Constructive axiomatic for the real numbers

Jean-Marie Madiot,
Pierre-Marie Pédrot

Coqtail Junior Laboratory
ENS Lyon

August, 26th

Real analysis in Coq

- Two major libraries of real analysis out there:
  - Coq `stdlib`: highly classical
  - C-CoRN: designed to be constructive
- Libraries to prove analysis theorems
- There already have been attempts to mix both

Real analysis in Coq

- Two major libraries of real analysis out there:
  - Coq `stdlib`: highly classical
  - C-CoRN: designed to be constructive
- Libraries to prove analysis theorems
- There already have been attempts to mix both
- Some other effective implementations, generally from the folks in computer arithmetic:
  - In general, not libraries of theorems
  - More about proved computation on real numbers

Coq `stdlib`

- Pros
  - Simple and abstract formalism
  - Designed with speed of development on mind
  - Usual proofs are easy to write
  - Shipped with Coq $\rightsquigarrow$ important base of users

Coq `stdlib`

- Pros
  - Simple and abstract formalism
  - Designed with speed of development on mind
  - Usual proofs are easy to write
  - Shipped with Coq $\rightsquigarrow$ important base of users
- Cons
  - lacks a lot of basic results (on sequences, etc.)
  - names quite messy: lemmas `Riemann_tech24` and alike
  - globally badly designed library ($\texttt{Ranalysis}_i$ for $0 \leq i \leq 4$)
  - worse than highly classical:

$$\forall A : \texttt{Prop}, \{\neg\neg A\} + \{\neg A\}$$

$$\{\forall n, Pn\} + \{\exists n, \neg(Pn)\} \quad \text{if } P : \texttt{nat} \to \texttt{Prop} \text{ decidable}$$

C-CoRN

- Pros
  - Constructive: compute with your proofs!
  - Huge database of intermediate structures
  - A lot of nice results

C-CoRN

- Pros
  - Constructive: compute with your proofs!
  - Huge database of intermediate structures
  - A lot of nice results
- Cons
  - Constructive: make mathematicians flee away!
    - ↪ "we want real maths!"
  - Too complicated to use (from compilation to dependency hell)
  - Oldish and overly module-relying Coq
  - Not really maintained anymore

Manifesto

We want to keep the best of both worlds:

- From `stdlib`:
  - Simplicity and abstraction
    - ↪ that means fresh axiomatic construction
  - Ability to do classical proofs
    - ↪ fancy axioms in `Prop` accepted
    - ↪ excluded middle, choice axiom...
    - ↪ may port classical tactics (`fourier`, `psatz`...)

Manifesto

We want to keep the best of both worlds:

- From `stdlib`:
  - Simplicity and abstraction
    - $\hookrightarrow$ that means fresh axiomatic construction
  - Ability to do classical proofs
    - $\hookrightarrow$ fancy axioms in `Prop` accepted
    - $\hookrightarrow$ excluded middle, choice axiom...
    - $\hookrightarrow$ may port classical tactics (`fourier`, `psatz`...)
- From C-CoRN:
  - Constructive axiomatic
    - $\hookrightarrow$ results in `Type` extractible
    - $\hookrightarrow$ plugging in C-CoRN?
  - *Tabula rasa* $\rightsquigarrow$ lot of work

Manifesto

We want to keep the best of both worlds:

- From `stdlib`:
  - Simplicity and abstraction
    - $\hookrightarrow$ that means fresh axiomatic construction
  - Ability to do classical proofs
    - $\hookrightarrow$ fancy axioms in `Prop` accepted
    - $\hookrightarrow$ excluded middle, choice axiom...
    - $\hookrightarrow$ may port classical tactics (`fourier`, `psatz`...)
- From C-CoRN:
  - Constructive axiomatic
    - $\hookrightarrow$ results in `Type` extractible
    - $\hookrightarrow$ plugging in C-CoRN?
  - *Tabula rasa* $\rightsquigarrow$ lot of work

Neither obvious nor easy!

Caveats

A fundamental problem: equality!

- Leibniz equality on real numbers is a quotient
  - $x \simeq y := \forall \varepsilon > 0, \delta(x, y) < \varepsilon$
- Whenever we can approximate $\mathbb{R}$ this is not constructive
  - ... as $\texttt{eval}_\varepsilon : \mathbb{R} \to \mathbb{Q}$ is not continuous
  - while any field operation must be compatible with $\simeq$
  - $\texttt{eval}_\varepsilon$ cannot be

  $\hookrightarrow$ this would break extraction

Caveats

A fundamental problem: equality!

- Leibniz equality on real numbers is a quotient
  - $x \simeq y := \forall \varepsilon > 0, \delta(x, y) < \varepsilon$
- Whenever we can approximate $\mathbb{R}$ this is not constructive
  - ... as $\texttt{eval}_\varepsilon : \mathbb{R} \to \mathbb{Q}$ is not continuous
  - while any field operation must be compatible with $\simeq$
  - $\texttt{eval}_\varepsilon$ cannot be

    $\hookrightarrow$ this would break extraction
- Really need to use setoids

Other problems related to constructivity: partial functions must take a proof, and other non-equivalences

Actual Axiomatic

- Structure and operations:
  - axiomatic: $\mathbb{R} : \texttt{Type}$, $<: \mathbb{R} \to \mathbb{R} \to \texttt{Prop}$
  - derived:

$$x \gtrless y := \{x < y\} + \{y < x\} \qquad \text{in } \texttt{Type}$$
$$x \simeq y := \neg(x < y) \wedge \neg(y < x) \qquad \text{in } \texttt{Prop}$$

  - axiomatic: $+$, $-$, $\times$, $(\cdot)^{-1}$ (with a proof that $x \gtrless 0$)
- Usual well-behavedness axioms
- Important issue: completeness!
  - As in C-CoRN
  - Constructive Cauchy sequences (`sig` in `Type`)
  - Any such sequence converges

EM and Markov's principle

Assuming EM in `Prop`, we get Markov's principle for free.

- Equivalent to $\neg\neg x > 0 \to x > 0$
  - which already implied $\{n : \mathbb{N} \mid x > 2^{-n}\}$
- Simplifies a lot of reasonings
  - we could implement Fourier elimination
- We can still kind of compute using tactic-based `Acc` trick

EM is not harmful for constructivity: things in `Prop` were considered irrelevant from the beginning.

Really Classical Maths

- Up to now, we're not a lot different from a slightly classical C-CoRN
- Castéran proposal: a violent axiom that confuse `Prop` and `Type`

$$\varepsilon : \forall A\ P, (\texttt{exists}\ x, P\,x) \to \{x \mid P\,x\}$$

- We do not want to mix non-constructive results with constructive ones
  - ... there is `stdlib` for this!
- We use a structure inherited from programming: monads!
  - A type constructor $T : \texttt{Type} \to \texttt{Type}$
  - A lift $A \to TA$ and a join $T^2A \to A$
  - in general, no information flow $TA \to A$
    - $\rightsquigarrow$ that's what we wanted

The Inhabited Monad

The inhabited monad : a singleton constructor in `Prop`

$$\texttt{Inductive inhabited } A : \texttt{Prop} := \texttt{inhabits} : A \to \texttt{inhabited } A.$$

The Inhabited Monad

The inhabited monad : a singleton constructor in `Prop`

$$\texttt{Inductive inhabited } A : \texttt{Prop} := \texttt{inhabits} : A \rightarrow \texttt{inhabited}\, A.$$

- turn any constructive predicate in a non-constructive one in `Prop`
- quite delicate to use
  - need explicit specifications (proof-irrelevance!)
- actually this is really useful together with the full axiom of choice

$$\forall A\ (B : A \rightarrow \texttt{Type}), (\forall x, \texttt{inhabited}\ (B\, x)) \rightarrow \texttt{inhabited}\ (\forall x, B\, x)$$

The Axiom Monad

A very generic and useful monad.

- The axiom monad : $T_X A = X \to A$
- used with $X = \varepsilon$ provides the full power of choice axiom
  - partial functions, elimination of dependence in proofs and much more!
  - easy to use in a mathematical fashion
  - equivalent to `inhabited` + AC
- can be refined with any other powerful axiom at will

We can virtually tag any result with its degree of classicality (here we're just interested in algorithmic realizability).

Conclusion

- Writing a manageable real arithmetic library is difficult
  - on a theoretical point of view: carefully choose your system!
  - on a practical point of view: naming conventions and usability
- Rewriting a whole generic library from scratch is a tedious work
- Actually we don't have any interesting result yet...
  - just a bunch of basic lemmas wich are the most cumbersome to write
  - we badly lack tactics for now
- Using monads to discriminate proof properties seems something new

Scribitur ad narrandum, not ad probandum

Thank you.