Complexity Theory for Operators in Analysis

Akitoshi Kawamura
University of Toronto
kawamura@cs.toronto.edu

Stephen Cook
University of Toronto
sacook@cs.toronto.edu

ABSTRACT

We propose a new framework for discussing computational complexity of problems involving uncountably many objects, such as real numbers, sets and functions, that can be represented only by approximation. The key idea is to use a certain class of string functions, which we call *regular functions*, as names representing these objects. These are more expressive than infinite sequences, which served as names in prior work that formulated complexity in more restricted settings. An important advantage of using regular functions is that we can define their *size* in the way inspired by higher-type complexity theory. This enables us to talk about computation on regular functions whose time or space is bounded polynomially in the input size, giving rise to more general analogues of the classes P, NP, and PSPACE. We also define NP- and PSPACE-completeness under suitable many-one reductions.

Because our framework separates machine computation and semantics, it can be applied to problems on sets of interest in analysis once we specify a suitable representation (encoding). As prototype applications, we consider the complexity of functions (operators) on real numbers, real sets, and real functions. The latter two cannot be represented succinctly using existing approaches based on infinite sequences, so ours is the first treatment of them. As an interesting example, the task of numerical algorithms for solving the initial value problem of differential equations is naturally viewed as an operator taking real functions to real functions. As there was no complexity theory for operators, previous results could only state how complex the solution can be. We now reformulate them to show that the operator itself is polynomial-space complete.

Categories and Subject Descriptors

F.1.3 [**Computation by Abstract Devices**]: Complexity Measures and Classes

General Terms

Theory

Keywords

computable analysis, computational complexity, higher-type complexity, second-order polynomials

1. INTRODUCTION

Computable analysis [19] studies problems involving real numbers, sets and functions from the viewpoint of computability. Uncountably many objects (such as real numbers) are represented by infinite sequences of approximations and processed by Turing machines. This framework is applicable not only to the real numbers but also with great generality to other spaces arising naturally in mathematical analysis. There is a unified way to discuss computability of real functions, sets of real numbers, operators taking real functions as inputs, and so on.

In contrast, the application of this approach to computational complexity has been limited in generality. For example, although there is a widely accepted notion of polynomial-time computable real functions $f : [0, 1] \to \mathbb{R}$ on the compact interval that has been studied extensively [14], the same approach does not give a nice class of real functions on $\mathbb{R}$. Most of the complexity results in computable analysis to date (with a few exceptions [6, 18, 20]) are essentially limited to the complexity of either real functions with compact domain or of bounded subsets of $\mathbb{R}$. They do not address the complexity of, say, an operator F that takes real functions $f : [0, 1] \to \mathbb{R}$ to another real function $F(f)$. There are many results [12] about such operators, but typically the positive and negative results are stated in the form

> if f is in the (small) complexity class X, then $F(f)$ is in complexity class Y, and
> there is f in complexity class X such that $F(f)$ is hard for Z.

More direct (and preferable) statements would be the "constructive" or "effectivized" form

> the operator F is in class $\mathcal{Y}$, and
> the operator F is $\mathcal{Z}$-hard,

where $\mathcal{Y}$ and $\mathcal{Z}$ are the "higher-order versions" of Y and Z. At the level of computability, it is common to ask, as soon as we see an ineffective result, whether it can be effectivized. For complexity, we cannot even ask this question because we do not know how to formulate $\mathcal{Y}$ and $\mathcal{Z}$. This limitation has been widely recognized; see, for example, [12, pp. 57–58], [20], and [3, p. 484].

The present paper addresses this problem. We start with the observation (Section 2) that the aforementioned limitation has to do with the fact that traditional formulations of computable analysis do not take into account the "size" of the infinite sequences given

to the machine as input. We then propose (Section 3) an extension on the machine model by replacing infinite sequences by what we call *regular functions* from strings to strings. An important advantage of regular functions is that we can define their *size* in the way suggested by type-two complexity theory [16, 7]. This enables us to measure the growth of running time (or space) in terms of the input size—exactly what we do in the usual (type-one) Complexity Theory. We thus obtain the complexity classes analogous to P, NP, PSPACE (and function classes FP and FPSPACE) by bounding the time or space by *second-order polynomials* in the input size. Analogues of many-one reductions and NP- and PSPACE-hardness will also be introduced.

We apply this framework to a few specific problems in analysis by using suitable representations of real numbers, real sets, and real functions (Section 4). For real numbers, the induced complexity notions turn out to be equivalent to what has been studied by Ko–Friedman [10] and Hoover [6]. For sets and functions, our approach seems to be the first to provide complexity notions in a unified manner. This is of particular interest, because many numerical problems in the real world are naturally formulated as operators taking sets or functions. For example, consider the operator F that finds the solution $F(f)$ of the differential equation (of a certain class) given by a function f. As mentioned above, the existing ineffective results [11, 8] only tell us *how complex the solution $F(f)$ can be when f is easy*; precisely, they say that if f is polynomial-time computable, $F(f)$ is polynomial-space computable and can be polynomial-space hard. But the practical concern for numerical analysis would be *how hard it is to compute F* (i.e., to compute $F(f)$ given f). We formulate and prove the first result of this kind: F itself is a polynomial-space complete operator. Our contribution is in introducing the framework making such formulations possible, solving an important open problem as explained above. The technically hard parts of the proofs of the specific results are already done in the proofs of the ineffective versions, and all we need to do is to check that they effectivize in our sense. The original ineffective versions are now corollaries of the effectivized statements.

2. TYPE-TWO THEORY OF EFFECTIVITY

Computable Analysis dates back at least to Grzegorczyk [5], and there have been several equivalent formulations. Weihrauch's Type-Two Theory of Effectivity (TTE) is a powerful framework. In this section, we briefly introduce computable analysis through TTE and clarify its limitations in dealing with complexity.

2.0 Multi-functions

Before discussing TTE, we begin with a few words about *multi-valued functions*, or *multi-functions*. A multi-function F from a set X to a set Y is formally a subset of $X \times Y$. For $x \in X$, we write $F[x]$ for the set of $y \in Y$ such that (x, y) belongs to this subset. These y are the "allowable outputs" on input x. We denote by dom F the set of $x \in X$ for which $F[x]$ is nonempty. When $F[x]$ is a singleton, its unique element is denoted by $F(x)$, as usual. If $F[x]$ is a singleton for all $x \in$ dom F, we say that F is a *partial function*. When in addition dom $F = X$, we say that F is a *total function*, or simply a *function*.

Like some authors, we regard the so-called "function problems" (also called *search problems*) as multi-functions. The classes FP and FPSPACE are the sets of multi-functions from strings to strings that are computed by a machine whose time/space is polynomially bounded. Here, the word "compute" is interpreted according to the "allowable outputs" semantics of multi-functions: A machine

is said to compute F if, on any input $x \in$ dom F, it outputs *some* element of $F[x]$. The classes FP^2 and $\mathsf{FPSPACE}^2$ that we will define later will also consist of multi-functions.

Note that we do not care what happens on inputs outside dom F, unlike some authors who require that such inputs be rejected explicitly. Thus, a multi-function can be easy to compute while having a nasty domain. We also note, however, that allowing dom F to be smaller than X is not so important in the context of time- or space-bounded computation, because a machine that runs past the time bound for some inputs can be modified so that it keeps track of the time and outputs an error message when it has run out of time.

2.1 Computability

In the usual Computability Theory, we introduce computability for functions from Σ^* to Σ^* under some machine model, where Σ^* is the set of strings. To discuss computation on other sets X, we specify an *encoding* of X—that is, a rule for interpreting an element of Σ^* as an element of X.

TTE roughly follows the same path. But since the countable set Σ^* cannot encode uncountable sets, such as the set $\mathbb{R}$ of real numbers, TTE uses the set $\Sigma^{\mathbb{N}}$ of *infinite sequences*.

Computability of functions from $\Sigma^{\mathbb{N}}$ to $\Sigma^{\mathbb{N}}$ is defined using Turing machines. The machine has an input tape, an output tape and a work tape, each of which is infinite to the right. We also assume that the output tape is one-way, that is, the only instruction for the output tape is "write $a \in \Sigma$ in the current cell and move the head to the right". The difference from the usual setting is in the convention by which the machine reads the input and delivers the output. The input is now an infinite string $a_0 a_1 \ldots \in \Sigma^{\mathbb{N}}$, and is written on the input tape before the computation starts (with the tape heads at the leftmost cell). We say the machine outputs an infinite string $b_0 b_1 \ldots \in \Sigma^{\mathbb{N}}$ if it never halts and writes the string indefinitely (that is, for each $n \in \mathbb{N}$, it eventually writes $b_0 \ldots b_{n-1}$ into the first n cells) on the output tape. This defines a class of (possibly partial) computable functions (without any time or space bound) from $\Sigma^{\mathbb{N}}$ to $\Sigma^{\mathbb{N}}$. The definition can be extended to multi-functions A: We say that a machine M computes A if M, on any input $\varphi \in$ dom A, always outputs some element of $A[\varphi]$.

A *representation* γ of a set X is formally a partial function from $\Sigma^{\mathbb{N}}$ to X which is surjective—that is, for each $x \in X$, there is at least one $\varphi \in \Sigma^{\mathbb{N}}$ with $\gamma(\varphi) = x$. We say that φ is a γ-*name* of x. Using representations, computability of multi-functions on represented sets is defined as follows.

Definition 2.1. Let γ and δ be representations of sets X and Y, respectively. We say that a machine (γ, δ)-*computes* a multi-function A from X to Y if it computes the multi-function $\delta^{-1} \circ A \circ \gamma$ given by

$$(\delta^{-1} \circ A \circ \gamma)[\varphi]$$
$$= \begin{cases} \{ \psi \in \text{dom } \delta : \delta(\psi) \in A[\gamma(\varphi)] \} & \text{if } \varphi \in \text{dom } \gamma, \\ \emptyset & \text{otherwise.} \end{cases} \quad (1)$$

In other words, whenever the machine is given a γ-name of an element $x \in$ dom A, it must output some δ-name of some element of $A[x]$ (Figure 1).

As an example, we define a representation $\rho_{\mathbb{R}}$ of $\mathbb{R}$, the set of real numbers. First, for each $n \in \mathbb{N}$, let $\mathbf{D}_n$ denote the set of strings of form

$$sx/1\underbrace{00\ldots0}_{n}, \quad (2)$$

where $s \in \{+, -\}$ and $x \in \{0, 1\}^*$. Let $\mathbf{D} = \bigcup_{n \in \mathbb{N}} \mathbf{D}_n$. A string in $\mathbf{D}$ *encodes* a rational number in the obvious sense (read (2) as a
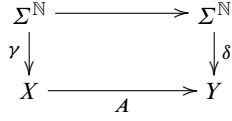
Figure 1: (γ, δ)-computing a multi-function A.

fraction whose numerator and denominator are integers written in binary with leading zeros allowed). We write $[\![u]\!]$ for the rational number encoded by $u \in \mathbf{D}$. We define a representation $\rho_{\mathbb{R}}$ of $\mathbb{R}$ by saying that an infinite string $\varphi \in \Sigma^{\mathbb{N}}$ is a $\rho_{\mathbb{R}}$-name of $x \in \mathbb{R}$ if φ is of the form $u_0 \# u_1 \# u_2 \# \dots$ (where # is a delimiter symbol not appearing in the u_i) such that $u_i \in \mathbf{D}$ and $|[\![u_i]\!] - x| < 2^{-i}$ for each $i \in \mathbb{N}$. Thus, this representation specifies a real number as a list of rational numbers converging to it.

It turns out that $\rho_{\mathbb{R}}$ is a natural representation with which to discuss computability of real functions. In particular, $\rho_{\mathbb{R}}$ is *admissible*, that is, it matches well with the usual topology of $\mathbb{R}$ [19, Lemma 4.1.6].

To deal with functions of two arguments, we define, for representations γ and δ of sets X and Y, a new representation $[\gamma, \delta]$ of $X \times Y$ by $[\gamma, \delta](a_0 b_0 a_1 b_1 a_2 b_2 \dots) = (\gamma(a_0 a_1 a_2 \dots), \delta(b_0 b_1 b_2 \dots))$.

Example 2.2. Binary addition $+ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is $([\rho_{\mathbb{R}}, \rho_{\mathbb{R}}], \rho_{\mathbb{R}})$-computable. For suppose that we are given names $\varphi = u_0 \# u_1 \# \dots$ and $\psi = v_0 \# v_1 \# \dots$ of real numbers s and t. An approximation of $s + t$ with precision 2^{-m}, for each m, is given by $[\![u_{m+1}]\!] + [\![v_{m+1}]\!]$.

Example 2.3. Multiplication $\times : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is $([\rho_{\mathbb{R}}, \rho_{\mathbb{R}}], \rho_{\mathbb{R}})$-computable. For suppose that we are given names $\varphi = u_0 \# u_1 \# \dots$ and $\psi = v_0 \# v_1 \# \dots$ of real numbers s and t. First, let $k = \max\{|u_0|, |v_0|\}$. Since $[\![u_0]\!]$ and $[\![v_0]\!]$ are near s and t, and it takes more than k digits to encode a number with absolute value $\geq 2^k$, we have $|s|, |t| < 2^k$. Hence, $s \times t$ is approximated with precision 2^{-m} by $[\![u_{m+k+1}]\!] \cdot [\![v_{m+k+1}]\!]$.

The strength of TTE is that, by using suitable representations, we can discuss computation over sets other than $\mathbb{R}$. Moreover, there are often standard ways to introduce representations of higher-type objects, such as sets or of functions. For example, since we have agreed on the representations $\rho_{\mathbb{R}}$ of $\mathbb{R}$, we can introduce a canonical representation of the set $C[\mathbb{R}]$ of continuous real functions, and there are reasons to believe that this is the "right" representation [19, Chapter 3].

2.2 Complexity

Now we ask whether the computation runs in polynomial time. We ask whether the time it takes to output the nth prefix of the output is bounded by a polynomial in n (independently of φ):

Definition 2.4. A machine M runs in *polynomial time* if there is a polynomial p such that for all $\varphi \in \Sigma^{\mathbb{N}}$ and $n \in \mathbb{N}$, the machine M on input φ finishes writing the first n symbols of the output within $p(n)$ steps. Define *polynomial space* analogously by counting the number of visited cells on all (input, work and output) tapes.

Can we use this notion to define polynomial-time computability of, say, a real function?

2.2.1 Representations must be chosen carefully

A little thought shows that the simple combination of Definition 2.4 and the representation $\rho_{\mathbb{R}}$ is useless [19, Examples 7.2.1,

7.2.3]. On the one hand, the machine M could "cheat" by writing redundant $\rho_{\mathbb{R}}$-names: By writing $+10000/100000$ instead of $+1/10$, it gets more time to compute the next approximation. On the other hand, the machine may suffer by receiving redundant names as input, such as the one in which the first approximation is too long to even read in time.

Thus, to develop a meaningful complexity theory, we need to disallow redundancy carefully. This leads to the use of *signed digit representation* ρ_{sd} of $\mathbb{R}$ [19, Definition 7.2.4], defined as follows: $\text{dom } \rho_{\text{sd}}$ consists of sequences $\varphi \in \Sigma^{\mathbb{N}}$ of form $a_k \dots a_1 a_0 \bullet$ $a_{-1} a_{-2} \dots$ for some k, where each a_i is either 0, 1 or -1, such that either $k = 0$ or $(a_k, a_{k-1}) \in \{(1, 0), (1, 1), (-1, 0), (-1, -1)\}$; if this is the case, set

$$\rho_{\text{sd}}(\varphi) = \sum_{i=-\infty}^{k} a_i \cdot 2^i \tag{3}$$

Thus, we read the digit sequence as a binary expansion of a real number (with decimal point $\bullet$) with digits 0, 1 and -1; we forbid certain patterns in the first two digits of the integer part in order to exclude redundancy. (See [19, Example 2.1.4.7] for the reason why the usual binary expansion without the "-1" digit does not work.)

Let $\rho_{\text{sd}}|^{[0,1]}$ denote the restriction of ρ_{sd} to (infinite sequences representing) real numbers in $[0, 1]$. By Definition 2.4, we know what it means for a real function $f : [0, 1] \to \mathbb{R}$ to be *polynomial-time* $(\rho_{\text{sd}}|^{[0,1]}, \rho_{\text{sd}})$-computable. This notion turns out to be robust and natural, and equivalent to the widely accepted polynomial-time computability of Ko and Friedman [10], so we will drop the prefix "$(\rho_{\text{sd}}|^{[0,1]}, \rho_{\text{sd}})$" from now on. The same goes for *polynomial-space computability*, and for functions on compact intervals or rectangles instead of $[0, 1]$ (use the pairing function as in Examples 2.2 and 2.3). It is routine to verify that, for example, addition and multiplication $+, \times : [0, 1] \times [0, 1] \to \mathbb{R}$ are polynomial-time computable. For more interesting results, see Ko's book [12], survey [14] or Weihrauch's book [19, Section 7.3].

2.2.2 Difficulties in generalizing to other spaces

Unfortunately, this approach does not extend much further. For example, a naive extension to real functions on $\mathbb{R}$ (instead of $[0, 1]$) does not work: polynomial-time $(\rho_{\text{sd}}, \rho_{\text{sd}})$-computability tends to fail for trivial reasons, as in the following example.

Example 2.5. Addition on $\mathbb{R}$ (Example 2.2) is not polynomial-time $([\rho_{\text{sd}}, \rho_{\text{sd}}], \rho_{\text{sd}})$-computable. For suppose that a machine $([\rho_{\text{sd}}, \rho_{\text{sd}}], \rho_{\text{sd}})$-computed it within polynomial time bound p. In particular, the machine has to write the first symbol of the output in $t := p(1)$ steps or fewer. Note that this first symbol must be 1 if the sum is greater than 1, and -1 if the sum is less than -1. In particular, it must be 1 if the two summands are 2^{t+100} and -2^{t+50}, and -1 if they are 2^{t+50} and -2^{t+100}. However, the machine cannot tell between these two cases, because it can read at most t symbols of the input in time.

The trouble seems to be that the time bound is independent of the input. Compare this with the addition of integers (written in binary) by the usual Turing machine. It is in polynomial time, because a large summand would make the "input size" big and thereby give the machine more time. For the same thing to happen for addition of the real numbers, we would need to talk about the "size" of the input and a time bound "polynomial in" it, but we do not have the notion of size for infinite sequences.

This problem is not just about real numbers. There are many other objects that we want to give representations to. The objects
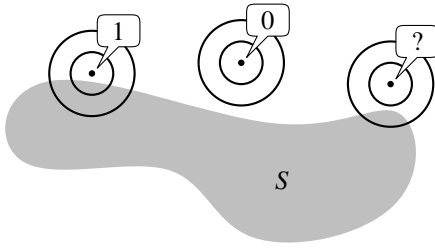
Figure 2: Computing a set S.

for which TTE gives reasonable notions of complexity are limited[1], compared to what we can do at the level of computability. This has long been recognized as a challenge; see, for example, [12, pp. 57–58], [20, Section 1], or [3, p. 484]. As a result of this limitation, the complexity of operators working on objects other than real numbers has been mostly formulated in non-constructive (ineffective) forms. We quote examples of such theorems below. We will reformulate them in effectivized forms later (Theorems 4.6 and 4.9).

2.2.3   Ineffective results

The first pair of results are the positive and negative statements about the operator of taking the convex hull $CH(S)$ of a closed set $S \subseteq [0, 1]^2$.

Polynomial-time computability (deterministic or nondeterministic) of a set $S \subseteq [0, 1]^2$ is defined using usual Turing machines as follows (see e.g. Braverman [4] for a discussion on this definition). We say that S is *(nondeterministic) polynomial-time computable* if there is a (nondeterministic) polynomial-time Turing machine that computes a function $\varphi : \Sigma^* \to \{0, 1\}$ such that, for any $n \in \mathbb{N}$ and $u, v \in \mathbf{D}$,

- $\varphi(u, v, 0^n) = 1$ if $dist((\llbracket u \rrbracket, \llbracket v \rrbracket), S) < 2^{-n}$, and

- $\varphi(u, v, 0^n) = 0$ if $dist((\llbracket u \rrbracket, \llbracket v \rrbracket), S) > 2 \cdot 2^{-n}$,

where $dist(p, S) := \inf_{q \in S} \| p - q \|$ denotes the Euclidean distance of point $p \in \mathbb{R}^2$ from S (Figure 2).

Theorem 2.6 (Essentially[2] in [15, Corollary 4.3])**.** *If a set $S \subseteq [0, 1]^2$ is polynomial-time computable, then $CH(S)$ is nondeterministic polynomial-time computable.*

Theorem 2.7 (Essentially[2] in [15, Corollary 4.6])**.** *Unless* $\mathsf{P} = \mathsf{NP}$, *there exists a set $S \subseteq [0, 1]^2$ which is polynomial-time computable, but whose convex hull $CH(S)$ is not.*

For $A \subseteq \mathbb{R}^d$, let $C[A]$ be the set of continuous functions from A to $\mathbb{R}$. The second pair of results concerns the differential equation

$$h(0) = 0, \qquad h'(t) = g(t, h(t)) \qquad (4)$$

called the initial value problem (IVP), where $g \in C[[0, 1] \times \mathbb{R}]$ is given and $h \in C[0, 1]$ is the unknown. The *Picard–Lindelöf* (or

---

[1]We note, however, that Weihrauch [20] and Schröder [17] study how far we can get with the infinite string model.

[2]Ko and Yu state these results for *strong recognizability* instead of computability, but their proof almost works for computability as well. See Braverman [4] for a comparison of the two notions ([4] says *weak computability* for Ko's strong recognizability). In the present paper, Theorems 2.6 and 2.7 will be derived as corollaries of the effective version, Theorem 4.6.

Cauchy–Lipschitz) *Theorem* states that the solution h exists and is unique if g is *Lipschitz continuous* (in the second argument), i.e.,

$$|g(t, y_0) - g(t, y_1)| \le L \cdot |y_0 - y_1| \qquad (5)$$

for some nonnegative constant L independent of t, y_0, y_1. The following results state how complex h can be, assuming that g is polynomial-time computable. Since polynomial-time computability is defined only for functions with compact domain, we restrict g to the rectangle $[0, 1] \times [-1, 1]$. If there is a solution $h \in C[0, 1]$ whose values stay in $[-1, 1]$ (in which case h is unique, as mentioned above), we write *LipIVP(g)* for this h. Thus, *LipIVP* is a partial function from $CL[[0, 1] \times [-1, 1]]$ to $C[0, 1]$, where the former set is the subset of $C[[0, 1] \times [-1, 1]]$ consisting of functions Lipschitz continuous in the second argument.

Theorem 2.8 ([11, Section 4][3])**.** *If $g \in \mathrm{dom}\, LipIVP$ is polynomial-time computable, then $LipIVP(g)$ is polynomial-space computable.*

Theorem 2.9 ([8, Theorem 3.2])**.** *There is a polynomial-time computable function $g \in \mathrm{dom}\, LipIVP$ such that $LipIVP(g)$ is polynomial-space complete (in the sense defined in [13] or [8]).*

We can derive from Theorem 2.9 a statement of a similar form to Theorem 2.7: unless $\mathsf{P} = \mathsf{PSPACE}$, there is a real function $g \in \mathrm{dom}\, LipIVP$ which is polynomial-time computable but $LipIVP(g)$ is not.

3.   FUNCTIONS AS NAMES

We present the main definitions for our framework here.

As we have noted, the limitations of the TTE approach have to do with the fact that the information carried by infinite sequences in $\Sigma^{\mathbb{N}}$ is not rich enough, and in particular we do not have the notion of their size. We replace $\Sigma^{\mathbb{N}}$ with **Reg**, a class of string functions which we will use as names of real numbers, sets and functions[4]. This section develops a complexity theory for computation over **Reg**, introducing the analogues of classes P, NP, PSPACE and the notions of completeness under many-one reductions.

3.1   Computation over regular functions

We say that a (total) function $\varphi : \Sigma^* \to \Sigma^*$ is *regular* if it preserves relative lengths of strings in the sense that $|\varphi(u)| \le |\varphi(v)|$ whenever $|u| \le |v|$. We write **Reg** for the set of all regular functions (the motivation for considering these functions will become clear in Section 3.2 where we define their lengths).

Now we begin replacing the role of $\Sigma^{\mathbb{N}}$ in TTE (see Section 2.1) by **Reg**. This is a generalization, since an infinite string $a_0 a_1 \ldots \in \Sigma^{\mathbb{N}}$ can be identified with a regular function $\varphi \in \mathbf{Reg}$ which

(a) takes values of length 1, and

(b) depends only on the length of the argument,

by setting $\varphi(0^n) = a_n$. In the following, observe that Definitions 3.1 and 3.2 extend their counterparts in TTE in this sense.

Since we will deal with an analogue of NP, and nondeterminism is better understood for predicates ($\{0, 1\}$-valued functions), it sometimes makes sense to stop the generalization halfway, removing (b) only and keeping (a). Let **Pred** $\subseteq$ **Reg** be the set of $\{0, 1\}$-valued regular functions.

Instead of the machine in TTE that converted infinite strings to infinite strings, we use an oracle Turing machine (henceforth just "machine") to convert regular functions to regular functions:

---

[3]The original theorem is stated with a condition slightly weaker than Lipschitz continuity.

[4]Ko's formulation [12] already uses string functions instead of infinite strings of TTE, but it does not make full use of this extension.

Definition 3.1. 1. A deterministic machine *computes* a function $F : \mathbf{Reg} \to \mathbf{Reg}$ if, given φ as oracle and u as string input, it outputs $F(\varphi)(u)$ and halts.

2. A nondeterministic machine *computes* a function $F : \mathbf{Reg} \to \mathbf{Pred}$ if, given φ as oracle and u as input, it has at least one accepting branch exactly when $F(\varphi)(u) = 1$.

For the precise conventions for issuing and answering queries, follow any of [16, 7, 12].

As before, we can extend these definitions to multi-functions F by saying that a machine computes F if on any input $\varphi \in \mathrm{dom}\, F$ it outputs *some* element of $F[\varphi]$.

3.2 Polynomial time and space

Recall that regular functions are those that respect lengths in the sense explained at the beginning of Section 3.1. In particular, they map strings of equal lengths to strings of equal lengths. Therefore, it makes sense to define the *size* $|\varphi| : \mathbb{N} \to \mathbb{N}$ of a regular function φ by $|\varphi|(|u|) = |\varphi(u)|$. It is a non-decreasing function from $\mathbb{N}$ to $\mathbb{N}$.

Now we want to define what it means for a machine to run in polynomial time. Since $|\varphi|$ is a function, we begin by defining polynomials "in" a function, following the idea of Kapron and Cook [7]. *Second-order polynomials* (in type-1 variable $\mathtt{L}$ and type-0 variable $\mathtt{n}$) are defined inductively as follows: a positive integer is a second-order polynomial; the variable $\mathtt{n}$ is also a second-order polynomial; if P and Q are second-order polynomials, then so are $P + Q$, $P \cdot Q$ and $\mathtt{L}(P)$. An example is

$$\mathtt{L}\big(\mathtt{L}(\mathtt{n} \cdot \mathtt{n})\big) + \mathtt{L}\big(\mathtt{L}(\mathtt{n}) \cdot \mathtt{L}(\mathtt{n})\big) + \mathtt{L}(\mathtt{n}) + 4. \tag{6}$$

A second-order polynomial P specifies a function, which we also denote by P, that takes a function $p : \mathbb{N} \to \mathbb{N}$ to another function $P(p) : \mathbb{N} \to \mathbb{N}$ in the obvious way. For example, if P is the above second-order polynomial (6) and $p(x) = x^2$, then $P(p)$ is given by

$$P(p)(x) = \big((x \cdot x)^2\big)^2 + (x^2 \cdot x^2)^2 + x^2 + 4$$
$$= x^8 + x^4 + x^2 + 4. \tag{7}$$

As in this example, $P(p)$ is a (usual first-order) polynomial if p is.

Definition 3.2. A (deterministic or nondeterministic) machine M runs in (*second-order*) *polynomial time* if there is a second-order polynomial P such that, given any $\varphi \in \mathbf{Reg}$ as oracle and any $u \in \Sigma^*$ as input, M halts within $P(|\varphi|)(|u|)$ steps (regardless of the nondeterministic choices). Define *polynomial space* analogously by counting the number of visited cells on all (input, work, output and query) tapes.

When φ satisfies (a), the size $|\varphi|$ is a constant function, so the bound $P(|\varphi|)(|u|)$ reduces to a (usual first-order) polynomial in $|u|$. Therefore, Definition 3.2 can be viewed as an extension of Definition 2.4.

Definition 3.3. 1. We write FP^2 (resp. $\mathsf{FPSPACE}^2$) to denote the class of multi-functions from $\mathbf{Reg}$ to $\mathbf{Reg}$ computed by a deterministic machine that runs in second-order polynomial time (resp. space).

2. We write P^2 (resp. NP^2) for the class of multi-functions from $\mathbf{Reg}$ to $\mathbf{Pred}$ computed by a deterministic (resp. nondeterministic) machine M that runs in polynomial time.

Note that unlike FP and $\mathsf{FPSPACE}$, it is easy to separate, e.g., FP^2 and $\mathsf{FPSPACE}^2$, because an $\mathsf{FPSPACE}^2$ machine can make exponentially many queries to the given oracle.

It is easy to see that the classes defined here respect the corresponding usual complexity classes:

Lemma 3.4. *1. Functions in* FP^2 *(resp.* $\mathsf{FPSPACE}^2$*) map regular functions in* FP *to* FP *(resp.* $\mathsf{FPSPACE}$ *to* $\mathsf{FPSPACE}$*).*

2. Functions in P^2 *(resp.* NP^2*) map regular functions in* FP *to* P *(resp.* NP*).*

Why we restrict ourselves to regular functions

The idea of using second-order polynomial as a bound on time and space comes from Kapron and Cook's characterization [7] of Mehlhorn's class [16] of *polynomial-time computable operators*[5]. This is a class of (total) functionals $F : (\Sigma^* \to \Sigma^*) \times \Sigma^* \to \Sigma^*$, but they can be regarded as $F : (\Sigma^* \to \Sigma^*) \to (\Sigma^* \to \Sigma^*)$ by writing $F(\varphi)(x)$ instead of $F(\varphi, x)$. Kapron and Cook define the size of $\varphi : \Sigma^* \to \Sigma^*$ by

$$|\varphi|(n) = \max_{|u| \leq n} |\varphi(u)|, \qquad n \in \mathbb{N}. \tag{8}$$

Note that our definition of size for regular φ is a special case of this. Then they defined the class of polynomial-time functionals in the same way as Definition 3.3.1. (We added $\mathsf{FPSPACE}^2$ by analogy.)

We have restricted attention to regular functions. This is because, in order to obtain reasonable complexity notions, it seems necessary to be able to simulate a given machine with known time bounds. Note that for usual (type-one) computation, it was easy to find $|x|$ given x, and thus to clock the machine with the time bound $p(|x|)$ for a fixed polynomial p. In contrast, finding the value (8) for a given φ in general requires exponentially many queries to φ and thus exponential time. For regular φ, we can easily find $|\varphi|(n)$ for each n, and thus the second-order polynomial $P(|\varphi|)(|u|)$ is a bound "time-constructible" from φ and u.

Regular functions φ suffice for our purpose, because our intention is to use φ as names of real numbers, sets and functions, and there seems to be little incentive to use irregular names. All representations in this paper (Section 4) will use regular functions only.

3.3 Reduction and completeness

Recall that the definition of the usual many-one reduction between multi-functions A, B from Σ^* to Σ^* is as follows: we say that A many-one reduces to B (written $A \leq^1_{\mathrm{mF}} B$) if there are (total) functions $r, t \in \mathsf{FP}$ such that for any $u \in \mathrm{dom}\, A$, we have $r(v) \in A[u]$ whenever $v \in B[t(u)]$—that is, we have a function t that converts an input for A to an input for B, and another function r that converts an output of B to an output of A (we omit r in the many-one reduction $\leq^1_{\mathrm{m}}$ between predicates). Since multi-functions over $\mathbf{Reg}$ also get a function as input, the analogous definition of reduction involves one more converter s:

Definition 3.5. 1. Let A and B be multi-functions from $\mathbf{Reg}$ to $\mathbf{Reg}$. We say that A *many-one reduces to* B (written $A \leq^2_{\mathrm{mF}} B$) if there are (total) functions $r, s, t \in \mathsf{FP}^2$ such that for any $\varphi \in \mathrm{dom}\, A$, we have $s(\varphi) \in \mathrm{dom}\, B$ and $r(\varphi) \circ \theta \circ t(\varphi) \in A[\varphi]$ whenever $\theta \in B[s(\varphi)]$ (Figure 3).

2. Let A and B be multi-functions from $\mathbf{Reg}$ to $\mathbf{Pred}$. We write $A \leq^2_{\mathrm{m}} B$ if there are functions $s, t \in \mathsf{FP}^2$ such that for any $\varphi \in \mathrm{dom}\, A$, we have $s(\varphi) \in \mathrm{dom}\, B$ and $\theta \circ t(\varphi) \in A[\varphi]$ whenever $\theta \in B[s(\varphi)]$.

---

[5]Kapron and Cook [7] call them *Basic Feasible Functionals* or *Basic Polynomial-Time Functionals*.
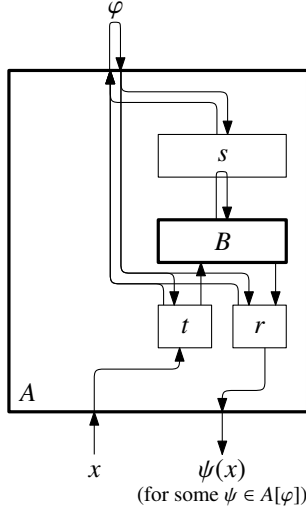
Figure 3: Reduction $A \leq^2_{\mathrm{mF}} B$.

Now that we have the classes (Definition 3.3) and reductions (Definition 3.5)[6], we can talk about completeness for these classes: we define $\mathsf{FPSPACE}^2$-completeness using the reduction $\leq^2_{\mathrm{mF}}$, and NP^2-completeness using $\leq^2_{\mathrm{m}}$.

The following lemma states that an $\mathsf{FPSPACE}^2$-complete multi-function maps some function in FP to an $\mathsf{FPSPACE}$-complete function with respect to the usual many-one reduction $\leq^1_{\mathrm{mF}}$. But since a multi-function has several values, we need the following definition: For a set F of (single-valued) functions from X to Y, we write $\bigcup F$ to mean the multi-function from X to Y defined by $(\bigcup F)[x] = \{ f(x) : f \in F \}$.

Lemma 3.6. *1. Let B be an $\mathsf{FPSPACE}^2$-complete multi-function from $\mathbf{Reg}$ to $\mathbf{Reg}$. Then there is $\psi \in \mathrm{dom}\, B \cap \mathsf{FP}$ such that $\bigcup(B[\psi])$ is $\mathsf{FPSPACE}$-complete (with respect to $\leq^1_{\mathrm{mF}}$).*

2. Let B be an NP^2-complete multi-function from $\mathbf{Reg}$ to $\mathbf{Pred}$. Then there is $\psi \in \mathrm{dom}\, B \cap \mathsf{FP}$ such that $\bigcup(B[\psi])$ is NP-complete (with respect to $\leq^1_{\mathrm{m}}$).

Proof. We only prove the first claim. There are a function $A \in \mathsf{FPSPACE}^2$ and a polynomial-time computable function $\varphi \in \mathbf{Reg}$ such that $A(\varphi)$ is $\mathsf{FPSPACE}$-complete. Since B is $\mathsf{FPSPACE}^2$-complete, there are total functions r, s, t as in Definition 3.5. Let $\psi = s(\varphi)$. Then $\psi \in \mathsf{FP}$. On the other hand, $\bigcup(B[\psi])$ is $\mathsf{FPSPACE}$-complete. To see why, note that $r(\varphi)$ and $t(\varphi)$ are in FP and, by Definition 3.5, give a many-one reduction from $A(\varphi)$ to $\bigcup(B[\psi])$ in the usual sense. Since $A(\varphi)$ is $\mathsf{FPSPACE}$-complete, so is $\bigcup(B[\psi])$. □

We define the pairing function for regular functions as follows: for φ, $\psi \in \mathbf{Reg}$, define $\langle \varphi, \psi \rangle \in \mathbf{Reg}$ by setting $\langle \varphi, \psi \rangle(0u) = \varphi(u)10^{|\psi(u)|}$ and $\langle \varphi, \psi \rangle(1u) = \psi(u)01^{|\varphi(u)|}$ (we are padding the string to make $\langle \varphi, \psi \rangle$ regular). Let $\langle \varphi, \psi, \theta \rangle = \langle \langle \varphi, \psi \rangle, \theta \rangle$, etc.

---

[6]The design of these reductions is somewhat arbitrary. We chose this definition simply because it is strong enough to make our examples (Theorems 4.6 and 4.9) complete. What Beame et al. [2] call the *many-one reduction* between type-2 problems is slightly stronger in that it passes the string input x not only to t (Figure 3) but also to r and s.

3.4 Representations

As we have moved from $\Sigma^{\mathbb{N}}$ to $\mathbf{Reg}$, we extend the notions of representations accordingly. A *representation* γ of a set X is a surjective partial function from $\mathbf{Reg}$ to X. Computation relative to representations is again formulated by Definition 2.1, now with the updated notion of representation. This defines what it means for a function F from X to Y, where X and Y are sets equipped with representations γ and δ, respectively, to be (γ, δ)-$\mathcal{C}$-computable, where $\mathcal{C}$ is one of the classes we have defined, such as FP^2 and $\mathsf{FPSPACE}^2$. This $\mathcal{C}$ can be P^2 or NP^2 if $\mathrm{dom}\,\delta \subseteq \mathbf{Pred}$. Also, we say that F is (γ, δ)-$\mathcal{C}$-complete (for $\mathcal{C} = \mathsf{FPSPACE}^2$, NP^2) if $\delta^{-1} \circ F \circ \gamma$ (see Definition 2.1) is $\mathcal{C}$-complete.

We say that an element $x \in X$ is γ-C-computable (where C is a usual complexity class of string functions, such as FP and $\mathsf{FPSPACE}$) if it has a γ-name in C. It is said to be γ-C-complete if $\bigcup(\gamma^{-1}[x])$ (where $\cdot^{-1}$ is the inverse image, and $\bigcup$ is defined before Lemma 3.6) is C-complete. Lemmas 3.4 and 3.6 yield:

Lemma 3.7. *Let γ and δ be representations of sets X and Y, respectively.*

1. A (γ, δ)-FP^2-computable function from X to Y maps γ-FP-computable elements of X to δ-FP-computable elements of Y. Similarly for $\mathsf{FPSPACE}^2$ and $\mathsf{FPSPACE}$ replacing FP^2 and FP.

2. Suppose that $\mathrm{dom}\,\delta \subseteq \mathbf{Pred}$. Then a (γ, δ)-P^2-computable function from X to Y maps γ-FP-computable elements of X to δ-P-computable elements of Y. Similarly for NP^2 and NP replacing P^2 and P.

Lemma 3.8. *Let γ and δ be representations of sets X and Y, respectively.*

1. A (γ, δ)-$\mathsf{FPSPACE}^2$-complete function from X to Y maps some γ-FP-computable element of X to a δ-$\mathsf{FPSPACE}$-complete element of Y.

2. Suppose that $\mathrm{dom}\,\delta \subseteq \mathbf{Pred}$. Then a (γ, δ)-NP^2-complete function from X to Y maps some γ-FP-computable element of X to a δ-NP-complete element of Y.

For representations γ_0 and γ_1 of X_0 and X_1, respectively, we can define the representation $[\gamma_0, \gamma_1]$ of the Cartesian product $X_0 \times X_1$ by $[\gamma_0, \gamma_1](\langle \varphi_0, \varphi_1 \rangle) = (\gamma_0(\varphi_0), \gamma_1(\varphi_1))$.

4. APPLICATIONS

As we noted in Section 3.1, our formulation can be viewed as a generalization of TTE obtained by removing the conditions (a) and (b) on the oracle that we use as names. In the following three subsections, we will apply our theory to real numbers, real sets and real functions through representations $\rho_{\mathbb{R}}$, ψ, and $\delta_\square$. These three representations exploit the removal of (a), (b), and both, respectively.

4.1 Computation over real numbers

Recall the representation $\rho_{\mathbb{R}}$ of $\mathbb{R}$ in TTE (Section 2.1) where a $\rho_{\mathbb{R}}$-name of a real number x was an infinite list $u_0 \# u_1 \# \ldots$ of (encodings of) rational numbers $[\![u_i]\!]$ satisfying $|[\![u_i]\!] - x| < 2^{-i}$. We modify this in a straightforward way to define a representation $\rho_{\mathbb{R}}$ that encodes real numbers into regular functions (we keep writing $\rho_{\mathbb{R}}$ by abuse of notation): We say that $\varphi \in \mathbf{Reg}$ is a $\rho_{\mathbb{R}}$-name of $x \in \mathbb{R}$ if $\varphi(0^i) \in \mathbf{D}$ and $|[\![\varphi(0^i)]\!] - x| < 2^{-i}$ for each $i \in \mathbb{N}$. Thus, we encode the same list into the values $\varphi(0^i)$. We write $\rho_{\mathbb{R}}|^{[0,1]}$ for the restriction of $\rho_{\mathbb{R}}$ to (names of) real numbers in the interval $[0, 1]$.

It turns out that $(\rho_{\mathbb{R}}|^{[0,1]}, \rho_{\mathbb{R}})$-$\mathsf{FP}^2$-computability coincides with the polynomial-time computability that was defined in Section 2.2.1 using the signed digit representation ρ_{sd}. Recall that in the definition of ρ_{sd}, we needed to forbid redundancy carefully so that the machine would not cheat by writing long names as output or suffer by reading long names as input. Observe how our formulation avoids these problems naturally: now we do not have to fret too much about defining concise representations.

Moreover, we obtain a reasonable notion of polynomial-time computability of real functions on $\mathbb{R}$ as well without additional work: $(\rho_{\mathbb{R}}, \rho_{\mathbb{R}})$-$\mathsf{FP}^2$-computability seems to be a reasonable notion and coincides with the one discussed by Hoover [6]. He extends the polynomial-time computability on $[0, 1]$ to that on $\mathbb{R}$ by requiring that the 2^{-m}-approximation of the value $f(t)$ of function $f : \mathbb{R} \to \mathbb{R}$ should be delivered within time polynomial in m and $\log|t|$. An equivalent definition appears also in Takeuti [18], independently inspired by Pour-El's approach to computable analysis.

Example 4.1. It is easy to verify that binary addition and multiplication on $\mathbb{R}$ can be $([\rho_{\mathbb{R}}, \rho_{\mathbb{R}}], \rho_{\mathbb{R}})$-$\mathsf{FP}^2$-computed by the algorithms suggested by Examples 2.2 and 2.3.

Example 4.2. The exponential function $\exp : \mathbb{R} \to \mathbb{R}$ restricted to $[0, 1]$ is $(\rho_{\mathbb{R}}|^{[0,1]}, \rho_{\mathbb{R}})$-$\mathsf{FP}^2$-computable, because $\exp t$ can be written as the sum of a series which is known to converge fast on $[0, 1]$ (that is, given a desired precision, the machine can tell how many initial terms it needs to compute). However, $\exp$ on the whole real line $\mathbb{R}$ is not $(\rho_{\mathbb{R}}, \rho_{\mathbb{R}})$-$\mathsf{FP}^2$-computable, because it grows too fast.

Example 4.3. The sine function $\sin : \mathbb{R} \to \mathbb{R}$ is $(\rho_{\mathbb{R}}, \rho_{\mathbb{R}})$-$\mathsf{FP}^2$-computable. To see this, note that just like $\exp$ in the previous example, $\sin$ is polynomial-time computable if restricted to, say, $[-4, 4]$. It is also possible, given $t \in \mathbb{R}$ as oracle and a desired precision, to find efficiently a number in $[-4, 4]$ which is close enough to t modulo 2π. We can compute $\sin t$ by combining these algorithms.

Example 4.4. A function can be $(\rho_{\mathbb{R}}, \rho_{\mathbb{R}})$-$\mathsf{FP}^2$-computable without even an explicit description known. The *trisector curves* between the points $(0, 1)$ and $(0, -1)$ in the plane are the pair of sets C_1, $C_2 \subseteq \mathbb{R}^2$ such that C_1 is the set of points equidistant from $(0, 1)$ and C_2, and C_2 is the set of points equidistant from $(0, -1)$ and C_1. Asano, Matoušek and Tokuyama [1] showed that such a pair (C_1, C_2) exists and is unique (see [9] for a simpler and more general proof), and that C_1 (as well as C_2, which is the mirror reflection) is a graph of a function $f : \mathbb{R} \to \mathbb{R}$ which is, in our terminology, $(\rho_{\mathbb{R}}, \rho_{\mathbb{R}})$-$\mathsf{FP}^2$-computable. Interestingly, they conjecture that the trisector curves are different from any curves that were previously known.

4.2 Computation over real sets

Let $\mathcal{A}$ be the set of closed subsets of $[0, 1]^2$. The operator *CH* of taking convex hulls (Section 2.2.3) is a function from $\mathcal{A}$ to $\mathcal{A}$. Define the representation ψ of $\mathcal{A}$ as follows[7]: let $\varphi \in \mathbf{Pred}$ be a ψ-name of $S \in \mathcal{A}$ if φ and S satisfy the two itemized conditions in Section 2.2.3. Since $\mathrm{dom}\,\psi \subseteq \mathbf{Pred}$, it makes sense to talk about ψ-NP-computability, (ψ, ψ)-NP^2-computability, etc. The following is immediate from the definition of polynomial-time computability in Section 2.2.3.

Lemma 4.5. *A set in $\mathcal{A}$ is (nondeterministic) polynomial-time computable if and only if it is ψ-P-computable (ψ-NP-computable).*

---

[7]Note that this representation is more succinct than the one that we would be able to define in TTE using infinite sequences [20, Example 6.9].

Now we can state and prove the following effectivized version of Theorems 2.6 and 2.7.

Theorem 4.6. *CH is (ψ, ψ)-NP^2-complete.*

The proof is a straightforward effectivization of those of Theorems 2.6 and 2.7, but we need to modify the hardness proof slightly, because, as we noted in footnote 2, Ko and Yu's original results were about a different notion of computability; this will be explained in the full version of this paper.

As corollaries of Theorem 4.6, we get Theorem 2.6 by Lemmas 3.7.2 and 4.5, and Theorem 2.7 by Lemmas 3.8.2 and 4.5.

4.3 Computation over real functions

For $\mu : \mathbb{N} \to \mathbb{N}$, define $\overline{\mu} \in \mathbf{Reg}$ by $\overline{\mu}(u) = 0^{\mu(|u|)}$. We say that μ is a *modulus of continuity* of a function $f \in \mathrm{C}[0, 1]$ if $|t - t'| \leq 2^{-\mu(n)}$ implies $|f(t) - f(t')| \leq 2^{-n}$ (for all $n \in \mathbb{N}$ and $t, t' \in [0, 1]$).

Define the representation $\delta_{\square}$ of $\mathrm{C}[0, 1]$ as follows: for $\mu : \mathbb{N} \to \mathbb{N}$ and $\varphi \in \mathbf{Reg}$, we set $\delta_{\square}(\langle\overline{\mu}, \varphi\rangle) = f$ if and only if μ is a modulus of continuity of f and for every $n \in \mathbb{N}$ and $u \in \mathbf{D}_{\mu(n)}$, we have $v := \varphi(u) \in \mathbf{D}_n$ and $|f(\llbracket u \rrbracket) - \llbracket v \rrbracket| < 2^{-n}$. This gives names to all functions in $\mathrm{C}[0, 1]$, because a continuous function on a compact interval is uniformly continuous.

Recall that the only reason that a real number can require long $\rho_{\mathbb{R}}$-names was having a large absolute value. In contrast, functions in $\mathrm{C}[0, 1]$ may have long $\delta_{\square}$-names for two possible reasons: having big values, and having a big modulus of continuity.

It is routine to prove the following.

Lemma 4.7. *A function in $\mathrm{C}[0, 1]$ is polynomial-time computable (resp. polynomial-space computable) if and only if it is $\delta_{\square}$-FP-computable (resp. $\delta_{\square}$-FPSPACE-computable).*

Lemma 4.8. *A function in $\mathrm{C}[0, 1]$ is PSPACE-complete in the sense of [8, Section 1.2] if it is $\delta_{\square}$-FPSPACE-complete.*

The above definitions and lemmas extend to compact domains other than $[0, 1]$ (we keep writing $\delta_{\square}$ by abuse of notation). To discuss the complexity of the operator *LipIVP* (Section 2.2.3), we define a representation $\delta_{\square\mathrm{L}}$ of the space $\mathrm{CL}[[0, 1] \times [-1, 1]]$ of Lipschitz continuous functions: Let $\delta_{\square\mathrm{L}}(\langle\varphi, \psi\rangle) = g$ if and only if φ is a $\delta_{\square}$-name of g and ψ is a constant function with value 0^L, where $L \in \mathbb{N}$ satisfies (5). Note that the Lipschitz constant is written in unary.

Now we can formulate the effectivized version of Theorems 2.8 and 2.9 as follows. The computability part can be verified by checking that the proof of Theorem 2.8 in [11] can be effectivized. For the hardness part, we need to modify slightly the construction in [8]; this will be worked out in the full version of this paper.

Theorem 4.9. *LipIVP is $(\delta_{\square\mathrm{L}}, \delta_{\square})$-$\mathsf{FPSPACE}^2$-complete.*

As corollaries, we have Theorem 2.8 by Lemmas 3.7.1 and 4.7, and Theorem 2.9 by Lemmas 3.8.1 and 4.8.

5. SUMMARY AND FUTURE WORK

- A TTE machine works on $\Sigma^{\mathbb{N}}$, the infinite strings. We replace $\Sigma^{\mathbb{N}}$ with $\mathbf{Reg}$, the regular functions. This is a generalization in two ways: regular functions (a) can have values of length greater than 1, and (b) take arguments written in binary.

- For time and space bounds we use second-order polynomials in the input size, which are defined in the way suggested by

type-two complexity theory. We define complexity classes P^2, NP^2 and FP^2, $\mathsf{FPSPACE}^2$. With a suitable notion of polynomial-time reductions, we can also define NP^2- and $\mathsf{FPSPACE}^2$-completeness. Formulating other classes is left for future work.

- To apply this to real problems, we introduced representations $\rho_\mathbb{R}$, ψ and $\delta_\square$ of real numbers, sets and functions. Both aspects (a) and (b) of our generalization turn out to be useful. With respect to these representations, we showed that taking the convex hull of a set is NP^2-complete, and that solving the Lipschitz continuous initial value problem is $\mathsf{FPSPACE}^2$-complete. These are effectivized versions of what have been known in ineffective forms, and tell us more about the hardness of numerical problems in practice. It is interesting to investigate which other known ineffective results about operators do (or do not) effectivize and how.

6. ACKNOWLEDGEMENTS

7. REFERENCES

[1] T. Asano, J. Matoušek, and T. Tokuyama. The distance trisector curve. *Adv. Math.*, 212(1):338–360, 2007.

[2] P. Beame, S. Cook, J. Edmonds, R. Impagliazzo, and T. Pitassi. The relative complexity of NP search problems. *J. Comput. Syst. Sci.*, 57(1):3–19, 1998.

[3] V. Brattka, P. Hertling, and K. Weihrauch. A tutorial on computable analysis. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *New Computational Paradigms: Changing Conceptions of What is Computable*, pages 425–491. Springer, 2008.

[4] M. Braverman. On the complexity of real functions. In *Proc. 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 155–164, 2005.

[5] A. Grzegorczyk. Computable functionals. *Fund. Math.*, 42:168–202, 1955.

[6] H. J. Hoover. Feasible real functions and arithmetic circuits. *SIAM J. Comput.*, 19(1):182–204, 1990.

[7] B. M. Kapron and S. A. Cook. A new characterization of type-2 feasibility. *SIAM J. Comput.*, 25(1):117–132, 1996.

[8] A. Kawamura. Lipschitz continuous ordinary differential equations are polynomial-space complete. *Comput. Complexity*, 2010, to appear.

[9] A. Kawamura, J. Matoušek, and T. Tokuyama. Zone diagrams in Euclidean spaces and in other normed spaces. In *Proc. 26th Annual ACM Symposium on Computational Geometry*, 2010, to appear.

[10] K. Ko and H. Friedman. Computational complexity of real functions. *Theoret. Comput. Sci.*, 20(3):323–352, 1982.

[11] K. Ko. On the computational complexity of ordinary differential equations. *Inform. Control*, 58:157–194, 1983.

[12] K. Ko. *Complexity Theory of Real Functions*. Birkhäuser Boston, 1991.

[13] K. Ko. On the computational complexity of integral equations. *Ann. Pure Appl. Log.*, 58(3):201–228, 1992.

[14] K. Ko. Polynomial-time computability in analysis. In Iurii Leonidovich Ershov et al., editors, *Handbook of Recursive Mathematics: Volume 2: Recursive Algebra, Analysis and Combinatorics*, vol. 139 of *Studies in Logic and the Foundations of Mathematics*, pages 1271–1317. North-Holland, 1998.

[15] K. Ko and F. Yu. On the complexity of convex hulls of subsets of the two-dimensional plane. In *Proc. 4th International Conference on Computability and Complexity in Analysis*, vol. 202 of *Electronic Notes in Theoretical Computer Science*, pages 121–135, 2008.

[16] K. Mehlhorn. Polynomial and abstract subrecursive classes. *J. Comput. Syst. Sci.*, 12(2):147–178, 1976.

[17] M. Schröder. Spaces allowing type-2 complexity theory revisited. *Math. Log. Q.*, 50(4-5):443–459, 2004.

[18] I. Takeuti. Effective fixed point theorem over a non-computably separable metric space. In Jens Blanck, Vasco Brattka, and Peter Hertling, editors, *Computability and Complexity in Analysis*, vol. 2064 of *Lecture Notes in Computer Science*, pages 310–322. Springer, 2001.

[19] K. Weihrauch. *Computable Analysis: An Introduction*. Texts in Theoretical Computer Science. Springer, 2000.

[20] K. Weihrauch. Computational complexity on computable metric spaces. *Math. Log. Q.*, 49(1):3–21, 2003.