

# AN EFFICIENT MODEL OF CONSTRUCTIVE REAL NUMBERS IN COQ

DANKO ILIĆ

## CONTENTS

1. Introduction	1
2. Real numbers	2
2.1. Theory of intervals	2
2.2. Theory of sets of intervals	4
2.3. Completeness and limits	6
3. Defining Euler's number	7
3.1. Consistency of $\mathbf{E}$	8
3.2. Fineness of $\mathbf{E}$	10
3.3. Speed of computation	11
Appendix A. Actual definitions and lemmas	11
A.1. Intervals with rational endpoints	11
A.2. Real numbers	17
A.3. Defining Euler's number	21
A.4. Intervals with real endpoints	27
A.5. Sets of intervals with real endpoints	31
A.6. Limits and completeness	31
A.7. Inverse function theorem	32
A.8. Proposed axiomatisation of the theories of intervals	33
References	35

## 1. INTRODUCTION

The work before you is an experiment in implementing exact real arithmetic in the Coq proof assistant. Although such efforts have already been done by others, see [1][2][3], this attempt has taken another theoretical starting point. Following Stolzenberg's approach from [4], we consider the real number system built as follows.

One considers a theory of closed intervals with rational endpoints which is an Archimedean ordered field and a lattice, and also a binary relation of 'consistency' on the intervals which holds when two intervals intersect. This relation is not transitive, and so is not an equality, but one may profit from thinking about it as a kind of equality.

Next, one considers *sets* of the above intervals equipped with the same algebraic structure, inherited directly from the previous one, with the exception that inheriting the consistency relation now gives us a real equality relation on the sets of intervals. One says that a set of intervals is 'fine' if it contains an interval of

arbitrary small length, and that it is ‘consistent’ if every two contained intervals intersect. A fine and consistent set of intervals is a ‘real number’.

One now reflects for a moment and sees no problem in defining a theory of intervals with ‘real’ endpoints by copying the developed theory of intervals with rational endpoints, and analogously defining a theory of *sets* of intervals with ‘real’ endpoints. One defines for the fine and consistent sets of intervals with ‘real’ endpoints a limit functor and shows that the theory from the third paragraph above is complete, that it contains all of its limits, and so that a ‘real number’ is really a real number.

What is the advantage of this approach in relation the the approaches based on Cauchy sequence, or Dedekind cut, representation of real numbers? The author is not really sure. He can only express the feeling, drawn from the experience in defining Euler’s number in it, that working with this framework does not pose heavy demands on using it in concrete situations.

When implementing a theory like this one, one should not be concerned solely with the theoretical model. In practice, having a good and fast library implementing integer and rational number arithmetic, has greater importance. For example, the rational arithmetic library from the standard library of Coq 8.1, although fast, is not completely complete, for it is missing, for example, the definition of min and max and their properties. We shall remark more on such points when time comes.

The notation for writing mathematics in this report will be the type theoretic notation of Coq. This is so because here we speak about implementations and this requires formality. The reader is advised to read the introduction chapter of the book [4] for a (better and) purely mathematical treatment.

## 2. REAL NUMBERS

**2.1. Theory of intervals.** An interval with rational endpoints is given by a left and right bound and a proof that left is left of right:

```
Record QI : Set :=
  QImake { QIleft    : Q;
          QIright   : Q;
          QInonempty : (QIleft<=QIright)%Q }.
```

2.1.1. *Order, consistency.* The less-than relation on this set determines whether the first operand (interval) is completely left of the second and is defined as:

```
Definition QIlt (I J:QI) := (QIright I) < (QIleft J).
```

We want to have another relation, consistency, which determines whether two intervals intersect. It is defined like this:

```
Definition QIc (I J:QI) :=
  (Qmax (QIleft I) (QIleft J)) <= (Qmin (QIright I) (QIright J)).
```

Already at this stage we encounter a problem with the Coq standard library, as Qmax and Qmin are not defined there, and so we have to develop a few lemmas about them ourselves.

The consistency relation plays the role of an equality very often, like one can see from the following lemmas:

```
Lemma QIlt_trichotomy : forall I J:QI, (I<J) \\/ (I~J) \\/ (J<I).
```

```
Definition QIlc (I J:QI) := I<J \\/ I~J.
```

```
Infix "<~" := QIlc (at level 70) : QI_scope.
```

Lemma QIlc\_not\_gt : forall I J:QI, (I<~J) <-> not (I>J).

Lemma QIc\_sym : forall I J:QI, I~J -> J~I.

Lemma QIc\_lc : forall I J:QI, I~J <-> I<~J /\ J<~I.

Lemma QIlt\_lc\_lt : forall I J K L:QI, I<J -> J<~K -> K<L -> I<L.

The proofs of all these go by unfolding the definitions and applying the lemmas about Qmax and Qmin. The reader himself can run the proofs if he is interested in the details.

2.1.2. *Superemum, infimum.* The following least upper bound and greatest lower bound functions create the lattice structure on the intervals:

Definition QImax (I J:QI) := QImake  
 (Qmax (QIleft I) (QIleft J)) (Qmax (QIright I) (QIright J))  
 (QImax\_nonempty I J) : QI.

Definition QImin (I J:QI) := QImake  
 (Qmin (QIleft I) (QIleft J)) (Qmin (QIright I) (QIright J))  
 (QImin\_nonempty I J) : QI.

We have these lemmas:

Lemma QImax\_el : forall I J:QI, forall x y:Q,  
 QIel x I -> QIel y J -> QIel (Qmax x y) (QImax I J).

Lemma QImin\_el : forall I J:QI, forall x y:Q,  
 QIel x I -> QIel y J -> QIel (Qmin x y) (QImin I J).

Lemma QImax\_lt : forall I J K:QI, (QImax I J)<K <-> I<K /\ J<K.

Lemma QIlt\_max : forall I J K:QI, K<(QImax I J) <-> K<I \/ K<J.

And we should also, following Stolzenberg, have these:

forall I J:QI, (QImax I J) ~ I <-> (QImin I J) ~ J <-> I<~J.

forall I J:QI, QIsubset (QImax I J) I \/ QIsubset (QImin I J) J.

But, they are not there yet, mostly because they were not needed in the development of the library yet.

2.1.3. *Arithmetic.*

Definition QIsum (I J:QI) :=  
 QImake (QIleft I + QIleft J) (QIright I + QIright J)  
 (Qplus\_le\_compat \_ \_ \_ (QInonempty I) (QInonempty J)) : QI.

Definition QIminus (I:QI) := QImake (- QIright I) (- QIleft I)  
 (Qopp\_le\_compat \_ \_ (QInonempty I)) : QI.

Definition QIdifference (I J:QI) := QIsum I (QIminus J) : QI.

Definition QIproduct (I J:QI) :=  
 let r:=QIleft I in  
 let s:=QIright I in  
 let u:=QIleft J in  
 let v:=QIright J in  
 QImake  
 (Qmin (Qmin (r\*u) (s\*v)) (Qmin (r\*v) (s\*u)))  
 (Qmax (Qmax (r\*u) (s\*v)) (Qmax (r\*v) (s\*u)))  
 (QIproduct\_nonempty I J) : QI.

Definition QIzero := QImake 0 0 (Qle\_refl 0) : QI.

Definition QImagnitude (I:QI) := QImax I (QIminus I).

Definition QIquotient1 (J:QI) (J\_correct:not (QIzero ~ (QImagnitude J))) :=

QImake (Qinv (QIright J))(Qinv (QIleft J))(QIquotient\_nonempty \_ J\_correct).  
 Definition QIquotient (I J:QI)(J\_correct:not (QIzero ~ (QImagnitude J))) :=  
 QIproduct I (QIquotient1 J J\_correct).

We have proved the following properties:

Lemma QIsum\_assoc : forall I J K:QI, ((I+J)+K) == (I+(J+K)).  
 Lemma QIsum\_comm : forall I J:QI, (I+J) == (J+I).  
 Lemma QIeq\_c : forall I J:QI, I==J -> I~J.  
 Lemma QIsum\_assoc\_c : forall I J K:QI, ((I+J)+K) ~ (I+(J+K)).  
 Lemma QIsum\_comm\_c : forall I J:QI, (I+J) ~ (J+I).  
 Lemma QIproduct\_comm : forall I J:QI, (I\*J) == (J\*I).  
 Lemma QIproduct\_comm\_c : forall I J:QI, (I\*J) ~ (J\*I).  
 Lemma QIproduct\_assoc : forall I J K:QI, ((I\*J)\*K) == (I\*(J\*K)).

The last one is actually not proved, because it requires a hand-written handling of a lot of conceptually trivial cases. There are many other properties which one might state and prove in order to get a complete library, but we did not want to state anything that we might not use for proving the completeness theorem.

We also have the following:

Lemma QIel\_sum : forall I J:QI, forall x y:Q,  
 QIel x I -> QIel y J -> QIel (x+y) (I+J).  
 Lemma QIel\_product : forall I J:QI, forall x y:Q,  
 QIel x I -> QIel y J -> QIel (x\*y) (I\*J).  
 Theorem QIsum\_c\_compat : forall I J K L:QI, I~K -> J~L -> (I+J)~(K+L).  
 Theorem QIproduct\_c\_compat : forall I J K L:QI, I~K -> J~L -> (I\*J)~(K\*L).  
 Theorem QIc\_distrib : forall i j k I J K L:QI, i~I -> j~J -> k~K -> k~L ->  
 ((i+j)\*k) ~ ((I\*K)+(J\*L)).

The proofs of ones involving the product have not been completed because they require a lot of formalistic transformations.

#### 2.1.4. Length.

Definition QIlength (I:QI) := (QIright I - QIleft I)%Q.

Theorem QIsum\_length : forall I J:QI, (QIlength (I+J) == QIlength I + QIlength J)%Q.

Theorem QIdifference\_length : forall I J:QI, (QIlength (I-J) == QIlength I + QIlength J)%Q.

The concept of length will be necessary in what follows.

**2.2. Theory of sets of intervals.** We now generalize to *sets* of intervals. The goal of this subsection will be to present real numbers as special kinds of sets of intervals, namely sets that are fine and consistent.

Type theoretically (in the calculus of constructions), a set of intervals will be represented by the predicate:

Definition QIS := QI -> Prop.

If  $p$  is of type QIS and  $I$  of type QI, we write  $p I$  when we want to say that the predicate  $p$  holds for the interval  $I$ , i.e. that the interval  $I$  is in the set  $p$ .

**2.2.1. Order, arithmetic, consistency and fineness. Real numbers.** Two sets will be *consistent* when any two belonging intervals are. A set will be *smaller than* another set when there exist belonging intervals, the first completely left to the second. The *smaller-or-equal* relation on sets is a generalisation of the less-than-or-consistent relation on intervals:

Definition QISc (p q:QIS) := forall I J:QI, p I -> q J -> QIc I J.

Definition QISlt (p q:QIS) := exists2 I:QI, p I & exists2 J:QI, q J & QIlt I J.

Definition QISle (p q:QIS) := forall I J:QI, p I -> q J -> QIlc I J.

A set of intervals will be *fine* if we can find a belonging interval of arbitrarily small length:

Definition QISfine (p:QIS) := forall epsilon:Q, (0<epsilon)%Q -> exists I:QI, p I /\ (QIlength I < epsilon)%Q.

The arithmetic on sets of intervals is again a simple generalisation of arithmetic on intervals:

Definition QISsum (p q:QIS) (K:QI) := exists2 I:QI, p I & exists2 J:QI, q J & K = (I + J).

Definition QISminus (p:QIS) (K:QI) := exists2 I:QI, p I & K = (-I).

Definition QISdifference (p q:QIS) (K:QI) := exists2 I:QI, p I & exists2 J:QI, q J & K = (I - J).

Definition QISproduct (p q:QIS) (K:QI) := exists2 I:QI, p I & exists2 J:QI, q J & K = (I \* J).

Definition QISMagnitude (p:QIS) (K:QI) := exists2 I:QI, p I & K = QImagnitude I.

Definition QISzero (K:QI) := K == QIzero.

Definition QISquotient1 (p:QIS) (p\_gt\_zero:not (QISc QISzero p)) (K:QI) := forall I:QI, forall I\_gt\_zero:not (QIzero ~ (QImagnitude I)), p I -> K == (QIquotient1 I I\_gt\_zero).

Definition QISquotient (q p:QIS) (p\_gt\_zero:not (QISc QISzero p)) (K:QI) := forall J I:QI, forall I\_gt\_zero:not (QIzero ~ (QImagnitude I)), q J -> p I -> K == (QIquotient J I I\_gt\_zero).

Definition QISmax (p q:QIS) (K:QI) := exists2 I:QI, p I & exists2 J:QI, q J & K = QImax I J.

Definition QISmin (p q:QIS) (K:QI) := exists2 I:QI, p I & exists2 J:QI, q J & K = QImin I J.

### 2.2.2. Properties.

Theorem QISle\_not\_lt : forall p q:QIS, QISle p q -> not (QISlt q p).

Theorem QISle\_le\_c : forall p q:QIS, QISle p q -> QISle q p -> QISc p q.

Theorem QISlt\_lt\_lt : forall p w q:QIS,

QISc w w -> QISlt p w -> QISlt w q -> QISlt p q.

Theorem QISlt\_le\_lt : forall p w q:QIS,

```

QISfine q -> QISlt p w -> QISle w q -> QISlt p q.
Lemma QISsum_fine : forall p q : QIS, fine p -> fine q -> fine (p+q).
Lemma QISsum_cons : forall p q : QIS, cons p -> cons q -> cons (p+q).
Lemma QISproduct_cons : forall p q : QIS, cons p -> cons q -> cons (p*q).
Lemma QISmax_cons : forall p q : QIS, cons p -> cons q -> cons (QISmax p q).
Lemma QISmax_fine : forall p q : QIS, fine p -> fine q -> fine (max p q).
Lemma QISMagnitude_fine : forall p : QIS, fine p -> fine (QISMagnitude p).

```

The proofs of all these are somewhat formalistic and are best left to the reader to read from the Coq proof script. The fineness of the product is not proved there, because of the more complicated definition of the product.

2.2.3. *‘Real’ numbers.* Now we can formally define ‘real’ numbers as fine and consistent sets of intervals with rational end-points:

```

Record R : Type := Rmake {
  Rset : QIS;
  Rfine : QISfine Rset;
  Rconsistent : QISc Rset Rset }.

```

The equality of ‘real’ numbers will arise through our consistency relation:

```

Definition Req (p q : R) := QISc (Rset p) (Rset q).

```

Here  $(Rset\ x)$  extracts the carrier set from  $x$ . The order and arithmetic on ‘real’ numbers is defined through the order and arithmetic on sets of intervals, ie. like this:

```

Notation Rlt := (fun p q : R => QISlt (Rset p) (Rset q)).
Notation Rgt := (fun p q : R => QISlt (Rset q) (Rset p)).
Notation Rc := (fun p q : R => QISc (Rset p) (Rset q)).
Notation Rlc := (fun p q : R => QISle (Rset p) (Rset q)).
Notation Rle := (fun p q : R => (Rlt p q) \ / (Req p q)).
...

```

At this point we can use the properties from the previous subsection, as well as properties which should have been there (but are not due to time constraints), to get that the arithmetic is well-defined (sum of real numbers is a real number, ...), that the order is really a total order, and that  $Rmin$  and  $Rmax$  create a lattice. This point, in case the formalisation was finished, would show that we have an Archimedean ordered field and a lattice.

2.3. **Completeness and limits.** To show that ‘real’ numbers are really real numbers we need to show their completeness. This is achieved through a generalisation of the concept of Cauchy sequence: the *number-like sets* of intervals with real end-points.

2.3.1. *Generalising the theories of intervals and sets of intervals.* Analogously to the previous two subsections one can define the theories of intervals with real end-points, and of sets of intervals with real end-points.

This is in practice (in Coq) not easy to do. The main reason is that we need to have a library for our ‘real’ numbers that would mirror Coq’s rational number library. An additional difficulty is the fact that the later library itself is not sufficiently rich.

For now, in order to achieve the “greater” goal of proving formally the completeness, we just copy and paste the definitions and theorems about intervals with rational end-points into ones with real end-points. This development can be seen in appendices A.4 – A.5.

Another, better, development would be to abstract from the concrete rational/real-endpoints into endpoints over a Set. An attempt at this can be seen in appendix A.8. This approach was not followed for two reasons. First, the additional layer of abstraction makes things beurocratically more difficult to prove. Second, trying this approached revealed a bug in the Coq kernel, related to handling of universes, and thus the continuation of the approach had to wait for the bug to be fixed.<sup>1</sup>

2.3.2. *The completeness theorem.* Counting on the reader having browsed the mentioned appendices, we define a number-like set of intervals with ‘real’ endpoints:

```
Record NL : Type := NLmake {
  NLset : RIS;
  NLfine : RISfine NLset;
  NLconsistent : RISc NLset NLset }.
```

The limit functor gives, for each number-like set  $\Lambda$ , an QIS such that an interval  $I$  is inside the QIS if there is a interval  $K$  (with ‘real’ end-points) which is inside both  $\Lambda$  and  $I$ :

```
Definition Lim (Lambda:NL) (I:QI) :=
  exists K:RI, (NLset Lambda) K /\ RIinQI K I.
```

It should be obvious that each Cauchy sequence of ‘real’ numbers is a number-like set. The completeness theorem is now:

```
Theorem completeness : forall Lambda:NL,
  QISfine (Lim Lambda) /\ QISc (Lim Lambda) (Lim Lambda).
```

Both the fineness and consistency of `Lim Lambda` follow directly from the fineness and consistency of ‘real’ numbers. The proof has, however not been formalised, due to the lack of library for ‘real’ numbers.

### 3. DEFINING EULER’S NUMBER

We define the number  $e$  by defining  $e^{-1}$ , as is usually done, for computational purposes, as the series  $\sum_{l=0}^{\infty} (-1)^l / l!$ . The  $l$ -finite partial sum is defined as

```
Definition Eapprox (l:positive) :=
  let f := fun arg =>
    let sum := snd arg in
    let n := fst (fst arg) in
    let nfact := snd (fst arg) in
    let sum' := Qred (sum + if (Peven n) then -(1#nfact) else (1#nfact))
    in (Psucc n, Pmult n nfact, sum')
  in iter_pos l (positive*positive*Q) f (2%positive, 1%positive, (1#1)).
```

We just note that `Qred` normalises the partial sum by factoring with the GCD of the denominator and numerator. This has significant impact on the efficiency of computations, because as the representation of the sum in Coq increases manipulating it becomes more expensive.

<sup>1</sup>This is done in the latest version of Coq available throught the SVN repository.

Now we define the set of intervals which will become, in our framework, the real number  $e$ , once we prove its fineness and consistency. (this is not the only possible way to define  $E$ , but it seems to be the simplest one to prove properties about)

**Definition E** : QIS :=  
 fun I:QI =>  
   exists n:positive, I = (Einterval n).

where

**Definition Einterval'** (n:positive) : Q\*Q :=  
 let c := (Eapprox (2\*n-1)) in  
   (snd c, ((snd c)+(1#(snd (fst c))))%Q).

**Theorem Einterval'\_nonempty** : forall n:positive,  
 (fst (Einterval' n) <= snd (Einterval' n))%Q.

**Definition Einterval** (n:positive) : QI :=  
 let bounds:=Einterval' n in  
   QImake (fst bounds) (snd bounds) (Einterval'\_nonempty n).

### 3.1. Consistency of E.

**Theorem Econsistent** : QISc E E.

*Proof.* We have to prove that for any two intervals  $I$  and  $J$  which belong to  $e$ ,  $I \sim J$  ( $I$  and  $J$  are consistent, ie. intersect). Formally, what we have to prove is

$$Q_{\max} (Q_{\text{Ileft}} I) (Q_{\text{Ileft}} J) \leq Q_{\min} (Q_{\text{Iright}} I) (Q_{\text{Iright}} J)$$

The proofs of  $I \in e$  and  $J \in e$  give us  $k$  and  $n$  such that  $I = \text{Einterval } n$  and  $J = \text{Einterval } k$ . We make a case distinction:

$k = n$  Then  $I = J$  and we have to prove

$$Q_{\max} (Q_{\text{Ileft}} J) (Q_{\text{Ileft}} J) \leq Q_{\min} (Q_{\text{Iright}} J) (Q_{\text{Iright}} J)$$

which we easily obtain from the proof that left is left of right, a constituent of the definition of any interval.

$k < n$  Using lemmas *Enested-left* and *Enested-right* bellow, we obtain that  $Q_{\text{Ileft}} I < Q_{\text{Ileft}} J$  and  $Q_{\text{Iright}} J < Q_{\text{Iright}} I$ . From *max* being an upper bound of its operands and *min* their lower bound we get the required.

$k > n$  The same proof as in the previous case.

□

The lemmas necessary for the previous prove express the fact that  $E$  is a set of nested intervals.

**Lemma Enested\_left\_succ** : forall n:positive,  
 (QIleft (Einterval n) < QIleft (Einterval (Psucc n)))%Q.

**Lemma Enested\_left** : forall k n:positive, Pgt k n ->  
 (QIleft (Einterval n) < QIleft (Einterval k))%Q.

**Lemma Enested\_right\_succ** : forall n:positive,  
 QIright (Einterval (Psucc n)) < QIright (Einterval n).

**Lemma Enested\_right** : forall k n:positive, Pgt k n ->  
 (QIright (Einterval k) < QIright (Einterval n))%Q.

*Proof of Enested-left.* By induction on  $k$ . The case  $k = 1$  is impossible (we work with the type *positive*). For the induction case, given the two hypotheses

```
forall n : positive, Pgt p n ->
  QIleft (Einterval n) < QIleft (Einterval p)
```

and

```
Pgt (Psucc p) n
```

we should prove that

```
QIleft (Einterval n) < QIleft (Einterval (Psucc p))
```

We prove it by proving

```
QIleft (Einterval n) <= QIleft (Einterval p)
```

and

```
QIleft (Einterval p) < QIleft (Einterval (Psucc p))
```

The second goal is more complex and left for the separate lemma `Einterval-left-succ` below. The first is handled by using the following case distinction coming from the second hypothesis:

```
p > n Use the first hypothesis.
p = n Use the equality.
```

Before ending the proof we want to remark that in order to make the distinction, and many such other conceptually trivial inferences, we had to write a couple of lines of obfuscated code, as in the Coq standard library there is very limited support for the type `positive`.  $\square$

*Proof of `Enested-left-succ`.* After unfolding the definitions one has to prove that

```
forall n : positive,
  (snd (Eapprox (2 * n - 1)) < snd (Eapprox (2 * (n + 1) - 1)))%Q
```

`Eapprox` is defined in terms of the iterator `iter-pos` from the standard library. If we denote by  $f$  the function that is iterated over, and by  $x$  the sum which is the left hand side of the above inequation, and use lemma `iter-pos-plus` from the standard library, we have to prove that the  $x < ffx$ . This is done in lemma `odd-F-twice` which in turn requires that  $2n - 1$  be odd. This last thing is of course true, but to prove that in Coq is a bit technical and not adequate for explanation. The same is true for the proof of `odd-F-twice`; the reader should run the proofs in Coq if he is interested.  $\square$

The proofs of `Enested-right` and `Enested-right-succ` are analogous to the last two.

During all these proofs there are various simple equations and inequations to be solved. For that we use the special Coq tactics `ring` and `omega` which implement the decidability algorithm for Pressburger arithmetic. As this is only implemented for the types `nat` and `Z`, we have to inject into those types by using the following of our lemmas: `Zpos-Peq-inject`, `Zpos-Pgt-inject` and `Zpos-minus-distr`; and also the lemmas `Zpos-*` from the standard library.

3.2. **Fineness of E.** The goal is to prove the following:

Theorem `Efine` : `QISfine E`.

ie

Lemma `Efine'` :

```
forall epsilon : Q, (0 < epsilon)%Q ->
  exists k:positive,
    E (Einterval k) /\ (QIlength (Einterval k) < epsilon)%Q.
```

Thus, we have to find, for each  $\epsilon$  an interval in  $e$  with a length smaller than  $\epsilon$ . When writing this in a programming language, one would normally calculate the finite partial sum of  $e$  until the required precision is achieved and then stop the computation. Although it is clear that this process terminates, and one can only write terminating programs in Coq, the author found out that it is not so easy to write it efficiently in Coq, as one needs to formally show that the computation is well founded.

As calculating which term of the partial sum achieves the required precision is not computationally expensive in comparison to computing the partial sum itself, and as this calculation is easy to write down with structural recursion, we instead first do this and then compute the partial sum to the desired precision.

We write this in Coq like this:

Definition `estimate (eps:Q) : positive*positive := estimate' (Qden eps) 1 1 eps`.

where

```
Fixpoint estimate' (n k kfact:positive) (eps:Q) {struct n} : positive*positive :=
  if (Qlt_bool (1#kfact) eps)
  then (k,kfact)
  else
    let k':=k+1 in
    let kfact':=k*kfact in
    match n with
    | xH => (k,kfact)
    | xO n' => estimate' n' k' kfact' eps
    | xI n' => estimate' n' k' kfact' eps
    end.
```

This definition may seem a bit ad hoc, but it also seems to work. `estimate'` decreases structurally on the denominator of the precision  $\epsilon$ , calculating the factorial until the reciprocal value of the factorial becomes less than  $\epsilon$ . In computational test `estimate` always gave a sufficient estimate for the required  $\epsilon$  precision. We are convinced that it works even in cases not tested because for positive values the factorial increases faster than the exponential function and the structural recursion decreases at most exponentially.

What we have to prove now is that `(estimate  $\epsilon$ )-th Einterval's length is smaller than  $\epsilon$ :`

```
Lemma Einterval_length : forall (eps:Q), (0 < eps)%Q ->
  (QIlength (Einterval (estimate eps)) < eps)%Q.
```

or, with definitions unfolded:

```
Lemma Einterval_length' : forall (eps:Q), (0 < eps)%Q ->
  (1 # snd (fst (Eapprox (fst (estimate eps)))) < eps)%Q.
```

$\epsilon$	vm-compute in (estimate $\epsilon$ )	vm-compute in (Eapprox (estimate $\epsilon$ ))
$1/10^{12}$	0.008001	0.020001
$1/10^{24}$	0.008001	0.104006
$1/10^{90}$	0.048003	3.68823
$1/10^{180}$	0.136009	26.481655

TABLE 1. Performance of ‘Eapprox’ and ‘estimate’ in seconds

Remark now that `snd (fst (Eapprox N))` computes the factorial of  $N$ . The same is true for `snd (estimate eps)`. Therefore, we could finish the proof with the following two lemmas:

```
Lemma lemma1 : forall (eps:Q), (0 < eps)%Q ->
  (snd (estim eps)) = (snd (fst (Eapprox ((fst (estim eps)))))).
Lemma lemma2 : forall (eps:Q), (0 < eps)%Q ->
  ((1#(snd (estimate eps))) < eps)%Q.
```

Now, both of these lemmas do not have (formal) proofs. Lemma 2 should be possible to be finished. If one unfolds the definition of ‘estimate’, one sees that the function returns a value either in the if-then case when the goal of lemma is met, or in the if-else case when the counter  $n$  has reached 1. Possibly, for this second case, one might have to prove that 1 is never reached (except maybe in trivial cases).

As for lemma 1, one should prove that both sides of the equation compute the factorial function. As far as we can say at this moment, this should be done by induction on the denominator of  $\epsilon$  together with the following lemma:

```
Lemma lem1 : forall (p:positive)(z:Z), (z#p)<1 ->
  (fst (estimate' p 1 1 (z#p))) =
  (fst (estimate' (Psucc p) 1 1 (z#(Psucc p)))).
```

The proof of this remains unfinished. Thus, the ‘real number’  $E$  is not yet the real number  $e$ .

**3.3. Speed of computation.** A goal of this experiment has been to produce a fast way to compute Euler’s number. This had effect on the choice of types to work in (`positive` instead of `N`) and the complexity of algorithms.

We give a table of data of benchmarks carried out on our own 800 MHz Pentium III running the development version of Coq 8.1 on Debian Linux 4.0.

## APPENDIX A. ACTUAL DEFINITIONS AND LEMMAS

### A.1. Intervals with rational endpoints. Module QI

Implementing reals la Stolzenberg

QI.v – Rational intervals

Danko Ilik, svn revision: *\$Id: QI.v 81 2007-09-09 13:54:12Z danko \$*

Require Import *QArith*.

Open Scope *Q\_scope*.

Some additional constructions over `Q` not in the standard library

Section *Q\_extras*.

```
Definition Qlt_bool (x y:Q) :=
  match (x ?= y)%Q with
```

—  $Lt \Rightarrow true$   
 —  $_ \Rightarrow false$   
 end.

Definition  $Qle\_bool (x y:Q) :=$   
 match  $(x ?= y)\%Q$  with  
 —  $Lt \Rightarrow true$   
 —  $Eq \Rightarrow true$   
 —  $_ \Rightarrow false$   
 end.

Definition  $Qmax (x y:Q) :=$  if  $(Qlt\_bool x y)$  then  $y$  else  $x$ .

Definition  $Qmin (x y:Q) :=$  if  $(Qlt\_bool x y)$  then  $x$  else  $y$ .

Lemma  $Qlt\_bool\_refl : \forall x:Q, Qlt\_bool x x = false$ .

Lemma  $Qmax\_case : \forall (n m:Q) (P:Q \rightarrow Type), P n \rightarrow P m \rightarrow P (Qmax n m)$ .

Lemma  $Qmin\_case : \forall (n m:Q) (P:Q \rightarrow Type), P n \rightarrow P m \rightarrow P (Qmin n m)$ .

Lemma  $Qmax\_lub : \forall n m p:Q, n \leq p \rightarrow m \leq p \rightarrow Qmax n m \leq p$ .

Lemma  $Qmin\_glb : \forall n m p:Q, p \leq n \rightarrow p \leq m \rightarrow p \leq Qmin n m$ .

Lemma  $Qlt\_Qlt\_bool : \forall p q:Q, p < q \rightarrow Qlt\_bool p q = true$ .

Lemma  $Qmax\_determined : \forall p q:Q, p < q \rightarrow Qmax p q = q$ .

Lemma  $Qmin\_determined : \forall p q:Q, p < q \rightarrow Qmin p q = p$ .

Require Import *Bool*.

Lemma  $Qmin\_sym : \forall p q:Q, Qmin p q = Qmin q p$ .

Lemma  $Qmax\_sym : \forall p q:Q, Qmax p q = Qmax q p$ .

Lemma  $Qmax\_ub : \forall p q:Q, p \leq Qmax p q$ .

Lemma  $Qmax\_ub' : \forall p q:Q, q \leq Qmax p q$ .

Lemma  $Qmin\_lb : \forall p q:Q, Qmin p q \leq p$ .

Lemma  $Qmin\_lb' : \forall p q:Q, Qmin p q \leq q$ .

Definition  $Qabs (q:Q) := Qmake (Zabs (Qnum q)) (Qden q)$ .

End  $Q\_extras$ .

Hint Resolve  $Qmax\_ub$ .

Hint Resolve  $Qmax\_ub'$ .

Hint Resolve  $Qmin\_lb$ .

Hint Resolve  $Qmin\_lb'$ .

The theory of rational intervals

Record  $QI : Set :=$

$QImake \{ QIleft : Q;$   
 $QIright : Q;$   
 $QInonempty : (QIleft \leq QIright)\%Q \}$ .

Delimit Scope  $QI\_scope$  with  $QI$ .

Open Scope  $QI\_scope$ .

Definition  $QIlt (I J:QI) := (QIright I) < (QIleft J)$ .

Notation  $QIgt := (\text{fun } I J : QI \Rightarrow QIlt J I)$ .

Definition  $QIel (x:Q)(I:QI) := ((QIleft I)_i=x) \wedge (x_i=(QIright I))$ .  
 Definition  $QIsubset (I J:QI) := \forall x:Q, QIel x I \rightarrow QIel x J$ .  
 Definition  $QIeq (I J:QI) := (QIleft I == QIleft J) \wedge (QIright I == QIright J)$ .  
 Definition  $QIc (I J:QI) := (** consistency *)$   
 $(Qmax (QIleft I) (QIleft J)) \leq (Qmin (QIright I) (QIright J))$ .  
 Infix " $==$ " :=  $QIeq$  (at level 70, no associativity) :  $QI\_scope$ .  
 Infix " $<$ " :=  $QIlt$  :  $QI\_scope$ .  
 Infix " $_i$ " :=  $QIgt$  :  $QI\_scope$ .  
 Infix " $\sim$ " :=  $QIc$  (at level 70) :  $QI\_scope$ .  
 Definition  $QIlt\_bool (I J:QI) :=$   
   match  $((QIright I) ?= (QIleft J))\%Q$  with  
   —  $Lt \Rightarrow true$   
   —  $_ \Rightarrow false$   
   end.  
 Section  $Q\_intervals\_order\_and\_consistency$ .  
 Definition  $QIsingleton (x:Q) := QImake x x (Qle\_refl x) : QI$ .  
 Lemma  $QIlt\_trichotomy'$  :  $\forall I J:QI, \{I < J\} + \{I \sim J\} + \{J < I\}$ .  
 Lemma  $QIlt\_trichotomy$  :  $\forall I J:QI, (I < J) \vee (I \sim J) \vee (J < I)$ .  
 Hint  $Resolve QIlt\_trichotomy$ .  
 Definition  $QIlc (I J:QI) := I < J \vee I \sim J$ .  
 Infix " $<\sim$ " :=  $QIlc$  (at level 70) :  $QI\_scope$ .  
 Lemma  $QIlc\_not\_gt$  :  $\forall I J:QI, (I <\sim J) \leftrightarrow not (I_i J)$ .  
 Lemma  $QIc\_sym$  :  $\forall I J:QI, I \sim J \rightarrow J \sim I$ .  
 Hint  $Resolve QIc\_sym$ .  
 Lemma  $QIc\_lc$  :  $\forall I J:QI, I \sim J \leftrightarrow I <\sim J \wedge J <\sim I$ .  
 Lemma  $QIlt\_trans$  :  $\forall I J K:QI, I < J \rightarrow J < K \rightarrow I < K$ .  
 Hint  $Resolve QIlt\_trans$ .  
 Lemma  $QIlt\_lc\_lt$  :  $\forall I J K L:QI, I < J \rightarrow J <\sim K \rightarrow K < L \rightarrow I < L$ .  
 Lemma  $QImax\_nonempty$  :  $\forall I J:QI,$   
 $(Qmax (QIleft I) (QIleft J)) \leq (Qmax (QIright I) (QIright J))$ .  
 Lemma  $QImin\_nonempty$  :  $\forall I J:QI,$   
 $(Qmin (QIleft I) (QIleft J)) \leq (Qmin (QIright I) (QIright J))$ .  
 Definition  $QImax (I J:QI) := QImake$   
 $(Qmax (QIleft I) (QIleft J)) (Qmax (QIright I) (QIright J))$   
 $(QImax\_nonempty I J) : QI$ .  
 Definition  $QImin (I J:QI) := QImake$   
 $(Qmin (QIleft I) (QIleft J)) (Qmin (QIright I) (QIright J))$   
 $(QImin\_nonempty I J) : QI$ .  
 Lemma  $QImax\_el$  :  $\forall I J:QI, \forall x y:Q,$   
 $QIel x I \rightarrow QIel y J \rightarrow QIel (Qmax x y) (QImax I J)$ .  
 Lemma  $QImin\_el$  :  $\forall I J:QI, \forall x y:Q,$   
 $QIel x I \rightarrow QIel y J \rightarrow QIel (Qmin x y) (QImin I J)$ .

Lemma  $QImax\_c : \forall I J K L:QI, I \sim J \rightarrow K \sim L \rightarrow QImax I K == QImax J L$ .

Lemma  $QImax\_lt : \forall I J K:QI, (QImax I J) < K \leftrightarrow I < K \wedge J < K$ .

Lemma  $Qmax\_one : \forall p q:Q, Qmax p q = p \vee Qmax p q = q$ .

Lemma  $QIlt\_max : \forall I J K:QI, K < (QImax I J) \leftrightarrow K < I \vee K < J$ .

Definition  $QIle (I J:QI) := I < J \vee I == J$ .

Infix " $\leq$ " :=  $QIle$  (at level 70) :  $QI\_scope$ .

Lemma  $QIle\_max : \forall I J:QI, I \leq (QImax I J)$ .

Definition  $QIbetween (K I J:QI) := QImin I J < \sim K \wedge K < \sim QImax I J$ .

Lemma  $QIbetween\_lc : \forall K I J:QI,$

$QIbetween K I J \leftrightarrow (I < \sim K \wedge K < \sim J) \vee (J < \sim K \wedge K < \sim I)$ .

Lemma  $QIbetween\_transitivity : \forall K I J L:QI,$

$QIbetween K I J \rightarrow QIbetween L K J \rightarrow QIbetween L I J$ .

Definition  $QI\_sequence := nat \rightarrow QI$ .

Open Scope  $nat\_scope$ .

Lemma  $QIbetween\_countable\_transitivity : \forall I:QI\_sequence,$

$(\forall n:nat, QIbetween (I(n+2)) (I(n)) (I(n+1))) \rightarrow$

$\forall n m:nat, (m \leq n) \rightarrow QIbetween (I(m)) (I(n)) (I(n+1))$ .

Close Scope  $nat\_scope$ .

End  $Q\_intervals\_order\_and\_consistency$ .

Definition  $QIsum (I J:QI) :=$

$QImake (QIleft I + QIleft J) (QIright I + QIright J)$   
 $(Qplus\_le\_compat - - - (QInonempty I) (QInonempty J)) : QI$ .

Definition  $QIminus (I:QI) := QImake (- QIright I) (- QIleft I)$

$(Qopp\_le\_compat - - (QInonempty I)) : QI$ .

Definition  $QIdifference (I J:QI) := QIsum I (QIminus J) : QI$ .

Lemma  $Qmin\_lt\_max : \forall p q:Q, Qmin p q \leq Qmax p q$ .

Lemma  $QIproduct\_nonempty : \forall I J:QI,$

let  $r := QIleft I$  in

let  $s := QIright I$  in

let  $u := QIleft J$  in

let  $v := QIright J$  in

$(Qmin (Qmin (r \times u) (s \times v)) (Qmin (r \times v) (s \times u))) \leq$

$(Qmax (Qmax (r \times u) (s \times v)) (Qmax (r \times v) (s \times u)))$ .

Definition  $QIproduct (I J:QI) :=$

let  $r := QIleft I$  in

let  $s := QIright I$  in

let  $u := QIleft J$  in

let  $v := QIright J$  in

$QImake$

$(Qmin (Qmin (r \times u) (s \times v)) (Qmin (r \times v) (s \times u)))$

$(Qmax (Qmax (r \times u) (s \times v)) (Qmax (r \times v) (s \times u)))$

$(QIproduct\_nonempty I J)$

:  $QI$ .

Definition  $QIzero := QImake\ 0\ 0\ (Qle\_refl\ 0) : QI$ .

Definition  $QImagnitude\ (I:QI) := QImax\ I\ (QIminus\ I)$ .

Lemma  $QIquotient\_nonempty :$

$$\forall J:QI, \forall J\_correct:(not\ (QIzero \sim (QImagnitude\ J))), \\ ((Qinv\ (QIright\ J)) \leq (Qinv\ (QIleft\ J)))\%Q.$$

Definition  $QIquotient1\ (J:QI)\ (J\_correct: not\ (QIzero \sim (QImagnitude\ J))) := \\ QImake\ (Qinv\ (QIright\ J))\ (Qinv\ (QIleft\ J))\ (QIquotient\_nonempty\ -\ J\_correct)$ .

Definition  $QIquotient\ (I\ J:QI)\ (J\_correct: not\ (QIzero \sim (QImagnitude\ J))) := \\ QIproduct\ I\ (QIquotient1\ J\ J\_correct)$ .

Infix  $+$  :=  $QIsum : QI\_scope$ .

Notation  $-\ x$  :=  $(QIminus\ x) : QI\_scope$ .

Infix  $-$  :=  $QIdifference : QI\_scope$ .

Infix  $\times$  :=  $QIproduct : QI\_scope$ .

Notation  $/\ x$  :=  $(QIquotient1\ x) : QI\_scope$ .

Infix  $/$  :=  $QIquotient : QI\_scope$ .

Section  $Q\_intervals\_arithmetic\_properties$ .

Lemma  $QI\_minus\_minus : \forall J:QI, -\ -\ J = J$ .

Lemma  $QIquotient1\_correct : \forall J:QI, \forall J\_correct: not\ (QIzero \sim (QImagnitude\ J)), \\ not\ (QIzero \sim (QImagnitude\ (QIquotient1\ J\ J\_correct)))$ .

Lemma  $QIquotient1\_quotient1 : \forall J:QI, \forall J\_correct: not\ (QIzero \sim (QImagnitude\ J)),$

$$\text{let } qJ\_correct := QIquotient1\_correct\ J\ J\_correct \\ \text{in } (QIquotient1\ (QIquotient1\ J\ J\_correct)\ qJ\_correct) == J.$$

Lemma  $QIsum\_assoc : \forall I\ J\ K:QI, ((I+J)+K) == (I+(J+K))$ .

Hint  $Resolve\ QIsum\_assoc$ .

Lemma  $QIsum\_comm : \forall I\ J:QI, (I+J) == (J+I)$ .

Hint  $Resolve\ QIsum\_comm$ .

Lemma  $QIeq\_c : \forall I\ J:QI, I==J \rightarrow I \sim J$ .

Hint  $Resolve\ QIeq\_c$ .

Lemma  $QIsum\_assoc\_c : \forall I\ J\ K:QI, ((I+J)+K) \sim (I+(J+K))$ .

Hint  $Resolve\ QIsum\_assoc\_c$ .

Lemma  $QIsum\_comm\_c : \forall I\ J:QI, (I+J) \sim (J+I)$ .

Hint  $Resolve\ QIsum\_comm\_c$ .

Lemma  $Qmin\_eq : \forall p\ q\ r\ s:Q, (p==r)\%Q \rightarrow (q==s)\%Q \rightarrow (Qmin\ p\ q == \\ Qmin\ r\ s)\%Q$ .

Lemma  $Qmax\_eq : \forall p\ q\ r\ s:Q, (p==r)\%Q \rightarrow (q==s)\%Q \rightarrow (Qmax\ p\ q == \\ Qmax\ r\ s)\%Q$ .

Lemma  $QIproduct\_comm : \forall I\ J:QI, (I \times J) == (J \times I)$ .

Hint  $Resolve\ QIproduct\_comm$ .

Lemma  $QIproduct\_comm\_c : \forall I\ J:QI, (I \times J) \sim (J \times I)$ .

Lemma  $QIproduct\_assoc : \forall I\ J\ K:QI, ((I \times J) * K) == (I * (J \times K))$ .

Hint  $Resolve\ QIproduct\_assoc$ .

Lemma *QIproduct\_assoc\_c* :  $\forall I J K:QI, ((I \times J) * K) \sim (I * (J \times K))$ .

Lemma *QIel\_sum* :  $\forall I J:QI, \forall x y:Q, QIel\ x\ I \rightarrow QIel\ y\ J \rightarrow QIel\ (x+y)\ (I+J)$ .

Lemma *QIel\_difference* :  $\forall I J:QI, \forall x y:Q,$   
 $QIel\ x\ I \rightarrow QIel\ y\ J \rightarrow QIel\ (x-y)\ (I-J)$ .

Lemma *QIel\_product* :  $\forall I J:QI, \forall x y:Q,$   
 $QIel\ x\ I \rightarrow QIel\ y\ J \rightarrow QIel\ (x \times y)\ (I \times J)$ .

Lemma *QIel\_magnitude* :  $\forall I:QI, \forall x:Q,$   
 $QIel\ x\ I \rightarrow QIel\ (Qabs\ x)\ (QImagnitude\ I)$ .

Lemma *QIel\_quotient* :  $\forall I J:QI, \forall x y:Q, QIel\ x\ I \rightarrow QIel\ y\ J \rightarrow$   
 $\forall J\_correct:not\ (QIzero \sim (QImagnitude\ J)),$   
 $QIel\ (x/y)\ ((I/J)\ J\_correct)$ .

Theorem *QIsum\_c\_compat* :  $\forall I J K L:QI, I \sim K \rightarrow J \sim L \rightarrow (I+J) \sim (K+L)$ .

Theorem *QIproduct\_c\_compat* :  $\forall I J K L:QI, I \sim K \rightarrow J \sim L \rightarrow (I \times J) \sim (K \times L)$ .

Theorem *QImagnitude\_c\_compat* :  $\forall I K:QI, I \sim K \rightarrow (QImagnitude\ I) \sim (QImagnitude\ K)$ .

Theorem *QIc\_distrib* :  $\forall i j k I J K L:QI, i \sim I \rightarrow j \sim J \rightarrow k \sim K \rightarrow k \sim L \rightarrow$   
 $((i+j) * k) \sim ((I \times K) + (J \times L))$ .

Theorem *QIc\_minus\_zero* :  $\forall I J:QI, I \sim J \leftrightarrow (I-J) \sim QIzero$ .

Definition *QIlength* ( $I:QI$ ) :=  $(QIright\ I - QIleft\ I) \% Q$ .

Theorem *QIsum\_length* :  $\forall I J:QI, (QIlength\ (I+J)) == QIlength\ I + QIlength\ J) \% Q$ .

Theorem *QIdifference\_length* :  $\forall I J:QI, (QIlength\ (I-J)) == QIlength\ I - QIlength\ J) \% Q$ .

Theorem *QIquotient1\_length* :  $\forall J:QI,$   
 $\forall J\_correct:not\ (QIzero \sim (QImagnitude\ J)),$   
 $\forall c:Q, (0 < c) \% Q \rightarrow (c \leq QIleft\ J) \% Q \rightarrow$   
 $(QIlength\ (QIquotient1\ J\ J\_correct)) \leq (QIlength\ J) / (c \times c) \% Q$ .

Theorem *QImax\_length* :  $\forall I J:QI,$   
 $(QIlength\ (QImax\ I\ J)) == Qmax\ (QIlength\ I)\ (QIlength\ J)) \% Q$ .

Theorem *QImagnitude\_length* :  $\forall I:QI,$   
 $(QIlength\ (QImagnitude\ I)) \leq QIlength\ I) \% Q$ .

Lemma *QIlength\_monotone* ( $I J:QI$ ) :  $(QIle\ I\ J) \% QI \rightarrow (QIlength\ I \leq QIlength\ J) \% Q$ .

End *Q\_intervals\_arithmetic\_properties*.

## A.2. Real numbers. Module R

Require Import *QArith*.

Require Import *QI*.

Open Scope *Q\_scope*.

Open Scope *QI\_scope*.

Definition *QIS* := *QI* → Prop.

Definition *QISc* (*p q:QIS*) := ∀ *I J:QI*, *p I* → *q J* → *QIc I J*.

Definition *QISlt* (*p q:QIS*) := *exists2 I:QI*, *p I* & *exists2 J:QI*, *q J* & *QIlt I J*.

Definition *QISle* (*p q:QIS*) := ∀ *I J:QI*, *p I* → *q J* → *QIle I J*.

Definition *QISsum* (*p q:QIS*) (*K:QI*) :=  
*exists2 I:QI*, *p I* & *exists2 J:QI*, *q J* & *K = (I + J)*.

Definition *QISminus* (*p:QIS*) (*K:QI*) := *exists2 I:QI*, *p I* & *K = (-I)*.

Definition *QISdifference* (*p q:QIS*) (*K:QI*) :=  
*exists2 I:QI*, *p I* & *exists2 J:QI*, *q J* & *K = (I - J)*.

Definition *QISproduct* (*p q:QIS*) (*K:QI*) :=  
*exists2 I:QI*, *p I* & *exists2 J:QI*, *q J* & *K = (I × J)*.

Definition *QISMagnitude* (*p:QIS*) (*K:QI*) :=  
*exists2 I:QI*, *p I* & *K = QIMagnitude I*.

Definition *QISzero* (*K:QI*) := *K == QIzero*.

Definition *QISquotient1* (*p:QIS*) (*p\_gt\_zero:not (QISc QISzero p)*)  
(*K:QI*) := ∀ *I:QI*,  
 ∀ *I\_gt\_zero:not (QIzero ¬ (QIMagnitude I))*,  
*p I* → *K == (QIquotient1 I I\_gt\_zero)*.

Definition *QISquotient* (*q p:QIS*) (*p\_gt\_zero:not (QISc QISzero p)*)  
(*K:QI*) := ∀ *J I:QI*,  
 ∀ *I\_gt\_zero:not (QIzero ¬ (QIMagnitude I))*,  
*q J* → *p I* → *K == (QIquotient J I I\_gt\_zero)*.

Definition *QISmax* (*p q:QIS*) (*K:QI*) :=  
*exists2 I:QI*, *p I* & *exists2 J:QI*, *q J* & *K = QImax I J*.

Definition *QISmin* (*p q:QIS*) (*K:QI*) :=  
*exists2 I:QI*, *p I* & *exists2 J:QI*, *q J* & *K = QImin I J*.

Definition *QISfine* (*p:QIS*) := ∀ *epsilon:Q*, (0|*epsilon*)%*Q* →  
 ∃ *I:QI*, *p I* ∧ (*QIlength I* | *epsilon*)%*Q*.

Theorem *QISc\_sym* : ∀ *p q:QIS*, *QISc p q* → *QISc q p*.

Theorem *QISc\_lt\_le* : ∀ *p q:QIS*, *QISc p p* → *QISc q q* → *QISlt p q* → *QISle p q*.

Theorem *QISle\_not\_lt* : ∀ *p q:QIS*, *QISle p q* → *not (QISlt q p)*.

Theorem *QISnot\_lt\_le* : ∀ *p q:QIS*, *not (QISlt q p)* → *QISle p q*.

Theorem *QISle\_le\_c* : ∀ *p q:QIS*, *QISle p q* → *QISle q p* → *QISc p q*.

Theorem *QISlt\_lt\_lt* : ∀ *p w q:QIS*,  
*QISc w w* → *QISlt p w* → *QISlt w q* → *QISlt p q*.

Theorem *QISlt\_le\_lt* : ∀ *p w q:QIS*,  
*QISfine q* → *QISlt p w* → *QISle w q* → *QISlt p q*.

Theorem *QISle\_le\_le* :  $\forall p w q:QIS,$   
 $QISfine\ w \rightarrow QISle\ p\ w \rightarrow QISle\ w\ q \rightarrow QISle\ p\ q.$

Theorem *QISc\_c\_c* :  $\forall p w q:QIS,$   
 $QISfine\ w \rightarrow QISc\ p\ w \rightarrow QISc\ w\ q \rightarrow QISc\ p\ q.$

Theorem *QISlt\_eps* :  $\forall p q:QIS, QISlt\ q\ p \rightarrow$   
 $exists2\ eps:QIS, QISlt\ QISzero\ eps \ \&\ QISlt\ (QISsum\ q\ eps)\ p.$

Theorem *thm\_3\_25* :  $\forall p q:QIS,$   
 $(\forall\ eps:QIS, QISlt\ QISzero\ eps \rightarrow QISle\ p\ (QISsum\ q\ eps)) \rightarrow QISle\ p\ q.$

*Delimit Scope QIS\_scope with QIS.*

*Open Scope QIS\_scope.*

Infix "*i*" := *QISlt* : *QIS\_scope*.

Infix "*¬*" := *QISc* (at level 70) : *QIS\_scope*.

Infix "*≤*" := *QISle* (at level 70) : *QIS\_scope*.

Infix "*+*" := *QISsum* : *QIS\_scope*.

Notation "*- x*" := (*QISminus x*) : *QIS\_scope*.

Infix "*×*" := *QISproduct* : *QIS\_scope*.

Theorem *QIS\_distrib* :  $\forall w p q:QIS, (w \times (p + q)) \neg (w \times p + w \times q).$

Notation *cons* := (fun *p* : *QIS*  $\Rightarrow$  *QISc p p*).

Notation *fine* := (fun *p* : *QIS*  $\Rightarrow$  *QISfine p*).

Notation *max* := (fun *p q* : *QIS*  $\Rightarrow$  *QISmax p q*).

Notation *min* := (fun *p q* : *QIS*  $\Rightarrow$  *QISmin p q*).

Notation "*2*" := (*2#1*) : *Q\_scope*.

*Open Scope Q\_scope.*

Lemma *Qdiv\_lt\_compat* :  $\forall x y z:Q, 0iz \rightarrow xiy \rightarrow x/ziy/z.$

Lemma *Qdiv\_lt\_compat\_zero* :  $\forall x z:Q, 0iz \rightarrow 0ix \rightarrow 0ix/z.$

Lemma *Qplus\_lt\_compat* :

$\forall x y z t, xiy \rightarrow zit \rightarrow x+z\ i\ y+t.$

*Close Scope Q\_scope.*

Lemma *QISsum\_fine* :  $\forall p q : QIS, fine\ p \rightarrow fine\ q \rightarrow fine\ (p+q).$

Lemma *QISsum\_cons* :  $\forall p q : QIS, cons\ p \rightarrow cons\ q \rightarrow cons\ (p+q).$

Lemma *QISproduct\_cons* :  $\forall p q : QIS, cons\ p \rightarrow cons\ q \rightarrow cons\ (p \times q).$

Lemma *QISmax\_cons* :  $\forall p q : QIS, cons\ p \rightarrow cons\ q \rightarrow cons\ (QISmax\ p\ q).$

*Open Scope Q\_scope.*

Theorem *Qeq\_le* :  $\forall x y:Q, x==y \rightarrow x \leq y.$

*Close Scope Q\_scope.*

Lemma *QISmax\_fine* :  $\forall p q : QIS, fine\ p \rightarrow fine\ q \rightarrow fine\ (max\ p\ q).$

Lemma *QISmagnitude\_fine* :  $\forall p : QIS, fine\ p \rightarrow fine\ (QISmagnitude\ p).$

Definition *Q2QIS* (*q:Q*)(*I:QI*) := *I*=(*QImake q q* (*Qle\_refl q*)).

Lemma *Qdense* :  $\forall p:QIS, fine\ p \rightarrow \exists q:Q, p_i\ Q2QIS\ q.$

*Open Scope Q\_scope.*

Lemma *Qopp\_pos\_lt* :  $\forall A:Q, 0iA \rightarrow -A\ i\ A.$

Definition  $I\_oppA\_A (A:Q)(zIA:0jA) : QI$ .

*Close Scope  $Q\_scope$ .*

Lemma  $lemma1 : \forall I:QI, \forall (A : Q)(zero\_lt\_A:(0 j A)\%Q),$   
 $(QIlength I j A)\%Q \rightarrow QIsubset I (I\_oppA\_A A zero\_lt\_A)$ .

Theorem  $QISproduct\_fine : \forall p q : QIS, fine p \rightarrow fine q \rightarrow fine (p \times q)$ .

Record  $R : Type := Rmake \{$   
 $Rset : QIS;$   
 $Rfine : QISfine Rset;$   
 $Rconsistent : QISc Rset Rset \}$ .

Definition  $Req (p q:R) := QISc (Rset p) (Rset q)$ .

*Delimit Scope  $R\_scope$  with  $R$ .*

*Open Scope  $R\_scope$ .*

Notation  $QISgt := (\text{fun } p q : QIS \Rightarrow QISlt q p)$ .

Notation  $Rlt := (\text{fun } p q : R \Rightarrow QISlt (Rset p) (Rset q))$ .

Notation  $Rgt := (\text{fun } p q : R \Rightarrow QISlt (Rset q) (Rset p))$ .

Notation  $Rc := (\text{fun } p q : R \Rightarrow QISc (Rset p) (Rset q))$ .

Notation  $Rlc := (\text{fun } p q : R \Rightarrow QISle (Rset p) (Rset q))$ .

Notation  $Rle := (\text{fun } p q : R \Rightarrow (Rlt p q) \vee (Req p q))$ .

Infix " $=$ " :=  $Req$  (at level 70, no associativity) :  $R\_scope$ .

Infix " $i$ " :=  $Rlt$  :  $R\_scope$ .

Infix " $i$ " :=  $Rgt$  :  $R\_scope$ .

Infix " $\neg$ " :=  $Rc$  (at level 70) :  $R\_scope$ .

Infix " $i$ " :=  $Rlc$  (at level 70) :  $R\_scope$ .

Infix " $\leq$ " :=  $Rle$  (at level 70) :  $R\_scope$ .

Lemma  $Rmax\_consistent : \forall p q : R,$   
 $QISc (QISmax (Rset p) (Rset q)) (QISmax (Rset p) (Rset q))$ .

Lemma  $Rmax\_fine : \forall p q : R, QISfine (QISmax (Rset p) (Rset q))$ .

Notation  $Rmax :=$   
 $(\text{fun } p q : R \Rightarrow$   
 $Rmake (QISmax (Rset p) (Rset q)) (Rmax\_fine p q) (Rmax\_consistent p q))$ .

Lemma  $Rmin\_consistent : \forall p q : R,$   
 $QISc (QISmin (Rset p) (Rset q)) (QISmin (Rset p) (Rset q))$ .

Lemma  $Rmin\_fine : \forall p q : R, QISfine (QISmin (Rset p) (Rset q))$ .

Notation  $Rmin :=$   
 $(\text{fun } p q : R \Rightarrow$   
 $Rmake (QISmin (Rset p) (Rset q)) (Rmin\_fine p q) (Rmin\_consistent p q))$ .

Lemma  $Rle\_refl : \forall x, x \leq x$ .

Lemma  $Rsum\_fine : \forall p q : R, QISfine (QISsum (Rset p) (Rset q))$ .

Lemma  $Rsum\_consistent : \forall p q : R,$   
 $QISc (QISsum (Rset p) (Rset q)) (QISsum (Rset p) (Rset q))$ .

Notation  $Rsum := (\text{fun } p q : R \Rightarrow$   
 $Rmake (QISsum (Rset p) (Rset q)) (Rsum\_fine p q) (Rsum\_consistent p q))$ .

Infix "+" := *Rsum* : *R\_scope*.

Lemma *Rplus\_le\_compat* :

$$\forall x y z t, x \leq y \rightarrow z \leq t \rightarrow (x+z) \leq (y+t).$$

Lemma *Rminus\_fine* :  $\forall p : R, \text{QISfine } (\text{QISminus } (Rset p))$ .

Lemma *Rminus\_consistent* :  $\forall p : R,$

$$\text{QISc } (\text{QISminus } (Rset p)) (\text{QISminus } (Rset p)).$$

Notation *Rminus* := (fun *p* : *R* ⇒

$$Rmake (\text{QISminus } (Rset p)) (\text{Rminus\_fine } p) (\text{Rminus\_consistent } p)).$$

Notation "- *x*" := (*Rminus* *x*) : *R\_scope*.

Lemma *Ropp\_le\_compat* :  $\forall p q, p \leq q \rightarrow -q \leq -p$ .

Lemma *Rdifference\_fine* :  $\forall p q : R, \text{QISfine } (\text{QISdifference } (Rset p) (Rset q))$ .

Lemma *Rdifference\_consistent* :  $\forall p q : R,$

$$\text{QISc } (\text{QISdifference } (Rset p) (Rset q)) (\text{QISdifference } (Rset p) (Rset q)).$$

Notation *Rdifference* := (fun *p q* : *R* ⇒

$$Rmake (\text{QISdifference } (Rset p) (Rset q)) (\text{Rdifference\_fine } p q) (\text{Rdifference\_consistent } p q)).$$

Infix "-" := *Rdifference* : *R\_scope*.

Lemma *Rproduct\_fine* :  $\forall p q : R, \text{QISfine } (\text{QISproduct } (Rset p) (Rset q))$ .

Lemma *Rproduct\_consistent* :  $\forall p q : R,$

$$\text{QISc } (\text{QISproduct } (Rset p) (Rset q)) (\text{QISproduct } (Rset p) (Rset q)).$$

Notation *Rproduct* := (fun *p q* : *R* ⇒

$$Rmake (\text{QISproduct } (Rset p) (Rset q)) (\text{Rproduct\_fine } p q) (\text{Rproduct\_consistent } p q)).$$

Infix "×" := *Rproduct* : *R\_scope*.

Lemma *Rzero\_fine* : *QISfine* *QISzero*.

Lemma *Rzero\_consistent* : *QISc* *QISzero* *QISzero*.

Notation *Rzero* := (*Rmake* *QISzero* *Rzero\_fine* *Rzero\_consistent*).

Lemma *Rquotient1\_fine* :  $\forall p : R,$

$$\forall p\_correct: \text{not } (\text{QISc } \text{QISzero } (Rset p)), \\ \text{QISfine } (\text{QISquotient1 } (Rset p) p\_correct).$$

Lemma *Rquotient1\_consistent* :  $\forall p : R,$

$$\forall p\_correct: \text{not } (\text{QISc } \text{QISzero } (Rset p)), \\ \text{QISc } (\text{QISquotient1 } (Rset p) p\_correct) (\text{QISquotient1 } (Rset p) p\_correct).$$

Lemma *strip\_correct* :  $\forall (p : R),$

$$\text{not } (Rc \text{ Rzero } p) \rightarrow \text{not } (\text{QISc } \text{QISzero } (Rset p)).$$

Notation *Rquotient1* := (fun (*p* : *R*)(*p\_correct* : not (*Rc* *Rzero* *p*)) ⇒

$$\text{let } pc := \text{strip\_correct } p \text{ } p\_correct \\ \text{in } Rmake (\text{QISquotient1 } (Rset p) pc) \\ (\text{Rquotient1\_fine } p pc) \\ (\text{Rquotient1\_consistent } p pc)).$$

Notation "/" *x*" := (*Rquotient1* *x*) : *R\_scope*.

Lemma *Rquotient\_fine* :  $\forall (p q : R)(q\_correct : \text{not } (Rc \text{ Rzero } q)),$

*QISfine* (*QISquotient* (*Rset* *p*) (*Rset* *q*) (*strip\_correct* *q* *q\_correct*)).

Lemma *Rquotient\_consistent* :  $\forall (p\ q : R)(q\_correct : \text{not } (Rc\ Rzero\ q))$ ,  
*QISc* (*QISquotient* (*Rset* *p*) (*Rset* *q*) (*strip\_correct* *q* *q\_correct*))  
(*QISquotient* (*Rset* *p*) (*Rset* *q*) (*strip\_correct* *q* *q\_correct*)).

Notation *Rquotient* := (**fun** (*p* *q* : *R*)(*q\_correct* :  $\text{not } (Rc\ Rzero\ q)$ )=<sub>i</sub>  
**let** *qc*:=*strip\_correct* *q* *q\_correct*  
**in** *Rmake* (*QISquotient* (*Rset* *p*) (*Rset* *q*) *qc*) (*Rquotient\_fine* *p* *q* *qc*) (*Rquotient\_consistent* *p* *q* *qc*)).

Infix ”/” := *Rquotient* : *R\_scope*.

Lemma *Rmagnitude\_fine* :  $\forall p : R$ , *QISfine* (*QISMagnitude* (*Rset* *p*)).

Lemma *Rmagnitude\_consistent* :  $\forall p : R$ ,  
*QISc* (*QISMagnitude* (*Rset* *p*)) (*QISMagnitude* (*Rset* *p*)).

Notation *Rmagnitude* := (**fun** *p* : *R*  $\Rightarrow$   
*Rmake* (*QISMagnitude* (*Rset* *p*)) (*Rmagnitude\_fine* *p*) (*Rmagnitude\_consistent* *p*)).

### A.3. Defining Euler’s number. Module Euler

Require Import *NArith*.

Require Import *QArith*.

Require Import *QI*.

Require Import *R*.

Section *Euler\_number\_intervals*.

*Open Scope* *Q\_scope*.

Definition *Peven* (*n*:*positive*) : *bool* :=  
**match** *n* **with**  
— *xO \_*  $\Rightarrow$  *true*  
— *\_*  $\Rightarrow$  *false*  
**end**.

Definition *Eapprox* (*l*:*positive*) :=  
**let** *f* :=  
**fun** *arg*  $\Rightarrow$   
**let** *sum* := *snd* *arg* **in**  
**let** *n* := *fst* (*fst* *arg*) **in**  
**let** *nfact* := *snd* (*fst* *arg*) **in**  
**let** *sum'* := *Qred* (*sum* + **if** (*Peven* *n*) **then**  $-(1\#n\text{fact})$  **else**  
 $(1\#n\text{fact})$ )  
**in** (*Psucc* *n*, *Pmult* *n* *nfact*, *sum'*)  
**in** *iter\_pos* *l* (*positive*  $\times$  *positive*  $\times$  *Q*) *f* (*2%positive*, *1%positive*,  $(1\#1)$ ).

**End** *Euler\_number\_intervals*.

Section *Which\_term\_of\_the\_series\_achieves\_the\_require\_precision*.

*Open Scope* *positive\_scope*.

Fixpoint *estimate'* (*n* *k* *kfact*:*positive*) (*eps*:*Q*) {*struct* *n*} : *positive*  $\times$  *positive* :=  
**if** (*Qlt\_bool*  $(1\#k\text{fact})$  *eps*)  
**then** (*k*, *kfact*)  
**else**

```

let k':=k+1 in
  let kfact':=k×kfact in
  match n with
  — xH ⇒ (k,kfact)
  — xO n' ⇒ estimate' n' k' kfact' eps
  — xI n' ⇒ estimate' n' k' kfact' eps
end.

```

Definition *estimate* ( $eps:Q$ ) :  $positive \times positive := estimate' (Qden\ eps) 1 1\ eps$ .

```

Fixpoint estim' (n k kfact:positive) (eps:Q) {struct n} : positive×positive :=
  if (Qle_bool (1#kfact) eps)
  then (k,(k+1)*k×kfact)
  else
    let k':=k+1 in
      let kfact':=k×kfact in
      match n with
      — xH ⇒ (k,(k+1)*k×kfact)
      — xO n' ⇒ estim' n' k' kfact' eps
      — xI n' ⇒ estim' n' k' kfact' eps
end.

```

Definition *estim* ( $eps:Q$ ) :  $positive \times positive := estim' (Qden\ eps) 1 1\ eps$ .

End *Which\_term\_of\_the\_series\_achieves\_the\_require\_precision*.

```

Definition Einterval' (n:positive) : Q×Q :=
  let c := (Eapprox (2×n-1)) in
  (snd c, ((snd c)+(1#(snd (fst c))))%Q).

```

Lemma *Qineq1* :  $\forall (q:Q)(k:positive), (q \leq q+(1\#k))\%Q$ .

Theorem *Einterval'\_nonempty* :  $\forall n:positive,$   
 $(fst (Einterval' n) \leq snd (Einterval' n))\%Q$ .

```

Definition Einterval (n:positive) : QI :=
  let bounds:=Einterval' n in
  QImake (fst bounds) (snd bounds) (Einterval'_nonempty n).

```

```

Definition E : QIS :=
  fun I:QI ⇒
    ∃ n:positive, I = (Einterval n).

```

Definition *Pgt* ( $k n:positive$ ) :=  $((k?=n) Eq = Gt)\%positive$ .

Section *Solving\_equations\_and\_inequations\_in\_Z*.

*Open Scope Z\_scope*.

Lemma *Zpos\_Peq\_inject* :  $\forall x y:positive, Zpos\ x = Zpos\ y \rightarrow x = y$ .

Lemma *Zpos\_Pgt\_inject* :  $\forall x y:positive, Zpos\ x \text{ ; } Zpos\ y \rightarrow Pgt\ x\ y$ .

Lemma *Zpos\_minus\_distr* :  $\forall x y:positive, Pgt\ x\ y \rightarrow$   
 $Zpos\ (x-y) = Zpos\ x - Zpos\ y$ .

End *Solving\_equations\_and\_inequations\_in\_Z*.

*Open Scope positive\_scope*.

Definition *F* :=  $(\text{fun arg} : positive \times positive \times Q \Rightarrow$

$$\begin{aligned} & (Psucc (fst (fst arg)), (fst (fst arg) \times snd (fst arg)))\%positive, \\ & \quad Qred \\ & (snd arg + \\ & \quad (if Peven (fst (fst arg)) \\ & \quad \quad then - (1 \# snd (fst arg))\%Q \\ & \quad \quad else (1 \# snd (fst arg))\%Q)\%Q)). \end{aligned}$$

Lemma *odd\_F\_twice* :  $\forall A:positive \times positive \times Q,$   
 $Peven (fst (fst A)) = false \rightarrow$   
 $(snd A \text{ ; } snd (F (F A)))\%Q.$

Lemma *Peven\_plus\_two* :  $\forall k:positive, Peven k = Peven (Psucc (Psucc k)).$

Lemma *Enested\_left\_succ* :  $\forall n:positive,$   
 $(QIleft (Einterval n) \text{ ; } QIleft (Einterval (Psucc n)))\%Q.$

*Open Scope Q\_scope.*

Lemma *Enested\_left* :  $\forall k n:positive, Pgt k n \rightarrow (QIleft (Einterval n) \text{ ; } QIleft (Einterval k))\%Q.$

Lemma *Einterval'\_simplified* :  $\forall n:positive,$   
 $snd (Einterval' n) == snd (Eapprox (2 \times n)).$   
*Open Scope positive\_scope.*

Lemma *Podd\_succ* :  $\forall k:positive, Peven k = true \rightarrow$   
 $Peven (Psucc k) = false.$

Lemma *F\_pair* :  $\forall A:positive \times positive \times Q,$   
 $Peven (fst (fst A)) = true \rightarrow$   
 $(snd (F (F A)) \text{ ; } snd A)\%Q.$

Lemma *Enested\_right\_succ* :  $\forall n:positive,$   
 $QIright (Einterval (Psucc n)) \text{ ; } QIright (Einterval n).$   
*Open Scope positive\_scope.*

Lemma *Enested\_right* :  $\forall k n:positive, Pgt k n \rightarrow (QIright (Einterval k) \text{ ; } QIright (Einterval n))\%Q.$

Theorem *Econsistent* :  $QISC E E.$

Lemma *gt\_quotients* :  $\forall m n:positive, ((m?=n)\%positive Eq = Gt) \rightarrow ((1\#m)\text{ ; } (1\#n))\%Q.$

Lemma *Einterval\_in\_E* :  $\forall k:positive, E (Einterval k).$

Lemma *lemma1* :  $\forall (eps:Q), (0 \text{ ; } eps)\%Q \rightarrow$   
 $(snd (estim eps)) =$   
 $(snd (fst (Eapprox ((fst (estim eps)))))).$

*Eval compute in (Eapprox 1).*

Lemma *lem1* :  $\forall (p:positive)(p0:Z),$   
 $(fst (estimate' p 1 1 (p0\#p))) = (fst (estimate' (Psucc p) 1 1 (p0\#(Psucc p))))).$

*Eval compute in ((fun (p0:Z)(p:positive) => fst (estim' (Psucc p) 1 1 (p0 \# Psucc p))) 1 %Z 1).*

*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' p 1 1 (p0 # p))) 1%Z 1).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 1%Z 2).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' p 1 1 (p0 # p))) 1%Z 2).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 1%Z 3).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' p 1 1 (p0 # p))) 1%Z 4).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 1%Z 100).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' p 1 1 (p0 # p))) 1%Z 100).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 1%Z 100000).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' p 1 1 (p0 # p))) 1%Z 100000).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 1%Z 100000000).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' p 1 1 (p0 # p))) 1%Z 100000000).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 3%Z 1).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' p 1 1 (p0 # p))) 3%Z 1).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 3%Z 2).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' p 1 1 (p0 # p))) 3%Z 2).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 3%Z 3).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' p 1 1 (p0 # p))) 3%Z 3).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 3%Z 4).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' p 1 1 (p0 # p))) 3%Z 4).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 3%Z 100).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' p 1 1 (p0 # p))) 3%Z 100).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 3%Z 100000).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' p 1 1 (p0 # p))) 3%Z 100000).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 3%Z 100000000).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' p 1 1 (p0 # p))) 3%Z 100000000).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 5%Z 1).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' p 1 1 (p0 # p))) 5%Z 1).*  
*Eval compute in ((fun (p0:Z)(p:positive)=jfst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 5%Z 2).*

*Eval compute in ((fun (p0:Z)(p:positive)=ifst (estim' p 1 1 (p0 # p))) 5%Z 2).*  
*Eval compute in ((fun (p0:Z)(p:positive)=ifst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 5%Z 3).*  
*Eval compute in ((fun (p0:Z)(p:positive)=ifst (estim' p 1 1 (p0 # p))) 5%Z 3).*  
*Eval compute in ((fun (p0:Z)(p:positive)=ifst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 5%Z 4).*  
*Eval compute in ((fun (p0:Z)(p:positive)=ifst (estim' p 1 1 (p0 # p))) 5%Z 4).*  
*Eval compute in ((fun (p0:Z)(p:positive)=ifst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 5%Z 5).*  
*Eval compute in ((fun (p0:Z)(p:positive)=ifst (estim' p 1 1 (p0 # p))) 5%Z 5).*  
*Eval compute in ((fun (p0:Z)(p:positive)=ifst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 5%Z 6).*  
*Eval compute in ((fun (p0:Z)(p:positive)=ifst (estim' p 1 1 (p0 # p))) 5%Z 6).*  
*Eval compute in ((fun (p0:Z)(p:positive)=ifst (estim' (Psucc p) 1 1 (p0 # Psucc p))) 5%Z 7).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' p 1 1 (p0 # p))) 1%Z 1).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' p 1 1 (p0 # p))) 1%Z 1).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 1%Z 2).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' p 1 1 (p0 # p))) 1%Z 2).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 1%Z 3).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' p 1 1 (p0 # p))) 1%Z 4).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 1%Z 100).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' p 1 1 (p0 # p))) 1%Z 100).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 1%Z 100000).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' p 1 1 (p0 # p))) 1%Z 100000).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 1%Z 100000000).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' p 1 1 (p0 # p))) 1%Z 100000000).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 3%Z 1).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' p 1 1 (p0 # p))) 3%Z 1).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 3%Z 2).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' p 1 1 (p0 # p))) 3%Z 2).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 3%Z 3).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' p 1 1 (p0 # p))) 3%Z 3).*  
*Eval compute in ((fun (p0:Z)(p:positive)=isnd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 3%Z 4).*

*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' p 1 1 (p0 # p))) 3%Z 4).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 3%Z 100).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' p 1 1 (p0 # p))) 3%Z 100).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 3%Z 100000).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' p 1 1 (p0 # p))) 3%Z 100000).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 3%Z 100000000).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' p 1 1 (p0 # p))) 3%Z 100000000).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 5%Z 1).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' p 1 1 (p0 # p))) 5%Z 1).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 5%Z 2).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' p 1 1 (p0 # p))) 5%Z 2).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 5%Z 3).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' p 1 1 (p0 # p))) 5%Z 3).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 5%Z 4).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' p 1 1 (p0 # p))) 5%Z 4).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 5%Z 5).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' p 1 1 (p0 # p))) 5%Z 5).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 5%Z 6).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' p 1 1 (p0 # p))) 5%Z 6).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' (Psucc p) 1 1 (p0 # Psucc p))) 5%Z 7).*  
*Eval compute in ((fun (p0:Z)(p:positive)=i.snd (estim' p 1 1 (p0 # p))) 5%Z 7).*

*Eval compute in ((fun (eps:Q) => (snd (estim eps))= (snd (fst (Eapprox ((fst (estim eps))))))) (1#1)).*  
*Eval compute in ((fun (eps:Q) => (snd (estim eps))= (snd (fst (Eapprox ((fst (estim eps))))))) (1#2)).*  
*Eval compute in ((fun (eps:Q) => (snd (estim eps))= (snd (fst (Eapprox ((fst (estim eps))))))) (1#3)).*  
*Eval compute in ((fun (eps:Q) => (snd (estim eps))= (snd (fst (Eapprox ((fst (estim eps))))))) (5#1)).*  
*Eval compute in ((fun (eps:Q) => (snd (estim eps))= (snd (fst (Eapprox ((fst (estim eps))))))) (5#2)).*  
*Eval compute in ((fun (eps:Q) => (snd (estim eps))= (snd (fst (Eapprox ((fst (estim eps))))))) (5#3)).*  
*Eval compute in ((fun (eps:Q) => (snd (estim eps))= (snd (fst (Eapprox ((fst (estim eps))))))) (5#4)).*

*Eval compute* in ((fun (eps:Q) ⇒ (snd (estim eps))= (snd (fst (Eapprox ((fst (estim eps))))))) (5#5)).

*Eval compute* in ((fun (eps:Q) ⇒ (snd (estim eps))= (snd (fst (Eapprox ((fst (estim eps))))))) (5#6)).

*Eval compute* in ((fun (eps:Q) ⇒ (snd (estim eps))= (snd (fst (Eapprox ((fst (estim eps))))))) (5#7)).

**Lemma** *lemma2* : ∀ (eps:Q), (0 i eps)%Q → ((1#(snd (estimate eps))) i eps)%Q.

**Lemma** *Einterval\_length'* : ∀ (eps:Q), (0 i eps)%Q → (1 # snd (fst (Eapprox (fst (estimate eps)))) i eps)%Q.

**Lemma** *Einterval\_length'* : ∀ (eps:Q), (0 i eps)%Q → (1 # snd (fst (Eapprox (xO (estimate eps) - 1))) i eps)%Q.

**Lemma** *Einterval\_length* : ∀ (eps:Q), (0 i eps)%Q → (QIlength (Einterval (estimate eps)) i eps)%Q.

**Lemma** *Efine'* :  
 ∀ epsilon : Q, (0 i epsilon)%Q →  
 ∃ k:positive,  
 E (Einterval k) ∧ (QIlength (Einterval k) i epsilon)%Q.

**Theorem** *Efine* : QISfine E.

#### A.4. Intervals with real endpoints. Module RI

Require Import QI.

Require Import R.

Open Scope QI\_scope.

Open Scope R\_scope.

**Record** *RI* :Type := RImake {  
 Rleft : R;  
 Rright : R;  
 RInonempty : Rle Rleft Rright }.

*Delimit Scope* RI\_scope with RI.

*Open Scope* RI\_scope.

**Definition** *RIlt* (I J:RI) := (Rright I) i (Rleft J).

**Notation** *RIgt* := (fun I J : RI ⇒ RIlt J I).

**Definition** *RIel* (x:R)(I:RI) := ((Rleft I)i=x) ∧ (xi=(Rright I)).

**Definition** *RIsubset* (I J:RI) := ∀ x:R, RIel x I → RIel x J.

**Definition** *RIeq* (I J:RI) := RIsubset I J ∧ RIsubset J I.

**Definition** *RIc* (I J:RI) := (Rmax (Rleft I) (Rleft J)) ≤ (Rmin (Rright I) (Rright J)).

**Infix** "==" := *RIeq* (at level 70, no associativity) : RI\_scope.

**Infix** "i" := *RIlt* : RI\_scope.

**Infix** "j" := *RIgt* : RI\_scope.

**Infix** "¬" := *RIc* (at level 70) : RI\_scope.

**Section** *R\_intervals\_order\_and\_consistency*.

Hypotheses  $I J K L:RI$ .

Definition  $RI_{\text{singleton}} (x:R) := R_{\text{make}} x x (R_{\text{le\_refl}} x) : RI$ .

Lemma  $RI_{\text{lt\_trichotomy}}' : (I_i J) \vee (I \neg J) \vee (J_i I)$ .

Definition  $RI_{\text{lc}} (I J:RI) := I_i J \vee I \neg J$ .

Infix " $_i \sim$ " :=  $RI_{\text{lc}}$  (at level 70) :  $RI_{\text{scope}}$ .

Lemma  $RI_{\text{lc\_not\_gt}} : (I_i \sim J) \leftrightarrow \text{not } (I_i J)$ .

Lemma  $RI_{\text{lc\_lc}} : I \neg J \leftrightarrow I_i \sim J \wedge J_i \sim I$ .

Lemma  $RI_{\text{lt\_transitivity}} : I_i J \wedge J_i K \rightarrow I_i K$ .

Lemma  $RI_{\text{lt\_lc}} : I_i J \wedge J_i \sim K \rightarrow I_i K$ .

Lemma  $RI_{\text{lt\_lc}}' : I_i J \wedge J_i \sim K \wedge K_i L \rightarrow I_i L$ .

Lemma  $RI_{\text{max\_nonempty}} : \forall I J:RI,$

$(R_{\text{max}} (R_{\text{left}} I) (R_{\text{left}} J)) \leq (R_{\text{max}} (R_{\text{right}} I) (R_{\text{right}} J)).$

Lemma  $RI_{\text{min\_nonempty}} : \forall I J:RI,$

$(R_{\text{min}} (R_{\text{left}} I) (R_{\text{left}} J)) \leq (R_{\text{min}} (R_{\text{right}} I) (R_{\text{right}} J)).$

Definition  $RI_{\text{max}} (I J:RI) :=$

$R_{\text{make}} (R_{\text{max}} (R_{\text{left}} I) (R_{\text{left}} J)) (R_{\text{max}} (R_{\text{right}} I) (R_{\text{right}} J))$

$(RI_{\text{max\_nonempty}} I J) : RI$ .

Definition  $RI_{\text{min}} (I J:RI) :=$

$R_{\text{make}} (R_{\text{min}} (R_{\text{left}} I) (R_{\text{left}} J)) (R_{\text{min}} (R_{\text{right}} I) (R_{\text{right}} J))$

$(RI_{\text{min\_nonempty}} I J) : RI$ .

Lemma  $RI_{\text{max\_el}} : \forall x y:R,$

$RI_{\text{el}} x I \rightarrow RI_{\text{el}} y J \rightarrow RI_{\text{el}} (R_{\text{max}} x y) (RI_{\text{max}} I J).$

Lemma  $RI_{\text{min\_el}} : \forall x y:R,$

$RI_{\text{el}} x I \rightarrow RI_{\text{el}} y J \rightarrow RI_{\text{el}} (R_{\text{min}} x y) (RI_{\text{min}} I J).$

Lemma  $RI_{\text{max\_c}} : I \neg J \rightarrow K \neg L \rightarrow RI_{\text{max}} I K == RI_{\text{max}} J L$ .

Lemma  $RI_{\text{c\_max\_min}} :$

$(RI_{\text{max}} I J) \sim J \leftrightarrow (RI_{\text{min}} I J) \sim I \leftrightarrow I_i \sim J$ .

Lemma  $RI_{\text{subset\_max\_min}} : RI_{\text{subset}} (RI_{\text{max}} I J) I \vee RI_{\text{subset}} (RI_{\text{max}} I J)$

$J$ .

Lemma  $RI_{\text{max\_lt}} : (RI_{\text{max}} I J)_i K \leftrightarrow I_i K \wedge J_i K$ .

Lemma  $RI_{\text{lt\_max}} : K_i (RI_{\text{max}} I J) \leftrightarrow K_i I \vee K_i J$ .

Definition  $RI_{\text{le}} (I J:RI) := I_i J \vee I == J$ .

Infix " $\leq$ " :=  $RI_{\text{le}}$  (at level 70) :  $RI_{\text{scope}}$ .

Lemma  $RI_{\text{le\_max}} : I \leq (RI_{\text{max}} I J)$ .

Definition  $RI_{\text{between}} (K I J:RI) := RI_{\text{min}} I J \sim K \wedge K \sim RI_{\text{max}} I J$ .

Lemma  $RI_{\text{between\_lc}} : RI_{\text{between}} K I J \leftrightarrow (I_i \sim K \wedge K_i \sim J) \vee (J_i \sim K \wedge K_i \sim I)$ .

Lemma  $RI_{\text{between\_transitivity}} :$

$RI_{\text{between}} K I J \rightarrow RI_{\text{between}} L K J \rightarrow RI_{\text{between}} L I J$ .

Definition  $RI_{\text{sequence}} := \text{nat} \rightarrow RI$ .

Open Scope  $\text{nat\_scope}$ .

Lemma *RIbetween\_countable\_transitivity* :  $\forall I:RI\_sequence,$   
 $(\forall n:nat, RIbetween (I(n+2)) (I(n)) (I(n+1))) \rightarrow$   
 $\forall n m:nat, (m \leq n) \rightarrow RIbetween (I(m)) (I(n)) (I(n+1)).$   
*Close Scope nat\_scope.*

End *R\_intervals\_order\_and\_consistency*.

Definition *RIsum* (*I J:RI*) :=  
 $RImake (RIleft I + RIleft J) (RIright I + RIright J)$   
 $(Rplus\_le\_compat \_ \_ \_ (RInonempty I) (RInonempty J)) : RI.$

Definition *RIminus* (*I:RI*) :=  $RImake (- RIright I) (- RIleft I)$   
 $(Ropp\_le\_compat \_ \_ (RInonempty I)) : RI.$

Definition *RIdifference* (*I J:RI*) :=  $RIsum I (RIminus J) : RI.$

Lemma *RIproduct\_nonempty* :  $\forall I J:RI,$   
 let  $r:=RIleft I$  in  
 let  $s:=RIright I$  in  
 let  $u:=RIleft J$  in  
 let  $v:=RIright J$  in  
 $(Rmin (r \times u) (Rmin (r \times v) (Rmin (s \times u) (s \times v)))) \leq$   
 $(Rmax (r \times u) (Rmax (r \times v) (Rmax (s \times u) (s \times v))))).$

Definition *RIproduct* (*I J:RI*) :=  
 let  $r:=RIleft I$  in  
 let  $s:=RIright I$  in  
 let  $u:=RIleft J$  in  
 let  $v:=RIright J$  in  
 $RImake$   
 $(Rmin (r \times u) (Rmin (r \times v) (Rmin (s \times u) (s \times v))))$   
 $(Rmax (r \times u) (Rmax (r \times v) (Rmax (s \times u) (s \times v))))$   
 $(RIproduct\_nonempty I J)$   
 : *RI*.

Definition *RIzero* :=  $RImake Rzero Rzero (Rle\_refl Rzero) : RI.$

Definition *RImagnitude* (*I:RI*) :=  $RImax I (RImax (RIminus I) RIzero).$

Lemma *RIleft\_correct* :  $\forall J:RI,$   
 $not (RIc RIzero (RImagnitude J)) \rightarrow not (Rc Rzero (RIleft J)).$

Lemma *RIright\_correct* :  $\forall J:RI,$   
 $not (RIc RIzero (RImagnitude J)) \rightarrow not (Rc Rzero (RIright J)).$

Lemma *RIquotient\_nonempty* :  
 $\forall J:RI, \forall J\_correct: not (RIc RIzero (RImagnitude J)),$   
 $((Rquotient1 (RIright J) (RIright\_correct J J\_correct))$   
 $\leq (Rquotient1 (RIleft J) (RIleft\_correct J J\_correct))) \% R.$

Definition *RIquotient1* (*J:RI*) (*J\\_correct: not (RIc RIzero (RImagnitude J))*) :=  
 $RImake$   
 $(Rquotient1 (RIright J) (RIright\_correct J J\_correct))$   
 $(Rquotient1 (RIleft J) (RIleft\_correct J J\_correct))$   
 $(RIquotient\_nonempty \_ J\_correct).$

Definition *RIquotient* (*I J:RI*) (*J\\_correct: not (RIc RIzero (RImagnitude J))*) :=

*R*product *I* (*R*Iquotient1 *J* *J*\_correct).

Infix "+" := *R*Isum : *RI*\_scope.

Notation "- *x*" := (*R*Iminus *x*) : *RI*\_scope.

Infix "-" := *R*Idifference : *RI*\_scope.

Infix "×" := *R*Iproduct : *RI*\_scope.

Notation "*x* / *y*" := (*R*Iquotient1 *x*) : *RI*\_scope.

Infix "/" := *R*Iquotient : *RI*\_scope.

Section *R\_intervals\_arithmetic\_properties*.

Hypotheses *I J K*:*RI*.

Lemma *RI\_minus\_minus* : - - *J* = *J*.

Lemma *RIquotient1\_correct* :  $\forall J\_correct: \text{not } (RIzero \neg (RI\text{magnitude } J)),$   
 $\text{not } (RIzero \neg (RI\text{magnitude } (RIquotient1 J J\_correct)))$ .

Lemma *RIquotient1\_quotient1*:  $\forall J\_correct: \text{not } (RIzero \neg (RI\text{magnitude } J)),$   
 $\text{let } qJ\_correct := RIquotient1\_correct J\_correct$   
 $\text{in } (RIquotient1 (RIquotient1 J J\_correct) qJ\_correct) == J$ .

Lemma *RIsum\_associative* :  $((I+J)+K) == (I+(J+K))$ .

Lemma *RIsum\_commutative* :  $(I+J) == (J+I)$ .

Lemma *RIsum\_associative\_c* :  $((I+J)+K) \neg (I+(J+K))$ .

Lemma *RIsum\_commutative\_c* :  $(I+J) \neg (J+I)$ .

Lemma *RIproduct\_associative* :  $((I \times J) * K) == (I * (J \times K))$ .

Lemma *RIproduct\_commutative* :  $(I \times J) == (J \times I)$ .

Lemma *RIel\_sum* :  $\forall x y:R, RIel x I \rightarrow RIel y J \rightarrow RIel (x+y) (I+J)$ .

Lemma *RIel\_difference* :  $\forall x y:R,$   
 $RIel x I \rightarrow RIel y J \rightarrow RIel (x-y) (I-J)$ .

Lemma *RIel\_product* :  $\forall x y:R,$   
 $RIel x I \rightarrow RIel y J \rightarrow RIel (x \times y) (I \times J)$ .

Lemma *RIel\_magnitude* :  $\forall x:R,$   
 $RIel x I \rightarrow RIel (R\text{magnitude } x) (RI\text{magnitude } I)$ .

Lemma *RIel\_quotient* :  $\forall x y:R, RIel x I \rightarrow RIel y J \rightarrow$   
 $\forall y\_correct: \text{not } (Rc Rzero y),$   
 $\forall J\_correct: \text{not } (RIc RIzero (RI\text{magnitude } J)),$   
 $RIel (Rquotient x y y\_correct) (RIquotient I J J\_correct)$ .

Definition *RIlength* (*I*:*RI*) := (*R*Iright *I* - *R*Ileft *I*)%*R*.

Lemma *RIlength\_monotone* (*I J*:*RI*) : (*R*Ile *I J*)%*RI*  $\rightarrow$  (*RIlength* *I*  $\leq$  *RIlength* *J*)%*R*.

End *R\_intervals\_arithmetic\_properties*.

**A.5. Sets of intervals with real endpoints.** Module RISRequire Import *QI*.Require Import *R*.Require Import *RI*.*Open Scope R\_scope.**Open Scope RI\_scope.*Definition *RIS* := *RI* → Prop.Definition *RISc* (*p q:RIS*) :=  $\forall I J:RI, p I \rightarrow q J \rightarrow RIc I J$ .Definition *RISlt* (*p q:RIS*) :=  $\forall I J:RI, p I \rightarrow q J \rightarrow RIlI I J$ .Definition *RISlc* (*p q:RIS*) :=  $\forall I J:RI, p I \rightarrow q J \rightarrow RIlc I J$ .Definition *RISsum* (*p q:RIS*) (*K:RI*) :=  $\forall I J:RI,$   
 $p I \rightarrow q J \rightarrow K == (I + J)$ .Definition *RISdifference* (*p q:RIS*) (*K:RI*) :=  
 $\forall I J:RI, p I \rightarrow q J \rightarrow K == (I - J)$ .Definition *RISproduct* (*p q:RIS*) (*K:RI*) :=  $\forall I J:RI,$   
 $p I \rightarrow q J \rightarrow K == (I \times J)$ .Definition *RISMagnitude* (*p:RIS*) (*K:RI*) :=  $\forall I:RI,$   
 $p I \rightarrow K == RIMagnitude I$ .Definition *RISzero* (*K:RI*) :=  $K == RIzero$ .Definition *RISfine* (*p:RIS*) :=  $\forall \epsilon:R, (Rzero \downarrow \epsilon) \% R \rightarrow$   
 $\exists I:RI, p I \wedge (\epsilon \downarrow Rlength I) \% R$ .**A.6. Limits and completeness.** Module LimitsRequire Import *QArith*.Require Import *QI*.Require Import *R*.Require Import *RI*.Require Import *RIS*.Record *NL* : Type := *NLmake* {*NLset* : *RIS*;*NLfine* : *RISfine* *NLset*;*NLconsistent* : *RISc* *NLset* *NLset* }.Definition *RIinQI* (*L:RI*)(*I:QI*) := $\forall (l:R)(J:QI), RIel l L \rightarrow (Rset l) J \rightarrow QIsubset J I$ .Definition *Lim* (*Lambda:NL*)(*I:QI*) := $\exists K:RI, (NLset Lambda) K \wedge RIinQI K I$ .Check *Lim*.Theorem *completeness* :  $\forall Lambda:NL,$ *QISfine* (*Lim Lambda*)  $\wedge$  *QISc* (*Lim Lambda*) (*Lim Lambda*).

## A.7. Inverse function theorem. Module IFTheorem

Require Import *QArith*.

Require Import *QI*.

Require Import *R*.

Require Import *RI*.

Require Import *RIS*.

Require Import *Limits*.

Open Scope *R\_scope*.

Lemma *notzero1* :  $\forall x:R, Rzero \mid x \rightarrow \text{not } (Rzero \neg x)$ .

Lemma *notzero2* :  $\forall x y:R, \text{not } (Rzero \neg x) \rightarrow x \leq y \rightarrow \text{not } (Rzero \neg y)$ .

Section *IFT*.

Variables  $(f:R \rightarrow R)(a b L K:R)(nz1:Rzero \mid L)(nz2:L \leq K)(a\_le\_b:Rle a b)$ .

Fixpoint *I* ( $n':nat$ ) {*struct n'*} : *RI* :=

  match *n'* with

    — (*S n*)  $\Rightarrow RImake a b a\_le\_b$       — *O*  $\Rightarrow RImake a b a\_le\_b$

  end.

Theorem *inverse* :

$(\forall x y:R, a \leq x \rightarrow x \leq y \rightarrow y \leq b \rightarrow$

$(L^*(y-x)) \mid = (f y - f x) \rightarrow (f y - f x) \mid = (K^*(y-x)) \rightarrow$

$\exists g:R \rightarrow R, \forall z w:R, (f a) \leq z \rightarrow z \leq w \rightarrow w \leq (f b) \rightarrow$

$(f (g z)) == z$

$\wedge (Rquotient (w-z) K (\text{notzero2 } L K (\text{notzero1 } L nz1) nz2)) \leq g w - g z$

$\wedge g w - g z \leq Rquotient (w-z) L (\text{notzero1 } L nz1))$ .

End *IFT*.

Section *define\_one\_and\_two\_and\_three*.

Definition *QIone* := *QImake* 1 1 (*Qle\_refl* 1) : *QI*.

Open Scope *Q\_scope*.

Definition *QItwo* := *QImake* (2#1)%*Q* (2#1)%*Q* (*Qle\_refl* (2#1)%*Q*) : *QI*.

Definition *QIthree* := *QImake* (3#1)%*Q* (3#1)%*Q* (*Qle\_refl* (3#1)%*Q*) : *QI*.

Close Scope *Q\_scope*.

Definition *QISone* (*K:QI*) := (*K* == *QIone*)%*QI*.

Definition *QIStwo* (*K:QI*) := (*K* == *QItwo*)%*QI*.

Definition *QISthree* (*K:QI*) := (*K* == *QIthree*)%*QI*.

Lemma *QISone\_fine* : *QISfine* *QISone*.

Lemma *QISone\_consistent* : *QISc* *QISone* *QISone*.

Lemma *QIStwo\_fine* : *QISfine* *QIStwo*.

Lemma *QIStwo\_consistent* : *QISc* *QIStwo* *QIStwo*.

Lemma *QISthree\_fine* : *QISfine* *QISthree*.

Lemma *QISthree\_consistent* : *QISc* *QISthree* *QISthree*.

Definition *Rone* := (*Rmake* *QISone* *QISone\_fine* *QISone\_consistent*).

Definition *Rtwo* := (*Rmake* *QIStwo* *QIStwo\_fine* *QIStwo\_consistent*).

Definition *Rthree* := (*Rmake* *QISthree* *QISthree\_fine* *QISthree\_consistent*).

End *define\_one\_and\_two\_and\_three*.

Definition *Rsquare* ( $x:R$ ) : *R* := *Rproduct* *x x*.

Section *Rsquare\_bounds*.

Variables  $L K a b x y : R$ .

Hypotheses  $(nz1 : Rzero) (L) (nz2 : L \leq K)$ .

Hypotheses  $(le\_ax : a \leq x) (le\_xy : x \leq y) (le\_yb : y \leq b)$ .

Lemma *Rsquare\_upper\_bound* :  $(L^*(y-x))_i = (Rsquare\ y - Rsquare\ x)$ .

Lemma *Rsquare\_lower\_bound* :  $(Rsquare\ y - Rsquare\ x)_i = (K^*(y-x))$ .

End *Rsquare\_bounds*.

Definition *Rsqrt* ( $x : R$ ) := *inverse Rsquare*.

Check *Rsqrt*.

**A.8. Proposed axiomatisation of the theories of intervals.** Module XField

Definition *associative* ( $A : Type$ ) ( $op : A \rightarrow A \rightarrow A$ ) :=

$\forall x\ y\ z : A, op\ (op\ x\ y)\ z = op\ x\ (op\ y\ z)$ .

Definition *commutative* ( $A : Type$ ) ( $op : A \rightarrow A \rightarrow A$ ) :=

$\forall x\ y : A, op\ x\ y = op\ y\ x$ .

Definition *trichotomous* ( $A : Type$ ) ( $R : A \rightarrow A \rightarrow Prop$ ) :=

$\forall x\ y : A, R\ x\ y \vee x = y \vee R\ y\ x$ .

Definition *relation* ( $A : Type$ ) :=  $A \rightarrow A \rightarrow Prop$ .

Definition *reflexive* ( $A : Type$ ) ( $R : relation\ A$ ) :=  $\forall x : A, R\ x\ x$ .

Definition *transitive* ( $A : Type$ ) ( $R : relation\ A$ ) :=

$\forall x\ y\ z : A, R\ x\ y \rightarrow R\ y\ z \rightarrow R\ x\ z$ .

Definition *symmetric* ( $A : Type$ ) ( $R : relation\ A$ ) :=  $\forall x\ y : A, R\ x\ y \rightarrow R\ y\ x$ .

Record *interval* ( $X : Set$ ) ( $le : X \rightarrow X \rightarrow Prop$ ) : Set :=

*interval\_make* {  
*interval\_left* :  $X$ ;  
*interval\_right* :  $X$ ;  
*interval\_nonempty* :  $le\ interval\_left\ interval\_right$   
}.

Record *I* ( $grnd : Set$ ) ( $le : grnd \rightarrow grnd \rightarrow Prop$ ) : Type := *Imake* {

*Icar* := *interval le*;  
*Iplus* :  $Icar \rightarrow Icar \rightarrow Icar$ ;  
*Imult* :  $Icar \rightarrow Icar \rightarrow Icar$ ;  
*Izero* :  $Icar$ ;  
*Ione* :  $Icar$ ;  
*Iopp* :  $Icar \rightarrow Icar$ ;  
*Iinv* :  $Icar \rightarrow Icar$ ;  
*Ic* :  $Icar \rightarrow Icar \rightarrow Prop$ ;     *Iplus\_assoc* : *associative Iplus*;  
*Imult\_assoc* : *associative Imult*;  
   *Iplus\_comm* : *commutative Iplus*;  
   *Imult\_comm* : *commutative Imult*;  
*Iplus\_0\_l* :  $\forall x : Icar, Ic\ (Iplus\ Izero\ x)\ x$ ;  
*Iplus\_0\_r* :  $\forall x : Icar, Ic\ (Iplus\ x\ Izero)\ x$ ;  
*Imult\_0\_l* :  $\forall x : Icar, Ic\ (Imult\ Ione\ x)\ x$ ;  
*Imult\_0\_r* :  $\forall x : Icar, Ic\ (Imult\ x\ Ione)\ x$ ;  
*Iplus\_opp\_r* :  $\forall x : Icar, Ic\ (Iplus\ x\ (Iopp\ x))\ (Izero)$ ;  
*Imult\_inv\_r* :  $\forall x : Icar, \sim(Ic\ x\ Izero) \rightarrow Ic\ (Imult\ x\ (Iinv\ x))\ Ione$ ;

```

    Imult_plus_distr_l :  $\forall x x' y y' z z' z''$ ,
    Ic  $x x' \rightarrow Ic y y' \rightarrow Ic z z' \rightarrow Ic z z'' \rightarrow$ 
    Ic (Imult (Iplus  $x y$ )  $z$ ) (Iplus (Imult  $x' z'$ ) (Imult  $y' z''$ ));
    Ilt : Icar  $\rightarrow Icar \rightarrow \mathbf{Prop}$ ;
    Ilc := fun (x y:Icar)  $\Rightarrow Ilt x y \vee Ic x y$ ;
    Isup : Icar  $\rightarrow Icar \rightarrow Icar$ ;
    Iinf : Icar  $\rightarrow Icar \rightarrow Icar$ ;
    Ilt_trans : transitive Ilt;
    Ilt_trich :  $\forall x y:Icar, Ilt x y \vee Ic x y \vee Ilt y x$ ;
    Isup_lub :  $\forall x y z:Icar, Ilc x z \rightarrow Ilc y z \rightarrow Ilc (Isup x y) z$ ;
    Iinf_glb :  $\forall x y z:Icar, Ilc x y \rightarrow Ilc x z \rightarrow Ilc x (Iinf y z)$ ;
    Ic_refl : reflexive Ic;
    Ic_sym : symmetric Ic
  }.

```

Definition *interval\_set* ( $X:\mathbf{Set}$ )( $le:X \rightarrow X \rightarrow \mathbf{Prop}$ ) :=  
 (*interval* *le*)  $\rightarrow \mathbf{Prop}$ .

Check *interval\_set*.

Check *Ic*.

Check *interval*.

Definition *consistent* ( $X:\mathbf{Set}$ )( $le:X \rightarrow X \rightarrow \mathbf{Prop}$ )( $TI:I le$ )( $p:interval\_set le$ )  
 :=  $\forall I J:interval le, p I \rightarrow p J \rightarrow Ic (i:=TI) I J$ .

Check *consistent*.

Check *Izero*.

Definition *fine* ( $X:\mathbf{Set}$ )( $zero:X$ )( $minus:X \rightarrow X \rightarrow X$ )( $le:X \rightarrow X \rightarrow \mathbf{Prop}$ )( $TI:I le$ )( $p:interval\_set le$ )

```

  :=  $\forall epsilon:X, le zero epsilon \rightarrow$ 
      $\exists I:(Icar TI), p I$ 
      $\wedge le epsilon (minus (interval\_right I) (interval\_left I))$ .

```

Record *N* ( $grnd:\mathbf{Set}$ )( $zero:grnd$ )( $minus:grnd \rightarrow grnd \rightarrow grnd$ )( $le:grnd \rightarrow grnd \rightarrow \mathbf{Prop}$ )( $grndI:I le$ ) :  $\mathbf{Type} := \mathbf{Nmake}$  {

```

  Ncar := interval_set le;
  Nconsistent := consistent grndI;
  Nfine := fine zero minus grndI;
  Nplus : Ncar  $\rightarrow Ncar \rightarrow Ncar$ ;
  Nmult : Ncar  $\rightarrow Ncar \rightarrow Ncar$ ;
  Nzero : Ncar;
  None : Ncar;
  Nopp : Ncar  $\rightarrow Ncar$ ;
  Ninv : Ncar  $\rightarrow Ncar$ ;
  Nc : Ncar  $\rightarrow Ncar \rightarrow \mathbf{Prop}$ ;    Nplus_assoc : associative Nplus;
  Nmult_assoc : associative Nmult;
  Nplus_comm : commutative Nplus;
  Nmult_comm : commutative Nmult;
  Nplus_0_l :  $\forall x:Ncar, Nc (Nplus Nzero x) x$ ;
  Nplus_0_r :  $\forall x:Ncar, Nc (Nplus x Nzero) x$ ;
  Nmult_0_l :  $\forall x:Ncar, Nc (Nmult None x) x$ ;
  Nmult_0_r :  $\forall x:Ncar, Nc (Nmult x None) x$ ;

```

$$\begin{aligned}
&Nplus\_opp\_r : \forall x:Ncar, Nc (Nplus x (Nopp x)) (Nzero); \\
&Nmult\_inv\_r : \forall x:Ncar, \sim(Nc x Nzero) \rightarrow Nc (Nmult x (Ninv x)) None; \\
&Nmult\_plus\_distr\_l : \forall x x' y y' z z' z'', \\
&\quad Nc x x' \rightarrow Nc y y' \rightarrow Nc z z' \rightarrow Nc z z'' \rightarrow \\
&\quad Nc (Nmult (Nplus x y) z) (Nplus (Nmult x' z') (Nmult y' z'')); \\
&Nlt : Ncar \rightarrow Ncar \rightarrow \mathbf{Prop}; \\
&Nlc := \mathbf{fun} (x y:Ncar) \Rightarrow Nlt x y \vee Nc x y; \\
&Nsup : Ncar \rightarrow Ncar \rightarrow Ncar; \\
&Ninf : Ncar \rightarrow Ncar \rightarrow Ncar; \\
&Nlt\_trans : \mathit{transitive} \mathit{lt}; \\
&Nlt\_trich : \forall x y:Ncar, Nlt x y \vee Nc x y \vee Nlt y x; \\
&Nsup\_lub : \forall x y z:Ncar, Nlc x z \rightarrow Nlc y z \rightarrow Nlc (Nsup x y) z; \\
&Ninf\_glb : \forall x y z:Ncar, Nlc x y \rightarrow Nlc x z \rightarrow Nlc x (Ninf y z); \\
&\quad Nc\_refl : \mathit{reflexive} \mathit{Nc}; \\
&Nc\_sym : \mathit{symmetric} \mathit{Nc} \\
&\}.
\end{aligned}$$

## REFERENCES

- [1] Milad Niqui. Formalising Exact Arithmetic: Representations, Algorithms and Proofs. PhD Thesis, Radboud University Nijmegen, September 2004.
- [2] L. Cruz-Filipe. A constructive formalization of the fundamental theorem of calculus. In H. Geuvers and F. Wiedijk, editors, Types for Proofs and Programs, volume 2646 of LCNS, pages 108–126. Springer-Verlag, 2003.
- [3] Russell O'Connor. Few Digits 0.5.0.
- [4] Gabriel Stolzenberg. A New Course of Analysis. Unpublished.