setoid properties of these relations are simple.

The basic arithmetic operations from Coq's real numbers are shown to satisfy all the properties required up to `COrdField`. The proofs of these properties follow straightforwardly from similar properties provided by the Coq standard library. For details, we refer the reader to CoRN source files [CoR09]. We present just the final step, the creation of the `CReals` structure based on the ordered field.

Two additional operations are required to form a constructive real numbers structure from a constructive ordered field: the limit operation and a function that realizes the Archimedian property. The limit operation is the only step where the facts about Coq's reals cannot naïvely be used to instantiate the required properties. This is because the convergence property of limits for Coq's reals only establishes that there exists a point where the sequence gets close to the limit using the `Prop` based quantifier, whereas `CReals` requires such a point to exist using the `Set` based quantifier. One cannot directly convert a `Prop` based existential into a `Set` based one, because information is not allowed to flow from the `Prop` universe to the `Set` universe.

The goal that remains to be proved in Coq is to show that if for any ϵ there is an index in a sequence N such that all further elements in the sequence are closer to the limit value than ϵ. The related property from the Coq standard library is shown as hypothesis `u`.

```
e : R
e0 : 0 < e
u : forall eps : R,  eps > 0 -> exists N : nat,
    forall n : nat,
    (n >= N)%nat -> Rfunctions.R_dist (s n) x < eps
______________________________________(1/1)
{N : nat | forall m : nat,
  (N <= m)%nat -> AbsSmall e (s m[-]x)}
```

In order to prove this goal, we first reduce the `Set` based existential to a `Prop` based one using the `constructive_indefinite_description_nat`.

Applying this lemma to the goal above reduces the problem to the following:

```
e : R
e0 : 0 < e
u : forall eps : R,  eps > 0 -> exists N : nat,
    forall n : nat,
    (n >= N)%nat -> Rfunctions.R_dist (s n) x < eps
______________________________________(2/2)
exists N : nat, forall m : nat,
  (N <= m)%nat -> AbsSmall e (s m[-]x)}
```

This now follows easily from the hypothesis. However, we are also required to prove the decidability of the predicate:

```
______________________________________(1/2)
{(forall m : nat, (x0 <= m)%nat -> AbsSmall e (s m[-]x))} +
{~ (forall m : nat, (x0 <= m)%nat -> AbsSmall e (s m[-]x))}
```

This goal appears hopeless at first because we are required to prove the decidability of a Π_1^0 sentence. However, we can use the `forall_dec` lemma from the previous section to prove the decidability of this sentence. This possible since the property:

```
P(m) := (x0 <= m)%nat -> AbsSmall e (s m[-]x))
```

is decidable. This completes the proof that the classical real numbers form a constructive real number structure.

3.2 The isomorphism

Niqui shows in Section 1.4 of his PhD thesis [Niq04] that all constructive reals structures are isomorphic, the proof is present in CoRN as `iso_CReals`. The constructed isomorphism defines two maps that are inverses of each other and proves that the isomorphism preserves the constants 0 and 1, arithmetic operations and limits. More details can be found in [Niq04].

In order to use the isomorphism in an effective way, we need to show that the definitions of basic constants and the operations are preserved. Since the reals of the Coq standard library are written as `R` and CoRN reals as `IR`, we called the two functions of the isomorphism `RasIR` and `IRasR`. From Niqui's construction, one obtains the basic properties of this isomorphism:

—Preserves equality and inequalities:

```
Lemma R_eq_as_IR : forall x y, (RasIR x [=] RasIR y -> x = y).
Lemma IR_eq_as_R : forall x y, (x = y -> RasIR x [=] RasIR y).
Lemma R_ap_as_IR : forall x y, (RasIR x [#] RasIR y -> x <> y).
Lemma IR_ap_as_R : forall x y, (x <> y -> RasIR x [#] RasIR y).
Lemma R_lt_as_IR : forall x y, (RasIR x [<] RasIR y -> x < y).
...
```

—Preserves constants: 0, 1 and basic arithmetic operations: $+$, $-$, $*$. In the properties listed below we do not show the dual theorems that state the same

facts for opposite translation `IRasR`. Those are easy to prove using the properties of `RasIR`.

```
Lemma R_Zero_as_IR : (RasIR R0 [=] Zero).
Lemma R_plus_as_IR :
  forall x y, (RasIR (x+y) [=] RasIR x [+] RasIR y).
...
```

An important difference between the definition of real numbers in the Coq standard library and in CoRN is the way partiality is handled. Partial functions are defined as total functions for Coq's reals, but their properties require proofs that the function parameters are in the appropriate domain. For example, division is defined as a total operation on real numbers; however, all the axioms that specify properties of division have assumptions that the reciprocal is not zero. This means that the term $\frac{1}{0}$ is some real number, but it is not possible to prove which one it is.

In CoRN, partial functions require an additional argument, the domain condition. Division is a three argument operation; the third argument is a proof that the divisor is apart from zero. Other partial functions, such as the logarithm, are defined in a similar way. We prove that this isomorphism preserves these partial functions. These lemmas require a proof that the arguments are in the proper domain to be passed to the domain conditions of the CoRN functions.

—Preserves the reciprocal and division for any proof:

```
Lemma R_div_as_IR : forall x y (Hy : Dom (f_rcpcl' IR) (RasIR y)),
  (RasIR (x/y) [=] (RasIR x [/] RasIR y [//] Hy)).
```

Niqui's theorem proves the basic arithmetic operations and limits are preserved by the isomorphism. However, the real number structure does not specify any transcendental functions. The existence of the transcendental functions follows from the axiomatization, but the actual definitions used in the axiomatizations do not need to be the same. Therefore it is necessarily to manually prove that these functions are preserved by the isomorphism. This is easy if the Coq and CoRN definitions are similar, but becomes difficult if the two systems choose different definitions for the same function. We thus prove some more properties that the isomorphism preserves:

—Preserves infinite sums:
The proof that the values of the sums are the same requires the decidability of Π^0_1 sentences and `constructive_indefinite_description_nat`. The term `prf` is the proof that the series converges.

```
Lemma R_infsum_as_IR : forall (y: R) a,
  Rfunctions.infinit_sum a y -> forall prf,
  RasIR y [=] series_sum (fun i : nat => RasIR (a i)) prf.
```

—Preserves transcendental functions: *exp*, *sin*, *cos*, *tan*, *ln*

```
Lemma R_exp_as_IR : forall x,
    RasIR (exp x) [=] Exp (RasIR x).
Lemma R_sin_as_IR : forall x,
    RasIR (sin x) [=] Sin (RasIR x).
```

```
Lemma R_cos_as_IR : forall x,
    RasIR (cos x) [=] Cos (RasIR x).
Lemma R_tan_as_IR : forall x dom,
    RasIR (tan x) [=] Tan (RasIR x) dom.
Lemma R_ln_as_IR : forall x dom,
    RasIR (ln x) [=] Log (RasIR x) dom.
```

We finally prove that the isomorphism preserves the constant π. This was more difficult because the π in Coq is defined as the infinite sum

$$\pi_{Coq} =_{\text{def}} \sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1}, \tag{3}$$

while in CoRN π is defined as the limit of the sequence

$$pi_n =_{\text{def}} \begin{cases} 0 & \text{if } n = 0 \\ pi_{n-1} + cos(pi_{n-1}) & \text{otherwise} \end{cases} \tag{4}$$

$$\pi_{CoRN} =_{\text{def}} \lim_{n\to\infty} pi_n. \tag{5}$$

Both libraries contain proofs that the sine of π is equal to zero[1], and additionally that it is the smallest positive number with this property. Using these properties it is possible to show that indeed the two definitions describe the same number:

```
Lemma R_pi_as_IR : RasIR (PI) [=] Pi.
```

4. COMPUTATION WITH CLASSICAL REALS

4.1 Solving ground inequalities

O'Connor's work on fast real numbers in CoRN includes a semi-decision procedure for solving strict inequalities on constructive real numbers. With the isomorphism it is possible to use it to solve some goals for classical reals.

Consider the example of proving $e^{\pi} - \pi < 20$ for the classical real numbers. The difference between these numbers is very small, so the proof is hard without using numeric approximations:

```
______________________________________(1/1)
exp PI - PI < 20
```

Our tactic first converts the Coq inequality to a CoRN inequality by using the fact that the isomorphism preserves inequalities. Then it recursively applies the facts about the isomorphism to convert the Coq terms on both sides of the inequality and their corresponding CoRN terms. This is done with using a rewrite database and the autorewrite mechanism for setoids. The advantage of using a rewrite database is that it can be easily extended with new facts about new functions being preserved under the isomorphism. The disadvantage of this method is that the setoid rewrite mechanism is fairly slow in Coq 8.1.

[1]The proof from the Coq standard library relies on an unproven lemma that $sin\left(\frac{\pi}{2}\right) = 1$ which is currently stated as an axiom.

```
______________________________________(1/1)
Exp Pi[-]Pi[<](One[+]One)[*]
              ((One[+]One)[*](One[+](One[+]One)[*](One[+]One)))
```

(Recall that, in Coq, the real number 20 is simply notation for $(1+1)*((1+1)*(1+(1+1)*(1+1)))$.)

Once the expression is converted to a CoRN expression, the semi-decision procedure from CoRN can be applied (which itself uses another rewrite database to change the representation again). This semi-decision procedure may not terminate. If the two sides of the inequality are different, it will approximate the real numbers accurately enough to either prove the required inequality (or fail if the inequality holds in the other direction). If the two sides are equal, then the search for a sufficient approximation will never terminate.

The decision procedure for CoRN takes an argument which is used for the starting precision of the approximation. Setting it to an appropriate value can make search faster, if the magnitude of difference between the sides is known *a priori*. Our decision procedure also takes this an argument and passes it on to the CoRN tactic.

We have shown the intermediate step above for illustration purposes only. The actual tactic proves the theorem in one step:

```
Example xkcd217 : (exp PI - PI < 20).
R_solve_ineq (1#1)%Qpos.
Qed.
```

Automatic rewriting is not enough to convert partial functions like division and logarithm. The additional parameters needed in CoRN are the domain conditions. The tactic itself could be called recursively to generate the assumptions. Unfortunately Coq 8.1 cannot automatically rewrite inside dependent products, making the recursive tactic more difficult to create. Coq 8.2's new setoid rewriting system will allow rewriting in dependent products, and we expect this to greatly simplify the creation of a recursive tactic.

4.2 Using facts about Coq's reals in CoRN

The Coq standard library contains more properties of real numbers than CoRN. It also contains more tactics, like `fourier` for solving linear constraints. By using the isomorphism the other way, it is possible to apply these tactics while working with CoRN. Using the isomorphism this way is controversial because using the classically defined real numbers means that the axioms of classical logic are assumed.

We will show how a goal that would normally be proved by the `fourier` tactic in Coq's reals can be done in CoRN. We will show it on a very simple goal, but the procedure is similar in other cases:

$$x \leq y \Rightarrow x < y + 1. \tag{6}$$

The goal written formally in Coq is:

```
Goal forall x y:IR, (x [<=] y) -> (x [<] y [+] One).
```

After introducing the assumptions we can apply the isomorphism to the inequalities both in the assumptions and in the goal:

```
intros x y H; rapply IR_lt_as_R_back.
assert (HH := R_le_as_IR_back _ _ H).
```

This shows the following goal:

```
1 subgoal
x : IR
y : IR
H : x[<=]y
HH : IRasR x <= IRasR y
______________________________________(1/1)
IRasR x < IRasR (y[+]One)
```

Since the isomorphism preserves all the functions in the goal and assumptions, we can apply the facts to change the terms that include the isomorphism on the top of the term to terms that include the application of the isomorphism only on variables.

```
replace RHS with (IRasR y + IRasR One)
  by symmetry; rapply IR_plus_as_R.
replace (IRasR One) with 1. 2: symmetry; apply IR_One_as_R.

1 subgoal
x : IR
y : IR
H : x[<=]y
HH : IRasR x <= IRasR y
______________________________________(1/1)
IRasR x < IRasR y + 1
```

Now the `fourier` tactic is applicable:

```
fourier.

Proof Completed.
```

A similar transformation can be performed to use other facts and tactics from the Coq library.

5. RELATED WORK

Melquiond has created a Coq tactic that can solve some linear inequalities over real number expressions using interval arithmetic and bisection [Mel08]. This tactic is currently limited to expressions from arithmetic operations and square root, but could support transcendental functions via polynomial approximations. It has the advantage that it can solve some problems that involve constrained variables.

Many other proof assistants include facts about transcendental functions that could be used for approximating expressions that involve them. However there are few mechanisms for approximating real numbers automatically since to compute effectively this has to be done either by constructing the real numbers with approximation in mind, or by using special features of the proof assistant. The latter

approach is used effectively for example in HOL Light, due to a close interplay between syntax and semantics. The way a real number expression is described in HOL Light can be analyzed and it can be used to prove an approximation.

The construction of HOL Light real numbers is described in [Har98]. The approximation mechanism of HOL Light is provided in `calc_real` part of the distribution of the prover as the `REALCALC_REL_CONV` conversion. This conversion uses the fact that terms are transparent and decomposes a term or goal into subterms, looking for underlying underlying real number operations or constants. Implementing approximation as a conversion means that it is not available as a function inside statements of theorems, but instead while proving a goal about real number expression it is possible to ask for an approximation of a particular closed term. The conversion uses rewriting to generate a theorem that approximates a particular term.

Obua developed a computing library for Isabelle [NPW02]. In his PhD [Obu08] he shows examples of computing bounds on real number expressions using computation rather than deduction.

Lester implemented approximation of real number expressions in PVS [Les08]. Results of real number functions are proved to have fast converging Cauchy sequences when their parameters have fast converging Cauchy sequences. Cauchy sequences for many real number functions are effective and can be evaluated inside PVS.

6. CONCLUSION

We have formalized a proof that the axioms of Coq's classical real numbers imply the decidability of Π^0_1 statements. We used this fact to prove that these classical real numbers form a constructive real number structure. Then we used the fact that all real number structures are isomorphic to use tactics designed for one domain to solve problems in the other domain. In particular, we showed how to automatically prove a class of strict inequalities on real number expressions.

The lemmas showing the decidability of Π^0_1 statements have been added to the standard library and are made available in the 8.2 release of Coq. The isomorphism and the tactics used to prove inequalities over Coq's reals have been added to the CoRN library. They are available with the version of CoRN compatible with Coq 8.2.

6.1 Future Work

We wish to extend our tactics to solve inequalities over terms that involve partial functions. This should be easier to do when CoRN uses the new setoid rewrite mechanisms available in Coq 8.2. Currently CoRN uses the old setoid rewrite mechanism which means that the translation of expressions from one domain to another is quite slow. We would like to investigate ways that this could be made faster. We would also like to automate the translation from CoRN expressions to Coq expressions so that CoRN can have its own `fourier` tactic assuming classical logic.

References

[BB85] Errett Bishop and Douglas Bridges. *Constructive Analysis*, chapter 1.3. Springer-Verlag, Berlin, October 1985.

[CDT08] The Coq Development Team. *The Coq Proof Assistant Reference Manual, version 8.2*. LogiCal project, 2008. Distributed electronically at `http://coq.inria.fr/doc-eng.html`.

[CF04] Luís Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. PhD thesis, University of Nijmegen, April 2004.

[CFGW04] Luís Cruz-Filipe, Herman Geuvers, and Freek Wiedijk. C-CoRN, the constructive coq repository at nijmegen. In Andrea Asperti, Grzegorz Bancerek, and Andrzej Trybulec, editors, *MKM*, volume 3119 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 2004.

[CoR09] Constructive Coq Repository at Nijmegen, 2009. `http://corn.cs.ru.nl/`.

[GPWZ02] Herman Geuvers, Randy Pollack, Freek Wiedijk, and Jan Zwanenburg. A constructive algebraic hierarchy in Coq. *Journal of Symbolic Computation, Special Issue on the Integration of Automated Reasoning and Computer Algebra Systems*, 34(4):271–286, 2002.

[Har98] John Harrison. *Theorem Proving with the Real Numbers*. Springer-Verlag, 1998.

[Les08] David R. Lester. Real number calculations and theorem: Proving validation and use of an exact arithmetic. In Otmane Ait-Mohamed, editor, *TPHOLs*, volume 5170 of *Lecture Notes in Computer Science*, pages 215–229. Springer, 2008.

[Let02] Pierre Letouzey. A new extraction for Coq. In Herman Geuvers and Freek Wiedijk, editors, *TYPES*, volume 2646 of *Lecture Notes in Computer Science*, pages 200–219. Springer, 2002.

[Mel08] Guillaume Melquiond. Proving bounds on real-valued functions with computations. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, Lectures Notes in Computer Science, Sydney, Australia, 2008.

[Niq04] Milad Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. PhD thesis, Radboud Universiteit Nijmegen, September 2004.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[Obu08] Steven Obua. *Flyspeck II: The Basic Linear Programs*. PhD thesis, Technische Universitat Munchen, 2008. submitted.