From computational analysis to thoughts about analysis in HoTT

Bas Spitters
Robbert Krebbers
Eelis van der Weegen

Supported by EU FP7 STREP FET-open ForMATH

MAP INTERNATIONAL SPRING SCHOOL ON FORMALIZATION OF MATHEMATICS 2012

SOPHIA ANTIPOLIS, FRANCE / 12-16 MARCH

Inria informatics mathematics

ihp Institut Henri Poincaré

MICROSOFT RESEARCH INRIA JOINT CENTRE

Why do we need certified exact arithmetic?

- There is a big gap between:
  - Numerical algorithms in research papers.
  - Actual implementations (MATHEMATICA, MATLAB, ...).

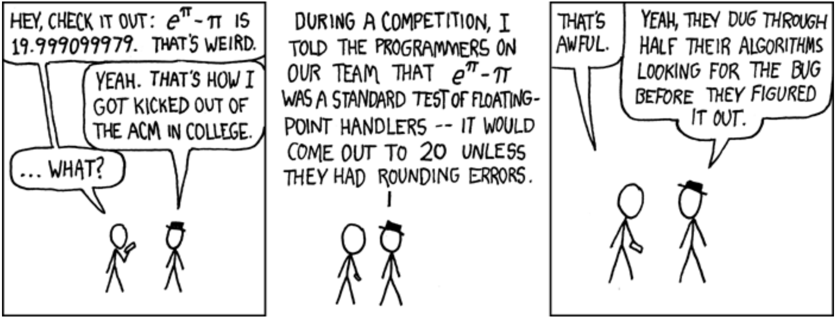
Why do we need certified exact arithmetic?

- There is a big gap between:
  - Numerical algorithms in research papers.
  - Actual implementations (MATHEMATICA, MATLAB, ...).
- This gap makes the code difficult to maintain.
- Makes it difficult to trust the code of these implementations!

Why do we need certified exact arithmetic?

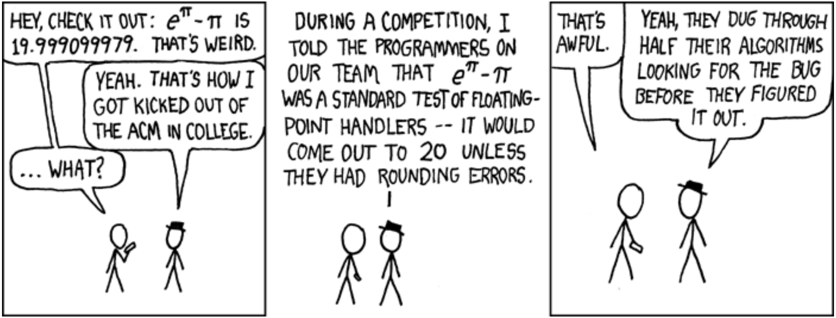
- There is a big gap between:
  - Numerical algorithms in research papers.
  - Actual implementations (MATHEMATICA, MATLAB, ...).
- This gap makes the code difficult to maintain.
- Makes it difficult to trust the code of these implementations!
- Undesirable in proofs that rely on the execution of this code.
  - Kepler conjecture.
  - Existence of the Lorentz attractor.

Why do we need certified exact real arithmetic?



(`http://xkcd.com/217/`)

Why do we need certified exact real arithmetic?



(http://xkcd.com/217/)

Bishop's proposal

Use constructive analysis to bridge this gap.

- Exact real numbers instead of floating point numbers.
- Functional programming instead of imperative programming.
- Dependent type theory.
- A proof assistant to verify the correctness proofs.
- Constructive mathematics to tightly connect mathematics with computations.

Real numbers

- Cannot be represented exactly in a computer.
- Approximation by rational numbers.
- Or any set that is dense in the rationals (e.g. the dyadics).

O'Connor's implementation in Coq

- Based on *metric spaces* and the *completion monad*.

$$\mathbb{R} := \mathfrak{C}\mathbb{Q} := \{f : \mathbb{Q}_+ \to \mathbb{Q} \mid f \text{ is regular}\}$$

- To define a function $\mathbb{R} \to \mathbb{R}$: define a *uniformly continuous function* $f : \mathbb{Q} \to \mathbb{R}$, and obtain $\check{f} : \mathbb{R} \to \mathbb{R}$.
- Efficient combination of proving and programming.

O'Connor's implementation in Coq

- Based on *metric spaces* and the *completion monad*.

$$\mathbb{R} := \mathfrak{C}\mathbb{Q} := \{f : \mathbb{Q}_+ \to \mathbb{Q} \mid f \textit{ is regular}\}$$

- To define a function $\mathbb{R} \to \mathbb{R}$: define a *uniformly continuous function* $f : \mathbb{Q} \to \mathbb{R}$, and obtain $\check{f} : \mathbb{R} \to \mathbb{R}$.
- Efficient combination of proving and programming.

O'Connor's implementation in Coq

Problem:

- A concrete representation of the rationals (`Q`) is used.
- Cannot swap implementations, e.g. use machine integers.

O'Connor's implementation in Coq

Problem:

- A concrete representation of the rationals (`Q`) is used.
- Cannot swap implementations, e.g. use machine integers.

Solution:
Build theory and programs on top of abstract interfaces instead of concrete implementations.

- Cleaner.
- Mathematically sound.
- Can swap implementations.

Our contribution

- Provide an abstract specification of the dense set.
- For which we provide an implementation using the dyadics:

$$n * 2^e \qquad \text{for} \qquad n, e \in \mathbb{Z}$$

- Use COQ's machine integers.
- Extend our algebraic hierarchy based on type classes
- Implement range reductions.
- Improve computation of power series:
  - Keep auxiliary results small.
  - Avoid evaluation of termination proofs.

Interfaces for mathematical structures

- Algebraic hierarchy (groups, rings, fields, ...)
- Relations, orders, ...
- Categories, functors, universal algebra, ...
- Numbers: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, ...

Need solid representations of these, providing:

- Structure inference.
- Multiple inheritance/sharing.
- Convenient algebraic manipulation (e.g. rewriting).
- Idiomatic use of names and notations.

S/and van der Weegen: use type classes

Type classes

- Useful for organizing interfaces of abstract structures.
- Similar to AXIOM's so-called categories.
- Great success in HASKELL and ISABELLE.
- Recently added to COQ.
- Based on already existing features (records, proof search, implicit arguments).

Proof engineering

Comparison(?) to canonical structures, unification hints

Unbundled using type classes

Define *operational type classes* for operations and relations.

```
Class Equiv A := equiv: relation A.
Infix "=" := equiv: type_scope.
Class RingPlus A := ring_plus: A → A → A.
Infix "+" := ring_plus.
```

Represent algebraic structures as predicate type classes.

```
Class SemiRing A {e plus mult zero one} : Prop := {
 semiring_mult_monoid :> @CommutativeMonoid A e mult one ;
 semiring_plus_monoid :> @CommutativeMonoid A e plus zero ;
 semiring_distr :> Distribute (.∗.) (+) ;
 semiring_left_absorb :> LeftAbsorb (.∗.) 0 }.
```

Examples

```
(* z & x = z & y → x = y *)
Instance group_cancel `{Group G} : ∀ z, LeftCancellation (&) z.
```

Examples

```
(* z & x = z & y → x = y *)
Instance group_cancel `{Group G} : ∀ z, LeftCancellation (&) z.

Lemma preserves_inv `{Group A} `{Group B}
 `{!Monoid_Morphism (f : A → B)} x : f (−x) = −f x.
Proof.
 apply (left_cancellation (&) (f x)).
 rewrite ← preserves_sg_op.
 rewrite 2!right_inverse.
 apply preserves_mon_unit.
Qed.
```

Examples

```
(* z & x = z & y → x = y *)
Instance group_cancel `{Group G} : ∀ z, LeftCancellation (&) z.

Lemma preserves_inv `{Group A} `{Group B}
 `{!Monoid_Morphism (f : A → B)} x : f (−x) = −f x.
Proof.
 apply (left_cancellation (&) (f x)).
 rewrite ← preserves_sg_op.
 rewrite 2!right_inverse.
 apply preserves_mon_unit.
Qed.

Lemma cancel_ring_test `{Ring R} x y z : x + y = z + x → y = z.
Proof.
 intros.
 apply (left_cancellation (+) x).
 now rewrite (commutativity x z).
Qed.
```

Number structures

S/van der Weegen specified:

- Naturals: initial semiring.
- Integers: initial ring.
- Rationals: field of fractions of $\mathbb{Z}$.

Approximate rationals

```
Class AppDiv AQ := app_div : AQ → AQ → Z → AQ.
Class AppApprox AQ := app_approx : AQ → Z → AQ.

Class AppRationals AQ {e plus mult zero one inv} `{!Order AQ}
  {AQtoQ : Coerce AQ Q_as_MetricSpace} `{!AppInverse AQtoQ}
  {ZtoAQ : Coerce 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_order_embed :> OrderEmbedding AQtoQ ;
  aq_ring_morphism :> SemiRing_Morphism AQtoQ ;
  aq_dense_embedding :> DenseEmbedding AQtoQ ;
  aq_div : ∀ x y k, B_{2^k}('app_div x y k) ('x / 'y) ;
  aq_approx : ∀ x k, B_{2^k}('app_approx x k) ('x) ;
  aq_shift :> ShiftLSpec AQ Z (≪) ;
  aq_nat_pow :> NatPowSpec AQ N (^) ;
  aq_ints_mor :> SemiRing_Morphism ZtoAQ }.
```

Approximate rationals

Compress

```
Class AppDiv AQ := app_div : AQ → AQ → Z → AQ.
Class AppApprox AQ := app_approx : AQ → Z → AQ.
Class AppRationals AQ  ... : Prop := {
   ...
  aq_div : ∀ x y k, B_{2^k}('app_div x y k) ('x / 'y) ;
  aq_approx : ∀ x k, B_{2^k}('app_approx x k) ('x) ;
   ... }
```

- app_approx is used to to keep the size of the numbers "small".
- Define compress := bind (λ ϵ, app_approx x (Qdlog2 ϵ)) such that compress x = x.
- Greatly improves the performance [O'Connor].

Power series

- Well suited for computation if:
  - its coefficients are alternating,
  - decreasing,
  - and have limit 0.
- For example, for $-1 \le x \le 0$:

$$\exp x = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

- To approximate $\exp x$ with error ε we find a k such that:

$$\frac{x^k}{k!} \le \varepsilon$$

Power series

Problem: we do not have exact division.

- Parametrize InfiniteAlternatingSum with streams n and d representing the numerators and denominators to postpone divisions.
- Need to find both the length and precision of division.

$$\underbrace{\frac{n_1}{d_1}}_{\frac{\varepsilon}{2k}\text{error}} + \underbrace{\frac{n_2}{d_2}}_{\frac{\varepsilon}{2k}\text{error}} + \ldots + \underbrace{\frac{n_k}{d_k}}_{\frac{\varepsilon}{2k}\text{error}} \qquad \text{such that} \qquad \frac{n_k}{d_k} \leq \varepsilon/2$$

Power series

Problem: we do not have exact division.

- Parametrize InfiniteAlternatingSum with streams n and d representing the numerators and denominators to postpone divisions.
- Need to find both the length and precision of division.

$$\underbrace{\frac{n_1}{d_1}}_{\frac{\varepsilon}{2k}\text{error}} + \underbrace{\frac{n_2}{d_2}}_{\frac{\varepsilon}{2k}\text{error}} + \ldots + \underbrace{\frac{n_k}{d_k}}_{\frac{\varepsilon}{2k}\text{error}} \quad \text{such that} \quad \frac{n_k}{d_k} \leq \varepsilon/2$$

- Thus, to approximate $\exp x$ with error ε we need a k such that:

$$\mathbf{B}_{\frac{\varepsilon}{2}} \left(\text{app\_div}\ n_k\ d_k\ (\log \frac{\varepsilon}{2k}) + \frac{\varepsilon}{2k}\right) 0.$$

Power series

- Computing the length can be optimized using shifts.
- Our approach only requires to compute few extra terms.
- Approximate division keeps the auxiliary numbers “small”.
- We use a method to avoid evaluation of termination proofs.

What have we implemented so far?

Verified versions of:

- Basic field operations ($+$, $*$, -, $/$)
- Exponentiation by a natural.
- Computation of power series.
- $\exp$, $\arctan$, $\sin$ and $\cos$.
- $\pi := 176 * \arctan\frac{1}{57} + 28 * \arctan\frac{1}{239} - 48 * \arctan\frac{1}{682} + 96 * \arctan\frac{1}{12943}$.
- Square root using Wolfram iteration.

Benchmarks

- Our Haskell prototype is ∼15 times faster.
- Our Coq implementation is ∼100 times faster.
- For example:
  - 500 decimals of $\mathsf{exp}\ (\pi * \sqrt{163})$ and $\mathsf{sin}\ (\mathsf{exp}\ 1)$,
  - 2000 decimals of $\mathsf{exp}\ 1000$,

  within 10 seconds in Coq!
- (Previously about 10 decimals)

Benchmarks

- Our HASKELL prototype is ∼15 times faster.
- Our COQ implementation is ∼100 times faster.
- For example:
  - 500 decimals of $\exp(\pi * \sqrt{163})$ and $\sin(\exp 1)$,
  - 2000 decimals of $\exp 1000$,

  within 10 seconds in COQ!
- (Previously about 10 decimals)
- Type classes only yield a 3% performance loss.
- COQ is still too slow compared to unoptimized HASKELL (factor 30 for Wolfram iteration).

Future work

- native_compute: evaluation by compilation to OCAML.
  gives COQ $10\times$ boost.
- FLOCQ/Tamadi: more fine grained floating point algorithms.
- Type classified theory on metric spaces.

Conclusions

- Greatly improved the performance of the reals.
- Abstract interfaces allow to swap implementations and share theory and proofs.
- Type classes yield no apparent performance penalty.
- Nice notations with unicode symbols.

Conclusions

- Greatly improved the performance of the reals.
- Abstract interfaces allow to swap implementations and share theory and proofs.
- Type classes yield no apparent performance penalty.
- Nice notations with unicode symbols.

Issues:

- Type classes are quite fragile.
- Instance resolution is too slow.
- Need to adapt definitions to avoid evaluation in Prop.

Views

Views

I want to present my interest in homotopy type theory
Practical motivation for combining type theory and topos theory

Views

I want to present my interest in homotopy type theory
Practical motivation for combining type theory and topos theory
Polymath/n-cafe spirit

Challenges of current Coq

For discrete mathematics the ssreflect machinery works very well!
The extension to infinitary mathematics is challenging.
No quotients, functional extensionality, subsets, ...
Voevodsky's univalence axiom provides a uniform solution.
Quest for a computational interpretation.
Univalence and analysis?

Challenges of current Coq

For discrete mathematics the ssreflect machinery works very well!
The extension to infinitary mathematics is challenging.
No quotients, functional extensionality, subsets, ...
Voevodsky's univalence axiom provides a uniform solution.
Quest for a computational interpretation.
Univalence and analysis?
Homotopy type theory (HoTT):
type theory with Prop replaced by hProp.

Direct consequences

Univalence implies:

- functional extensionality
- equivalent propositions are equal.
  subset types
- isomorphic hSets are equal:
  all type theoretical constructions respect isomorphisms!

Direct consequences

Univalence implies:

- functional extensionality
- equivalent propositions are equal.
  subset types
- isomorphic hSets are equal:
  all type theoretical constructions respect isomorphisms!

Harper/Licata computational interpretation for $h = 2$.
Example:
Lists and vectors are isomorphic.
Lists form a monoid. Hence, so do vectors.

Higher inductive types

Inductive types introduce new objects.
Lumsdaine/Shulman: higher inductive types.
Also introduce new equalities.
Algebraic description of spaces in homotopical interpretation
Currently not in Coq.

isInhab

Impredicative encoding:

```
Definition ishinh (X : Type) := forall P: hProp, ( X -> P ) -> P.
```

isInhab

Impredicative encoding:

```
Definition ishinh (X : Type) := forall P: hProp, ( X -> P ) -> P.
```

Higher inductive definition:

```
Inductive is_inhab (A : Type) : Type :=
  | inhab : A -> is_inhab A
  | inhab_path : forall (x y: is_inhab A), x = y
```

Gives a ‘mechanical’ way to define introduction, elimination and computation rules.
Bauer: isInhab is a strong monad on Type.

IsInhab

```
Axiom is_inhab : forall (A : Type), Type.
Axiom inhab : forall {A : Type}, A -> is_inhab A.
Axiom inhab_path : forall {A : Type} (x y : is_inhab A), x = y.

Axiom is_inhab_rect : forall {A : Type} {P : is_inhab A -> Type}
  (dinhab : forall (a : A), P (inhab a))
  (dpath : forall (x y : is_inhab A) (z : P x) (w : P y),
     transport (inhab_path x y) z = w),
  forall (x : is_inhab A), P x.
Axiom is_inhab_compute_inhab : forall {A : Type} {P : is_inhab A -> Type}
  (dinhab : forall (a : A), P (inhab a))
  (dpath : forall (x y : is_inhab A) (z : P x) (w : P y),
     transport (inhab_path x y) z = w),
  forall (a : A), is_inhab_rect dinhab dpath (inhab a) = dinhab a.
Axiom is_inhab_compute_path : forall {A : Type} {P : is_inhab A -> Type}
  (dinhab : forall (a : A), P (inhab a))
  (dpath : forall (x y : is_inhab A) (z : P x) (w : P y),
     transport (inhab_path x y) z = w),
  forall (x y : is_inhab A),
    map_dep (is_inhab_rect dinhab dpath) (inhab_path x y) =
    dpath x y (is_inhab_rect dinhab dpath x) (is_inhab_rect dinhab dpath y).
```

Logic

Awodey/Bauer: Propositions as [ ]-types.

```
Definition hexists{X} (P:X->Type):=(is_inhab (sigT P)).
Definition hor (A B:hProp):=(is_inhab (A + B)).
```

models first-order intuitionistic logic.
Enforce proof irrelevance.

Logic of HoTT?

iHOL is the internal language of a topos
Conjecture (Awodey):
HoTT as the internal language of an ∞-topos
(Shulman: still some hard open questions.)
Outlook: categorical models for Coq.

Logic of HoTT?

hSets form a predicative topos.
Using resizing axioms, it becomes a topos.
No formal proof yet.
We present some key theorems:

Axiom of description

```
Definition hexists{X} (P:X->Type):=(is_inhab (sigT P)).
Definition atmost1P {X} (P:X->Type):=
  forall x1 x2 :X, P x1 -> P x2 -> (x1 = x2 ).
Definition hunique {X} (P:X->Type):=(hexists P) * (atmost1P P).
Lemma iota {X} (P:X->hProp): (hunique P) -> sigT P.
```

Note: in Coq, we cannot escape Prop.

iota breaks program extraction, we cannot remove hProps.

Epis are surjective

Consequences of univalence:

```
Axiom uahp : forall P P':hProp, (P -> P') -> (P' -> P) -> paths P P'.
Axiom isasethProp: is_set hProp.

Definition epi {X Y:type1} `(f:X->Y):=
  forall Z:hSet, forall g h: Y -> Z, g o f = h o f -> g = h.
Definition surj {X Y:type1} `(f:X->Y):type1 :=
  forall y:Y , hexists (fun x:X => (f x) = y).
Lemma epi_surj {X Y:type1} (f:X->Y): epi f -> surj f.
```

Need proper universe management.

Quotients

Coq does not have quotients.
Voevodsky: univalence provides quotients.
Quotients can be defined as a higher inductive type.

```
Inductive Quot (A : Type) (R:hrel A) : Type :=
  | quot : A -> Quot A
  | quot_path : forall x y, (R x y), quot x = quot y
```

Voevodsky's quotient indeed verify the universal properties generated by the higher inductive type.
Useful for a practical implementation.

Reals as a quotient

How about the reals? Currently, reals are a setoid.
With quotients we have a type of Cauchy reals.
Their theory in a topos is well-understood.
Compare with alternatives (Dedekind).

Research questions

hSets provide us with a predicative topos.
Allows us to define sheaves.

Research questions

hSets provide us with a predicative topos.
Allows us to define sheaves.
Recent interest in presheaves:
Kripke models for Coq to add complex programming language features to Coq (Jaber, Tabareau, Sozeau):
recursive types, stateful programs, ...

Research questions

hSets provide us with a predicative topos.
Allows us to define sheaves.
Recent interest in presheaves:
Kripke models for Coq to add complex programming language features to Coq (Jaber, Tabareau, Sozeau):
recursive types, stateful programs, ...
They define presheaves in the (somewhat) proof irrelevant, extensional type theory of RUSSELL.
Needs to be extended to identity types.

Research questions

Conjecture I: presheaves in HoTT like JTS, but including proper treatment of identity types.
Conjecture II: compare with model structures on simplicial presheaves.
Motivated by both programming and semantics.
Extend to simplicial sheaves (cf. Joyal/Jardin).

Outlook

Promises to combine two approaches to constructive mathematics: types and toposes.
Sheaves have many uses:

- Constructive interpretation of classical logic (dynamic evaluation): e.g. algebraic closure (Coquand, Mannaa)
- Non-derivability in Coq via model constructions.
- Proof mining: obtain a modulus of uniform continuity from a continuous function $f : [0,1] \to \mathbb{R}$.
- Complex programming language features (JTS).
- Nominal techniques using Schanuel topos
- ...