Parametrised second-order complexity theory with applications to interval computation

Eike Neumann
Aston University
Birmingham, UK

Florian Steinberg[1]
INRIA
Sophia Antipolis, France

A full version of this paper is available on the arXiv [13]. This abstract is kept fairly brief and assumes familiarity with many of the notions central to its topic. A nice and exhaustive introduction to the topic can be found in [6]. Other sources for lookup of details are [15] and the full version of this paper.

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 [17] on the halting problem. 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` [12], Ariadne [1], AERN [9], and RealLib [10].

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. One of the main reasons for such a notion not being available until recently was the lack of an accepted notion of feasibility for second-order functionals. A candidate solution had been proposed by Mehlhorn already in 1975 [11], but it remained a point of debate for a long time to which extent this class fully captures the intuitive notion of feasibility [3]. A characterization of this class by means of resource bounded oracle Turing machines due to Kapron and Cook [5] opened the field up for applications. Based on this characterization, Kawamura and Cook introduced a framework for complexity of operators in analysis [7].

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 encodings that underlie these approaches are known to exhibit highly pathological behaviour within the framework of Kawamura and Cook [8, 16] and the encodings that are used as substitutes do not reflect the behaviour of the software appropriately. For instance, in the Kawamura-Cook model any encoding 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 [2]. 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 encodings.

[1] This author was partially funded by the ANR project *FastRelax*(ANR-14-CE25-0018-01) of the French National Agency for Research.

1 Second-order complexity theory

Let $M^?$ be an oracle machine, where an oracle is an element of Baire space $\mathcal{B} := \Sigma^{*\Sigma^*}$. For a fixed oracle φ such that the machine converges on all inputs, let M^φ denote the function it computes. Every such machine computes a partial operator on Baire space via $\varphi \mapsto M^\varphi$. In second-order complexity theory, the oracle φ is considered an input of the computation and a function bounding the time consumption is allowed to depend on the size of the oracle. The size of an oracle is the worst-case length-increase from input to output and denoted by $|\varphi| : \mathbb{N} \to \mathbb{N}$. A time bound for an oracle machine should thus be of type $\mathbb{N}^\mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

The class of **second-order polynomials** is the smallest class of functions $\mathbb{N}^\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ containing all constant functions returning a polynomial, closed under point-wise addition and multiplication and closed under application of the functional argument. A partial operator $F :\subseteq \mathcal{B} \to \mathcal{B}$ is **polytime computable**, if there is an oracle machine $M^?$ that computes F and a second-order polynomial P which bounds its running time. It was proved by Kapron and Cook [5] that a total operator is polytime computable in this sense if and only if the corresponding functional is basic feasible or polytime in the sense of Mehlhorn [11].

For more general spaces representations are used: A **representation ξ of a set** X is a partial surjective function $\xi :\subseteq \mathcal{B} \to X$. Here, Baire space may be replaced with spaces of functions on easily encodable sets like $\mathbb{N} \to \mathbb{Q}$. A **represented space** is a pair $\mathbf{X} = (X, \xi_\mathbf{X})$ of a set and a representation of that set. For $x \in X$, the elements of $\xi_\mathbf{X}^{-1}(x)$ are called the **names** of x. A **realiser** of $f : \mathbf{X} \to \mathbf{Y}$ is some operator $F :\subseteq \mathcal{B} \to \mathcal{B}$ that maps names of x to names of $f(x)$, *i.e.*, $\xi_\mathbf{Y}(F(\varphi)) = f(\xi_\mathbf{X}(\varphi))$ for all φ from the domain of $\xi_\mathbf{X}$. 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.

2 Complexity on the reals

Define the **Cauchy representation of** $\mathbb{R}$ as follows: A function $\varphi : \mathbb{N} \to \mathbb{D}$ from the naturals to the dyadic rational numbers is a name of a real number x if and only if $|\varphi(n) - x| \leq 2^{-n}$ for all $n \in \mathbb{N}$. While the Cauchy representation is straightforward to realize on a physical computer, a naive implementation can suffer exponential overhead in many calculations if aliased branches of a computation tree need to be evaluated twice to the same accuracy. Any serious implementation has to rely on caching- and memoisation-techniques, which introduces conceptual complexity, and tends to lead to high memory consumption. The algorithms in rigorous numerical analysis and exact real computation are usually based on interval methods. Let $\mathbb{ID}$ denote the set of closed intervals with dyadic rational endpoints.

▶ **Definition 1.** A sequence $(I_n)_{n\in\mathbb{N}} \subseteq \mathbb{ID}$ is a $\xi_{\mathbb{R}_i}$-name of $x \in \mathbb{R}$ if it is monotone, i.e. $I_{n+1} \subseteq I_n$ for all n, and such that $\{x\} = \bigcap_{n\in\mathbb{N}} I_n$.

The use of the interval representation is avoided in real complexity theory as it does not lead to a good notion of complexity: Every real number has names that keep the sequence of intervals constant for an arbitrary long time and these names are of slowly increasing size. As a consequence, the represented space has very pathological complexity theoretical properties.

3 Parametrised spaces

Let $\xi :\subseteq \mathcal{B} \to X$ be a representation. A **parameter for** ξ is a single-valued total map μ from $\mathrm{dom}(\xi)$ to $\mathbb{N}^\mathbb{N}$ with monotone values. A triple (X, ξ, μ) is called a **preparametrised space**. A familiar class of parameters are restrictions of the size

function $\varphi \mapsto |\varphi|$. For any representation, the restriction of the size function to its domain is a parameter and is called the **standard parameter** for the representation.

▶ **Definition 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 an oracle machine $M^?$ that fulfills the following three conditions: Firstly it computes a realizer of f. Secondly, there exists a second-order polynomial P that bounds its running time, *i.e.*, $\mathrm{time}_{M^?}(\varphi, \mathbf{a}) \leq P(\mu_X(\varphi), |\mathbf{a}|)$. And finally there exists another second-order polynomial Q that bounds its parameter blowup, *i.e.*, $\mu_Y(M^\varphi)(n) \leq Q(\mu_X(\varphi), n)$.

We call a realiser $F :\subseteq \mathcal{B} \to \mathcal{B}$ computed by a machine as in Definition 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 second condition from the previous definition turns into the usual time constraint and the third one becomes automatic, as writing the output takes time.

Definition 2 does not guarantee the identity to be polytime computable.

▶ **Definition 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.

If the parameter is a restriction of the size function, the identity on Baire space is always a witness for the polytime computability of the identity. Definition 3 implicitly connects the parameter to the size function: While for an arbitrary name there need not be any relation between its parameter and its size, another name of the same element whose size is bounded polynomially in its parameter may be found in polytime by applying a witnesses for the polytime of the identity function. The definition thus requires the existence of an efficient 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.

▶ **Theorem 4.** *Let* $\mathbf{X}$, $\mathbf{Y}$ *and* $\mathbf{Z}$ *be parameterised spaces and* $f\colon \mathbf{X} \to \mathbf{Y}$ *and* $g\colon \mathbf{Y} \to \mathbf{Z}$ *polytime functions. Then the composition* $g \circ f\colon \mathbf{X} \to \mathbf{Z}$ *is also polytime.*

A product of parametrised spaces can be constructed in a straightforward way.

4 A parametrised space of real numbers

Let diam denote the diameter of an interval and lb the binary logarithm function. Recall the interval representation $\xi_{\mathbb{R}_i}$ of the real numbers from Definition 1. For a $\xi_{\mathbb{R}_i}$-name $(I_n)_{n\in\mathbb{N}}$ of $x \in \mathbb{R}$, define the parameter $\mu_{\mathbb{R}_i}$ by

$$\mu_{\mathbb{R}_i}((I_n)_{n\in\mathbb{N}})(n) := \min\{N \mid \mathrm{diam}(I_N) \leq 2^{-n}\} + \lceil \mathrm{lb}(|x| + 1)\rceil.$$

This parameter mainly encodes the rate of convergence of the sequence of intervals and we call the triple $\mathbb{R}_i = (\mathbb{R}, \xi_{\mathbb{R}_i}, \mu_{\mathbb{R}_i})$ the **parametrised space of interval reals**. Small parameter blowup for a realiser of a function $f\colon \mathbb{R}_i \to \mathbb{R}_i$ means that the rate of convergence of the output sequence is similar to that of the input sequence.

▶ **Lemma 5** ($\mathbb{R}_i \simeq \mathbb{R}_c$)**.** *The reals with Cauchy representation and standard parameter and the space of interval reals are polytime isomorphic as parametrised spaces.*

The following is a property that distinguishes $\mathbb{R}_i$ from the Cauchy reals and is therefore not preserved under isomorphism.

▶ **Theorem 6.** *Whenever* $f\colon \mathbb{R}_i \to \mathbb{R}_i$ *is polytime computable, then* f *can be computed by an oracle machine* $M^?$ *such that there eixsts a* $C \in \mathbb{N}$ *and a second-order polynomial* P *satisfying* $\mathrm{time}_{M^?}(\varphi, \mathbf{a}) \leq C \cdot |\mathbf{a}| + C$ *and* $\mu_i(M^\varphi)(n) \leq P(\mu_X(\varphi), n)$.

The proof uses a construction that was invented to show that the interval representation, when equipped with the restriction of the size function, is complexity theoretically ill-behaved [14, 8, 16]. In contrast to the size, the parameter contains meaningful information about a name. Thus, the construction that used to remove computational cost with respect to the size, now trades time needed to produce approximations against convergence behaviour. This eliminates the use of higher-order bounds in the time constraint while maintaining a reasonable convergence behaviour.

5 A parametrised space of continuous functions

Let $I = [0,1]$ denote the closed unit interval. Define the **interval function representation** ξ_{if} as follows: $\psi\colon \mathbb{ID} \to \mathbb{ID}$ is a name of $f \in C(I)$ if for any monotone sequence of intervals J_n that intersects to a singleton set $\{x\}$, the sequence $\psi(J_n)$ is monotone and intersects to $\{f(x)\}$. That is, if the operator $(J_n)_{n\in\mathbb{N}} \mapsto (\psi(J_n))_{n\in\mathbb{N}}$ is a realiser of f with respect to the interval representation. Let μ_{if} be defined by $\mu_{if}(\psi)(n) := \min\left\{N \in \mathbb{N} \mid \forall J \in \mathbb{ID} : \operatorname{diam}(J) \leq 2^{-N} \Rightarrow \operatorname{diam}(\psi(J)) \leq 2^{-n}\right\} + \lceil \operatorname{lb}(\|\xi_{if}(\psi)\|_\infty + 1)\rceil$. That is: the parameter is an appropriate bound on the modulus of continuity and the supremum norm of the encoded function. We call the space $C(I)_i = (C(I), \xi_{if}, \mu_{if})$ the **parametrised space of interval functions**.

▶ **Lemma 7.** *Evaluation as function from $C(I)_i \times I$ to $\mathbb{R}_i$ is polytime computable.*

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 among those representations which render evaluation polytime computable.

▶ **Theorem 8** (Minimality)**.** *A parametrised representation (ξ, μ) of $C(I)$ renders evaluation polytime computable if and only if it is polytime translatable to (ξ_{if}, μ_{if}).*

▶ **Theorem 9.** *Arithmetic operations and composition are polytime on $C(I)_i$.*

6 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 the first section. Kawamura and Cook add the assumption of length-monotonicity. We call a parametrised space with a length-monotone representation and the standard parameter a **Kawamura-Cook space**. These spaces can also be characterised as follows:

▶ **Theorem 10.** *A parametrised space is polytime isomorphic to a Kawamura-Cook space if and only if it is polytime isomorphic to a space whose parameter is polytime.*

Within their framework, Kawamura and Cook have introduced the a representation $\delta_\square$ of $C(I)$. Kawamura and Cook have proven this representation to be minimal with the property that evaluation is possible in polytime within the class of Kawamura-Cook spaces. This representation is currently the standard representation for complexity theoretical considerations about continuous functions for this reason.

▶ **Theorem 11.** *The representation $\delta_\square$ can be translated to interval function representation in polynomial time. No polytime translation in the other direction exists.*

The reason for the non-existence of a backwards translation is that computing a modulus of continuity takes exponential time with respect to the interval representation. From the minimality of the Kawamura-Cook representation it follows that the space of interval functions is not isomorphic to any Kawamura-Cook space.

Future work

All of [8], [14] and [16] use constructions that produce representation with highly pathological properties. For parametrised spaces these constructions can be adjusted and turn into useful tools. Theorem 6 is a special case of this and a detailed description of the general case will be part of future work. Recently there has been some interest in exponentiable objects in the category of represented spaces and polytime functions [8, 4]. We hope that the study of parametrised spaces can shed some light on this, as these seem to allow for more natural exponential constructions.

References

1. 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.
2. Franz Brauße and Florian Steinberg. A minimal representation for continuous functions. `https://arxiv.org/abs/1703.10044`, 2017. preprint.
3. Stephen A. Cook. *Computability and Complexity of Higher Type Functions*, pages 51–72. Springer New York, 1992. `doi:10.1007/978-1-4612-2822-6_3`.
4. Hugo Férée and Mathieu Hoyrup. Higher order complexity in analysis, 2013. CCA. URL: `https://hal.inria.fr/hal-00915973/document`.
5. B. M. Kapron and S. A. Cook. A new characterization of type-2 feasibility. *SIAM J. Comput.*, 25(1):117–132, 1996. `doi:10.1137/S0097539794263452`.
6. Akitoshi Kawamura. *Computational Complexity in Analysis and Geometry*. PhD thesis, University of Toronto, 2011.
7. 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`.
8. 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`.
9. Michal Konečný. A Haskell library for Approximating Exact Real Numbers (AERN). `https://github.com/michalkonecny/aern2`, retrieved 6th November 2017, 16:00.
10. Branimir Lambov. The basic feasible functionals in computable analysis. *J. Complexity*, 22(6):909–917, 2006. `doi:10.1016/j.jco.2006.06.005`.
11. 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).
12. Norbert Th. Müller. iRRAM - Exact Arithmetic in C++. http://irram.uni-trier.de/, https://github.com/norbert-mueller/iRRAM, 2017. [Online; accessed 6-January-2017].
13. Eike Neumann and Florian Steinberg. Parametrised second-order complexity theory with applications to the study of interval computation. preprint. URL: `https://arxiv.org/abs/1711.10530`.
14. Matthias Schröder. Spaces allowing type-2 complexity theory revisited. *MLQ Math. Log. Q.*, 50(4-5):443–459, 2004. `doi:10.1002/malq.200310111`.
15. 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`.
16. Florian Steinberg. *Computational Complexity Theory for Advanced Function Spaces in Analysis*. PhD thesis, Technische Universität Darmstadt, 2016.
17. Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.