Logical Methods in Computer Science
Volume 17, Issue 2, 2021, pp. 16:1–16:43
https://lmcs.episciences.org/

Submitted May 01, 2019
Published May 12, 2021

COMPUTABLE ANALYSIS AND NOTIONS OF CONTINUITY IN COQ

FLORIAN STEINBERG [a], LAURENT THÉRY [b], AND HOLGER THIES [c]

[a] INRIA Saclay, Bât 650, Rue Noetzlin, 91190 Gif-sur-Yvette, France

[b] INRIA Sophia Antipolis, 2004 route des Lucioles, BP 93, 06902, Sophia Antipolis Cedex, France

[c] Kyushu University, 744 Motooka, Nishi-ku, Fukuoka 819-0395, Japan

Abstract. We give a number of formal proofs of theorems from computable analysis. Many of our results specify executable algorithms that work on infinite inputs by means of operating on finite approximations. The proofs that these algorithms are correct in the sense of computable analysis are verified in the proof assistant Coq heavily relying on the Incone library for information theoretic continuity. This library is developed by one of the authors and the paper can be used as an introduction to it. Incone formulates the continuity-theoretic aspects of computable analysis. It is designed in such a way that it can be combined with Coq's Type/Prop distinction to provide a general purpose interface for algorithmic reasoning on continuous structures and many of our results provide complete computational content.

The results that provide complete computational content include that the algebraic operations and the efficient limit operator on the reals are computable, that the space of infinite sequences is isomorphic to a space of functions, compatibility of the enumeration representation of subsets of natural numbers with the abstract definition as space of functions to Sierpinski space and that continuous realizability implies sequential continuity. We also formalize proofs of non-computational results to support the correctness of the definitions: We show that the information theoretic notion of continuity used in Incone is equivalent to the metric notion of continuity on Baire space. We give a complete comparison of the different concepts of continuity that arise from metric and represented-space structures. Finally we prove discontinuity of the tasks of finding the limit of a converging sequence of real numbers and selecting an element from a closed subset of the natural numbers.

1. Introduction

Computable analysis is the theory of computing on continuous structures. Its roots are often cited as going back to Turing's fundamental paper from 1936 in which he introduced his mathematical model of computation later known as Turing machine [Tur36]. Turing's original definitions relied on the binary representation and was flawed. In his 1937 correction he adapted it to use a signed binary representation that is still common today [Tur38]. For

Key words and phrases: Computable Analysis, Formal proofs, Coq proof assistant, Closed Choice.

* This work was supported by JSPS KAKENHI Grant Numbers JP18J10407 and JP20K19744, by the Japan Society for the Promotion of Science (JSPS), Core-to-Core Program (A. Advanced Research Networks), by the ANR project FastRelax (ANR-14-CE25-0018-01) of the French National Agency for Research and by EU-MSCA-RISE project 731143 "Computing with Infinite Data" (CID).

the idea behind the corrected definition he pointed to earlier work from constructive analysis by Brouwer [Bro75]. The theory of computable functions on the real numbers was further developed in the 1950s by Grzegorczyk and Lacombe in parallel [Grz57, Lac58]. Finally, Kreitz and Weihrauch extended the results to apply to more general spaces and introduced the formal framework of representations that is standard today [KW85].

The basic idea behind computable analysis is fairly simple: To make uncountable structures available to computation, one encodes them by infinitary objects that can still be operated on mechanically. Most commonly infinite strings are used, but more conveniently one may use functions between discrete structures. An example of a reasonable encoding of real numbers is to describe them by functions that provide arbitrarily accurate approximations. The inputs and outputs of such functions can be rational numbers and thus be described by finite means. To compute functions on the real numbers, one operates on these encodings and algorithms use a model of computation that can handle infinite inputs while remaining realistic in the sense of being implementable.

Operating on continuous data such as real numbers is necessary in many applications. The use of software based on computable analysis for solving practically relevant problems is fairly uncommon. Partly this is due to the strict correctness assumptions that are imposed by computable analysis. These involve high-level mathematical concepts and require algorithms to scale to arbitrary precision. On one hand, due to the conceptual complexity, checking correctness of programs in the sense of computable analysis can be subtle and mathematical simplicity of algorithms is valued highly. On the other hand, for practical applications performance is a central topic. Optimizations that rely on the specifics of the hardware are not uncommon and may fail to scale. Algorithms from computable analysis are typically difficult to optimize for performance and thus their main area of impact are applications that require utmost reliability.

Software packages based on computable analysis require their users to keep track of the details of the semantics to maintain the high reliability that computable analysis provides in principle. This task becomes increasingly challenging to do by hand when programs become more complicated. For instance, when dividing a program into components to be optimized separately, each subtask has to be carefully annotated with the properties that are needed for correctness of the combined procedure. These problems are not specific to computable analysis, however, in computable analysis understanding annotations typically requires a fair amount of mathematical background knowledge. As a consequence, many of the developers of the more popular software packages for computable analysis have started to look into possibilities to link their software to some sort of formal verification. Ideally, this should make it possible for mathematicians to implement their ideas without having to deal with the low level details of programming languages and relying on components that computer scientists can develop and optimize. Formal specifications guarantee that the overall programs do not suddenly become incorrect due to an optimization that changes the semantics in a subtle way and has not been properly communicated.

This paper describes a formal development where we formulated some well-known facts from computable analysis and give computer verified correctness proofs. For the more basic operations we proceed by first obtaining a target specification from the theoretical background provided by computable analysis and then specifying a fully executable algorithm and prove it to fulfill the specification. For more advanced operations we argue that the desired algorithm can be constructed from any set of solutions of more basic operations and finally obtain an executable program by combining what we, or other formal developments, provided earlier. The algorithms we produce are far from being competitive in terms of speed or memory consumption. Their relevance is more to witness theoretical implementability or prove impossibility of such an implementation. This means that the developed theory can safely be used as a basis for verification of software packages that operate more efficiently [Mül01, Kon08, BCC$^+$06]. Due to recent developments in verified numerics it may also be possible to optimize the algorithms underlying our proofs quite a bit without sacrificing the additional safety provided by exclusively working inside of a proof assistant [BM17].

The standard references to find basic facts from computable analysis are [PER89, Ko91, Wei00]. In [Bau00, Sch02a, Pau16] similar topics are presented in a form that is somewhat closer to how this paper proceeds. This paper restates many of the basic definitions from computable analysis to outline how our formal development diverges from a traditional treatment. However, we do not attempt to give exhaustive explanations or justify more than the differences to the traditional definitions. We assume the reader to have enough background in computable analysis to see the reasonability of the definitions used there. For readers without background in computable analysis let us quickly point out that the model of computation used in computable analysis is by far not the only popular model for operating on functional inputs and that differences to other models can be quite subtle. To avoid confusion, readers from adjacent fields may want to consult sources that give a systematical comparison [AB14, LN15].

1.1. **Coq and proofs about continuous structures and computation.**

As a framework for formal verification of proofs from computable analysis, we use the proof assistant Coq. This proof assistant is type theory based, but the content of this paper should be understandable with little to no background in type theory or experience with Coq for that matter. We hope that the basic facts that do make an appearance are comprehensible from the brief description of the basic workings of Coq provided in this section. For it to be possible to link the results from this paper to the formal development we provide some additional information in Appendix A.

Coq is a proof assistant that supports mathematicians in giving fully formal proofs of their results. Ideally, a Coq development looks very similar to a mathematical paper. It consists mostly of definitions and lemmas with some explanations and some documentation in between. The definitions specify objects that the developer is interested in and properties he wants these objects to have. Lemmas provide evidence that some of the objects actually have some of the properties. In practice, formal developments fall short in readability due to the need to specify every last detail and the tendency to optimize proofs for brevity instead of readability. However, any person that has access to a computer with a Coq installation can verify the correctness of a Coq development without the need to understand all of its details. Furthermore, even without understanding the details of the proofs of lemmas defined by others, it is possible to reuse them without being afraid of making errors as Coq will check all the details and prevent the user from mixing up definitions or missing details.

There is a wide variety of proof assistants to choose from. The Coq system is particularly appropriate for the purpose of this paper. This is because it is designed to support a mathematician in maintaining and extracting computational meaning while doing mathematical proofs. The most common way of doing so is by resorting to exclusively constructive reasoning and the mathematician who chooses to do so is rewarded by programs that can be executed directly in Coq. This provides a very high standard of reliability. For achieving this executability Coq relies on the Curry-Howard correspondence to identify proofs and programs. Indeed, proving a lemma in Coq can be seen as using a high level programming language to specify how evidence of its truth can be obtained from the given inputs. Some of the inputs here may not be traditional input data but instead evidence that the assumptions of the lemma are fulfilled. Thus, proving a lemma may be understood as a special case of a defining a function. One may also invert this and follow a Coq definition with a proof providing the value in the high level language instead of specifying it manually.

However, it is important to keep up the distinction between providing evidence of a property and producing a value. Just like in mathematics, the details of a value can be important while a lemma should be chosen such that all information is contained in its statement so that it can be used independently of why it is true: It should be such that it is never necessary to unfold the proof and the exact form of the evidence provided is irrelevant. This means that the corresponding lemmas do not have computational content and are purely for specification purposes and is independent from whether one chooses to work constructively or not. Such lemmas can be marked as correct and the details of their proofs can be hidden. In Coq a collection `Prop` of values that should be considered properties is provided and meant to be used to mark parts that do not have computational content in the above sense. The distinguishing feature is that a definition of a function cannot depend on details (i.e. the proof) of inputs that are marked properties. This rule is what allows the code extraction machinery of the Coq system to safely disregard these parts as non-computational.

A mathematician working with Coq will quickly run into statements that should intuitively be true but cannot be proven. An example that often causes mathematicians to be confused is functional extensionality: For functions f and g of the same type mathematicians would assume that the statement `(forall a, f(a) = g(a)) -> f = g` is true, but this is not provable in Coq. Depending on the upbringing of the mathematician, another example may be the law of excluded middle. Coq allows to assume the truth of properties such as the two above by stating them as axioms. Many mathematical developments force the truth of functional extensionality by assuming it in this way. Other popular axioms include choice principles. Of course, one has to make sure that the axioms are compatible with Coq's internal logic and compatible with each other. Coq's official webpages[1] list some known facts about consistencies of axioms that are often used.

1.2. **Computational content in presence of axioms.** In the previous paragraph we spoke about "axioms" as assumptions about the logical background theory. This should be distinguished from local axioms such as those for algebraic structures that only apply to a local context. The best practice would be to never assert axioms as global assumptions and always prove implications that their truth has certain consequences. However, if certain assumptions about the logical background theory are consistently used throughout a development, stating them as axioms leads to considerably cleaner overall appearance. In our development we do this for the law of excluded middle, for functional extensionality and for some choice principles.

In principle, assuming any consistent set of axioms that state properties is an acceptable practice, but one should be aware that the use of axioms can make preserving and extracting computational content challenging. Depending on the nature of the axioms, the loss in executability is gradual. More specifically, axioms that assert properties typically prevent

[1]`https://github.com/coq/coq/wiki/The-Logic-of-Coq#axioms`

execution inside of the Coq proof assistant, but an executable program in a functional programming language can still be obtained using Coq's code extraction capabilities. Note that it is thus a plausible assumption that any function that can be defined in Coq is computable even if it relies on this kind of axiom. This remains true if the set of axioms is inconsistent: extracted programs are still executable but due to the inconsistency it can be proven that an executable program exhibits incomputable behavior as any specification can be proven in this case.

However, Coq also allows to use axioms to introduce objects that cannot realistically be constructed. In this case the extracted algorithm can be understood to make use of a subroutine that is not provided and if it is provably impossible to implement this missing part appropriately, the produced code should be considered useless. As example let us consider the real numbers that are axiomatized as an archimedean closed field in Coq's standard library. This axiomatization asserts a function up: $\mathbb{R} \to \mathbb{Z}$ to be available of which it is assumed that it returns the least integer bigger than its input. No computable such function exists and as a consequence, code extracted from statements about the real numbers in the standard library is rarely of practical use. At best one obtains algorithms whose correctness relies on availability of an algorithm implementing the up function and other assumptions such as exact operations on real numbers. A replacement of the reals by any realistically implementable type leads to an almost guaranteed loss of correctness. This is not to say that the axiomatization is useless for specification of algorithms.

While the up function may not realistically implementable, a mathematician may take the position that it still exists and having direct access to such objects can be desirable. The existence of the up function could also be stated as a property of the real numbers by hiding it behind an existential quantifier. For the mathematical development this would result in a lot of inconveniences where an existential quantifier has to be resolved and a uniqueness lemma has to be used. Moreover, the very mechanism that the Coq system uses to support its users in keeping the distinction between computational and non-computational content will disallow the use of the up function definitionally if this is done. This system can in more general cases appear as a hurdle to users who do classical mathematics, where more liberal definitional thinking is common practice. For instance, it generally disallows branching over properties and as a result a definition by cases is often not possible even if one can prove that one of the cases must always be true. To do branching one needs to first find an algorithm that decides the truth by returning a Boolean and such an algorithm is at times hard to come by with. More recent developments in Coq's community for formalization of results from analysis take an even clearer stance on these topics and assume the full strength ε-axiom to make the development of abstract mathematics more convenient [ACR18].

1.3. **Related work.**

Verification of real and numerical analysis is an active field of research and several projects in Coq and other proof assistants exist. We do not attempt to give an overview here but instead refer to Boldo, Lelay and Melquiond's survey paper [BLM16]. Let us still briefly mention the projects that are most relevant for our formalization.

As described in the previous section, the definition of the reals in the standard library makes it difficult to strictly separate computational and non-computational content. A solution for the real numbers that neither gives up on convenience nor interferes with the code extraction capabilities is to treat non-computational assumptions such as the up function as parameters of a real number structure and is currently under development as "mathcomp analysis" [ACK$^+$20]. However, there exists a vast body of work building on the classical axiomatization of the reals in the standard library. A good example for this is the Coquelicot library [BLM15], a widely used library for real analysis that is conservative over this axiomatization. To make it possible to reuse previous results of this kind, we rely on the axiomatization from the standard library for the present work and manually avoid improper use of the logical strength provided by this axiomatization.

A quite different approach is taken by the C-CoRn library [CFGW04]. The library defines a fully computational formalization of real numbers. It provides a wide range of results about functions on real numbers and some about operators on function spaces and includes an exhaustive treatment of metric spaces and uniformly continuous functions between metric spaces [O'C09]. Its design follows the development of constructive analysis by Bishop and Bridges [BB12]. Our treatment of the real numbers rarely goes beyond what can already be found in C-CoRn and many parts are inspired by it. This said, it should also be noted that the constructive nature may make the C-CoRn library and the publications related to it difficult to access for some classically trained mathematicians.

In contrast to that, the Incone library follows the usual approach that is taken in computable analysis: The mathematical background theory is developed classically and the algorithmic content is considered extra information about data representation that should follow the mathematical understanding. That is, it distinguishes between computational results in the form of algorithms and their correctness proofs. A similar approach is taken in recent work in verified numerics where an abstract mathematical theory is developed in a first step and computational content provided in an additional step by using the mathematical libraries to prove floating-point algorithms correct [BM17]. We hope that the "backward approach" taken by the Incone library allows for synergy with such developments and complement the "forwards approach" of working completely constructively as is done in developments such as C-CoRn.

1.4. **Structure of the paper and main results.** The main contributions of this paper are described in Section 4 with some exceptions that appear earlier in Section 3. All theorems, propositions and lemmas in this paper have been formally proven in Coq and have explicit pointers to their name in the Incone library. However, before we get to the core topics, some discussion of background is necessary. Section 1 contains the introduction, pointers to literature for reading up on computable analysis and a short description of some aspects of Coq that are relevant for understanding some of the more important remarks that directly address the formal development.

In Section 2 we introduce the concept of continuity of partial operators on Baire space. As a preparation for a proper treatment of partiality in Coq, we introduce basic concepts from the theory of multivalued functions, a formalization of which is provided by Incone's sublibrary mf for specification of functions through relations. We then discuss how to capture partial computable functions and operators in Coq by relying on a modification of the fuel based approach to diverging computation in type theories. This construction is one of the core concepts that is revisited many times throughout the rest of the paper. The second part gives an information theoretic description of continuity on Baire space and an overview over the formalization of this notion in the Incone library. The third part presents the universal that the Incone library uses to implement the function space construction from computable analysis.

Section 3 deals with the basic concepts from computable analysis, explains how they are realized in Incone and introduces the real numbers as an example that is used through the rest of the section. The first part of the section explicitly describes how a few of the simple type constructions like products are automated in the library. Arithmetic operations on the real numbers are used as a basic example to demonstrate the workings of these constructions. The second part describes how spaces of sequences can be constructed and considers point-wise operations on spaces of sequences and the limit operator on the real numbers as concrete examples. From a category theoretical point of view the space of sequences is a countably infinite product of a space with itself. As such it is of particular interest as the existence of countably infinite products is only guaranteed in the case where all continuous functions are considered as morphisms and may fail to exist if one restricts to computable ones. The final third part builds exponentials using a construction that is known to work for both these categories. It presents a formal proof that the space of sequences can be recovered as an exponential.

The final section (Section 4) starts with a brief description of the metric library and a comparison to other formalizations that have a similar purpose. The first part presents a formal proof that information theoretic notion of continuity that the Incone library uses internally is equivalent to the more traditional approach of equipping Baire space with an appropriate metric. The second part presents formal proofs about the relation of different concepts of continuity in metric spaces and represented spaces that can be constructed from such. The final part introduces Sierpinski space as a space that can be used to abstractly reason about open and closed subsets of represented spaces. For the concrete case of the natural numbers, the spaces of open and closed sets can be given more concrete representation that encode such sets by enumeration of themselves or their complement. These concrete representations are proven equivalent to the abstract representations and the equivalence is used to prove that the task of selecting an element of a closed set does not have a continuous solution.

The formal proofs that we consider related to this publication and part of its main contributions are that the space of infinite sequences in a space isomorphic to a space of functions (Theorem 3.14), that the algebraic operations and a limit operator for fast-converging Cauchy sequences on the reals are computable (Examples 3.5 and 3.8), compatibility of the enumeration representation of subsets of natural numbers with the abstract definition of the space of open subsets of the natural numbers (Theorem 4.10), and that continuous realizability implies sequential continuity (Theorem 2.6). The previous results are fully algorithmic, but we also describe many non-computational theorems. These include numerous specification results for the constructions in the Incone library (in particular Theorems 2.7, 2.11, 3.13 and Proposition 3.7), a proof that the information theoretic notion of continuity used in the library is equivalent to the metric notion of continuity on Baire space (Theorem 4.4), a complete comparison of the different concepts of continuity that arise from metric and represented-space structures (Theorems 4.7 and 4.8) and the discontinuity of the unrestricted limit operator on the real numbers (Example 3.8) and the task of selecting an element of a closed subset of the natural numbers (Theorem 4.13).

2. Multifunctions and partial operators on Baire space

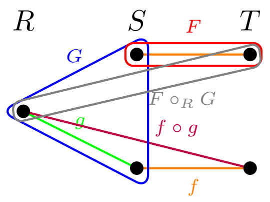
Baire space $\mathbb{N}^{\mathbb{N}}$ is the space of all total functions from natural numbers to natural numbers. Baire space comes with the structure of a topological space and a notion of computability of its elements. Computable analysis transfers the computability and topological structure of Baire space to more general spaces by means of encodings that are called representations.

Before we go into detail about how this can be done, this chapter describes the structure on Baire space that we need. We will be interested in functions on Baire space and for such the natural numbers appear in a number of different roles. To avoid confusion, one may be tempted to name the distinct copies of the natural numbers differently. We will do so and name the set of inputs Q for questions and the outputs A for answers so that Baire space takes the form A^Q. This naming is chosen such that it matches the use of Baire space in computable analysis and to avoid coding it can often be convenient to chose Q and A to not be the natural numbers but something that can sufficiently concretely be encoded by natural numbers.

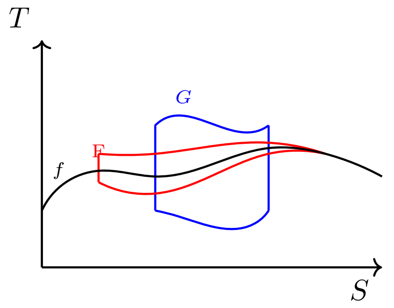
Motivated by the above we call a space of the form $\mathcal{B} = A^Q$ a **naming space** if Q and A are countable and non-empty. Here, the letter $\mathcal{B}$ is chosen because these spaces take the role that is traditionally taken by Baire space in computable analysis. The phrase "naming space" reflects how Baire space, and thus in our treatment also any of these spaces, is used in computable analysis. Classically the assumptions about Q and A imply that these sets are either finite or bijectively related to the natural numbers. Constructively this need not be true, or at least depends on the notion of countability that is used. Indeed, if computability considerations come in, that is, if the surjection whose existence is guaranteed by the countability is considered an encoding of the elements of the set by natural numbers, more care has to be taken. The critical reader may in the following replace any occurrence of Q and A and their dashed variants by $\mathbb{N}$. In the applications that we look at, these substitutions can be carried out by hand.

We will mainly be interested in partial functions between naming spaces. While Coq comes with a native concept of a function, for Coq to accept a definition of a function, the function must be total. Thus, for our purposes, simply relying on this concept is not an option. There are several ways to model partial functions in Coq. The most common one is to replace the target space of a function by a different space that has an additional value that stands for being undefined. Doing this naively works well in a purely mathematical setting, but quickly leads to problems when computability considerations come in. Another way to model partiality in Coq is to rely on the dependent type system. That is, an input of a partial function is a pair consisting of the input together with a proof that the input is from the domain of the partial function. Such a treatment is the most natural one, but heavy use of Coq's dependent type system brings its own disadvantages. For our purposes we decided it to be best to use aspects of the theory of multivalued functions to capture partiality.

Multifunctions are a very popular tool for specification and classification of problems in computable analysis [BKMP16, BGP17, BG11, BDBP12, PS18]. Within this field, multifunctions form a topic of research of their own [Pau17, PZ13]. This is not to say that this concept was invented for computable analysis, multifunctions also have applications in computational complexity, in particular the theory of promise problems and non-deterministic computation [Sel94, ASBZ13], and even in the treatment of non-smooth and non-linear problems in functional analysis [EM46, Dei92]. A **multivalued function** $F\colon S \rightrightarrows T$ assigns to each $s \in S$ a possibly empty subset $F(s) \subseteq T$. While this gives F the type of a relation, the intuition behind a multivalued function is different. By contrast to relations, multivalued functions are directed and S is treated as input type and T as output type. The domain of a multifunction F is given by $\operatorname{dom}(F) := \{s \in S \mid \exists t\colon t \in F(s)\}$ and for $s \in \operatorname{dom}(F)$ the set $F(s)$ is non-empty and should be interpreted as the set of eligible return values. A



(A) f chooses through F, g chooses through G. $f \circ g$ does not choose through $F \circ_R G$, but through $F \circ G$ which is empty. $G^{-1} \circ F^{-1} = (F \circ_R G)^{-1}$ is not the empty function.



(B) f chooses through F which tightens G. Thus f also chooses through G.

Figure 1

multivalued function is called **total** if its domain is all of S, and **singlevalued** if each $F(s)$ has at most one element.

Any multivalued function can and should be considered a specification for functions: A function $f\colon S \to T$ fulfills the specification $F\colon S \rightrightarrows T$ if for any $s \in \operatorname{dom}(F)$ it holds that $f(s) \in F(s)$. In this case we say that f **is a choice for** F. The operations on multivalued functions are chosen such that they behave well with the interpretation as specifications. For instance, the **composition** $F \circ G$ of two multivalued functions $G\colon R \rightrightarrows S$ and $F\colon S \rightrightarrows T$ is of type $R \rightrightarrows T$ and its value sets are given by

$$(F \circ G)(r) := \{t \in T \mid G(r) \subseteq \operatorname{dom}(F) \wedge \exists s\colon t \in F(s) \wedge s \in G(r)\}.$$

This should be compared to $(F \circ_R G)(r) := \{t \mid \exists s\colon t \in F(s) \wedge s \in G(r)\}$, which is what is commonly used as composition for relations. Both the relational and the multifunction composition are associative operations. The domain condition added in the multifunction composition is a modifier that addresses the difference in interpretations. For the multifunction composition it is true that if f is a choice for F and g is a choice for G then $f \circ g$ is a choice for $F \circ G$, which may fail for the relational composition as illustrated in Figure 1a. From the same figure it can be seen that the multifunction composition is not symmetric under changing the direction of the multifunctions while for relations this is the case. The relational and the multifunction composition produce identical results if the multifunction that is applied last is total (`comp_rcmp`) resp. the first one is singlevalued (`sing_comp`).

Functions and partial functions are special cases of multifunctions. A function $f\colon S \to T$ may be identified with the specification that on input s it makes $f(s)$ the only eligible return value, i.e. with $s \mapsto \{t \in T \mid t = f(s)\}$. The multifunction associated to a function is always total and singlevalued (`F2MF_tot` and `F2MF_sing`) and assuming that T is not empty and an appropriate choice principle, each total singlevalued multifunction arises in this way (`fun_spec`). We call this connection between functions and total singlevalued multifunctions the `F2MF` **correspondence**. This construction can be extended to partial functions by assigning to $g\colon \subseteq S \to T$ the multifunction function

$$s \mapsto \{t \in T \mid g(s) \text{ is defined and equals } t\}.$$

The resulting multifunction is singlevalued (`PF2MF_sing`) but need not be total. Note that we are being unspecific about how to encode partial functions here. One may use functions to an option type as outlined previously but should keep in mind that this can be understood to imply the domain of the partial function to be decidable which is rarely the case in our applications. It is not difficult to check that both relational and multifunction composition restrict to regular composition of functions or partial functions when considered only on singlevalued resp. total and singlevalued multifunctions (`F2MF_comp` and `PF2MF_comp`).

Any multifunction can be assigned a reverse multifunction by switching input and output. Each property of a multifunction has a co-version that requires the same property for the reverse multifunction. Many of the co-properties have nice characterizations for the special cases of functions. For instance, a function is injective if and only if the associated multifunction is co-singlevalued (`mfinv_inj_sing`). For later reference we list:

Lemma 2.1 (`PF2MF_cotot`)**.** *A partial function is surjective if and only if its associated multifunction is co-total.*

For multifunctions $F, G\colon S \rightrightarrows T$ we say that F **tightens** G if it is more restrictive as a specification. This can be spelled out elementary as

$$\operatorname{dom}(G) \subseteq \operatorname{dom}(F) \quad \text{and} \quad \forall s \in \operatorname{dom}(G)\colon F(s) \subseteq G(s).$$

Under weak additional assumptions this is equivalent to the statement that each choice function for F is also a choice function G (`icf_tight` and `tight_icf`, see Figure 1b). A function is a choice for a multifunction F if and only if its associated multifunction tightens F (`icf_spec`). For a partial function we say that it is a **partial choice** for F if its associated multifunction tightens F. This can be spelled out to still mean that f is a partial choice for F if whenever $s \in \operatorname{dom}(F)$ then $f(s)$ is defined and an element of $F(s)$. Many further properties of multifunctions are proven in the MF library and we just pick an example:

Lemma 2.2 (`tight_comp`)**.** *If F tightens F' and G tightens G' then $F \circ G$ tightens $F' \circ G'$.*

2.1. Capturing the computable partial functions and relativization.

Next we discuss another way to generate multifunctions from functions that is related to the standard way of modeling diverging computation in intuitionistic type theories. Recall that for a type T the option type $\operatorname{opt} T$ adds a single element to T. That is, each $t \in T$ corresponds to an element $\operatorname{Some} t \in \operatorname{opt}(T)$ and there is a single additional element $\operatorname{None} \in \operatorname{opt}(T)$. Given a function $N\colon \mathbb{N} \times S \to \operatorname{opt} T$ define a multifunction $\Phi_N\colon S \rightrightarrows T$ via

$$\Phi_N(s) := \{t \in T \mid \exists n\colon N(n, s) = \operatorname{Some} t\}.$$

We call the additional input $n \in \mathbb{N}$ the **effort parameter** and the reason for this should become apparent from the next paragraph where it takes the role of the number of steps taken by a Turing machine.

Consider the special case where $S = \mathbb{N} = T$. Given a partial computable function $f\colon \subseteq S \to T$ we want to argue that there exists a primitive recursive N that recovers it through the Φ-assignment above. Indeed fix some Turing machine that computes f and consider the function N that on input (n, s) returns $\operatorname{Some} t$ if the machine on input s terminates within the first n time-steps and returns t, and None otherwise. For this function N it holds that $\Phi_N(s)$ is empty if the machine diverges and otherwise consists of the single value $f(s)$, thus $\Phi_N(s) = f$ if we identify partial functions with their induced multifunction.

Although there are technical differences, it should be clear that the core idea behind this is a version of the Kleene normal-form theorem [Soa78]. The above makes the Φ assignment particularly interesting to us as any primitive recursive function is definable in Coq [O'C05].

Even for primitive recursive N, the multifunction Φ_N need neither be total nor singlevalued. In the case where S and T are the natural numbers, $N(n, s) :=$ None leads Φ_N to be the empty function and $N(n, s) := \mathrm{Some}(n)$ results in the total specification for that renders any return value eligible. However, for any N, it is possible to obtain a singlevalued tightening of the form $\Phi_{N'}$: Choose N' to be the function that on input n and s evaluates N on inputs $(0, s), \ldots, (n, s)$ and returns the first value returned by N if such exists and None otherwise. This N' is **monotone** in the sense that if it returns a value on some effort, then it returns the same value on all bigger efforts. Whenever a function is monotone in this sense, the corresponding multifunction is singlevalued. As the procedure of searching is computable, it is also true that for any computable N the function Φ_N has a computable choice function. Thus, assuming that the functions definable in Coq are computable (in the sense in which this was discussed in the introduction), we may use the Φ assignment to capture computability in Coq.

Since we are interested in specification of partial operators, we relativize the construction. That is, we start from the Φ assignment and add an additional argument that reaches over Baire space. For a fixed φ from Baire space we thus capture computability by an oracle Turing machine relative to φ just as the original assignment captured computability by a regular Turing machine. As the use of the word oracle indicates, φ need not be computable. In case the oracle is incomputable, the functions computable relative to it may be incomputable as well. However, if one does not consider φ to be fixed but instead as an input to the computation, one recovers a realistic model of computation and this is the model that is at the basis of computable analysis. We further adapt to the conventions of computable analysis by interpreting the discrete input to the oracle Turing machine as a curried function input. That is, for a given functional input φ, an oracle Turing machine is considered to return an element of Baire space if the relativized computation with oracle φ terminates for each possible input. If the computation on oracle φ diverges for any of the possible inputs, the return value is considered undefined.

To make this precise, first recall that we use the notations $\mathcal{B} = A^Q$ and $\mathcal{B}' = A'^{Q'}$. Given a function $M : \mathbb{N} \times \mathcal{B} \times Q' \to \mathrm{opt}\, A'$ we consider the multifunction $F_M \colon \mathcal{B} \rightrightarrows \mathcal{B}'$ defined by

$$F_M(\varphi) := \{\psi \in \mathcal{B}' \mid \forall q', \exists n \colon M(n, \varphi, q') = \mathrm{Some}(\psi(q'))\}.$$

One may repeat the discussion for the Φ assignment above to verify that for each operator computable by an oracle Turing machine, there exists some M definable in Gödels system T such that F_M is the multivalued function assigned to the operator computed by the oracle Turing machine. In the same way as was done for the unrelativized case, one may force singlevaluedness. It is also still true that for any M that is computable by an oracle Turing machine this singlevalued selection of F_M is computable by an oracle Turing machine. Thus, under the assumption that all functions definable in Coq are computable, and whenever all the question and answer types come with canonical bijections with the natural numbers, we may use a definition of M as certificate for the computability of an operator. In the rest of the paper, whenever we mention computability, we mean computability in this sense.

As an informal example consider sections of elements of Baire space, i.e. let all question and answer sets be the natural numbers so that $\mathcal{B}$ and $\mathcal{B}'$ both are $\mathbb{N}^\mathbb{N}$. Consider the function M that on input of an effort $n \in \mathbb{N}$, some $\varphi \in \mathcal{B} = \mathbb{N}^\mathbb{N}$ and a $q' \in Q' = \mathbb{N}$ returns the value

$M(n, \varphi, q') \in \operatorname{opt}(A') = \operatorname{opt}(\mathbb{N})$ given by

$$M(n, \varphi, q') := \begin{cases} \mathrm{Some}(n) & \text{if } \varphi(n) = q' \\ \mathrm{None} & \text{otherwise.} \end{cases}$$

Then the values of the corresponding operator F_M are given by

$$F_M(\varphi) = \{\psi \in \mathbb{N}^{\mathbb{N}} \mid \forall q', \exists a' \colon \varphi(a') = q' \wedge \mathrm{Some}(a') = \mathrm{Some}(\psi(q'))\} = \{\psi \mid \forall q' \colon \varphi(\psi(q')) = q'\}.$$

That is $F_M(\varphi)$ is the set of all sections of φ. Consequentially, the domain of F_M is the collection of all surjective elements of Baire space. As a given surjective function can have several different sections, F_M is not singlevalued. However, one can make the values unique by deciding on the minimal section, where a section ψ of a surjection φ is called minimal if it is pointwise less or equal to any other section. An M' such that $F_{M'}$ is the specification of being a minimal section can be found by following the construction of a singlevalued selection for the Φ assignment given above.

A version of this example that is of particular interest for us, we have made formal:

Example 2.3 (`examples/continuous_search.v`). Consider the case where $A := \{0, 1\}$ are the Booleans, $Q' := \{\star\}$ is the canonical one point set and the other question and answer sets are the natural numbers. This means that $\mathcal{B} = \{0, 1\}^{\mathbb{N}}$ is Cantor space and $\mathcal{B}' = \mathbb{N}^{\{\star\}}$ can be understood as the natural numbers by identifying $\psi \in \mathcal{B}'$ with $\psi(\star) \in \mathbb{N}$. Consider the operator $F \colon\subseteq \mathcal{B} \to \mathcal{B}'$ that on input of some χ from Cantor space that is not the constant function returning 1, returns the first $q' \in \mathbb{N}$ such that $\chi(q') = 0$. Then $F = F_M$ if we choose for $M \colon \mathbb{N} \times \mathcal{B} \times \{\star\} \to \operatorname{opt}(\mathbb{N})$ the function such that $M(n, \chi, \star)$ equals $\mathrm{Some}(k)$ if k is the smallest number no bigger than n such that $\chi(k) = 0$ and None if no such k exists. Just like in the informal example above, this M may be constructed by first considering the multifunction returning any zero and then moving to a singlevalued selection function.

Note that the Φ and the F assignments differ considerably in their interpretation of what is considered input and output. As sketched above this is not relevant for finding a singlevalued selection. However, it renders different operations natural for the Φ and the F assignment and the natural operations for the latter tend to be more problematic than for the former. As illustration of this, and as an additional motivation for the next section, let us briefly look into composition.

First consider the Φ assignment. Given functions $N \colon \mathbb{N} \times R \to \operatorname{opt} S$ and $N' \colon \mathbb{N} \times S \to \operatorname{opt} T$ one may define a new function $N' \circ_\Phi N \colon \mathbb{N} \times R \to \operatorname{opt} T$ via

$$N' \circ_\Phi N(\langle n, m \rangle, r) := \begin{cases} N'(n, t) & \text{if } N(m, r) = \mathrm{Some}\, t \\ \mathrm{None} & \text{otherwise,} \end{cases}$$

where $\langle \cdot, \cdot \rangle$ is the Cantor (or any standard) pairing function. This captures the relational composition in the sense that $\Phi_{N' \circ_\Phi N} = \Phi_{N'} \circ_R \Phi_N$ which in turn tightens $\Phi_{N'} \circ \Phi_N$. Under the assumption that N is monotone one may further simplify the construction by replacing the pair by the maximum. Let us try to adapt the above to the setting of the F assignment. Given M' and M one would most likely be interested in $F_{M'} \circ F_M$. For simplicity let us assume that $F_{M'}$ and F_M are singlevalued so that the values of the composition are given by $(F_{M'} \circ F_M)(\varphi) := F_{M'}(F_M(\varphi))$. A straightforward adaption of the proof for the Φ assignment does not give information about this operator but instead about an operation that would be more natural to consider for functionals. Namely given functionals $\mathcal{F} \colon \mathcal{B} \times Q' \to A'$ and $\mathcal{G} \colon \mathcal{B} \times Q'' \to Q'$ it makes a statement about the functional $(\varphi, q'') \mapsto \mathcal{F}(\varphi, \mathcal{G}(\varphi, q''))$.

The construction for functionals that corresponds to what we were originally interested in is “functional substitution”, spells out $(\varphi, q'') \mapsto \mathcal{F}(\lambda q'.\mathcal{G}(\varphi, q'), q'')$ in the language for functionals and requires different typing of $\mathcal{F}$ and $\mathcal{G}$.

Let us thus assume we are given $M\colon \mathbb{N} \times \mathcal{B} \times Q' \to \mathrm{opt}\, A'$ and $M'\colon \mathbb{N} \times \mathcal{B}' \times Q'' \to \mathrm{opt}\, A''$ and let us try to find some $M' \circ_F M\colon \mathbb{N} \times \mathcal{B} \times Q'' \to \mathrm{opt}\, A''$ such that $F_{M' \circ_F M}$ is a tightening of $F_{M'} \circ F_M$. Due to the definition of the F assignment, we can recover from M for each fixed effort n and input $\varphi \in \mathcal{B}$ an approximation $M(n, \varphi, \cdot)\colon Q' \to \mathrm{opt}(A')$ to possible functional inputs to M'. However, M' expects an input of type $Q' \to A'$ and not of type $Q' \to \mathrm{opt}(A')$ and to move from the latter to the former one has to pick a value whenever None occurs as return-value. Without further information about M' it is not clear why the choices of these values should be irrelevant. And in particular it is not clear why the return values of M' on an arbitrary extension should have anything to do with the return-value of the composition. However, Coq is consistent with functional extensionality and any functions defined in Coq without use of non-computational axioms can be evaluated in a finite amount of time. One might thus tend to believe that if M' is defined in this way, then its value on fixed discrete input only relies on a finite number of the return values of its functional input. If the original φ is from the domain of F_M it is thus possible to choose the effort big enough for the way in which we extend to be irrelevant. The additional information that is needed about M' for being able to carry out this kind of composition concretely is effective information about its continuity as presented in the upcoming section.

2.2. **Continuity of partial operators between naming spaces.**

This section presents an information theoretic development of a notion of continuity of operators between naming spaces. The Incone library provides proofs that the definitions presented here are equivalent to more traditional notions of continuity, but the discussion of these equivalences is postponed to Section 4 since it requires some background about metric spaces and topology that are not necessary for the presentation in the current section. For the following fix some types Q, A, Q' and A' and set $\mathcal{B} := A^Q$ and $\mathcal{B}' := A'^{Q'}$.

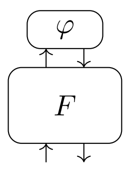


FIGURE 2. A continuous operator.

Intuitively continuity says that the values of an operator $F\colon \mathcal{B} \to \mathcal{B}'$ interpreted as functional of type $F\colon \mathcal{B} \times Q' \to A'$ only depend on finite information about the values of the functional input from $\mathcal{B}$ and thus can be thought of as being represented by a diagram as depicted in Figure 2. Mathematically, continuity can be described as follows: A function $F\colon \mathcal{B} \to \mathcal{B}'$ is **continuous** if for any element φ of $\mathcal{B}$ and any $q' \in Q'$ there exists a **certificate**, i.e. a finite list $\mathbf{q} \in \mathrm{seq}\, Q$ such that for any ψ that coincides with φ on $\mathbf{q}$ it holds that $F(\psi)(q') = F(\varphi)(q')$. Here, two functions are said to **coincide on** a finite list $\mathbf{q}$ if $\varphi(q) = \psi(q)$ for any q that appears in $\mathbf{q}$. A partial operator $F\colon \subseteq \mathcal{B} \to \mathcal{B}'$ is continuous if for all $\varphi \in \mathrm{dom}(F)$ and $q' \in Q'$ there exists a certificate, i.e. a finite list $\mathbf{q} \in \mathrm{seq}\, Q$ such that the above statement holds for any $\psi \in \mathrm{dom}(F)$ that coincides with φ on $\mathbf{q}$.

Example 2.4 (`examples/continuous_search.v`)**.** Consider the function $F_0\colon \mathbb{N}^\mathbb{N} \to \mathbb{N}^\mathbb{N}$ defined by $F_0(\varphi)(n) := \varphi(n) + \varphi(0)$. Then for any inputs φ and n the finite list $\mathbf{q} := (n, 0)$ is a certificate and thus F_0 is continuous. For the operator $F_1(\varphi)(n) := \varphi(\varphi(n))$ the list $\mathbf{q} := (\varphi(n), n)$ is appropriate.

Most functions that can be explicitly defined are continuous. For instance any function definable in Gödels system T is continuous and as long as no strictly non-computational axioms are involved, it is reasonable to assume that the functions definable in Coq are computable and therefore continuous (compare discussion in Section 1.2).

The same remains true for operators whose specification can be given as F_M for some function $M\colon \mathbb{N}\times\mathbb{N}^{\mathbb{N}}\times\mathbb{N}\to \operatorname{opt}\mathbb{N}$ such that M has a Coq-definition that does not rely on non-computational axioms. For instance for the search operator F from Example 2.3 the function M can be defined in Coq only using very elementary means and the list $(0,\ldots,F(\varphi)(\star))$ is a certificate for functional input φ and discrete input n. The search operator does not have a continuous total extension. Thus, one should not expect such an extension to be definable in Coq without reliance on non-computational axioms.

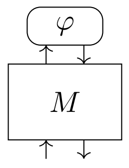


Figure 3. A computable operator.

The definition of continuity in the Incone library follows the mathematical definition given earlier mostly literally. It superficially looks more complicated due to the use of multifunction to substitute partial functions, but the definition is chosen such that it implies a continuous multifunction to be singlevalued and does thus only really apply to partial functions. Another difference is that instead of a list for each question the definition of continuity in Incone uses a Skolem-function $L\colon Q'\to \operatorname{seq} Q$. This switches the order of the corresponding existential and universal quantification. Whenever an appropriate choice principle is available, these definitions are equivalent (`choice_cont`). The definition used in Incone has the advantage that it allows for a fully constructive proof of the fact that the composition of continuous operators is continuous.

Theorem 2.5 (`cont_comp`). *Let $F\colon\subseteq\mathcal{B}\to\mathcal{B}'$ and $G\colon\subseteq\mathcal{B}'\to\mathcal{B}''$ be continuous partial operators. The operator $F\circ G\colon\subseteq\mathcal{B}\to\mathcal{B}''$ is continuous.*

summary. The idea behind the proof is that the certificate functions L and L' whose existence is guaranteed by the continuity of F and G can be interpreted as multivalued functions and composed relationally to obtain a certificate function for the composition of the operators. Furthermore, the needed relational composition can be realized constructively on the level of combining lists. □

As we compare different notions of continuity in the later chapters, let us briefly discuss sequential continuity on naming spaces. Let $\mathcal{B}=A^Q$, note that $\mathcal{B}$ is a naming space if Q and A are countable and non-empty but for the following definition we do not need these assumptions. An element $\varphi\in\mathcal{B}$ is said to be the **limit** of a sequence $(\varphi_n)_{n\in\mathbb{N}}\subseteq\mathcal{B}$ if for each fixed argument $q\in Q$ the sequence $(\varphi_n(q))$ is eventually constantly $\varphi(q)$. Formally

$$\lim\nolimits_{\mathcal{B}}(\varphi_n)=\varphi \quad\iff\quad \forall q,\exists n_0,\forall n\geq n_0\colon \varphi_n(q)=\varphi(q).$$

If a sequence in Baire space has a limit, this limit is uniquely determined (`lim_sing`) and thus the above defines a partial function $\lim_{\mathcal{B}}\colon\subseteq\mathcal{B}^{\mathbb{N}}\to\mathcal{B}$.

A partial operator $F\colon\subseteq\mathcal{B}\to\mathcal{B}'$ is called **sequentially continuous** if for any $\varphi\in\operatorname{dom}(F)$ and any sequence $(\varphi_n)_{n\in\mathbb{N}}\subseteq\operatorname{dom}(F)$ such that $\lim_{\mathcal{B}}(\varphi_n)=\varphi$ it also holds that $\lim_{\mathcal{B}'}(F(\varphi_n))=F(\varphi)$. It is well known that the topological structure of Baire space $\mathbb{N}^{\mathbb{N}}$ is such that sequential continuity of partial operators on Baire space is equivalent to their continuity and the Incone library includes a formal proof of this and that it remains true for naming spaces. However, this is a classical fact and constructively sequential continuity need not imply continuity, thus the library separates the equivalence into two implications.

Theorem 2.6 (`cont_scnt` and `scnt_cont`). *A partial operator between naming spaces is continuous if and only if it is sequentially continuous.*

Section 4.1 discusses how to prove that any naming space is metrizable, or more specifically it defines a metric on $\mathcal{B}$ from each enumeration of Q. It also presents proofs that the notions of convergence and continuity induced by this metric coincide with those given above. Thus, the formal proofs that continuity and sequential continuity are equivalent to what is described in Section 4.2 imply the above theorem. However, metric spaces use real numbers, which leads to the axioms of the real numbers appearing in the assumptions printed when inspecting the proofs. This is even though the proofs do not use these axioms in an essential way. Thus, the above statement is given a separate proof in the Incone library.

2.3. **Construction of a universal and some of its properties.**

Recall that a naming space is a space of functions whose target and argument spaces are countable and non-empty. A **continuous universal**, or just **universal**, is an assignment that for each pair $\mathcal{B}$ and $\mathcal{B}'$ of naming spaces provides another naming space $\mathcal{B}''$ and an operation $F_{M(\cdot)}\colon \mathcal{B}'' \to \mathcal{B} \rightrightarrows \mathcal{B}'$ such that each of its values is continuous and for each continuous $F\colon\subseteq \mathcal{B} \to \mathcal{B}'$ there exists an element $\psi \in \mathcal{B}''$ such that $F_{M(\psi)}$ tightens F. That is: A universal provides a way to code any continuous operator between naming spaces by an element of another naming space. We call such a code, i.e. some ψ such that $F_{M(\psi)}$ tightens F, an **associate** of F with respect to the universal or just an associate if the universal is clear from the context. Note that the type of ψ can be inferred from the universal together with the type of F.

Let us give some motivation for the terminology chosen here. To justify the term "universal" note that replacing all the naming spaces by the set of finite binary strings and the word "continuous" by "computable" one recovers a specification that is fulfilled by the universal Turing machine. While the construction of a universal Turing machine takes some effort, continuous universals can be chosen very simple: Classically any naming space can be replaced by Baire space and modulo this one may use Kleene-Kreisel associateship to obtain a universal [Kle59, Kre59]. A more popular variant in computable analysis is to move from Baire space to the space of infinite binary strings, i.e. Cantor space and use Weihrauch's η operator [Wei00]. The former of these is conceptually more well adapted to our setting and there are several excellent sources to read up about its background [LN15, EX16]. A previous version of Incone's universal closely followed generalization of Kleene-Kreisel associateship first presented by van Oosten [vO11]. The current universal modifies that construction slightly by moving away from the idea that questions should be asked sequentially and allowing for a finite number of questions to be asked in parallel.

A mathematical description of Incone's universal can be given as follows: for fixed naming spaces $\mathcal{B} = A^Q$ and $\mathcal{B}' = A'^{Q'}$ set $Q'' := \operatorname{seq} A \times Q'$ and $A'' := \operatorname{seq} Q + A'$. That is let $\mathcal{B}'' := (\operatorname{seq} Q + A')^{\operatorname{seq} A \times Q'}$. That $\mathcal{B}''$ is a naming space, i.e. that Q'' and A'' are countable and non-empty, follows directly from $\mathcal{B}$ and $\mathcal{B}'$ being naming spaces. Assign to $\psi \in \mathcal{B}'$ the multifunction $F_{U(\psi)}\colon \mathcal{B} \rightrightarrows \mathcal{B}'$ defined as follows: $\varphi' \in F_{U(\psi)}(\varphi)$ if and only if for any $q' \in Q'$ there exists an N and a finite sequence of lists $(\mathbf{a}_i)_{i\in\{1,\dots,N\}} \subseteq \operatorname{seq} A$ such that for $i < N$ it holds that $\psi(\mathbf{a}_i, q') = ?\,\mathbf{q}$ for some $\mathbf{q} \in \operatorname{seq} Q$ (where ? denotes the left inclusion in the sum) and $\mathbf{a}_{i+1} = \mathbf{a}_i \mathbin{+\!\!+} (\varphi(q_1), \dots, \varphi(q_{|\mathbf{q}|}))$ and $\psi(\mathbf{a}_N, q') = !\,\varphi'(q')$ (where ! denotes the right inclusion of the sum). The above is best understood as running a small while program: For given functional input φ and input q', the universal attempts to extract a value from ψ by first calling it on input (ϵ, q'), and then branching according to the return value: if the return value is a list of questions it updates the list in the first argument with the respective

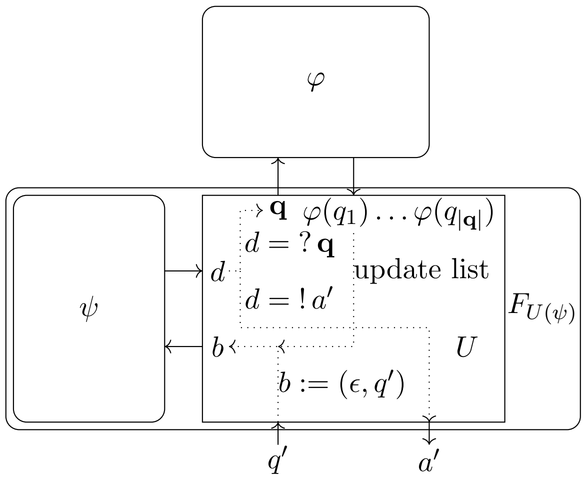


Figure 4. The universal used in Incone.

answers according to φ. If the return value is an answer, it interprets this answer as the return value of the operator (cf. Figure 4).

The notation we used strongly suggests that the universal can be specified by means of a function $U\colon \mathcal{B}'' \to \mathbb{N} \times \mathcal{B} \times Q \to \operatorname{opt} A'$ where the universal as described above can be recovered using the operator assignment $M \mapsto F_M$ described in Section 2.1 via $\psi \mapsto F_{U(\psi)}$. Indeed, Incone defines such a function `U` using only very elementary means. We refrain from writing out the exact definition of U here and point the interested reader to the Incone library, where the definition can be printed and a formal proof that it fulfills the above specification can be found (`FU_spec`). Furthermore, `U`(ψ) is always monotone in the sense of the previous section (`U_mon`) and in particular $F_{U(\psi)}$ is always singlevalued (`FU_sing`).

That this actually defines a universal can be separated into two statements. Let us first argue that all operators of the form $F_{U(\psi)}$ are continuous. We actually prove the stronger statement, that from an associate ψ one can obtain a self modulating modulus of continuity for $F_{U(\psi)}$. Let us start by introducing the notion of a modulus of continuity for a multifunction $F\colon \mathcal{B} \rightrightarrows \mathcal{B}'$. Recall from Section 2.1 that we decided to call a finite list a certificate for $\varphi \in \mathcal{B}$ and $q' \in Q'$ if for any ψ that coincides with φ on this list the return-values of F are identical. Note that previously we assumed F to be a partial function and here we talk about a multifunction, so we have to elaborate. Call a list certificate for φ and q' if for each ψ that coincides with φ on this list and any $\varphi' \in F(\varphi)$ and $\psi' \in F(\psi)$ it holds that $\varphi'(q') = \psi'(q')$. Note that if F is singlevalued, being a certificate in this sense is equivalent to being a certificate for the corresponding multivalued function. Furthermore note that the existence of a certificate implies that the elements of $F(\varphi)$ can only take one possible value in q'. In particular, if continuity of multifunctions is defined as before, i.e. by requiring a certificate to exist for all inputs, the corresponding notion implies singlevaluedness and should be understood as a notion of continuity for partial functions specified by relations and not as a notion for multifunctions (cf. [PZ13]).

Define the **multivalued modulus of continuity** $C_F\colon \mathcal{B} \rightrightarrows \operatorname{seq} Q^{Q'}$ of a multifunction $F\colon \mathcal{B} \rightrightarrows \mathcal{B}'$ by

$$C_F(\varphi) := \{L\colon Q' \to \operatorname{seq} Q \mid \forall q'\colon L(q') \text{ is a certificate for } \varphi \text{ and } q'\}.$$

The definition of C_F makes sense for any multifunction $F\colon \mathcal{B} \rightrightarrows \mathcal{B}'$ but $C_F(\varphi)$ can only be non-empty if $F(\varphi)$ has at most one element. The continuous partial operators $F\colon\subseteq \mathcal{B} \to \mathcal{B}'$ can be specified as exactly those multifunctions such that the domain of C_F is a super-set of the domain of F (`cont_spec`). If F is continuous, then we call any partial choice function $\mu\colon\subseteq \mathcal{B} \to \operatorname{seq} Q^{Q'}$ of C_F a **modulus of continuity** for F. Note that if μ is a modulus of continuity of F, then C_μ has the same type as C_F. Thus it makes sense to call a modulus of continuity μ for F **self-modulating** if it is continuous and a modulus of continuity for itself, that is if it is also a choice function for C_μ.

Let ψ be an associate of a partial operator F with respect to the universal U from above, i.e. let ψ be such that $F_{U(\psi)}$ tightens F. A self-modulating modulus for F can readily be obtained from ψ by tracking the queries in the evaluation of the universal. The same can be done for the values that the universal calls the function ψ on and one defines functions U_Q and U_S such that $F_{U_Q(\psi)}$ and $F_{U_S(\psi)}$ are the corresponding operators.

Theorem 2.7 (`FqM_mod_FU`, `FqM_mod_FqM` and `FqM_mod_FsM`)**.** *For any fixed ψ of appropriate type the operator $F_{U_Q(\psi)}$ is a modulus of continuity for $F_{U(\psi)}$, for itself and for $F_{U_S(\psi)}$.*

The universal is used in Incone to construct exponentials in the category of represented spaces or, more simply put, to construct spaces of functions. The above Theorem 2.7 in particular implies that for any ψ the operator $F_{U(\psi)}$ is continuous. Its more general statement is enough to provide what is needed to prove the evaluation procedure on the constructed space of functions to be a continuous operation. The functions U_Q and U_S are of more theoretical than practical importance. For the purpose of inspecting the evaluation of an associate Incone provides a more useful function `gather_queries` that on input of an associate ψ, a functional input φ, a discrete input q' and an effort n returns the list of all queries posed up to the n-th loop of the evaluation of the universal on these values (see `examples/KleeneKreisel.v` for examples).

To finish the proof that U is an universal it is left to show that any partial continuous operator has an associate with respect to U.

Theorem 2.8 (`U_universal`)**.** *Any partial continuous operator $F\colon\subseteq \mathcal{B} \to \mathcal{B}'$ between naming spaces has an associate with respect to the universal U described above. I.e. there exists some ψ such that $F_{U(\psi)}$ tightens F.*

We do not give a lot of details for this proof here, but let us sketch the most important parts and point out some interesting details. Let μ be any function that chooses through the multivalued modulus of continuity C_F of F and let f be any function that chooses through F. For any fixed enumeration $(q_i)_{i\in\mathbb{N}}$ of Q, one can attempt to define an associate ψ for F as follows: On input $(\mathbf{a}, q')$ interpret the list $\mathbf{a}$ as a partial function by assuming that its elements are the return values on the first $|\mathbf{a}|$ elements mentioned in the enumeration of Q, i.e. interpret it as the finite function such that $q_i \mapsto a_i$ for all $0 < i \leq |\mathbf{a}|$. Extend this function to a total function $\varphi_{\mathbf{a}}$ that is from the domain of F if this is possible. If $\mu(\varphi_{\mathbf{a}}, q') \subseteq (q_1, \ldots, q_{|\mathbf{a}|})$ then return $!\, f(\varphi_{\mathbf{a}})(q')$, otherwise ask for the $(|\mathbf{a}|+1)$-st element mentioned in the enumeration.

As μ is a modulus for F, evaluating this associate using the universal U only results in correct return values. That $\mu(\varphi, q')$ must be contained in some initial segment of the enumeration and the sequence of functions $\varphi_{\mathbf{a}}$ converges to φ gives hope that the iteration may often be finite. Without further assumptions about μ, however, this can not be proven and the function ψ defined above might fail to be an associate of F. Since it is always possible to extract a self-modulating modulus from an associate it may not be surprising that the proof can be completed if the modulus is additionally known to be self-modulating. In the library the existence of a self-modulating modulus is proven by picking μ to be minimal with respect to subset inclusion under the additional condition that it can only return initial segments with respect to the enumeration of Q. We call such a modulus a **minimal modulus** with respect to the enumeration.

Lemma 2.9 (`mod_minmod`). *A minimal modulus is always self-modulating.*

The reason for the indefinite article in this lemma is that the existence of a minimal modulus is not constructively provable [TvD88], and indeed the Coq proof of its existence is classical and relies on a fairly strong choice principle, namely one that covers naming spaces and is thus strictly stronger than countable choice.

Lemma 2.10 (`exists_minmod`). *For any continuous partial operator there exists a minimal modulus of continuity.*

A minimal modulus is unique up to choice of the sequence and as it is known to be impossible to constructively prove the existence of an extensional way to obtain a modulus of continuity, the proof is inherently classical. In particular there are computable operators on Baire space whose minimal modulus with respect to the identity as enumeration is not computable. By contrast, from a computable associate a computable self-modulating modulus can be read of. Incone thus makes some efforts to avoid the use of the minimal modulus and instead allow to construct a self-modulating modulus from additional information about the operator. The details of this leave the scope of this paper we do not elaborate further on this aspect.

Finally, the library defines a function D that exchanges the arguments of the universal.

Theorem 2.11 (`D_spec`). *For all φ and ψ it holds that $F_{U(\psi)}(\varphi) = F_{U(D\varphi)}(\psi)$.*

Here, the types have been purposefully omitted, details can be found in the library. Note that, while $U(\psi)$ has the more complicated type and is interpreted as $F_{U(\psi)}$ using the operator assignment, D can be directly interpreted as a function or using the associated multifunction. The above theorem is interesting because it is related to the Cartesian closure of the category of represented spaces (see Section 3 for details on the category). However, it falls short in strength as it only considers a special case in which it is not necessary to talk about tupling of elements of naming spaces.

Corollary 2.12 (`FsM_mod_FU`, `FsM_mod_FsM` and `FsM_mod_FqM`). *Theorem 2.7 remains true if U_Q and U_S are exchanged and ψ and φ are exchanged.*

3. Represented spaces and continuous realizability

Computable analysis is the theory of computation on sets of continuum cardinality. To make such spaces available to computation, computable analysis considers encodings of such sets over Baire space $\mathcal{B} = \mathbb{N}^{\mathbb{N}}$. Typically, such an encoding is understood to be a surjective partial function $\delta\colon \subseteq \mathcal{B} \rightrightarrows X$ from Baire space to the set that is called a representation. Instead of a partial function, the representation can also be considered a singlevalued multifunction and we will go back and forth between these two views seamlessly. According to Lemma 2.1 the requirement of being surjective translates to the corresponding multifunction being co-total.

In the formal development in INCONE everything is formulated using singlevalued, co-total multifunctions. Moreover, instead of using Baire space, $\mathcal{B}$ is allowed to be any naming space in the sense of the previous chapter, i.e. any space of the form $\mathcal{B} = A^Q$ where A and Q are countable and non-empty.

A **representation** of a set X is defined as a naming space $\mathcal{B}$ together with a singlevalued, co-total multifunction $\delta\colon \mathcal{B} \rightrightarrows X$. Note that $\mathcal{B}$ can be inferred from δ. This justifies the fact to just call δ the representation. Just like in computable analysis we call those $\varphi \in \mathcal{B}$ such that $x \in \delta(\varphi)$ the names of x. The surjectivity of δ can now be reformulated as each element of the space having at least one name and the singlevaluedness as each name uniquely identifying the element. An alternate maybe more descriptive phrase for "name" would be "description". As $\mathcal{B}$ is a naming space it is of the form $\mathcal{B} = A^Q$ for some countable and non-empty sets Q of questions and A of answers. A useful interpretation of φ being a name is that it provides on-demand information about x. If φ is a name of x then for each question $q \in Q$ about the abstract object x the value $\varphi(q) \in A$ can be considered a valid answer to that question. A **represented space** is a pair $\mathbf{X} = (X, \delta_{\mathbf{X}})$ of a set X and a representation $\delta_{\mathbf{X}}$ of X. When appropriate we decorate the naming space of the representation and its question and answer sets with indices as well, i.e. we assume $\delta\colon \mathcal{B}_{\mathbf{X}} \rightrightarrows X$, where $\mathcal{B}_{\mathbf{X}} = A_{\mathbf{X}}^{Q_{\mathbf{X}}}$ and $Q_{\mathbf{X}}$ and $A_{\mathbf{X}}$ are countable and non-empty.

As an example let us equip the real numbers with a representation.

Example 3.1 (`examples/Q_reals.v`)**.** Let $X := \mathbb{R}$ be the set of real numbers and pick the question and answer sets to be the rational numbers, i.e. $Q_{\mathbb{R}} = A_{\mathbb{R}} := \mathbb{Q}$ and thus $\mathcal{B}_{\mathbb{R}} = \mathbb{Q}^{\mathbb{Q}}$. Clearly $\mathbb{Q}$ is countable and non-empty so that $\mathcal{B}_{\mathbb{R}}$ is a naming space. Let $\delta_{\mathbb{R}}\colon \subseteq \mathcal{B}_{\mathbb{R}} \to \mathbb{R}$ be the partial function specified by

$$\delta_{\mathbb{R}}(\varphi) = x \quad\iff\quad \forall \varepsilon \in \mathbb{Q}, 0 < \varepsilon \implies |x - \varphi(\varepsilon)| \leq \varepsilon.$$

Then $\delta_{\mathbb{R}}$ is a representation of $\mathbb{R}$. Indeed, using the axiomatization of the real numbers provided by COQ's standard library $\delta_{\mathbb{R}}$ can be proven singlevalued and surjective (`rep_RQ_sing` and `rep_RQ_sur`) and we refer to the represented space $(\mathbb{R}, \delta_{\mathbb{R}})$ simply as $\mathbb{R}$.

Computability and continuity of partial operators on naming space can be used to define computability and continuity of functions between represented spaces by means of realizers. For represented spaces $\mathbf{X}$ and $\mathbf{X}'$, a partial operator $F\colon \subseteq \mathcal{B}_{\mathbf{X}} \to \mathcal{B}_{\mathbf{X}'}$ is a **realizer** of a function $f\colon \mathbf{X} \to \mathbf{X}'$ if for each name φ of x the value $F(\varphi)$ is defined and a name of $f(x)$ (cf. Figure 5). A function between represented spaces is **continuous** if it has a continuous realizer and **computable** if it has a computable realizer. The represented spaces form a Cartesian closed category both if the continuous functions are used as morphisms, and if the computable functions are used. The use of the word "continuous" here is often contested and many say it should be reserved for the topological concept. Others argue that the above is the correct notion and that the topological one is only a syntactic approximation of it. It is one of the objectives of this paper to give a formal proof that the above and topological continuity have a considerable overlap in that they coincide for metric spaces. For instance for the real numbers represented as indicated in Example 3.1 it is true that a function from $\mathbb{R} \to \mathbb{R}$ has a continuous realizer if and only if it is continuous

$$\begin{tikzcd} X \arrow[r, "f"] & Y \\ \mathcal{B} \arrow[u, "\delta_{\mathbf{X}}"] \arrow[r, "F"'] & \mathcal{B}' \arrow[u, "\delta_{\mathbf{Y}}"'] \end{tikzcd}$$

FIGURE 5. $F\colon \subseteq \mathcal{B} \to \mathcal{B}'$ is a realizer of $f\colon \mathbf{X} \to \mathbf{Y}$.

in any of the more traditional ways to make sense of being continuous. This is more generally true for functions between metric spaces and we go into detail about this in Section 4.

It is possible to extend the definition of being a realizer to allow for both the realizer and the realized function to be multivalued. We refrain from stating the elementary definition as for the purpose of this paper it is sufficient to know that the extended definition fulfills the following specification:

Lemma 3.2 (rlzr_spec). *A multifunction $F\colon \mathcal{B}_{\mathbf{X}} \rightrightarrows \mathcal{B}_{\mathbf{X}'}$ realizes another multifunction $f\colon \mathbf{X} \rightrightarrows \mathbf{X}'$ if and only if $\delta_{\mathbf{X}'} \circ F$ tightens $f \circ \delta_{\mathbf{X}}$.*

The above lemma can also be used backwards to express the notion of a tightening as a special case of being a realizer by using the identity function as representation (`id_rlzr_tight`). One may even further extend the definition of a realizer by dropping the requirement that the representations are singlevalued. In the terminology of computable analysis this would mean dealing with multi-representations and while in the manipulation of discrete data the use of non-singlevalued encodings is fairly common, in computable analysis the use of multirepresentations is rare. The above characterization does not generalize to multirepresentations. For the full definitions we point the interested reader to the Rlzrs library.

Here, we are only interested in representations and mostly in continuous, and therefore singlevalued, realizers. However, in Section 4.3 we discuss closed choice on the natural numbers as an elementary example of a multifunction between represented spaces. We call a multifunction between represented spaces **continuously realizable** if there exists a continuous realizer in the sense of the previous lemma. Note that under the usual identification of a function with the multifunction that uniquely specifies it, continuity is a special case of continuous realizability and we sometimes use the latter to distinguish it from other notions of continuity if confusion is possible. One reason that the use of continuously realizable multifunctions is common in computable analysis is that continuity of functions often fails for extensionality reasons. For instance, one may formulate a multifunction corresponding to the parallel or on the space introduced in Section 4.3 as Sierpinski space to see that there exist continuously realizable functions that do not have any continuous choice function. A special case where exactly the opposite behavior appears is that of a naming space equipped with the identity function as a representation: a multifunction between naming spaces is continuously realizable if and only if it has a singlevalued tightening that is continuous in the sense of Section 2.2 and therefore allows for a continuous choice function.

That continuity and continuous realizability is preserved under composition follows from content of the Rlzrs library together with the fact that continuity of operators on Baire space is preserved under composition (Theorem 2.5).

Lemma 3.3 (comp_cont and comp_hcr). *The composition of continuous functions is continuous and the composition of continuously realizable multifunctions is continuously realizable.*

3.1. Examples and basic constructions such as products and sums.

Now that we can talk about continuity and computability on the real numbers, a reasonable next step is to attempt to prove addition and multiplication computable. Both of these functions are of type $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and to make sense of continuity of functions of these types we need to specify how $\mathbb{R} \times \mathbb{R}$ should be made a represented space. The Incone library automatically generates such a represented space $\mathbf{X} \times \mathbf{Y}$ from arbitrary represented spaces $\mathbf{X}$ and $\mathbf{Y}$ by using the query type $Q_{\mathbf{X}\times\mathbf{Y}} := Q_{\mathbf{X}} + Q_{\mathbf{Y}}$, the answer type $A_{\mathbf{X}\times\mathbf{Y}} := A_{\mathbf{X}} \times A_{\mathbf{Y}}$ and the representation $\delta_{\mathbf{X}\times\mathbf{Y}}$ defined by

$$\delta_{\mathbf{X}\times\mathbf{Y}}(\psi) = (x, y) \qquad \iff \qquad \delta_{\mathbf{X}}(\mathrm{fst} \circ \psi \circ \mathrm{inl}) = x \wedge \delta_{\mathbf{Y}}(\mathrm{snd} \circ \psi \circ \mathrm{inr}) = y.$$

This can be decoded as follows: A name of the pair (x, y) should be a pair (φ, φ') of a name for x and a name for y. Since the set of pairs $\mathcal{B}_{\mathbf{X}} \times \mathcal{B}_{\mathbf{Y}}$ does not have the type that we required a naming space to have, we embed it into the naming space $\mathcal{B}_{\mathbf{X}\times\mathbf{Y}} := (A_{\mathbf{X}} \times A_{\mathbf{Y}})^{Q_{\mathbf{X}}+Q_{\mathbf{Y}}}$. There are several possible choices for $\mathcal{B}_{\mathbf{X}\times\mathbf{Y}}$, but for the one picked by INCONE the projection function $\pi\colon \mathcal{B}_{\mathbf{X}\times\mathbf{Y}} \to \mathcal{B}_{\mathbf{X}}$ can particularly conveniently be expressed by the natural operations on the question and answer spaces, namely $\pi(\psi) := \mathrm{fst} \circ \psi \circ \mathrm{inl}$ and the formula for the second projection is similar.

Proposition 3.4 (`prod_rep_sing`, `prod_rep_sur` and `prod_uprp_cont`)**.** *For any represented spaces $\mathbf{X}$ and $\mathbf{Y}$ the space $(X \times Y, \delta_{\mathbf{X}\times\mathbf{Y}})$ is a represented space and it is the product of $\mathbf{X}$ and $\mathbf{Y}$ in the category of represented spaces.*

Part of verifying the universal property of a product in the category of represented spaces with continuous resp. computable functions as morphisms is to prove the projections computable resp. continuous (`fst_cont` and `snd_cont`).

Example 3.5 (`examples/Q_reals.v`)**.** Addition and multiplication of real numbers is computable (`Rplus_cont` and `Rmult_cont`). As described in more detail in Section 2.1 this should be taken to mean that the operations are continuous and the realizers can be explicitly specified as COQ-functions whose definitions contain no axioms. Indeed, the realizers are defined not through the more complicated operator assignment but more directly using the `F2MF` correspondence. Furthermore, their definition only uses very simple tools and the operations should therefore even be considered primitive recursive.

INCONE additionally proves some other basic functions on product spaces computable. Most notably it provides the possibility to glue continuous functions $f\colon \mathbf{X} \to \mathbf{X}'$ and $g\colon \mathbf{Y} \to \mathbf{Y}'$ together to obtain another continuous function $f \times g\colon \mathbf{X} \times \mathbf{Y} \to \mathbf{X}' \times \mathbf{Y}'$ (`fprd_cont`). More generally, such a construction is provided for continuously realizable multifunctions.

As another basic example of a represented space that is needed below let I be any countable and non-empty set. Set $Q_{\mathbf{I}} := \{\star\}$ and $A_{\mathbf{I}} := I$. Then the function $\delta_{\mathbf{I}}(\varphi) := \varphi(\star)$ makes the pair $\mathbf{I} := (I, \delta_{\mathbf{I}})$ a represented space that is discrete in the following sense:

Lemma 3.6 (`cs_id_dscrt`)**.** *For any countable, non-empty set I the represented space $\mathbf{I}$ described above is discrete in the sense that any function that has $\mathbf{I}$ as its domain is continuous. Moreover, any multivalued function with $\mathbf{I}$ as input space is continuously realizable.*

In particular, the natural numbers can be assigned a discrete represented space. We denote both the set and the represented space of natural numbers by $\mathbb{N}$.

Let us briefly mention a couple of additional constructions. The INCONE library proves that the represented space $\mathbf{1}$ constructed from the unit type as above is a terminal object in the category of represented spaces. It defines for each pair of represented spaces $\mathbf{X}$ and $\mathbf{Y}$ a space $\mathbf{X} + \mathbf{Y}$ that is proven to be the category-theoretical sum. It gives a separate option type construction and proves the resulting space to be isomorphic to $\mathbf{X} + \mathbf{1}$. There is also an elementary construction of the space of finite lists of elements from a represented space $\mathbf{X}$, where asking a question about an element of x results in a list of answers to question for each of the elements of the list.

3.2. **The space of infinite sequences, limits and pointwise operations.** Let I be a countable, non-empty set and let $\mathbf{X}$ be a represented space. Define a represented space $\prod_I \mathbf{X}$ whose underlying set are the functions of type $I \to X$ by setting $Q_{\prod_I \mathbf{X}} := I \times Q_{\mathbf{X}}$, $A_{\prod_I} := A_{\mathbf{X}}$ and

$$(x_i) \in \delta_{\prod_I \mathbf{X}}(\varphi) \quad \iff \quad \forall i \colon I, x_i \in \delta_{\mathbf{X}}(q \mapsto \varphi(i,q)),$$

where (x_i) is short for the function $i \mapsto x_i$.

Proposition 3.7 (`rep_Iprod_sing`, `rep_Iprod_sur` and `cprd_uprp_cont`)**.** *Let I be countable and non-empty. Then $\prod_I \mathbf{X} := (X^I, \delta_{\prod_I \mathbf{X}})$ is a represented space and $\mathbf{X}^\omega := \prod_{\mathbb{N}} \mathbf{X}$ is a countably infinite product in the category of represented spaces and continuous functions.*

The use of the symbol ω instead of $\mathbb{N}$ is to differentiate the space $\mathbf{X}^\omega$ of infinite sequences in $\mathbf{X}$ from the space of functions from the natural numbers to $\mathbf{X}$ that is discussed in the next section. The proof that $\delta_{\prod_I \mathbf{X}}$ is singlevalued assumes functional extensionality and the proof of surjectivity needs a choice principle over I. Since $I = \mathbb{N}$ is by far the most common use-case and I is assumed to be countable anyway, this usually boils down to the axiom of countable choice. The proof of the universal property relies on stronger choice principles and the law of excluded middle. Since the category of represented spaces with computable functions fails to have countably infinite products, the corresponding result is inherently inefficient. Given a sequence of computable functions $f_n \colon \mathbf{Y} \to \mathbf{X}$ the function $F \colon \mathbf{Y} \to \mathbf{X}^{\mathbb{N}}$ that glues them together need only be continuous. Thus we only prove the existence of its realizer. To obtain a computable realizer for F one needs one algorithm that uniformly computes the f_n and not just the existence of an algorithm for each f_n. How far our use of axioms can be optimized in this case is difficult to tell at this point in time since the current proof uses a part of the library that has not yet been optimized in terms of axiom use. Since it is more a sanity result than something that may actually be of use, optimizations here are not our highest priority.

An example of a partial function whose natural domain is a subset of the space of sequences is the limit operator. Consider the multivalued function $\lim_{\mathbf{X}} \colon \mathbf{X}^\omega \rightrightarrows \mathbf{X}$ where $x \in \lim_{\mathbf{X}}(x_n)$ if and only if there is a convergent sequence of names $(\varphi_n) \subseteq \mathcal{B}_{\mathbf{X}}$ and some φ such that φ is a name of x, each φ_n is a name for x_n and the sequence (φ_n) converges to φ in $\mathcal{B}_{\mathbf{X}}$, i.e. $\lim_{\mathcal{B}_{\mathbf{X}}}(\varphi_n) = \varphi$ where the limit in $\mathcal{B}_{\mathbf{X}}$ is taken point-wise as explained in Section 2.2. While the limit operator on Baire space is singlevalued, this need not be true for the limit operator on a general represented space, as can be seen at the example of Sierpinski space that is discussed in Section 4.3. In most spaces that are relevant for numerical analysis, the limit operator is singlevalued but discontinuous. It is often the case that computability of the limit operator can be recovered by restricting it to an appropriate set of efficiently convergent sequences.

Example 3.8 (`examples/Q_reals.v`)**.** The limit operator $\lim_{\mathbb{R}}$ where $\mathbb{R}$ is represented as in Example 3.5 is discontinuous (`lim_not_cont`). Its restriction to those sequences (x_n) that are efficiently Cauchy in the sense that $|x_n - x_m| \leq 2^{-n} + 2^{-m}$ is computable (`lim_eff_hcr`).

To be strict, the example file proves these properties with respect to the metric notion of convergence that is introduced later in Section 4.2. To really obtain what is claimed here one has to additionally use that metric convergence and convergence in the represented space of real numbers are equivalent. The major part of this equivalence is proven in Theorem 4.6 and the rest, namely that the representation of real numbers introduced in Example 3.1 is equivalent (in the sense introduced shortly) to the representation of real numbers as metric space, is also provided by the INCONE library.

A function $f\colon \mathbf{X} \to \mathbf{Y}$ between represented spaces is called sequentially continuous if it preserves limits, i.e. if $\lim_{\mathbf{X}} x_n = x$ implies that $\lim_{\mathbf{Y}} f(x_n) = f(x)$. In general, sequential continuity need not imply continuity and this is not a matter of whether one works constructively or not. From the point of view of a classical mathematician there exist sequentially continuous functions between represented spaces that are simply not continuously realizable. The difference can be eliminated by making additional assumptions about the involved represented spaces. One such assumption is referred to as admissibility and while we do not go into the details here, this condition is of major importance throughout different parts of computable analysis [Sch02b]. Most spaces encountered in practice are admissible, in particular the space of real numbers from Example 3.1, all the representations of metric spaces considered in Section 4, and also the representations of hyper-spaces that are topic of Section 4.3 are admissible. The relation between sequential continuity and continuous realizability is discussed in more detail in Section 4.2.

The INCONE library proves some further lemmas about spaces of sequences that might be useful in applications and should thus not go unmentioned. Two represented spaces $\mathbf{X}$ and $\mathbf{Y}$ are **isomorphic**, in symbols $\mathbf{X} \simeq \mathbf{Y}$, if there exists a continuous bijection with continuous inverse. The spaces are computably isomorphic if there exists a computable bijection with computable inverse. In the special case where the underlying set of two represented spaces are identical we call their representations **equivalent** if the identity function is an isomorphism. While mathematicians are usually fairly liberal in identifying sets and equivalence of representations is a widely used concept, in our formal development isomorphy is by far the more common notion.

Lemma 3.9 (`cprd_prd`). *For any represented spaces $\mathbf{X}$, $\mathbf{Y}$ and countable, non-empty set I it holds that $\prod_I(\mathbf{X} \times \mathbf{Y}) \simeq \big(\prod_I \mathbf{X}\big) \times \big(\prod_I \mathbf{Y}\big)$, the spaces are even computably isomorphic.*

The realizers are defined using very limited means and interpreted using the `F2MF` assignment, thus they should be considered primitive recursive.

Any function $f\colon \mathbf{X} \to \mathbf{Y}$ can be extended to a function $f^I\colon \prod_I \mathbf{X} \to \prod_I \mathbf{Y}$ that applies f pointwise, i.e. is defined by $f^I((x_i)_{i\in I}) := (f(x_i))_{i\in I}$.

Lemma 3.10 (`ptw_cont`). *Whenever f is a continuous function, then also f^I is continuous.*

This lemma has a multivalued variant (`ptw_hcr`). If the realizer of f can be expressed as a COQ-function through the `F2MF` interpretation, then so can f^I. The same should hold true if a realizer of f can be expressed via a COQ function through the operator assignment $M \mapsto F_M$ from Section 2.1, although we have not carried out the details yet.

Another common situation is that an operation $*\colon \mathbf{X} \times \mathbf{Y} \to \mathbf{Z}$ is used to construct an operation $*^I\colon \prod_I \mathbf{X} \times \prod_I \mathbf{Y} \to \prod_I \mathbf{Z}$ via $(x_i) *^I (y_i) := (x_i * y_i)$. For instance the natural operations on diverse spaces of sequences of real numbers are introduced by pointwise applying the arithmetic operations of the real numbers. More concretely, the most common vector space structure on the space $\mathbb{R}^\omega$ of sequences in the real numbers is to set $\mathbf{X} = \mathbf{Y} = \mathbf{Z} := \mathbb{R}$ in the above and use $+^{\mathbb{N}}$ where $+$ is addition of real numbers. A proof that this extension also preserves continuity can be directly obtained from the previous two lemmas.

Corollary 3.11 (`cptw_op_cont`). *If $*$ is a continuous operation, then the corresponding pointwise operation $*^I$ is also continuous.*

As mentioned before, examples where these results are useful is when reasoning about pointwise addition and multiplication as operations on $\mathbb{R}^\omega$. The operation of multiplying a sequence with a scalar is also covered by first embedding the scalars into the sequences as the constant sequences and then using pointwise multiplication.

3.3. **Function spaces and infinite sequences as functions.** Let $\mathbf{X}$ and $\mathbf{Y}$ be represented spaces and denote by $\mathbf{Y}^{\mathbf{X}}$ the collection of all continuous functions from $\mathbf{X}$ to $\mathbf{Y}$. Recall that Section 2.3 gave an explicit description of a continuous universal U. This universal can be used to equip the space $\mathbf{Y}^{\mathbf{X}}$ with a representation as follows: Let $\mathcal{B}_{\mathbf{Y}^{\mathbf{X}}}$ be the space that the universal U points to for input naming spaces $\mathcal{B}_{\mathbf{X}} = A_{\mathbf{X}}^{Q_{\mathbf{X}}}$ and $\mathcal{B}_{\mathbf{Y}} = A_{\mathbf{Y}}^{Q_{\mathbf{Y}}}$. This expands to $Q_{\mathbf{Y}^{\mathbf{X}}} = \operatorname{seq} A_{\mathbf{X}} \times Q_{\mathbf{Y}}$ and $A_{\mathbf{Y}^{\mathbf{X}}} = \operatorname{seq} Q_{\mathbf{X}} + A_{\mathbf{Y}}$, where $\mathcal{B}_{\mathbf{Y}^{\mathbf{X}}} = A_{\mathbf{Y}^{\mathbf{X}}}^{Q_{\mathbf{Y}^{\mathbf{X}}}}$. Note that a function (as opposed to a partial or multifunction) is uniquely determined by each of its realizers. Thus there exists a unique partial function $\delta_{\mathbf{Y}^{\mathbf{X}}} \colon \mathcal{B}_{\mathbf{Y}^{\mathbf{X}}} \rightrightarrows \mathbf{Y}^{\mathbf{X}}$ specified by

$$\delta_{\mathbf{Y}^{\mathbf{X}}}(\psi) = f \qquad \iff \qquad F_{U(\psi)} \text{ realizes } f.$$

Put differently, although $\delta_{\mathbf{Y}^{\mathbf{X}}}$ is specified as a multifunction, it is singlevalued. Any $f \in \mathbf{Y}^{\mathbf{X}}$ is continuous which by definition means that there exists some continuous $F \colon \subseteq \mathcal{B}_{\mathbf{X}} \to \mathcal{B}_{\mathbf{Y}}$ that realizes f. Because U is a universal there exists some ψ such that $F_{U(\psi)}$ tightens F (cf. Theorem 2.8). As being a realizer is preserved under tightening also $F_{U(\psi)}$ realizes f. Thus $\delta_{\mathbf{Y}^{\mathbf{X}}}$ is surjective and co-total when considered a multifunction.

Proposition 3.12 (`fun_rep_sing` and `fun_rep_sur`)**.** *For any represented spaces $\mathbf{X}$ and $\mathbf{Y}$ the space $(\mathbf{Y}^{\mathbf{X}}, \delta_{\mathbf{Y}^{\mathbf{X}}})$ of continuous functions as defined above is a represented space.*

The proof of singlevaluedness assumes proof irrelevance and functional extensionality. Here proof irrelevance is another commonly assumed axiom that asserts a Coq property (`Prop`) and that over the internal logic of Coq is implied by the law of excluded middle.

The following theorem uses the finite products as they were constructed in Section 3.1 and can be proven from the continuity properties of the universal from Theorem 2.7.

Theorem 3.13 (`eval_cont`)**.** *Evaluation as operation $\mathbf{Y}^{\mathbf{X}} \times \mathbf{X} \to \mathbf{Y}$ is computable.*

The function space construction overlaps in its scope with the space of infinite sequences: For a countable, non-empty index set I and a represented space $\mathbf{X}$, the set underlying the space $\prod_I \mathbf{X}$ is the set of functions from I to $\mathbf{X}$. Recall what it meant for the discrete space $\mathbf{I}$ generated from I as in Section 3.1 to be discrete: Proposition 3.6 says that any function starting from $\mathbf{I}$ is continuous. As a consequence, the sets underlying $\prod_I \mathbf{X}$ and $\mathbf{X}^{\mathbf{I}}$ are identical. Indeed these spaces are computably isomorphic.

Theorem 3.14 (`sig_iso_fun`)**.** *For any represented space $\mathbf{X}$ and any countable, non-empty set I the space $\prod_I \mathbf{X}$ from the last section is computably isomorphic to the function space $\mathbf{X}^{\mathbf{I}}$, where $\mathbf{I}$ the discrete space constructed from I as described in Section 3.1.*

sketch. Recall that $A_{\mathbf{X}^{\mathbf{I}}} = \operatorname{seq} Q_{\mathbf{I}} + A_{\mathbf{X}}$ and that we decided to refer to the left inclusion into this sum by ? and to the right inclusion by !. A realizer $T \colon \mathcal{B}_{\prod_I \mathbf{X}} \to \mathcal{B}_{\mathbf{X}^{\mathbf{I}}}$ that translates a name of a sequence to a name of the corresponding function can be directly specified via

$$T(\varphi)(\mathbf{a}, q') := \begin{cases} ?(\star) & \text{if } \mathbf{a} = \epsilon \\ !\varphi(a_1, q') & \text{, where } \mathbf{a} = (a_1, \dots, a_{|\mathbf{a}|}). \end{cases}$$

To see that this realizer is correct we need to prove that whenever φ is a $\prod_I \mathbf{X}$-name of some (x_i), then $F_{U(T(\varphi))}$ is a realizer of the function $i \mapsto x_i \colon \mathbf{I} \to \mathbf{X}$. The former means that for any question q about x_i, $\varphi(i,q)$ is a valid answer. To check that $F_{U(T(\varphi))}$ is a realizer note that $\star \mapsto i$ is the only valid name of i in $\mathbf{I}$ and that for the correctness of T it is thus enough to argue that $F_{U(T(\varphi))}(\star \mapsto i)$ on any input q returns $\varphi(i,q)$. Now the values of $F_{U(T(\varphi))}$ are obtained by evaluating the universal. If the evaluation of $U(T(\varphi))$ is started on functional input $\star \mapsto i$ and input q it will first call $T(\varphi)$ on input (ϵ, q) and be returned $?(\star)$. This will lead U to evaluate $\star \mapsto i$ in $\star$ and call $T(\varphi)$ on the updated input $((i), q)$. As $T(\varphi)$ returns $!\varphi(i,q)$ on this input U will finish its run by returning $\varphi(i,q)$ and we have proven correctness of the translation.

The translation in the other direction, i.e. constructing a sequence from a continuous function proceeds by using a variation of the realizer of evaluation. □

Note that the first translation can be defined by very elementary means but is specific to the details of the universal U. The translation in the other direction is independent of the implementation of the universal but only relies on availability of an algorithm for evaluation. The algorithm for evaluation, however, executes the universal which requires an unbounded search and is considerably less elementary. More specifically, the first translation can be defined as a function that is interpreted directly or as its associated multifunction, while the second translation requires the use of the more complicated operator assignment discussed in detail in Section 2.1. Indeed, the second translation need in general not have a total continuous realizer and thus one should not expect it to be possible to give a definition of a translation as elementary as the one for the first direction.

The above theorem is a good example of a result that is usually stated as an equivalence but that was formulated as an isomorphism in our formal development. The sets underlying $\mathbf{X}^\omega$ and $\mathbf{X}^\mathbb{N}$ are considered equal from a mathematical point of view, but formally the elements of the latter are pairs of a function and a proof that this function is continuous. The set of such pairs is still in bijection with its first components because we work in a setting where different proofs of the same property are considered equal. More specifically the equality of these proofs is what the axiom of proof irrelevance states and over the logic of Coq proof irrelevance is implied by the law of excluded middle that we regularly assume anyway. The bijection can thus be given as the function that just drops the proof in the second component of the pair and for the inverse one lifts an arbitrary function by adding the proof obtained from the fact that $\mathbb{N}$ is discrete.

Theorem 3.14 above is an important building block for obtaining the results about hyper-spaces that are presented Section 4.3. As a more basic example of an application of this theorem let us give some additional justification for our previous choice to use the identity function as canonical representation of a given naming space. More specifically let us prove that the structure thus obtained is the same as considering a naming space as space of functions from its question set as discrete space to its answer set as such.

Corollary 3.15. *For any naming space $\mathcal{B} = A^Q$ it holds that $(\mathcal{B}, \mathrm{id}_\mathcal{B}) \simeq \mathbf{A}^\mathbf{Q}$.*

Proof. By the previous theorem $\mathbf{A}^\mathbf{Q} \simeq \prod_Q \mathbf{A}$. The equivalence of the representation of $\prod_Q \mathbf{A}$ with the identity representation is straightforward to define directly. To be exact, $T(\varphi)(q,\star) := \varphi(q)$ continuously translates from the identity representation to that of $\prod_Q \mathbf{A}$ and $T'(\varphi')(q) := \varphi'(q,\star)$ continuously translates back. □

4. Application to continuity in metric and hyper spaces

A map $d\colon M \times M \to \mathbb{R}$ is called a **pseudo-metric** on a set M if it returns non-negative values, is symmetric under exchange of its arguments and fulfills $d(x,x) = 0$ and the triangle inequality

$$d(x,z) \leq d(x,y) + d(y,z).$$

It is called a **metric** if it is additionally true that $d(x,y) = 0$ implies $x = y$. A pair (M,d) is called a **pseudo-metric space** if d is a pseudo-metric on M and a **metric space** if d is a metric. Every pseudo-metric space comes with a topology that is generated by the open balls with respect to the pseudo-metric and therefore with notions of continuity of functions on and convergence of sequences in this space. The notion of convergence is of particular importance since any pseudo-metric space is first-countable and thus knowing the limits of sequences is sufficient for characterizing continuity. The notion of a metric captures the properties that one expects a notion of distance to have in a very general sense. Metric spaces are a widely applicable tool for talking about continuity on many spaces of practical interest and a common sight in many branches of mathematics. As such, metric spaces have received considerable attention in their formal treatment.

A definition of continuity that does not require any knowledge of topology can be given using the ε-δ-criterion. A function $f\colon M \to M'$ between pseudo-metric spaces (M, d_M) and $(M', d_{M'})$ is called **continuous in** $x \in M$ if

$$\forall \varepsilon > 0, \exists \delta > 0, \forall y\colon d_M(x,y) \leq \delta \implies d_{M'}(f(x), f(y)) \leq \varepsilon.$$

Here, ε and δ are real numbers but replacing them by rational numbers results in an equivalent definition. The function is called **continuous** if it is continuous in every point of M. An element x of a pseudo-metric or metric space (M,d) is said to be the **limit** of a sequence (x_n) in M, in symbols $\lim_{(M,d)}(x_n) = x$, if

$$\forall \varepsilon > 0, \exists N, \forall n \geq N\colon d(x, x_n) \leq \varepsilon.$$

Again, ε is taken to be a real number but may equivalently be restricted to be rational. A function f between pseudo-metric spaces is said to be **sequentially continuous in** x if for each sequence x_n in M with $\lim_{(M,d_M)}(x_n) = x$ it holds that $\lim_{(M',d_{M'})}(f(x_n)) = f(x)$ and **sequentially continuous** if it is sequentially continuous in every point of M.

Note that above for metric spaces we reuse the terms that we already used for represented spaces: We give an alternative definition of what it means to be continuous, a limit and sequentially continuous. Since we make a clear distinction of whether the spaces we operate on are metric spaces or represented spaces, this overloading rarely leads to confusion. Note that while the names of the concepts are identical, there are significant differences for their behavior in a metric and a represented space context respectively: A function between metric spaces is continuous if and only if it is sequentially continuous while for represented spaces the backward implication can fail.

4.1. **Recovering continuity on Baire space from a metric structure.** Let $\mathcal{B}$ be a naming space, i.e. $\mathcal{B} = A^Q$ for some countable, non-empty sets Q and A. Since Q is countable and non-empty there exists a surjective function from $\mathbb{N}$ to Q. Such a function may be understood as an enumeration $\mathbf{q} = (q_n)_{n\in\mathbb{N}}$ of Q. For each such enumeration $\mathbf{q}$ define a mapping $d_{\mathbf{q}}\colon \mathcal{B} \times \mathcal{B} \to \mathbb{R}$ by

$$d_{\mathbf{q}}(\varphi, \psi) := \begin{cases} 0 & \text{if } \varphi = \psi \\ 2^{-\min\{n \in \mathbb{N} \mid \varphi(q_n) \neq \psi(q_n)\}} & \text{otherwise.} \end{cases}$$

To argue that this definition is well-formed note that the second case is only assumed if it is not true that $\varphi = \psi$ but in this case there exists some q such that $\varphi(q) \neq \psi(q)$ and since $\mathbf{q}$ is an enumeration of Q there exists some n such that $q = q_n$ and thus $\varphi(q_n) \neq \psi(q_n)$. Thus, the set that the minimum is taken over is not empty and the minimum takes a natural number value.

Proposition 4.1 (`dst_pos`, `dst_sym`, `dstxx`, `dst_trngl`, `dst_eq`)**.** *$(\mathcal{B}, d_{\mathbf{q}})$ is a metric space.*

Let us briefly discuss some details specific to our Coq proof of this proposition. The discussion before the proposition implies that the definition of $d_{\mathbf{q}}$ branches over the undecidable proposition whether two elements of a naming space are equal and thus a priori only specifies a relation and not a function. Now, the axiomatization of the real numbers implies a choice principle that is strong enough to move from this relation to an actual function [BLM15]. To be able to prove that the thus defined function is a metric it is additionally necessary to assume functional extensionality. Another tool that is used in the proof to deal with the occurring minima somewhat efficiently is a function named `search`. There are several adaptations of such a function in other developments some of which could have been reused by assumption of rather mild additional axioms. The `search` function made an earlier appearance in this paper: It is what is behind the implementation of the search operator from Example 2.3.

Recall that in Section 2.2 we introduced for any naming space $\mathcal{B}$ the limit operator $\lim_{\mathcal{B}}$ corresponding to pointwise convergence. We can now compare this limit operator to the one that corresponds to the metric notion of convergence and was introduced as $\lim_{(M,d)}$ in the introduction of this section.

Theorem 4.2 (`lim_lim`)**.** *Let $\mathcal{B}$ be a naming space and $\mathbf{q}$ an enumeration of its question set. Convergence with respect to $d_{\mathbf{q}}$ is exactly pointwise convergence. I.e.* $\lim_{(\mathcal{B},d_{\mathbf{q}})} = \lim_{\mathcal{B}}$.

As sequential continuity is defined directly from the notions of convergence, the above theorem implies that the notion of sequential continuity for naming spaces as introduced in Section 2.2 is also reproduced.

Corollary 4.3. *A function between naming spaces is sequentially continuous if and only if it is sequentially continuous as function between the corresponding metric spaces.*

It is possible to generalize the above corollary to apply not only to total but also to partial operators. We omit the generalized statement here and instead formulate a theorem about the non-sequential version of continuity.

Theorem 4.4 (`cont_cont`)**.** *Let $\mathcal{B}$ and $\mathcal{B}'$ be naming spaces and $\mathbf{q}$ and $\mathbf{q}'$ enumerations of their respective question sets. A partial operator $F\colon \subseteq \mathcal{B} \to \mathcal{B}'$ is continuous in the sense of Section 2.2 if and only if it is continuous as a function from* $(\mathrm{dom}(F), d_{\mathbf{q}})$ *to* $(\mathcal{B}', d_{\mathbf{q}'})$.

We omit details of the straightforward proofs of Theorem 4.2 and Theorem 4.4.

Example 4.5 (`examples/continuous_search.v`)**.** The traditional notion of continuity of partial operators on Baire space $\mathbb{N}^{\mathbb{N}}$ is captured by the continuity introduced in Section 2.2 if all of the questions and answer sets are taken to be the natural numbers.

$$\begin{tikzcd}
\text{metric spaces} & \text{represented spaces} \\
\text{sequentially continuous} \arrow[r, Rightarrow, shift left] \arrow[d, Rightarrow, shift left] & \text{sequentially continuous} \arrow[l, Rightarrow, shift left] \\
\varepsilon\text{-}\delta\text{ continuous} \arrow[u, Rightarrow, shift left] \arrow[r, Rightarrow, shift left] & \text{continuously realizable} \arrow[l, Rightarrow, shift left] \arrow[u, Rightarrow, "\text{constructive}"']
\end{tikzcd}$$

Figure 6. Implications between different notions of continuity on metric spaces.

4.2. **Comparing continuity in represented and in metric spaces.** A metric space is called **separable** if there exists a dense sequence $(r_n)_{n\in\mathbb{N}}$ in it. Here, density of a sequence can equivalently be taken to mean that any open ball in the metric space contains an element of the sequence or that for any point of the metric space there exists a subsequence of (r_n) that converges to that point. Separable metric spaces are well investigated in computable analysis [Wei93]. To each dense sequence $\mathbf{r} = (r_n)_{n\in\mathbb{N}}$ in a metric space (M, d) assign a multifunction $\delta_{\mathbf{M_r}} \colon \mathbb{N}^\mathbb{N} \rightrightarrows M$ defined by

$$x \in \delta_{\mathbf{M_r}}(\varphi) \quad\iff\quad \forall n, d(x, r_{\varphi(n)}) \le 2^{-n}.$$

Clearly $\mathbb{N}^\mathbb{N}$ is a naming space and $\delta_{\mathbf{M_r}}$ can be proven singlevalued and surjective (`mrep_sing` and `mrep_sur`). Thus $\delta_{\mathbf{M_r}}$ defines a representation and we denote the corresponding represented space by $\mathbf{M_r} := (M, \delta_{\mathbf{M_r}})$.

Note that the construction of $\mathbf{M_r}$ is very similar to how we chose to represent the real numbers through rational approximations in Example 3.1. A name of an element of a metric space produces an index of an approximation from an accuracy requirement. The only difference is that for real numbers we decided to minimize the number of types involved by using rational numbers as both accuracy requirements and approximations while for metric spaces we use accuracy requirements of the form 2^{-n} to minimize the number of distinct types by matching the type of indices and setting both question and answer types to be $\mathbb{N}$.

A sequence (x_n) in a separable metric space (M, d) converges to a limit x from M if and only if the same is true in the represented space $\mathbf{M_r}$ or in symbols if and only if it holds that $\lim_{\mathbf{M_r}}(x_n) = x$. This remains true irrespectively of the choice of the dense sequence that is used for constructing the representation and can more concisely be formulated as $\lim_{(M,d)} = \lim_{\mathbf{M_r}}$. Furthermore, if (M', d') is another separable metric space, then a function $f\colon M \to M'$ is continuous as a function between metric spaces if and only if it is continuously realizable as a function $f\colon \mathbf{M_r} \to \mathbf{M}'_{\mathbf{r}'}$ and this remains true irrespectively of the choices of the dense sequences $\mathbf{r}$ and $\mathbf{r}'$.

This section describes our formal proofs of statements comparing the two continuity notions and their sequential versions (cf. Figure 6). The proofs have been kept as constructive as possible. Since the definition of a metric space relies on the axiomatic reals, only one of the implications is fully constructive, the others are conservative over the background theory of real numbers and do not rely on the axioms of the real numbers in an essential way.

Let us reiterate that the construction of a represented space from a separable metric space explicitly relies on the choice of a dense sequence. I.e. for each dense sequence $\mathbf{r}$ in the metric space one obtains a represented space $\mathbf{M}_{\mathbf{r}}$. In the following we usually drop the index $\mathbf{r}$ for brevity. This is justified in a continuity setting as different choices of dense sequences lead to isomorphic represented spaces. As always, the situation is more complicated if computability is considered and in this case one should assume in the following that two metric spaces with dense sequences are fixed. In the formal development, the dense sequences are always present as parameters.

Let us first describe the proof of the equivalence of the notions of sequential continuity. The main part of the proof is that the notions of limit in the metric space and the corresponding represented space coincide.

Theorem 4.6 (`lim_mlim`)**.** *Whenever (M, d) is a separable metric space and $\mathbf{M}$ as above then*

$$\lim_{(M,d)} = \lim_{\mathbf{M}} .$$

idea. The proof that convergence in the represented space implies convergence in the metric space is straightforward. The idea behind the other direction can be sketched as follows: If (x_n) converges to x in the metric space then there exists a modulus of convergence, i.e. some $\mu\colon \mathbb{N} \to \mathbb{N}$ such that

$$\forall n, m \geq \mu(n)\colon d(x_m, x) \leq 2^{-n}.$$

From an arbitrary sequence (φ'_m) of names of x_m and a name φ' of x an appropriate convergent sequence of names can be defined by

$$\varphi_m(n) := \begin{cases} \varphi'(n+1), & \text{if } \mu(n+1) \leq m \\ \varphi'_m(n), & \text{otherwise} \end{cases}$$

and its limit is given by $\varphi(n) := \varphi'(n+1)$ which is clearly a name of x again. □

The availability of a modulus of convergence as a function relies on a use of the axiom of countable choice. This could probably be eliminated by appropriate assumptions about the values of the metric being approximable on the elements of the dense sequence (i.e. by working with computable metric spaces).

That the sequential notions of continuity on metric and represented spaces coincide follows immediately from the above Theorem 4.6. As the proof of each direction requires to translate limits in both directions, either of the directions is as constructive or non-constructive as the worse direction of the previous theorem.

Corollary 4.7 (`scnt_mscnt`)**.** *Let (M, d) and (M', d') be separable metric spaces and $\mathbf{M}$ and $\mathbf{M}'$ the derived represented spaces. Then $f : M \to M'$, is sequentially continuous as a function between metric spaces if and only if it is sequentially continuous as function $f\colon \mathbf{M} \to \mathbf{M}'$.*

Next let us state the equivalence of continuous realizability and ε-δ-continuity. One implication, namely proving continuous realizability from ε-δ-continuity needs stronger assumptions and for the INCONE library we thus separated the proofs.

Lemma 4.8 (`cont_mcont` and `mcont_cont`)**.** *Let (M, d) and (M', d') be two separable metric spaces. A function $f\colon M \to M'$ is ε-δ-continuous if and only if $f\colon \mathbf{M} \to \mathbf{M}'$ is continuous.*

While the proof that continuous realizability implies ε-δ-continuity is straightforward, the proof of the other implication required some work and we sketch some of the details.

Interestingly, the tools needed for this proof are in spirit fairly close to those that were used to prove the existence of associates in Section 2.3, more specifically we also use minimal moduli. Call a function $m\colon \mathbb{N} \to \mathbb{N}$ a **metric modulus of continuity** of f in $x \in M$ if

$$\forall y \in M\colon d(x,y) \leq 2^{-m(n)} \implies d'(f(x), f(y)) \leq 2^{-n}$$

and call such a modulus **minimal** if it is minimal in the obvious way. This notion generalizes the one used to prove the existence of associates in Section 2.3 if the naming spaces are equipped with the metric space structures that were introduced in Section 4.1 and reasonable assumptions about the enumeration used for this are made.

Lemma 4.9 (`exists_minmod_met`). *For any continuous function f between metric spaces and any argument x for f there exists a minimal modulus of f in x.*

This lemma implies its version for naming spaces that we have earlier proven in Section 2.3. As we noted in the discussion following Lemma 2.10 this weaker version cannot be proven constructively. It follows that the same remains true for the lemma above.

The key idea behind the rest of the proof is to also use some notion of being self-modulating. This is slightly complicated by the fact the notion that made sense for moduli of continuity on naming spaces does not translate to the metric setting. Moreover, in general the minimal modulus of continuity fails to have some of the nice properties that we relied on for naming spaces. The function assigning to each x the minimal modulus function of f in x is usually not even continuous. This is, for instance, not possible if (M, d) is connected as the function takes values in a totally disconnected space and functions from connected to totally disconnected spaces can only be continuous if they are constant. One might suspect that the awkward typing is the cause for this, and that quantifying the continuity of f in x by use of a function of type $\mathbb{R} \to \mathbb{R}$ instead would help, but it does not. It is known that also in this case returning the minimal modulus of continuity need not be a continuous operation. A construction of a continuous such assignment, while possible in general, takes considerably more effort [Gut83, Ena00]. Thus, for our proof that ε-δ-continuity implies continuous realizability we rely on a notion of being almost self-modulating instead: The value of the minimal modulus on slightly disturbed input from the metric space can be bounded in terms of a shift of the minimal modulus in the original value.

4.3. Sierpiński space and closed choice on the naturals.

As a somewhat orthogonal class of examples of represented spaces that are far from being metrizable let us consider certain hyper spaces. A hyper space is a space whose underlying set consists of subsets of a given represented space and depending on the application very different ways to represent such sets of subsets can be appropriate. An important tool for introducing representations of hyper spaces is Sierpinski space $\mathbb{S}$. The base set of $\mathbb{S}$ is the two point set $\{\top, \bot\}$ equipped with the total representation $\delta_{\mathbb{S}}$ with names of type $\mathbb{N} \to \mathbb{B}$ specified by

$$\delta_{\mathbb{S}}(\varphi) = \top \quad \iff \quad \exists n \in \mathbb{N}\ \varphi(n) = \text{true}.$$

The elements of Sierpinski space should be understood as symbols for termination and divergence. More specifically: From any kind of computational process for which the notion of an elementary computational step is meaningful one may obtain a sequence of Booleans by taking the n-th element to be the truth-value of the statement "the process finishes within the first n basic steps". This way one obtains a name of $\top$ if the process terminates and $\bot$ otherwise.

The final topology of this representation is $\{\emptyset, \{\top\}, \mathbb{S}\}$ and as a non-Hausdorff space this space is not metrizable. Without going into details let us mention that also for Sierpinski space there is a tight connection between continuous realizability and topological continuity. To connect Sierpinski space to hyperspaces consider for a subset $U \subseteq \mathbf{X}$ its characteristic function $\chi_U\colon \mathbf{X} \to \mathbb{S}$ defined by

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

For illustration let us assume that we can also safely equip $\mathbf{X}$ with the final topology of the representation without changing the notion of continuity. Recall that a function is topologically continuous if and only if preimages of open sets are open. From the topology of Sierpinski space specified above it is evident that in this setting a subset of $\mathbf{X}$ is open if and only if its characteristic function is continuous. Inspired by this we may generalize and follow sources such as [Pau16] in identifying the space $\mathcal{O}(\mathbf{X})$ of open subsets of $\mathbf{X}$ with the space of continuous functions $\mathbb{S}^{\mathbf{X}}$ as defined in Section 3.3. Similarly, the space $\mathcal{A}(\mathbf{X})$ of closed subsets of $\mathbf{X}$ can be introduced as the space of complements of opens.

If $\mathbf{X}$ is taken to be a concrete space it is often possible to give simpler descriptions of $\mathcal{O}(\mathbf{X})$ and $\mathcal{A}(\mathbf{X})$. For instance if $\mathbf{X} := \mathbb{N}$ is the discrete represented space of natural numbers one may make use of a space of infinite sequences as introduced in Section 3.2 and in particular of the special case $I = \mathbb{N}$ and $\mathbf{X} = \mathbb{S}$ of Lemma 3.14 which guarantees that $\mathcal{O}(\mathbb{N}) = \mathbb{S}^{\mathbb{N}} \simeq \prod_{\mathbb{N}} \mathbb{S} = \mathbb{S}^{\omega}$. There exists a fully concrete description of $\mathcal{O}(\mathbb{N})$ that is often used for reasoning about this space in computable analysis: The enumeration representation, where a name of an open set enumerates its elements. We call the corresponding space $\mathcal{O}_{\mathbb{N}}$. The representation of the corresponding concrete space $\mathcal{A}_{\mathbb{N}}$ of closed subsets encodes sets as functions $\varphi\colon \mathbb{N} \to \mathbb{N}$ via

$$\delta_{\mathcal{A}_{\mathbb{N}}}(\varphi) := \mathbb{N} \setminus \{n \in \mathbb{N} \mid \exists m\colon \varphi(m) = n+1\}.$$

That is: the information that a name φ specifies about a closed set is an enumeration of its complement.

We provide a formal proof that the enumeration representations of the open and closed subsets of the natural numbers capture the abstract structure of these spaces through the exponential in the category of represented spaces.

Theorem 4.10 (`AN_iso_Anat`, `ON_iso_Onat` and `clsd_iso_open`)**.** $\mathcal{A}(\mathbb{N}) \simeq \mathcal{O}(\mathbb{N})$ *and the concrete spaces of open and closed sets above are correct, i.e.* $\mathcal{A}(\mathbb{N}) \simeq \mathcal{A}_{\mathbb{N}}$ *and* $\mathcal{O}(\mathbb{N}) \simeq \mathcal{O}_{\mathbb{N}}$*.*

The first of these isomorphies may look surprising but is evident on closer inspection: the isomorphism is taking the complement and it is realized by the identity function. The isomorphism of $\mathcal{O}(\mathbb{N})$ and $\mathcal{O}_{\mathbb{N}}$ is proven by first replacing $\mathcal{O}(\mathbb{N})$ by $\mathbb{S}^{\omega}$ as described above. The realizers for the isomorphisms between $\mathbb{S}^{\omega}$ and $\mathcal{O}_{\mathbb{N}}$ can be defined as functions directly by relying on the Cantor paring function or any standard pairing function. We chose to use the standard pairing function provided by the mathematical components library.

As an application let us consider choice operators. To implement a choice operator one has to select an element of a closed set. As closed sets are represented as complements of open sets and thus by providing positive information about their complement, a realistic implementation of a choice operator is rarely possible. Such an implementation still solves a problem that can be described by fairly basic means and thus asserting the existence of a solution is popular for classifying unsolvability of other tasks according to their logical strength. More details about this can be found in survey articles about Weihrauch reducibility such as [BGP17], more concrete examples of classifications where the problems discussed here make an appearance can be found for instance in [PS18].

Let us give a precise formulation of what it means to select an element from a closed subset. For a fixed represented space $\mathbf{X}$ let closed choice on $\mathbf{X}$ be the task of finding a realizer of the multifunction $C_{\mathbf{X}}$ defined by

$$C_{\mathbf{X}}\colon \mathcal{A}(\mathbf{X}) \rightrightarrows \mathbf{X}, \quad a \in C_{\mathbf{X}}(A) \iff a \in A.$$

Or in words: a is an acceptable return value of $C_{\mathbf{X}}$ on input A if and only if a is an element of A. This in particular means that the domain of $C_{\mathbf{X}}$ are the non-empty subsets of $\mathbf{X}$. This is relevant as it means that a potential realizer of $C_{\mathbf{X}}$ can behave arbitrarily when handed a name of the empty set. The realizer need not produce any name of an element of $\mathbf{X}$ in this case and may even be undefined.

Consider the special case where $\mathbf{X} = \mathbb{N}$. While the argument space of the multivalued function $C_{\mathbb{N}}$ is $\mathcal{A}(\mathbb{N})$ we may use the same definition to obtain a multifunction $C'_{\mathbb{N}}\colon \mathcal{A}_{\mathbb{N}} \rightrightarrows \mathbb{N}$. A mathematician may even consider it pointless to give this function a new name as $\mathcal{A}(\mathbb{N}) \simeq \mathcal{A}_{\mathbb{N}}$ and isomorphic spaces are regularly identified. Indeed, for the question of whether $C_{\mathbb{N}}$ has a continuous realizer the space $\mathcal{A}(\mathbb{N})$ may be substituted with $\mathcal{A}_{\mathbb{N}}$ for this exact reason.

Lemma 4.11 (`CN_CN'_hcr`)**.** *$C_{\mathbb{N}}$ has a continuous realizer if and only if $C'_{\mathbb{N}}$ does.*

On the concrete space $A_{\mathbb{N}}$ a standard argument can be used to see that a continuous realizer cannot exist.

Theorem 4.12 (`CN'_not_cont`)**.** *$C'_{\mathbb{N}}$ does not have a continuous realizer.*

outline. The proof proceeds by contradiction. Assume that to the contrary $C'_{\mathbb{N}}$ is continuous and that F is a continuous realizer. Pick any name φ of the one point set $\{0\}$. As F is a realizer, it has to return a name of 0 on input φ, i.e. $F(\varphi)(\star) = 0$. Since F is continuous there is a list $\mathbf{q} \subseteq \mathbb{N}$ such that $F(\varphi)(\star) = F(\psi)(\star)$ for all $\psi\colon \mathbb{N} \to \mathbb{N}$ that coincide with φ on $\mathbf{q}$. Consider the name φ' of the non-empty set $A := \mathbb{N} \setminus (\{n \mid \exists m \in \mathbf{q}, \varphi(m) = n+1\} \cup \{0\})$ defined by

$$\varphi'(n) := \begin{cases} \varphi(n), & \text{if } n \in \mathbf{q} \\ 1, & \text{otherwise.} \end{cases}$$

On one hand, $F(\varphi')(\star) \in A$ since F is a realizer. On the other hand $F(\varphi')(\star) = F(\varphi)(\star) = 0$ as φ and φ' coincide on $\mathbf{q}$. Now since $0 \notin A$ we arrive at a contradiction and this completes the proof. □

Combining the theorem with the lemma we conclude that closed choice on the natural numbers does not have a continuous realizer:

Corollary 4.13 (`CN_not_cont`)**.** *Closed choice on the natural numbers does not have a continuous solution.*

Clearly the existence of a continuous solution also rules out the possibility that a computable solution exists.

5. Conclusion

The Incone library provides general tools for enriching abstract mathematical structures of interest with computational structures and for comparing different such encodings. It can be used to investigate reasonability of encodings in a fully formal setting, to provide computational content for mathematical statements or to prove it impossible to do so. We feel that the examples from this paper showcase these capabilities well. Moreover, they involve many of the most prominent features of Incone: The abstract definition of the space of open subsets is based on the library's function space construction and our proof of isomorphy relies on the fact that in this specific case the corresponding space of functions is isomorphic to a space of sequences. We believe the Incone library to be reasonably accessible for the computable analysis community and hope that its combination with methods from that community [MPPZ16] could help to make parts of it more accessible to the numerical analysis community.

Let us make an effort to outline some relations to other approaches. For a given representation, i.e. singlevalued and co-total (surjective) multifunction $\delta\colon \mathcal{B} \rightrightarrows X$, one may define a partial equivalence relation on Baire space via

$$\varphi \sim \psi \iff \exists x \in X\colon x \in \delta(\varphi) \wedge x \in \delta(\psi).$$

If δ is considered a partial function the right-hand side can be replaced by $\delta(\varphi) = \delta(\psi)$ if the equality is interpreted in the appropriate way. The resulting structure $(\mathcal{B}, \sim)$ is what is called a setoid in constructive mathematics. Conversely, for each such setoid one may take X to be the set of equivalence classes with respect to $\sim$. This makes the quotient mapping a representation and one may check that the constructions are "inverse" in an appropriate sense. Other notions for representations can be translated accordingly. For instance, a function that preserves equivalence induces a function on the equivalence classes and realizes this function with respect to the quotient mappings as representations. If the function does not preserve the equivalence relation, it induces a multifunction on the equivalence classes that it still realizes.

While mathematicians take quotients without thinking twice, in a type theoretic setting quotients are a problematic operation. Our proof assistant of choice, Coq, does not support quotient types and there are reasons why it refrains from doing so. As setoids allow to still reason about the quotient without really taking it, they are a very popular tool in the Coq community and constructive mathematics in general. The specification of an appropriate type X and a representation that acts as quotient mapping is additional information that has to be provided by the user. This extra information is exclusively used for specification purposes and we aim to give the greatest possible amount of freedom in how the user wants to present this information. For instance our choice to treat representations as relations and not as functions can be justified by this. This comes with the drawback that definability of a function on the description of the quotient need no longer be related to it being realistically implementable. For instance consider Sierpinski space: In this case the description of the quotient is a discrete two element set which means that all self-maps are definable. However, the one that switches $\top$ and $\bot$ is not implementable on the level of names. As the information from the description of the quotient is computationally irrelevant, another way of introducing it is via an axiomatization similar to that of the real numbers from Coq's standard library that was used in this paper. Ideally such an axiomatization should avoid stating unrealistic assumptions as computationally relevant facts but the example of the reals shows that even if it does, this can be worked around.

The idea behind the constructions provided by Incone is to support a user in conveniently obtaining descriptions of quotient types by piecing them together from basic building blocks. As basic spaces Incone defines discrete spaces, separable metric spaces with a distinguished dense sequence and Sierpinski space. One may combine these basic spaces via building products, sums, infinite sequences and spaces of functions. The space $\mathcal{O}(\mathbb{N})$ is an example for such a combination as it is constructed as a space of functions between elementary defined spaces. This particular space, sometimes also called $\mathcal{P}(\omega)$, is a common sight in work on constructive or computational analysis. This is even though as a space it is very different from the objects that are actually of interest in these fields. However, many spaces of practical importance can be identified with a subspace of this space: Any quasi-Polish space in the sense of de Brecht and therefore in particular any Polish space embeds isomorphically into $\mathcal{O}(\mathbb{N})$ as a subspace of low descriptive complexity [dB13]. This covers all of the spaces considered in Section 4. Let us give a concrete example by outlining of how to isomorphically embed the real numbers into $\mathcal{O}(\mathbb{N}) \simeq \mathbb{S}^{\mathbb{N}}$. First enumerate the open rational intervals and then send a real number x to the function that on input of a natural number n returns $\top$ if x is contained in the n-th interval and $\bot$ if this is not the case. As checking inclusion of a real number into an open interval is semi-decidable relative to availability of approximations to the real number, we obtain an element of $\mathcal{O}(\mathbb{N})$ that continuously depends on the real number. This can be checked to lead to a continuous, and even computable, embedding of the reals into $\mathcal{O}(\mathbb{N})$. Of course, this is nothing but the identification of a real number with a Dedekind cut.

For future directions in the development of Incone we plan to work on additional applications. One particularly fitting extension of the contents of this paper would be a proof that $C([0,1]) \simeq \mathbb{R}^{[0,1]}$. This statement is called the Computable Weierstraß Theorem [PEC75]: Here, $C([0,1])$ are the continuous real-valued functions on the unit interval represented as separable metric space with supremum norm and the rational polynomials as dense sequence. The function-space $\mathbb{R}^{[0,1]}$ has the same underlying set but uses a different representation that the theorem states to be computably equivalent. Other possibilities include:

- Results about ODE solving [IH12, MS13, KST18]. This may be done by providing an interface with C-CoRn, parts of it could also be done separately by relying on libraries such as Coq-Interval [Mel08].
- Duality theory for spaces of summable sequences (ℓ_p-spaces) which provide a pool of examples where sub-spaces of exponentials can be treated complexity theoretically [Sch04, SS17]. Additionally it constitutes a step towards capturing popular methods for solving partial differential equations [BY06, SS08, BCF$^+$17].
- A characterization of continuity via preimages of open sets, general considerations about admissibility, discreteness, compactness and many other similar results [Pau16, Sch02a].

There are also still some gaps in the Incone library that we plan to fill. Currently there is no complete proof that the category of represented spaces is Cartesian closed as the corresponding universal property is missing. The library proves a restricted case by providing a duality operator, but a full proof would be desirable. Further steps would be to use a formalization of a model of computation to make it possible to talk about computability without moving to the meta level and using the Prop/Type distinction. This would have the additional advantage of opening the possibility to do complexity theory. Even without reliance on a model of computation it should be possible to add the capability to do qualitative complexity theory in terms of tracking the decrease in accuracy of approximations [KC12, Fér17, NS20, KST19]. As the Incone library keeps close to recent work about complexity theory for computable analysis, this should be possible with relatively low effort. A full treatment of step-counting complexity might become available in the not too distant future due to recent progress on the formalization of models of computation [FS18, Wut18] and methods from implicit complexity theory [FHM$^+$18]. Another way to gain insight into such efficiency considerations would be to capture the trace of the basic feasible functionals on the operators on Baire space [Meh76, KC96, KS18].

References

[AB14] Jeremy Avigad and Vasco Brattka. *Computability and analysis: the legacy of Alan Turing*, pages 1–47. Lecture Notes in Logic. Cambridge University Press, 2014.

[ACK$^+$20] Reynald Affeldt, Cyril Cohen, Marie Kerjean, Assia Mahboubi, Damien Rouhling, and Kazuhiko Sakaguchi. Formalizing functional analysis structures in dependent type theory. working paper or preprint, January 2020.

[ACR18] Reynald Affeldt, Cyril Cohen, and Damien Rouhling. Formalization techniques for asymptotic reasoning in classical analysis. *Journal of Formalized Reasoning*, 11(1):43–76, 2018.

[ASBZ13] Klaus Ambos-Spies, Ulrike Brandt, and Martin Ziegler. Real benefit of promises and advice. In Paola Bonizzoni, Vasco Brattka, and Benedikt Löwe, editors, *The Nature of Computation. Logic, Algorithms, Applications*, pages 1–11, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Bau00] Andrej Bauer. *The Realizability Approach to Computable Analysis and Topology*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2000. AAI3002721.

[BB12] Errett Bishop and Douglas Bridges. *Constructive analysis*, volume 279. Springer Science & Business Media, 2012.

[BCC$^+$06] Andrea Balluchi, Alberto Casagrande, Pieter Collins, Alberto Ferrari, Tiziano Villa, and Alberto L Sangiovanni-Vincentelli. Ariadne: a framework for reachability analysis of hybrid automata. In *In: Proceedings of the International Syposium on Mathematical Theory of Networks and Systems.(2006*. Citeseer, 2006.

[BCF$^+$17] Sylvie Boldo, François Clément, Florian Faissole, Vincent Martin, and Micaela Mayero. A Coq Formal Proof of the Lax–Milgram theorem. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2017, pages 79–89, Paris, France, January 2017. ACM.

[BDBP12] Vasco Brattka, Matthew De Brecht, and Arno Pauly. Closed choice and a uniform low basis theorem. *Annals of Pure and Applied Logic*, 163(8):986–1008, 2012.

[BG11] Vasco Brattka and Guido Gherardi. Effective choice and boundedness principles in computable analysis. *Bulletin of Symbolic Logic*, 17(1):73–117, 2011.

[BGP17] Vasco Brattka, Guido Gherardi, and Arno Pauly. Weihrauch complexity in computable analysis. *arXiv preprint arXiv:1707.03202*, 2017.

[BKMP16] Vasco Brattka, Akitoshi Kawamura, Alberto Marcone, and Arno Pauly. Measuring the Complexity of Computational Content (Dagstuhl Seminar 15392). *Dagstuhl Reports*, 5(9):77–104, 2016.

[BLM15] Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Coquelicot: A user-friendly library of real analysis for Coq. *Mathematics in Computer Science*, 9(1):41–62, 2015.

[BLM16] 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.

[BM17] Sylvie Boldo and Guillaume Melquiond. *Computer Arithmetic and Formal Proofs*. ISTE Press – Elsevier, 2017.

[Bro75] L. E. J. Brouwer. *Collected works. Vol. 1.* North-Holland Publishing Co., Amsterdam-Oxford; American Elsevier Publishing Co., Inc., New York, 1975. Philosophy and foundations of mathematics, Edited by A. Heyting.

[BY06] Vasco Brattka and Atsushi Yoshikawa. Towards computability of elliptic boundary value problems in variational formulation. *Journal of Complexity*, 22(6):858 – 880, 2006. Computability and Complexity in Analysis.

[CFGW04] 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.

[dB13] Matthew de Brecht. Quasi-polish spaces. *Annals of Pure and Applied Logic*, 164(3):356 – 381, 2013.

[Dei92] K. Deimling. *Multivalued Differential Equations*. De Gruyter series in nonlinear analysis and applications. W. de Gruyter, 1992.

[EM46] Samuel Eilenberg and Deane Montgomery. Fixed point theorems for multi-valued transformations. *American Journal of Mathematics*, 68(2):214–222, 1946.

[Ena00] Ali Enayat. δ as a continuous function of x and ε. *The American Mathematical Monthly*, 107(2):151–155, 2000.

[EX16] Martín Escardó and Chuangjie Xu. A constructive manifestation of the Kleene-Kreisel continuous functionals. *Annals of Pure and Applied Logic*, 167(9):770 – 793, 2016. Fourth Workshop on Formal Topology (4WFTop).

[Fér17] Hugo Férée. Game semantics approach to higher-order complexity. *J. Comput. Syst. Sci.*, 87(C):1–15, August 2017.

[FHM$^+$18] Hugo Férée, Samuel Hym, Micaela Mayero, Jean-Yves Moyen, and David Nowak. Formal proof of polynomial-time complexity with quasi-interpretations. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 146–157. ACM, 2018.

[FS18] Yannick Forster and Gert Smolka. Call-by-Value Lambda Calculus as a Model of Computation in Coq. *Journal of Automated Reasoning*, 2018.

[GMT16] Georges Gonthier, Assia Mahboubi, and Enrico Tassi. A Small Scale Reflection Extension for the Coq system. Research Report RR-6455, Inria Saclay Ile de France, 2016.

[Grz57] A. Grzegorczyk. On the definitions of computable real continuous functions. *Fund. Math.*, 44:61–71, 1957.

[Gut83] Joe A Guthrie. A continuous modulus of continuity. *The American Mathematical Monthly*, 90(2):126–127, 1983.

[IH12] Fabian Immler and Johannes Hölzl. Numerical analysis of ordinary differential equations in Isabelle/HOL. In *ITP*, volume 7406 of *LNCS*, pages 377–392, 2012.

[KC96] Bruce M. Kapron and Stephen A. Cook. A new characterization of type-2 feasibility. *SIAM J. Comput.*, 25:117–132, 1996.

[KC12] Akitoshi Kawamura and Stephen Cook. Complexity theory for operators in analysis. *ACM Transactions in Computation Theory*, 4(2):Article 5, 2012.

[Kle59] S.C. Kleene. Countable functionals. *Constructivity in Mathematics: proceedings of the colloquium held at Amsterdam*, 1959.

[Ko91] Ker-I Ko. *Complexity theory of real functions*. Progress in Theoretical Computer Science. Birkhäuser Boston Inc., Boston, MA, 1991.

[Kon08] Michal Konecnỳ. AERN-Real: Arbitrary-precision interval arithmetic for approximating exact real numbers, 2008.

[Kre59] Georg Kreisel. Interpretation of analysis by means of constructive functionals of finite type, Constructivity in Mathematics, 1959.

[KS18] Bruce M. Kapron and Florian Steinberg. Type-two polynomial-time and restricted lookahead. In *LICS*, 2018.

[KST18] 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.

[KST19] Akitoshi Kawamura, Florian Steinberg, and Holger Thies. Second-order linear-time computability with applications in computable analysis. *15th Annual Conference on Theory and Applications of Models of Computation*, 2019. extended abstract accepted for presentation at TAMC 2019.

[KW85] Christoph Kreitz and Klaus Weihrauch. Theory of representations. *Theoretical computer science*, 38:35–53, 1985.

[Lac58] Daniel Lacombe. Sur les possibilités d'extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles. In *Le raisonnement en mathématiques et en sciences expérimentales*, Colloques Internationaux du Centre National de la Recherche Scientifique, LXX, pages 67–75. Editions du Centre National de la Recherche Scientifique, Paris, 1958.

[LN15] John Longley and Dag Normann. *Higher-order computability*, volume 100. Springer, 2015.

[Meh76] Kurt Mehlhorn. Polynomial and abstract subrecursive classes. *Journal of Computer and System Sciences*, 12(2):147 – 178, 1976.

[Mel08] Guillaume Melquiond. Proving bounds on real-valued functions with computations. In *International Joint Conference on Automated Reasoning*, pages 2–17. Springer, 2008.

[MPPZ16] Norbert Th. Müller, Sewon Park, Norbert Preining, and Martin Ziegler. On formal verification in imperative multivalued programming over continuous data types. *CoRR*, abs/1608.05787, 2016.

[MS13] Evgeny Makarov and Bas Spitters. The Picard algorithm for ordinary differential equations in Coq. In *ITP*, volume 7998 of *Lecture Notes in Computer Science*, pages 463–468. Springer, 2013.

[Mül01] Norbert Th. Müller. The iRRAM: Exact arithmetic in C++. In *Computability and complexity in analysis. 4th international workshop, CCA 2000. Swansea, GB, September 17–19, 2000. Selected papers*, pages 222–252. Berlin: Springer, 2001.

[NS20] Eike Neumann and Florian Steinberg. Parametrised second-order complexity theory with applications to the study of interval computation. *Theoret. Comput. Sci.*, 806:281–304, 2020.

[O'C05] Russell O'Connor. Essential incompleteness of arithmetic verified by Coq. In Joe Hurd and Tom Melham, editors, *Theorem Proving in Higher Order Logics*, pages 245–260, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[O'C09] Russell O'Connor. *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory*. PhD thesis, Radboud Universiteit Nijmegen, 2009.

[Pau16] Arno Pauly. On the topological aspects of the theory of represented spaces. *Computability*, 5(2):159–180, 2016.

[Pau17] Arno Pauly. Many-one reductions and the category of multivalued functions. volume 27, pages 376–404. Cambridge University Press, 2017.

[PEC75] Marian Boykan Pour-El and Jerome Caldwell. On a simple definition of computable function of a real variable-with applications to functions of a complex variable. *Mathematical Logic Quarterly*, 21(1):1–19, 1975.

[PER89] Marian B. Pour-El and J. Ian Richards. *Computability in Analysis and Physics*, volume Volume 1 of *Perspectives in Mathematical Logic*. Springer-Verlag, Berlin, 1989.

[PS18] Arno Pauly and Florian Steinberg. Comparing representations for function spaces in computable analysis. *Theory of Computing Systems*, 62(3):557–582, Apr 2018.

[PZ13] Arno Pauly and Martin Ziegler. Relative computability and uniform continuity of relations. *J. Logic & Analysis*, 5, 2013.

[Sch02a] Matthias Schröder. *Admissible Representations for Continuous Computations*. PhD thesis, FernUniversität Hagen, 2002.

[Sch02b] Matthias Schröder. Extended admissibility. *Theoretical computer science*, 284(2):519–538, 2002.

[Sch04] Matthias Schröder. Spaces allowing type-2 complexity theory revisited. *Math. Log. Q.*, 50:443–459, 09 2004.

[Sel94] Alan L. Selman. A taxonomy of complexity classes of functions. *Journal of Computer and System Sciences*, 48(2):357 – 381, 1994.

[Soa78] Robert I Soare. Recursively enumerable sets and degrees. *Bulletin of the American Mathematical Society*, 84(6):1149–1181, 1978.

[SS08] Svetlana Selivanova and Victor Selivanov. Computing solutions of symmetric hyperbolic systems of pde's. *Electron. Notes Theor. Comput. Sci.*, 221:243–255, December 2008.

[SS17] Matthias Schröder and Florian Steinberg. Bounded time computation on metric spaces and Banach spaces. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017.

[Ste19a] Florian Steinberg. The Incone library. `https://github.com/FlorianSteinberg/incone`, 2019. release v1.0.

[Ste19b] Florian Steinberg. The Metric library. `https://github.com/FlorianSteinberg/metric`, 2019. release v1.0.

[Ste19c] Florian Steinberg. The Mf library. `https://github.com/FlorianSteinberg/mf`, 2019. release v1.0.

[Ste19d] Florian Steinberg. The Rlzrs library. `https://github.com/FlorianSteinberg/rlzrs`, 2019. release v1.0.

[Tur36] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(1):230–265, 1936.

[Tur38] Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem. a correction. *Proceedings of the London Mathematical Society*, 2(1):544–546, 1938.

[TvD88] A. S. Troelstra and D. van Dalen. *Constructivism in mathematics. Vol. II*, volume 123 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Co., Amsterdam, 1988.

[vO11] Jaap van Oosten. Partial combinatory algebras of functions. *Notre Dame J. Formal Logic*, 52(4):431–448, 07 2011.

[Wei93] Klaus Weihrauch. Computability on computable metric spaces. *Theoretical Computer Science*, 113(2):191–210, 1993.

[Wei00] Klaus Weihrauch. *Computable Analysis*. Springer, Berlin/Heidelberg, 2000.

[Wut18] Maximilian Wuttke. Verified programming of Turing Machines in Coq. Master's thesis, Saarland University, 2018.

Appendix A. An overview of the Incone library

The goal of the Incone library is to provide a Coq formalization of some of the most important concepts and basic results from computable analysis. The library uses notations that allow to state theorems similar to how they would be formulated in natural language and therefore should be readable for a mathematician familiar with computable analysis without having a deep knowledge of Coq. For example, Lemma 2.2 corresponds to the following statement in the library.

```
Lemma tight_comp R S T (f f': T ->> R) (g g': S ->> T):
f \tightens f' -> g \tightens g' -> (f \o g) \tightens (f' \o g').
```

Here, `tight_comp` is the name of the lemma and the symbols following this name until the colon at the end of the line are parameters of its statement. These parameters can alternatively be thought of as universally quantified. The phrase `(f f': T ->> R)` thus translates to the mathematical language used in the paper as "for all $f, f' \colon T \rightrightarrows R$", or "let f and f' be multifunctions from T to R". That R and T are sets can be omitted as it can be inferred from their use. The notation for multifunctions may seem unfortunate at first but the closer match of a notation `=>` seemed unwise to use as it is commonly overloaded with a number of different meanings. The colon is followed by the body of the lemma, where `->` is the logical implication and `_ \o _` and `_ \tightens _` are the notations for composition and the tightening order on multivalued functions. Many of the infix notations in the library start in a `\` to avoid blocking too many keywords.

Most statements in this paper and many additional results can be found in the files of the Incone library. In the paper, we labeled each theorem and lemma with the names of the relevant statements in the library. This should make it easy to locate and inspect them in Incone. The structure of this paper and the formal development of the concepts are quite similar. For most parts making the connection is about as complicated as indicated by the example above. The most important exception where one needs additional information for the translation is that the library uses the expression "continuity space" for what is usually referred to as "represented space". We decided to use this notion since represented spaces are tied to computability theory, which the Incone library avoids except on the meta-level. This has another important consequence for the translation between the results of the paper and the formal development: The formal versions will only prove continuity where the paper claims computability. The user can verify the computability claim by inspecting the definitions of the realizers. Whenever a result proves a function with name `fun_name` continuous, there will be a function whose name contains the string `fun_name_rlzr`. It can be checked that this definition does not rely on any assumptions, so that it certifies computability. How to check this manually is described in more detail below.

The smaller differences between the formal and the informal development of concepts include the definition of the space of functions $\mathbf{Y}^{\mathbf{X}}$: While in this paper we chose the underlying set to be the continuous functions, in the formal development we use the co-domain of the function-space representation for the definition. In Incone the space of functions is denoted by `cs_fun` with the notation `_ c-> _` and that its underlying set contains exactly the continuous functions is proven separately (`ass_cont`). The reason for this choice is to minimize the strength of the axioms that are automatically assumed whenever spaces of functions are used. The proof that every continuous function has an associate requires a considerably stronger background theory than most of the other basic results one would be interested in. Another design choice for the sake of minimizing the use of axioms is that Incone does not use the Coq-internal notion of equality for equality of multivalued functions. Instead, it defines this equality as an equivalence relation `_ =~= _`. This avoids the need to assume the axiom of propositional extensionality that, while being theoretically justified and fairly commonly used, is known for having caused inconsistencies in the past. As such equivalence relations are a common sight in constructive mathematics, Coq provides the necessary infrastructure to make most operations equally effortless for equalities and equivalence relations. Thus this detail can mostly be ignored by the user.

The Incone library has some sublibraries that are of separate interest and have fewer dependencies. These parts can be obtained individually and we give an overview over their purpose in Section A.2. Before we do so, let us sketch what steps need to be taken to get an installation of Incone up and running and verify the correctness of the results described in this paper. We will assume that the user has successfully installed Coq on his machine, instructions of how to do so are available online.

A.1. **How to get started.** The paper describes release version 1.0 of Incone. As Incone is an ongoing project, newer versions will become available and for a description of the newest version, the GitHub repository can be consulted [Ste19a]. For readers only interested in verifying the results of the paper, we give instructions how to install and use version 1.0 of the library here. This version of Incone has been tested with version 8.9.0 of the Coq proof assistant. Additionally, the Coquelicot library (version 3.0) and the ssreflect package from math-comp (version 1.7.0) needs to be installed. To install Incone, the libraries mf, Rlzrs, Metric and Incone have to be downloaded and installed in this order. They can be found in their respective GitHub repositories [Ste19c, Ste19b, Ste19d, Ste19a]. Alternatively, the paper's project page `https://holgerthies.github.io/continuity` offers a package containing all libraries.

All of the claims the paper makes about the incone library can be checked by installing the library, opening a new Coq file with the following includes in the preamble.

```
From mf Require Import choice_mf.
From rlzrs Require Import all_rlzrs.
From metric Require Import all_metric Qmetric.
From incone Require Import all_cs classical_func
                           classical_cont classical_mach
                           Duop Q_reals baire_metric.
```

Each result in the paper states the name of the corresponding lemma in Incone in brackets after the number of the result. The formal lemmas and theorems can be inspected via `Check lem_name`, definitions can be detailed using `Print def_name` and notations by `Locate "not_name"`. In the case where multiple results share a name, `Locate "lem_name"` lists all lemmas with that name and unique identifiers and `Print Assumptions lem_name` shows the axioms that the result assumes. To list all results involving a concept Coq's search function can be used via `Search _ (concept)` and `Search "phrase"` might be useful where `phrase` is an expected substring of a lemma's name. Coq's search function can also be used with multiple arguments and the underscore above is due to the first argument being treated differently. Omitting it will lead to only functions returning `concept` being listed as a result and a warning that this may not be the intended behavior.

For example, one of the theorems from the paper has the name `cont_comp`. Typing

```
Check cont_comp.
```

prints the following statement:

```
cont_comp
     : forall (X Y Z : cs) (f : Y -> Z) (g : X -> Y),
        f \is_continuous -> g \is_continuous -> (f \o_f g) \is_continuous
```

Here, `cs` is a short notation for `continuity_space`, which is the Incone equivalent of a represented space. The notation `_ \o_f _` is for composition of functions, the shorter `_ \o _` is preserved for composition of multivalued functions. The rest of the notations should be self-explanatory but it may be worth noting that the use of a \ for non-prefix notations is to avoid blocking too many keywords. The user may first use `Locate "_ \is_continuous"` to find that this is a notation for the definition `continuous`. Then the details of the definition can be printed using the command `Print continuous`. The above may or may not be the desired theorem from the paper. Typing

```
Locate cont_comp.
```

shows that there are several lemmas with this name and the location of the files that contain them.

For users that are not only interested in verifying the results from the paper, the examples folder is a good starting point. Most of the the examples are documented via comments in the files and contain some rudimentary proofs that can be used to get familiar with using the library. A reader familiar with Coq will notice right away that these proofs heavily use the ssreflect proof language. This is true for the whole of the development and a comprehensible overview over this language and its advantages can be found in [GMT16].

A.2. **The structure of Incone and its sub-libraries.**

Some concepts described in this paper are expected to have applications that are unrelated to computable analysis and the most important ones have been exported to small libraries that can be installed separately and have less dependencies. For instance, due to its potential applications outside of Incone the development of a convenient environment for manipulation of multifunctions was exported and can be obtained separately as the mf-library [Ste19c]. Already the Incone library uses multifunctions for several different purposes: Through the Rlzrs library for the formulation of realizability, but also for dealing with partiality issues in Coq and in their traditional role in computable analysis as formalization of computational tasks.

Let us list all the packages that we decided to export from Incone with a short description of their contents.

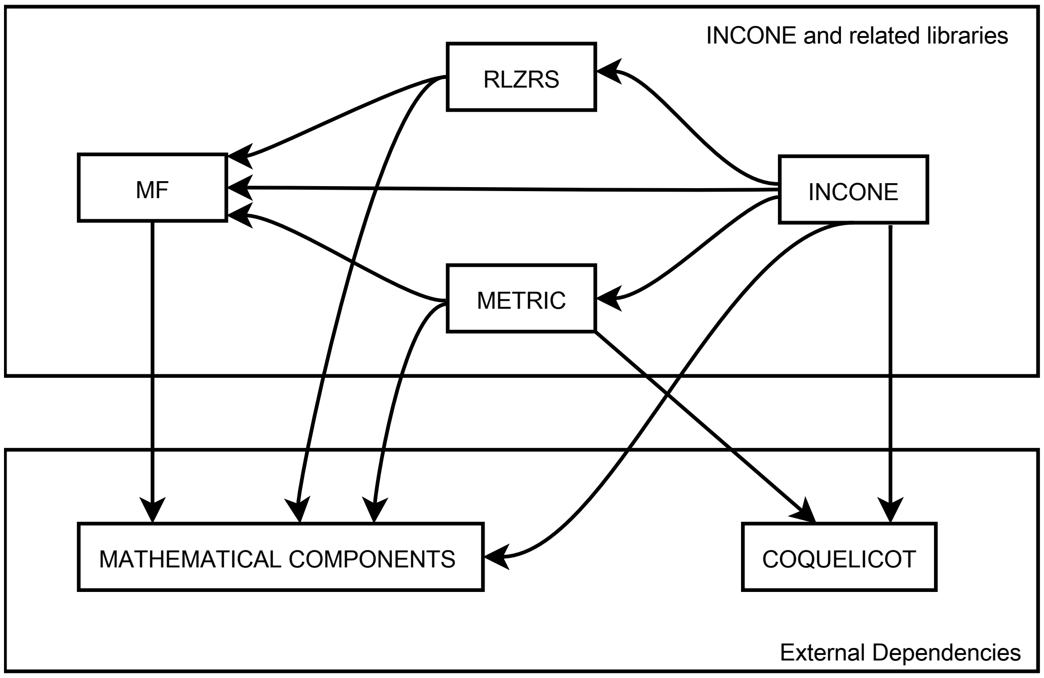


FIGURE 7. Internal and external dependencies of the incone library.

- **mf:** The MF library contains statements from the theory of multivalued functions. The formalization closely follows the description in the first part of Section 2. Several useful notations are introduced, for example `_ ->> _` for multivalued functions and `_ \tightens _` for the tightening relation. An exhaustive list of the important concepts can be found in the preamble of the file `mf.v` in the library.
- **Rlzrs:** A library for basic realizability using the question and answer structure described in the beginning of Section 3. It defines basic notations such as `_ \is_name_of _` or `_ \realizes _` and heavily relies on the MF library.
- **Metric:** A classical formalization of metric spaces. This library contains results that are independent from the theory of represented spaces. It also defines the metric notion of a limit, an efficient limit and proves some statements about continuity. The layout of METRIC roughly follows that of the Coquelicot [BLM15] library. However, the definitions are kept close to the classical mathematical treatment and are thus most similar to the metric spaces that can be found in COQ's standard library.

The libraries above can, and have to be, installed separately but should be considered part of the INCONE system. However, INCONE also has some external dependencies. As mentioned before it uses the ssreflect proof language. This language originates from the mathematical components development and INCONE also uses other parts of that library [GMT16]. Another external dependency that INCONE shares with METRIC is Coquelicot, a popular formalization of some facts from analysis that is conservative over the axiomatization of the real numbers in the standard library [BLM15]. See Figure 7 for an overview of INCONE, its sublibraries and dependencies.

The mf and Rlzrs libraries are fairly manageable and their few files can be navigated by hand and are well documented. The Metric library is less accessible and heavily interleaved with both the Coquelicot library and the development of metric spaces in Coq's standard library. Thus, we take some space to sketch some of its more important features and give additional justification of its existence.

A.3. The library for metric spaces.

Metric spaces are available in Coq's standard library and there are several other developments that include a treatment of metric spaces or related concepts. The original intention of the Metric library was to provide an interface between the the metric spaces from the standard library and a similar concept provided by Coquelicot. Coquelicot is a popular library for doing analysis in Coq and is conservative over the axiomatization of the real numbers in the standard library. The concept from Coquelicot that we are referring to is named a "uniform space" in Coquelicot. However, these spaces are more restrictive than what a mathematician would expect to be called a uniform space and instead closely resemble pseudo-metric spaces. For Coquelicot the choice of pseudo-metric spaces over metric spaces is due to neither Coq, nor the axioms of the real numbers, implying functional extensionality. This makes it challenging to define a metric on any kind of space of functions. A pseudo-metric can often be defined in a straightforward manner. The definitions of limits and continuity used in Coquelicot rely on filters instead of sequences. As the "uniform spaces" are Coquelicot's most general structure and are first-countable, the definitions via filters are equivalent to those using sequences. However, it is not clear whether the equivalence of the derived concepts can be proven in the setting that Coquelicot works in.

The Metric library provides interfaces with both the standard library of Coq (`MS2M_S`, `M_S2MS`, `Uncv_lim`, `cont_limin`, etc.) and the Coquelicot library (`US2MS`, `MS2US`, `cntp_cntp`, etc.) so that it is possible to reuse results proven there (for instance the lemmas `limD`, `limM`, `R_cmplt` that assert the limit to commute with additional multiplication and the real numbers to be complete are proven this way). The definitions of continuity that the metric library uses are as stated in this paper in the chapter about metric spaces. While these definitions quantify over real numbers ε and δ, restricting the quantifiers to only reach over a discrete subset of the real numbers often leads to equivalent definitions. For an easy back and forth between the different convergence statements Incone provides a line of lemmas that contain the phrase "tpmn" (for "two to the power minus n") in their name (`tpmnP`, `lim_tpmn`, `dns_tpmn`, etc.). For instance `tpmnP` proves that $2^{-n} \leq 2^{-m}$ on real numbers reflects $m \leq n$ on natural numbers and `lim_tpmn` says that in the definition of the limit one may replace ε by 2^{-n} and thereby quantification over $\mathbb{R}$ by quantification over $\mathbb{N}$. A similar set of lemmas is provided for replacement of ε and δ with rational numbers. For rational numbers Incone also provides a constructive instantiation of the restriction of the up function (`upQ`, `limQ`, `archimedQ`, etc.). The function `upQ`, for instance, is useful for recovering computational content from proofs in the standard library, as it may in some cases be used as substitute for the non-computable function `up` defined on all reals.

Finally, the Metric library proves some results that are not proven in Coquelicot, most of these rely on the axioms that our development assumes over the assumptions made by Coquelicot. It also defines an interface for handling sub-spaces as continuity of partial functions is important for our purposes. In the case of metric spaces both continuity and its sequential variant can be recovered from analogous point-wise notions while for represented spaces this is only the case for sequential continuity. The availability of point-wise notions is not always an advantage and can introduce subtle problems in the treatment of sub-spaces. Even in the most well-behaved cases there is a difference between a function being continuous in each point of the subset and the restriction of the function being continuous. For instance, the characteristic function of the closed unit interval has a continuous restriction but is not continuous in either end-point. This can easily lead to confusion and as a result some of the statements of important theorems about continuous functions on the real numbers from the standard library differ slightly from what a mathematician would expect them to say. Examples for this include the mean value and the intermediate value theorem.