Accepted Manuscript

Parametrised second-order complexity theory with applications to the study of interval computation

Eike Neumann, Florian Steinberg

PII: S0304-3975(19)30288-9
DOI: https://doi.org/10.1016/j.tcs.2019.05.009
Reference: TCS 11993

To appear in: *Theoretical Computer Science*

Received date: 4 October 2018
Revised date: 1 March 2019
Accepted date: 8 May 2019

Please cite this article in press as: E. Neumann, F. Steinberg, Parametrised second-order complexity theory with applications to the study of interval computation, *Theoret. Comput. Sci.* (2019), https://doi.org/10.1016/j.tcs.2019.05.009

Parametrised second-order complexity theory with applications to the study of interval computation[1]

Eike Neumann

Aston University
School of Engineering & Applied Science
Aston Triangle
Birmingham B4 7ET

Florian Steinberg[2]

INRIA Saclay
Toccata Team
Bât 650, Rue Noetzlin
91190 Gif-sur-Yvette

Abstract

We extend the framework for complexity of operators in analysis devised by Kawamura and Cook (2012) to allow for the treatment of a wider class of representations. The main novelty is to endow represented spaces of interest with an additional function on names, called a parameter, which measures the complexity of a given name. This parameter generalises the size function which is usually used in second-order complexity theory and therefore also central to the framework of Kawamura and Cook. The complexity of an algorithm is measured in terms of its running time as a second-order function in the parameter, as well as in terms of how much it increases the complexity of a given name, as measured by the parameters on the input and output side.

As an application we develop a rigorous computational complexity theory for interval computation. In the framework of Kawamura and Cook the representation of real numbers based on nested interval enclosures does not yield a reasonable complexity theory. In our new framework this representation is polytime equivalent to the usual Cauchy representation based on dyadic rational approximation. By contrast, the representation of continuous real functions based on interval enclosures is strictly smaller in the polytime reducibility lattice than the usual representation, which encodes a modulus of continuity. Furthermore, the function space representation based on interval enclosures is optimal in the sense that it contains the minimal amount of information amongst those representations which render evaluation polytime computable.

[1] This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 731143.

[2] The second author was supported by the ANR project *FastRelax*(ANR-14-CE25-0018-01) of the French National Agency for Research.

Contents

1 Introduction

Computable analysis is an extension of the theory of computation over the natural numbers to continuous data, such as real numbers and real functions, based on the Turing machine model of computation. Computability of real numbers is studied already in Turing's paper [Tur37] on the halting problem. Computability of real functions was introduced by Grzegorczyk [Grz57, Grz55a, Grz55b] and Lacombe [Lac55]. Kreitz and Weihrauch [KW85, Wei00] introduced a general theory of computation on second-countable T_0-spaces. This was further generalised by Schröder [Sch02a, Sch02b] to T_0 quotients of countably based spaces, which constitute in a certain sense the largest class of topological spaces which can be endowed with a reasonable computability structure [Sch02b, Theorem 13].

One of the goals of computable analysis is to provide mathematically rigorous semantics for computation over continuous data structures. Algorithms in numerical analysis are usually described using the real number model, where real numbers are regarded as atomic entities. A widely used mathematically rigorous formalisation of this idea is the Blum-Shub-Smale [BSS89, BCSS98] machine. Such algorithms cannot be implemented directly on a digital computer, as real numbers cannot be encoded with a finite number of bits. The usual substitution for real numbers are floating point numbers, which behave quite differently. For instance, addition and multiplication on floating point numbers are not even associative. Thus, the behaviour of floating point algorithms depends on phenomena that are absent in the real number model, such as numerical stability and error propagation. These issues have to be studied separately, which usually requires a substantial amount of additional effort. Even then, the precise contract that an implementation fulfils is usually not fully specified and the semantics of the implementation remain vague. As a consequence the semantics of an algorithm can differ considerably from the semantics of its implementation and different implementations may well have different semantics.

By contrast, any algorithm based on computable analysis can be implemented directly on a physical computer. It consists of a rigorous specification of input and output, so that it precisely describes the steps that have to be taken to obtain the desired result to a given accuracy. Software packages based on computable analysis include `iRRAM` [Mül], Ariadne [BBC+08], AERN [Kon], and RealLib [Lam06].

For the study of practical algorithms it is clear that computational complexity should play a central role. Whilst the notion of computability over continuous data is robust, well understood, and universally accepted in the computable analysis community, computational complexity in analysis is far less developed and even some basic definitions are the subject of ongoing debate. The study of computational complexity in analysis was initiated by Friedman and Ko [KF82]. They defined the computational complexity of real numbers and of real functions on compact intervals and proved some famous hardness results for problems such as integration, maximisation, or solving ordinary differential equations (see [Fri84, Ko83], cf. also [Kaw10]). This line of research is summarised nicely in Ko's book [Ko91]. The main gap in the work of Friedman and Ko is that, while their definition of computational complexity for real functions carries over to functions on compact metric spaces, it does not generalise easily to functions on non-compact spaces. In practice one is most interested in the study of operators on infinite dimensional vector spaces, such as spaces of continuous functions, L^p-spaces or Sobolev spaces. The aforementioned hardness results concern operators of this kind, but they are non-uniform in the sense that they establish that the operators in question map certain feasibly computable functions to functions of conjecturally [1] high complexity. While such non-uniform lower bounds are sufficient to show

[1] assuming standard conjectures in computational complexity such as P $\neq$ NP.

that an operator is infeasible, it remained unclear which operators should be considered feasible.

One of the main reasons for such a notion not being available was the lack of an accepted notion of feasibility for second-order functionals. A candidate solution had been proposed by Mehlhorn already in 1975 [Meh76]: The class of **basic feasible functionals**, which he defined by means of a generalisation of a limited recursion scheme that leads to a well-known characterisation of the polytime computable functions on the natural numbers. However, it remained a point of debate for a long time to which extent this class fully captures the intuitive notion of feasibility [Coo92]. Further investigations into this topic revealed the type-two basic feasible functionals to be a very stable class that became established as the foundation of second-order complexity theory [Pez98, IRK01]. An important step in this process, and something that opened the field up for applications, was the characterization of the basic feasible functionals by means of resource bounded oracle Turing machines due to Kapron and Cook [KC96]. Based on this characterization, Kawamura and Cook introduced a framework for complexity of operators in analysis [KC12] that generalizes the definition of feasibly computable functions of Friedman and Ko to a wider class of spaces, including the aforementioned examples. This kicked off a series of investigations [FGH14, FZ15, Ste17, SS17, and many more].

However, there remains a gap between theory and practice. Within the framework of Kawamura and Cook it is impossible to model the behaviour of software based on computable analysis such as the libraries mentioned above. The reason for this is that all these implementations are based on interval arithmetic or extensions thereof, such as Taylor models. The representations which underlie these approaches are known to exhibit highly pathological behaviour within the framework of Kawamura and Cook [KP14a, Ste16] and those representations which can be used within their framework do not always seem to be an appropriate substitute. For instance, in the Kawamura-Cook model any representation of the space of continuous functions which renders evaluation polytime computable also allows for the computation of some modulus of continuity of a given function in polynomial time. In `iRRAM` this requires an exponential exhaustive search, see [BS17].

The present work is an attempt to bridge this gap by extending the framework of Kawamura and Cook in order to develop a meaningful complexity theory for a broader class of representations.

We do so by endowing a represented space (X, ξ) with an additional function $\mu\colon \mathrm{dom}(\xi) \to \mathbb{N}^{\mathbb{N}}$, called the **parameter**, which is intended to measure the complexity of the names of elements of X. These parameters are a generalisation of the size function which is used to measure complexity of string functions in second-order complexity theory. The pair (ξ, μ) is called a **parametrised representation** and the triple (X, ξ, μ) is called a **preparametrised space**. The complexity of an algorithm for computing a function $f\colon X \to Y$ between preparametrised spaces is measured by the dependence of the running time of the algorithm in terms of the parameter of the input name and by the growth in size of the parameter of the output name compared with the parameter of the input name. As in the Kawamura-Cook model, polytime computability is defined using second-order polynomials: An algorithm runs in polynomial time if and only if both its running time and the parameter of the output name are bounded by a second-order polynomial in the parameter of the input name. A preparametrised space is called a **parametrised space** if its identity function is polynomial time computable.

The pathological behaviour of the representation of real numbers based on interval enclosures is eliminated by the natural choice of a parameter for this representation. The resulting parametrised representation is polytime equivalent to the usual Cauchy representation with the size function as parameter (Proposition 2.11). In particular, a real function is polytime comput-

able with respect to the parametrised interval representation if and only if it is polytime computable in the usual sense. While this result might suggest that nothing much is gained from this new definition, we show that the natural uniform complexity structure on the space of real functions viewed as a parametrised space is different from the complexity structure induced by the Kawamura-Cook representation, and that the complexity induced by the natural parametrised space structure corresponds more closely to the complexity of operators in practical implementations. We define a parametrised space $C(I)_i$ of continuous functions based on interval enclosures, and show that this space most precisely reflects the behaviour of functions expected from implementations: On one hand, it leads to the right notion of polytime computability of functions (Corollary 2.21), function evaluation is polytime computable (Proposition 2.18) and the same is true for many of the usual operations one wants to compute quickly (Theorem 2.22, Proposition 2.23). On the other hand, finding a modulus of continuity, which is notoriously slow in practice, is provably not polytime computable (Corollary 3.14). It is proved in the appendix that this space is isomorphic to a natural model of the space of real functions used in `iRRAM` (Proposition A.6).

We investigate the parametrised space of interval functions further and prove that any other parametrised representation of this space such that evaluation is polytime computable can be translated to it (Theorem 2.19). There are two reasons why we consider this result to be especially important: Firstly it resembles a result that Kawamura and Cook proved about a representation they introduced, which is currently considered to be the standard representation for the continuous functions on the unit interval for this reason. We compare their representation to the representation using interval enclosures and show that it sits strictly higher in the lattice of polytime translatability (Corollary 3.13). This reflects the fact that the minimality result by Kawamura and Cook ranges over a restricted class of parametrised spaces. We characterise the spaces they consider as essentially those that have a polytime computable parameter (Theorem 3.5). The second reason why we consider the minimality result for interval functions to be important is that it includes a quantification over all parametrised representations. This demonstrates that the definition of a parametrised space is not chosen too general to allow for meaningful results. Throughout the paper we provide more support for our belief that parametrised spaces are a good general framework for complexity considerations in computable analysis (for instance Theorem 2.13).

Related work. Parameters and parametrised complexity in our sense are present in the work of Rettinger [Ret13] and Lambov [Lam05]. Rettinger works in a different setting, avoiding second-order polynomials and we significantly add to and modify Rettinger's ideas. Lambov's work includes a good part of our results on interval representations in a different language. However, Lambov does not attempt to build a general framework of parametrised complexity in analysis, which requires further ideas that are not present in his work. We hence believe that the present work extends his results considerably.

For a restricted case some of the core definitions proposed in this paper are also present in the work of Kawamura, Müller, Rösnick and Ziegler [KMRZ12]. The authors introduce parameter functions with integer values. This covers only a very special case of preparametrised spaces, namely those where the value of the parameter on a name is a constant function. More significantly, their applications all make the value of the parameter accessible to the algorithm and therefore allow for formulation in the framework of Kawamura and Cook by modification of the representations involved. For their applications this is unavoidable, as the functions they consider fail to be feasibly computable unless the parameter is provided as extra advice. Most of the content of the preprint [KMRZ12] was published in [KMRZ15]. Unfortunately, in the published version the authors decided to further restrict the definition

of parameters, by making it a function on the represented space, rather than a function on the domain of the representation. They have to recover the desired behaviour by first introducing suitable covering spaces of the spaces of interest. This makes it a lot more difficult to see the connection between their work and ours. Similar applications of parametrised complexity are found for instance in [KTZ18, KST18, PG16].

We also consider work by Schröder to be related [Sch04]. There are two ways in which his results relate to the contents of this paper: The first connection is that he equips a representation with an additional integer-valued size function that can, just like in the previous paragraph, be considered a special case of a parameter. The second connection is more interesting but also more difficult to make: Schröder provides conditions for represented spaces under which every machine that computes a function between these spaces has a well-defined first-order running time. This can be interpreted as devising a pair of a representation and a parameter for the computable functions between two spaces such that evaluation is polytime computable: The representation takes an index of a machine computing a realiser of a function to that function and the parameter assigns to such an index the running time of the machine it encodes. The running time of the evaluation operation is the overhead needed for simulating a machine. However, one should note that these observations are not explicitly stated in [Sch04]. Also, Schröder does not consider polytime computability but only reasons about the existence of time bounds in general.

Kawamura and Pauly [KP14a] study exponential objects in the category of polytime computable mappings. They consider one of the standard function space constructions from computable analysis, which is obtained by encoding a continuous function by an index of a Turing machine together with an oracle, such that the machine computes the function relative to the oracle. They show that well-behaved function spaces can be constructed for a class of spaces which they call "effectively polynomially bounded". This class of spaces is very similar to the one considered by Schröder. Their function space construction can be viewed as an extension of the construction sketched in the previous paragraph by adding arbitrary oracles. Like Schröder they measure the size of objects in effectively polynomially bounded spaces by means of a parameter function with values in the natural numbers. Their work is also the only example in the literature that we are aware of that discusses the issue of polytime computability of the identity function on a parametrised space. Curiously, in the published version of the paper the connections to parametrised complexity are significantly obscured. The connection to our work is much more visible in an early preprint [KP14b].

Outline of the paper. Section 1 recalls the most important concepts from second-order complexity theory. The approach via resource bounded oracle Turing machines is chosen, and this is essential for rest of the paper. The discussion of the framework of Kawamura and Cook , which imposes an additional condition on the names, is postponed to the later Section 3. The second part 1.2 recalls the Cauchy and the interval representation of the real numbers and discusses how second-order complexity theory can be applied to find a well behaved notion of complexity of the former and why the same approach fails for the latter.

Section 2 introduces the general concept of parametrised space. Polytime computable functions are introduced using second-order polynomials. It is shown that they are closed under composition. Section 2.1 applies this to the representation of real numbers based on sequences of nested intervals. It is shown that the parametrised space of interval reals is polytime isomorphic to the parametrised space of Cauchy reals with the size function as parameter. In particular, the polytime computable points of the interval representation are the usual polytime computable real numbers.

Section 2.2 introduces a parameter for the space of continuous functions

on the unit interval, where a function is represented by an interval enclosure. It is shown that this choice of representation and parameter is optimal in the sense that the resulting structure of parametrised space on the space of continuous functions is minimal amongst those structures which render evaluation polytime computable. In particular, the polytime computable points of this representation are the usual polytime computable functions.

Section 3 compares the approach presented here to the one of Kawamura and Cook. Theorem 3.5 characterises those parametrised spaces which admit a polytime equivalent length-monotone representation and thus can be treated within the framework of Kawamura and Cook. Section 3.2 shows that the interval representation of continuous functions is not of this kind. Hence, the interval representation of continuous functions contains strictly less information than the function space representation which is used by Kawamura and Cook. This result mainly relies on an auxiliary result due to Brauße and Steinberg [BS17].

Appendix A discusses some alternative choices of representations to demonstrate the robustness of our definitions. The representation for real numbers used in Appendix A also coincides with the one used by Müller to model the behaviour of `iRRAM` [Mül01].

1.1 Second-order complexity theory

Fix a non-empty alphabet Σ. Let $M^?$ be an oracle machine, that is, a Turing machine with two designated tapes called the 'oracle query' and the 'oracle answer' tape and a special state called the 'oracle state'. An oracle for such an oracle machine is an element of Baire space $\mathcal{B} := \Sigma^{*\Sigma^*}$. The oracle machine $M^?$ can be executed on a given string $\mathbf{a} \in \Sigma^*$ with a given oracle $\varphi \in \mathcal{B}$. It is executed like a regular Turing machine, except that whenever it enters the oracle state, the current content of the oracle answer tape is replaced with $\varphi(\mathbf{b})$, where $\mathbf{b}$ is the current content of the oracle query tape. If the oracle machine $M^?$ terminates on input $\mathbf{a}$ with oracle φ, the string that is written on its output tape after termination is denoted by $M^\varphi(\mathbf{a})$. This defines a partial function M^φ which maps a string $\mathbf{a}$ such that $M^?$ terminates on $\mathbf{a}$ and with oracle φ to the string $M^\varphi(\mathbf{a})$. Every oracle machine $M^?$ computes a partial operator $F_{M^?} :\subseteq \mathcal{B} \to \mathcal{B}$ on Baire space: The domain of $F_{M^?}$ consists of all oracles $\varphi \in \mathcal{B}$ such that M^φ is total. If φ is an element of the domain of F then the value of F in φ is given by $F_{M^?}(\varphi) := M^\varphi$.

This is slightly different from the definition of oracle Turing machine in classical computability theory, where the oracles are subsets of Σ^*. While this is not important for computability considerations, it does make a difference when it comes to computational complexity. To be able to reason about complexity in this extended setting it is necessary to fix a convention for how to count oracle calls for the time consumption of a machine. We adapt the most common convention: An oracle call is considered to take one time step in which the entire answer appears on the oracle answer tape. Essentially this means that the machine is not forced to read the whole oracle answer. Another detail is the position of the read/write heads. We assume that the head position does not change during the oracle interaction. Denote the number of steps the oracle machine $M^?$ takes on input $\mathbf{a}$ and with oracle φ by $\mathrm{time}_{M^?}(\varphi, \mathbf{a})$. Of course, one of the sanity checks for any reasonable computational model is that the details fixed above should be irrelevant. One should end up with the same computational complexity classes when, for instance, the machine returns its head to the beginning of the oracle query tape during the oracle interaction.

Since the oracle φ is considered an input of the computation, a function bounding the time consumption of an oracle machine should be allowed to depend on the size of the oracle in addition to the size of the input string. The most common way to measure the size of a string function is to use the

worst-case length-increase from input to output. That is, for a string function φ let its **size** $|\varphi| : \mathbb{N} \to \mathbb{N}$ be defined by

$$|\varphi|(n) := \max_{|\mathbf{a}| \leq n} \{|\varphi(\mathbf{a})|\}. \tag{s}$$

Since a time bound for an oracle machine should produce a bound on the number of steps the execution takes from a size of the oracle and a size of the input it should be of type $\mathbb{N}^{\mathbb{N}} \times \mathbb{N} \to \mathbb{N}$. To talk about polytime computability it is necessary to find a subclass of functions of this type that is considered to have "polynomial" growth.

Definition 1.1 The class of **second-order polynomials SOP** $\subseteq \mathbb{N}^{\mathbb{N}^{\mathbb{N}} \times \mathbb{N}}$ is the smallest class such that the following conditions hold:

- For all $p \in \mathbb{N}[X]$ we have $(l, n) \mapsto p(n) \in \mathbf{SOP}$.
- Whenever $P \in \mathbf{SOP}$ then also $(l, n) \mapsto l(P(l, n)) \in \mathbf{SOP}$.
- Whenever $P, Q \in \mathbf{SOP}$ then both the point-wise sum $P + Q$ and the point-wise product $P \cdot Q$ are contained in **SOP**.

Since second-order polynomials are used as running time bounds, only the values on functions that turn up as sizes of string functions are relevant. These are exactly the non-decreasing functions, *i.e.*, functions l satisfying $l(n+1) \geq l(n)$. This restriction is important, as the following lemma fails for more general arguments:

Lemma 1.2 (Monotonicity) *Let P be a second-order polynomial and let $l, k : \mathbb{N} \to \mathbb{N}$ be non-decreasing functions such that l is point-wise bigger than k. Then*

$$\forall n \in \mathbb{N} \colon P(l, n) \geq P(k, n).$$

The proof is a straightforward induction.

Definition 1.3 We call a partial operator $F \colon \subseteq \mathcal{B} \to \mathcal{B}$ **polytime computable**, if there is an oracle machine $M^?$ that computes F and a second-order polynomial P such that

$$\forall \varphi \in \mathrm{dom}(F), \forall \mathbf{a} \in \Sigma^* \colon \mathrm{time}_{M^?}(\varphi, \mathbf{a}) \leq P(|\varphi|, |\mathbf{a}|).$$

It was proved by Kapron and Cook [KC96] that a total operator $F : \mathcal{B} \to \mathcal{B}$ is polytime computable if and only if the corresponding functional $(\varphi, \mathbf{a}) \mapsto F(\varphi)(\mathbf{a})$ is basic feasible in the sense of Mehlhorn [Meh76]. The generalization to partial operators adds an additional choice: By our choice the machine is only required to comply with the time bound in the case where the oracle is in the domain of the operator. Another approach would be to require the existence of a total extension that runs in polynomial time. This corresponds to replacing the quantification over $\mathrm{dom}(F)$ by a quantification over all of Baire space. It is possible to prove that this does indeed lead to a more restrictive notion of polytime computability [KS17].

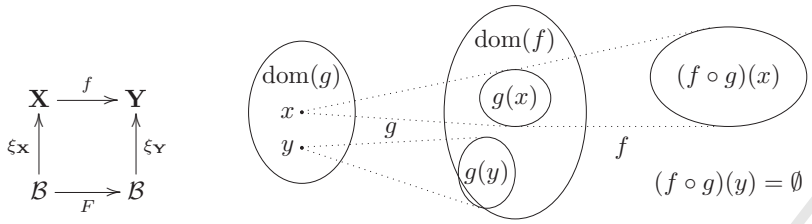
To show closure of the class of polytime computable operators under composition, one needs monotonicity from Lemma 1.2 and the following closure property of the second-order polynomials:

Proposition 1.4 *Whenever P and Q are second-order polynomials, then so are the following mappings:*

$$(l, n) \mapsto P(l, Q(l, n)) \quad \textit{and} \quad P(Q(l, .), n).$$

Just like Lemma 1.2, Proposition 1.4 can be proven by a straightforward induction on the structure of second-order polynomials.

Theorem 1.5 (Composition) *Whenever F and G are polytime computable, then so is $F \circ G$.*



(a) F realises f (b) Multifunction composition visualized

Figure 1: The diagram (a) need not commute. The function F realises f if $\xi_{\mathbf{Y}} \circ F$ extends $f \circ \xi_{\mathbf{X}}$ (or tightens it if f is multivalued).

This is proven in a more general setting in Theorem 2.4 and we refrain from restating the proof here.

To compute on more general spaces representations are used:

Definition 1.6 Let X be a set. A **representation ξ of X** is a partial surjective function $\xi :\subseteq \mathcal{B} \to X$. A **represented space** is a pair $\mathbf{X} = (X, \xi)$ of a set and a representation of that set.

The elements of $\xi^{-1}(x)$ are called the **names** of x. An element of a represented space is called **computable** if it has a computable name.

Computability of functions between represented spaces can be defined via realisers.

Definition 1.7 Let $f : \mathbf{X} \to \mathbf{Y}$ be a function between represented spaces. A function $F :\subseteq \mathcal{B} \to \mathcal{B}$ is called a **realiser** of f if it translates names of the input to names of the output, that is if

$$\varphi \in \mathrm{dom}(\xi_{\mathbf{X}}) \Rightarrow \xi_{\mathbf{Y}}(F(\varphi)) = f(\xi_{\mathbf{X}}(\varphi)).$$

(also compare Figure 1a.) A function between represented spaces is called **computable** if it has a computable realiser. It is called **polytime computable** if it has a polytime computable realiser.

The later parts of this paper need a slight generalisation of the above to multi-valued functions. Recall that a **multifunction** $f : X \rightrightarrows Y$ assigns to each element $x \in X$ a subset $f(x) \subseteq Y$. Its **domain** $\mathrm{dom}(f)$ consists of all $x \in X$ whose image under f is nonempty. A partial single-valued function $g :\subseteq X \to Y$ can be identified with the multifunction which sends elements $x \in \mathrm{dom}(g)$ to the singleton $\{g(x)\}$ and elements outside of the domain of g to the empty set. The definition of computability using realisers generalizes to multifunctions in a straightforward way: A function $F :\subseteq \mathcal{B} \to \mathcal{B}$ is called a realiser of f if $\xi_{\mathbf{Y}}(F(\varphi)) \in f(\xi_{\mathbf{X}}(\varphi))$ for all $\varphi \in \mathrm{dom}(\xi_{\mathbf{X}})$. The elements of $f(x)$ are thus interpreted as "acceptable return values" for an algorithm which computes f. If $f : X \rightrightarrows Y$ and $g : Y \rightrightarrows Z$ are multifunctions then their composition $g \circ f$ is the multifunction with

$$\mathrm{dom}(g \circ f) = \{x \in X \mid f(x) \subseteq \mathrm{dom}(g)\}$$

and

$$g \circ f(x) = \{z \in g(y) \mid y \in f(x)\}.$$

This definition ensures that the composition of two realisers is a realiser of the composition. Note that while multivalued functions can formally be identified with relations, from conceptual point of view it is better not do do so. For instance, the composition defined above is different from the natural notion of composition for relations. Multifunctions are a standard tool in computable analysis to avoid certain kinds of continuity issues and are needed in Section 3.2 for this exact reason.

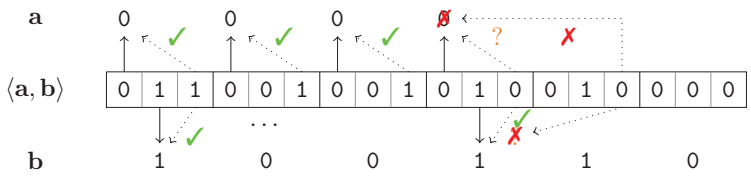


Figure 2: The pairing $\langle \mathbf{a}, \mathbf{b} \rangle$ of the strings $\mathbf{a} := \mathtt{000}$ and $\mathbf{b} := \mathtt{100110}$ as an example

1.2 Notations and complexity on the reals

A name of an element of a represented space should be understood as a black box that provides on-demand information about the object it encodes. For real numbers, for instance, a reasonable query to such a black box could be 'provide me with a 2^{-n} approximation to the real number', and the answer that the name provides should be such an approximation. The input and the output of the name are finite binary strings, and questions like the above can be formulated by encoding elements of discrete structures like the integers and the rational numbers.

A **notation of a space** X is a partial surjective mapping $\nu_X :\subseteq \Sigma^* \to X$. Fix the following standard notations: Let $\nu_{\mathbb{Z}}$ be the mapping defined on a string $\mathbf{a} = a_0 \mathtt{1} a_2 \dots a_{|\mathbf{a}|}$ whose second digit is a $\mathtt{1}$ by

$$\nu_{\mathbb{Z}}(\mathbf{a}) = (-1)^{a_0} \sum_{i=1}^{|\mathbf{a}|-1} a_i 2^{|\mathbf{a}|-i}$$

and zero on strings that do not have a second digit. A dyadic rational is a rational number of the form $\frac{r}{2^n}$ for some $r \in \mathbb{Z}$ and $n \in \mathbb{N}$. The set of these numbers is denoted by $\mathbb{D}$. The reason for their use is that they are a good model for machine numbers, as they are precisely those rational numbers which have a finite binary expansion. Encode a dyadic number as its unique finite binary expansion starting in a code of an integer followed by a separator symbol that is used to mark the position of the decimal point. To avoid confusion with unary and binary notations, we do not specify a notation for the natural numbers. Instead of working on $n \in \mathbb{N}$ directly, we use the integer 2^n. This means that implicitly use the unary encoding of natural numbers while we use the binary encoding for the integers.

To also be able to accept or return pairs of integers or dyadic numbers use a pairing function for strings. For technical reasons that become apparent in Section 2.1, we choose to use a very specific pairing function. For two strings $\mathbf{a}$ and $\mathbf{b}$ let the pairing $\langle \mathbf{a}, \mathbf{b} \rangle$ be the string which is constructed as follows: Let $\mathbf{c}$ be the string that that starts in the digit $\mathtt{1}$, repeated $\min\{|\mathbf{a}|, |\mathbf{b}|\}$ times, followed by a $\mathtt{0}$, then a bit indicating which of $\mathbf{a}$ and $\mathbf{b}$ is longer and finally enough $\mathtt{0}$'s to make it as long as the longer of the two strings. Then pad the strings $\mathbf{a}$ and $\mathbf{b}$ to the length of the longer of the two strings by adding zeros to the end. $\langle \mathbf{a}, \mathbf{b} \rangle$ is the string whose bits alternate between the digits of $\mathbf{c}$, $\mathbf{a}$ and $\mathbf{b}$. It is important for this paper that the n initial segment of $\mathbf{a}$ and $\mathbf{b}$ can be read from a $3(n+2)$ initial segment of $\langle \mathbf{a}, \mathbf{b} \rangle$.

In the following we use these encodings to identify Baire space with the space of functions between the encoded structures. For instance the statement '$\varphi : \mathbb{N} \to \mathbb{D}$ is a name of an element' is used as an abbreviation of the statement 'any function $\psi : \Sigma^* \to \Sigma^*$ such that $\nu_{\mathbb{Z}}(\mathbf{a}) = 2^n$ implies $\nu_{\mathbb{D}}(\psi(\mathbf{a})) = \varphi(n)$ is a name of the element'.

Definition 1.8 Define the **Cauchy representation** $\xi_{\mathbb{R}_c}$ **of** $\mathbb{R}$ as follows: A function $\varphi : \mathbb{N} \to \mathbb{D}$ is a name of a real number x if and only if $|\varphi(n) - x| \leq 2^{-n}$ holds for all $n \in \mathbb{N}$.

This adopts the widespread convention used in real complexity theory to provide accuracy requirements as natural numbers in unary. It would have equivalently been possible to provide a natural number n in binary and require the return value to be a $\frac{1}{n+1}$-approximation or to provide a dyadic rational ε and require the return value to be an ε-approximation. We refer to the space $\mathbb{R}_c := (\mathbb{R}, \xi_{\mathbb{R}_c})$, as the **represented space of Cauchy reals**. The representation $\xi_{\mathbb{R}_c}$ is used throughout literature with great confidence that it induces the right notion of complexity for real numbers and there are many results supporting this: The functions that have a polytime computable realiser are exactly those that are polytime computable in the sense of Ko [Ko91] as proved by Lambov [Lam06]. It is well known that Ko's notion can be reproduced in Weihrauch's type two theory of effectivity [Wei00].

While the Cauchy representation is in principle straightforward to realise on a physical computer, the inherent laziness of the datatype leads to undesirable memory overheads.

To illustrate this, consider the task of computing the iterations of the logistic map, *i.e.*, the n^{th} number of the sequence recursively defined by

$$x_0 := \tilde{x}, \quad \text{and} \quad x_{i+1} := r x_i (1 - x_i),$$

where r and $\tilde{x}$ are real numbers. An algorithm for computing x_n can be written in imperative-style pseudo-code as follows:

$$\begin{aligned} &x \leftarrow \tilde{x} \\ &\text{for } i \text{ in } 1 \,..\, n: \\ &\quad x \leftarrow r x (1 - x) \end{aligned}$$

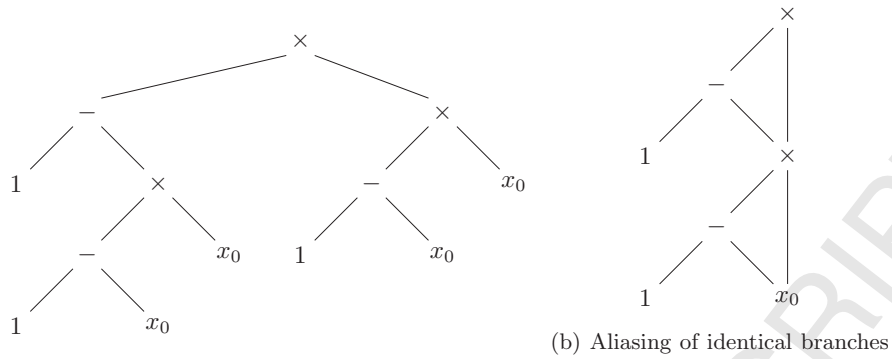
If x is taken to be, say, a floating point number or an interval with fixed precision endpoints, then the memory consumption of this program is essentially constant in n and linear in the number of bits used to encode x.

Now imagine that x is implemented as a Cauchy real instead. Then x is given as a function and therefore cannot be encoded with finitely many bits. Hence, the straightforward way to execute the above program in this case would be to first build the computation tree (compare Figure 3a). The evaluation algorithm would then proceed by propagating accuracy requirements from the root to the leaves, computing an approximation of each leaf to the respective required accuracy, and finally evaluate the tree bottom-up using these approximations. Not taking into account the extensive recomputation of approximations that the second step may lead to, the construction of the tree alone can lead to exponential memory consumption in n when done naively. This exponential overhead can be avoided by identifying identical subtrees, making the tree into a Directed Acyclic Graph (DAG) (compare Figure 3b) of linear size in n. Implementing this identification of subtrees is non-trivial in general, and it still leaves us with a linear memory overhead. For this reason, implementations like `iRRAM` use a different evaluation strategy: Guess an accuracy requirement and approximate all real numbers involved in the program to that accuracy. Use interval arithmetic to evaluate the program. If the end result is not sufficiently accurate, rerun the program with higher accuracy. This entirely avoids the memory overhead incurred from constructing the DAG.

While this approach comes with its own drawbacks, such as recomputation and frequent overestimation of the needed precision, software based on interval computation is empirically superior in speed and memory consumption to implementations based on the Cauchy representation.

This leads to a different choice of real number representation: Let $\mathbb{ID}$ denote the set of finite dyadic intervals together with the infinite interval $[-\infty, \infty]$. We use the abbreviation

$$[r \pm \varepsilon] := [r - \varepsilon, r + \varepsilon].$$



(a) The entire computation tree has exponential size

(b) Aliasing of identical branches leads to a DAG of linear size.

Figure 3: Computing a DAG for two iterations of the logistic map with $r = 1$.

Any finite dyadic interval can be written in this form and we encode such an interval as the pair of a code of r and a code of ε as dyadic numbers. Denote the length of a dyadic interval by $\operatorname{diam}([r \pm \varepsilon]) := 2\varepsilon$.

Definition 1.9 A function $\varphi\colon \mathbb{N} \to \mathbb{ID}$ is a $\xi_{\mathbb{R}_i}$-name of $x \in \mathbb{R}$ if $(\varphi(n))_n$ is a nested sequence of intervals with $\{x\} = \bigcap_{n\in\mathbb{N}} \varphi(n)$.

In certain implementations, notably in `iRRAM` [Mül], the monotone convergence assumption of Definition 1.9 is relaxed to convergence in the Hausdorff metric. A more thorough discussion of this can be found in Appendix A, where a proof is given that this choice makes no difference up to polytime equivalence. From a theoretical point of view it is much more convenient to work with monotone sequences of intervals.

The use of the interval representation is avoided in real complexity theory since it does not seem to lead to a good notion of complexity: Every real number has names that keep the sequence of intervals constant for an arbitrary long time before decreasing the size of the next interval and these names are of slowly increasing size. As a consequence, the represented space has very pathological complexity theoretical properties: On the one hand a function operating on names of this kind may need to read a very long initial segment before having any information about the encoded object available, while not being granted any time due to the small size of the input. As a consequence there are usually very few polytime computable functions whose domain is the space of real numbers endowed with the interval representation. On the other hand, a function that has to produce an interval name of a real number may delay the time until it returns information about the function value indefinitely. Consequentially, all computable functions with values in the real numbers with the interval representation are computable in linear time [Sch04, KP14a, Ste16].

The goal of the next section is to give a definition of computational complexity for spaces like the space of real numbers equipped with the interval representation which avoids such pathological behaviour.

2 Parametrised spaces

Definition 2.1 Let $\xi : \mathcal{B} \to X$ be a representation. A **parameter for** ξ is a single-valued total map μ from $\operatorname{dom}(\xi)$ to $\mathbb{N}^{\mathbb{N}}$ such that

$$\forall \varphi \in \operatorname{dom}(\xi), \forall n \in \mathbb{N}\colon \mu(\varphi)(n+1) \geq \mu(\varphi)(n).$$

The pair (ξ, μ) is called a **parametrised representation** of X. The triple (X, ξ, μ) is called a **preparametrised space**.

The monotonicity assumption guarantees that the second-order polynomials behave as expected, *i.e.*, it makes it possible to use the monotonicity of second-order polynomials from Lemma 1.2.

We do not make any assumptions about the computability or even continuity of the parameter here. This is for a few reasons: The first is that no assumptions of this kind are needed for the content of this paper. Results like the minimality from Theorem 2.19 do provide a construction that works without this assumption. Being more restrictive in the definitions would make these results less general. Another reason is that we do encounter discontinuous parameters in situations we consider to be of practical relevance. An example is discussed in more detail in Appendix A. While in this example an isomorphic space with continuous parameter can be found, it is easy to construct examples where this is not the case. Allowing discontinuous parameters hence allows us to investigate spaces that could otherwise not be equipped with a meaningful complexity notion.

A familiar class of parameters are restrictions of the size function

$$|\varphi|(n) := \max\left\{|\varphi(\mathbf{a})| \;\middle|\; |\mathbf{a}| \leq n\right\}.$$

Since the function $|\cdot| : \mathcal{B} \to \mathbb{N}^{\mathbb{N}}$ is total and all its values are non-decreasing, it can be used as a parameter for any representation. Its restriction to the domain of a representation is called the **standard parameter** for the representation. In principle, any represented space can be made into a preparametrised space by equipping it with the standard parameter of its representation.

We now come to the definition of computational complexity. We will specialise all definitions immediately to second-order polynomial time. While it is in principle straightforward to consider other classes of second-order resource bounds to obtain different complexity classes, second-order polynomial time computability is arguably the only higher-order complexity class that is sufficiently well-established in the literature.

Definition 2.2 Let (X, ξ_X, μ_X) and (Y, ξ_Y, μ_Y) be preparametrised spaces. A function $f\colon X \to Y$ is called **computable in polynomial time** if there is a machine $M^?$ that computes a realiser of f and two second-order polynomials P and Q such that the following conditions are satisfied:

- P bounds the running time of $M^?$ in terms of the parameter, *i.e.*, for all oracles $\varphi \in \mathrm{dom}(\xi_X)$ and strings $\mathbf{a}$ we have
$$\mathrm{time}_{M^?}(\varphi, \mathbf{a}) \leq P(\mu_X(\varphi), |\mathbf{a}|).$$
- Q bounds the parameter blowup of $M^?$, that is for all $\varphi \in \mathrm{dom}(\xi_X)$ and all $n \in \mathbb{N}$ we have
$$\mu_Y(M^\varphi)(n) \leq Q(\mu_X(\varphi), n).$$

We say that $M^?$ has polynomial running time and polynomially bounded parameter blowup with respect to the parameters.

We often conflate P and Q to a single polynomial which bounds both the running time and the parameter blowup. Let us call a realiser $F\colon \subseteq \mathcal{B} \to \mathcal{B}$ computed by a machine as in Definition 2.2 a **witness for the polytime computability of** f. In the case where both spaces come with the standard parameter a realiser F is a witness for the polytime computability of the function f if and only if it is polytime computable in the usual sense: The first condition coincides with the usual running time restriction and the second condition is automatic, as writing the output counts towards the total time consumption of the machine.

As we make no assumption on the relation between the parameter of a preparametrised space and the size function, Definition 2.2 does not guarantee that the identity on a preparametrised space is polytime computable. If it

is, the identity on Baire space need not be a witness for the polytime computability of the identity on the space. We hence need the following additional definition:

Definition 2.3 A preparametrised space $\mathbf{X} = (X, \xi_{\mathbf{X}}, \mu_{\mathbf{X}})$ is called a **parametrised space**, if the identity function $\mathrm{id}_{\mathbf{X}} : \mathbf{X} \to \mathbf{X}, x \mapsto x$ is polytime computable.

In the case where the parameter is the standard parameter polytime computability of the identity is automatic as the identity on Baire space is a witness for its polytime computability. In general the parameter might not provide enough time to read all of an oracle answer and proving polytime computability of the identity function usually boils down to proving that limited information can be read from a beginning segment of the result of an oracle query. An example of this is discussed in Proposition 2.10.

While Definition 2.3 might look innocent, its implications should not be underestimated. It implicitly connects the parameter of the space to the size function: While for an arbitrary name there need not be any relation, the time constraint imposed on a witness of polytime computability of the identity function forces that the size of the name it returns is bounded by a second-order polynomial in the parameter of the input name. The application of such a witness hence constitutes a normalisation procedure which reduces the size of excessively large names. This connection is in particular important as it guarantees the stability under small changes in the model of computation as discussed in Section 1.1. One could alternatively require from the beginning that the parameter be point-wise bigger than the size function, or that the identity on Baire space be a witness of polytime computability of the identity. While these alternatives are slightly more restrictive than our chosen definition, all three choices are essentially equivalent.

Why we chose the above definition over these alternatives is a subtle point. An obvious drawback of our choice is that the stability under changes to the model of computation is only true once a normalisation procedure has been applied. The proof that a preparametrised space is a parametrised space usually relies on the details of the computational model and the details of the encodings of the discrete structures and pairs. Once this fact has been established a space can be specified that is stable under changes of the model and isomorphic with respect to the present model. Despite this somewhat peculiar property, our chosen approach has the advantage that it allows for the most natural definition of both the representations and the parameters that we are interested in. The other definitions usually force that either the size function shows up in the definition of the parameter or the normalisation procedure is hard-coded into the definition of the representation. The former can sometimes lead to waste of resources, as it allows for wasteful encodings, while the latter usually includes a non-canonical choice. The proof of Proposition 2.10 is an instructive illustration of this.

Theorem 2.4 (Composition) *Let* $\mathbf{X}$*,* $\mathbf{Y}$*, and* $\mathbf{Z}$ *be parametrised spaces. If* $f : \mathbf{X} \to \mathbf{Y}$ *and* $g : \mathbf{Y} \to \mathbf{Z}$ *are computable in polynomial time, then their composition* $g \circ f : \mathbf{X} \to \mathbf{Z}$ *is also computable in polynomial time.*

Proof Let $\mu_{\mathbf{X}}, \mu_{\mathbf{Y}}$ and $\mu_{\mathbf{Z}}$ be the parameters of $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$. Let $M^?$ be a machine that computes a realiser G of g in time and with parameter blow-up bounded by P. Let $N^?$ be a machine that computes a realiser F of f in time and with blow-up bounded by Q. A machine ${}_M N^?$ for computing $G \circ F$ can be obtained by replacing each oracle call of the machine $M^?$ with a subroutine that carries out the operations that $N^?$ would perform. To estimate the time this machine takes to run on input $\mathbf{a}$ with oracle φ, first note that the steps the machine takes can be divided into the steps it takes when executing the commands from $M^?$ and the ones it takes when executing commands from $N^?$.

The number of steps that are taken while executing the commands from $M^?$ is the same as the number of steps that $M^?$ would take on input $N^\varphi = F(\varphi)$ and therefore bounded by $P(\mu_{\mathbf{Y}}(F(\varphi)), |\mathbf{a}|)$. By the second condition of Definition 2.2 we have $\mu_{\mathbf{Y}}(F(\varphi)) \leq Q(\mu_{\mathbf{X}}(\varphi), .)$. Therefore, by the monotonicity of second-order polynomials from Lemma 1.2 we have

$$\mathrm{time}_{M^?}(F(\varphi), \mathbf{a}) \leq P(Q(\mu_{\mathbf{X}}(\varphi), .), |\mathbf{a}|).$$

The number of steps ${}_M N^?$ takes with each execution of $N^?$ is bounded by $Q(\mu_{\mathbf{X}}(\varphi), |\mathbf{b}|)$, where $\mathbf{b}$ is the content of the tape that replaces the oracle query tape of $M^?$. Due to the limited time available to $M^?$ to write this query, we have $|\mathbf{b}| \leq P(Q(\mu_{\mathbf{X}}(\varphi), .), |\mathbf{a}|)$. Thus,

$$\mathrm{time}_{N^?}(\varphi, \mathbf{b}) \leq Q(\mu_{\mathbf{X}}(\varphi), P(Q(\mu_{\mathbf{X}}(\varphi), .), |\mathbf{a}|)).$$

The number of times the oracle is called in the computation of $M^?$ with oracle N^φ and on input $\mathbf{a}$ is also bounded by the time of steps the machine $M^?$ may take. Thus, a bound on the total number of steps that ${}_M N^?$ takes on input $\mathbf{a}$ with oracle φ can be obtained by multiplying the two time bounds above. This can be seen to be a second-order polynomial in $\mu_{\mathbf{X}}(\varphi)$ and $|\mathbf{a}|$ using the closure properties of second-order polynomials from Proposition 1.4.

Finally, to obtain the bound on the output parameter, note that

$$\mu_{\mathbf{Z}}({}_M N^\varphi)(n) = \mu_{\mathbf{Z}}(M^{F(\varphi)})(n) \leq P(\mu_{\mathbf{Y}}(N^\varphi), n) \leq P(Q(\mu_{\mathbf{X}}(\varphi), .), n).$$

This completes the proof that ${}_M N^?$ computes $G \circ F$ in polynomial time. Since $G \circ F$ is a realiser of $g \circ f$ it follows that $g \circ f$ is polytime computable. ∎

Theorem 2.4 shows that parametrised spaces form a category with polytime computable mappings as morphisms. It includes the closure of second-order polytime computable operators under composition as a special case. The proof of Theorem 2.4 is considerably more uniform than its statement: a polytime algorithm for computing $g \circ f$ is obtained by composing any two polytime algorithms for f and g in the natural way.

The rest of this section introduces some basic notions and constructions that are needed for reasoning about parametrised spaces throughout the paper. We use straightforward adaptations from the theory of represented spaces. The correctness of our choices is supported by category theory in the sense that they are the 'usual' ones in the category of parametrised spaces with polytime computable functions as morphisms.

Real complexity theory has a history of non-uniformity and as a result the point-wise complexity structure is often known in more detail than the uniform structure. This makes it desirable to be able to reason about points of parametrised spaces. We arrive at the following notion:

Definition 2.5 An element of a preparametrised space is called **computable in polynomial time** if it has a polytime computable name whose parameter is bounded by a polynomial.

Another way of thinking about a polytime computable point in a parametrised space is as a polytime computable map from the one-point space. Here, the one-point space is equipped with the unique total representation and the constant zero parameter and is the terminal object of the category of parametrised spaces. A polytime computable point is therefore what is referred to as 'global element' in category theory. We obtain the following corollary:

Corollary 2.6 *Polytime computable functions between parametrised spaces take polytime computable points to polytime computable points.*

The usual construction of the product of two represented spaces can be extended to define a product of parametrised spaces. Define the **pairing function** $\langle .,. \rangle : \mathcal{B} \times \mathcal{B} \to \mathcal{B}$ by

$$\langle \varphi, \psi \rangle(\mathbf{a}) := \begin{cases} \epsilon & \text{if } \mathbf{a} = \epsilon \\ \varphi(\mathbf{b}) & \text{if } \mathbf{a} = 0\mathbf{b} \\ \psi(\mathbf{b}) & \text{if } \mathbf{a} = 1\mathbf{b} \end{cases}$$

Definition 2.7 Let $\mathbf{X} = (X, \xi_{\mathbf{X}}, \mu_{\mathbf{X}})$ and $\mathbf{Y} = (Y, \xi_{\mathbf{Y}}, \mu_{\mathbf{Y}})$ be parametrised spaces. Equip the product $X \times Y$ with the representation

$$\xi_{\mathbf{X}\times\mathbf{Y}}(\varphi) = (x, y) \quad \Leftrightarrow \quad \exists \psi, \tilde{\psi} : \varphi = \langle \psi, \tilde{\psi} \rangle \text{ and } \xi_{\mathbf{X}}(\psi) = x \text{ and } \xi_{\mathbf{Y}}(\tilde{\psi}) = y,$$

i.e., a name of a pair is a pair of names of the components. Furthermore, equip this space with the parameter $\mu_{\mathbf{X}\times\mathbf{Y}}$ defined by

$$\mu_{\mathbf{X}\times\mathbf{Y}}(\langle \psi, \tilde{\psi} \rangle)(n) := \max\{\mu_{\mathbf{X}}(\psi)(n), \mu_{\mathbf{Y}}(\tilde{\psi})(n)\}$$

The triple $(X \times Y, \xi_{\mathbf{X}\times\mathbf{Y}}, \mu_{\mathbf{X}\times\mathbf{Y}})$ is denoted by $\mathbf{X} \times \mathbf{Y}$. It is straightforward to see that $\mathbf{X} \times \mathbf{Y}$ is indeed the product of $\mathbf{X}$ and $\mathbf{Y}$ in the category of parametrised spaces and polytime computable functions.

In the theory of representations the notion of reduction plays a central role. It generalises easily to parametrised representations:

Definition 2.8 Let X be a set and let (ξ_0, μ_0) and (ξ_1, μ_1) be parametrised representations of X. We say that (ξ_0, μ_0) is **polytime translatable** to (ξ_1, μ_1) if the map $(X, \xi_0, \mu_0) \to (X, \xi_1, \mu_1)$, $x \mapsto x$ is polytime computable. If (ξ_0, μ_0) is polytime translatable to (ξ_1, μ_1) and (ξ_1, μ_1) is polytime translatable to (ξ_0, μ_0), we say that (ξ_0, μ_0) and (ξ_1, μ_1) are **polytime equivalent**.

We chose the word "translatable" over the more common term "reducible" as this leads to less confusion about the direction of the translations. If a parametrised representation (ξ_0, μ_0) is polytime translatable to a parametrised representation (ξ_1, μ_1) we also say that (ξ_1, μ_1) **contains less information than** (ξ_0, μ_0). This is a slight abuse of language as "information content" is more appropriately measured by topological or computable translatability.

If (ξ_0, μ_0) and (ξ_1, μ_1) are polytime equivalent parametrised representations of X then the preparametrised spaces (X, ξ_0, μ_0) and (X, ξ_1, μ_1) are polytime isomorphic, the underlying map of the isomorphism being the identity on X. As usual, we call preparametrised spaces A and B **isomorphic** if there exist polytime computable maps $\alpha\colon A \to B$ and $\beta\colon B \to A$ with $\alpha \circ \beta = \mathrm{id}_B$ and $\beta \circ \alpha = \mathrm{id}_A$. Note that polytime isomorphic preparametrised spaces with the same underlying set have polytime equivalent parametrised representations up to renaming of elements. We prefer to state our results in terms of isomorphism of parametrised spaces rather than in terms of equivalence of parametrised representations, although all isomorphisms we construct in this paper will have the identity as underlying map.

2.1 A parametrised space of real numbers

Recall the interval representation $\xi_{\mathbb{R}_i}$ of the real numbers from Definition 1.9:

> A function $\varphi\colon \mathbb{N} \to \mathbb{ID}$ is a $\xi_{\mathbb{R}_i}$-name of $x \in \mathbb{R}$ if $(\varphi(n))_n$ is a nested sequence of intervals with $\{x\} = \bigcap_{n\in\mathbb{N}} \varphi(n)$.

Also recall that the represented space $\mathbb{R}_i = (\mathbb{R}, \xi_{\mathbb{R}_i})$ has very pathological complexity properties. This rules out the standard parameter for making this space into a parametrised space. It is possible to endow the space with a different parameter which yields a sensible complexity theory. For a real number x, let $\lceil x \rceil$ denote the least integer number bigger than or equal to x and let lb denote the binary logarithm function.

Definition 2.9 For a $\xi_{\mathbb{R}_i}$-name φ of $x \in \mathbb{R}$, define the parameter $\mu_{\mathbb{R}_i}$ as

$$\mu_{\mathbb{R}_i}(\varphi)(n) = \min\{N \mid \operatorname{diam}(\varphi(N)) \leq 2^{-n}\} + \lceil \operatorname{lb}(|x|+1)\rceil.$$

The **parametrised space of interval reals** is the triple

$$\mathbb{R}_i = (\mathbb{R}, \xi_{\mathbb{R}_i}, \mu_{\mathbb{R}_i})\,.$$

The parameter mainly encodes the rate of convergence of a sequence of intervals. Small parameter blowup for a realiser of a function $f\colon \mathbb{R}_i \to \mathbb{R}_i$ hence means that the rate of convergence of the output sequence is similar to the rate of convergence of the input sequence. It remains to show that this really defines a parametrised space, *i.e.*, that the parameter is well-defined on the domain of the representation and that the identity is polytime computable.

Proposition 2.10 *The space $\mathbb{R}_i$ is a parametrised space.*

Proof That the parameter is well-defined follows directly from the definitions.

A family of witnesses of the polytime computability of the identity on the space $\mathbb{R}_i$ can be specified as follows: For a fixed non-constant polynomial $p \in \mathbb{N}[X]$, let $M_p^?$ be the oracle machine that on input $n \in \mathbb{N}$ (as usual encoded in unary) and with oracle φ queries the oracle for the interval $[r \pm \varepsilon] := \varphi(n)$. If the interval is infinite, it returns the infinite interval. Otherwise it reads approximations r' and ε' to precision $2^{-p(n)-1}$ from initial segments of r and ε to compute numbers a_n and b_n such that a_n is the largest dyadic number with denominator $2^{p(n)}$ with $a_n < r' - \varepsilon'$ and b_n is the smallest dyadic number with denominator $2^{p(n)}$ with $b_n > r' + \varepsilon'$. It then returns the interval $[a_n, b_n]$. To see that each of the machines $M_p^?$ computes a witnesses of the polytime computability of the identity on $\mathbb{R}_i$, first note that it computes the identity: By construction we have $[r \pm \varepsilon] \subseteq [a_n, b_n]$. Also by construction, the resulting sequence of intervals is nested. Thus any of the intervals returned by M_p^φ contains the real number that φ encodes. To see that the diameter of the intervals still goes to zero let some $\delta > 0$ be given. Since φ is a name, there exists an N such that for all n bigger than N the corresponding ε is smaller than $\frac{\delta}{3}$. The polynomial p is non-constant and therefore $p(n) \geq n$ holds for all $n \neq 0$. Thus we may choose n so big that $|b_n - a_n| \leq 2^{p(n)+1} + 2\varepsilon \leq \delta$.

To obtain a polynomial bound on the running time of $M_p^?$, first note that the time that M_p^φ takes on input of length n can be bounded in terms of $p(n)$ and a bound on the absolute value of $\xi_{\mathbb{R}_i}(\varphi)$. Furthermore,

$$\mu_{\mathbb{R}_i}(M_p^\varphi)(n) = \min\left\{N \mid \operatorname{diam}(M_p^\varphi(m)) \leq 2^{-n}\right\}$$

and

$$\operatorname{diam}(M_p^\varphi)(m) \leq \operatorname{diam}(\varphi(m)) + 2^{-p(m)+1}.$$

This, together with $p(m) \geq m$ implies

$$\mu_{\mathbb{R}_i}(M_p^\varphi)(n) \leq \mu_{\mathbb{R}_i}(\varphi)(n) + n + 1.$$

Since the right hand side is a second-order polynomial, this proves that $M_p^?$ has polynomially bounded parameter blowup. ∎

Choosing a witness of polytime computability of the identity corresponds to restricting the maximal precision that may be present in the n^{th} component of a name. However, the way in which the precision is restricted is mostly arbitrary and it may be beneficial in practice to use different cut-off precisions in different computations.

We often indirectly use the polytime computability of the identity by assuming that the value of the parameter on all names of real numbers is linear. A machine that works correctly on names of linear parameter can be transformed into one that works correctly on all names by precomposing it with

one of the realisers from the previous proof, where the corresponding polynomial is chosen linear. In the following we use the big-o notation: For integer functions f and g say that $f \in \mathcal{O}(g)$ if there exists a constant C such that for all n we have $f(n) \leq Cg(n) + C$.

Proposition 2.11 ($\mathbb{R}_i \simeq \mathbb{R}_c$) *The space of Cauchy reals and the space of interval reals are polytime isomorphic as parametrised spaces.*

Proof First construct the translation from the Cauchy reals to the interval reals: Let $M^?$ return on oracle $\varphi \in \xi_{cR}^{-1}(x)$ and input $n \in \mathbb{N}$ the interval $[\varphi(n) \pm 2^{-n}]$. Then M^φ is a $\xi_{\mathbb{R}_i}$-name of x. To produce this result, the machine needs to make $\mathcal{O}(n)$ steps for copying n (which is given in unary) and $\mathcal{O}(|\varphi|(n) + n)$ steps produce the return value from n and the dyadic approximation returned by φ. It remains to check that the machine has appropriate parameter blowup. By definition of the parameter of the interval reals we have

$$\mu_{\mathbb{R}_i}(M^\varphi)(n) = \min\left\{N \in \mathbb{N} \mid \operatorname{diam}(M^\varphi(N)) \leq 2^{-n}\right\} + \lceil \operatorname{lb}(|x| + 1)\rceil.$$

By the construction of $M^?$ we have $\operatorname{diam}(M^\varphi(n-1)) = 2^{-n}$, and therefore the first summand in the above equation is always bounded by $n - 1$. The absolute value of x on the other hand is bounded by the size of the encoding of the dyadic number returned by φ. That is

$$\mu_{\mathbb{R}_i}(M^\varphi)(n) \leq n - 1 + \lceil \operatorname{lb}(|x| + 1)\rceil \leq n - 1 + |\varphi(0)| \leq n + |\varphi|(c),$$

where $c = 2$ is the length of the encoding of 0. This proves that $M^?$ computes a witness of polytime computability of the translation.

For the other direction first note that, by applying an appropriate witness of the polytime computability of the identity from Proposition 2.10, it may be assumed that the size of any $\xi_{\mathbb{R}_i}$-name of some $x \in [0,1]$ satisfies $|\varphi| \in \mathcal{O}(n)$. Define a machine $N^?$ that on such an oracle φ and on input n proceeds as follows: It searches for an m such that $\operatorname{diam}(\varphi(m)) \leq 2^{-n+1}$. This condition can be checked in time $\mathcal{O}(n)$ since the name is short. The search halts as soon as the value of m is equal to the first summand of the parameter $\mu_{\mathbb{R}_i}(\varphi)(n)$. Let the machine return the midpoint of the interval. Since all names are short and the encodings reasonable, obtaining the midpoint takes at most time $\mathcal{O}(n)$. By construction the machine $N^?$ does at most $\mu_{\mathbb{R}_i}(\varphi)(n)$ loops of a computation that takes $\mathcal{O}(n)$ steps to carry out and thus runs in polynomial time. Since the size of the return value is in $\mathcal{O}(n)$, the machine has polynomially bounded parameter blowup. ∎

Since polytime computable functions preserve polytime computable points by Corollary 2.6 we obtain:

Corollary 2.12 *A real number is polytime computable if and only if it is polytime computable as an element of the parametrised space $\mathbb{R}_i$.*

A final property to mention about the space $\mathbb{R}_i$ and a property that distinguishes it from the Cauchy reals and is therefore not preserved under isomorphism is that any polytime computable function on the interval reals can be computed by a machine whose running time is bounded by a first-order function.

Theorem 2.13 *Whenever $f :\subseteq \mathbb{R}_i \to \mathbb{R}_i$ is polytime computable, then f can be computed by an oracle machine $M^?$ such that there exists a $C \in \mathbb{N}$ and a second-order polynomial P satisfying*

$$\operatorname{time}_{M^?}(\varphi, \mathbf{a}) \leq C \cdot |\mathbf{a}| + C \quad \text{and} \quad \mu_i(M^\varphi)(n) \leq P(\mu_X(\varphi), n).$$

Proof Since f is polytime computable, there exists a machine $N^?$ and a second-order polynomial Q that bounds the running time and the parameter

blowup. Without loss of generality assume $Q(l,n) \geq n$. Let the machine $M^?$ with oracle φ and input $\mathbf{a}$ spend $|\mathbf{a}|$ steps on simulating what $N^?$ does on oracle φ and binary encodings of 2^i as input for i starting from zero and counting up. Return the return value of the machine $N^?$ on the biggest input where the simulation finished in time. In case none of the computations have terminated, return the infinite interval. Obviously, $M^?$ takes not more than $C \cdot |\mathbf{a}| + C$ steps, thus it is left to specify an appropriate polynomial P. For this, note that for a given n, since Q bounds the running time of $N^?$, choosing $|\mathbf{a}|$ bigger than

$$\sum_{i=0}^{n} Q(\mu_{\mathbb{R}_i}(\varphi), i) \leq nQ(\mu_{\mathbb{R}_i}(\varphi), n)$$

forces all the simulations of $N^?$ with oracle φ and on input smaller than 2^n to come to an end. Since the parameter blowup of $N^?$ is bounded by Q, the absolute value of the number encoded by N^φ is bounded by $Q(\mu_{\mathbb{R}_i}(\varphi), 0) + 1$ and choosing n bigger than $Q(\mu_{\mathbb{R}_i}(\varphi), m)$ forces the diameter of the returned interval to be smaller than 2^{-m}. This implies that on input $\mathbf{a}$ of size bigger than

$$Q(\mu_{\mathbb{R}_i}(\varphi), m) \cdot Q(\mu_{\mathbb{R}_i}(\varphi), Q(\mu_{\mathbb{R}_i}(\varphi), m))$$

the interval that $M^?$ returns has diameter smaller than 2^{-m} and that P can be picked as

$$P(l,n) := Q(l,n) \cdot Q(l, Q(l,n)) + Q(l,0) + 1.$$

That P is a second-order polynomial follows from the closure properties of second-order polynomials from Proposition 1.4. ∎

Note that the proof follows the construction which is used to show that the computational complexity of the interval representation is ill-behaved with respect to running time bounds in terms of the size function [Sch04, KP14a, Ste16]. However, in contrast to the size of a name, the parameter of a name contains meaningful information and as a consequence delaying the time until a meaningful output is produced leads to an increase of the parameter blow-up. That is, instead of removing computational cost, the construction trades time needed to produce approximations for a worse convergence behaviour. While the extent to which this is done in Theorem 2.13 may not be appropriate, this construction can be viewed as a means of separating the reasoning about an algorithm into two parts. The first part is the computation of approximations to the function value from approximations to the input. The complexity of these computations can be expressed using first-order bounds. The second part is the convergence analysis, which does require second-order bounds, as it relates the rate of convergence of the input sequence to the rate of convergence of the output sequence. This seems to be more in line with how the complexity of algorithms is studied in the real number model and related models. Most "natural" algorithms for computing functions $f\colon \mathbb{R}_i \to \mathbb{R}_i$ do indeed have first-order time bounds.

2.2 A parametrised space of continuous functions

Let $I = [0,1]$ denote the closed unit interval. Let I_i be the parametrised space that is obtained by considering the unit interval as a subspace of the parametrised space $\mathbb{R}_i$ of interval reals. By this we mean that the representation ξ_{I_i} of I_i is the range restriction of $\xi_{\mathbb{R}_i}$ to I and the parameter μ_{I_i} is the restriction of $\mu_{\mathbb{R}_i}$ to the domain of the representation.

Consider the space $C(I)$ of continuous functions from the unit interval to the real numbers. Having made $\mathbb{R}$ and I into parametrised spaces which are closely tied to the complexity of interval methods, it is natural to ask whether the function space $C(I)$ admits a similar structure.

Definition 2.14 Define the **interval function representation** ξ_{if} as follows: A function $\psi\colon \mathbb{ID} \to \mathbb{ID}$ is a name of $f \in C([0,1])$ if and only if

$$\forall \varphi \in \mathrm{dom}(\xi_{\mathbb{R}_i})\colon \big(\xi_{\mathbb{R}_i}(\varphi) \in [0,1] \quad \Rightarrow \quad \xi_{\mathbb{R}_i}(\psi \circ \varphi) = f(\xi_{\mathbb{R}_i}(\varphi))\big).$$

Note that if ψ is a name of a continuous function f, then ψ is necessarily monotone as an interval function, *i.e.*, $J \subseteq I$ implies that $\psi(J) \subseteq \psi(I)$. Unlike the Kawamura-Cook representation of continuous functions, which is recalled in Section 3, the present Definition 2.14 employs a canonical exponential construction to represent the function space. The restriction to compact intervals of reals is mainly necessary in order to ensure the well-definedness of the parameter, which essentially encodes a modulus of uniform continuity of the represented function:

Definition 2.15 Let the **parameter** $\mu_{if}\colon \mathrm{dom}\,(\xi_{if}) \to \mathbb{N}^{\mathbb{N}}$ be defined by

$$\begin{aligned}\mu_{if}(\psi)(n) &:= \lceil \mathrm{lb}(\|\xi_{if}(\psi)\|_\infty + 1)\rceil \\ &\quad + \min\left\{N \in \mathbb{N} \mid \forall J \in \mathbb{ID} : \mathrm{diam}(J) \le 2^{-N} \Rightarrow \mathrm{diam}(\psi(J)) \le 2^{-n}\right\},\end{aligned}$$

whenever ψ is a ξ_{if}-name of a function.

While the time taken by a polytime machine operating on parametrised space may depend on the parameter of the input name, the machine does not have direct access to the parameter and hence can in general not use it in its computation. The best way to compute a modulus of continuity from a name of a function in the interval function representation seems to be a search that may in the worst case take time exponential in the value of the parameter defined above. Indeed, in Section 3.2 we show that the modulus function cannot be computed in polynomial time with respect to the above parameter. This is a difference to the representation used by Kawamura and Cook, where a name comes with explicit information about the modulus of continuity.

Lemma 2.16 *The parameter μ_{if} is well-defined on* $\mathrm{dom}(\mu_{if})$.

Proof Assume that ψ is a name of a function. Our goal is to show that for every $n \in \mathbb{N}$ the minimum

$$\min\{N \in \mathbb{N} \mid \forall J \in \mathbb{ID}\colon\ \mathrm{diam}(J) \le 2^{-N} \Rightarrow \mathrm{diam}(\psi(J)) \le 2^{-n}\}$$

exists. For a fixed x denote the smallest dyadic number with denominator 2^n that is bigger than x by x_n. Then $\varphi_x(n) := [x_n \pm 2^{-n}]$ is a $\xi_{\mathbb{R}_i}$-name of x. Since we assumed ψ to be a name, it follows that $\psi(\varphi_x(n)) \to \{f(x)\}$ as $n \to \infty$. Hence, there exists $N_x \in \mathbb{N}$ with $\mathrm{diam}(\psi(\varphi_x(N_x)))) \le 2^{-n}$. The family $((x_{N_x} \pm 2^{-N_x}))_{x\in[0,1]}$ of interiors of the intervals $\varphi_x(N_x)$ is an open cover of $[0,1]$. Since the unit interval is compact, there exists a finite subcover. As a finite family of open intervals, this family has a smallest overlap. Let N be big enough such that 2^{-N} is smaller than this overlap. It follows that every interval J whose diameter is smaller than 2^{-N} is contained in an interval I from this finite family. By the monotonicity of ψ, for each such interval it follows that $\psi(J) \subseteq \psi(I)$ and since the diameter of $\psi(I)$ is smaller than 2^{-n}, the same holds for $\psi(J)$. ∎

Consider the space $C(I)_i = (C([0,1]), \xi_{if}, \mu_{if})$. We call this the **parametrised space of interval functions**. Just like the corresponding result for the interval reals, proving that the interval functions are a parametrised space mainly consists of the introduction of a rounding procedure, which includes the non-canonical choice of a polynomial that controls the cut-off precision.

Proposition 2.17 *The space $C(I)_i$ is a parametrised space.*

PROOF First note that, while copying the input interval to the oracle query tape is possible in polynomial time, it may not be possible to copy the answer to the output tape. This is because a name of a function may return intervals $[r \pm \varepsilon]$ such that ε is fairly big and r is fairly small but still r and ε have large encodings. The time the machine is granted, however, only increases with the diameter of the interval getting smaller or the midpoint getting bigger. Instead of copying r and ε directly, rounded versions of these numbers can be read from a beginning segment of the oracle answer. The details of this rounding procedure are given in Proposition 2.10: The machines $M_p^?$ introduced there make a single oracle query at the very beginning of the computation and may therefore instead be considered as machines that do not have an oracle and take an interval as input. Fix such a machine M_p. A witness of polytime computability for the identity is computed by the machine which maps a given name φ of a function to the composition of the machine M_p and φ. ∎

Consider the evaluation operator defined as follows:

$$\mathrm{eval} : C([0,1]) \times [0,1] \to \mathbb{R}, \quad (f,x) \mapsto f(x).$$

A sanity check for the definition of $C([0,1])_i$ is that evaluation should be polytime computable. For this statement to make sense, it is necessary to use the product of parametrised spaces given at the end of the introduction of Section 2.

Proposition 2.18 (Evaluation) *Evaluation as operator from $C(I)_i \times I_i$ to $\mathbb{R}_i$ is polytime computable.*

PROOF Consider the machine $M^?$ that when given oracle $\langle \psi, \varphi \rangle$ with $\xi_{if}(\psi) = f$ and $\xi_{\mathbb{R}_i}(\varphi) = x$, such that φ has linear size returns a string function of linear size obtained from $\psi(\varphi(n))$ by applying an appropriate realiser of the identity constructed in Proposition 2.10. This machine computes a realiser of the evaluation operator by the definition of the representation ξ_{if} from Definition 2.14. Due to the appropriate truncations, the time the machine takes is bounded polynomially. To see that the machine does not inflate the parameter too much, note that

$$\begin{aligned}\mu_{\mathbb{R}_i}(M^{\langle\psi,\varphi\rangle})(n) &= \min\left\{N \mid \forall m \geq N \colon \psi(\varphi(m)) \leq 2^{-n+1}\right\} + \lceil \mathrm{lb}(|f(x)|+1)\rceil \\ &\leq \mu_{if}(\psi)(\mu_{\mathbb{R}_i}(\varphi)(n)) \\ &\leq (\mu_{if} \times \mu_{\mathbb{R}_i})(\langle\psi,\varphi\rangle)\,((\mu_{if} \times \mu_{\mathbb{R}_i})(\langle\psi,\varphi\rangle)(n))\,.\end{aligned}$$

Thus $M^?$ computes a witness of the polytime computability of the evaluation operator. ∎

Recall the notion of polytime translatability from Definition 2.8. The parametrised representation (ξ_{if}, μ_{if}) is the "correct" representation for $C(I)$, viewed as a function space, as it contains the least amount of information (in the sense of Definition 2.8) among those representations which render evaluation polytime computable.

Theorem 2.19 (Minimality) *For a parametrised representation (ξ, μ) of $C([0,1])$ the following are equivalent:*

1. *Evaluation is polytime computable with respect to (ξ, μ).*
2. *The pair (ξ, μ) is polytime translatable to (ξ_{if}, μ_{if}).*

PROOF The implication *2.⇒1.* directly follows from the closure of the polytime computable operators under composition from Theorem 2.4 and the polytime computability of the evaluation operator on the interval functions from Proposition 2.18.

For the other implication assume that the evaluation operator is polytime computable with respect to (ξ, μ). Note that due to the equivalence of the

Cauchy reals and the interval reals from Proposition 2.11, the evaluation operator is also polytime computable if $[0,1]$ and $\mathbb{R}$ are equipped with the Cauchy representation. Let $M^?$ be a machine that computes evaluation with time consumption and parameter blowup bounded by a second-order polynomial P.

Define a machine $N^?$ that computes a translation of (ξ, μ) into (ξ_{if}, μ_{if}) in polytime as follows: Fix some ξ-name ψ of some function $f \in C([0,1])$ as oracle and an interval $[r \pm \varepsilon]$ with $r \in [0,1] \cap \mathbb{D}$ as input. Let n be the largest natural number such that $\varepsilon \leq 2^{-n}$ (if the input interval has diameter bigger than one, return the interval $[0 \pm \infty]$). Let r_m be r rounded to precision 2^{-m}. The machine $N^?$ follows the steps that $M^?$ would take on oracle $\langle \psi, m \mapsto r_m \rangle$ and inputs i going from n to zero. In each of the runs it saves the maximal query k that is posed to the oracle $m \mapsto r_m$ and when the computation on i ends, it compares k to n. If k is bigger than n the machine decreases i and starts over, unless i is already zero, in which case it returns $[0 \pm \infty]$. If k is smaller than n, and $M^?$ returns d, then let $N^?$ return the interval $[d \pm 2^{-i}]$.

To see that this produces a ξ_{if}-name of f from the ξ-name ψ, let P be the polynomial time-bound of $M^?$. Let $([r_s \pm \varepsilon_s])_{s \in \mathbb{N}}$ be a sequence of intervals that converge to a point $x \in [0,1]$. Let $([d_s \pm \delta_s])_{s \in \mathbb{N}}$ be the sequence of intervals which are returned by $N^?$ on input $([r_s \pm \varepsilon_s])_{s \in \mathbb{N}}$. We may assume without loss of generality that $([d_s \pm \delta_s])_{s \in \mathbb{N}}$ is a nested sequence. First note that each of the intervals $[d_s \pm \delta_s]$ contains $f(x)$, so that it remains to show that the sequence $([d_s \pm \delta_s])_{s \in \mathbb{N}}$ converges to a point. There exists an integer constant K such that for any $r \in [0,1] \cap \mathbb{D}$ the function $m \mapsto r_m$ has size smaller than $K \cdot (m+1)$. Thus, the number of steps of $N^?$ in any of the simulations of $M^?$ and in particular the value of k in each such simulation is bounded by

$$k_n := P(m \mapsto \max\{\mu(\psi)(m), K \cdot (m+1)\}, n). \tag{k}$$

Since this value is independent of r, the computation on all intervals of diameter smaller or equal 2^{-k_n} results in return values of diameter smaller than 2^{-n}. In particular the sequence $([d_s \pm \delta_s])_{s \in \mathbb{N}}$ converges to a point.

Finally, the machine $N^?$ runs in polytime: To bound the number of steps $N^?$ takes by a second-order polynomial in the length of the input and the parameter of the oracle note that the number of steps taken in each simulation of $M^?$ is bounded by k_n as above and that at most n of these simulations need to be carried out. Let us now show that $N^?$ has polynomial parameter blowup. We have

$$\begin{aligned}\mu_{\mathbb{R}_i}(N^\psi)(n) &= \lceil \mathrm{lb}\left(\|f\|_\infty + 1\right) \rceil \\ &\quad + \min\left\{ m \in \mathbb{N} \mid \forall J \in \mathbb{ID} : \mathrm{diam}(J) \leq 2^{-m} \Rightarrow \mathrm{diam}(N^\psi(J)) \leq 2^{-n} \right\} \\ &\leq \lceil \mathrm{lb}\left(\|f\|_\infty + 1\right) \rceil + k_n.\end{aligned}$$

Since k_n is polynomially bounded in $\mu(\psi)$ by (k), it remains to provide a bound on the supremum norm of the function f. Cover $[0,1]$ with finitely many intervals of the form $[r \pm 2^{-k_0}]$ where r is a dyadic rational number with $\mathcal{O}(k_0)$ bits. When these are fed into the machine N^ψ, it will produce approximations to the range of f over these intervals to error 1. Hence a bound on the output size of the machine over these intervals yields a bound on the supremum norm of f. By our previous considerations the running time (and hence the output size) of the machine on each interval is bounded by

$$\begin{aligned}k_{k_0} &= P(\max\{\mu(\psi), m \mapsto K \cdot (m+1)\}, k_0) \\ &= P(\max\{\mu(\psi), m \mapsto K \cdot (m+1)\}, P(\max\{\mu(\psi), m \mapsto K \cdot (m+1)\}, 0))\end{aligned}$$

so that the supremum norm of f is bounded polynomially in $\mu(\psi)$. ∎

As is usually the case for minimality results of this kind, the proof generalizes to the slightly stronger statement that $(C(I)_i, \mathrm{eval})$ is an exponential object in the category of parametrised spaces. More explicitly:

Corollary 2.20 *For any parametrised space* $\mathbf{Z}$ *and any polytime computable* $F : \mathbf{Z} \times I_i \to \mathbb{R}_i$ *there exists a unique polytime computable mapping* $c(F) : \mathbf{Z} \to C(I)_i$ *such that for all* $z \in \mathbf{Z}$ *and* $x \in I$ *we have* $c(F)(z)(x) = F(z, x)$.

Together with the equivalence of the Cauchy and the interval reals from Proposition 2.11 we obtain the following result:

Corollary 2.21 *The following are equivalent for a function* $f\colon [0,1] \to \mathbb{R}$*:*

1. f *is computable in polynomial time with respect to the Cauchy representation of* $[0,1]$ *and* $\mathbb{R}$.
2. f *is computable in polynomial time with respect to the parametrised interval representation of* $[0,1]$ *and* $\mathbb{R}$.
3. f *is a polytime computable point of the parametrised space* $C(I)_i$.

Corollary 2.21 in particular shows that for every polytime computable function $f\colon [0,1]_i \to \mathbb{R}_i$ there exists a polytime computable $\psi\colon \mathbb{ID} \to \mathbb{ID}$ with $\psi \circ \varphi = f(\xi_{\mathbb{R}_i}(\varphi))$ for all $\xi_{\mathbb{R}_i}$-names φ. In other words, every polytime computable function is computed in polynomial time by an interval algorithm.

Finally, consider the composition of functions, namely the operator

$$\circ\colon C([0,1]) \times C([0,1],[0,1]) \to C([0,1]), \quad (f,g) \mapsto f \circ g, \qquad \text{(comp)}$$

where $(f \circ g)(x) := f(g(x))$. Here $C([0,1],[0,1])$ is the set of all continuous functions on the unit interval whose image is contained in the unit interval. It makes sense to ask about the polytime computability of this operator whenever $C([0,1])$ is given the structure of a parametrised space, as it is possible to consider $C([0,1],[0,1])$ to be equipped with the range restriction of the representation of $C([0,1])$ and the restriction of the parameter to the domain of the range restriction.

Theorem 2.22 (Operations) *The arithmetic operations and composition are polytime computable on the interval functions.*

Proof Witnesses of the polynomial-time computability of the arithmetic operations can easily be written down by doing interval arithmetic on the return values of the names of two functions.

A realiser of the composition operator is the composition operator on Baire space. It remains to check, that the restriction of this operator to the domain of ξ_{if} is a witness of the polytime computability of the composition of functions in the sense of Definition 2.2. This can easily be checked by verifying that $\mu_{if}(\varphi \circ \psi) \leq \mu_{if}(\varphi) \circ \mu_{if}(\psi)$ for names of elements from the domain of the composition operator. ∎

The polytime computability of the arithmetic operations may also be deduced from the minimality of the interval-function representation and can in the process be generalised considerably.

Proposition 2.23 *Let* $H\colon \mathbb{R}_i \times \mathbb{R}_i \to \mathbb{R}_i$ *be a polytime computable function. Then the map*

$$C(I)_i \times C(I)_i \to C(I)_i,\ (f,g) \mapsto (x \mapsto H(f(x), g(x)))$$

is polytime computable.

Proof Consider the parametrised space $\mathbf{Z} := C(I)_i \times C(I)_i$ and the mapping

$$F\colon \mathbf{Z} \times I_i \to \mathbb{R}_i,\ (f,g,x) \mapsto H(f(x), g(x)).$$

As a composition of the polytime computable evaluation on $C(I)_i$ and H, the function F is polytime computable. Use the currying property from Corollary 2.20 to obtain polytime computability of the function

$$c(F)\colon \mathbf{Z} \to C(I)_i, \quad c(F)(f,g)(x) = F(f,g,x).$$

Note that $c(F)$ coincides with the function whose polytime computability was to be proven. ∎

In the above, $\mathbb{R}_i$ could have been replaced by the Cauchy reals without changing the class of polytime computable functions on $\mathbb{R}$ or $\mathbb{R} \times \mathbb{R}$. This class is the same as the one Ko introduces in his book [Ko91]. It should be noted that functions on unbounded domains are rarely considered complexity theoretically. In particular Ko seldom uses said definition in his book.

3 Comparison to Kawamura and Cook

The most used and best developed framework for complexity considerations in computable analysis is the framework of Kawamura and Cook. This framework is based on second-order complexity theory as presented in Section 1.1. However, Kawamura and Cook add the following assumption about the elements that are allowed as names:

Definition 3.1 A string function $\varphi : \Sigma^* \to \Sigma^*$ is called **length-monotone** if for all strings $\mathbf{a}$ and $\mathbf{b}$ we have

$$|\mathbf{a}| \leq |\mathbf{b}| \quad \Rightarrow \quad |\varphi(\mathbf{a})| \leq |\varphi(\mathbf{b})|\,.$$

The set of all length-monotone string functions is denoted by LM.

Note that length-monotone string functions map strings of equal length to strings of equal length and we have $|\varphi|\,(n) = |\varphi(\mathtt{0}^n)|$.

Definition 3.2 A representation is called **length-monotone** if its domain is contained in LM. A parametrised space with a length-monotone representation and the standard parameter is called a **Kawamura-Cook space**.

The notion of polytime computability of functions between Kawamura-Cook spaces as parametrised spaces coincides with the definition given by Kawamura and Cook for represented spaces equipped with a length-monotone representation.

The names in most representations considered in this paper are functions which return dyadic numbers. As any dyadic number has encodings of arbitrarily large size, we can pad an arbitrary name of an element to a length-monotone one. Thus, in many cases, representations can be restricted to LM to obtain a Kawamura-Cook space from a represented space. Depending on the concrete example, this may or may not change the complexity structure. The restriction of the Cauchy representation of reals as introduced in Definition 1.8 to length-monotone names leads to a polytime equivalent representation. An example where the restriction to length-monotone names does make sense but is not polytime equivalent is the hyper-linear representation considered in the upcoming Section 3.1.

The main motivation for considering Kawamura-Cook spaces is that in these spaces the size of an oracle is accessible to a polytime machine. It is possible to prove that this characterizes the Kawamura-Cook spaces within the category of parametrised spaces up to isomorphism. Recall that this paper exclusively uses the unary encoding for natural numbers. Consider the following structure of a parametrised space on the space of possible sizes of oracles:

Definition 3.3 Equip the space $\mathbb{N}^{\mathbb{N}}$ with the total representation defined by

$$\xi(\varphi)(n) = |\varphi(\mathtt{1}^n)|$$

and the standard parameter.

The space $(\mathbb{N}^{\mathbb{N}}, \xi, \mu)$ is a parametrised space: The identity can be computed by the machine that checks if the input is of the form $\mathtt{1}^n$, aborts if it is not and otherwise copies φ.

Lemma 3.4 *Every second-order polynomial is polytime computable as a mapping from* $\mathbb{N}^{\mathbb{N}} \times \mathbb{N}$ *to* $\mathbb{N}$.

The proof is a straightforward induction on the structure of second-order polynomials. This does not contradict the failure of time-constructibility of second-order polynomials, as the difficult part, namely evaluating the size function of an oracle, has been skipped.

Now the informal statement 'the size of an oracle is accessible' can be replaced with the formal statement 'the operation of sending an oracle to its size is polytime computable' and the promised characterisation of the Kawamura-Cook spaces follows.

Theorem 3.5 *A parametrised space is polytime isomorphic to a Kawamura-Cook space if and only if it is polytime isomorphic to a space with polytime computable parameter.*

Proof For the first direction assume that the parametrised space (X, ξ, μ) has a polytime computable parameter. Let $N^?$ be a machine that computes the identity, running time and parameter blowup being bounded by a second-order polynomial P. Define a representation δ of X as follows: A string-function φ is a δ-name of $x \in X$ if and only if there exists a ξ-name ψ of x such that

$$\varphi(\mathbf{a}) = N^{\psi}(\mathbf{a})\mathtt{01}^{P(\mu(\psi),|\mathbf{a}|)-\left|N^{\psi}(\mathbf{a})\right|}.$$

Note that since P bounds the running time of $N^?$, the exponent above is always positive and we have $|\varphi(\mathbf{a})| = P(\mu(\psi), |\mathbf{a}|) + 1$. In particular each such φ, and therefore also δ, is length-monotone. Since the parameter is polytime computable and second-order polynomials can be evaluated in polynomial time, the obvious translation from ξ to δ runs in polynomial time. A translation in the other direction can be computed by truncating the tail. The bounded parameter blowup of both of these translations follows from the equality $|\varphi|(n) = P(\mu(\psi), n) + 1$.

For the other direction recall that the standard parameter is a restriction of the size function and due to the representation being length-monotone the restriction of the size function to the domain of the representation is polytime computable. ∎

3.1 Representations of $C([0,1])$

Within their framework of second-order complexity theory, Kawamura and Cook have introduced a universal representation of univariate continuous functions based on the following classical characterisation of polytime computable real functions, see e.g. [Ko91, Corollary 2.14]:

Theorem 3.6 *A real function* $f\colon [0,1] \to \mathbb{R}$ *is polytime computable if and only if the sequence* $(f(d))_{d\in\mathbb{D}\cap[0,1]}$ *is a polytime computable sequence of reals and* f *has a polynomial modulus of continuity.*

Recall that a function $\nu : \mathbb{N} \to \mathbb{N}$ is called a **modulus of continuity** of $f \in C([0,1])$ if we have

$$\forall x, y \in [0,1], \forall n \in \mathbb{N}\colon\ |x - y| \le 2^{-\nu(n)} \Rightarrow |f(x) - f(y)| \le 2^{-n}.$$

Definition 3.7 ([KC12]) Define the **Kawamura-Cook representation** $\delta_\square$ **of** $C([0,1])$ as follows: A length-monotone function $\varphi : \mathbb{D} \times \mathbb{N} \to \mathbb{D}$ is a name of $f \in C([0,1])$ if it satisfies

$$\forall r \in \mathbb{D}, \forall n \in \mathbb{N}\colon\ |\varphi(r,n) - f(r)| \le 2^{-n}$$

and $|\varphi|$ is a modulus of continuity of f.

Note that it is possible to require the names to be length-monotone and to use the size of a name for encoding the modulus since encodings of dyadic numbers can be chosen arbitrarily large. Let $C(I)_{KC}$ be the parametrised space induced by the representation $\delta_\square$ and the standard parameter. This space is obviously a Kawamura-Cook space. Its representation is universal in the sense that within the class of Kawamura-Cook spaces, it provides the minimal amount of information about a continuous function such that evaluation is possible in polynomial time. (For details see [KC12], in particular Lemma 4.9.) Recall that 'evaluation' refers to the functional

$$\text{eval} : C([0,1]) \times [0,1] \to \mathbb{R}, \quad (f,x) \mapsto f(x). \tag{eval}$$

Kawamura and Cook's proof of minimality equips $[0,1]$ and $\mathbb{R}$ with the length-monotone Cauchy representation and the standard parameter. Due to the polytime equivalence of the Cauchy and the interval reals from Proposition 2.11, this does not make a difference up to polytime equivalence.

Theorem 3.8 (Minimality [KC12]) *For a length-monotone representation δ the following are equivalent:*

- *Evaluation* (eval) *is polytime computable.*
- *The representation δ is polytime translatable to $\delta_\square$.*

In particular, evaluation is polytime computable in the Kawamura-Cook representation. The above uses the concept of polytime computability used by Kawamura and Cook which is equivalent to having a polytime computable realiser. The notion is also equivalent to polytime computability if the spaces are turned into Kawamura-Cook spaces using the standard parameter. This means that the polytime translatability coincides with polytime reducibility as Kawamura and Cook consider it for length-monotone representations.

Theorem 3.8 can also be stated in a weaker form that avoids mentioning length-monotonicity. Consider the multi-valued function which sends a real function to some modulus of continuity:

$$\text{mod} : C([0,1]) \rightrightarrows \mathbb{N}^\mathbb{N}, \quad f \mapsto \left\{ \nu \in \mathbb{N}^\mathbb{N} \mid \nu \text{ is mod. of cont. of } f \right\}. \tag{mod}$$

We call this function the **modulus-function** and it has to be multivalued to be computable: Since $C([0,1])$ is connected and $\mathbb{N}^\mathbb{N}$ is totally disconnected, the only single-valued continuous functions from $C([0,1])$ to $\mathbb{N}^\mathbb{N}$ are constant functions.

Theorem 3.9 (Minimality. General version) *For a representation ξ of $C([0,1])$ the following are equivalent:*

1. *Evaluation* (eval) *and modulus* (mod) *are polytime computable.*
2. *The representation ξ is polytime translatable to $\delta_\square$.*

A representation that can be regarded as an intermediate of the interval function representation and the Kawamura-Cook representation was introduced and investigated by Brauße and Steinberg.

Definition 3.10 ([BS17]) The **hyper-linear representation ξ_{hlin} of** $C([0,1])$ is defined as follows: A function $\varphi : \mathbb{D}\times\mathbb{N} \to \mathbb{D}\times\mathbb{N}$, $\varphi(r,n) = (\varphi_1(r,n), \varphi_2(r,n))$ is a name of a function $f \in C([0,1])$ if

- $f([r \pm 2^{-\varphi_2(r,n)}]) \subseteq [\varphi_1(r,n) \pm 2^{-n}]$.
- $\forall r \in \mathbb{D}, \forall n \in \mathbb{N}\colon \varphi_2(r,n) \leq |\varphi|\,(n)$.

Denote the parametrised space that arises by equipping $C([0,1])$ with the hyper-linear representation and the standard parameter by $C(I)_{hlin}$.

It is possible to restrict the hyper-linear representation to length-monotone names and the resulting length-monotone representation is polytime equivalent to the Kawamura-Cook representation $\delta_\square$. In [BS17] it is proven that the hyper-linear representation itself is not polytime equivalent to any length-monotone representation. This is an example where the restriction to length-monotone names is possible but changes the complexity structure.

	$C(I)_{KC}$	$C(I)_{hlin}$	$C(I)_i$
mod	polytime	not p.t.	not p.t.
eval, $+, \times$	polytime	polytime	polytime
$\circ$	polytime	not p.t.	polytime

Figure 4: an overview over the complexity of operations with respect to different structures of $C([0,1])$ as represented or parametrised spaces.

3.2 Comparison

Braußе and Steinberg prove the following properties of the hyper-linear representation that are relevant for this paper:

Theorem 3.11 ([BS17])

1. *The representation ξ_{hlin} is polytime translatable to $\delta_\square$ but no polytime translation in the other direction exists.*
2. *The modulus function is not polytime computable on $C(I)_{hlin}$.*
3. *Composition is not polytime computable on $C(I)_{hlin}$.*

They also provide a proof that the hyper-linear representation is the least representation such that hyper-linear-time evaluation is possible [BS17]. The exact formulation of this result is rather technical and of no importance to the contents of this paper. It should be mentioned, however, that the restriction to hyper-linear-time is necessary and technical difficulties are due to hyper-linear-time computability not working well as an replacement of polytime computability. Therefore the minimality property is slightly unsatisfactory and this, together with the failure of polytime computability of the composition, indicates that a different approach is necessary.

The Kawamura-Cook representation is polytime translatable to the hyper-linear representation, which is polytime translatable to the interval function representation and none of these reductions reverse. In symbols, indicating the order by information content as discussed in Definition 2.8 this can be written as

$$C(I)_i <_P C(I)_{hlin} <_P C(I)_{KC}.$$

Only the left inequality remains to be proven.

Theorem 3.12 *The hyper-linear representation can be translated to interval function representation in polynomial time. No polytime translation in the other direction exists.*

Proof In [BS17, Theorem 2.4] it is proven that the hyper-linear representation renders evaluation polytime computable. By the minimality of the interval functions with respect to polytime evaluation from Theorem 2.19 it follows, that the functions in the hyper-linear representation can be translated to the interval functions in polynomial time.

On the other hand, composition is polytime computable on the interval functions, and is not polytime computable with respect to the hyper-linear representation [BS17, Theorem 2.6]. Thus, no translation in the other direction can exist. ∎

Since the Kawamura-Cook representation of continuous functions seems to be of more relevance than the hyper-linear representation we state the following corollary separately:

Corollary 3.13 *The interval function representation cannot be translated to the Kawamura-Cook representation in polynomial time. A polytime translation in the other direction does exists.*

In [BS17] it is proven that the modulus function is not polytime computable with respect to the hyper-linear representation. This has the following important implication for the parametrised space of interval functions:

Corollary 3.14 *The modulus function is not polytime computable on the parametrised space of interval functions.*

Proof In [BS17, Theorem 2.6] it is proven, that the modulus function is not polytime computable with respect to ξ_{hlin}. Since ξ_{hlin} is polytime translatable to (ξ_{if}, μ_{if}), the same must hold for the parametrised space of interval functions. ■

Thus, while the polynomial computable points of $C(I)_i$ and $C(I)_{KC}$ are the same by Corollary Corollary 2.21, they do not behave equivalently in terms of uniform polytime computability, at least for multi-valued functions. Since the Kawamura-Cook space of continuous functions is universal amongst the Kawamura-Cook spaces which admit polytime evaluation, it follows that $C(I)_i$ is not isomorphic to any Kawamura-Cook space.

An overview of the properties of the three spaces considered in this section can be found in Figure 4. These properties show that the function space $C(I)_i$, unlike function spaces in previously considered models, reflects the empirically observable properties of `iRRAM`-functions: Evaluation and composition are easy to compute, while the modulus of continuity is hard to compute. This is no coincidence: The Appendix discusses how the function space used in `iRRAM` can be modelled as a parametrised space which is polynomial-time isomorphic to $C(I)_i$.

4 Conclusion

Kapron and Cook's characterisation of second-order polytime computability by means of resource bounded oracle machines came as quite a surprise. The basic feasible functionals satisfy the so-called Ritchie-Cobham property: the running time of an oracle machine which computes a basic feasible functional can be bounded by a basic feasible functional [Meh76]. However, while the size of return values of a basic feasible functional are always bounded by a second-order polynomial, for most second-order polynomials P there is no basic feasible functional T which satisfies

$$\forall \varphi \in \mathcal{B}, \mathbf{a} \in \Sigma^* \colon |T(\varphi, \mathbf{a})| \geq P(|\varphi|, |\mathbf{a}|). \tag{1}$$

This seems to suggest that the class of functions which can be computed with a second-order polynomial time bound is strictly larger than the class of basic feasible functionals. The reason for the characterisation to still go through is that the oracle access of oracle machines is very restricted already by second-order polynomial time bounds, prohibiting detection of big inputs.

Using the characterisation as a definition, the proof of the Ritchie-Cobham property makes essential use of the totality of the functionals at hand. While totality is a standard assumption in second-order complexity theory, applications in computable analysis require a very specific kind of partiality. From this point of view, the failure of Equation (1) may be considered pathological. Adding length-monotonicity of names as an assumption removes this pathological behaviour, as the restriction of the size function to the length-monotone functions is polytime computable. Furthermore, it seems to be a necessary restriction for the proof that Kawamura and Cook provide that their representation of the continuous functions on the unit interval has the minimality property.

We believe that the framework of Kawamura and Cook is the best solution that allows for formulation within second-order complexity theory in its

traditional form. The extension proposed in this paper does not allow formulation within the scope of second-order complexity in the sense that whether an algorithm runs in polynomial time is allowed to depend on properties of its inputs, namely the parameter, that need not make sense independently of the interpretation of the inputs. This is the reason for the use of the term 'parameter' that indicates that semantic information about the objects has to be taken into account in addition to the size of a name. We believe this step to be necessary to allow for the description of the behaviour of efficient software based on computable analysis. Previous attempts to recover a description within second-order complexity theory have run into serious problems that where due to specific properties of the size function and the necessity to modify it on the output side [BS17].

It is not very difficult to produce parametrised spaces that are likely to be of importance and not isomorphic to any space that uses the standard parameter. Sierpinski space with the standard representation and the parameter that assigns the position of the first non-zero value to a name of $\top$ and zero to the unique name of $\bot$ is an example of such a space. We conjecture that the parametrised space of interval functions is another example. A reason for this being a tricky question could be that the running time of the parameter of the interval functions and the running time of the size function on Baire spaces are fairly similar. An affirmative answer to this question would provide an even stronger motivation for the introduction of parametrised spaces.

While we started the investigation of parametrised spaces out of what we believe to be necessity, we have found them to provide a very natural framework for complexity considerations in computable analysis. We believe that parametrised spaces broaden the scope of real complexity theory considerably and we intend to find new applications and explore the limits of the method. We expect that additional insight can already be gained by re-evaluating existing work in this generalised setting. The rest of this conclusion lists some ideas we consider especially promising and plan to pursue in the future.

A special case of the following is discussed in Theorem 2.13: All of [KP14a], [Sch04] and [Ste16] use very similar constructions to produce certain kinds of counterexamples. They give general constructions which make an arbitrary given representation into one with highly pathological properties. In the category of parametrised spaces these constructions can be modified so as to remove their pathological behaviour: If, in addition to modifying the representation, one modifies the parameter appropriately, one obtains a polytime isomorphic parametrised space. This space has the desirable property that polytime computability can be characterised using first-order time bounds in the running time constraint. This may be desirable as it allows to separate an algorithm into two parts: Firstly, a complexity-theoretical part that is mostly about efficiently manipulating approximations and may stay in a first-order setting. Secondly, a mathematical part that is about providing bounds on rates of convergence where the second-order nature of the problem at hand can no longer be ignored. A detailed description of this will be part of future work. For now we have to point to Theorem 2.13 for special case of this.

The present work proves that the interval reals are polytime equivalent to the Cauchy reals. However, we suspect that it is no coincidence that interval representations are favoured over the Cauchy representation in practice. In practice, memory consumption is often significantly more important than running time. Libraries like `iRRAM` are developed with this in mind and value space efficiency over time efficiency. It is therefore desirable to settle the question whether the interval reals and the Cauchy reals are also logarithmic space equivalent. Unfortunately, the tools to investigate this question are not available to us at this point in time. There is work about space restricted computation in the framework of Kawamura and Cook that one can check results about parametrised spaces against [KO14] and we are highly interested in pursuing this line of research in the future.

Recently there has been some interest in the study of exponentiable objects in the category of represented spaces with polytime computable functions as morphisms [KP14a, FH13]. We hope that the study of parametrised spaces can shed some light on this, as it seems like parametrised spaces allow for more natural exponential constructions than Kawamura-Cook spaces. For instance, our parametrised space of interval functions (Definition 2.14) enriches a natural exponential representation with a suitable parameter, whilst the Kawamura-Cook representation of $C(I)$ requires the hard-coding of a modulus of uniform continuity. It would be interesting to investigate to which extent this construction can be generalised. Exponentials in the category of parametrised spaces should yield exponentials in the category of Kawamura-Cook spaces by enriching the representation with bounds on the parameter.

References

[BBC+08] Luca Benvenuti, Davide Bresolin, Alberto Casagrande, Pieter Collins, Alberto Ferrari, Emanuele Mazzi, Alberto Sangiovanni-Vincentelli, and Tiziano Villa. Reachability computation for hybrid systems with Ariadne. *IFAC Proceedings Volumes*, 41(2):8960 – 8965, 2008. 17th IFAC World Congress.

[BCSS98] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and Real Computation*. Springer, 1998.

[BS17] Franz Brauße and Florian Steinberg. A minimal representation for continuous functions. `https://arxiv.org/abs/1703.10044`, 2017. preprint.

[BSS89] Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: NP- completeness, recursive functions and universal machines. *Bull. Amer. Math. Soc.*, 21(1):1 – 46, 1989.

[Coo92] Stephen A. Cook. Computability and complexity of higher type functions. In Yiannis N. Moschovakis, editor, *Logic from Computer Science: Proceedings of a Workshop held November 13–17, 1989*, pages 51–72. Springer New York, New York, NY, 1992. `doi:10.1007/978-1-4612-2822-6_3`.

[FGH14] Hugo Férée, Walid Gomaa, and Mathieu Hoyrup. Analytical properties of resource-bounded real functionals. *J. Complexity*, 30(5):647–671, 2014. `doi:10.1016/j.jco.2014.02.008`.

[FH13] Hugo Férée and Mathieu Hoyrup. Higher order complexity in analysis. preprint, extended abstract presented at CCA conference 2013, 2013. URL: `https://hal.inria.fr/hal-00915973/document`.

[Fri84] Harvey Friedman. On the computational complexity of maximization and integration. *Advances in Math.*, 53:80–98, 1984.

[FZ15] Hugo Férée and Martin Ziegler. On the computational complexity of positive linear functionals on C[0;1], 2015. MACIS conference. URL: `https://hugo.feree.fr/macis2015.pdf`.

[Grz55a] Andrzej Grzegorczyk. Computable functionals. *Fundamenta Mathematicae*, 42:168–202, 1955.

[Grz55b] Andrzej Grzegorczyk. On the definition of computable functionals. *Fundamenta Mathematicae*, 42:232–239, 1955.

[Grz57] Andrzej Grzegorczyk. On the definitions of computable real continuous functions. *Fundamenta Mathematicae*, 44(1):61–71, 1957.

[IRK01] Robert J. Irwin, James S. Royer, and Bruce M. Kapron. On characterizations of the basic feasible functionals, part I. *Journal of Functional Programming*, 11(1):117153, 2001.

[Kaw10] Akitoshi Kawamura. Lipschitz continuous ordinary differential equations are polynomial-space complete. *Computational Complexity*, 19(2):305 – 332, 2010.

[KC96] Bruce M. Kapron and Stephen A. Cook. A new characterization of type-2 feasibility. *SIAM J. Comput.*, 25(1):117–132, 1996. doi:10.1137/S0097539794263452.

[KC12] Akitoshi Kawamura and Stephen Cook. Complexity theory for operators in analysis. *ACM Trans. Comput. Theory*, 4(2):5:1–5:24, May 2012. doi:10.1145/2189778.2189780.

[KF82] Ker-I Ko and Harvey Friedman. Computational complexity of real functions. *Theoretical Computer Science*, 20:323–352, 1982.

[KMRZ12] Akitoshi Kawamura, Norbert Th. Müller, Carsten Rösnick, and Martin Ziegler. Parameterized Uniform Complexity in Numerics: from Smooth to Analytic, from NP-hard to Polytime. *CoRR*, abs/1211.4974, 2012. URL: http://arxiv.org/abs/1211.4974, arXiv:1211.4974.

[KMRZ15] Akitoshi Kawamura, Norbert Müller, Carsten Rösnick, and Martin Ziegler. Computational benefit of smoothness: Parameterized bit-complexity of numerical operators on analytic functions and Gevreys hierarchy. *Journal of Complexity*, 31(5):689 – 714, 2015. doi:https://doi.org/10.1016/j.jco.2015.05.001.

[Ko83] Ker-I Ko. On the computational complexity of ordinary differential equations. *Inform. Contr.*, 58:157–194, 1983.

[Ko91] Ker-I Ko. *Complexity theory of real functions*. Progress in Theoretical Computer Science. Birkhäuser Boston, Inc., Boston, MA, 1991. doi:10.1007/978-1-4684-6802-1.

[KO14] Akitoshi Kawamura and Hiroyuki Ota. Small complexity classes for computable analysis. In *Mathematical foundations of computer science 2014. Part II*, volume 8635 of *Lecture Notes in Comput. Sci.*, pages 432–444. Springer, Heidelberg, 2014. URL: http://dx.doi.org/10.1007/978-3-662-44465-8_37, doi:10.1007/978-3-662-44465-8_37.

[Kon] Michal Konečný. A Haskell library for Approximating Exact Real Numbers (AERN). https://github.com/michalkonecny/aern2, retrieved 6th November 2017, 16:00.

[KP14a] Akitoshi Kawamura and Arno Pauly. Function spaces for second-order polynomial time. In *Language, life, limits*, volume 8493 of *Lecture Notes in Comput. Sci.*, pages 245–254. Springer, Cham, 2014. doi:10.1007/978-3-319-08019-2_25.

[KP14b] Akitoshi Kawamura and Arno Pauly. On function spaces and polynomial-time computability. *CoRR*, abs/1401.2861v1, 2014. URL: https://arxiv.org/abs/1401.2861v1.

[KS17] Akitoshi Kawamura and Florian Steinberg. Polynomial Running Times for Polynomial-Time Oracle Machines. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*, volume 84 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:18, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSCD.2017.23.

[KST18] Akitoshi Kawamura, Florian Steinberg, and Holger Thies. Parameterized Complexity for Uniform Operators on Multidimensional Analytic Functions and ODE Solving. In *Logic, Language, Information, and Computation - 25th International Workshop, WoLLIC 2018, Bogota, Colombia, July 24-27, 2018, Proceedings*, pages 223–236, 2018.

[KTZ18] Akitoshi Kawamura, Holger Thies, and Martin Ziegler. Average-Case Polynomial-Time Computability of Hamiltonian Dynamics. In *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, pages 30:1–30:17, 2018.

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

[Lac55] Daniel Lacombe. Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles. II, III. *C. R. Acad. Sci. Paris*, 241:13–14, 151–153, 1955.

[Lam05] Branimir Lambov. *Complexity and Intensionality in a Type-1 Framework for Computable Analysis*, pages 442–461. Springer, Berlin Heidelberg, 2005. doi:10.1007/11538363_31.

[Lam06] Branimir Lambov. The basic feasible functionals in computable analysis. *J. Complexity*, 22(6):909–917, 2006. doi:10.1016/j.jco.2006.06.005.

[Meh76] Kurt Mehlhorn. Polynomial and abstract subrecursive classes. *J. Comput. System Sci.*, 12(2):147–178, 1976. Sixth Annual ACM Symposium on the Theory of Computing (Seattle, Wash., 1974).

[Mül] Norbert Th. Müller. iRRAM - Exact Arithmetic in C++. http://irram.uni-trier.de/, https://github.com/norbert-mueller/iRRAM. [Online; accessed 1-March-2019].

[Mül01] Norbert Th. Müller. The iRRAM: Exact Arithmetic in C++. In Jens Blanck, Vasco Brattka, and Peter Hertling, editors, *Computability and Complexity in Analysis: 4th International Workshop, CCA 2000 Swansea, UK, September 17–19, 2000 Selected Papers*, pages 222–252. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. doi:10.1007/3-540-45335-0_14.

[Pez98] Elena Pezzoli. On the computational complexity of type 2 functionals. In Mogens Nielsen and Wolfgang Thomas, editors, *Computer Science Logic: 11th International Workshop, CSL '97 Annual Conference of the EACSL Aarhus, Denmark, August 23–29, 1997 Selected Papers*, pages 373–388. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. doi:10.1007/BFb0028026.

[PG16] Amaury Pouly and Daniel S. Graa. Computational complexity of solving polynomial differential equations over unbounded domains. *Theoretical Computer Science*, 626:67 – 82, 2016.

[Ret13] Robert Rettinger. Computational complexity in analysis, 2013. CCA. URL: https://www.fernuni-hagen.de/imperia/md/content/fakultaetfuermathematikundinformatik/ak/complexityanalysis.pdf.

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

[Sch02b] Matthias Schröder. Extended admissibility. *Theoret. Comput. Sci.*, 284(2):519–538, 2002. Computability and complexity in analysis (Castle Dagstuhl, 1999). doi:10.1016/S0304-3975(01)00109-8.

[Sch04] Matthias Schröder. Spaces allowing type-2 complexity theory revisited. *Math. Log. Q.*, 50(4-5):443–459, 2004. doi:10.1002/malq.200310111.

[SS17] Matthias Schröder and Florian Steinberg. Bounded time computation on metric spaces and Banach spaces. *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 00:1–12, 2017. doi:doi.ieeecomputersociety.org/10.1109/LICS.2017.8005139.

[Ste16] Florian Steinberg. *Computational Complexity Theory for Advanced Function Spaces in Analysis*. PhD thesis, Technische Universität Darmstadt, 2016.

[Ste17] Florian Steinberg. Complexity theory for spaces of integrable functions. *Logical Methods in Computer Science*, Volume 13, Issue 3, Sep 2017. doi:10.23638/LMCS-13(3:21)2017.

[Tur37] Alan M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 01 1937. doi:10.1112/plms/s2-42.1.230.

[Wei00] Klaus Weihrauch. *Computable analysis*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2000. doi:10.1007/978-3-642-56999-9.

A Non-monotone interval enclosures

From a theoretical point of view, the equivalence to the Cauchy representation from Proposition 2.11 justifies our choice of a representation and parameter for the real numbers and the minimality Theorem 2.19 supports that the choices for the real functions are satisfactory. Still, to describe the representations used in certain implementations precisely, one might have to relax the definitions somewhat. Notably, the `iRRAM` library works with enclosures which are not necessarily monotone. We show that this choice is inconsequential for real numbers, in the sense that the resulting parametrised space is equivalent to the space of interval reals. It does however cause difficulties in the treatment of real functions, as the natural function space construction on "`iRRAM` reals" does not allow for a well-defined extension of the function space parameter on interval functions. On the other hand, the restriction of this representation to names with well-defined parameters is again polytime equivalent to the interval function representation. Roughly speaking, a name has a well-defined parameter if it encodes a modulus of continuity of the function it is representing. We argue that all "reasonable" algorithms define names with well-defined parameters.

Definition A.1 A function $\varphi\colon \mathbb{N} \to \mathbb{ID}$ is a $\xi_{\mathbb{R}_{\texttt{iRRAM}}}$-name of $x \in \mathbb{R}$ if

1. $x \in \varphi(n)$ for all $n \in \mathbb{N}$.
2. $\varphi(n) \to \{x\}$ in the Hausdorff metric as $n \to \infty$.

Define the space of `iRRAM` reals to be the parametrised space

$$\mathbb{R}_{\texttt{iRRAM}} = (\mathbb{R}, \xi_{\mathbb{R}_{\texttt{iRRAM}}}, \mu_{\texttt{iRRAM}}),$$

where $\mu_{\mathbb{R}_{\texttt{iRRAM}}}$ is the natural extension of $\mu_{\mathbb{R}_i}$ to the domain of $\xi_{\mathbb{R}_{\texttt{iRRAM}}}$:

$$\mu_{\mathbb{R}_{\texttt{iRRAM}}}(\varphi)(n) = \min\{N \in \mathbb{N} \mid \forall m \geq N : \operatorname{diam}(\varphi(m)) \leq 2^{-m}\} + \lceil \operatorname{lb}(|\xi_{\mathbb{R}_{\texttt{iRRAM}}}(\varphi)|+1)\rceil.$$

The proof that $\mathbb{R}_{\texttt{iRRAM}}$ is indeed a parametrised space proceeds similar to the same proof for the interval reals from Proposition 2.10. Note that unlike the parameter $\mu_{\mathbb{R}_i}$, the parameter $\mu_{\mathbb{R}_{\texttt{iRRAM}}}$ is not computable. Nevertheless, we have the following result:

Proposition A.2 ($\mathbb{R}_{\texttt{iRRAM}} \equiv_{\boldsymbol{P}} \mathbb{R}_{\boldsymbol{i}}$) *The space of `iRRAM` reals and the space of interval reals are isomorphic as parametrised spaces.*

Proof Clearly, $\mathbb{R}_i$ translates into $\mathbb{R}_{\texttt{iRRAM}}$ (using Proposition 2.10), so we just have to find a translation in the opposite direction. Suppose we are given a $\xi_{\mathbb{R}_{\texttt{iRRAM}}}$-name φ of some $x \in \mathbb{R}$. Again, up to applying a polytime computable realiser of the identity as in Proposition 2.10, we can assume that the φ has polynomially bounded size. Then we can compute in polynomial time the function $\psi(n) = \bigcap_{k=0}^{n} \varphi(k)$ which is a $\xi_{\mathbb{R}_i}$-name of x with $\mu_{\mathbb{R}_i}(\psi(n)) = \mu_{\mathbb{R}_{\texttt{iRRAM}}}(\varphi(n))$. ∎

The canonical function space construction from Definition 2.14 immediately yields an analogous representation for the functions on `iRRAM` reals.

Definition A.3 Define the `iRRAM` **function representation** $\xi_{\texttt{iRRAM}f}$ as follows: A function $\psi\colon \mathbb{ID} \to \mathbb{ID}$ is a name of $f \in C([0,1])$ if and only if

$$\forall \varphi \in \operatorname{dom}(\xi_{\mathbb{R}_{\texttt{iRRAM}}}) : (\xi_{\mathbb{R}_{\texttt{iRRAM}}}(\varphi) \in [0,1] \Rightarrow \xi_{\mathbb{R}_{\texttt{iRRAM}}}(\psi \circ \varphi) = f(\xi_{\mathbb{R}_{\texttt{iRRAM}}}(\varphi))).$$

Note that the domain of $\xi_{\texttt{iRRAM}f}$ is strictly larger than the domain of ξ_{if}. In fact, the ξ_{if}-names are precisely the monotone $\xi_{\texttt{iRRAM}f}$-names. The representation $\xi_{\texttt{iRRAM}f}$ has the disadvantage that the parameter μ_{if} cannot be extended to the whole domain of the representation:

Example A.4 Consider the function $\psi\colon \mathbb{ID} \to \mathbb{ID}$ defined by

$$\psi(\mathbf{a}) := \begin{cases} [\frac{1}{2} \pm \frac{1}{2}] & \text{if } \mathbf{a} = [3 \cdot 2^{-n-2} \pm 2^{-n-2}] \text{ for some } n \\ [0 \pm 0] & \text{otherwise.} \end{cases}$$

This function is a $\xi_{\texttt{iRRAM}f}$-name of the zero function and the minimum in the definition of μ_{if} is undefined.

Therefore in order to obtain a parametrised space one has to restrict the representation further:

Definition A.5 The **parametrised space of `iRRAM` functions** is the parametrised space $C(I)_{\texttt{iRRAM}} = (C([0,1]), \xi_{\texttt{iRRAM}f}|_{\mathrm{dom}(\mu_{if})}, \mu_{if})$.

By Lemma 2.16, monotonicity is a sufficient condition for the parameter to be well-defined. Even more generally, by essentially the same argument, continuity with respect to the Hausdorff metric on the dyadic intervals is sufficient. Hence any "reasonable" interval algorithm computes a function with well-defined parameter.

Proposition A.6 ($\boldsymbol{C(I)_{\texttt{iRRAM}} \equiv_P C(I)_i}$) *The space of `iRRAM` functions and the space of interval functions are isomorphic as parametrised spaces.*

Proof It is easy to see that ξ_{if} reduces in polynomial time to $\xi_{\texttt{iRRAM}f}$, using some polytime realiser of the identity in $C(I)_i$ together with the fact that ξ_{if}-names are precisely the monotone $\xi_{\texttt{iRRAM}f}$-names. Conversely, on the space $C(I)_{\texttt{iRRAM}}$ the evaluation functional eval$\colon C(I)_{\texttt{iRRAM}} \times [0,1]_i \to \mathbb{R}_i$ is easily seen to be polytime computable (using Proposition A.2). It follows from Theorem 2.19 that $\xi_{\texttt{iRRAM}f}$ reduces in polynomial time to ξ_{if}. ∎