A Characterisation of Functions Computable in Polynomial Time and Space over the Reals with Discrete Ordinary Differential Equations

Simulation of Turing Machines with Analytic Discrete ODEs

Manon Blanc
Institut Polytechnique de Paris, Ecole Polytechnique, LIX, Palaiseau, France

Olivier Bournez
Institut Polytechnique de Paris, Ecole Polytechnique, LIX, Palaiseau, France

Abstract

We prove that functions over the reals computable in polynomial time can be characterised using discrete ordinary differential equations (ODE), also known as finite differences. We also provide a characterisation of functions computable in polynomial space over the reals. In particular, this covers space complexity, while existing characterisations were only able to cover time complexity, and were restricted to functions over the integers, and we prove that no artificial sign or test function is needed even for time complexity. At a technical level, this is obtained by proving that Turing machines can be simulated with analytic discrete ordinary differential equations. We believe this result opens the way to many applications, as it opens the possibility of programming with ODEs, with an underlying well-understood time and space complexity.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Computability; Theory of computation → Complexity classes; Mathematics of computing → Ordinary differential equations

Keywords and phrases Discrete ordinary differential equations, Finite Differences, Implicit complexity, Recursion scheme, Ordinary differential equations, Models of computation, Analog Computations

Digital Object Identifier 10.4230/LIPIcs.MFCS.2023.21

1 Introduction

Recursion schemes constitute a major approach to classical computability theory and, to some extent, to complexity theory. The foundational characterisation of **FPTIME**, based on bounded primitive recursion on notations, due to Cobham [10] gave birth to the field of *implicit complexity* at the interplay of logic and theory of programming. Alternative characterizations, based on safe recursion [1] or on ramification ([16, 15]) or for other classes [17] followed: see [8, 9] for monographs.

Initially motivated to help understanding how analogue models of computations compare to classical digital ones, in an orthogonal way, various computability and complexity classes have been recently characterised using Ordinary Differential Equations (ODE). An unexpected side effect of these proofs is the possibility of programming with classical ODEs, over the continuum. It recently led to solving various open problems. This includes the proof of the existence of a universal ODE [6], the proof of the Turing-completeness of chemical reactions [11], or hardness of problems related to dynamical systems [12].

Discrete ODEs, that we consider in this article, are an approach in-between born from the attempt of [4, 5] to explain some of the constructions for continuous ODEs in an easier way. The basic principle is, for a function $\mathbf{f}(x)$, to consider its discrete derivative defined as $\Delta\mathbf{f}(x) = \mathbf{f}(x+1) - \mathbf{f}(x)$ (also denoted $\mathbf{f}'(x)$ in what follows to help analogy with classical continuous counterparts). A consequence of this attempt is the characterisation obtained in [4, 5]. They provided a characterisation of **FPTIME** for functions over the integers that does not require the specification of an explicit bound in the recursion, in contrast to Cobham's work [10], nor the assignment of a specific role or type to variables, in contrast to safe recursion or ramification [1, 14]. Instead, they only assume involved ODEs to be linear, a very classical natural concept for differential equations.

▶ Remark 1. Unfortunately, even if it was the original motivation, both approaches for characterising complexity classes for continuous and discrete ODEs are currently not directly connected. A key difference is that there is no simple expression (no analogue of the Leibniz rule) for the derivative of the composition of functions in the discrete settings. The Leibniz rule is a very basic tool for establishing results over the continuum, using various stability properties, but that cannot be used easily over discrete settings.

In the context of algebraic classes of functions, the following notation is classical: call *operation* an operator that takes finitely many functions and returns some new function defined from them. Then $[f_1, f_2, \dots, f_k; op_1, op_2, \dots, op_\ell]$ denotes the smallest set of functions containing $f_1, f_2, \dots, f_k$ that is closed under the operations op_1, op_2, ... op_ℓ. Call *discrete function* a function of type $f : S_1 \times \cdots \times S_d \to S'_1 \times \dots S'_{d'}$, where each S_i, S'_i is either $\mathbb{N}$ or $\mathbb{Z}$. Write **FPTIME** for the class of functions computable in polynomial time, and **FPSPACE** for the class of functions computable in polynomial space.

▶ Remark 2. The literature considers two possible definitions for **FPSPACE**, according to whether functions with non-polynomial size values are allowed or not. In our case, we should add "whose outputs remain of polynomial size", to resolve the ambiguity[1].

A main result of [4, 5] is the following ($\mathbb{LDL}$ stands for linear derivation on length):

▶ **Theorem 3** ([4]). *For functions over the reals, we have* $\mathbb{LDL} = \mathbf{FPTIME}$ *where* $\mathbb{LDL} = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \times, sg(x)\ ; composition, linear\ length\ ODE]$.

In particular, writing as usual B^A for functions from A to B, we deduce:

▶ **Corollary 4** (Functions over the integers). $\mathbb{LDL} \cap \mathbb{N}^{\mathbb{N}} = \mathbf{FPTIME} \cap \mathbb{N}^{\mathbb{N}}$.

That is to say, $\mathbb{LDL}$ (and hence **FPTIME** for functions over the integers) is the smallest class of functions that contains the constant functions $\mathbf{0}$ and $\mathbf{1}$, the projections π_i^k of the i^e coordinate of a vector of size k, the length function $\ell(x)$, mapping an integer to the length of its binary representation, the addition $x+y$, the subtraction $x-y$, the multiplication $x \times y$, the **sign** function $\mathrm{sg}(x)$ and that is closed under composition (when defined) and linear length-ODE scheme: the linear length-ODE scheme, formally given by Definition 15, corresponds to defining a function from a linear ODEs with respect to derivation along the length of the argument, so of the form $\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \ell} = \mathbf{A}[\mathbf{f}(x,\mathbf{y}), x, \mathbf{y}] \cdot \mathbf{f}(x,\mathbf{y}) + \mathbf{B}[\mathbf{f}(x,\mathbf{y}), x, \mathbf{y}]$.

Here, we use the notation $\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \ell}$ which corresponds to the derivation of $\mathbf{f}$ along the length function: given some function $\mathcal{L} : \mathbb{N}^{p+1} \to \mathbb{Z}$ and in particular for the case where $\mathcal{L}(x, \mathbf{y}) = \ell(x)$,

$$\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \mathcal{L}} = \frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \mathcal{L}(x,\mathbf{y})} = \mathbf{h}(\mathbf{f}(x,\mathbf{y}), x, \mathbf{y}) \tag{1}$$

is a formal synonym for $\mathbf{f}(x+1,\mathbf{y}) = \mathbf{f}(x,\mathbf{y}) + (\mathcal{L}(x+1,\mathbf{y}) - \mathcal{L}(x,\mathbf{y})) \cdot \mathbf{h}(\mathbf{f}(x,\mathbf{y}), x, \mathbf{y})$.

[1] Otherwise, the class is not closed by composition: this may be considered as a basic requirement when talking about the complexity of functions. The issue is about the usage of not counting the output as part of the total space used. In this model, given f computable in polynomial space, and g in logarithmic space, $f \circ g$ (and $g \circ f$) is computable in polynomial space. But this is not true, if we assume only f and g to be computable in polynomial space, since the first might give an output of exponential size.

▶ Remark 5. This concept introduced in [4, 5], is motivated by the fact that the latter expression is similar to the classical formula for continuous ODEs:

$$\frac{\delta f(x,\mathbf{y})}{\delta x} = \frac{\delta \mathcal{L}(x,\mathbf{y})}{\delta x} \cdot \frac{\delta f(x,\mathbf{y})}{\delta \mathcal{L}(x,\mathbf{y})},$$

and hence is similar to a change of variable. Consequently, a linear length-ODE is basically a linear ODE over a variable t once the change of variable $t = \ell(x)$ is done.

However, in the context of (classical) ODEs, considering functions over the reals is more natural than only functions over the integers. Call *real function* a function $f : S_1 \times \cdots \times S_d \to S'_1 \times \ldots S'_{d'}$, where each S_i, S'_i is either $\mathbb{R}$, $\mathbb{N}$ or $\mathbb{Z}$. A natural question about the characterisation of **FPTIME** for real functions arises, and not only discrete functions: we consider here computability over the reals in its most classical approach, namely computable analysis [19].

As a first step, the class $\mathbb{LDL}^\bullet = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \times, \overline{\mathrm{cond}}(x), \frac{x}{2}\ ; \textit{composition}, \textit{linear length ODE}]$ has been considered in [3, 2] where some characterisation of **PTIME** was obtained, but only for functions from the integers to the reals (i.e. sequences) while it would be more natural to characterise functions from the reals to the reals. More importantly, this was obtained by assuming that some **non-analytic exact function** is among the basic available functions to simulate a Turing machine: $\overline{\mathrm{cond}}$ valuing 1 for $x > \frac{3}{4}$ and 0 for $x < \frac{1}{4}$.

We prove first this is not needed, and mainly, we extend all previous results to real functions, furthermore covering not only time complexity but also space complexity. Consider

$$\mathbb{LDL}^\circ = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \tanh, \frac{x}{2}, \frac{x}{3}; \textit{composition}, \textit{linear length ODE}],$$

where $\ell : \mathbb{N} \to \mathbb{N}$ is the length function, mapping some integer to the length of its binary representation, $\frac{x}{2} : \mathbb{R} \to \mathbb{R}$ is the function dividing by 2 (similarly for $\frac{x}{3}$) and all other basic functions defined exactly as for $\mathbb{LDL}$, but considered here as functions from the reals to reals.

▶ Remark 6. This class is $\mathbb{LDL}$ but without the $\mathrm{sg}(x)$ function, nor the multiplication function, or $\mathbb{LDL}^\bullet$ but without the $\overline{\mathrm{cond}}$ function, nor the multiplication. This is done by adding the analytic tanh functions as a substitute (and adding $x/3$).

▶ Remark 7. We can consider $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{R}$ but as functions may have different types of outputs, the composition is an issue. We consider, as in [3, 2], that composition may not be defined in some cases: it is a partial operator. For example, given $f : \mathbb{N} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$, the composition of g and f is defined as expected, but f cannot be composed with a function such as $h : \mathbb{N} \to \mathbb{N}$.

First, we improve Theorem 3 by stating **FPTIME** over the integers can be characterised algebraically using linear length ODEs and only analytic functions (i.e. no need for sign function). Since $\mathbb{LDL}^\circ$ is about functions over the reals, and Theorem 3 is about functions over the integers, we need a way to compare these classes. Given a function $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^{d'}$ sending every integer $\mathbf{n} \in \mathbb{N}^d$ to the vicinity of some integer of $\mathbb{N}^d$, say at distance less than 1/4, we write $\mathrm{DP}(f)$ for its discrete part: this is the function from $\mathbb{N}^d \to \mathbb{N}^{d'}$ mapping $\mathbf{n} \in \mathbb{N}^d$ to the integer rounding of $\mathbf{f}(n)$. Given a class $\mathcal{C}$ of such functions, we write $\mathrm{DP}(\mathcal{C})$ for the class of the discrete parts of the functions of $\mathcal{C}$.

▶ **Theorem 8.** $\mathrm{DP}(\mathbb{LDL}^\circ) = \mathbf{FPTIME} \cap \mathbb{N}^{\mathbb{N}}$.

Write $\overline{\mathbb{LDL}^\circ}$ for the class obtained by adding some effective limit operation similar to the one considered in [3] to get $\overline{\mathbb{LDL}^\bullet}$. We get a characterization of functions over the reals (and not only sequences as in [3]) computable in polynomial time.

▶ **Theorem 9** (Generic functions over the reals). $\overline{\mathbb{LDL}^\circ} \cap \mathbb{R}^{\mathbb{R}} = \mathbf{FPTIME} \cap \mathbb{R}^{\mathbb{R}}$
More generally: $\overline{\mathbb{LDL}^\circ} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}} = \mathbf{FPTIME} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^d}$.

We also prove that, by adding a robust linear ODE scheme (Definition 18), we get a class $\mathbb{RLD}^\circ$ (this stands for robust linear derivation) with the similar statements but for **FPSPACE**.

▶ **Theorem 10.** $\mathrm{DP}(\mathbb{RLD}^\circ) = \mathbf{FPSPACE} \cap \mathbb{N}^{\mathbb{N}}$.

▶ **Theorem 11** (Generic functions over the reals). $\overline{\mathbb{RLD}^\circ} \cap \mathbb{R}^{\mathbb{R}} = \mathbf{FPSPACE} \cap \mathbb{R}^{\mathbb{R}}$
More generally: $\overline{\mathbb{RLD}^\circ} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}} = \mathbf{FPSPACE} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^d}$.

As far as we know, this is the first time a characterisation of **FPSPACE** with discrete ODEs is provided. If we forget the context of discrete ODEs, **FPSPACE** has been characterised in [18] but using a bounded recursion scheme, i.e. requiring some explicit bound in the spirit of Cobham's statement [10]. We avoid this issue by considering numerically stable schemes, which are very natural in the context of ODEs.

At a technical level, all our results are obtained by proving Turing machines can be simulated with analytic discrete ODEs in a suitable manner. We believe our constructions could be applied to many other situations, where programming with ODEs is needed.

In Section 2, we recall some basic statements about the theory of discrete ODEs. In Section 3, we establish some properties about particular functions required for our proofs. In Section 4 we prove our main technical result: Turing machines can be simulated using functions from $\mathbb{LDL}^\circ$. Section 5 is about converting integers and reals (dyadic) to words of a specific form. Section 6 is about applications of our toolbox. We prove in particular all above theorems.

2 Some concepts related to discrete ODEs

In this section, we recall some concepts and definitions from discrete ODEs, either well-known or established in [4, 5, 3]. We consider here that $\mathfrak{tanh}$ is tanh, the hyperbolic tangent. The papers [4, 5] use similar definitions with the sign function sg and [3] with the piecewise affine function $\overline{\text{cond}}$, that values 1 for $x > \frac{3}{4}$ and 0 for $x < \frac{1}{4}$, instead of tanh.

▶ **Definition 12** ([3]). *A* $\mathfrak{tanh}$*-polynomial expression* $P(x_1, ..., x_h)$ *is an expression built-on* $+, -, \times$ *(often denoted* $\cdot$*) and* $\mathfrak{tanh}$ *functions over a set of variables* $V = \{x_1, ..., x_h\}$ *and integer constants.*

We need to measure the degree, similarly to the classical notion of degree in polynomial expression, but considering all subterms that are within the scope of a $\mathfrak{tanh}$ function contributes to 0 to the degree.

▶ **Definition 13** ([3]). *The degree* $\deg(x, P)$ *of a term* P *in* $x \in V$ *is defined inductively as follows:* $\deg(x, x) = 1$ *and for* $x' \in V \cup \mathbb{Z}$ *such that* $x' \neq x$, $\deg(x, x') = 0$; $\deg(x, P + Q) = \max\{\deg(x, P), \deg(x, Q)\}$; $\deg(x, P \times Q) = \deg(x, P) + \deg(x, Q)$; $\deg(x, \mathfrak{tanh}(P)) = 0$. *A* $\mathfrak{tanh}$*-polynomial expression* P *is essentially constant in* x *if* $\deg(x, P) = 0$.

A vectorial function (resp. a matrix or a vector) is said to be a $\mathfrak{tanh}$-polynomial expression if all its coordinates (resp. coefficients) are, and *essentially constant* if all its coefficients are.

▶ **Definition 14** ([4, 5, 3]). *A* $\mathfrak{tanh}$*-polynomial expression* $\mathbf{g}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})$ *is essentially linear in* $\mathbf{f}(x, \mathbf{y})$ *if it is of the form:* $\mathbf{A}[\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}] \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}[\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}]$ *where* $\mathbf{A}$ *and* $\mathbf{B}$ *are* $\mathfrak{tanh}$*-polynomial expressions essentially constant in* $\mathbf{f}(x, \mathbf{y})$.

For example, the expression $P(x,y,z) = x \cdot \mathfrak{tanh}\,(x^2 - z) \cdot y + y^3$ is essentially linear in x, essentially constant in z and not linear in y. The expression: $z + (1 - \mathfrak{tanh}(x)) \cdot (1 - \mathfrak{tanh}(-x)) \cdot (y - z)$ is essentially constant in x and linear in y and z.

▶ **Definition 15** (Linear length ODE [4, 5]). *A function $\mathbf{f}$ is linear length-ODE definable from $\mathbf{u}$ essentially linear in $\mathbf{f}(x,\mathbf{y})$, $\mathbf{g}$ and $\mathbf{h}$, if it corresponds to the solution of*

$$f(0,\mathbf{y}) = \mathbf{g}(\mathbf{y}) \quad \textit{and} \quad \frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \ell} = \mathbf{u}(\mathbf{f}(x,\mathbf{y}), \mathbf{h}(x,\mathbf{y}), x, \mathbf{y}). \tag{2}$$

A fundamental fact is that the derivation with respect to length provides a way to do some change of variables:

▶ **Lemma 16** ([4, 5]). *Assume that (2) holds. Then $\mathbf{f}(x,\mathbf{y})$ is given by $\mathbf{f}(x,\mathbf{y}) = \mathbf{F}(\ell(x),\mathbf{y})$ where $\mathbf{F}$ is the solution of the initial value problem*

$$\mathbf{F}(1,\mathbf{y}) = \mathbf{g}(\mathbf{y}), \quad \textit{and} \quad \frac{\partial \mathbf{F}(t,\mathbf{y})}{\partial t} = \mathbf{u}(\mathbf{F}(t,\mathbf{y}), \mathbf{h}(2^t-1,\mathbf{y}), 2^t-1, \mathbf{y}). \tag{3}$$

This means $\mathbf{f}(x,\mathbf{y})$ depends only on the length of its first argument: $\mathbf{f}(x,\mathbf{y}) = \mathbf{f}(2^{\ell(x)},\mathbf{y})$. Then (3) can be seen as defining a function (with this latter property) by a recurrence of type

$$\mathbf{f}(2^0,\mathbf{y}) = \mathbf{g}(\mathbf{y}), \quad \textit{and} \quad \mathbf{f}(2^{t+1},\mathbf{y}) = \overline{\mathbf{u}}(\mathbf{f}(2^t,\mathbf{y}), \mathbf{h}(2^t-1,\mathbf{y}), 2^t, \mathbf{y}). \tag{4}$$

for some $\overline{\mathbf{u}}$ is *essentially linear* in $\mathbf{f}(2^t,\mathbf{y})$. As recurrence (3) is basically equivalent to (2):

▶ **Corollary 17** (Linear length ODE presented with powers of 2). *A function $\mathbf{f}$ is linear $\mathcal{L}$-ODE definable iff the value of $\mathbf{f}(x,\mathbf{y})$ depends only on the length of its first argument and satisfies (4), for some $\mathbf{g}$ and $\mathbf{h}$, and $\overline{\mathbf{u}}$, essentially linear in $\mathbf{f}(2^t,\mathbf{y})$.*

We assume it is easier for our reader to deal with recurrences of the form (4) than with ODEs of the form (2). Consequently, this is how we will describe many functions from now on, starting with some basic functions, authorising compositions, and the above schemes. As an example, $n \mapsto 2^n$ can easily be defined that way (by $2^0 = 1$, and $2^{n+1} = 2 \cdot 2^n = 2^n + 2^n$) and we can produce $n \mapsto 2^{p(n)}$ for any polynomial p. For example, $(n_1,\dots,n_k) \to 2^{n_1 n_2 \cdots n_k}$ can be obtained, using k such schemes in turn, providing the case of the polynomial $p(n) = n^k$.

When talking about space complexity, we will also consider the case where the ODE is not derivated with respect to length but with classical derivation. For functions over the reals an important issue is numerical stability.

▶ **Definition 18** (Robust linear ODE [4, 5]). *A bounded function $\mathbf{f}$ is robustly linear ODE definable from $\mathbf{u}$ essentially linear in $\mathbf{f}(x,\mathbf{y})$, $\mathbf{g}$ and $\mathbf{h}$ if:*

1. *it corresponds to the solution of*

$$\mathbf{f}(0,\mathbf{y}) = \mathbf{g}(\mathbf{y}) \quad \textit{and} \quad \frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial x} = \mathbf{u}(\mathbf{f}(x,\mathbf{y}), \mathbf{h}(x,\mathbf{y}), x, \mathbf{y}), \tag{5}$$

2. *where the schema (5) is polynomially numerically stable.*

Here, writing $a =_n b$ for $\|a - b\| \le 2^{-n}$ for conciseness, **2.** means formally there exists some polynomial p such that, for all integer n, writing $\epsilon(n) = p(n + \ell(\mathbf{y}))$, if you consider any solution of $\tilde{\mathbf{y}} =_{\epsilon(n)} \mathbf{y}$ *and* $\tilde{\mathbf{h}}(x,\tilde{\mathbf{y}}) =_{\epsilon(n)} \mathbf{h}(x,\tilde{\mathbf{y}})$, and $\tilde{\mathbf{f}}(0,\tilde{\mathbf{y}}) =_{\epsilon(n)} \mathbf{g}(\mathbf{y})$ and $\frac{\partial \tilde{\mathbf{f}}(x,\tilde{\mathbf{y}})}{\partial x} =_{\epsilon(n)} \mathbf{u}(\tilde{\mathbf{f}}(x,\tilde{\mathbf{y}}), \tilde{\mathbf{h}}(x,\tilde{\mathbf{y}}), x, \tilde{\mathbf{y}})$ then $\tilde{\mathbf{f}}(x,\tilde{\mathbf{y}}) =_{\epsilon(n)} \mathbf{f}(x,\mathbf{y})$.

3 Some results about various functions

A key part of our proofs is the construction of very specific functions in $\mathbb{LDL}^{\circ}$: we write $\{x\}$ for the fractional part of the real x, i.e. $\{x\} = x - \lfloor x \rfloor$. We provide some graphical representations of some of them to show that these functions are sometimes highly non-trivial (see for e.g. Figures 3 or 6).

A first observation is that we can uniformly approximate the $\mathrm{ReLU}(x) = \max(0, x)$ function using a essentially constant function:

▶ **Lemma 19.** *We denote by $Y(x, 2^{m+2})$ the function $Y(x, 2^{m+2}) = \frac{1+\tanh(2^{m+2}x)}{2}$ (illustrated by Figure 1). For all integer m, for all $x \in \mathbb{R}$, $|\mathrm{ReLU}(x) - xY(x, 2^{m+2})| \leq 2^{-m}$.*

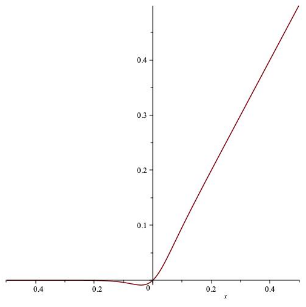


Figure 1 Graphical representation of $xY(x, 2^{2+2})$ obtained with maple.

We deduce we can uniformly approximate the continuous sigmoid functions (when $1/(b-a)$ is in $\mathbb{LDL}^{\circ}$) defined as: $\mathfrak{s}(a, b, x) = 0$ whenever $w \leq a$, $\frac{x-a}{b-a}$ whenever $a \leq x \leq b$, and 1 whenever $b \leq x$.

▶ **Lemma 20** (Uniform approximation of any piecewise continuous sigmoid)**.** *Assume $\frac{1}{b-a}$ is in $\mathbb{LDL}^{\circ}$. Then there is some function $\mathcal{C}\text{-}\mathfrak{s}(m, a, b, x) \in \mathbb{LDL}^{\circ}$ (illustrated by Figure 2) such that for all integer m, $|\mathcal{C}\text{-}\mathfrak{s}(m, a, b, x) - \mathfrak{s}(a, b, x)| \leq 2^{-m}$.*

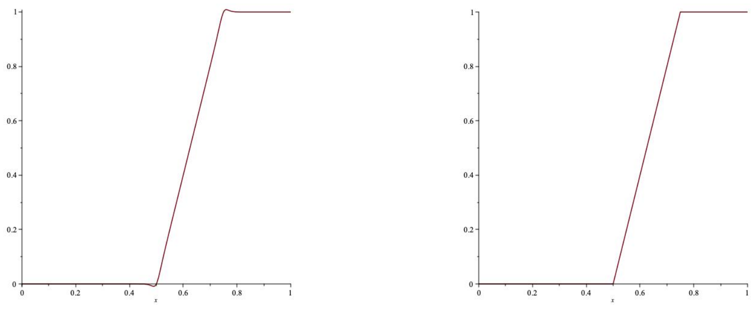
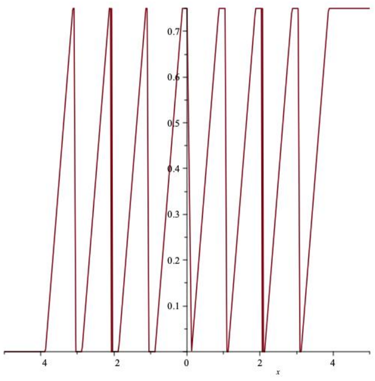


Figure 2 Graphical representation of $\mathcal{C}\text{-}\mathfrak{s}(2, \frac{1}{2}, \frac{3}{4}, x)$ and $\mathcal{C}\text{-}\mathfrak{s}(2^5, \frac{1}{2}, \frac{3}{4}, x)$ obtained with maple.

Proof. We can write $\mathfrak{s}(a, b, x) = \frac{\mathrm{ReLU}(x-a)-\mathrm{ReLU}(x-b)}{b-a}$. Thus, $|\mathcal{C}\text{-}\mathfrak{s}(m + 1 + c, a, b, x) - \mathfrak{s}(a, b, x)| \leq \frac{2.2^{-m-1-c}}{b-a}$, using the triangle inequality. Take c such that $\frac{1}{b-a} \leq 2^c$. ◀

The existence of the following function will play an important role to obtain the various functions of the next corollary.

▶ **Theorem 21.** *There exists some function $\xi : \mathbb{N}^2 \to \mathbb{R}$ in $\mathbb{LDL}^\circ$ (illustrated in Figure 3) such that for all $n, m \in \mathbb{N}$ and $x \in [-2^n, 2^n]$, whenever $x \in [\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}]$, $\left|\xi(2^m, 2^n, x) - \{x - \frac{1}{8}\}\right| \le 2^{-m}$.*



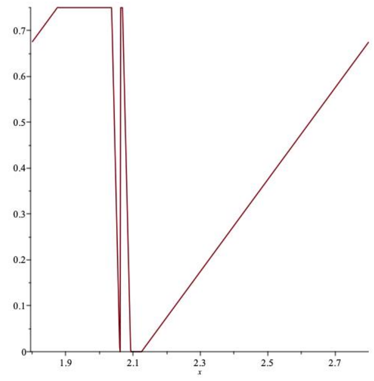
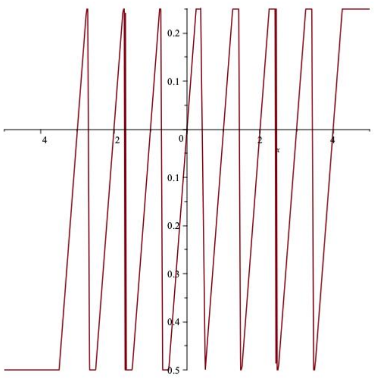


Figure 3 Graphical representations of $\xi(2, 4, x)$ obtained with maple: some details on the right.

The main idea of the proof is, by parity, to reduce the problem to construct an auxiliary function ξ' that works for $x \ge 0$, writing $\xi(2^m, N, x) = \xi'(2^{m+2}, N, x) - \xi'(2^{m+2}, N, -x) + \frac{3}{4} - \frac{3}{4}\mathcal{C}\text{-}\mathfrak{s}(2^{m+2}, 0, \frac{1}{8}, x)$, and then proving that ξ' is definable in $\mathbb{LDL}^\circ$, using an adhoc recursive (in n) definition of it.

▶ **Corollary 22** (A bestiary of functions)**.** *There exist*

1. *$\xi_1, \xi_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto \mathbb{R} \ \in \mathbb{LDL}^\circ$ such that, for all $n, m \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever $x \in [\lfloor x \rfloor - \frac{1}{2}, \lfloor x \rfloor + \frac{1}{4}]$, $|\xi_1(2^m, 2^n, x) - \{x\}| \le 2^{-m}$, and whenever $x \in [\lfloor x \rfloor, \lfloor x \rfloor + \frac{3}{4}]$, $|\xi_2(2^m, 2^n, x) - \{x\}| \le 2^{-m}$ (see Figure 4).*
2. *$\sigma_1, \sigma_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto \mathbb{R} \ \in \mathbb{LDL}^\circ$ such that, for all $n, m \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever $x \in [\lfloor x \rfloor - \frac{1}{2}, \lfloor x \rfloor + \frac{1}{4}]$, $\sigma_1(2^m, 2^n, x) - \lfloor x \rfloor \le 2^{-m}$, and whenever $x \in [\lfloor x \rfloor, \lfloor x \rfloor + \frac{3}{4}]$, $|\sigma_2(2^m, 2^n, x) - \lfloor x \rfloor| \le 2^{-m}$ (see Figure 5).*
3. *$\lambda : \mathbb{N}^2 \times \mathbb{R} \mapsto [0, 1] \in \mathbb{LDL}^\circ$ such that for all $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever $x \in [\lfloor x \rfloor + \frac{1}{4}, \lfloor x \rfloor + \frac{1}{2}]$, $|\lambda(2^m, 2^n, x) - 0| \ \le 2^{-m}$, and whenever $x \in [\lfloor x \rfloor + \frac{3}{4}, \lfloor x \rfloor + 1]$, $|\lambda(2^m, 2^n, x) - 1| \le 2^{-m}$.*
4. *$\mathrm{mod}_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto [0, 1] \in \mathbb{LDL}^\circ$ such that for all $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever $x \in [\lfloor x \rfloor - \frac{1}{4}, \lfloor x \rfloor + \frac{1}{4}]$, $|\,\mathrm{mod}_2(2^m, 2^n, x)\text{-}\lfloor x \rfloor \bmod 2| \le 2^{-m}$.*



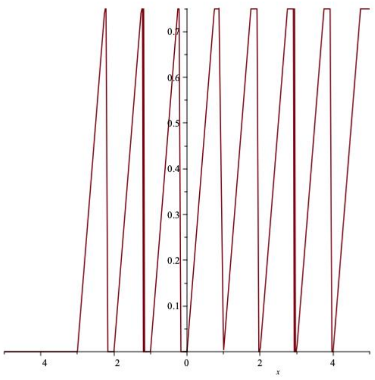


Figure 4 Graphical representation of $\xi_1(2, 4, x)$ and $\xi_2(2, 4, x)$ obtained with maple.

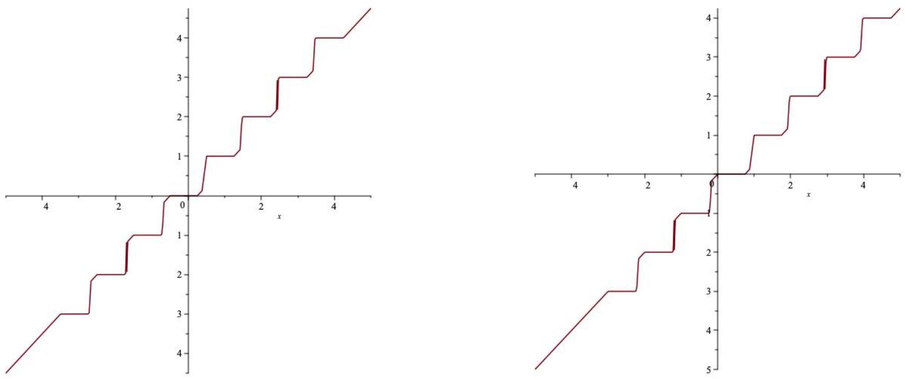


Figure 5 Graphical representation of $\sigma_1(2,4,x)$ and $\sigma_2(2,4,x)$ obtained with maple.

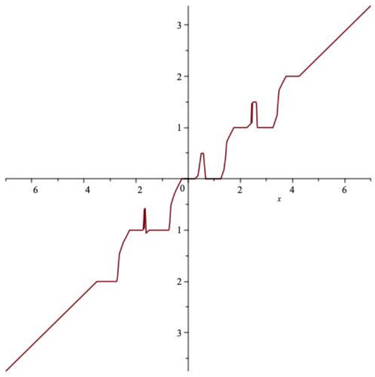


Figure 6 Graphical representation of $\div_2(2,4,x)$ obtained with maple.

5. $\div_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto [0,1] \in \mathbb{LDL}^\circ$ *such that for all* $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n+1, 2^n]$, *whenever* $x \in [\lfloor x \rfloor - \frac{1}{4}, \lfloor x \rfloor + \frac{1}{4}]$, $| \div_2 (2^m, 2^n, x) - \lfloor x \rfloor // 2| \leq 2^{-m}$, *with* $//$ *the integer division (see Figure 6).*

Proof. Take $\xi_1(M,N,x) = \xi(M,N,x-\frac{3}{8}) - \frac{1}{2}$, $\xi_2(M,N,x) = \xi(N,x-\frac{7}{8})$, $\sigma_i(M,N,x) = x - \xi_i(M,N,x)$, $\lambda(M,N,x) = \mathcal{C}\text{-}\mathfrak{s}(2M, 1/4, 1/2, \xi(2M,N,x-9/8))$, $\text{mod}_2(M,N,x) = 1 - \lambda(M,N/2,\frac{1}{2}x+\frac{7}{8})$, $\div 2(M,N,x) = \frac{1}{2}(\sigma_1(M,N,x) - \text{mod}_2(M,N,x))$. ◀

Observing that for $\overline{\text{if}}(d,l) = 4\,\mathfrak{s}(1,2,1/2+d+l/4) - 2$, for $l \in [0,1]$, we have $\overline{\text{if}}(0,l) = 0$, and $\overline{\text{if}}(1,l) = l$, and using Lemma 20 on this sigmoid, we get:

▶ **Lemma 23.** *There exists* $\mathcal{C}\text{-if} \in \mathbb{LDL}^\circ$ *such that,* $l \in [0,1]$, *if we take* $|d' - 0| \leq 1/4$, *then* $|\mathcal{C}\text{-if}(d',l) - 0| \leq 2^{-m}$, *and if we take* $|d'-1| \leq 1/4$, *then* $|\mathcal{C}\text{-if}(d',l) - l| \leq 2^{-m}$.

▶ **Lemma 24.** *Let* $\alpha_1, \alpha_2, \dots, \alpha_n$ *be some integers, and* $V_1, V_2, \dots, V_n$ *some constants. We write* $\text{send}(\alpha_i \mapsto V_i)_{i \in \{1,\dots,n\}}$ *for the function that maps any* $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ *to* V_i, *for all* $i \in \{1, \dots, n\}$.

There is some function in $\mathbb{LDL}^\circ$, *that we write* $\mathcal{C}\text{-send}(2^m, \alpha_i \mapsto V_i)_{i \in \{1,\dots,n\}}$, *that maps any* $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ *to a real at distance at most* 2^{-m} *of* V_i, *for all* $i \in \{1, \dots, n\}$.

▶ **Lemma 25.** *Let* N *be some integer. Let* $\alpha_1, \alpha_2, \dots, \alpha_n$ *be some integers, and* $V_{i,j}$ *for* $1 \leq i \leq n$ *some constants, with* $0 \leq j < N$. *We write* $\text{send}((\alpha_i, j) \mapsto V_{i,j})_{i \in \{1,\dots,n\}, j \in \{0,\dots,N-1\}}$ *for the function that maps any* $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ *and* $y \in [j-1/4, j+1/4]$ *to* $V_{i,j}$, *for all* $i \in \{1, \dots, n\}$, $j \in \{0, \dots, N-1\}$.

There is some function in $\mathbb{LDL}^\circ$, *that we write* $\mathcal{C}$-send$(2^m, (\alpha_i, j) \mapsto V_{i,j})_{i\in\{1,\dots,n\}, j\in\{0,\dots,N-1\}}$, *that maps any* $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ *and* $y \in [j - 1/4, j + 1/4]$ *to a real at distance at most* 2^{-m} *of* $V_{i,j}$*, for all* $i \in \{1, \dots, n\}$, $j \in \{0, \dots, N-1\}$.

4 Simulating Turing machines with functions of $\mathbb{LDL}^\circ$

This section is devoted to the simulation of a Turing machine using some analytic functions, and in particular functions from $\mathbb{LDL}^\circ$. We use some ideas from [3] but with several improvements, as we need to deal with errors and avoid multiplications.

Consider without loss of generality some Turing machine $M = (Q, \{0, 1, 3\}, q_{init}, \delta, F)$ using the symbols $0, 1, 3$, where $B = 0$ is the blank symbol.

▶ Remark 26. The reason of the choice of symbols 1 and 3 will be made clear later.

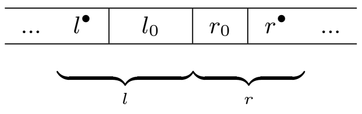
We assume $Q = \{0, 1, \dots, |Q|-1\}$. Let $\dots l_{-k} l_{-k+1} \dots l_{-1} l_0 r_0 r_1 \dots r_n \dots$. denote the content of the tape of the Turing machine M. In this representation, the head is in front of symbol r_0, and $l_i, r_i \in \{0, 1, 3\}$ for all i. Such a configuration C can be denoted by $C = (q, l, r)$, where $l, r \in \Sigma^\omega$ are words over alphabet $\Sigma = \{0, 1, 3\}$ and $q \in Q$ denotes the internal state of M. Write: $\gamma_{word} : \Sigma^\omega \to \mathbb{R}$ for the function that maps a word $w = w_0 w_1 w_2 \dots$ to the dyadic $\gamma_{word}(w) = \sum_{n\geq 0} w_n 4^{-(n+1)}$.

The idea is that such a configuration C can also be encoded by some element $\overline{C} = (q, \overline{l}, \overline{r}) \in \mathbb{N} \times \mathbb{R}^2$, by considering $\overline{r} = \gamma_{word}(r)$ and $\overline{l} = \gamma_{word}(l)$. In other words, we encode the configuration of a bi-infinite tape Turing machine M by real numbers using their radix 4 encoding, but using only digits 1,3. Notice that this lives in $Q \times [0, 1]^2$. Denoting the image of $\gamma_{word} : \Sigma^\omega \to \mathbb{R}$ by $\mathcal{I}$, this even lives in $Q \times \mathcal{I}^2$.

▶ Remark 27. Notice that $\mathcal{I}$ is a Cantor-like set: it corresponds to the rational numbers that can be written using only 1 and 3 in base 4. We write $\mathcal{I}_S$ for those with at most S digits after the point (i.e. of the form $n/4^S$ for some integer n).

▶ **Lemma 28.** *We can construct some function* $\overline{Next}$ *in* $\mathbb{LDL}^\circ$ *that simulates one step of* M*: given a configuration* C*, writing* C' *for the next configuration, we have for all integer* m*,* $\|\overline{Next}(2^m, \overline{C}) - \overline{C}'\| \leq 2^{-m}$.

Proof. We can write $l = l_0 l^\bullet$ and $r = r_0 r^\bullet$, where l_0 and r_0 are the first letters of l and r, and $l^\bullet$ and $r^\bullet$ corresponding to the (possibly infinite) word $l_{-1} l_{-2} \dots$ and $r_1 r_2 \dots$ respectively.



The function $Next$ is of the form $Next(q, l, r) = Next(q, l^\bullet l_0, r_0 r^\bullet) = (q', l', r')$ defined as a definition by case of type:

$$(q', l', r') = \begin{cases} (q', l^\bullet l_0 x, r^\bullet) & \text{whenever } \delta(q, r_0) = (q', x, \to) \\ (q', l^\bullet, l_0 x r^\bullet) & \text{whenever } \delta(q, r_0) = (q', x, \leftarrow) \end{cases}$$

This can be rewritten as a first candidate for the function $\overline{Next}$. Consider the similar function working over the representation of the configurations as reals, considering $r_0 = \lfloor 4\overline{r} \rfloor$

$$\begin{aligned} \overline{Next}(q, \overline{l}, \overline{r}) &= \overline{Next}(q, \overline{l^\bullet l_0}, \overline{r_0 r^\bullet}) = (q', \overline{l'}, \overline{r'}) \\ &= \begin{cases} (q', \overline{l^\bullet l_0 x}, \overline{r^\bullet}) & \text{whenever } \delta(q, r_0) = (q', x, \to) \\ (q', \overline{l^\bullet}, \overline{l_0 x r^\bullet}) & \text{whenever } \delta(q, r_0) = (q', x, \leftarrow) \end{cases} \end{aligned}$$

- in the first case "$\rightarrow$" : $\overline{l'} = 4^{-1}\overline{l} + 4^{-1}x$ and $\overline{r'} = \overline{r^\bullet} = \{4\overline{r}\}$
- in the second case "$\leftarrow$" : $\overline{l'} = \overline{l^\bullet} = \{4\overline{l}\}$ and $\overline{r'} = 4^{-2}\{4\overline{r}\} + 4^{-2}x + \lfloor 4\overline{l}\rfloor/4$ (6)

We introduce the following functions: $\rightarrow: Q \times \{0,1,3\} \mapsto \{0,1\}$ and $\leftarrow: Q \times \{0,1,3\} \mapsto \{0,1\}$ such that $\rightarrow(q,a)$ (respectively: $\leftarrow(q,a)$) is 1 when $\delta(q,a) = (\cdot,\cdot,\rightarrow)$ (resp. $(\cdot,\cdot,\leftarrow)$), i.e. the head moves right (resp. left), and 0 otherwise. We define $nextq^q_a = q'$ if $\delta(q,a) = (q',\cdot,\cdot)$, i.e. values (q',x,m) for some x and $m \in \{\leftarrow,\rightarrow\}$.

We can rewrite $\overline{Next}(q,\overline{l},\overline{r}) = (q',\overline{l}',\overline{r}')$ as $\overline{l}' = \sum_{q,r_0}\left[\rightarrow(q,r_0)\left(\frac{\overline{l}}{4}+\frac{x}{4}\right) + \leftarrow(q,r_0)\{4\overline{l}\}\right]$ and $\overline{r'} = \sum_{q,r_0}\left[\rightarrow(q,r_0)\{4\overline{r}\} + \leftarrow(q,r_0)\left(\frac{\{4r\}}{4^2}+\frac{x}{4^2}+\frac{\lfloor 4\overline{l}\rfloor}{4}\right)\right]$, and, using notation of Lemma 25, $q' = \text{send}((q,r) \mapsto nextq^q_r)_{q\in Q, r\in\{0,1,3\}}(q,\lfloor 4\overline{r}\rfloor)$.

Our problem with such expressions is that they involve some discontinuous functions such as the integer part and the fractional part function, and we would rather have analytic (hence continuous) functions. A key point is that from our trick of using only symbols 1 and 3, we are sure that in an expression like $\lfloor 4\overline{r}\rfloor$, either it values 0 (this is the specific case where there remain only blanks in r), or that $4\overline{r}$ lives in an interval $[1,2]$ or in interval $[3,4]$. That means that we could replace $\lfloor 4\overline{r}\rfloor$ by $\sigma(4\overline{r})$ if we take σ as some continuous function that would be affine and values respectively 0, 1 and 3 on $\{0\}\cup[1,2]\cup[3,4]$ (that is to say matches $\lfloor 4\overline{r}\rfloor$ on this domain). A possible candidate is $\sigma(x) = \mathfrak{s}(1/4,3/4,x) + \mathfrak{s}(9/4,11/4,x)$. Then considering $\xi(x) = x - \sigma(x)$, then $\xi(4\overline{r})$ would be the same as $\{4\overline{r}\}$: that is, considering $r_0 = \sigma(4\overline{r})$, replacing in the above expression every $\{4\cdot\}$ by $\xi(\cdot)$, and every $\lfloor\cdot\rfloor$ by $\sigma(\cdot)$, and get something that would still work the same, but using only continuous functions.

But, we would like to go to some analytic functions and not only continuous functions, and it is well-known that an analytic function that equals some affine function on some interval (e.g. on [1,2]) must be affine, and hence cannot be 3 on $[3,4]$. But the point is that we can try to tolerate errors, and replace $\mathfrak{s}(\cdot,\cdot)$ by $\mathcal{C}\text{-}\mathfrak{s}(2^{m+c},\cdot,\cdot)$ in the expressions above for σ and ξ, taking c such that $(3+1/4^2)3|Q| \le 2^c$. This would just introduce some error at most $(3+1/4^2)3|Q|2^{-c}2^{-m} \le 2^{-m}$.

▶ Remark 29. We could also replace every $\rightarrow(q,r)$ in above expressions for $\overline{l}'$ and $\overline{r}'$ by $\mathcal{C}\text{-send}(k,(q,r)\mapsto\rightarrow(q,r))(q,\sigma(4\overline{r}))$, for a suitable error bound k, and symmetrically for $\leftarrow(q,r)$. However, if we do so, we still might have some multiplications in the above expressions.

The key is to use Lemma 23: we can also write the above expressions as

$$\begin{aligned}\overline{l'} = \textstyle\sum_{q,r} \quad & \Big[\mathcal{C}\text{-if}\left(2^{m+c},\mathcal{C}\text{-send}(2^2,(q,r)\mapsto\rightarrow(q,r))(q,\sigma(4\overline{r})),\tfrac{\overline{l}}{4}+\tfrac{x}{4}\right)\\ & + \mathcal{C}\text{-if}\left(2^{m+c},\mathcal{C}\text{-send}(2^2,(q,r)\mapsto\leftarrow(q,r))(q,\sigma(4\overline{r})),\xi(4\overline{l})\right)\Big]\end{aligned}$$

$$\begin{aligned}\overline{r'} = \textstyle\sum_{q,r} \quad & \Big[\mathcal{C}\text{-if}\left(2^{m+c},\mathcal{C}\text{-send}(2^2,(q,r)\mapsto\rightarrow(q,r))(q,\sigma(4\overline{r})),\xi(4\overline{r})\right)\\ & + \mathcal{C}\text{-if}\left(2^{m+c},\mathcal{C}\text{-send}(2^2,(q,r)\mapsto\leftarrow(q,r))(q,\sigma(4\overline{r})),\tfrac{\xi(4r)}{4^2}+\tfrac{x}{4^2}+\tfrac{\sigma(4\overline{l})}{4}\right)\Big]\end{aligned}$$

and still have the same bound on the error. ◀

Once we have one step, we would like to simulate some arbitrary computation of a Turing machine, by considering the iterations of function *Next*.

The problem of above construction, is that, even if we start from the exact encoding $\overline{C}$ of a configuration, it introduces some error (even if at most 2^{-m}). If we want to apply again the function *Next*, then we will start not exactly from the encoding of a configuration.

Looking at the choice of the function σ, a small error can be tolerated (roughly if the process does not involve points at distance less than $1/4$ of $\mathcal{I}$), but this error is amplified (roughly multiplied by 4 on some component), before introducing some new errors (even if at most 2^{-m}). The point is that if we repeat the process, very soon it will be amplified, up to a level where we have no true idea or control about what becomes the value of above function.

However, if we know some bound on the space used by the Turing machine, we can correct it to get at most some fixed additive error: a Turing machine using a space S uses at most S cells to the right and to the left of the initial position of its head. Consequently, a configuration $C = (q, l, r)$ of such a machine involves words l and r of length at most S. Their encoding $\bar{l}$, and $\bar{r}$ are expected to remain in $\mathcal{I}_{S+1}$. Consider $\text{round}_{S+1}(\bar{l}) = \lfloor 4^{S+1}\bar{l} \rfloor / 4^{S+1}$. For a point $\bar{l}$ of $\mathcal{I}_{S+1}$, $4^{S+1}\bar{l}$ is an integer, and $\bar{l} = \text{round}_{S+1}(\bar{l})$. But now, for a point $\tilde{\bar{l}}$ at distance less than $4^{-(S+2)}$ from a point $\bar{l} \in \mathcal{I}_{S+1}$, $\text{round}_{S+1}(\tilde{\bar{l}}) = \bar{l}$. In other words, round_{S+1} "deletes" errors of order $4^{-(S+2)}$. Consequently, we can replace every $\bar{l}$ in above expressions by $\sigma_1(2^{2S+4}, 2^{2S+3}, 4^{S+1}\bar{l})/4^{S+1}$, as this is close to $\text{round}_{S+1}(\bar{l})$, and the same for $\bar{r}$, where σ_1 is the function from Corollary 22. We could also replace m by $m + 2S + 4$ to guarantee that $2^{-m} \leq 4^{-(S+2)}$. We get the following important improvement of the previous lemma:

▶ **Lemma 30.** *We can construct some function $\overline{Next}$ in $\mathbb{LDL}^{\circ}$ that simulates one step of M, i.e. that computes the Next function sending a configuration $\overline{C}$ of Turing machine M to $\overline{C}'$, where C' is the next one: $\|Next(2^m, 2^S, \overline{C}) - \overline{C}'\| \leq 2^{-m}$. Furthermore, it is robust to errors on its input, up to space S: considering $\|\tilde{C} - \overline{C}\| \leq 4^{-(S+2)}$, $\|Next(2^m, 2^S, \tilde{C}) - \overline{C}'\| \leq 2^{-m}$ remains true.*

▶ **Proposition 31.** *Consider some Turing machine M that computes some function $f : \Sigma^* \to \Sigma^*$ in some time $T(\ell(\omega))$ on input ω. One can construct some function $\tilde{f} : \mathbb{N}^2 \times \mathbb{R} \to \mathbb{R}$ in $\mathbb{LDL}^{\circ}$ that does the same: $\tilde{f}(2^m, 2^{T(\ell(\omega))}, \gamma_{word}(\omega))$ that is at most 2^{-m} far from $\gamma_{word}(f(\omega))$.*

Proof. The idea is to define the function $\overline{Exec}$ that maps some time 2^t and some initial configuration C to the configuration at time t. This can be obtained using previous lemma by $\overline{Exec}(2^m, 0, 2^T, C) = C$ and $\overline{Exec}(2^m, 2^{t+1}, 2^T, C) = \overline{Next}(2^m, 2^T, \overline{Exec}(2^m, 2^t, 2^T, C))$.

We can then get the value of the computation as $\overline{Exec}(2^m, 2^{T(\ell(\omega))}, 2^{T(\ell(\omega))}, C_{init})$ on input ω, considering $C_{init} = (q_0, 0, \gamma_{word}(\omega))$. By applying some projection, we get the following function $\tilde{f}(2^m, 2^T, y) = \pi_3^3(\overline{Exec}(2^m, 2^T, 2^T, (q_0, 0, y)))$ that satisfies the property. ◀

Actually, in order to get **FPSPACE**, observe that we can also replace the linear length ODE by a linear ODE.

▶ **Proposition 32.** *Consider some Turing machine M that computes some function $f : \Sigma^* \to \Sigma^*$ in some polynomial space $S(\ell(\omega))$ on input ω. One can construct some function $\tilde{f} : \mathbb{N}^2 \times \mathbb{R} \to \mathbb{R}$ in $\mathbb{RLD}^{\circ}$ that does the same: we have $\tilde{f}(2^m, 2^{S(\ell(\omega))}, \gamma_{word}(\omega))$ that is at most 2^{-m} far from $\gamma_{word}(f(\omega))$.*

Proof. The idea is the same, but not working with powers of 2, and with linear ODE: define the function $\overline{Exec}$ that maps some time t and some initial configuration C to the configuration at time t. This can be obtained using previous lemma by $\overline{Exec}(2^m, 0, 2^S, C) = C$ and $\overline{Exec}(2^m, t + 1, 2^S, C) = \overline{Next}(2^m, 2^S, \overline{Exec}(2^m, t, 2^S, C))$.

In order to claim this is a robust linear ODE, we need to state that $Exec(2^m, t, 2^S, C)$ is polynomially numerically stable: but this holds, since to estimate this value at 2^{-n} it is sufficient to work at precision $4^{-max(m,n,S+2)}$ (independently of t, from the rounding).

We can then get the value of the computation as $\overline{Exec}(2^m, 2^{S(\ell(\omega))}, 2^{S(\ell(\omega))}, C_{init})$ on input ω, considering $C_{init} = (q_0, 0, \gamma_{word}(\omega))$. By applying some projection, we get the following function $\tilde{f}(2^m, 2^S, y) = \pi_3^3(\overline{Exec}(2^m, S, 2^S, (q_0, 0, y)))$ that satisfies the property. ◀

5 Converting integers and dyadics to words, and conversely

One point of our simulations of Turing machines is that they work over $\mathcal{I}$, through encoding γ_{word}, while we would like to talk about integers and real numbers: we need to be able to convert an integer (more generally a dyadic) into some encoding over $\mathcal{I}$ and conversely.

Fix the following encoding: every digit in the binary expansion of d is encoded by a pair of symbols in the radix 4 expansion of $\overline{d} \in \mathcal{I} \cap [0,1]$: digit 0 (respectively: 1) is encoded by 11 (resp. 13) if before the "decimal" point in d, and digit 0 (respectively: 1) is encoded by 31 (resp. 33) if after. For example, for $d = 101.1$ in base 2, $\overline{d} = 0.13111333$ in base 4.

By iterating $\ell(n)$ times the function

$$F(\overline{r_1}, \overline{l_2}) = \begin{cases} (\div_2(\overline{r_1}), (\overline{l_2} + 5)/4) & \text{whenever } \operatorname{mod}_2(\overline{r_1}) = 0 \\ (\div_2(\overline{r_1}), (\overline{l_2} + 7)/4) & \text{whenever } \operatorname{mod}_2(\overline{r_1}) = 1. \end{cases}$$

over $(n, 0)$, and then projecting on the second argument, we can prove:

▶ **Lemma 33** (From $\mathbb{N}$ to $\mathcal{I}$). *We can construct some function $Decode : \mathbb{N}^2 \to \mathbb{R}$ in $\mathbb{LDL}^\circ$ that maps m and n to some point at distance less than 2^{-m} from $\gamma_{word}(\overline{n})$.*

This technique can be extended to consider decoding of tuples: there is a function $Decode : \mathbb{N}^{d+1} \to \mathbb{R}$ in $\mathbb{LDL}^\circ$ that maps m and $\mathbf{n}$ to some point at distance less than 2^{-m} from $\gamma_{word}(\overline{\mathbf{n}})$, with $\overline{\mathbf{n}}$ defined componentwise.

Conversely, given $\overline{d}$, we need a way to construct d. Actually, as we will need to avoid multiplications, we state that we can even do something stronger: given $\overline{d}$, and (some bounded) λ we can construct λd.

▶ **Lemma 34** (From $\mathcal{I}$ to $\mathbb{R}$, and multiplying in parallel). *We can construct some function $EncodeMul : \mathbb{N}^2 \times [0,1] \times \mathbb{R} \to \mathbb{R}$ in $\mathbb{LDL}^\circ$ that maps m, 2^S, $\gamma_{word}(\overline{d})$ and (bounded) λ to some real at distance at most 2^{-m} from λd, whenever $\overline{d}$ is of length less than S.*

6 Proofs and applications

When we say that a function $f : S_1 \times \cdots \times S_d \to \mathbb{R}^{d'}$ is (respectively: polynomial time or space) computable this will always be in the sense of computable analysis: see e.g. [7, 19]. We actually follow the formalisation in [3] of required concepts from computable analysis, able to mix complexity issues dealing with integer and real arguments. Theorem 8 follows from point 1. of next Proposition for one inclusion, and previous simulation of Turing machines for the other.

▶ **Proposition 35.**

1. *All functions of $\mathbb{LDL}^\circ$ are computable (in the sense of computable analysis) in polynomial time.*
2. *All functions of $\mathbb{RLD}$ are computable (in the sense of computable analysis) in polynomial space.*

The proof of the proposition consists in observing this holds for the basic functions and that composition preserves polynomial time (respectively: space) computability and also by linear length ODEs. This latter fact is established by computable analysis arguments, reasoning on some explicit formula giving the solution of linear length ODE. Regarding space, the main issue is the need to prove the schema given by Definition 18 guarantees $\mathbf{f}$ is in **FPSPACE**, when $\mathbf{u}$, $\mathbf{g}$, and $\mathbf{h}$ are. Assuming condition 1. of Definition 18 would not be sufficient: the problem is $\mathbf{f}(x, \mathbf{y})$ may polynomially grow too fast or have a modulus function that would grow too fast. The point is, in Definition 18, we assumed $\mathbf{f}$ to be both bounded and satisfying **2.**, i.e. polynomial numerical robustness. With these hypotheses, it is sufficient to work with the precision given by this robustness condition and these conditions guarantee the validity of computing with such approximated values.

We now go to various applications of it and of our toolbox. First, we state a characterisation of **FPTIME** for general functions, covering both the case of a function $\mathbf{f} : \mathbb{N}^d \to \mathbb{R}^{d'}$, $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^{d'}$ as a special case: only the first type (sequences) was covered by [3].

▶ **Theorem 36** (Theorem 9). *A function $\mathbf{f} : \mathbb{R}^d \times \mathbb{N}^{d''} \to \mathbb{R}^{d'}$ is computable in polynomial time iff there exists $\tilde{\mathbf{f}} : \mathbb{R}^d \times \mathbb{N}^{d''+2} \to \mathbb{R}^{d'} \in \mathbb{LDL}^{\circ}$ such that for all $\mathbf{x} \in \mathbb{R}^d$, $X \in \mathbb{N}$, $\mathbf{x} \in \left[-2^X, 2^X\right]$, $\mathbf{m} \in \mathbb{N}^{d''}$, $n \in \mathbb{N}$, $\|\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^n) - \mathbf{f}(\mathbf{x}, \mathbf{m})\| \leq 2^{-n}$.*

The reverse implication of Theorem 36 follows from Proposition 35, (1.) and arguments from computable analysis. For the direct implication, for sequences, that is to say, functions of type $\mathbf{f} : \mathbb{N}^{d''} \to \mathbb{R}^{d'}$ (i.e. $d = 0$, the case considered in [3]) we are almost done: reasoning componentwise, we only need to consider $f : \mathbb{N}^{d''} \to \mathbb{R}$ (i.e. $d' = 1$). As the function is polynomial time computable, this means that there is a polynomial time computable function $g : \mathbb{N}^{d''+1} \to \{1, 3\}^*$ so that on $\mathbf{m}, 2^n$, it provides the encoding $\overline{\phi(\mathbf{m}, n)}$ of some dyadic $\phi(\mathbf{m}, n)$ with $\|\phi(\mathbf{m}, n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$ for all $\mathbf{m}$. The problem is then to decode, compute and encode the result to produce this dyadic, using our previous toolbox.

More precisely, from Proposition 31, we get $\tilde{g}$ with $|\tilde{g}(2^e, 2^{p(max(\mathbf{m},n))}, \mathit{Decode}(2^e, \mathbf{m}, n)) - \gamma_{word}(g(\mathbf{m}, n))| \leq 2^{-e}$ for some polynomial p corresponding to the time required to compute g, and $e = \max(p(max(\mathbf{m}, n)), n)$. Then we need to transform the value to the correct dyadic: we mean $\tilde{\mathbf{f}}(\mathbf{m}, n) = \mathit{EncodeMul}(2^e, 2^t, \tilde{g}(2^e, 2^t, \mathit{Decode}(2^e, \mathbf{m}, n)), 1)$, where $t = p(max(\mathbf{m}, n))$, $e = \max(p(max(\mathbf{m}, n)), n)$ provides a solution such that $\|\tilde{\mathbf{f}}(\mathbf{m}, 2^n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$.

▶ Remark 37. This is basically what is done in [3], except that we do it here with analytic functions. However, as already observed in [3], this cannot be done for the case $d \geq 1$, i.e. for example for $f : \mathbb{R} \to \mathbb{R}$. The problem is that we used the fact that we can decode: *Decode* maps an integer n to its encoding $\overline{n}$ (but is not guaranteed to do something valid on non-integers). There cannot exist such functions that would be valid over all reals, as such function must be continuous, and there is no way to map continuously real numbers to finite words. This is where the approach of the article [3] is stuck.

To solve this, we use an adaptive barycentric technique. By lack of space, we discuss only the case of a polynomial time computable function $f : \mathbb{R} \times \mathbb{N} \to \mathbb{R}$. From standard arguments from computable analysis (see e.g. [Corollary 2.21][13]), the following holds and the point is to be able to realise all this with functions from $\mathbb{LDL}^{\circ}$.

▶ **Lemma 38.** *Assume $f : \mathbb{R} \times \mathbb{N} \to \mathbb{R}$ is computable in polynomial time. There exists some polynomial $m : \mathbb{N}^2 \to \mathbb{N}$ and some $\tilde{f} : \mathbb{N}^4 \to \mathbb{Z}$ computable in polynomial time such that for all $x \in \mathbb{R}$, $\|2^{-n}\tilde{f}(\lfloor 2^{m(n,M)}x \rfloor, u, 2^M, 2^n) - f(x, u)\| \leq 2^{-n}$ whenever $\frac{x}{2^{m(n,M)}} \in [-2^M, 2^M]$.*

Assume we consider an approximation σ_i (with either $i = 1$ or $i = 2$) of the integer part function given by Lemma 22. Then, given n, M, when $2^{m(n,M)}x$ falls in some suitable interval I_i for σ_i (see statement of Lemma 22), we are sure that $\sigma_i(2^e, 2^{m(n,M)+X+1}, 2^{m(n,M)}x)$ is at some distance upon control from $\lfloor 2^{m(n,M)}x \rfloor$. Consequently, $2^{-n}\tilde{f}(\sigma_i(2^{m(n,M)+X+1}, 2^{m(n,M)} x), u, 2^M, 2^n)$ provides some 2^{-n}-approximation of $f(x, u)$, up to some error upon control. When this holds, we then use an argument similar to what we describe for sequences: using functions from $\mathbb{LDL}^{\circ}$, we can decode, compute, and encode the result to provide this dyadic. It is provided by an expression $\mathit{Formula}_i(x, u, M, n)$ of the form $\mathit{EncodeMul}(2^e, 2^t, \tilde{\tilde{f}}(2^2, 2^t, \mathit{Decode}(2^e, \sigma_i(2^e, 2^M, 2^{m(n,M)}x))), 2^{-n})$.

The problem is that it might also be the case that $2^{m(n,M)}x$ falls in the complement of the intervals $(I_i)_i$. In that case, we have no clear idea of what could be the value of $\sigma_i(2^e, 2^{m(n,M)+X+1}, 2^{m(n,M)}x)$, and consequently of what might be the value of the above expression $\mathit{Formula}_i(x, u, M, n)$. But the point is that when it happens for an x for σ_1, we could have used σ_2, and this would work, as one can check that the intervals of type I_1 covers the complements of the intervals of type I_2 and conversely. They also overlap, but when x is both in some I_1 and I_2, $\mathit{Formula}_1(x, u, M, n)$ and $\mathit{Formula}_2(x, u, M, n)$ may differ, but they are both 2^{-n} approximation of $f(x)$.

The key is to compute some suitable "adaptive" barycenter, using function λ, provided by Corollary 22. Writing $\approx$ for the fact that two values are closed up to some controlled bounded error, observe from the statements of Lemma 22 that whenever $\lambda(\cdot, 2^n, x) \approx 0$, we know that $\sigma_2(\cdot, 2^n, x) \approx \lfloor x \rfloor$; whenever $\lambda(\cdot, 2^n, x) \approx 1$ we know that $\sigma_1(\cdot, 2^n, x) \approx \lfloor x \rfloor$; whenever $\lambda(\cdot, 2^n, x) \in (0, 1)$, we know that $\sigma_1(\cdot, 2^n, x) \approx \lfloor x \rfloor + 1$ and $\sigma_2(\cdot, 2^n, x) \approx \lfloor x \rfloor$. That means that if we consider $\lambda(\cdot, 2^n, x)\mathit{Formula}_1(x, u, M, n) + (1 - \lambda(\cdot, 2^n, n))\mathit{Formula}_2(x, u, M, n)$ we are sure to be close (up to some bounded error) to some 2^{-n} approximation of $f(x)$. There remains that this requires some multiplication with λ. But from the form of $\mathit{Formula}_i(x, u, M, n)$, this could be also be written as follows, ending the proof of Theorem 36.

$$\begin{aligned}&\mathit{EncodeMul}(2^e, 2^t, \tilde{\tilde{f}}(2^e, 2^t, \mathit{Decode}(2^e, \sigma_1(2^e, 2^M, 2^{m(n,M)}x))), \lambda(2^e, 2^M, 2^{m(n,M)}x)2^{-n}) +\\ &\quad \mathit{EncodeMul}(2^e, 2^t, \tilde{\tilde{f}}(2^e, 2^t, \mathit{Decode}(2^e, \sigma_2(2^e, 2^M, 2^{m(n,M)}x))), (1 - \lambda(2^e, 2^M, 2^{m(n,M)}x))2^{-n})\end{aligned} \tag{7}$$

From the fact that we have the reverse direction in Theorem 36, it is natural to consider the operation that maps $\tilde{\mathbf{f}}$ to $\mathbf{f}$. Namely, we introduce the operation ELim (ELim stands for Effective Limit):

▶ **Definition 39** (Operation ELim). *Given* $\tilde{\mathbf{f}} : \mathbb{R}^d \times \mathbb{N}^{d''} \times \mathbb{N} \to \mathbb{R}^{d'} \in \mathbb{LDL}^\circ$ *such that for all* $\mathbf{x} \in \mathbb{R}^d$, $X \in \mathbb{N}$, $\mathbf{x} \in [-2^X, 2^X]$, $\mathbf{m} \in \mathbb{N}^{d''}$, $n \in \mathbb{N}$, $\|\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^n) - \mathbf{f}(\mathbf{x}, \mathbf{m})\| \leq 2^{-n}$, *then* $\mathit{ELim}(\tilde{\mathbf{f}})$ *is the (clearly uniquely defined) corresponding function* $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^{d'}$.

▶ **Theorem 40.** *A continuous function* $\mathbf{f}$ *is computable in polynomial time if and only if all its components belong to* $\overline{\mathbb{LDL}^\circ}$, *where* $\overline{\mathbb{LDL}^\circ} = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \overline{cond}(x), \frac{x}{2}, \frac{x}{3}$; *composition, linear length ODE*, $\mathit{ELim}]$.

For the reverse direction, by induction, the only thing to prove is that the class of functions from to the integers computable in polynomial time is preserved by the operation ELim. Take such a function $\tilde{\mathbf{f}}$. By definition, given $\mathbf{x}, \mathbf{m}$, X we can compute $\tilde{f}(\mathbf{x}, \mathbf{m}, 2^X, 2^n)$ with precision 2^{-n} in time polynomial in n. This must be, by definition of ELim schema, some approximation of $\mathbf{f}(\mathbf{x}, \mathbf{m})$ over $[-2^X, 2^X]$, and hence $\mathbf{f}$ is computable in polynomial time. This also gives directly Theorem 9 as a corollary.

We obtain the statements for polynomial space computability (Theorems 10 and 11) replacing $\mathbb{LDL}^\circ$ by $\mathbb{RLD}^\circ$, using similar reasoning about space instead of time, considering point 2. instead of 1. of Proposition 35, and Proposition 32 instead of Proposition 31.

References

1 Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity*, 2:97–110, 1992.

2 Manon Blanc and Olivier Bournez. A characterization of polynomial time computable functions from the integers to the reals using discrete ordinary differential equations. Submitted. Journal version of [3]. Preliminary version available on `arXiv:2209.13599`.

3 Manon Blanc and Olivier Bournez. A characterization of polynomial time computable functions from the integers to the reals using discrete ordinary differential equations. In Jérôme Durand-Lose and György Vaszil, editors, *Machines, Computations, and Universality – 9th International Conference, MCU 2022, Debrecen, Hungary, August 31 – September 2, 2022, Proceedings*, volume 13419 of *Lecture Notes in Computer Science*, pages 58–74. Springer, 2022. MCU'22 Best Student Paper Award. `doi:10.1007/978-3-031-13502-6_4`.

4 Olivier Bournez and Arnaud Durand. Recursion schemes, discrete differential equations and characterization of polynomial time computation. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th Int Symposium on Mathematical Foundations of Computer Science, MFCS*, volume 138 of *LIPIcs*, pages 23:1–23:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

5 Olivier Bournez and Arnaud Durand. A characterization of functions over the integers computable in polynomial time using discrete ordinary differential equations. *Computational Complexity*, 32(2):7, 2023.

6 Olivier Bournez and Amaury Pouly. A universal ordinary differential equation. In *International Colloquium on Automata Language Programming, ICALP'2017*, 2017.

7 Vasco Brattka, Peter Hertling, and Klaus Weihrauch. A tutorial on computable analysis. In *New computational paradigms*, pages 425–491. Springer, 2008.

8 P. Clote. Computational models and function algebras. In Edward R. Griffor, editor, *Handbook of Computability Theory*, pages 589–681. North-Holland, Amsterdam, 1998.

9 Peter Clote and Evangelos Kranakis. *Boolean functions and computation models*. Springer Science & Business Media, 2013.

10 Alan Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam, 1962.

11 Francois Fages, Guillaume Le Guludec, Olivier Bournez, and Amaury Pouly. Strong turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs. In *Computational Methods in Systems Biology-CMSB 2017*, 2017. CMSB'2017 Best Paper Award.

12 Daniel S. Graça and Ning Zhong. *Handbook of Computability and Complexity in Analysis*, chapter Computability of Differential Equations, pages 71–99. Springer, 2018.

13 Ker-I Ko. *Complexity Theory of Real Functions*. Progress in Theoretical Computer Science. Birkhaüser, Boston, 1991.

14 D. Leivant. Intrinsic theories and computational complexity. In *LCC'94*, number 960 in Lecture Notes in Computer Science, pages 177–194, 1995.

15 Daniel Leivant. Predicative recurrence and computational complexity I: Word recurrence and poly-time. In Peter Clote and Jeffery Remmel, editors, *Feasible Mathematics II*, pages 320–343. Birkhäuser, 1994.

16 Daniel Leivant and Jean-Yves Marion. Lambda calculus characterizations of Poly-Time. *Fundamenta Informatica*, 19(1,2):167,184, 1993.

17 Daniel Leivant and Jean-Yves Marion. Ramified recurrence and computational complexity II: substitution and poly-space. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic, 8th Workshop, CSL'94*, volume 933 of *Lecture Notes in Computer Science*, pages 369–380, Kazimierz, Poland, 1995. Springer.

18 David B Thompson. Subrecursiveness: Machine-independent notions of computability in restricted time and storage. *Mathematical Systems Theory*, 6(1-2):3–15, 1972.

19 Klaus Weihrauch. *Computable Analysis: an Introduction*. Springer, 2000.