Formalizing Hyperspaces for Extracting Efficient Exact Real Computation

Michal Konečný
School of Computing, Aston University, UK

Sewon Park
Research Institute for Mathematical Sciences, Kyoto University, Japan

Holger Thies
Graduate School of Human and Environmental Science, Kyoto University, Japan

Abstract

We propose a framework for certified computation on hyperspaces by formalizing various higher-order data types and operations in a constructive dependent type theory. Our approach builds on our previous work on axiomatization of exact real computation where we formalize nondeterministic first-order partial computations over real and complex numbers. Based on the axiomatization, we first define open, closed, compact and overt subsets in an abstract topological way that allows short and elegant proofs with computational content coinciding with standard definitions in computable analysis. From these proofs we extract programs for testing inclusion, overlapping of sets, et cetera.

To improve extracted programs, our framework specializes the Euclidean space $\mathbb{R}^m$ making use of metric properties. To define interesting operations over hyperspaces of Euclidean space, we introduce a nondeterministic version of a continuity principle valid under the standard type-2 realizability interpretation. Instead of choosing one of the usual formulations, we define it in a way similar to an interval extension operator, which often is already available in exact real computation software.

We prove that the operations on subsets preserve the encoding, and thereby define a small calculus to built new subsets from given ones, including limits of converging sequences with regards to the Hausdorff metric. From the proofs, we extract programs that generate drawings of subsets of $\mathbb{R}^m$ with any given precision efficiently. As an application we provide a function that constructs fractals, such as the Sierpinski triangle, from iterated function systems using the limit operation, resulting in certified programs that errorlessly draw such fractals up to any desired resolution.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Computable analysis, type theory, program extraction

Digital Object Identifier 10.4230/LIPIcs.MFCS.2023.59

Supplementary Material *Software*: `https://github.com/holgerthies/coq-aern`
archived at `swh:1:dir:6bfc91ee518681f328e69d08125f48ac0db1ffad`

Funding This project has received funding from the EU's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 731143.
Sewon Park: Supported by JSPS KAKENHI (Grant-in-Aid for JSPS Fellows) JP22F22071 as a JSPS International Research Fellow.
Holger Thies: Supported by JSPS KAKENHI Grant Numbers JP20K19744 and JP23H03346.

Acknowledgements The authors thank Pieter Collins, Arno Pauly and Hideki Tsuiki for interesting discussions.

1 Introduction

In exact real computation, real numbers are often presented as abstract data type hiding tedious multi-precision computations used to eliminate rounding errors in the background from the user. Extensionally to the user, the real numbers resemble the classical structure of real numbers which makes reasoning procedures more intuitive, emancipating the programmers from tracking artificial numerical errors. The theoretical framework to study computability of algorithms in this context is computable analysis [16, 28]. Recent developments in implementations demonstrate the usefulness of this approach in practice [1, 20, 15].

The project cAERN [14] aims to provide an axiomatic formalization of exact real number computations in a constructive type theory, modeling real numbers similar as observed by users of exact real computation software; cf., [2, 25] where explicit representations are required for computations. Instead of being constructed as sequences of approximations, real numbers are axiomatized in a way that two reals are provably equal when they represent the same numbers; cf., [3, 9, 26]. This can be regarded as a modification of the classical real numbers by replacing uncomputable operations (e.g., rounding to nearest integers) and computably invalid axioms (e.g., the law of trichotomy) with their computable variants. As the axiomatization is representation irrelevant, reals appear similar to the classical real numbers allowing many classical results to be transported into our setting (see [14, Section 6]). At the same time, by allowing only computably valid axioms, it admits a program extraction mechanism built on top of an existing exact real computation framework.

The extraction is achieved by mapping the axiomatic real type R to the abstract data type of real numbers `CReal` and axioms to primitive operations in AERN, a Haskell library for exact real number computation which is being developed by one of the authors [15]. Thus, we do not need to focus on the concrete representation of real numbers and its efficient implementation which is challenging on its own (see e.g. [19]).

More concretely, in our system a proof of a theorem of the form

$$\Pi(x : \mathsf{R}).\ \mathsf{M}\Sigma(y : \mathsf{R}).\ P\ x\ y$$

yields a nondeterministic AERN function of type $\texttt{CReal} \to \texttt{CReal}$. Here, $\mathsf{M} : \mathsf{Type} \to \mathsf{Type}$ formalizes nondeterminism, which is an inevitable effect of identifying real numbers extensionally [17].[1]

The theory formalized in cAERN is sufficient to build interesting first-order nondeterministic functions [13]. Many applications in exact real computation, e.g. reachability questions for dynamical systems, computing integrals and solving differential equations, however deal with higher order objects such as hyperspaces of functions and subsets as primitive data type [8]. Such applications are often safety-critical and a framework for formally proving correctness of algorithms on such higher-order objects is thus highly desirable.

In this paper, we extend our previous axiomatic formalization towards higher-order exact real computations. Based on the axiomatization of real numbers, partial lazy Boolean, and nondeterminism, we define various higher-order types including Euclidean spaces, classical subsets, open subsets, closed subsets, compact subsets and overt subsets. Furthermore, we provide various operations on these sets, including limit operations on compact sets based on Hausdorff distance. We further provide a rich theory of subsets of Euclidean spaces and operations on them, encoding them in a way that lets us efficiently generate verified drawings.

An important fact in computable analysis is that every computable function is continuous. When we operate on higher-order objects, this fact becomes computationally relevant. To make continuity available in our system, we include an axiomatic formalization of the continuity principle saying that any function from real numbers *nondeterministically* admits a modulus of continuity function. As the goal of the project is to use axioms to model

[1] Nondeterminism in this context is also called multivaluedness [28] or non-extensionality [5].

functionality that is typically available in exact real computation, instead of assuming a continuity principle by saying that any functional $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ is continuous and then derive various continuity lemmas on $X \to Y$ by reasoning on specific constructions of the types X and Y, decomposing them to the natural numbers, we formalize the continuity principle directly on our axiomatic types in terms of an interval extension. Such an operation is natural in exact real computation software and our continuity principle can be extracted to a simple operation in AERN.

Using the continuity principle, we show that our efficient encoding of euclidean subsets is computationally equivalent to a more general way to represent subsets. We further show that it can be used to generate verified drawings of images of functions on certain subsets.

As an application we devise a general construction of fractals defined by a recurrence relation on compact sets. The recurrence relation can be used to define the structure, which then can be drawn exactly up to any desired resolution using the drawing operation. We further can use the limit operator for a simple and elegant formulation of the theorem.

This paper is organized as follows. In Section 2 we briefly summarize the background from our previous work, introduce notations used throughout the paper and explain some extensions to the theory necessary for the current work. In Section 3 we define the continutity principle and prove some of its consequences. We define open, closed, compact and overt subsets in a generic setting and prove some of the properties in Section 4. We consider subsets of Euclidean space for which we use specialized encodings in Section 5 and finally show some applications to generate certified drawings of two-dimensional sets in Section 6.

All statements in this paper except for those in Appendix A have been implemented and fully formally verified in the Coq proof assistant as part of the cAERN library[2]. We extend the program extraction mechanism so that the newly added axioms, including the continuity principle, are mapped to appropriate AERN functions. The constructive content of the proofs can thus be extracted to AERN programs using Coq's Haskell program extraction functionality and mapping axiomatically defined operations to basic operations in AERN providing a certified module in AERN for defining and manipulating higher-order objects.

2 Preliminaries

2.1 Type Theory and Realizability

As in [14] we assume to work in a simple type theory with basic types $\mathsf{0}, \mathsf{1}, \mathsf{2}, \mathsf{N}, \mathsf{Z}$, a universe of classical propositions Prop, and a universe of types Type. We assume that the identity types $=$ are in Prop and that Prop is a type universe closed under $\to, \times, \vee, \exists, \Pi$, containing two types $\mathsf{True}, \mathsf{False} : \mathsf{Prop}$ which are the unit and the empty type respectively. We denote by $P \vee Q : \mathsf{Prop}$ the classical fact that P or Q holds, and by $P + Q : \mathsf{Type}$ the fact that there is a computational procedure deciding if P or Q holds. Similarly, when we have a family of classical propositions $P : X \to \mathsf{Prop}$, the type $\exists(x : X).\ P\,x : \mathsf{Prop}$ belonging to Prop denotes the classical existence of $x : X$ satisfying $P\,x$ while the ordinary dependent pair type (also called Σ-type), $\Sigma(x : X).\ P\,x : \mathsf{Type}$ belonging to Type denotes the constructive existence.

We assume enough axioms that make Prop indeed a universe of classical propositions and that are valid under this interpretation. That includes the law of excluded middle

$$\Pi(P : \mathsf{Prop}).\ P \vee \neg P$$

[2] `https://github.com/holgerthies/coq-aern`

where $\neg P$ is defined by $P \to \mathsf{False}$ and propositional extensionality

$$\Pi(P, Q : \mathsf{Prop}).\ (P \to Q) \to (Q \to P) \to P = Q.$$

These axioms allow us to reason classically when we are dealing with classical statements such as $\exists(x : X).\ P\,x : \mathsf{Prop}$. Furthermore, we assume a general (dependent) function extensionality

$$\Pi(P : X \to \mathsf{Type}).\ \Pi(f, g : \Pi(x : X).\ P\ x).\ (\Pi(x : X).\ f\ x = g\ x) \to f = g.$$

Based on this foundational setting of a constructive type theory we axiomatically formalize various types and operations for real number computation that are briefly introduced in the next subsections. The soundness of the basic setting of type theory and the newly introduced axioms are justified by extending a realizability interpretation in the category of assemblies over Kleene's second algebra [11, 23] by mapping each axiom to a computable operation in computable analysis.

2.2 Kleenean and Sierpinski Spaces

One important characteristic of exact real computation, besides exact operations, is nontermination. As real numbers are expressed exactly using infinite representations, comparing real numbers is a partial operation where testing $x < y$ does not terminate when $x = y$ regardless of the specific representations that are used [28, Theorem 4.1.16]. To deal with such partiality, instead of making computations continue indefinitely when the same numbers are compared, AERN and other exact real computation software provide a data type for lazy Boolean $\{\mathit{ff}, \mathit{tt}, \bot\}$, generalizing Booleans by adding a third element $\bot$ as an explicit state of divergence. The third value $\bot$ is an admissible value that a variable can store whose nontermination is delayed until it is required to test whether the value is tt or ff.

Observing that the space satisfies the algebraic structure of Kleene's three-valued logic where $\bot$ is the third truth value for indeterminacy, the data type is also often called *Kleenean*. In particular, we assume that there are three distinct constants $\mathsf{true}, \mathsf{false}, \mathsf{bot} : \mathsf{K}$, although we do not assume that they are the only elements by adding an induction rule. We write $\lceil k \rceil$ for $k = \mathsf{true}$.

Later we often need to deal with infinite sequences of Kleenean expressions. This has not yet been considered in our previous work and requires to characterize the Kleeneans further by assuming some additional axioms. The *nondeterministic countable choice principle* is an axiom of the following type:

$$\Pi(x : \mathsf{N} \to \mathsf{K}).\ \left(\exists(n : \mathsf{N}).\ \lceil x\ n \rceil \to \mathsf{M}\Sigma(n : \mathsf{N}).\ \lceil x\ n \rceil\right). \tag{1}$$

Here, M is a nondeterminism monad such that for any type $X : \mathsf{Type}$, we automatically get a type $\mathsf{M}X : \mathsf{Type}$ modelling the result of a nondeterministic computation in X. The axiom says whenever we have a sequence of Kleenean terms x and know that there (classically) is at least one index n such that $x\ n$ is true, then we can nondeterministically find such an index.

Sometimes we only need to know if there is an element of the sequence that is true. For this, we add the following axiom.

$$\Pi(x : \mathsf{N} \to \mathsf{K}).\ \Sigma(k : \mathsf{K}).\ \left(\lceil k \rceil \leftrightarrow \exists(n : \mathsf{N}).\ \lceil x\ n \rceil\right).$$

That is, we can find a $k : \mathsf{K}$ that is true iff any of the sequence elements is true and can be false or bot otherwise. To see that the axiom is valid, it suffices to show that such a k is computable which can be done by iterating over all the outputs of the realizers of $x\ n$ in parallel.

A space similar to the Kleenean that is more commonly used in computability theory is the Sierpinski space. Sierpinski space is the topological space $\{\downarrow, \uparrow\}$ whose only nontrivial open is $\{\downarrow\}$. In a certain way, $\mathbf{K}$ can be seen as an extension of Sierpinski space by adding another decidable element false. However, as the type of Kleenean K already exists in our system, we instead consider Sierpinski as a subtype of K by defining it as

$$\mathsf{S} :\equiv \Sigma(b : \mathsf{K}).\ b \neq \mathsf{false}$$

and name $\downarrow :\equiv (\mathsf{true}, t_\downarrow)$ and $\uparrow :\equiv (\mathsf{bot}, t_\uparrow)$ where the $t_\downarrow$ is the unique proof term for $\mathsf{true} \neq \mathsf{false}$ and $t_\uparrow$ is the unique proof term for $\bot \neq \mathsf{false}$. The uniqueness of the proof terms is due to the Prop axioms. From the computational point of view, S can be thought of as K where it is promised that false does not appear.

As a further axiom, we assume that the first projection $\mathsf{proj}_1 : \mathsf{S} \to \mathsf{K}$ admits a retraction $\mathsf{KtoS} : \mathsf{K} \to \mathsf{S}$. Observe that the assumption can be validated by a program on the Kleenean that on its input $b : \mathsf{K}$ checks if $b = \mathsf{true}$ or false and simply diverges when $b = \mathsf{false}$.

2.3 Real Numbers and Euclidean Space

We assume real numbers by declaring that there is a type R for real numbers containing two distinct constants 0 and 1, the standard arithmetical operators and a semi-decidable K-valued comparison operator. From this axiomatization we can define further operations such as $\max(x, y)$, $|x|$, etc. Again, the full axiomatization can be found in [14]. For any $m\colon \mathsf{N}$ we further define the type R^m simply as an m-element list of R with maximum norm $\|\cdot\|$, and define the vector space operations point-wise.

We often approximate real numbers by elements of a countable dense subset. We call a type X enumerable if there exists a map $f : \mathsf{N} \to X$ that is surjective in the classical sense. We define dyadic numbers by pairs $\mathsf{D} := \mathsf{Z} \times \mathsf{N}$ and identify the pair (z, n) with the real $z \cdot 2^{-n}$. It is not hard to show that D is enumerable. Similarly we write D^m for the m-element list of D when $m : \mathsf{N}$ and suppose that the trivial coercion from D^m to R^m is taken implicitly.

We use dyadic numbers to approximate real numbers. In our theory, we can prove that D is indeed dense in R in the sense that for any real number x, we can nondeterministically find a dyadic rational approximation up to any prescribed error 2^{-p}. More generally, we can prove the following statement in the Euclidean space:

$$\Pi(x : \mathsf{R}^m).\ \Pi(p : \mathsf{N}).\ \mathsf{M}\Sigma(d : \mathsf{D}^m).\ \|x - d\| \leq 2^{-p}.$$

3 Continuity

Exact real computation software often internally represents real numbers by converging sequences of intervals with dyadic rational endpoints. Thus, for any function $f : \mathsf{R} \to \mathsf{S}$, whenever $(f\ x)\downarrow$, there is some dyadic interval containing x that made the software computing $f\ x$ return $\top$. We thus formulate continuity on an abstract level similar to the above without referring to specific constructions of continuous types.

The Sierpinski space has two open subsets $\{\downarrow\}$ and $\{\downarrow, \uparrow\}$. An interval extension of a function $f : \mathsf{R} \to \mathsf{S}$ is a discrete function that indicates in which of the two open sets all values for f lie for any dyadic interval. As there are exactly two values, we can represent $\{\downarrow\}$ and $\{\downarrow, \uparrow\}$ space by the Boolean constants true and false, respectively. We can then see a map $\hat{f}$ of type $\mathsf{D} \times \mathsf{N} \to \mathsf{B}$ as a set-valued function from dyadic intervals: $\hat{f}$ returns a subset of Sierpinski space on its argument $(d, n) : \mathsf{D} \times \mathsf{N}$ representing the interval $(d - 2^{-n}, d + 2^{-n}) \subset \mathbb{R}$.

For a map $f : \mathsf{R} \to \mathsf{S}$, we say that a set-valued function on dyadic intervals $\hat{f}$ is an interval extension of f when it satisfies the inclusion property:

$$\mathrm{incl}\ \hat{f} :\equiv \Pi(d : \mathsf{D}).\ \Pi(n : \mathsf{N}).\ \hat{f}\ (d, n) = \mathsf{true} \to \big(\Pi(x : \mathsf{R}).\ |x - d| \le 2^{-n} \to (f\ x){\downarrow}\big)\ .$$

In words, whenever the interval extension $\hat{f}$ of f returns true on the pair (d, n), $(fx){\downarrow}$ for all $x \in (d - 2^{-n}, d + 2^{-n})$.

However, not all interval extensions are interesting as the inclusion property holds also for the trivial map $\hat{f}$ that returns false for all inputs. We say that a map $\hat{f} : \mathsf{D} \times \mathsf{N} \to \mathsf{B}$ seen as an interval extension of $f : \mathsf{R} \to \mathsf{S}$ is tight if whenever $(f\ x){\downarrow}$, there (classically) exists an interval containing x for which $\hat{f}$ indicates that $\downarrow$ is the only valid answer on that interval:

$$\mathrm{tight}\ \hat{f} :\equiv \Pi(x : \mathsf{R}).\ (f\ x){\downarrow} \to \exists(d : \mathsf{D}).\ \exists(n : \mathsf{N}).\ |x - d| < 2^{-n} \wedge \hat{f}\ (d, n) = \mathsf{true}$$

Our continuity principle is the following axiom saying that for any map $f : \mathsf{R} \to \mathsf{S}$, there nondeterministically exists an interval extension of it that is tight:

$$\Pi(f : \mathsf{R} \to \mathsf{S}).\ \mathsf{M}\Sigma(\hat{f} : \mathsf{D} \times \mathsf{N} \to \mathsf{bool}).\ \mathrm{incl}\ \hat{f} \wedge \mathrm{tight}\ \hat{f}.$$

It is realized by the trivial operation that given f extracts the realizer $\hat{f}$ of f assuming that the underlying exact real computation is done via dealing real numbers by converging sequences of intervals with dyadic rational endpoints. Since a single function admits several different realizers, the procedure of extracting realizers is nondeterministic.

From the above, we can derive the more standard form of the continuity principle which says that there exists a modulus function:

$$\Pi(f : \mathsf{R} \to \mathsf{S}).\ \Pi(x : \mathsf{R}).\ (f\ x){\downarrow} \to \mathsf{M}\Sigma(n : \mathsf{N}).\ \Pi(y : \mathsf{N}).\ |x - y| < 2^{-n} \to (f\ y){\downarrow}\ .$$

To derive this version of the principle, assume that we are given x with $f\ x{\downarrow}$. Due to the continuity axiom, we further have nondeterministically a tight interval extension $\hat{f}$. From the tightness, we then know that there (classically) exist d, n such that $|d - x| < 2^{-n}$ and $\hat{f}\ (d, n) = \mathsf{true}$. Though $\hat{f}$ is Boolean-valued, we can post-compose the embedding from B to S and use the nondeterministic countable choice principle in Equation (1) to nondeterministically find such d and n. We then (again nondeterministically) choose some m such that $2^{-m} < 2^{-n} - |x - d|$. As for any y with $|x - y| < 2^{-m}$, $|y - d| < 2^{-n}$ holds, it is guaranteed that $f\ y{\downarrow}$ by the properties of the interval extension. Thus, any such m fulfills the condition.

Further, we can derive a continuity principle for real valued functions $f : \mathsf{R} \to \mathsf{R}$:

$$\Pi(x : \mathsf{R}).\ \Pi(m : \mathsf{N}).\ \mathsf{M}\Sigma(n : \mathsf{N}).\ \Pi(y : \mathsf{N}).\ |x - y| < 2^{-n} \to |(f\ x) - (f\ y)| < 2^{-m}$$

To prove that this continuity principle holds, take any $f : \mathsf{R} \to \mathsf{R}$, $x : \mathsf{R}$, and $m : \mathsf{N}$, and define a function $g_{x,m} : \mathsf{R} \to \mathsf{S}$ by $g_{x,m} :\equiv \lambda(y : \mathsf{R}).\ |(f\ x) - (f\ y)| < 2^{-m}$. Here, we let $(x < y)$ be the embedding of the Kleenean valued comparison to Sierpinski space, i.e. such that $(x < y){\downarrow}$ if and only if $x < y$ holds. The continuity principle on Sierpinski valued functions yields

$$|(f\ x) - (f\ x)| < 2^{-m} \to \mathsf{M}\Sigma(n : \mathsf{N}).\ \Pi(y : \mathsf{R}).\ |x - y| < 2^{-n} \to |(f\ x) - (f\ y)| < 2^{-m}\ .$$

As $|(f\ x) - (f\ x)| < 2^{-m}$ is trivially true, we get the desired bound.

As a last application, consider any discrete type X having decidable equality $\Pi(x, y : X).\ (x = y) + (x \neq y)$. We can then prove that any $f : \mathsf{R} \to X$ is constant: $\Pi(x : \mathsf{R}).\ \Pi(y : \mathsf{R}).\ f\ x = f\ y$. Similarly as above, assuming $x : \mathsf{R}$, we define a function g_x where $(g_x\ y)$ is defined if $(f\ x) = (f\ y)$ and undefined if $(f\ x) \neq (f\ y)$, constructed based on the decidability of elements of X. Again, applying the continuity principle, we get

$$\mathsf{M}\Sigma(n : \mathsf{N}).\ \Pi(y : \mathsf{R}).\ |x - y| < 2^{-n} \to (f\ x) = (f\ y)\ .$$

Since we can prove that any locally constant function from R is constant in the classical part of our system, we can conclude that f is constant.

All results can be easily generalized to arbitrary Euclidean space R^m by replacing the absolute value with the maximum norm and proofs are contained in our formal development.

4 Subsets

For any type X, we first express subsets classically as predicates $X \to \mathsf{Prop}$. That is, we set

$$\mathrm{csubset}(X) :\equiv X \to \mathsf{Prop}$$

and define classical operations on classical subsets such as $\in, \cup, \cap, \subseteq, \dots$ in the obvious way. For example, $(x \in S) :\equiv S\ x$, $S \cap T :\equiv \lambda(x : X).\ x \in S \wedge x \in T$, and so on.

Our goal is to assign computational content to classically defined subsets which in turn allows to extract programs that perform operations on these sets. In computable analysis, computational content is given by defining representations for certain spaces of subsets [22, 8] and perform computations on these representations. This can be done in a very general setting without restricting to specific spaces, although the computational procedures are usually extremely inefficient and not useful for practical applications.

While we are mostly interested in subsets of Euclidean spaces for which we can specialize the theorems and get more efficient algorithms, the general results are still interesting mathematically and can be used for basic facts that are not meant to be used computationally.

Classically, a subset $A \subseteq X$ of a topological space X is open iff its characteristic function $\chi_A : X \to \mathbb{S}$ is continuous. We thus identify open sets with their characteristic function:

$$\mathrm{open}\ A :\equiv \Sigma(f : X \to \mathsf{S}).\ \Pi(x : X).\ (f\ x)\!\downarrow \leftrightarrow\ x \in A\ .$$

Similarly, we define closed sets as their complement

$$\mathrm{closed}\ A :\equiv \Sigma(f : X \to \mathsf{S}).\ \Pi(x : X).\ (f\ x)\!\downarrow \leftrightarrow\ x \notin A\ .$$

Computationally for an open set we can verify (i.e. semi-decide) if a point is contained in the set and for a closed set we can verify if a point is outside.

We state some simple properties of open and closed sets. We omit the proofs here, but pen and paper proofs can e.g. be found in [22] and full formal proofs can be found in our Coq development.

1. If $(A_i)_{i\in\mathbb{N}}$ are open then so is the countable union $\bigcup A_i$.
2. If A and B are open then so is their finite intersection $A \cap B$.
3. If A and B are closed then so is their finite union $A \cup B$.
4. If $(A_i)_{i\in\mathbb{N}}$ are closed then so is the countable intersection $\bigcap A_i$.

Other classes of subsets that are often considered in computable analysis are the compact and overt subsets. (Computably) compact subsets are those for which we can effectively verify that an open set covers them. That is, a subset $A \subseteq X$ is compact iff

$$\Sigma(f : \mathrm{open}\ X \to \mathsf{S}).\ \Pi(U : \mathrm{open}\ X).\ (f\ U)\!\downarrow \leftrightarrow\ A \subseteq U.$$

Overtness is a dual notion to compactness. For a (computably) overt subset, we can effectively verify if an open set touches it. That is, a subset $A \subseteq X$ is overt iff

$$\Sigma(f : \text{open } X \to \mathsf{S}).\ \Pi(U : \text{open } X).\ (f\ U)\downarrow \leftrightarrow\ A \cap U \neq \emptyset.$$

The notion of overtness is less known to classical mathematicians as it does not carry any information outside a constructive or computational context (i.e. any set is classically overt).

Another way to look at these notions is that for a compact subset A it can be verified that a semidecidable property P holds for each point, i.e. if $\forall x \in A, P\ x$ and for an overt subset it can be verified that it holds for at least one point, i.e. if $\exists x \in A, P\ x$.

Similar as for open and closed sets, we prove several properties of compact and overt sets in our Coq development but omit the details here for space reasons.

5 Subsets of Euclidean Space

Using basic properties of the Kleenean and Sierpinski spaces, we get short and elegant proofs for simple properties of open sets. However, from the point of view of doing actual computations, using Sierpinski valued functions to represent basic objects is often far from optimal and programs extracted from the proofs are inefficient. As we are interested in extracting exact real computation programs, our main focus is on subsets of Euclidean spaces. Most statements in this section can be proven in a much more general setting (and our Coq development contains some of them). However, the proofs in this section are optimized to encode efficient algorithms by making use of particular properties of the representation.

5.1 Characterization of Euclidean Subsets

We prove the following characterization for subsets $A \subseteq \mathbb{R}^m$ of Euclidean space:

$$\begin{array}{rcl} \text{open } A & \leftrightarrow & \mathsf{M}\Sigma(F : \mathsf{N} \to \mathsf{R}^m \times \mathsf{R}). \\ & & \Pi(n : \mathsf{N}).\ \mathrm{B}((F\ n)) \subseteq A \\ & \wedge & \Pi(x : \mathsf{R}^m).\ x \in A \to \exists(n : \mathsf{N}).\ x \in \mathrm{B}((F\ n))). \end{array}$$

Here, $\mathrm{B}(x, r)$ encodes a ball with radius r around x. That is, a subset $A \subseteq \mathbb{R}^m$ is open iff we can find a sequence of (possibly empty) balls, all contained in A, that eventually cover all A.

To show the equivalence let us call the statement on the right of the equivalence Euclidean open. Thus, first assume that A is open in the sense of the previous section, that is, there is a function $f : \mathsf{R}^m \to \mathsf{S}$ such that $(f\ x)\downarrow$ iff $x \in A$. We can enumerate all possible inputs $\mathsf{D}^d \times \mathsf{N}$ and define a sequence of balls by choosing $\mathrm{B}(d, 2^{-n})$ whenever $(\hat{f}\ d\ n) = \mathsf{True}$. By the properties of the interval extension, any such ball is contained in A and for any $x \in A$ we will find at least one ball that contains x.

Now assume A is Euclidean open. For any x, y, r, testing if $x \in \mathrm{B}(y, r)$ is semidecidable as we only need to check if $d(x, y) < r$. Thus, given $x : \mathsf{R}$ we define a sequence $s : \mathsf{N} \to \mathsf{S}$ such that $(s\ n)\downarrow$ iff $x \in \mathrm{B}((F\ n))$. Then $\exists(n : \mathsf{N}).\ (s\ n)\downarrow \leftrightarrow x \in A$, i.e., we need to check if there is at least one element in the sequence that is defined, which is again semidecidable.

For practical purposes, we are mostly interested in Euclidean subsets that are both compact and overt. Such sets (and some variations) have been considered in constructive mathematics and computable analysis under different names, sometimes using the same terminology for slightly different concepts. In Bishop's constructive mathematics, a set is compact if it is closed and totally bounded [10]. Here, totally bounded means that it can be covered by finitely many subsets of fixed size. Brattka and Weihrauch [4] define a representation $\nu_{\text{min}-\text{cover}}$ for compact sets based on such coverings.

We only consider subsets of Euclidean space that are classically closed and characterize their subsets in terms of “drawings” with arbitrary precision, meaning that for each n we can generate a picture of A in terms of “pixels” (i.e. small boxes or closed balls) of size 2^{-n}. It turns out that subsets for which such drawings exist are precisely those that are both compact and overt. It can further shown that they coincide with the *located* subsets, i.e., the subsets for which the distance of the set and a real number is computable (cf. [27, §12] and [24]). We thus use the term located for these subsets. We can define a closed ball $\mathrm{B}(c, r) \subseteq \mathbb{R}^m$ with center c and radius r (w.r.t. the maximum norm) simply as the tuple $(c, r) : \mathsf{R}^m \times \mathsf{R}$. We denote the type of all such balls by B and use [B] to denote finite lists of balls. For any $b : \mathrm{B}$ we further let $||b||$ denote its radius. We then formally define the located sets by

$$\begin{aligned} \text{located } A :\equiv \quad & \Pi(n : \mathsf{N}).\ \Sigma(L : [\mathrm{B}]). \\ & \Pi(b \in L).\ ||b|| \le 2^{-n} && \text{(fast convergence)} \\ \wedge \quad & \Pi(b \in L).\ b \cap A \neq \emptyset && \text{(intersection)} \\ \wedge \quad & \textstyle\bigcup_{b \in L} b \supseteq A && \text{(cover).} \end{aligned}$$

Thus, a set is located if for each n we get a finite list of closed balls, each with radius at most 2^{-n} so that their union covers the whole set A and such that each ball intersects A. Note that we need to restrict to classically closed subsets as the coverings only define a set up to its closure. The empty set is located as we allow closed balls to be empty (encoded by a negative radius).

We call the set defined by the union of the elements of the n-th list of the sequence the n-th approximation or the n-th covering of A and denote it by A_n.

For $A \subseteq \mathsf{R}^m$ and $x \in \mathsf{R}^m$, let us write $d(x, A)$ for $\inf_{y \in A} \|x - y\|$.

The Hausdorff distance $d_H(A, B)$ of two sets is defined by

$$d_{\mathrm{H}}(X, Y) = \max\left\{ \sup_{x \in X} d(x, Y),\ \sup_{y \in Y} d(X, y) \right\}.$$

Note that the set defined by the n-th approximation has Hausdorff distance at most 2^{-n} to the set that is approximated. In fact, the other direction also holds, i.e. we can show

$$\text{located } A \leftrightarrow \Pi(n : \mathsf{N}).\ \Sigma(L : [\mathrm{B}]).\ d_{\mathrm{H}}(A, \bigcup_{b \in L} b) \le 2^{-n}.$$

To see this, assume for each $n \in \mathbb{N}$ we can get a list of balls L such that $d_{\mathrm{H}}(A, \bigcup_{b \in L} b) \le 2^{-n}$. We take the $(n + 1)$-st list and double the radius of each of the balls. This guarantees that the resulting set covers A while still being small enough.

Note that center and radius of the balls can be arbitrary real numbers, as in the context of exact real computation it is more convenient to work with real numbers than with rationals. However, decidable equality on balls can be useful. Restricting to only dyadic rationals for center and radius does not make a difference as we can approximate real numbers arbitrarily well by dyadic rationals and thus can convert any real covering to a dyadic covering.

It can be shown that a set is located if and only if it is compact and overt. Although a formal proof of the equivalence could be used to prove that some of the subset operations preserve locatedness, recall that the main goal of our project is to extract efficient programs from proofs. It is thus reasonable to have different proofs using the more efficient representation, and a formal proof of the equivalence is less important. While the algorithmic content of the proofs is rather simple, showing correctness of the procedures needs some facts from classical analysis complicating the formalization of the proof. We therefore only give a pen and paper proof of the equivalence in Appendix A and state the fact as a meta-theorem.

5.2 Operations on located subsets

Let us first show that we can effectively get the distance of a point and a located set. For any nonempty located sets we can compute the distance function. That is,

$$\text{located } A \wedge A \neq \emptyset \to \Pi(x : \mathsf{R}^m).\ \Sigma(r : \mathsf{R}).\ r = d(x, A).$$

For any ball $b = \mathrm{B}(c, r)$, the distance $d(b, x)$ is given by $\max\{0, \|x - c\| - r\}$. We choose the n-th approximation and take the minimum over all $d(b, x)$. This gives a 2^{-n} approximation of $d(A, x)$. Taking the limit of all the approximations for each n thus corresponds to $d(A, x)$.

Let us next show that several standard operations on located subsets preserve locatedness. Although the results already follow from located sets being both compact and overt, our proofs encode more efficient algorithms, making use of the representation of located sets. The extracted programs can be used as a small calculus to combine subsets using set operations.

The properties stated below mostly can be shown by directly applying the operations on the coverings. We thus state them without proof.

1. For located sets A, B their union $A \cup B$ is located.
2. For a located set A and any $\lambda > 0$, the scaled set $\lambda A := \{\lambda x \ : \ x \in A\}$ is located.
3. For a located set A and any $v : \mathsf{R}^m$, the set $A + v := \{x + v \ : \ x \in A\}$ is located.

Using the continuity principle on real numbers we can generalize the above to images of arbitrary real functions. For any set $A \subseteq \mathsf{R}^m$, and function $f : \mathsf{R}^m \to \mathsf{R}^k$ let us define the image $f[A] \subseteq \mathsf{R}^k$ by $y \in f[A] :\leftrightarrow \exists(x : A).\ f(x) = y$. Then we can show that the image of located sets is again located:

$$\Pi(A \subseteq \mathsf{R}^m).\ \Pi(f : \mathsf{R}^m \to \mathsf{R}^k).\ \text{located } A \to \text{located } f[A]$$

To show this, for each $n \in \mathbb{N}$, we need to find a covering of $f[A]$ with balls with radius at most 2^{-n}. We can use the continuity principle on real functions to define an extension $\hat{f}$ that maps balls $b \subseteq \mathsf{R}^m$ to balls (with possibly infinite radius) $b' \subseteq \mathsf{R}^k$ such that $f[b] \subseteq b'$. We can further show that the radius of $\hat{f}(b)$ goes to zero when the radius of b goes to zero. Thus we generate all coverings, apply $\hat{f}$ to each of the balls of the coverings and check if the radius of each of them is bounded by 2^{-n} which we know will eventually happen. While generating images under arbitrary functions is very powerful, the procedure is obviously quite inefficient and thus generating sets using the above specialized operations is preferred.

Another very useful operation that lets us easily generate more complicated sets from simple ones is the limit operation. Here, by limit operation we mean that we are given a sequence of located sets which converge to another set in terms of Hausdorff distance. That is, a set K is defined to be the limit of a sequence of located sets if

$$\text{isLim } K :\equiv \Pi(n : \mathsf{N}).\ \Sigma(X : \text{located } \mathsf{R}^m).\ d_H(X, K) \leq 2^{-n}$$

holds. We can show that the limit of a sequence of located sets is again located, i.e.

$$\text{isLim } K \to \text{located } K.$$

To see this, let X_i denote the i-th set in the sequence and $X_{i,j}$ denote its j-th covering. As $d_H(X_i, K) \leq 2^{-i}$ and $d(X_{i,j}, X_i) \leq 2^{-j}$ it suffices to take $X_{n+1,n+1}$ and double the radius of each of the balls to guarantee that the resulting set covers K.

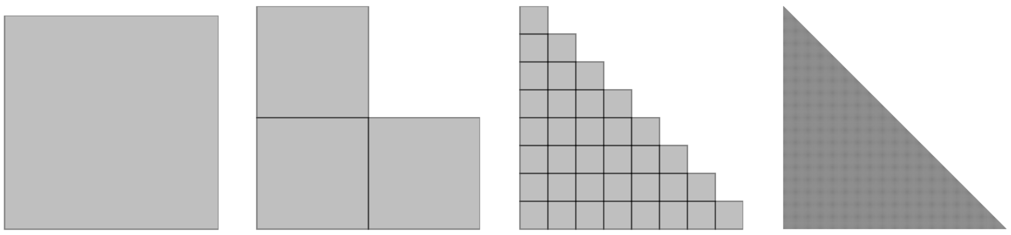

Figure 1 Approximations of the triangle.

6 Examples

From a proof that a set is located in our Coq implementation, we can extract an AERN program that computes the coverings. We implemented the examples below in Coq and extracted programs to generate for each n a finite list of real numbers, encoding the center and radii of each ball in the n-th covering. We can then use AERN to give us arbitrarily exact rational approximations of these real numbers, which we in turn can output and visualize by drawing the approximate boxes.

6.1 A Simple Triangle

Let us start with a simple example. We define a triangle $T \subseteq \mathbb{R}^2$ by

$$(x, y) \in T \leftrightarrow x \geq 0 \wedge y \geq 0 \wedge x + y \leq 1.$$

To show that the set is located, let $b_{i,j,n} := \mathrm{B}((\frac{2i+1}{2^{n+1}}, \frac{2j+1}{2^{n+1}}), 2^{-n})$. We define the n-th covering as the list L containing all $b_{i,j,n}$ with $i+j < 2^n$. Figure 1 shows the coverings defined by this procedure. All of the balls have radius 2^{-n} by definition and each ball intersects the triangle as $i+j < 2^n$ implies $\frac{2i+1+2j+1}{2^{n+1}} \leq 1$. Each point of the triangle is contained in one of the $b_{i,j,n}$ as for any $x \in [0,1]$ and $n \in \mathbb{N}$ we can find some $k \in \mathbb{N}$ such that $2^{-n}k \leq x < 2^{-n}(k+1)$. We can thus find such coordinates (i, j) for a point $(x, y) \in T$ and as $x + y \leq 1$ it follows that $i + j < 2^{-n}$ as claimed.

6.2 Fractals

As a more interesting application, let us look at a procedure to generate certified drawing of fractals. We consider simple self-similar fractals generated by iterated function systems (IFPs) without rotation. For simplicity we only consider fractals that are contained in the unit cube $[0,1]^m$. For a finite set of points $D := (d_1, d_2, \ldots, d_k) \subseteq [0,1]^m$ we define a classical subset $\mathcal{I}(D) \subseteq \mathsf{R}^m$ as the smallest subset such that

1. $D \subseteq \mathcal{I}(D)$, and
2. $\Pi(x : \mathcal{I}(D)).\ \Pi(d : D).\ \frac{x+d}{2} \in \mathcal{I}(D)$.

Classically it can be shown that there is a unique compact subset with these properties. We show that any set defined in this way is located. To do so, we need to define a sequence of coverings of the set. We start with the first covering L_0 being the list containing only the unit disc. As each $d \in D$ is contained in the unit disc, the intersection and covering properties hold. We then recursively define the covering L_{n+1} from the previous covering L_n by following the construction rule, i.e., by making for each $d \in D$ a copy of the previous covering where the center of each ball is moved halfway towards d and has half the radius.

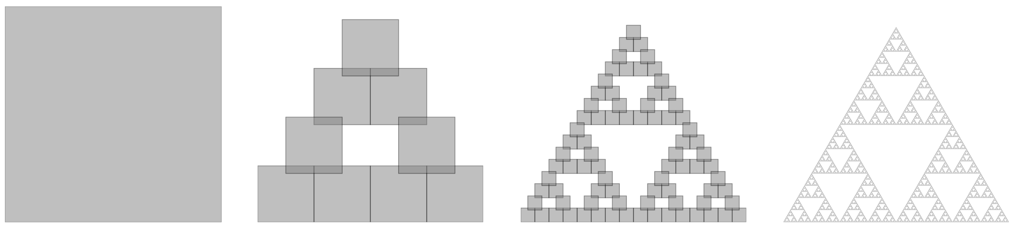

Figure 2 Approximations of the Sierpinski triangle.

Obviously, each of the balls in L_{n+1} has radius $2^{-(n+1)}$. To show that this procedure preserves the intersection property, let $b \in L_{n+1}$. Then there is some $b' \in L_n$ and $d \in D$ such that $x \in b$ iff there is an $x' \in b'$ with $x = \frac{x'+d}{2}$. As L_n has the intersection property, there is some $x' \in b'$ with $x' \in \mathcal{I}(D)$. By definition of $\mathcal{I}(D)$, then $\frac{x'+d}{2} \in \mathcal{I}(D)$, i.e., $b \cap \mathcal{I}(D) \neq \emptyset$. Similarly, to show that the covering property is preserved, assume $x \in \mathcal{I}(D)$. Then by definition either $x = d$ for some $d \in D$ or there is some $x' \in \mathcal{I}(D)$ and some $d \in D$ such that $x = \frac{x'+d}{2}$. Any $d \in D$ is contained in L_{n+1} as $d \in L_n$ and $d = \frac{d+d}{2}$. In the other case, there is some $b \in L_n$ such that $x' \in b$ as L_n is a covering. But then by definition of the procedure, there is some $b' \in L_{n+1}$ that contains x.

A more elegant way to prove the locatedness of $\mathcal{I}(D)$ is using the limit operation. We can define a sequence of located sets $(T_i)_{i\in\mathbb{N}}$ by letting T_0 be the unit disc and then applying the iteration given by the IFP. It is not hard to show that each T_i has Hausdorff distance at most 2^{-i} to the fractal and thus applying the limit operation suffices. However the extracted program is slightly less efficient than the direct encoding as the limit uses T_{n+1} to get the n-th approximations and increases the radius leading to coverings where balls overlap.

As an application of the result, consider the Sierpinski triangle which is defined as the two dimensional subset $\mathcal{I}(D)$ with $D = \{(-1,-1),(1,-1),(0,\sqrt{3}-1)\}$.

From the above we get a proof that the set is located, from which we can extract a Haskell program computing the coverings. Figure 2 is a plot of the output of the extracted program for increasing n. To generate the plot, AERN has been used to approximate the real numbers for center and radius in the n-th approximation with error bounded by 2^{-n}. The overlap at the top and bottom of the boxes is due to the approximation of the real number $\sqrt{3}-1$.

7 Conclusion and future work

We considered procedures to work with different classes of subsets in a computational setting. Currently, we only consider simple operations on subsets to build new subsets and extract programs for certified drawings up to any desired resolution. Producing coverings yields a representation that is easy to manipulate computationally and efficient if our goal is to draw the whole set or do other operations that require global knowledge. However, naturally the size of the list of coverings grows exponentially when increasing the precision. One might argue that high precision approximations are more useful locally, i.e. instead of getting a picture of the whole set one might want to zoom into a small part of the set and draw it with arbitrarily high precision.

Instead of outputting the whole set, the list can be replaced by a nondeterministic test function, that tests whether a ball is close enough to the set or not:

$$\Pi(x : \mathsf{R}^d).\ \Pi(r : \mathsf{R}).\ \mathsf{M}\ \big((d(x,A) < r) + (d(x,A) > 2r)\big).$$

That is, given a ball with center x and radius r, the function tells us if the ball intersects the set or not, but it is allowed to slightly over-approximate the set, i.e., both answers are possible if a ball with twice the radius intersects the set. Similar representations are often used in complexity theory (e.g. [6, 7]). In future work, we plan to include such representations in our development and show the equivalence to the covering representation for located sets.

More interesting operations can be achieved from computing images of arbitrary functions. Currently, we only have a simple implementation using the continuity principle on real numbers which is not efficient enough to be used in actual applications. Similar to the representations of Euclidean subsets, we can think of more efficient representations for the space of real functions. One possibility that comes to mind is using polynomial models such as Taylor models [18]. This would also allow for extensions to more advanced operations like integration and solution operators for ordinary differential equations [12, 21].

References

1 A. Balluchi, A. Casagrande, P. Collins, A. Ferrari, T. Villa, and A.L. Sangiovanni-Vincentelli. Ariadne: a Framework for Reachability Analysis of Hybrid Automata. In *Proc. 17th Int. Symp. on Mathematical Theory of Networks and Systems*, Kyoto, 2006.

2 Ulrich Berger and Hideki Tsuiki. Intuitionistic fixed point logic. *Ann. Pure Appl. Log.*, 172(3):102903, 2021. doi:10.1016/j.apal.2020.102903.

3 Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Formalization of real analysis: A survey of proof assistants and libraries. *Mathematical Structures in Computer Science*, 26(7):1196–1233, 2016. URL: http://hal.inria.fr/hal-00806920.

4 Vasco Brattka and Klaus Weihrauch. Computability on subsets of euclidean space i: closed and compact subsets. *Theoretical Computer Science*, 219(1):65–93, 1999. doi:10.1016/S0304-3975(98)00284-9.

5 Franz Brauße, Pieter Collins, and Martin Ziegler. Computer science for continuous data. In François Boulier, Matthew England, Timur M. Sadykov, and Evgenii V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing*, pages 62–82, Cham, 2022. Springer International Publishing.

6 Mark Braverman. On the complexity of real functions. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 155–164. IEEE, 2005.

7 Mark Braverman and Michael Yampolsky. *Computability of Julia sets*, volume 23. Springer, 2009.

8 Pieter Collins. Computable analysis with applications to dynamic systems. *Mathematical Structures in Computer Science*, 30(2):173–233, 2020. doi:10.1017/S096012952000002X.

9 Luís Cruz-Filipe, Herman Geuvers, and Freek Wiedijk. C-CoRN, the constructive Coq repository at Nijmegen. In *International Conference on Mathematical Knowledge Management*, pages 88–103. Springer, 2004.

10 Hannes Diener. *Compactness under constructive scrutiny.* PhD thesis, University of Canterbury. Mathematics and Statistics, 2008.

11 Martin Hofmann. On the interpretation of type theory in locally cartesian closed categories. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic*, pages 427–441, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

12 Akitoshi Kawamura, Florian Steinberg, and Holger Thies. Parameterized complexity for uniform operators on multidimensional analytic functions and ODE solving. In *International Workshop on Logic, Language, Information, and Computation*, pages 223–236. Springer, 2018.

13 Michal Konečný, Sewon Park, and Holger Thies. Certified computation of nondeterministic limits. In *NASA Formal Methods Symposium*, pages 771–789. Springer, 2022.

14 Michal Konečný, Sewon Park, and Holger Thies. Extracting efficient exact real number computation from proofs in constructive type theory. *arXiv preprint*, 2022. arXiv:2202.00891.

15 Michal Konečný. aern2-real: A Haskell library for exact real number computation. `https://hackage.haskell.org/package/aern2-real`, 2021.
16 Christoph Kreitz and Klaus Weihrauch. Theory of representations. *Theoretical computer science*, 38:35–53, 1985.
17 Horst Luckhardt. A fundamental effect in computations on real numbers. *Theoretical Computer Science*, 5(3):321–324, 1977. `doi:10.1016/0304-3975(77)90048-2`.
18 Kyoko Makino and Martin Berz. Taylor models and other validated functional inclusion methods. *International Journal of Pure and Applied Mathematics*, 6:239–316, 2003.
19 Valérie Ménissier-Morain. Arbitrary precision real arithmetic: design and algorithms. *J. Log. Algebr. Program.*, 64(1):13–39, 2005. `doi:10.1016/j.jlap.2004.07.003`.
20 Norbert Th Müller. The iRRAM: Exact arithmetic in C++. In *International Workshop on Computability and Complexity in Analysis*, pages 222–252. Springer, 2000.
21 Nedialko S Nedialkov. Interval tools for odes and daes. In *12th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN 2006)*, pages 4–4. IEEE, 2006.
22 Arno Pauly. On the topological aspects of the theory of represented spaces. *Comput.*, 5:159–180, 2016.
23 R. A. G. Seely. Locally cartesian closed categories and type theory. *Mathematical Proceedings of the Cambridge Philosophical Society*, 95(1):33–48, 1984. `doi:10.1017/S0305004100061284`.
24 Bas Spitters. Locatedness and overt sublocales. *Annals of Pure and Applied Logic*, 162(1):36–54, 2010.
25 Dieter Spreen and Ulrich Berger. Computing with Infinite Objects: the Gray Code Case. *Logical Methods in Computer Science*, Volume 19, Issue 3, July 2023. URL: `https://lmcs.episciences.org/11550`.
26 Florian Steinberg, Laurent Thery, and Holger Thies. Computable analysis and notions of continuity in Coq. *Logical Methods in Computer Science*, Volume 17, Issue 2, May 2021. URL: `https://lmcs.episciences.org/7478`.
27 Paul Taylor. A lambda calculus for real analysis. *Journal of Logic and Analysis*, 2, 2010.
28 K. Weihrauch. Computable analysis. Springer, Berlin, 2000.

A Equivalence of representations

We prove that a classically closed set is located if and only if it is both compact and overt. Since the *if* part is straightforward, let us focus on the *only if* direction.

Let us assume that A is compact and overt and show that A is located as well. We need to find for any natural n a finite set of balls of radius at most 2^{-n}, each intersecting A, and collectively covering A. We will also assume $n \geq 1$ and assign to $n = 0$ the same set of balls as for $n = 1$.

As A is computationally compact, we can find a ball containing A. Without loss of generality, assume $A \subseteq B(0, 1)$.

Let J be the set of all d-dimensional integer vectors j whose integer components j_i satisfy $-m \leq j_i \leq m$ where $m = 3 \cdot 2^{n-1}$. The points $c_j = j/m$ for $j \in J$ form a uniform grid over the (max-norm) unit ball $B(0, 1)$. Therefore the union of balls of radius $r = 1/2m$ around c_j covers $B(0, 1)$ and therefore A:

$$A \subseteq B(0,1) \subset \bigcup_{j \in J} \overline{B}(c_j, r) \tag{2}$$

We will apply `compact` A and `overt` A on open and closed sets, respectively, derived from the above cover of A, except that we increase the radius to $3r = 3/2m = 2^{-n}$. This will make the cover sufficiently redundant to compensate for potential non-termination of the compactness and overtness applications on specific sets.

For each $j \in J$, let

$$\begin{aligned} \text{disjoint}_j &= \quad \text{compact } A \ \big(A \cap \overline{B}(c_j, 3r)\big) \\ \text{intersects}_j &= \qquad \text{overt } A \ (A \cap B(c_j, 3r)) \\ \text{haveinfo}_j &= \text{empty}_j \sqcup \text{nonempty}_j \end{aligned}$$

Note that as $\text{disjoint}_j, \text{intersects}_j, \text{haveinfo}_j \in \mathsf{S}$, their computation may not terminate. Moreover, haveinfo_j terminates iff either of the other two terminate.

We compute all haveinfo_j computations in parallel until enough of them succeed to cover A, more precisely, until for the set of terminated indices $J_\text{T} \subseteq J$, it holds:

$$B(0,1) \subseteq \bigcup_{j \in J_\text{T}} \overline{B}(c_j, 3r) \tag{3}$$

Once Equation (3) holds, the set $\{\overline{B}(c_j, 3r) \mid j \in J_\text{T} \wedge \text{intersects}_j\}$ is the desired covering, namely, the balls have sufficiently small radius, each of them intersects A, and their union covers A thanks to Equation (3) and the definition of disjoint_j, which means that we omit only balls that are disjoint from A.

It remains to prove that Equation (3) must hold in finite time. This will follow from Equation (2) when we show that for each $j \in J$ the ball $\overline{B}(c_j, r)$ is covered by some $\overline{B}(c_k, 3r)$ such that haveinfo_k terminates. Note that the ball $\overline{B}(c_j, r)$ is covered by $\overline{B}(c_k, 3r)$ for all $k \in J$ with $||j - k|| \leq 1$, i.e., the “neighbouring” indices in addition to j itself.

Now, if haveinfo_j does not terminate, i.e., if both empty_j and nonempty_j do not terminate, the ball $\overline{B}(c_j, 3r)$ is not disjoint from A but its interior is disjoint from A. This means that there is $a \in A$ with $||c_j - a|| = 3r$ as illustrated in Figure 3.

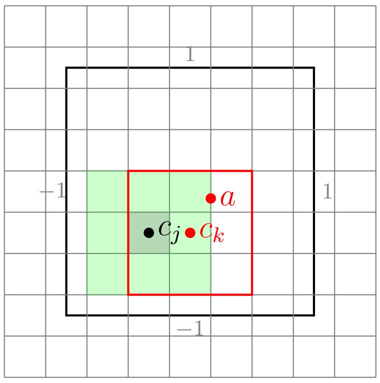

Figure 3 $\overline{B}(c_j, r)$ covered by $\overline{B}(c_k, 3r)$; $n = 1$, $m = 3$, $r = 1/6$, $j = (-1, -1)$.

Let us define the vector k component-wise as follows:

$$k_i = \begin{cases} j_i + 1 & a_i - (c_j)_i = 3r \\ j_i - 1 & a_i - (c_j)_i = -3r \\ j_i & |a_i - (c_j)_i| < 3r \end{cases} \tag{4}$$

Note that k is a “neighbour” of index j. Now, $k \in J$ because if it was not, some component k_i would be outside the interval $[-m, m]$, which would mean that either $a_i = 1 + 3r$ or $a_i = -1 - 3r$, which would contradict $A \subseteq B(0, 1)$.

Using the same three cases as in Equation (4), we get:

$$|a_i - (c_k)_i| = \begin{cases} |a_i - (c_j)_i - 1/m| = 3r - 1/m < 3r & k_i = j_i + 1 \\ |a_i - (c_j)_i + 1/m| = |-3r + 1/m| < 3r & k_i = j_i - 1 \\ |a_i - (c_j)_i| < 3r & k_i = j_i \end{cases} \tag{5}$$

In all three cases we have $|(c_k)_i - a_i| < 3r$. Over all i, we get $a \in B(c_k, 3r)$.

Finally, $a \in B(c_k, r)$ implies that nonempty$_k$ terminates, and therefore $\overline{B}(c_j, r)$ is covered by $\overline{B}(c_k, 3r)$ in Equation (3) as illustrated in Figure 3.