Verified Implementation of Exact Real Arithmetic in Type Theory

Bas Spitters
EU STREP-FET ForMath

Dec 9th 2013
Computable Analysis and Rigorous Numerics

Rigorous Numerics through Computable Analysis

Constructive analysis as an internal language for TTE
Type theory as a language for constructive mathematics
Type theory as a framework for computer proofs

Computer verified implementation of exact analysis

Dependent type theory

- Dependent type theory makes Bishop's notion of operation/construction precise.
- Gives a functional programming language like haskell, SML, OCaml with very expressive type system.
- Framework for proofs
- Implementations: Coq, agda, epigram, Idris, ...
- Extensional DTT is an abstract language for TTE via realizability. Locally Cartesian closed category, ΠW-pretopos.

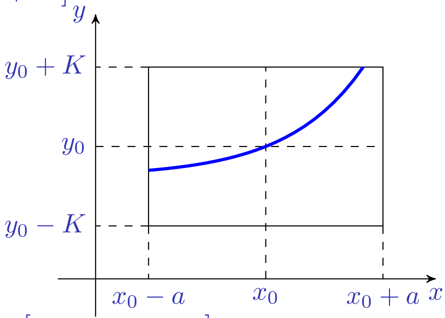
Picard-Lindelöf Theorem

Consider the initial value problem

$$y'(x) = v(x, y(x)), \quad y(x_0) = y_0$$

where

- $v : [x_0 - a, x_0 + a] \times [y_0 - K, y_0 + K] \to \mathbf{R}$
- v is continuous
- v is Lipschitz continuous in y: $|v(x,y) - v(x,y')| \leq L|y - y'|$ for some $L > 0$
- $|v(x,y)| \leq M$
- $aL < 1$
- $aM \leq K$



Such problem has a unique solution on $[x_0 - a, x_0 + a]$.

Proof Idea

$$y'(x) = v(x, y(x)), \quad y(x_0) = y_0$$

is equivalent to

$$y(x) = y(x_0) + \int_{x_0}^{x} v(t, y(t))\, dt$$

Define

$$(Tf)(x) = y_0 + \int_{x_0}^{x} F(t, f(t))\, dt$$
$$f_0(x) = y_0$$
$$f_{n+1} = Tf_n$$

Under the assumptions, T is a contraction on $C([x_0 - a, x_0 + a], [y_0 - K, y_0 + K])$.
By the Banach fixpoint theorem, T has a fixpoint f and $f_n \to f$.

Formalization: Makarov, S - The Picard Algorithm for Ordinary Differential Equations in Coq

Metric Spaces

Let (X, d) where $d : X \to X \to \mathbf{R}$ be a metric space.

Let $Brxy$ denote $d(x, y) \le r$.

A function $f : \mathbf{Q}^+ \to X$ is called regular (Strongly Cauchy) if $\forall \varepsilon_1\ \varepsilon_2 : \mathbf{Q}^+,\ B(\varepsilon_1 + \varepsilon_2)(f\varepsilon_1)(f\varepsilon_2)$.

The completion $\mathfrak{C}\,X$ of X is the set of regular functions.

Let X and Y be metric spaces. A function $f : X \to Y$ is called uniformly continuous with modulus μ if $\forall \varepsilon : \mathbf{Q}^+\ \forall x_1\ x_2 : X,\ B(\mu\varepsilon)x_1x_2 \to B\varepsilon(fx_1)(fx_2)$.

For $x_1, x_2 : \mathfrak{C}\,X$, the metric on the completion $B_{\mathfrak{C}\,X}\varepsilon x_1 x_2 := \forall \varepsilon_1\varepsilon_2 : \mathbf{Q}^+,\ B_X(\varepsilon_1 + \varepsilon + \varepsilon_2)(x_1\varepsilon_1)(x_2\varepsilon_2)$.

Metric spaces with uniformly continuous functions form a category.

Completion forms a monad in the category of metric spaces and uniformly continuous functions.

R. O'Connor, extending work by E. Bishop.

Completion as a Monad

$\mathsf{unit} : X \to \mathfrak{C}\, X := \lambda x \lambda \varepsilon,\ x$

$\mathsf{join} : \mathfrak{C}\,\mathfrak{C}\, X \to \mathfrak{C}\, X := \lambda x \lambda \varepsilon,\ x(\varepsilon/2)(\varepsilon/2)$

$\mathsf{map} : (X \to Y) \to (\mathfrak{C}\, X \to \mathfrak{C}\, Y) := \lambda f \lambda x,\ f \circ x \circ \mu_f$

$\mathsf{bind} : (X \to \mathfrak{C}\, Y) \to (\mathfrak{C}\, X \to \mathfrak{C}\, Y) := \mathsf{join} \circ \mathsf{map}$

Define functions $\mathbf{Q} \to \mathbf{Q}$; lift to $\mathfrak{C}\,\mathbf{Q} \to \mathfrak{C}\,\mathbf{Q}$.

Type Classes

Organize the library using, so-called *type classes*.
Type classes are parametric record types.
Coq searches for terms of these types automatically during unification.
Logic programming at the type level automates many mathematical reflexes.

- Gives uniform notation
- Algebra hierarchy, abstractions, diamond inheritance
- Abstract interfaces (like haskell).

S, van der Weegen, *Type Classes for Mathematics in Type Theory*.

Type Classes for Mathematical Structures

```
Class AppRationals AQ {e plus mult zero one inv} `{Apart AQ}
    `{Le AQ} `{Lt AQ}
     {AQtoQ : Cast AQ Q_as_MetricSpace}
    `{!AppInverse AQtoQ} {ZtoAQ : Cast Z AQ}
    `{!AppDiv AQ} `{!AppApprox AQ}
    `{!Abs AQ} `{!Pow AQ N} `{!ShiftL AQ Z}
    `{∀ x y : AQ, Decision (x = y)}
    `{∀ x y : AQ, Decision (x ≤ y)} : Prop := {
  aq_ring :> @Ring AQ e plus mult zero one inv ;
  aq_trivial_apart :> TrivialApart AQ ;
  aq_order_embed :> OrderEmbedding AQtoQ ;
  aq_strict_order_embed :> StrictOrderEmbedding AQtoQ ;
  aq_ring_morphism :> SemiRing_Morphism AQtoQ ;
  aq_dense_embedding :> DenseEmbedding AQtoQ ;
  aq_div : ∀ x y k, ball (2 ^ k) ('app_div x y k) ('x / 'y) ;
  aq_compress : ∀ x k, ball (2 ^ k) ('app_approx x k) ('x) ;
  aq_shift :> ShiftLSpec AQ Z (≪) ;
  aq_nat_pow :> NatPowSpec AQ N (^) ;
  aq_ints_mor :> SemiRing_Morphism ZtoAQ
}.
```

S, Krebbers, *Type classes for efficient exact real arithmetic in Coq*

Instances of Approximate Rationals

```
Record Dyadic Z := dyadic { mant: Z; expo: Z }.
```

Represents $\texttt{mant} \cdot 2^{\texttt{expo}}$

```
Instance dy_mult: Mult Dyadic :=
  λ x y, dyadic (mant x * mant y) (expo x + expo y).
```

```
Instance : AppRationals (Dyadic bigZ).
```

```
Instance : AppRationals bigQ.
```

```
Instance : AppRationals Q.
```

Waiting for MPFR/Coq interval/floqc . . .

Efficient Reals

```
Coq < Check Complete.
Complete : MetricSpace -> MetricSpace

Coq < Check Q_as_MetricSpace.
Q_as_MetricSpace : MetricSpace

Coq < Check AQ_as_MetricSpace.
AQ_as_MetricSpace :
  ∀ (AQ : Type) ..., AppRationals AQ -> MetricSpace

Coq < Definition CR := Complete Q_as_MetricSpace.

Coq < Definition AR := Complete AQ_as_MetricSpace.
```

AR is an instance of `Le`, `Field`, `SemiRingOrder`, etc., from the MathClasses library.

Integral

Following M. Bridger, *Real Analysis: A Constructive Approach*.

```
Class Integral (f: Q -> CR) :=
  integrate: forall (from: Q) (w: QnonNeg), CR.

Notation "∫" := integrate.

Class Integrable `{!Integral f}: Prop := {
  integral_additive:
    forall (a: Q) b c, ∫ f a b + ∫ f (a + b) c == ∫ f a (b + c);

  integral_bounded_prim: forall (from: Q) (width: Qpos) (mid: Q)
    (forall x, from <= x <= from + width -> ball r (f x) mid) ->
    ball (width * r) (∫ f from width) (width * mid);

}.
```

Earlier (abstract, but slower) implementation of integral by O'Connor and S

Complexity

Rectangle rule:

$$\left|\int_a^b f(x)\,dx - f(a)(b-a)\right| \leq \frac{(b-a)^3}{24}M$$

where $|f''(x)| \leq M$ for $a \leq x \leq b$.

Number of intervals to have the error $\leq \varepsilon$: $\geq \sqrt{\dfrac{(b-a)^3 M}{24\varepsilon}}$

Simpson's rule:

$$\left|\int_a^b f(x)\,dx - \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)\right| \leq \frac{(b-a)^5}{2880}M$$

where $|f^{(4)}(x)| \leq M$ for $a \leq x \leq b$.

Number of intervals: $\geq \sqrt[4]{\dfrac{(b-a)^5 M}{2880\varepsilon}}$

Coquand, S *A constructive proof of Simpson's Rule*, 2012:
Replace mean value theorem with law of bounded change
Use divided differences, Hermite-Genocci

Conclusions

- Computer *verified* implementation of simple ODE solver.
- Computing with exact functions.
- May be seen as an executable specification, speed up with refinement.