Computable analysis, exact real arithmetic and analytic functions in Coq

Holger Thies, Kyushu University
Florian Steinberg, INRIA Saclay

September 8, 2019

The Coq Workshop
Portland,OR, USA

Computable analysis

Computability & Complexity in Analysis
NETWORK

http://www.cca-net.de

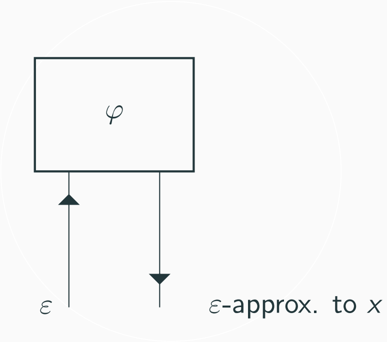
Study problems from real analysis using methods from computability theory and computational complexity theory.

- What problems can be computed in principle e.g. by Turing machines?
- Which of these problems can be computed efficiently?
- The model provides a realistic model of computation not only for real numbers but also spaces of functions etc.

Exact computation with real numbers

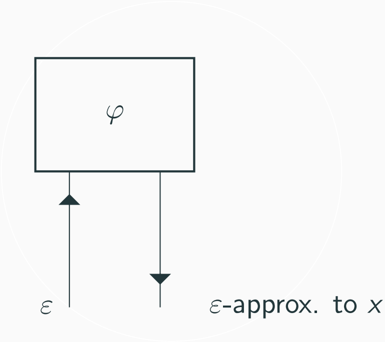
Computing real numbers

A real number $x \in \mathbb{R}$ is called computable if there is a computable function $\varphi : \mathbb{Q} \to \mathbb{Q}$ such that $\forall \varepsilon \in \mathbb{Q}, |\varphi(n) - x| \leq \varepsilon$.



Computing real numbers

A real number $x \in \mathbb{R}$ is called computable if there is a computable function $\varphi : \mathbb{Q} \to \mathbb{Q}$ such that $\forall \varepsilon \in \mathbb{Q}, |\varphi(n) - x| \leq \varepsilon$.



A function φ as above is called a name of x $\rightsquigarrow$ A real number is computable if it has a computable name.

Computing with real numbers

Real numbers are encoded by rational approximation functions.

- Computable functions: relate $f : \mathbb{R} \to \mathbb{R}$ to approximation function $F : (\mathbb{Q} \to \mathbb{Q}) \to \mathbb{Q} \to \mathbb{Q}$ s.t. $|F(\varphi)(\varepsilon) - f(x)| \leq \varepsilon$ for all φ that are names for x and all $\varepsilon > 0$.

Computing with real numbers

Real numbers are encoded by rational approximation functions.

- Computable functions: relate $f : \mathbb{R} \to \mathbb{R}$ to approximation function $F : (\mathbb{Q} \to \mathbb{Q}) \to \mathbb{Q} \to \mathbb{Q}$ s.t. $|F(\varphi)(\varepsilon) - f(x)| \leq \varepsilon$ for all φ that are names for x and all $\varepsilon > 0$.
- Equivalently the function F maps names of x to names of $f(x)$.

Computing with real numbers

Real numbers are encoded by rational approximation functions.

- Computable functions: relate $f : \mathbb{R} \to \mathbb{R}$ to approximation function $F : (\mathbb{Q} \to \mathbb{Q}) \to \mathbb{Q} \to \mathbb{Q}$ s.t. $|F(\varphi)(\varepsilon) - f(x)| \leq \varepsilon$ for all φ that are names for x and all $\varepsilon > 0$.
- Equivalently the function F maps names of x to names of $f(x)$.
- Such a function F is called a realizer for F in computable analysis.

Computing with reals in coq

Easy to define in Coq using the axiomatization of the reals in the standard library:

```
(* A name for x encodes x by rational approximations *)
Definition is_name (phi : (Q -> Q)) (x : R) :=
      forall eps, (0 < (Q2R eps)) ->
        Rabs (x - (phi eps)) <= eps.
```

Computing with reals in coq

Easy to define in Coq using the axiomatization of the reals in the standard library:

```
(* A name for x encodes x by rational approximations *)
Definition is_name (phi : (Q -> Q)) (x : R) :=
      forall eps, (0 < (Q2R eps)) ->
        Rabs (x - (phi eps)) <= eps.

(* A name for zero *)
Lemma zero_name : (is_name (fun eps => eps) 0).
```

Realizers for reals in coq

```
(* A realizer maps names to names *)
Definition is_realizer
        (F: (Q -> Q) -> Q -> Q) (f : R -> R) :=
          forall phi x, (is_name phi x) ->
              (is_name (F phi) (f x)).
```

Realizers for reals in coq

```
(* A realizer maps names to names *)
Definition is_realizer
        (F: (Q -> Q) -> Q -> Q) (f : R -> R) :=
          forall phi x, (is_name phi x) ->
              (is_name (F phi) (f x)).

Definition double_realizer (phi : Q -> Q) eps :=
    (2*(phi (eps/2)))%Q.

Lemma double_realizer_correct :
   (is_realizer double_realizer (fun x => 2*x)).
   [...]
```

Realizers for reals in coq

```
(* A realizer maps names to names *)
Definition is_realizer
         (F: (Q -> Q) -> Q -> Q) (f : R -> R) :=
           forall phi x, (is_name phi x) ->
               (is_name (F phi) (f x)).

Definition double_realizer (phi : Q -> Q) eps :=
    (2*(phi (eps/2)))%Q.


Lemma double_realizer_correct :
   (is_realizer double_realizer (fun x => 2*x)).
   [...]
```

Define realizers to specify algorithms and correctness proofs using classical mathematics.

Representations

- Also want to consider other ways to encode reals.

Representations

- Also want to consider other ways to encode reals.
- We do not only want to compute with real numbers but also with e.g. real vectors, complex numbers or more complicated spaces.

Representations

- Also want to consider other ways to encode reals.
- We do not only want to compute with real numbers but also with e.g. real vectors, complex numbers or more complicated spaces.
- In particular function spaces like $C([0, 1])$ are interesting to define computation of numerical operators such as integration, differentiation, ODE solving, ...

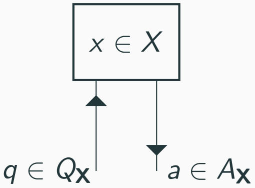
Representations

- Also want to consider other ways to encode reals.
- We do not only want to compute with real numbers but also with e.g. real vectors, complex numbers or more complicated spaces.
- In particular function spaces like $C([0,1])$ are interesting to define computation of numerical operators such as integration, differentiation, ODE solving, ...

Representations

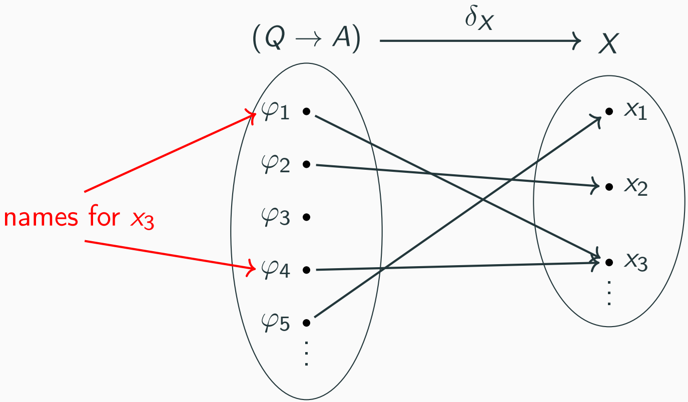
- Also want to consider other ways to encode reals.
- We do not only want to compute with real numbers but also with e.g. real vectors, complex numbers or more complicated spaces.
- In particular function spaces like $C([0,1])$ are interesting to define computation of numerical operators such as integration, differentiation, ODE solving, ...

More general encodings: Encode by a function from "questions" to "answers".
In computable analysis encodings of spaces of continuum cardinality are called representations.



Representations

Representation for a space X: A partial surjective function $\delta :\subseteq \mathcal{B} \to X$.



Represented space $\mathbf{X} := (X, \delta_X)$.

Computable analysis in Coq

- Partial functions important in computable analysis $\rightsquigarrow$ use relations.

Computable analysis in Coq

- Partial functions important in computable analysis $\rightsquigarrow$ use relations.
- The incone library (Steinberg) is an attempt to formalize results from computable analysis in Coq.

Computable analysis in Coq

- Partial functions important in computable analysis $\rightsquigarrow$ use relations.
- The incone library (Steinberg) is an attempt to formalize results from computable analysis in Coq.
- 21528 loc in 109 files

Computable analysis in Coq

- Partial functions important in computable analysis $\rightsquigarrow$ use relations.
- The incone library (Steinberg) is an attempt to formalize results from computable analysis in Coq.
- 21528 loc in 109 files
- It provides formal definitions of represented spaces and continuity similar to those in computable analysis.

Computable analysis in Coq

- Partial functions important in computable analysis $\rightsquigarrow$ use relations.
- The incone library (Steinberg) is an attempt to formalize results from computable analysis in Coq.
- 21528 loc in 109 files
- It provides formal definitions of represented spaces and continuity similar to those in computable analysis.
- Realizers should be constructive, i.e., executable inside Coq.

Computable analysis in Coq

- Partial functions important in computable analysis $\rightsquigarrow$ use relations.
- The incone library (Steinberg) is an attempt to formalize results from computable analysis in Coq.
- 21528 loc in 109 files
- It provides formal definitions of represented spaces and continuity similar to those in computable analysis.
- Realizers should be constructive, i.e., executable inside Coq.
- Reasoning about correctness and the relation between represented space and abstract space non-constructive (e.g. use axiomatization of the reals in the standard library, classical reasoning, countable choice)

Represented spaces in Coq

Instead of encoding everything by string functions encode by "simple" types in Coq.

Represented spaces in Coq

Instead of encoding everything by string functions encode by "simple" types in Coq.

Represented space

A represented space **X** consists of

- An abstract base type X.

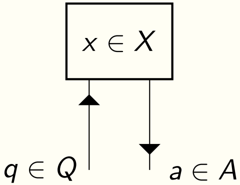
Represented spaces in Coq

Instead of encoding everything by string functions encode by "simple" types in Coq.

Represented space

A represented space **X** consists of

- An abstract base type X.
- Types of questions Q and answers A.



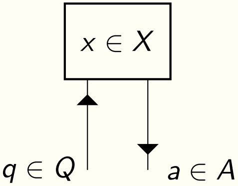
Represented spaces in Coq

Instead of encoding everything by string functions encode by "simple" types in Coq.

Represented space

A represented space **X** consists of

- An abstract base type X.
- Types of questions Q and answers A.
- Proofs that Q and A are countable and non-empty.



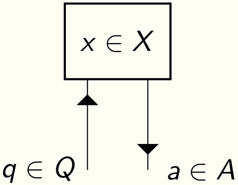
Represented spaces in Coq

Instead of encoding everything by string functions encode by "simple" types in Coq.

Represented space

A represented space **X** consists of

- An abstract base type X.
- Types of questions Q and answers A.
- Proofs that Q and A are countable and non-empty.
- A relation $\delta : (Q \to A) \to X \to Prop$.



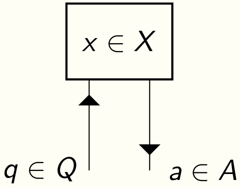
Represented spaces in Coq

Instead of encoding everything by string functions encode by "simple" types in Coq.

Represented space

A represented space **X** consists of

- An abstract base type X.
- Types of questions Q and answers A.
- Proofs that Q and A are countable and non-empty.
- A relation $\delta : (Q \to A) \to X \to Prop$.
- A proof that δ is single-valued and surjective.



Basic constructions

For represented space $\mathbf{X}$ and $\mathbf{Y}$ we can define a representation for the product $X \times Y$.

- Question space $Q_{\mathbf{X}\times\mathbf{Y}} := Q_{\mathbf{X}} + Q_{\mathbf{Y}}$.

Basic constructions

For represented space $\mathbf{X}$ and $\mathbf{Y}$ we can define a representation for the product $X \times Y$.

- Question space $Q_{\mathbf{X}\times\mathbf{Y}} := Q_{\mathbf{X}} + Q_{\mathbf{Y}}$.
- Answer space $A_{\mathbf{X}\times\mathbf{Y}} := A_{\mathbf{X}} \times A_{\mathbf{Y}}$.

Basic constructions

For represented space **X** and **Y** we can define a representation for the product $X \times Y$.

- Question space $Q_{\mathbf{X}\times\mathbf{Y}} := Q_{\mathbf{X}} + Q_{\mathbf{Y}}$.
- Answer space $A_{\mathbf{X}\times\mathbf{Y}} := A_{\mathbf{X}} \times A_{\mathbf{Y}}$.
- $\delta_{\mathbf{X}\times\mathbf{Y}}(\varphi) = (x, y) \iff$
  $\delta_{\mathbf{X}}(\mathit{lprj} \circ \varphi \circ \mathit{inl}) = x \wedge \delta_{\mathbf{Y}}(\mathit{rprj} \circ \varphi \circ \mathit{inr}) = y$.

Basic constructions

For represented space **X** and **Y** we can define a representation for the product $X \times Y$.

- Question space $Q_{\mathbf{X}\times\mathbf{Y}} := Q_{\mathbf{X}} + Q_{\mathbf{Y}}$.
- Answer space $A_{\mathbf{X}\times\mathbf{Y}} := A_{\mathbf{X}} \times A_{\mathbf{Y}}$.
- $\delta_{\mathbf{X}\times\mathbf{Y}}(\varphi) = (x, y) \iff$
  $\delta_{\mathbf{X}}(lprj \circ \varphi \circ inl) = x \wedge \delta_{\mathbf{Y}}(rprj \circ \varphi \circ inr) = y$.

Basic constructions

For represented space **X** and **Y** we can define a representation for the product $X \times Y$.

- Question space $Q_{\mathbf{X}\times\mathbf{Y}} := Q_{\mathbf{X}} + Q_{\mathbf{Y}}$.
- Answer space $A_{\mathbf{X}\times\mathbf{Y}} := A_{\mathbf{X}} \times A_{\mathbf{Y}}$.
- $\delta_{\mathbf{X}\times\mathbf{Y}}(\varphi) = (x, y) \iff$ $\delta_{\mathbf{X}}(\mathit{lprj} \circ \varphi \circ \mathit{inl}) = x \wedge \delta_{\mathbf{Y}}(\mathit{rprj} \circ \varphi \circ \mathit{inr}) = y$.

Similar construction for infinite products (sequences), functions, subspaces, hyperspaces.

Example (Cauchy reals)

```
(* A name for a real is a rational approx. function *)
Definition rep_RQ : (Q -> Q) ->> R :=
  make_mf (
  fun phi x => forall eps, 0 < Q2R eps->
        Rabs(x - Q2R(phi eps)) <= Q2R eps
   ).
```

Example (Cauchy reals)

```
(* A name for a real is a rational approx. function *)
Definition rep_RQ : (Q -> Q) ->> R :=
  make_mf (
  fun phi x => forall eps, 0 < Q2R eps->
        Rabs(x - Q2R(phi eps)) <= Q2R eps
   ).

(* f \is_cototal == forall t, exists s, (f s t) *)
Lemma rep_RQ_sur: rep_RQ \is_cototal.
(* f \is_singlevalued == forall s t t',
                (f s) t -> (f s ) t' -> t = t' *)
Lemma rep_RQ_sing: rep_RQ \is_singlevalued.
Definition RQ := (make_cs 0%Q 0%Q
                       count.Q_count count.Q_count
                       rep_RQ_sur rep_RQ_sing).
```

Example (Addition)

```
(* phi is a name for (x,y) *)
Definition Rplus_rlzrf phi (eps: Q) :=
    ((lprj phi eps/2) + (rprj phi eps/2))%Q.

Lemma Rplus_rlzr_spec:
   (F2MF Rplus_rlzrf) \realizes
     (F2MF (fun x => Rplus x.1 x.2)
     : (RQ \*_cs RQ) ->> RQ).
    [...]
```

Example (Limit)

```
(* efficient limit *)
Definition lim_eff (xn : nat -> R) (x : R) :=
          forall n, (Rabs x - (xn n)) <= (/ 2 ^ n).

(* computes lim phin for n->infinity *)
Definition lim_eff_rlzrf phin eps :=
    phin ((Pos_size (Qden eps)).+1,
      (eps / 2)%Q): Q.

(* correctness of limit *)
Lemma lim_eff_rlzr_spec:
   lim_eff_rlzrf \realizes lim_eff.
```

Exact real computation in Coq

Interval arithmetic

A representation for real numbers can be defined exactly as before using rational approximations.

However, computing with rationals is not very efficient. Alternative: approximate real numbers by intervals with dyadic endpoints.

Definition

Let $\mathbb{ID}$ be the set of intervals with dyadic endpoints. A representation $\mathbb{R}_{\mathbb{ID}}$ of the reals is given by $Q_{\mathbb{ID}} = \mathbb{N}$, $A_{\mathbb{ID}} = \mathbb{ID}$.

$$\delta_{\mathbb{R}_{\mathbb{ID}}}((I_n)_{n\in\mathbb{N}}) = x \quad \Longleftrightarrow \quad x \in \bigcap_{n\in\mathbb{N}} I_n \text{ and } \lim_{n\to\infty} |I_n| = 0.$$

Use interval arithmetic for definition of realizers.

Interval arithmetic

A representation using rational a However, compu approximate rea

Definition

Let $\mathbb{ID}$ be the s representation

$$\delta_{\mathbb{R}_{\mathbb{ID}}}((I_n)_{n\in\mathbb{I}}$$

Interval Arithmetic

$[a] = [a^-, a^+]$, $[b] = [b^-, b^+]$

$$
\begin{aligned}
[a] + [b] &= \left[a^- + b^-, a^+ + b^+\right] \\
[a] - [b] &= \left[a^- - b^+, a^+ - b^-\right] \\
[a] \times [b] &= \big[\min(a^-b^-, a^-b^+, a^+b^-, a^+b^+), \\
&\quad \max(a^-b^-, a^-b^+, a^+b^-, a^+b^+)\big]
\end{aligned}
$$

Rounded versions for efficiency.

Use interval arithmetic for definition of realizers.

Interval arithmetic in Coq

- The Coq interval library (Melquiond) provides interval arithmetic in Coq.

Interval arithmetic in Coq

- The Coq interval library (Melquiond) provides interval arithmetic in Coq.
- Main purpose of the interval library: Automatically proof inequalities over the reals.

Interval arithmetic in Coq

- The Coq interval library (Melquiond) provides interval arithmetic in Coq.
- Main purpose of the interval library: Automatically proof inequalities over the reals.

Interval arithmetic in Coq

- The Coq interval library (Melquiond) provides interval arithmetic in Coq.
- Main purpose of the interval library: Automatically proof inequalities over the reals.

Type `ID` for intervals with (arbitrary precision) floating point endpoints ($m \cdot 2^e$ with $m, e \in \mathbb{Z}$).

Interval arithmetic in Coq

- The Coq interval library (Melquiond) provides interval arithmetic in Coq.
- Main purpose of the interval library: Automatically proof inequalities over the reals.

Type `ID` for intervals with (arbitrary precision) floating point endpoints ($m \cdot 2^e$ with $m, e \in \mathbb{Z}$).

```
(* add p I J == add intervals I and J
   and round mantissas to p digits*)
I.add : SFBI2.precision -> ID -> ID -> ID
```

Interval arithmetic in Coq

Correctness in interval arithmetic:

```
Lemma add_correct_R prec x y I J:
 x \contained_in I -> y \contained_in J ->
   (x + y) \contained_in (I.add prec I J).
```

Interval arithmetic in Coq

Correctness in interval arithmetic:

```
Lemma add_correct_R prec x y I J:
 x \contained_in I -> y \contained_in J ->
   (x + y) \contained_in (I.add prec I J).
```

To define a realizer we additionally need absolute error bounds to show that intervals get arbitrarily small:

```
Lemma add_error' I J n m p x y N:
  diam I <= /2^n ->  diam J <= /2^m ->
  (x \contained_in I) -> (y \contained_in J) ->
  (Rabs x) <=  (powerRZ 2 N) -> (Rabs y) <= (powerRZ 2 N)
  -> diam (I.add p I J) <= /2 ^ n + /2 ^ m +
                          (powerRZ 2 (N+5-[p]%bigZ)).
```

Analytic functions in coq

Motivation

Numerical operators are functions from function spaces:

$$\text{integral} : C([0,1]) \to C([0,1]), f \mapsto (t \mapsto \int_0^t f(x)\,dx).$$

Motivation

Numerical operators are functions from function spaces:

$$\text{integral} : C([0,1]) \to C([0,1]), f \mapsto (t \mapsto \int_0^t f(x)\,dx).$$

A representation for $C([0,1])$ can be defined in Incone.

However, real complexity theory suggests that working with such a general space is not feasible:

- Parametric maximization $MAX(f)(x) := \max\{f(t) : 0 \leq t \leq x\}$ is NP-hard (Ko/Friedman).
- Integration is $\#\mathcal{P}$-hard (Friedman).
- Solving initial value problems for ordinary differential equations with Lipschitz-continuous right-hand side is PSPACE-hard (Kawamura).

Analytic Function

Idea: Restrict to a subset of functions where operations can be done more efficiently.

Analytic Function

Idea: Restrict to a subset of functions where operations can be done more efficiently.

Definition (Analytic Function)

$f : D \to \mathbb{C}$, $D \subseteq \mathbb{C}$ is analytic if for any $x_0 \in D$ the power series

$$T(x) := \sum_{m=0}^{\infty} a_m (x - x_0)^m$$

converges to $f(x)$ for x in a neighborhood of x_0.
A function $f : D \to \mathbb{R}$, $D \subseteq \mathbb{R}$ is called real analytic if it has a complex analytic extension.

Analytic Function

Idea: Restrict to a subset of functions where operations can be done more efficiently.

Definition (Analytic Function)

$f : D \to \mathbb{C}$, $D \subseteq \mathbb{C}$ is analytic if for any $x_0 \in D$ the power series

$$T(x) := \sum_{m=0}^{\infty} a_m (x - x_0)^m$$

converges to $f(x)$ for x in a neighborhood of x_0.
A function $f : D \to \mathbb{R}$, $D \subseteq \mathbb{R}$ is called real analytic if it has a complex analytic extension.

Many operations on analytic functions (derivatives, integrals, etc.) correspond to simple transformations of the power series.

Computing with power series

Many operators on analytic functions correspond to simple transformations on the power series so allowing to operate directly on the series seems reasonable $\rightsquigarrow$ encode power series?

Computing with power series

Many operators on analytic functions correspond to simple transformations on the power series so allowing to operate directly on the series seems reasonable $\rightsquigarrow$ encode power series?
Want to compute evaluation $((a_n)_{n\in\mathbb{N}}, x) \mapsto \sum_{i=0}^{\infty} a_n x^n$.

Computing with power series

Many operators on analytic functions correspond to simple transformations on the power series so allowing to operate directly on the series seems reasonable $\rightsquigarrow$ encode power series?
Want to compute evaluation $((a_n)_{n\in\mathbb{N}}, x) \mapsto \sum_{i=0}^{\infty} a_n x^n$.
Do not know how big the error is when only summing finitely many coefficients $\rightsquigarrow$ add this as additional information.
Following ideas from Kawamura, Rösnick, Müller, Ziegler (2013) we consider computation on power series with radius of convergence larger than 1 enriched by additional integers A, k such that

$$\forall n,\ |a_n| \leq A\left(1+\frac{1}{k}\right)^{-n}$$

.

Computing with power series

The Coquelicot library already has some useful definitions and facts about power series. We can add computational content by defining a representation.

```
(* CV_radius == radius of convergence  *)
Definition series1 a := (Rbar_lt (1%R) (CV_radius a)).
Definition series_bound a A k := (0 < k)%nat /\
  (0 < A)%nat /\ forall n, ((Rabs (a n)) <=
                      (INR A) * (/ (1+/(INR k))) ^ n)%R.
Lemma Ak_exists a : (series1 a) ->
               exists A k : nat, (series_bound a A k).
Definition powerseries1 := {a : nat -> R | series1 a }.
```

Computing with power series

The Coquelicot library already has some useful definitions and facts about power series. We can add computational content by defining a representation.

```
(* CV_radius == radius of convergence  *)
Definition series1 a := (Rbar_lt (1%R) (CV_radius a)).
Definition series_bound a A k := (0 < k)%nat /\
  (0 < A)%nat /\ forall n, ((Rabs (a n)) <=
                        (INR A) * (/ (1+/(INR k))) ^ n)%R.
Lemma Ak_exists a : (series1 a) ->
                exists A k : nat, (series_bound a A k).
Definition powerseries1 := {a : nat -> R | series1 a }.
```

Representation for `powerseries1`: real sequence + series bound

Problems

- Integer parameters for bounds quite coarse $\rightsquigarrow$ must read too many coefficients of the power series.

Problems

- Integer parameters for bounds quite coarse $\rightsquigarrow$ must read too many coefficients of the power series.
- Better more abstract representation: Can compute power series around any point + truncation error on an interval.

Problems

- Integer parameters for bounds quite coarse $\rightsquigarrow$ must read too many coefficients of the power series.
- Better more abstract representation: Can compute power series around any point + truncation error on an interval.
- Realizers can be extracted to Haskell or Ocaml code but currently this is not very efficient.

Problems

- Integer parameters for bounds quite coarse $\rightsquigarrow$ must read too many coefficients of the power series.
- Better more abstract representation: Can compute power series around any point + truncation error on an interval.
- Realizers can be extracted to Haskell or Ocaml code but currently this is not very efficient.
- How to make extraction work the way we want it to?

Conclusion and Future work

Multivariate analytic functions

The main idea is to reduce all operations to operations on power series in one variable.
Bound on power series: $|a_{i,j}| \leq Al^{i+j}$.

Multivariate analytic functions

The main idea is to reduce all operations to operations on power series in one variable.
Bound on power series: $|a_{i,j}| \leq Al^{i+j}$.

$$\begin{aligned} \sum_{i \in \mathbb{N}} \sum_{j \in \mathbb{N}} a_{i,j} x_1^i x_2^j &= \sum_{i \in \mathbb{N}} b_i x_1^i \\ \text{with } b_i &:= \sum_{j \in \mathbb{N}} a_{i,j} x_2^j \end{aligned}$$

Computing $b_i \rightsquigarrow$ evaluating an analytic function.

ODE solving

ODEs of the form

$$\dot{y}(t) = F(y(t)),\ y(0) = y_0 \in [0, 1]$$

with right-hand side function F analytic can be locally solved by computing the power series of the solution from the power series of F.

For higher precision, use more coefficients of the power series (variable order), number of coefficients grows linear in the precision.

Iterating this method gives a single-step method where the step-size does only depend on the function and not the required precision.

Conclusion and Future work

- The incone library can be used to implement real number computations in Coq and do proofs in the style of computable analysis.

Conclusion and Future work

- The incone library can be used to implement real number computations in Coq and do proofs in the style of computable analysis.
- Use more advanced methods: Affine arithmetic, taylor models, etc.

Conclusion and Future work

- The incone library can be used to implement real number computations in Coq and do proofs in the style of computable analysis.
- Use more advanced methods: Affine arithmetic, taylor models, etc.
- Next step: Complete the ODE solver in Coq

Conclusion and Future work

- The incone library can be used to implement real number computations in Coq and do proofs in the style of computable analysis.
- Use more advanced methods: Affine arithmetic, taylor models, etc.
- Next step: Complete the ODE solver in Coq
- Probably need to prove some classical mathematical facts about ODEs

Conclusion and Future work

- The incone library can be used to implement real number computations in Coq and do proofs in the style of computable analysis.
- Use more advanced methods: Affine arithmetic, taylor models, etc.
- Next step: Complete the ODE solver in Coq
- Probably need to prove some classical mathematical facts about ODEs
- Code extraction

[1] F. Steinberg, L. Théry, T. Quantitative continuity and computable analysis in Coq. Proc. of the 10th International Conference on Interactive Theorem Proving (ITP 2019).

[2] Akitoshi Kawamura, Florian Steinberg, T., Parameterized Complexity for Uniform Operators on Multidimensional Analytic Functions and ODE Solving, Proceedings of the 25th International Workshop on Logic, Language, Information, and Computation, Springer, 2018, pp. 223–236.

Thank you!

Questions, Comments, Remarks?