

COMPLEXITY THEORY FOR OPERATORS IN ANALYSIS

AKITOSHI KAWAMURA AND STEPHEN COOK

ABSTRACT. We propose a new framework for discussing the computational complexity of problems involving uncountably many objects, such as real numbers, sets and functions, that can be represented only through 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 functions on 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 and show that the operator itself is polynomial-space complete.

1. INTRODUCTION

Computable analysis [3, 20] studies problems involving real numbers, sets and functions from the viewpoint of computability. Elements of uncountable sets (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 $f: [0, 1] \rightarrow \mathbb{R}$ 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, 19, 21]) 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

A short preliminary version of this work was presented at the 42nd ACM Symposium on Theory of Computing (STOC 2010).

takes real functions $f: [0, 1] \rightarrow \mathbb{R}$ to another real function $F(f)$. There are many positive and negative results [12] about such operators, but typically they are stated in the form

if f is in the 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 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], [21], 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 [7, 17]. 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 FSPACE) 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 [8, 11] 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.

Notation and terminology. A *multi-valued function* (or *multi-function*) F from set X to 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 $\text{dom } F$ 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 $x \in \text{dom } F$, we say that F is a *partial function*. When in addition $\text{dom } F = X$, 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 $x \in \text{dom } F$, it outputs *some* element of $F[x]$. The classes FP^2 and FPSPACE^2 that we will define later will also consist of multi-functions.

Note that we do not care what happens on inputs outside $\text{dom } F$, 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 $\text{dom } F$ 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 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 or space.

Throughout the paper, Σ^* denotes the set of finite strings over the alphabet Σ . We will tacitly assume that $\Sigma = \{0, 1\}$ in some contexts, and in other contexts that Σ contains all symbols appearing in the discussion.

Since we will apply our theory mainly to real numbers, it will be convenient to fix a dense subset of \mathbb{R} and its encoding. For each $n \in \mathbb{N}$, let \mathbf{D}_n denote the set of strings of the form

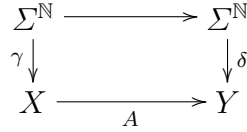
$$(1) \quad sx/1\underbrace{00\dots 0}_n,$$

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 number in the obvious sense—namely, read (1) as a fraction whose numerator and denominator are integers written in binary with leading zeros allowed. We write $\llbracket u \rrbracket$ for the number encoded by $u \in \mathbf{D}$. The numbers that can be encoded in this way are called *dyadic numbers*.

2. TYPE-TWO THEORY OF EFFECTIVITY

Computable Analysis dates back at least to Grzegorzczuk [5], and there have been several equivalent formulations. One powerful framework is Weihrauch’s Type-Two Theory of Effectivity (TTE). In this section, we briefly introduce Computable Analysis through TTE and clarify its limitations in dealing with complexity. We will address these limitations by modifying the computation model in Section 3.

2.1. Computability. In the usual Computability Theory we use some machine model that computes functions from Σ^* to Σ^* . 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 .

FIGURE 1. (γ, δ) -computing a multi-function A .

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 $a_0a_1 \dots \in \Sigma^{\mathbb{N}}$, 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 $b_0b_1 \dots \in \Sigma^{\mathbb{N}}$ if it never halts and writes the string indefinitely (that is, for each $n \in \mathbb{N}$, it eventually writes $b_0 \dots b_{n-1}$ 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 $\varphi \in \text{dom } A$, 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 . 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

$$(2) \quad (\delta^{-1} \circ A \circ \gamma)[\varphi] = \begin{cases} \{ \psi \in \text{dom } \delta : \delta(\psi) \in A[\gamma(\varphi)] \} & \text{if } \varphi \in \text{dom } \gamma, \\ \emptyset & \text{otherwise.} \end{cases}$$

In other words, whenever the machine is given a γ -name of an element $x \in \text{dom } A$, 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 the set \mathbb{R} of real numbers 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 $|\llbracket u_i \rrbracket - x| < 2^{-i}$ 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} [20, Lemma 4.1.6].

To deal with functions that take two arguments, we define, given representations γ and δ of sets X and Y , a new representation $[\gamma, \delta]$ of $X \times Y$ by $[\gamma, \delta](a_0b_0a_1b_1a_2b_2\dots) = (\gamma(a_0a_1a_2\dots), \delta(b_0b_1b_2\dots))$.

Example 2.2. Addition $+$: $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ 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 $\llbracket u_{m+1} \rrbracket + \llbracket v_{m+1} \rrbracket$.

Example 2.3. Multiplication \times : $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ 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 $\llbracket u_0 \rrbracket$ and $\llbracket v_0 \rrbracket$ 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 $\llbracket u_{m+k+1} \rrbracket \cdot \llbracket v_{m+k+1} \rrbracket$.

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 and 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 [20, Chapter 3].

2.2. Complexity. Now we ask whether the machine runs in polynomial time. This means whether it outputs the n th prefix of the output within time bounded polynomially in n (and 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 [20, 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} [20, 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

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

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 [20, 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 $f: [0, 1] \rightarrow \mathbb{R}$ to be *polynomial-time* $(\rho_{\text{sd}}|^{[0,1]}, \rho_{\text{sd}})$ -computable. This notion turns out to be robust 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 $+, \times: [0, 1] \times [0, 1] \rightarrow \mathbb{R}$ are polynomial-time computable. For more interesting results, see Ko’s book [12], survey [14] or Weihrauch’s book [20, 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 for which TTE gives reasonable notions of complexity are limited¹, 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], [21, 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 4.10).

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

¹We note, however, that Weihrauch [21] and Schröder [18] study how far we can get with the infinite string model.

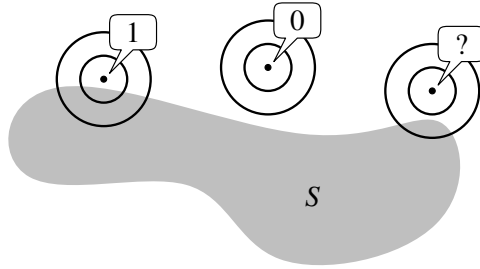


FIGURE 2. Computing a set S .

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 $\varphi: \Sigma^* \rightarrow \{0, 1\}$ such that, for any $n \in \mathbb{N}$ and $u, v \in \mathbf{D}$,

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

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

Theorem 2.6 (Essentially² in [15, Corollary 4.3]). *If a closed set $S \subseteq [0, 1]^2$ is polynomial-time computable, then $CH(S)$ is nondeterministic polynomial-time computable.*

Theorem 2.7 (Essentially² in [15, Corollary 4.6]). *Unless $\mathbf{P} = \mathbf{NP}$, there exists a closed 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

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

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. It is well known (see [8, beginning of Section 3]) that the solution h exists and is unique if g is *Lipschitz continuous* (in the second argument), that is,

$$(5) \quad |g(t, y_0) - g(t, y_1)| \leq L \cdot |y_0 - y_1|$$

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.

²Ko 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.

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

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

We can derive from Theorem 2.9 a statement of the form similar to Theorem 2.7:

Corollary 2.10 ([8, Corollary 3.3]). *Unless $\text{P} = \text{PSPACE}$, there is a real function $g \in \text{dom LipIVP}$ which is polynomial-time computable but $\text{LipIVP}(g)$ 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 functions⁴. 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.

In Section 3.1, we introduce the class **Reg** of regular functions and define what it means for a machine to compute a multi-function from **Reg** to **Reg**. Section 3.2 defines what it means for such a machine to run in polynomial time or space, thus introducing a few complexity classes of multi-functions on **Reg**. In Section 3.3, we define reductions between such multi-functions, and discuss the resulting notions of hardness. Section 3.4 extends this theory of computation on **Reg** to that on other sets X by using representations of X by **Reg**.

3.1. Computation over regular functions. We say that a (total) function $\varphi: \Sigma^* \rightarrow \Sigma^*$ is *regular* if it preserves relative lengths of strings in the sense that $|\varphi(u)| \leq |\varphi(v)|$ whenever $|u| \leq |v|$. We write **Reg** for the set of all regular functions. For the rest of this paper, we will be discussing the complexity of multi-functions from **Reg** to **Reg**. The motivation for considering regular functions (rather than all functions from Σ^* to Σ^*) will be explained 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, because an infinite string $a_0a_1 \dots \in \Sigma^{\mathbb{N}}$ 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 3.1.1 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

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

⁴Ko's formulation [12] already uses string functions instead of infinite strings of TTE, but it does not make full use of this extension, because in [12] queries to these functions are mostly unary strings 0^n only.

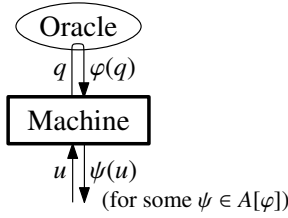


FIGURE 3. A deterministic machine computing a function A from **Reg** to **Reg**.

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 (Figure 3):

- Definition 3.1.** (1) A deterministic machine M computes a multi-function A from **Reg** to **Reg** if for any $\varphi \in \text{dom } A$, there is $\psi \in A[\varphi]$ such that M on oracle φ and any string u outputs $\psi(u)$ and halts.
- (2) A nondeterministic machine M computes a multi-function A from **Reg** to **Pred** if for any $\varphi \in \text{dom } A$, there is $\psi \in A[\varphi]$ such that $\psi(u) = 1$ if and only if M on oracle φ and string u has at least one accepting computation path.

For the precise conventions for issuing and answering queries, follow any of [7, 12, 17].

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 length to strings of equal length. Therefore it makes sense to define the size $|\varphi|: \mathbb{N} \rightarrow \mathbb{N}$ 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 L and type-0 variable n) are defined inductively as follows: a positive integer is a second-order polynomial; the variable n is also a second-order polynomial; if P and Q are second-order polynomials, then so are $P + Q$, $P \cdot Q$ and $L(P)$. An example is

$$(6) \quad L(L(n \cdot n)) + L(L(n) \cdot L(n)) + L(n) + 4.$$

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

$$(7) \quad P(L)(x) = ((x \cdot x)^2)^2 + (x^2 \cdot x^2)^2 + x^2 + 4 = x^8 + x^4 + x^2 + 4.$$

As in this example, $P(L)$ is a (usual first-order) polynomial if L 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) on page 8 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 is indeed an extension of Definition 2.4.

Definition 3.3. (1) We write \mathbf{FP}^2 (resp. $\mathbf{FPSPACE}^2$) for 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 \mathbf{P}^2 (resp. \mathbf{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 \mathbf{FP} and $\mathbf{FPSPACE}$, it is easy to separate, e.g., \mathbf{FP}^2 and $\mathbf{FPSPACE}^2$, because an $\mathbf{FPSPACE}^2$ 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 \mathbf{FP}^2 (resp. $\mathbf{FPSPACE}^2$) map regular functions in \mathbf{FP} into \mathbf{FP} (resp. $\mathbf{FPSPACE}$ into $\mathbf{FPSPACE}$).*
 (2) *Functions in \mathbf{P}^2 (resp. \mathbf{NP}^2) map regular functions in \mathbf{FP} into \mathbf{P} (resp. \mathbf{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 [17] of *polynomial-time computable operators*⁵. This is a class of (total) functionals $F: (\Sigma^* \rightarrow \Sigma^*) \times \Sigma^* \rightarrow \Sigma^*$, but they can be regarded as $F: (\Sigma^* \rightarrow \Sigma^*) \rightarrow (\Sigma^* \rightarrow \Sigma^*)$ by writing $F(\varphi)(x)$ instead of $F(\varphi, x)$. Kapron and Cook define the size of $\varphi: \Sigma^* \rightarrow \Sigma^*$ by

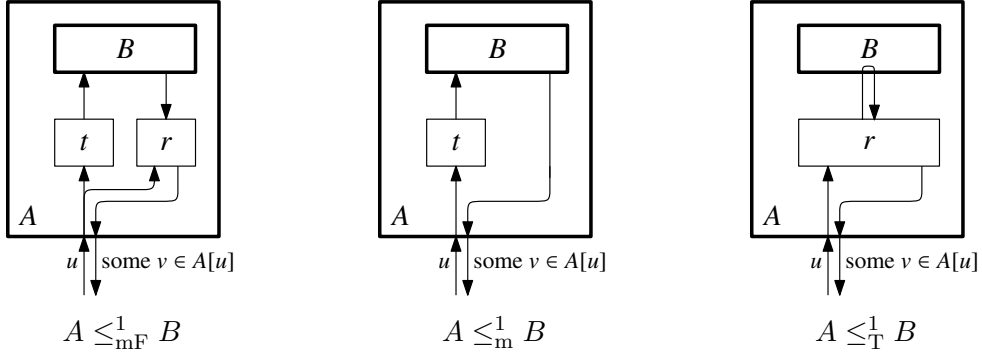
$$(8) \quad |\varphi|(n) = \max_{|u| \leq n} |\varphi(u)|, \quad n \in \mathbb{N}.$$

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

Imposing regularity is hardly a restriction for our purpose, because our intention is to use these functions as names of real numbers, sets and functions, and we can always declare that valid names are those that have been “padded” to be regular. More precisely, there is a polynomial-time machine that takes as oracles a possibly irregular function φ'

⁵Kapron and Cook [7] call them *Basic Feasible Functionals* or *Basic Polynomial-Time Functionals*.


 FIGURE 4. Reductions between multi-functions A, B on Σ^* .

and a regular function ψ dominating its length (i.e., $|\varphi'(u)| \leq |\psi|(|u|)$ for any string u), and delivers a regular function φ such that $|\varphi| = |\psi|$ and $\varphi(u) = \varphi'(u)\#\#\dots\#$. Thus we use φ , instead of φ' , as the name. In any situation where φ' can be computed from a machine whose time/space is polynomially bounded, so can φ , because in this case the time/space bound can serve as the length bound $|\psi|$.

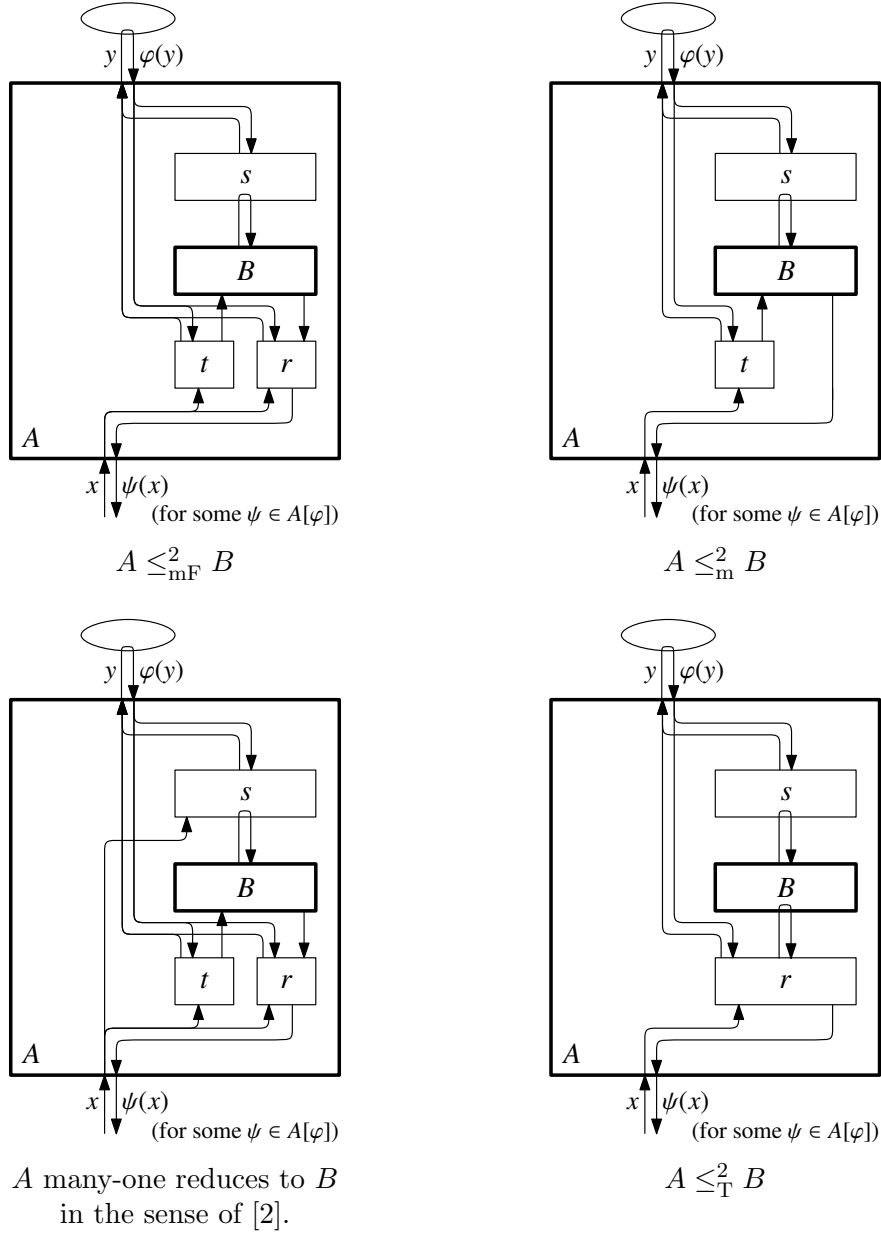
For later use we define the pairing function for regular functions as follows: for $\varphi, \psi \in \mathbf{Reg}$, define $\langle \varphi, \psi \rangle \in \mathbf{Reg}$ by setting $\langle \varphi, \psi \rangle(0u) = \varphi(u)0^{|\psi(u)|}$ and $\langle \varphi, \psi \rangle(1u) = \psi(u)1^{|\varphi(u)|}$ (we pad the strings to make $\langle \varphi, \psi \rangle$ regular). Let $\langle \varphi, \psi, \theta \rangle = \langle \langle \varphi, \psi \rangle, \theta \rangle$, etc.

3.3. Reduction and completeness. Here we define reductions between multi-functions on \mathbf{Reg} and discuss hardness with respect to these reductions.

3.3.1. Reductions. Recall that the usual many-one reduction between multi-functions A, B from Σ^* to Σ^* is defined as follows: we say that A many-one reduces to B (written $A \leq_{\text{mF}}^1 B$) if there are (total) functions $r, t \in \mathbf{FP}$ such that for any $u \in \text{dom } A$, we have $r(u, v) \in A[u]$ 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 (Figure 4, left). The many-one reduction \leq_{m}^1 between predicates is defined as the special case where we do not convert the output, i.e., $r(u, v) = v$ (Figure 4, middle). Since multi-functions over \mathbf{Reg} also get a function as input, the analogous definition of reductions 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 $A \leq_{\text{mF}}^2 B$) if there are functions $r, s, t \in \mathbf{FP}^2$ such that for any $\varphi \in \text{dom } A$, we have $s(\varphi) \in \text{dom } B$ and, for any $\theta \in B[s(\varphi)]$, the function that maps each string x to $r(\varphi)(x, \theta(t(\varphi)(x)))$ is in $A[\varphi]$ (Figure 5, top left).
- (2) Let A and B be multi-functions from \mathbf{Reg} to \mathbf{Pred} . We write $A \leq_{\text{m}}^2 B$ if there are functions $s, t \in \mathbf{FP}^2$ such that for any $\varphi \in \text{dom } A$, we have $s(\varphi) \in \text{dom } B$ and, for any $\theta \in B[s(\varphi)]$, the function $\theta \circ t(\varphi)$ is in $A[\varphi]$ (Figure 5, top right).

The design of these reductions is somewhat arbitrary. We chose these definitions simply because they are strong enough to make our examples (Theorems 4.6 and 4.10) complete

FIGURE 5. Reductions between multi-functions on **Reg**.

with respect to them. What Beame et al. [2] call the “many-one reduction” between type-2 problems is slightly stronger than our \leq_{mF}^2 in that it passes the string input x not only to t and r but also to s (Figure 5, bottom left). Another reasonable notion of reduction is the one on the bottom right of Figure 5: we write $A \leq_T^2 B$ if there are functions $r, s \in \text{FP}^2$ such that for any $\varphi \in \text{dom } A$, we have $r(\langle \varphi, \psi \rangle) \in A[\varphi]$ whenever $\psi \in B[s(\varphi)]$. Note that, while this is somewhat analogous to the standard Turing reduction (Figure 4, right), it also formally resembles the definition of \leq_{mF}^1 . The many-one reduction \leq_{mF}^2 is the special case of this reduction \leq_T^2 where r can query ψ only once. Beame et al. [2] have an even

stronger “Turing reduction”. In this paper, we will formulate our hardness results mainly using \leq_{mF}^2 and \leq_{m}^2 because they give stronger results than \leq_{T}^2 would. The advantage and disadvantage of this choice will be discussed in Section 4.3 after Theorem 4.10 and Corollary 4.11.

3.3.2. Hardness. Now that we have the classes (Definition 3.3) and reductions (Definition 3.5), we can talk about hardness. A multi-function B from **Reg** to **Reg** is FPSPACE^2 - \leq_{mF}^2 -hard if $A \leq_{\text{mF}}^2 B$ for every $A \in \text{FPSPACE}^2$. It is said to be FPSPACE^2 - \leq_{mF}^2 -complete if moreover it is in FPSPACE^2 . We define NP^2 - \leq_{m}^2 -hardness of multi-functions from **Reg** to **Pred** similarly (note that NP^2 is not closed under \leq_{mF}^2).

The following lemma states roughly that an FPSPACE^2 - \leq_{mF}^2 -complete multi-function maps some function in **FP** to an **FPSPACE**-complete function with respect to the usual many-one reduction \leq_{mF}^1 . But since a multi-function may have 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 FPSPACE^2 - \leq_{mF}^2 -complete multi-function from **Reg** to **Reg**. Then there is $\psi \in \text{dom } B \cap \text{FP}$ such that $\bigcup(B[\psi])$ is FPSPACE - \leq_{mF}^1 -complete.*
(2) *Let B be an NP^2 - \leq_{m}^2 -complete multi-function from **Reg** to **Pred**. Then there is $\psi \in \text{dom } B \cap \text{FP}$ such that $\bigcup(B[\psi])$ is NP - \leq_{m}^1 -complete.*

Proof. We only prove the first claim (the second claim is similar). There is a function $A \in \text{FPSPACE}^2$ that maps some function $\varphi \in \text{FP} \cap \text{Reg}$ to an FPSPACE - \leq_{mF}^1 -complete function. Since B is FPSPACE^2 - \leq_{mF}^2 -complete, there are functions $r, s, t \in \text{FP}^2$ as in Definition 3.5. By Lemma 3.4, we have $r(\varphi), s(\varphi), t(\varphi) \in \text{FP}$. Let $\psi = s(\varphi)$. Since $r(\varphi)$ and $t(\varphi)$ give a reduction $A(\varphi) \leq_{\text{mF}}^1 \bigcup(B[\psi])$, and $A(\varphi)$ is FPSPACE - \leq_{mF}^1 -complete, $\bigcup(B[\psi])$ is also FPSPACE - \leq_{mF}^1 -complete. \square

Saying that the function $\bigcup F$ is complete is stronger than the claim that the functions in F are each complete, in that the former requires that one reduction work for all functions in F . We need the stronger form in Lemma 3.6 in order to derive Lemma 3.12 later.

3.3.3. Some complete problems. Let PSPACE^2 be the subclass of FPSPACE^2 consisting of multi-functions from **Reg** to **Pred**. Here we list some NP^2 - and PSPACE^2 - \leq_{m}^2 -complete problems. The proofs of their completeness are all easily relativized versions of well-known NP - and PSPACE - \leq_{m}^1 -completeness.

We begin with NP^2 - \leq_{m}^2 -complete problems. For a non-decreasing function $\mu: \mathbb{N} \rightarrow \mathbb{N}$, define $\bar{\mu} \in \text{Reg}$ by $\bar{\mu}(u) = 0^{\mu(|u|)}$. A *boolean formula involving predicate symbol p* is an expression built up inductively from boolean variables a_1, a_2, \dots using the connectives $f_{i_1} \wedge f_{i_2}, f_{i_1} \vee f_{i_2}, \neg f_{i_1}$ and $p(f_{i_1}, \dots, f_{i_n})$ (the arity n can vary) for any previously obtained formulas f_{i_1}, f_{i_2}, \dots .

Lemma 3.7. *The following partial functions NTIME^2 , EXIST^2 and SAT^2 from **Reg** to **Pred** are NP^2 - \leq_{m}^2 -complete:*

- $\text{dom } \text{NTIME}^2$ consists of all triples $\langle M, \bar{\mu}, \varphi \rangle$ such that M is a (program of a) non-deterministic (oracle Turing) machine, $\mu: \mathbb{N} \rightarrow \mathbb{N}$ is non-decreasing, $\varphi \in \text{Reg}$, and for any string u , all computation paths of $M^\varphi(u)$ halt in time $\mu(|u|)$ (this M is a string, so we encode it as the constant function taking this string as value).

For any such triple and a string u , we have $\text{NTIME}^2(\langle M, \bar{\mu}, \varphi \rangle)(u) = 1$ if and only if $M^\varphi(u)$ has an accepting path.

- $\text{dom EXIST}^2 = \mathbf{Pred}$. For any $p \in \mathbf{Pred}$, $u \in \Sigma^*$ and $n \in \mathbb{N}$, we have $\text{EXIST}^2(p)(u, 0^n) = 1$ if and only if there is a string v of length n with $p(u, v) = 1$.
- $\text{dom SAT}^2 = \mathbf{Pred}$. For any $p \in \mathbf{Pred}$ and any string u , we have $\text{SAT}^2(p)(u) = 1$ if and only if u is a boolean formula involving a predicate symbol and it is made satisfiable when this predicate symbol is interpreted as p .

Proof. For NTIME^2 , let $A \in \mathbf{NP}^2$. It is computed by a nondeterministic machine M with time bound P (some second-order polynomial). To see that $A \leq_m^2 \text{NTIME}^2$, define the functions s and t of Definition 3.5 by $s(\varphi) = \langle M, \overline{P(|\varphi|)}, \varphi \rangle$ and $t(\varphi)(u) = u$.

To see that $\text{NTIME}^2 \leq_m^2 \text{EXIST}^2$, define the functions s and t of Definition 3.5 as follows. Let $s(\langle M, \bar{\mu}, \varphi \rangle)$ be the function that maps each pair (u, v) to 1 if v encodes an accepting (path of the) computation $M^\varphi(u)$ of length $\mu(|u|)$. Let $t(\langle M, \bar{\mu}, \varphi \rangle)(u)$ be the function that maps each string u to the pair $(u, 0^n)$, where n is a number such that any computation of length $\mu(|u|)$ on input u is encoded into a string of length n .

To see that $\text{EXIST}^2 \leq_m^2 \text{SAT}^2$, define the functions s and t of Definition 3.5 as follows. Let $s(p) = p$. Let $t(p)(u, 0^n)$ be the formula $p(u, a_1 a_2 \dots a_n)$, where the a_i are boolean variables. \square

Now we move on to PSPACE^2 - \leq_m^2 -complete problems. If φ_p is a boolean formula involving predicate symbol p , then an expression of the form

$$(9) \quad Q_1 a_1. Q_2 a_2. Q_3 a_3 \dots Q_k a_k. \varphi_p(a_1, \dots, a_k),$$

where each Q_i is either \forall or \exists , is called a *quantified boolean formula involving predicate symbol p* . Its truth value is determined in the obvious way relative to the predicate to be substituted into p .

Lemma 3.8. *The following partial functions SPACE^2 , POWER^2 , QBF^2 from \mathbf{Reg} to \mathbf{Pred} are PSPACE^2 - \leq_m^2 -complete:*

- dom SPACE^2 consists of all triples $\langle M, \bar{\mu}, \varphi \rangle$ such that M is a (program of a) deterministic (oracle Turing) machine, $\mu: \mathbb{N} \rightarrow \mathbb{N}$ is non-decreasing, $\varphi \in \mathbf{Reg}$, and for any string u , the computation $M^\varphi(u)$ uses no more than $\mu(|u|)$ tape cells and either accepts or rejects (this M is a string, so we encode it as the constant function taking this string as value). For any such triple and a string u , we have $\text{SPACE}^2(\langle M, \bar{\mu}, \varphi \rangle)(u) = 1$ if and only if $M^\varphi(u)$ accepts.
- dom POWER^2 consists of all $f \in \mathbf{Reg}$ that are length-preserving (i.e., $|f| = \text{id}$). For any such f and a string u , we have $\text{POWER}^2(f)(u) = 1$ if and only if $f^{2^{|u|}}(u) = 0^{|u|}$, where we write f^k for f iterated k times.
- $\text{dom QBF}^2 = \mathbf{Pred}$. For any $p \in \mathbf{Pred}$ and any string u , we have $\text{QBF}^2(p)(u) = 1$ if and only if u is a quantified boolean formula involving a predicate symbol and it is made true by p .

Proof. The proof for SPACE^2 is similar to that of the \mathbf{NP}^2 - \leq_m^2 -completeness of NTIME^2 .

We reduce SPACE^2 to POWER^2 . Fix a reasonable way to encode a configuration at any time in any computation of an oracle Turing machine into a string called the instantaneous description (ID). We may assume that if the computation uses at most k tape cells, then

each configuration can be encoded into an ID (called its space- k ID) whose length is exactly $l(k)$, where l is a strictly increasing polynomial. We may also assume that the accepting configuration is unique and is represented by the space- k ID $0^{l(k)}$.

We may assume that SPACE^2 is restricted to those triples $\langle M, \bar{\mu}, \varphi \rangle$ such that M^φ , on any input u , halts in exactly $2^{l(\mu(|u|))}$ steps. To show that this restricted SPACE^2 reduces to POWER^2 , we define the functions s and t of Definition 3.5 as follows. Define s by saying that $s(\langle M, \bar{\mu}, \varphi \rangle)$ is the length-preserving function that maps each ID of the machine M^φ to its successor ID (of the same length); if such a successor ID does not exist (this happens if M^φ has just used up too much space to have an ID of the same length), then the value is an ID corresponding to an irrevocable rejecting state. Define t by saying that $t(\langle M, \bar{\mu}, \varphi \rangle)(u)$ is the space- $\mu(|u|)$ initial ID of the machine M on input u .

We reduce POWER^2 to QBF^2 . For each length-preserving string function f , define $[f] \in \mathbf{Pred}$ by $[f](v, w) = 1$ if and only if $w = f(v)$. Note that $[f^2](v, w)$ can be written

$$(10) \quad \exists x. \forall y. \forall z. ((y, z) = (v, x) \vee (y, z) = (x, w)) \longrightarrow [f](y, z),$$

where x, y, z range over binary strings of the same length as v and w . Using this inductively, we can express $[f^{2^n}](v, w)$ as a quantified boolean formula involving $[f]$ whose length is bounded polynomially in $n, |v|, |w|$. Thus to give the functions s and t of Definition 3.5 for $\text{POWER}^2 \leq_m^2 \text{QBF}^2$, let $s(f) = [f]$ and let $t(f)$ be the formula that means $[f^{2^{|u|}}](u, 0^{|u|})$ relative to $[f]$. \square

3.4. Representations. As we have moved from $\Sigma^{\mathbb{N}}$ to \mathbf{Reg} , we extend the notion 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 multi-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 FPSPACE^2 . This \mathcal{C} can be P^2 or NP^2 if $\text{dom } \delta \subseteq \mathbf{Pred}$. Also, we say that F is (γ, δ) - \mathcal{C} - \leq -hard/complete (for $\mathcal{C} = \text{FPSPACE}^2, \text{NP}^2$) if $\delta^{-1} \circ F \circ \gamma$ (see Definition 2.1) is \mathcal{C} - \leq -hard/complete.

3.4.1. Translation and equivalence. Here we discuss how the above notions of (γ, δ) - \mathcal{C} -computability and (γ, δ) - \mathcal{C} - \leq -hardness depend on the choice of representations γ and δ . This material will be needed only in the discussion after Corollary 4.11 and can be skipped on a first reading.

For two representations δ and δ' of the same set, we write $\delta \leq_{\text{T}} \delta'$ if there is a function $F \in \text{FP}^2$ that *Turing translates* δ to δ' in the sense that for all $\varphi \in \text{dom } \delta$, we have $F(\varphi) \in \text{dom } \delta'$ and $\delta(\varphi) = \delta'(F(\varphi))$. Thus δ is “more informative” or “less generic” than δ' . It is easy to see the following.

Lemma 3.9. *Let \mathcal{C} be either FP^2 or FPSPACE^2 . Let γ and γ' be representations of a set A , and δ and δ' be representations of a set B . If $\gamma' \leq_{\text{T}} \gamma$ and $\delta \leq_{\text{T}} \delta'$, then a (γ, δ) - \mathcal{C} -computable multi-function is (γ', δ') - \mathcal{C} -computable.*

We write $\gamma \equiv_{\mathbf{T}} \gamma'$ if $\gamma \leq_{\mathbf{T}} \gamma'$ and $\gamma' \leq_{\mathbf{T}} \gamma$. Lemma 3.9 implies that the notion of (γ, δ) -FP²-computability is invariant under replacing γ or δ with $\equiv_{\mathbf{T}}$ -equivalent representations. This is the polynomial-time analogue of the similar observation about computably equivalent representations [20, Corollary 3.1.9]

When $\text{dom } \delta, \text{dom } \delta' \subseteq \mathbf{Pred}$, a similar statement to Lemma 3.9 holds for \mathbf{P}^2 , but not for \mathbf{NP}^2 . The statement for \mathbf{NP}^2 is true if we replace the assumption $\delta \leq_{\mathbf{T}} \delta'$ by a stricter (“many-one”) translation $\delta \leq_{\mathbf{m}} \delta'$, which means that there is a function $t \in \mathbf{FP}$ such that for any $\varphi \in \text{dom } \delta$, the function $\varphi' = \varphi \circ t$ satisfies $\varphi' \in \text{dom } \delta'$ and $\delta'(\varphi') = \delta(\varphi)$.

For the case $\mathcal{C} = \mathbf{FPSPACE}^2$ of Lemma 3.9, there is another direction to look in for an analogue: what if we replace $\mathbf{FPSPACE}^2$ -computability by $\mathbf{FPSPACE}^2 \leq_{\mathbf{mF}}^2$ -hardness? We should of course first reverse the directions of the translations between γ, γ' and δ, δ' in the assumption, but that is not enough, because of the “many-one” nature of the reduction $\leq_{\mathbf{mF}}^2$. One way to remedy this is to replace the assumption $\delta' \leq_{\mathbf{T}} \delta$ by a more restrictive kind of translation, but we leave it to the reader. Another way is to replace the reduction $\leq_{\mathbf{mF}}^2$ with the stronger reduction $\leq_{\mathbf{T}}^2$ mentioned after Definition 3.5:

Lemma 3.10. *Let γ and γ' be representations of a set A , and δ and δ' be representations of a set B . If $\gamma \leq_{\mathbf{T}} \gamma'$ and $\delta' \leq_{\mathbf{T}} \delta$, then a (γ, δ) - $\mathbf{FPSPACE}^2 \leq_{\mathbf{T}}^2$ -hard multi-function is (γ', δ') - $\mathbf{FPSPACE}^2 \leq_{\mathbf{T}}^2$ -hard.*

Thus the notion of (γ, δ) - $\mathbf{FPSPACE}^2 \leq_{\mathbf{T}}^2$ -completeness is invariant under replacing γ or δ by $\equiv_{\mathbf{T}}$ -equivalent representations.

3.4.2. Relation between effective and ineffective statements. We say that an element $x \in X$ is γ - C -computable (where C is a usual complexity class of string functions, such as \mathbf{FP} and $\mathbf{FPSPACE}$) if it has a γ -name in C . It is γ - $C \leq$ -complete (where \leq is either $\leq_{\mathbf{mF}}^1, \leq_{\mathbf{m}}^1$ or $\leq_{\mathbf{T}}^1$) if $\bigcup(\gamma^{-1}[x])$ (where \cdot^{-1} is the inverse image, and \bigcup is defined before Lemma 3.6) is $C \leq$ -complete. Lemmas 3.4 and 3.6 yield the following.

Lemma 3.11. *Let γ and δ be representations of sets X and Y , respectively.*

- (1) *A (γ, δ) -FP²-computable partial function F from X to Y maps γ -FP-computable elements of $\text{dom } F$ to δ -FP-computable elements of Y . Similarly for $\mathbf{FPSPACE}^2$ and $\mathbf{FPSPACE}$ replacing \mathbf{FP}^2 and \mathbf{FP} .*
- (2) *Suppose that $\text{dom } \delta \subseteq \mathbf{Pred}$. Then a (γ, δ) -P²-computable partial function F from X to Y maps γ -FP-computable elements of $\text{dom } F$ to δ -P-computable elements of Y . Similarly for \mathbf{NP}^2 and \mathbf{NP} replacing \mathbf{P}^2 and \mathbf{P} .*

Lemma 3.12. *Let γ and δ be representations of sets X and Y , respectively.*

- (1) *A (γ, δ) - $\mathbf{FPSPACE}^2 \leq_{\mathbf{mF}}^2$ -complete partial function from X to Y maps some γ -FP-computable element of X to a δ - $\mathbf{FPSPACE} \leq_{\mathbf{mF}}^1$ -complete element of Y .*
- (2) *Suppose that $\text{dom } \delta \subseteq \mathbf{Pred}$. Then a (γ, δ) - $\mathbf{NP}^2 \leq_{\mathbf{m}}^2$ -complete partial function from X to Y maps some γ -FP-computable element of X to a δ - $\mathbf{NP} \leq_{\mathbf{m}}^1$ -complete element of Y .*

These lemmas will be used in Sections 4.2 and 4.3 to derive the ineffective theorems from their effective counterparts.

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 δ_{\square} . These three representations exploit the removal of (a), (b), and both, respectively.

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 $[\gamma_0, \gamma_1](\langle \varphi_0, \varphi_1 \rangle) = (\gamma_0(\varphi_0), \gamma_1(\varphi_1))$.

4.1. 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 $u_0 \# u_1 \# \dots$ of (encodings of) rational numbers $\llbracket u_i \rrbracket$ with $|\llbracket u_i \rrbracket - x| < 2^{-i}$. 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 $|\llbracket \varphi(0^i) \rrbracket - x| < 2^{-i}$ 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 $(\rho_{\mathbb{R}}|^{[0,1]}, \rho_{\mathbb{R}})$ -FP²-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. Now we do not have to worry too much about defining concise representations.

Moreover, we obtain a reasonable notion of polynomial-time computability of real functions f on \mathbb{R} (not just $[0, 1]$) as well without additional work: $(\rho_{\mathbb{R}}, \rho_{\mathbb{R}})$ -FP²-computability turns out to be a reasonable notion that coincides with the one by Hoover [6], who required that the 2^{-m} -approximation of the value $f(t)$ should be delivered within time polynomial in m and $\log|t|$ (this equivalence was already essentially pointed out by Lambov [16]). Another equivalent definition appears also in Takeuti [19], independently inspired by Pour-El’s approach to computable analysis.

Example 4.1. It is easy to see that binary addition and multiplication on \mathbb{R} can be $([\rho_{\mathbb{R}}, \rho_{\mathbb{R}}], \rho_{\mathbb{R}})$ -FP²-computed by the algorithms suggested by Examples 2.2 and 2.3.

Example 4.2. The exponential function $\exp: \mathbb{R} \rightarrow \mathbb{R}$ restricted to $[0, 1]$ is $(\rho_{\mathbb{R}}|^{[0,1]}, \rho_{\mathbb{R}})$ -FP²-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 $(\rho_{\mathbb{R}}, \rho_{\mathbb{R}})$ -FP²-computable, because it grows too fast.

Example 4.3. The sine function $\sin: \mathbb{R} \rightarrow \mathbb{R}$ is $(\rho_{\mathbb{R}}, \rho_{\mathbb{R}})$ -FP²-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 $(\rho_{\mathbb{R}}, \rho_{\mathbb{R}})$ -FP²-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 $f: \mathbb{R} \rightarrow \mathbb{R}$ which is, in our terminology, $(\rho_{\mathbb{R}}, \rho_{\mathbb{R}})$ - 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} .

4.2.1. Representation of real sets. Define the representation ψ of \mathcal{A} as follows: let $\varphi \in \mathbf{Pred}$ be a ψ -name of $S \in \mathcal{A}$ if it satisfies the two itemized conditions in Section 2.2.3. Note that this representation is more succinct than the one that we would be able to define in TTE using infinite sequences [21, Example 6.9].

Since $\text{dom } \psi \subseteq \mathbf{Pred}$, it makes sense to talk about ψ -NP-computability, (ψ, ψ) - NP^2 -computability, etc. (Section 3.4.2). 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).*

4.2.2. Complexity of the convex hull operator. 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 - \leq_m^2 -complete.*

As corollaries of Theorem 4.6, we get Theorem 2.6 by Lemmas 3.11.2 and 4.5, and Theorem 2.7 by Lemmas 3.12.2 and 4.5.

Proof of Theorem 4.6. The main technical ideas are already in Ko and Yu's proof of the corresponding ineffective versions (Theorems 2.6 and 2.7), so we will only sketch the proof.

The (ψ, ψ) - NP^2 -computability is no surprise: A point p belongs to $CH(S)$ if there are two points p' and p'' in S such that p is on the line segment $p'p''$. All we need is to guess p' and p'' using nondeterminism, with appropriate consideration of precision.

For hardness, we need to modify the proof slightly, because, as we noted in footnote 2, Ko and Yu's original results were about a different notion of computability: our computability of sets is more demanding in the sense that on query $(u, v, 0^n)$, where $u, v \in \mathbf{D}$ and $n \in \mathbb{N}$, if $(\llbracket u \rrbracket, \llbracket v \rrbracket)$ is within distance 2^{-n} from the set, then we must say 1 (see the definition before Theorems 2.6 and 2.7), whereas for weak computability both 0 or 1 are allowed in this case.

We assume that the reader has Ko and Yu's proof [15, Corollary 4.6] at hand. The proof of their Lemma 4.4 begins by taking an arbitrary set $A \in \mathbf{NP}$ and noting that there are $B \in \mathbf{P}$ and a polynomial p such that $w \in A$ if and only if $(w, u) \in B$ for some string u of length exactly $p(|w|)$. Relativizing this, we take $A \in \mathbf{NP}^2$, and note that there are $B \in \mathbf{P}^2$ and a second-order polynomial P such that $A(\varphi)(w) = 1$ if and only if $B(\varphi)(w, u) = 1$ for some string u of length $P(|\varphi|)(|w|)$.

We need to provide s and t of Definition 3.5. We define s by describing the set $S = \psi(s(\varphi))$ for a given $\varphi \in \mathbf{Reg}$. The construction of S is similar to that of the original proof, replacing $p(n)$ by $P(|\varphi|)(n)$ and B by $B(\varphi)$. But we need to modify one part to get $s \in \mathbf{P}^2$.

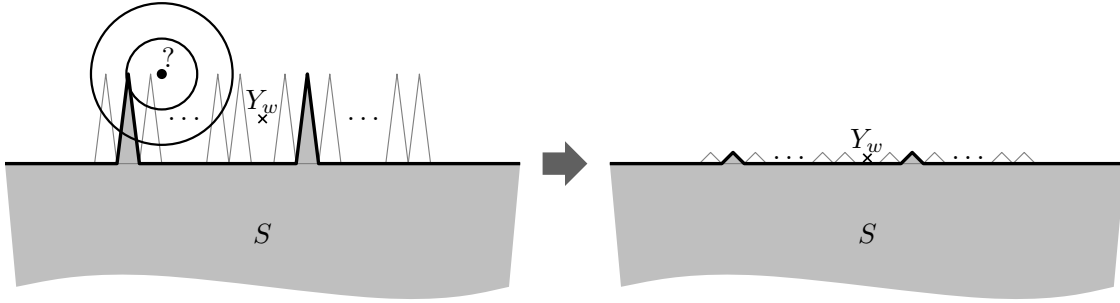


FIGURE 6. Widget for reducing NP^2 to CH . We have $Y_w \in CH(S)$ if and only if there is u such that the slot for $\langle w, u \rangle$ has a bump. In Ko and Yu’s construction of S , the bumps can be high (left), and there can be a query that requires the knowledge of $B(\langle w, u \rangle)$ for many u . We make the bumps low (right) in order to make S polynomial-time computable in our sense.

The original proof constructs a widget depicted in Figure 6 left (or Figure 2 of [15]) for each string w . In each of the left and right halves, there are exponentially many slots, one for each u , that has a bump if and only if (w, u) is in B (or $B(\varphi)$ for us). The point of this construction is that, while the set S is easy to compute, $CH(S)$ is hard in the sense that we can tell if w is in A (or $A(\varphi)$) by checking whether the middle point Y_w belongs to $CH(S)$.

However S is not easy in our sense, because in order to answer the query shown in Figure 6, we need to know $B(\varphi)(w, u)$ for exponentially many u . To avoid this, we make the bumps low, so they make an angle of at most 45° (Figure 6 right). This ensures that any one query to the ψ -name of S can be answered by checking $B(\varphi)(w, u)$ for at most one (w, u) . Thus a name of S can be P^2 -computed from φ .

The function t of Definition 3.5 queries whether the point Y_w is in $CH(S)$ with appropriate precision. Note that t needs access to φ in order to find the location of Y_w and the necessary precision. \square

4.3. Computation over real functions.

4.3.1. *Representation of real functions.* We say that a non-decreasing function $\mu: \mathbb{N} \rightarrow \mathbb{N}$ is a *modulus of continuity* of a function $f \in C[0, 1]$ if for all $n \in \mathbb{N}$ and $t, t' \in [0, 1]$ such that $|t - t'| \leq 2^{-\mu(n)}$, we have $|f(t) - f(t')| \leq 2^{-n}$ (Figure 7). Note that any $f \in C[0, 1]$ is uniformly continuous and thus has a modulus of continuity.

Define the representation δ_\square of $C[0, 1]$ as follows (see Lemmas 4.7 and 4.9 below for the reasons why we believe δ_\square to be a natural representation): for $\mu: \mathbb{N} \rightarrow \mathbb{N}$ and $\varphi \in \mathbf{Reg}$, we set $\delta_\square(\langle \bar{\mu}, \varphi \rangle) = f$ if and only if μ is a modulus of continuity of f and for every $n \in \mathbb{N}$ and $u \in \mathbf{D}$, we have $v := \varphi(0^n, u) \in \mathbf{D}$ and $|f(\llbracket u \rrbracket) - \llbracket v \rrbracket| < 2^{-n}$ (the string v may have to have leading 0s padded in order to make φ regular—but this need for padding is inconsequential, see the penultimate paragraph of Section 3.2; in what follows, we sometimes omit this padding in the description of algorithms). To see that $\delta_\square(\varphi)$ is well-defined, suppose that the above condition holds for two real functions f and f' . Let

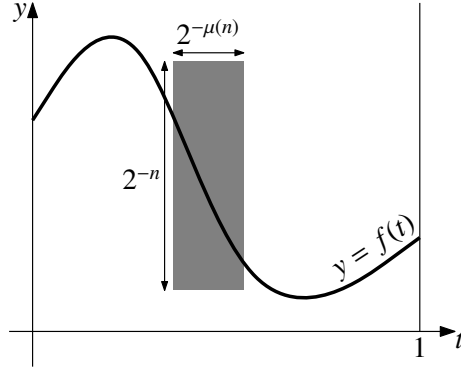


FIGURE 7. Modulus of continuity μ of a real function $f \in C[0, 1]$.

$t \in [0, 1]$ be arbitrary. Then for each $n \in \mathbb{N}$, there is $u \in \mathbf{D}$ with $|t - \llbracket u \rrbracket| \leq 2^{-\mu(n)}$ and thus

$$\begin{aligned}
 (11) \quad |f(t) - f'(t)| &\leq |f(t) - f(\llbracket u \rrbracket)| + |f(\llbracket u \rrbracket) - \llbracket \varphi(0^n, u) \rrbracket| \\
 &\quad + |f'(\llbracket u \rrbracket) - \llbracket \varphi(0^n, u) \rrbracket| + |f'(t) - f'(\llbracket u \rrbracket)| \\
 &\leq 2^{-n} + 2^{-n} + 2^{-n} + 2^{-n} = 2^{-n+2}.
 \end{aligned}$$

Since $n \in \mathbb{N}$ was arbitrary, $f(t) = f'(t)$. Since $t \in [0, 1]$ was arbitrary, $f = f'$.

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 $C[0, 1]$ may have long δ_{\square} -names for two possible reasons: having big values, and having a big modulus of continuity.

The following lemma says that the complexity of δ_{\square} -names of $f \in C[0, 1]$ matches the complexity of f that was discussed in Section 4.1:

Lemma 4.7 ([12, Corollary 2.21]). *A function in $C[0, 1]$ is polynomial-time (resp. polynomial-space) computable if and only if it is δ_{\square} -FP-computable (resp. δ_{\square} -FPSPACE-computable).*

Lemma 4.8. *A function in $C[0, 1]$ is PSPACE-complete in the sense of [8, Section 1.2] if it is δ_{\square} -FPSPACE- \leq_{mF}^1 -complete and has a polynomial modulus of continuity.*

Proof. Suppose that $f \in C[0, 1]$ is δ_{\square} -FPSPACE- \leq_{mF}^1 -complete and has a polynomial modulus of continuity μ . Then for any $A \in \text{PSPACE}$ there are polynomial-time functions t and r that satisfy the left picture of Figure 4 for any $B \in \delta_{\square}^{-1}[f]$ —and thus for any B of the form $\langle \bar{\mu}, \varphi \rangle$ (that is, those which have this particular polynomial μ as the first component). A query to B can ask either a value of μ or a value of φ , but μ is just a polynomial, so we may assume that t only asks a query of form “ $\varphi(0^n, v)$?”. Thus given a quantified boolean formula u , its truth value can be computed by r from a 2^{-n} -approximation of $f(\llbracket v \rrbracket)$. This implies that f is PSPACE-complete in the sense of [8]. \square

The representation δ_{\square} of $C[0, 1]$ may look somewhat arbitrary at first sight. Here we show a property of δ_{\square} that seems to make it a “natural” representation. Define the function $Apply: C[0, 1] \times [0, 1] \rightarrow \mathbb{R}$ by $Apply(f, x) = f(x)$. The following lemma says that (the \equiv_{T} -equivalence class of) the representation δ_{\square} is the most generic representation of

$C[0, 1]$ that makes *Apply* efficiently computable (see Section 3.4.1 for the definitions of \leq_T and \equiv_T).

Lemma 4.9. *Let δ be any representation of $C[0, 1]$. Then *Apply* is $([\delta, \rho_{\mathbb{R}}|^{[0,1]}], \rho_{\mathbb{R}})$ -FP²-computable if and only if $\delta \leq_T \delta_{\square}$.*

Proof. For the “if” part, it suffices to prove that *Apply* is $([\delta_{\square}, \rho_{\mathbb{R}}|^{[0,1]}], \rho_{\mathbb{R}})$ -FP²-computable. Let a δ_{\square} -name $\langle \bar{\mu}, \varphi \rangle$ of $f \in C[0, 1]$ and a $\rho_{\mathbb{R}}|^{[0,1]}$ -name θ of $t \in [0, 1]$ be given. We get a $\rho_{\mathbb{R}}$ -name ξ of $f(t)$ by computing $\xi(0^n)$ (that is, a 2^{-n} -approximation of $f(t)$) as follows. First read $m := \mu(n+1)$ to see how precisely we need to know t in order to determine $f(t)$ within error bound 2^{-n-1} . Then read $u := \theta(0^m)$ to get a 2^{-m} -approximation $\llbracket u \rrbracket$ of t . Because μ is a modulus of continuity, $|f(\llbracket u \rrbracket) - f(t)| < 2^{-n-1}$. Finally, read $v := \varphi(0^{n+1}, u)$. Then $\llbracket v \rrbracket$ is a 2^{-n-1} -approximation of $f(\llbracket u \rrbracket)$, and hence a 2^{-n} -approximation of $f(t)$. All this can be done in second-order polynomial time in μ and $|\varphi|$.

For the other direction, suppose that *Apply* is $([\delta, \rho_{\mathbb{R}}|^{[0,1]}], \rho_{\mathbb{R}})$ -FP²-computable. That is, there is a second-order polynomial P such that given a δ -name ψ of a function $f \in C[0, 1]$, a $\rho_{\mathbb{R}}|^{[0,1]}$ -name θ of a number $t \in [0, 1]$, and a unary string 0^n , it is possible to compute a $\rho_{\mathbb{R}}$ -name of $f(t)$ in time $P(|\psi|, |\theta|)(n)$. We need to translate δ to δ_{\square} . Let a δ -name ψ of $f \in C[0, 1]$ be given. To get a δ_{\square} -name $\langle \bar{\mu}, \varphi \rangle$ of f , we do as follows. Let $\mu = P(|\psi|, \lambda)$, where λ is a (usual) polynomial such that any number in $[0, 1]$ has a $\rho_{\mathbb{R}}|^{[0,1]}$ -name of size λ ; for example, $\lambda(n) = 2n+4$ will do, because every number in $[0, 1]$ has a 2^{-n} -approximation of form (1) where x has length $n+1$. Then μ is a modulus of continuity of f , because a 2^{-n} -approximation of $f(t)$, where t is given as a $\rho_{\mathbb{R}}|^{[0,1]}$ -name θ of size λ , can be computed within time bound $P(|\psi|, \lambda)(n)$ and hence without reference to θ 's values on strings longer than this bound. The other part φ of the δ_{\square} -name is such that $\varphi(0^n, u)$, where $u \in \mathbf{D}$, is a 2^{-n} -approximation of $f(\llbracket u \rrbracket)$. This can be computed from ψ , 0^n , u as follows. Let $\theta \in \mathbf{Reg}$ be the constant function taking value u . Note that θ is a $\rho_{\mathbb{R}}$ -name of $\llbracket u \rrbracket$. We then run the assumed algorithm for *Apply* on oracles ψ and θ and string 0^n . This takes time $P(|\psi|, |\theta|)(n)$, which is polynomial in $|\psi|$ and $|u|$, n . \square

An analogous fact has been known for computability (without a time bound): let \leq and \equiv be the analogues of \leq_T and \equiv_T without the time bound [20, Definition 2.3.2]; then *Apply* is $([\delta, \rho_{\mathbb{R}}|^{[0,1]}], \rho_{\mathbb{R}})$ -computable if and only if $\delta \leq \delta_{\square}$. This is stated as [20, Lemma 3.3.14] for the representation $[\rho_{\mathbb{R}}|^{[0,1]} \rightarrow \rho_{\mathbb{R}}]_{[0,1]}$ instead of our δ_{\square} , but it is \equiv -equivalent to δ_{\square} .

The above definitions and lemmas extend to compact domains other than $[0, 1]$ (we keep writing δ_{\square} by abuse of notation). To discuss the complexity of the operator *LipIVP* (Section 2.2.3), we define a representation $\delta_{\square, L}$ of the space $\text{CL}[[0, 1] \times [-1, 1]]$ of Lipschitz continuous functions by setting $\delta_{\square, L}(\langle \varphi, 0^L \rangle) = g$ if and only if φ is a δ_{\square} -name of g and $L \in \mathbb{N}$ satisfies (5) (regard the string 0^L as the constant function whose value is 0^L).

4.3.2. *Complexity of the operator of solving differential equations.* Now we can formulate the effectivized version of Theorems 2.8 and 2.9 as follows (a proof will be given shortly).

Theorem 4.10. *LipIVP is $(\delta_{\square, L}, \delta_{\square})$ -FPSPACE²- \leq_{mF}^2 -complete.*

As corollaries, we have Theorem 2.8 by Lemmas 3.11.1 and 4.7, and Theorem 2.9 by Lemmas 3.12.1 and 4.8.

Recall the alternative reduction $\leq_{\mathbf{T}}^2$ mentioned after Definition 3.5. The following weaker version of Theorem 4.10 is slightly easier to prove (see the end of the section):

Corollary 4.11. *LipIVP is $(\delta_{\square L}, \delta_{\square})$ -FPSPACE²- $\leq_{\mathbf{T}}^2$ -complete.*

As we noted in Lemma 3.10, this is a more robust result in the sense that it is invariant under replacing representations to $\equiv_{\mathbf{T}}$ -equivalent ones. This seems to be especially nice in view of Lemma 4.9. A drawback is that Corollary 4.11 does not directly yield Theorem 2.9, because Lemma 4.8 requires FPSPACE- \leq_{mF}^1 -completeness, whereas replacing \leq_{mF}^2 by $\leq_{\mathbf{T}}^2$ in the assumption of 3.12.1 also changes \leq_{mF}^1 to $\leq_{\mathbf{T}}^1$ in the conclusion. We can still obtain Corollary 2.10.

The rest of the section is devoted to the proof of Theorem 4.10. The positive part ($(\delta_{\square L}, \delta_{\square})$ -FPSPACE²-computability) will be verified by checking that the proof of Theorem 2.8 can be effectivized. For the hardness part, we need to modify slightly the construction in the original proof of Theorem 2.9 in order to get Theorem 4.10 (this modification is not needed if we only want Corollary 4.11). We begin with the positive part.

Proof of the positive part of Theorem 4.10. Given a $\delta_{\square L}$ -name $\langle \bar{\mu}, \varphi, 0^L \rangle$ of g , we need to find a δ_{\square} -name $\langle \bar{v}, \psi \rangle$ of $h = \text{LipIVP}(g)$. Recall that μ is a modulus of continuity of g , and φ gives approximations of the values g at dyadic points, that is,

$$(12) \quad |[\varphi(0^q, u, v)] - f([u], [v])| < 2^{-q}$$

for each $u, v \in \mathbf{D}$ (such that $([u], [v]) \in [0, 1] \times [-1, 1]$).

It is easy to find a modulus of continuity ν of h : let $\nu(n) = n + M$, where $M \in \mathbb{N}$ is any number such that 2^M bounds the maximum absolute value of g . For example, $M = \lceil \log_2(|[\varphi(\varepsilon, +0/1, +0/1)]| + 1 + 2^{\mu(0)}) \rceil$ will do.

To obtain the ψ part of a δ_{\square} -name $\langle \bar{v}, \psi \rangle$ of h , we apply the forward Euler method with step size 2^{-p} to the approximation of g with precision 2^{-q} (we will specify p and q shortly). That is, we define an approximation $\tilde{h}_{p,q} \in C[0, 1]$ of h by letting $\tilde{h}_{p,q}(0) = 0$ and then saying, for each $T = 0, \dots, 2^p - 1$ inductively, that $\tilde{h}_{p,q}(0) = 0$ is linear on the interval $[2^{-p}T, 2^{-p}(T+1)]$, with slope approximately $g(2^{-p}T, h(2^{-p}T))$: formally,

$$(13) \quad \tilde{h}_{p,q}(2^{-p}T + \Delta t) = \tilde{h}_{p,q}(2^{-p}T) + \Delta t[\varphi(0^q, u, v)], \quad 0 \leq \Delta t \leq 2^{-p},$$

for some $u, v \in \mathbf{D}$ with $[u] = 2^{-p}T$ and $[v] = \tilde{h}_{p,q}(2^{-p}T)$. Obviously, we can compute such a function $\tilde{h}_{p,q}$ in space polynomial in p and q in the sense that there is $Euler \in \text{FPSPACE}^2$ such that $[Euler(\varphi)(0^p, 0^q, u)] = \tilde{h}_{p,q}([u])$ for every $u \in \mathbf{D}$.

Let $\psi(0^n, u) = Euler(\varphi)(0^p, 0^q, u)$, where $p = \max\{\mu(n + 8L), n + 8L + M\}$ and $q = n + 8L$. We claim that $\langle \bar{v}, \psi \rangle$ is a δ_{\square} -name of h (note that this shows the desired FPSPACE²-computability, since p and q are bounded polynomially in $|\varphi|$, μ and n, L). This means that $|\tilde{h}_{p,q}(t) - h(t)| \leq 2^{-n}$ for any $t \in [0, 1]$. More strongly, we prove, by induction on $T = 0, \dots, 2^p - 1$, that

$$(14) \quad |\tilde{h}_{p,q}(t) - h(t)| \leq 2^{-n} e^{4L(t-1)}$$

for all $t \in [2^{-p}T, 2^{-p}(T+1)]$. We may assume (14) for $t = 2^{-p}$ as the induction hypothesis. The approximate value $\tilde{h}_{p,q}(2^{-p}T + \Delta t)$ is defined by (13), whereas the true solution h

satisfies

$$(15) \quad h(2^{-p}T + \Delta t) = h(2^{-p}T) + \int_{2^{-p}T}^{2^{-p}T + \Delta t} g(\tau, h(\tau)) \, d\tau.$$

The error added by this approximation is

$$(16) \quad \left| \Delta t \llbracket \varphi(0^q, u, v) \rrbracket - \int_{2^{-p}T}^{2^{-p}T + \Delta t} g(\tau, h(\tau)) \, d\tau \right| \leq 4L2^{-n} e^{4L(2^{-p}T-1)} \Delta t,$$

because

$$(17) \quad \begin{aligned} & \left| \llbracket \varphi(0^q, u, v) \rrbracket - g(\tau, h(\tau)) \right| \\ & \leq \left| \llbracket \varphi(0^q, u, v) \rrbracket - g(\llbracket u \rrbracket, \llbracket v \rrbracket) \right| + \left| g(\llbracket u \rrbracket, \llbracket v \rrbracket) - g(\tau, \llbracket v \rrbracket) \right| + \left| g(\tau, \llbracket v \rrbracket) - g(\tau, h(\tau)) \right| \\ & \leq 2^{-q} + 2^{-n-8L} + L \left| \llbracket v \rrbracket - h(\tau) \right| \\ & \leq 2^{-n-8L} + 2^{-n-8L} + L \left(\left| \llbracket v \rrbracket - h(2^{-p}T) \right| + \left| h(2^{-p}T) - h(\tau) \right| \right) \\ & \leq 2^{-n-8L} + 2^{-n-8L} + L \left(2^{-n} e^{4L(2^{-p}T-1)} + 2^{-p} 2^M \right) \\ & \leq L \left(2^{-n-8L} + 2^{-n-8L} + 2^{-n} e^{4L(2^{-p}T-1)} + 2^{-n-8L} \right) \leq 4L2^{-n} e^{4L(2^{-p}T-1)}, \end{aligned}$$

where the second, third and fifth inequalities come from $p \geq \mu(n + 8L)$, $q \geq n + 8L$, $p \geq M + n + 8L$, respectively. Using (16) and the induction hypothesis, we compare (13) and (15) to obtain

$$(18) \quad \begin{aligned} \left| \tilde{h}_{p,q}(2^{-p}T + \Delta t) - h(2^{-p}T + \Delta t) \right| & \leq 2^{-n} e^{4L(2^{-p}T-1)} + 4L2^{-n} e^{4L(2^{-p}T-1)} \Delta t \\ & = 2^{-n} e^{4L(2^{-p}T-1)} (1 + 4L\Delta t) \leq 2^{-n} e^{4L(2^{-p}T-1)} e^{4L\Delta t} = 2^{-n} e^{4L(2^{-p}T-1+\Delta t)}, \end{aligned}$$

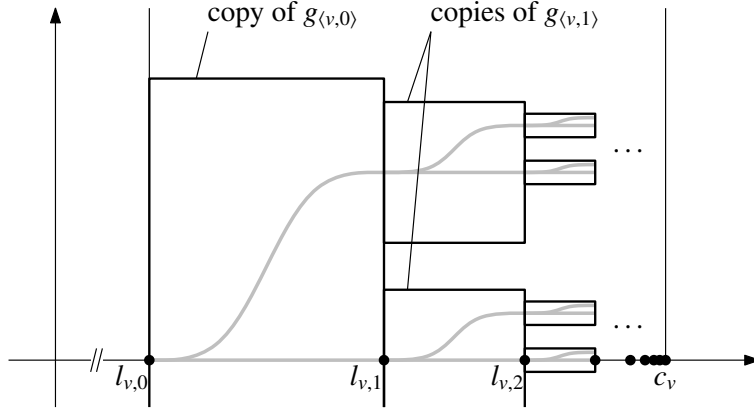
as desired. \square

For the hardness, the core part of the proof can be done by relativizing the argument for the ineffective version [8]. Since the proof was by reduction from the problem QBF (quantified boolean formulas), we use the relativized version QBF² from Lemma 3.8. Starting from this QBF², we follow the construction for [8, Lemma 4.1], which effectivizes and yields:

Lemma 4.12. *Let $L \in \text{PSPACE}^2$. Then there are a second-order polynomial P and a function $G \in \text{FP}^2$ such that the following holds for each $\varphi \in \text{dom } L$, $\lambda: \mathbb{N} \rightarrow \mathbb{N}$ and $u \in \Sigma^*$. Consider the partial function from **Reg** to **Reg** whose domain is $\text{dom}[\rho_{\mathbb{R}}|^{[0,1]}, \rho_{\mathbb{R}}|^{[-1,1]}]$ and which maps each θ in it to the function that maps each string v to $G(\varphi, \bar{\lambda}, \theta)(u, v)$. Then this function $([\rho_{\mathbb{R}}|^{[0,1]}, \rho_{\mathbb{R}}|^{[-1,1]}, \rho_{\mathbb{R}})$ -computes a real function $g_u^\varphi: [0, 1] \times [-1, 1] \rightarrow \mathbb{R}$ such that*

- (a) $g_u^\varphi(0, y) = g_u^\varphi(1, y) = 0$ for all $y \in [-1, 1]$;
- (b) $|g_u^\varphi(t, y_0) - g_u^\varphi(t, y_1)| \leq 2^{-\lambda(|u|)} |y_0 - y_1|$ for any $t \in [0, 1]$ and $y_0, y_1 \in [-1, 1]$;
- (c) g_u^φ belongs to $\text{dom } \text{LipIVP}$, and $h_u^\varphi := \text{LipIVP}(g_u^\varphi)$ satisfies $h_u^\varphi(1) = 2^{-P(\lambda)(|u|)}$.

Thus G encodes a family of functions g_u^φ , indexed by φ , λ and u , such that $h_u^\varphi := \text{LipIVP}(g_u^\varphi)$ contains the information to answer $L(\varphi)(u)$.

FIGURE 8. Widget for reducing FPSPACE^2 to LipIVP .

Proof of the negative part of Theorem 4.10. Let F be any multifunction in FPSPACE^2 . We need to show that $F \leq_{\text{mF}}^2 \delta_{\square}^{-1} \circ \text{LipIVP} \circ \delta_{\square}$. We may assume that F is a total function and that there is a second-order polynomial Q such that $F(\varphi)(v)$ has length exactly $Q(|\varphi|)(|v|)$ for all $\varphi \in \mathbf{Reg}$ and $v \in \Sigma^*$. There is $L \in \text{PSPACE}^2$ such that $L(\varphi)(v, 0^i)$ equals the i th symbol of $F(\varphi)(v)$ for any $\varphi \in \mathbf{Reg}$, $v \in \Sigma^*$ and $i \in \{0, 1, \dots, Q(|\varphi|)(|v|) - 1\}$.

Apply Lemma 4.12 to this L to obtain the P and G . Let g_u^φ and h_u^φ be as in the Lemma, corresponding to $\lambda(k) = 3k + 2$.

We define s (of Definition 3.5) by describing the real function $g = \delta_{\square}^{-1}(s(\varphi)) \in \text{CL}[[0, 1] \times [-1, 1]]$ for a given φ . It has Lipschitz constant 1. It will be straightforward to check that, by the FP^2 -computability of G , a δ_{\square} -name (and hence a δ_{\square} -name) of g can be FP^2 -computed from φ .

For each binary string v , let $\Lambda_v = 2^{-2|v|-2}$ and

$$(19) \quad c_v = 1 - \frac{1}{2^{|v|}} + \frac{2\bar{v} + 1}{\Lambda_v}, \quad l_v^\mp = c_v \mp \frac{1}{\Lambda_v},$$

where $\bar{v} \in \{0, \dots, 2^{|v|} - 1\}$ means v read as an integer in binary notation. This divides $[0, 1]$ into intervals $[l_v^-, l_v^+]$ indexed by $v \in \{0, 1\}^*$. We further divide the left half $[l_v^-, c_v]$ into $Q(|\varphi|)(|v|) + 1$ subintervals $[l_{v,0}, l_{v,1}]$, $[l_{v,1}, l_{v,2}]$, \dots , $[l_{v,Q(|\varphi|)(|v|)-1}, l_{Q(|\varphi|)(|v|)}]$, $[l_{v,Q(|\varphi|)(|v|)}, c_v]$, where

$$(20) \quad l_{v,i} = c_v - \frac{1}{2^i \Lambda_v}, \quad i = 0, 1, \dots, Q(|\varphi|)(|v|).$$

On each strip $[l_{v,i}, l_{v,i+1}] \times [-1, 1]$, we define g by putting the copies of $g_{(v,0^i)}^\varphi$ as in Figure 8. Precisely, on the strip $[l_{v,i}, l_{v,i+1}] \times \mathbb{R}$, we define g by

$$(21) \quad g\left(l_{v,i} + \frac{t}{2^{i+1}\Lambda_v}, \frac{2m + (-1)^m y}{2^{i+1}\Lambda_v \Gamma_{v,i}}\right) = \frac{g_{(v,0^i)}^\varphi(t, y)}{\Gamma_{v,i}}$$

for each $t \in [0, 1]$ and $m \in \mathbb{N}$, $y \in [-1, 1]$, where $\Gamma_{v,i}$ is an appropriate factor (of vertical shrinkage) that can be written exponentially in second-order polynomial in $|\varphi|$ and $|v|, i$.

On the last strip $[l_{v, Q(|\varphi|)(|v|)}, c_v] \times [-1, 1]$, we define g to be constantly 0. On the right half $[c_v, l_v^+]$, we define g symmetrically: $g(l^+ - t, y) = -g(l^- + t, y)$ for $0 \leq t \leq 1/\Lambda_v$.

Recall that $g_{(v, 0^i)}^\varphi$ is a vector field such that $h_{(v, 0^i)}^\varphi := \text{LipIVP}(g_{(v, 0^i)}^\varphi)$ tells us whether $L(\varphi)(v, i) = 1$ in the way described in item (c) of Lemma 4.12. Thus by arranging the copies of $g_{(v, 0^i)}^\varphi$ in this way, we can see the values $L(\varphi)(\langle v, 0 \rangle), \dots, L(\varphi)(\langle v, Q(|\varphi|)(|v|) - 1 \rangle)$ by looking at $h(c_v)$. The reducing functions r and t (of Definition 3.5) do this lookup. That is, $t(\varphi)(v)$ is the encoding of the rational number c_v with appropriate precision, and $r(\varphi)$ is the function that, given the encoding of (an approximation of) $h(c_v)$, extracts the values $L(\varphi)(v, i)$. \square

In [8], the ineffective version of Lemma 4.12 was used to construct a function that proved the ineffective Theorem 2.9 (Theorem 3.2 of [8]). We needed to use a different construction, because for our Theorem 4.10 (with the reduction \leq_{mF}^2), we needed to get the values $L(\varphi)(\langle v, 0 \rangle), \dots, L(\varphi)(\langle v, Q(|\varphi|)(|v|) - 1 \rangle)$ in one query. For Corollary 4.11 (with the reduction \leq_{T}^2), we are allowed to make many queries, so the straightforward effectivization of a slightly simpler construction used for Theorems 3.2 of [8] would have worked (we would not have to stack the copies of $g_{(v, i)}$ vertically).

5. SUMMARY AND FUTURE WORK

- A TTE machine works on $\Sigma^{\mathbb{N}}$, the infinite strings. We replace $\Sigma^{\mathbb{N}}$ with **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 , FPSPACE^2 . With a suitable notion of polynomial-time reductions, we can also define NP^2 - and 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 δ_{\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 FPSPACE^2 -complete. These are effectivized versions of what has 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. One can also look at known computability results and study whether analogous statements hold true for time- or space-bounded classes.

ACKNOWLEDGEMENTS

We thank Kaveh Ghasemloo for comments on the manuscript that helped improve the presentation. The first author is supported by the Nakajima Foundation; both authors are supported by the Natural Sciences and Engineering Research Council of Canada.

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. Grzegorzczuk. 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*, 19(2):305–332, 2010.
- [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*, pages 216–221, 2010.
- [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] B. Lambov. The basic feasible functionals in computable analysis. *J. Complexity*, 22(6):909–917, 2006.
- [17] K. Mehlhorn. Polynomial and abstract subrecursive classes. *J. Comput. Syst. Sci.*, 12(2):147–178, 1976.
- [18] M. Schröder. Spaces allowing type-2 complexity theory revisited. *Math. Log. Q.*, 50(4-5):443–459, 2004.
- [19] 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.
- [20] K. Weihrauch. *Computable Analysis: An Introduction*. Texts in Theoretical Computer Science. Springer, 2000.
- [21] K. Weihrauch. Computational complexity on computable metric spaces. *Math. Log. Q.*, 49(1):3–21, 2003.

DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF TORONTO