Advances in verified set and function calculi in Coq

Holger Thies (Kyoto University)
j.w.w. Pieter Collins (Maastricht U) and Sewon Park (Kyoto U)

September 26, 2023

Continuity, Computability, Constructivity – From Logic to Algorithms
September 25 - 29
Kyoto, Japan

This work is supported by KAKENHI grant numbers JP20K19744, JP23H03346 and JP22F22071 and has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 731143.

Motivation

- Dynamical systems mathematically model phenomena which change in time.
  - They have an internal state, which describes the system at a given moment in time, and some kind of law governing the future evolution.
  - A trajectory of a system is a particular evolution.
- Discrete-time systems are closely related to function iterations.
- Continuous-time systems are often modeled by ordinary differential equations.
- We are interested in formally verifying (computational) properties of dynamical systems.
- This requires a sound formalization of spaces of sets and functions.

Computable analysis in Coq

In recent work the authors and their collaborators have been working on verifying different parts of computable analysis and exact real computation in the Coq proof assistant.

- Incone (Steinberg, Théry, T.) is a formalization of the realizability notion of computable analysis.
- Definitions closely follow those from computable analysis.
- cAERN (Park, Konecný, T.) is a high-level axiomatic formalization of exact real computation.
- It allows extraction of efficient exact real computation programs built on top of the AERN Haskell library.
- In this talk we discuss how to combine these approaches to achieve fully verified exact real computation, in particular with applications to hyperspaces of subsets and functions.

Incone: Realizability in Coq

The Incone library

- In Incone types correspond to classical mathematical objects which are introduced axiomatically (e.g. real numbers).
- Computational content is then assigned to such objects by attaching representations and realizers performing operations on them.
- Incone thus distinguishes between algorithms/programs and their specification (although this is not strictly enforced).
- Correctness proofs can use classical mathematics.
- Computability is not formalized and only reasoned about on the meta-level in accordance to other recent works on computability in Coq.

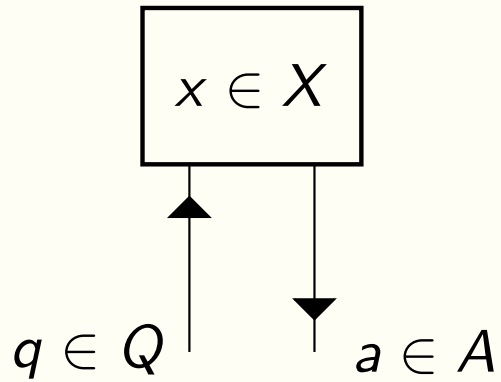
Represented spaces

Computable analysis uses explicit encodings by natural numbers or finite binary strings.
However, here we consider more general encodings by "basic types".

Represented space

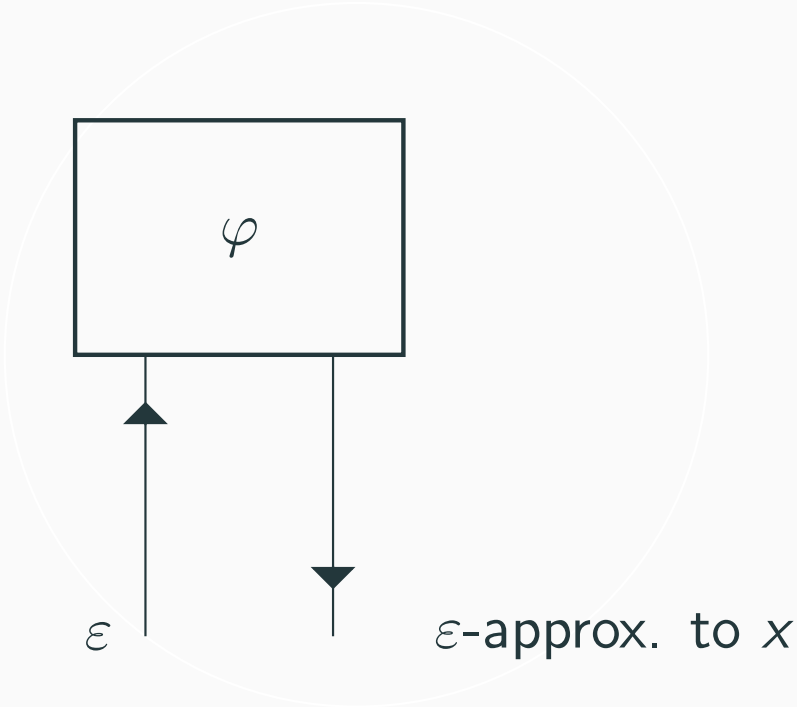
A represented space X consists of

- An abstract base type X.
- A space of names $\mathcal{B}_{\mathsf{X}} := Q \to A$ with questions Q and answers A and proofs that they are countable.
- A partial, surjective function $\delta :\subseteq \mathcal{B}_{\mathsf{X}} \to X$ called representation.



Example: $\mathbb{R}_{\mathbb{Q}}$

A representation for the reals is given by choosing questions and answers to be $\mathbb{Q}$ and $\varphi : \mathbb{Q} \to \mathbb{Q}$ is a name for $x \in \mathbb{R}$ if $\forall \varepsilon > 0, |\varphi(\varepsilon) - x| \leq \varepsilon$.



Cauchy Representation in Coq

Easy to define in Coq using the axiomatization of the reals in the standard library:
φ is name for x: $\forall \varepsilon > 0, |x - \varphi(\varepsilon)| \leq \varepsilon$.

```
(* 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.
```

Example: $\varepsilon \mapsto \varepsilon$ is a name for 0.

```
(* 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)).
```

Incone also formalizes multifunctions and realizers for multifunctions are defined in the same way.

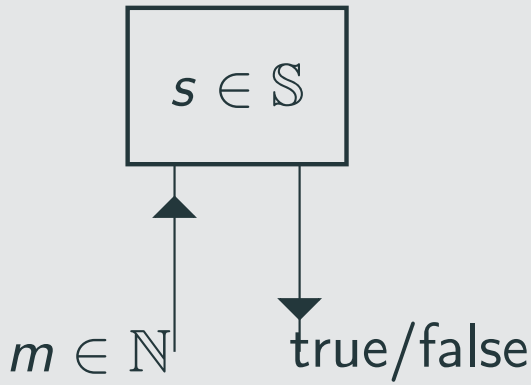
Example: Sierpiński space

Definition (Sierpiński space)

Sierpiński space is the topological space with point set $\{\top, \bot\}$ and open sets $\emptyset$, $\{\top\}$ and $\{\top, \bot\}$.

Example (Representation for Sierpiński space)

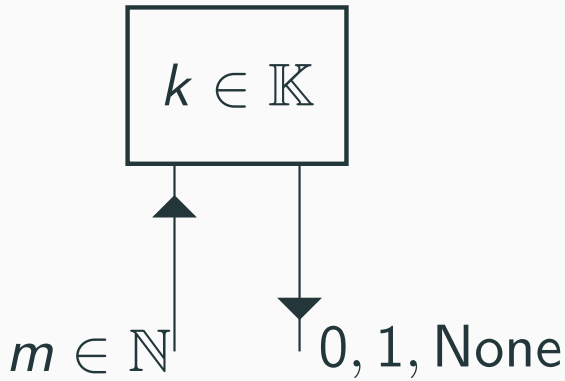
- Question space $Q_{\mathbb{S}} := \mathbb{N}$.
- Answer space $A_{\mathbb{S}} := \texttt{bool}$.
- $\delta_{\mathbb{S}}(\varphi) = \top \iff \exists n, \varphi(n) = \texttt{true}$.



The Kleeneans

Kleene's three-valued logic is the logic on $\{0, 1, \bot\}$. It can be turned into a represented space as follows.

- Question space $Q_{\mathbb{K}} := \mathbb{N}$.
- Answer space $A_{\mathbb{K}} := \{0, 1, \text{None}\}$.
- Representation $\delta_{\mathbb{K}}$ such that $\delta_{\mathbb{K}}(\varphi)$ is the first value of φ different from None or $\bot$ if no such value exists.
- Can define realizers for logical operations



The Kleeneans can be used to define a computable total real comparison $x <_{\mathbb{K}} y := \begin{cases} (x < y)_{\mathcal{B}} & \text{if } x \neq y \\ \bot_{\mathbb{K}} & \text{otherwise.} \end{cases}$

Basic constructions on represented spaces

For represented spaces X and Y we can automatically cosntruct representations for

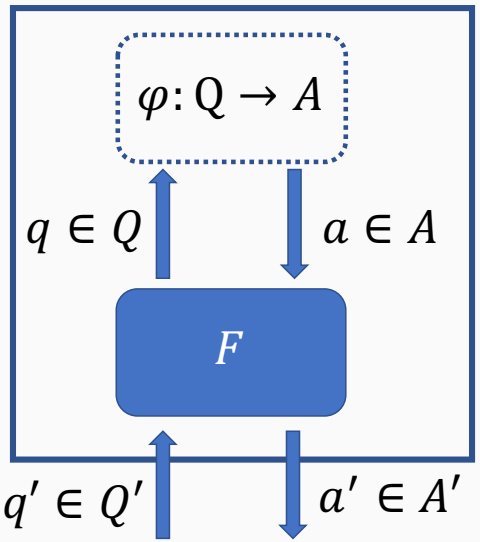
- The product space $\mathsf{X} \times \mathsf{Y}$.
- The co-product space $\mathsf{X} + \mathsf{Y}$.
- Infinite sequences X^ω.
- Spaces of subsets $A \subseteq \mathsf{X}$.
- The space of (continuous) functions $\mathsf{X} \to \mathsf{Y}$.

Continuity

Definition

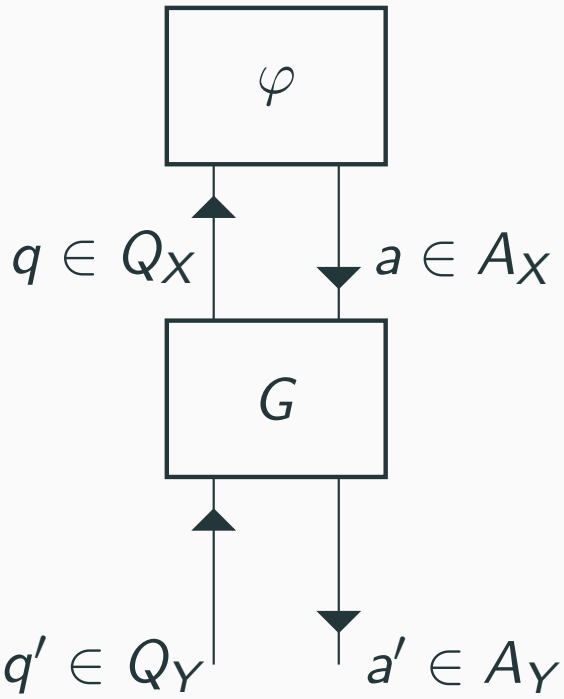
A function $F : \mathcal{B} \to \mathcal{B}'$ is continuous if for all $\varphi \in \mathcal{B}$ and $q' \in Q'$ there is a finite list $L \subseteq Q$ such that

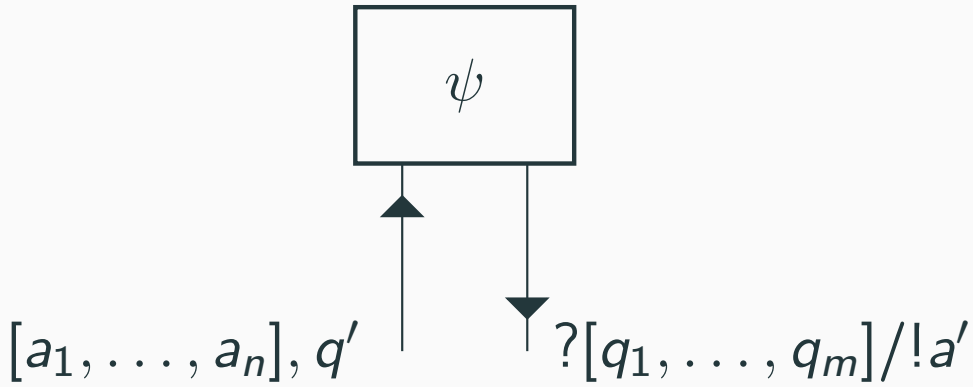
$$\forall \psi \in \mathcal{B}, \varphi \text{ and } \psi \text{ coincide on } L \Rightarrow F(\varphi)(q') = F(\psi)(q').$$



Function space construction

Assume $G : X \to Y$ is a continuous function.





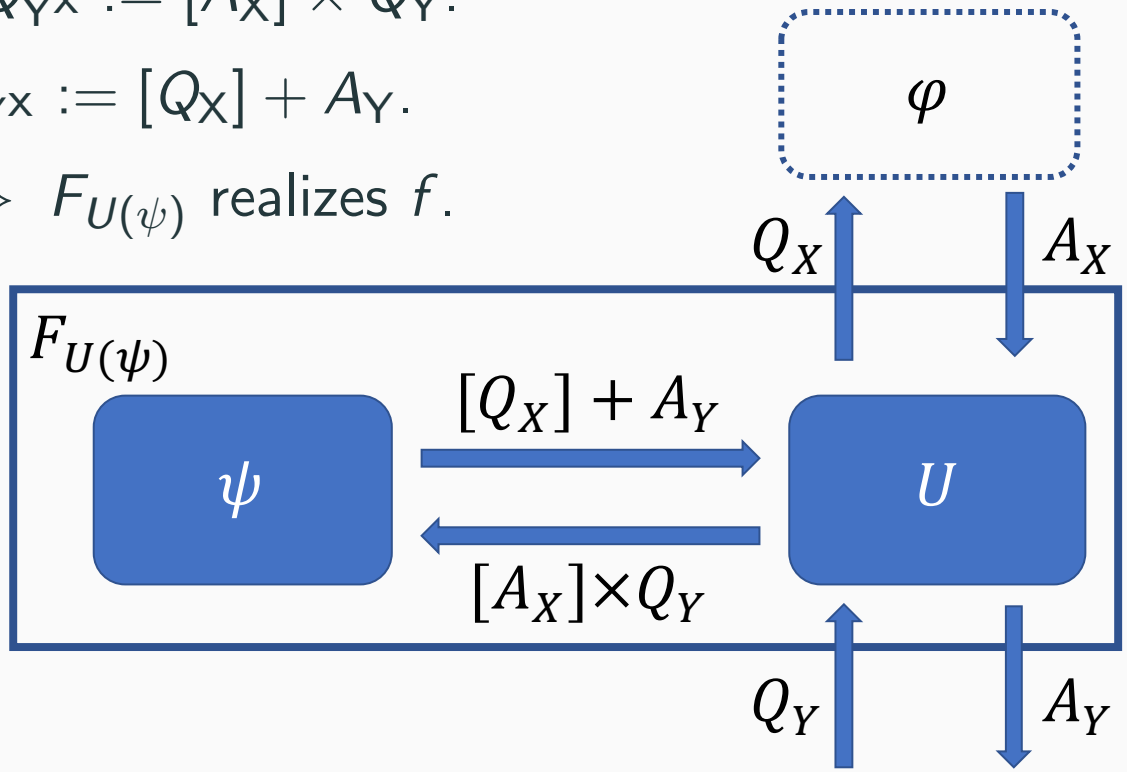
The value of $G(\varphi)(q')$ only depends on a finite list of values of φ.

We encode the communication protocol of G with φ.

Function space

We can define a representation $[\delta_X \to \delta_Y]$ for the function space Y^X of continuous functions $f : X \to Y$.The construction makes use of the existence of a universal U s.t. for any continuous $G : \mathcal{B} \to \mathcal{B}$ there is a function $\psi \in \mathcal{B}$ such that "$F_{U(\psi)} = G$".

- Question space $Q_{Y^X} := [A_X] \times Q_Y$.
- Answer space $A_{Y^X} := [Q_X] + A_Y$.
- $\delta_{Y^X}(\psi) = f \iff F_{U(\psi)}$ realizes f.



Representations for spaces of sets

The function space construction allows a general construction to get representations for spaces of subsets of a represented space. Let $\mathsf{X} = (X, \delta_{\mathsf{X}})$ be a represented space.

The space of open subsets

We can identify a subset U of X with its characteristic function

$$\chi_U : X \to \mathbb{S}, \quad \chi_U(x) := \begin{cases} \top & \text{if } x \in U, \\ \bot & \text{otherwise.} \end{cases}$$

Note that χ_U is continuous iff U is open. We therefore can identify the space $\mathcal{O}(\mathsf{X})$ of open subsets of X with the function space $\mathbb{S}^{\mathsf{X}}$.

Execution and Program Extraction

- Realizers can be executed inside the Coq proof assistant or extracted to Ocaml/Haskell programs.
- Some difficulties executing in Coq when allowing unbounded search.
- Extracted programs are not very efficient/readable.
- Difficult to combine with nonverified code.

cAERN: Axiomatized exact real computation and program extraction

Axiomatized exact real computation

In exact real computation frameworks users do not really need to deal with realizers, instead higher-level datatypes for reals and other continuous objects are provided by the framework.

For example, AERN provides real number arithmetic and can output approximations up to any desired precision:

```
    ...> pi + pi*pi+2^(-3) ? (bits 60)
  [13.136197054679151857297107 6... ± 5.0568e-23 2^(-74)]
```

Comparisons return an object of type Kleenean:

```
...> pi > pi + 2^(-100) ? (prec 50)
TrueOrFalse

...> (pi > pi + 2^(-100)) ? (prec 1000)
CertainFalse
```

cAERN

- The cAERN library reflects the idea of using an exact real computation framework by axiomatically introducing types for basic objects in exact real computation.
- Thus, types are meant to encode represented spaces without making the representation explicit.
- The types and operations that are axiomatized are inspired by the AERN library.
  - Introduce a type K for Kleeneans with elements true, false, $\bot$.
  - Introduce a type R for real numbers with standard arithmetic operations and Kleenean comparison operator.
  - Introduce computationally valid axioms such as $\lim s : \forall(n, m).\ |s_n - s_m| < 2^{-n-m} \to \exists r : \mathrm{R} \forall(k).\ |r - s_k| < 2^{-k}$
- Coq's code extraction features can be modified to extract AERN programs from proofs in cAERN.

Spaces of subsets and functions

- We can define Sierpinski space as a type in cAERN and then define open subsets of a space X by

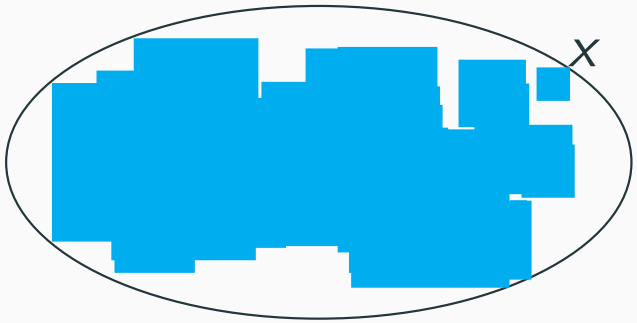
$$\text{open } A :\equiv \exists(f : X \to \mathsf{S}).\ \forall(x : X).\ f(x) = \top \leftrightarrow x \in A\ .$$

- Note that constructing a function $f : \mathsf{R} \to \mathsf{S}$ can be extracted to an implementation of this function in AERN.
- Thus, any function we construct is continuous.
- As in AERN real number computation is internally performed on sequences of intervals with rational endpoints, we formulate continuity in terms of a rational interval extension.
- For a function $f : \mathsf{R} \to \mathsf{S}$, there is an interval extension $F : \mathbb{I}_{\mathbb{Q}} \to \text{bool}$ such that
  - If $F(I) = \text{True}$, then $\forall x \in I, f(x) = \top$, and
  - If $f(x) = \top$, then there is an $I \ni x$ such that $F(I) = \text{True}$.

Alternative representations for Euclidean space

For subsets of Euclidean space we can define more efficient representations exploiting the metric structure of the space.
$A \subseteq \mathbb{R}^d$ is Euclidean open if there is a sequence $C : \mathbb{N} \to$ ball of (possibly empty) balls such that

1. $\forall n,\ C_n \subseteq A$,
2. $\forall x \in A,\ \exists n,\ x \in C_n$.



We can show that a subset $A \subseteq \mathbb{R}^d$ is euclidean open iff it is open.

- Checking if x contained in $B(y, r)$ is semidecidable, thus we can check all balls in parallel to find out if $x \in A$.
- The other direction follows from the continuity of $f : \mathrm{R} \to \mathrm{S}$.

More efficient function spaces

- The interval extension lets us compute operations on functions such as computing images of subsets etc.
- However, this usually involves some unbounded search and is thus highly inefficient.
- For more efficient computations, we would like to consider other representations of functions.
- Pieter Collins' verified rigorous numerics library provides polynomial models that represent function types.
- We plan to integrate this approach in the cAERN library.

Conclusion and Future Work

- Combining Incone and cAERN we can both verify basic operations of exact real computation and reason over high-level specifications.
- We can extract efficient AERN programs from proofs over abstract types.
- For working with subsets and function spaces, representing them in an efficient way is crucial.
- In future work, we plan to put the different approaches together to built a unified library for exact real computation.

Thank you!

References

- Github repository:
  `http://www.github.com/holgerthies/coq-aern/`
  (new-subsets branch)
- Michal Konečný, Sewon Park, and Holger Thies.
  Axiomatic Reals and Certified Efficient Exact Real Computation.
  WoLLIC 2021. Springer, 2021.
- Michal Konečný, Sewon Park, and Holger Thies.
  Certified Computation of Nondeterministic Limits.
  NASA Formal Methods, 14th International Symposium, 2022.
- Michal Konečný, Sewon Park, and Holger Thies.
  Extracting efficient exact real number computation from proofs in constructive type theory.
  arxiv preprint, 2022.

Experimental results

Benchmark		Average execution time (s)			
Formula	Accuracy	Extracted	Hand-written	Native	iRRAM
$\max(0, \pi - \pi)$	10^6 bits	3.5	3.8	3.8	1.59
$\sqrt{2}$	10^6 bits	0.72	0.70	0.40	0.62
$\sqrt{\sqrt{2}}$	10^6 bits	1.52	1.38	0.85	1.15
$x - 0.5 = 0$	10^3 bits	1.44	0.32	—	0.03
$x(2 - x) - 0.5 = 0$	10^3 bits	2.02	0.35	—	0.04
$\sqrt{x + 0.5} - 1 = 0$	10^3 bits	12.9	2.35	—	0.29

(i7-4710MQ CPU, 16GB RAM, Ubuntu 18.04, Haskell Stackage LTS 17.2)