



HAL
open science

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, Olivier Bournez

► **To cite this version:**

Manon Blanc, Olivier Bournez. 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. 48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023), Aug 2023, Bordeaux, France. 10.4230/LIPIcs.MFCS.2023.21 . hal-04259380

HAL Id: hal-04259380

<https://hal.science/hal-04259380>

Submitted on 26 Oct 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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; Computer systems organisation → Analog computers; 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.2023.19

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



© Manon Blanc and Olivier Bournez;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 19; pp. 19:1–19:15



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

44 in [4, 5]. They provided a characterisation of **FPTIME** for functions over the integers
 45 that does not require the specification of an explicit bound in the recursion, in contrast to
 46 Cobham’s work [10], nor the assignment of a specific role or type to variables, in contrast to
 47 safe recursion or ramification [1, 14]. Instead, they only assume involved ODEs to be linear,
 48 a very classical natural concept for differential equations.

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

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

62 ▶ **Remark 2.** The literature considers two possible definitions for **FPSPACE**, according to
 63 whether functions with non-polynomial size values are allowed or not. In our case, we should
 64 add “whose outputs remain of polynomial size”, to resolve the ambiguity¹.

65 A main result of [4, 5] is the following (LDL stands for linear derivation on length):

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

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

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

70 That is to say, LDL (and hence **FPTIME** for functions over the integers) is the smallest
 71 class of functions that contains the constant functions $\mathbf{0}$ and $\mathbf{1}$, the projections π_i^k of the
 72 i^e coordinate of a vector of size k , the length function $\ell(x)$, mapping an integer to the
 73 length of its binary representation, the addition $x+y$, the subtraction $x-y$, the multiplication
 74 $x \times y$, the **sign** function $\text{sg}(x)$ and that is closed under composition (when defined) and
 75 linear length-ODE scheme: the linear length-ODE scheme, formally given by Definition 15,
 76 corresponds to defining a function from a linear ODEs with respect to derivation along the
 77 length of the argument, so of the form $\frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \ell} = \mathbf{A}[\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}] \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}[\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}]$.

78 Here, we use the notation $\frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \ell}$ which corresponds to the derivation of \mathbf{f} along the
 79 length function: given some function $\mathcal{L} : \mathbb{N}^{p+1} \rightarrow \mathbb{Z}$ and in particular for the case where
 80 $\mathcal{L}(x, \mathbf{y}) = \ell(x)$,

$$81 \quad \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \mathcal{L}} = \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \mathcal{L}(x, \mathbf{y})} = \mathbf{h}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}) \quad (1)$$

83 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:

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

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

86 However, in the context of (classical) ODEs, considering functions over the reals is more
87 natural than only functions over the integers. Call *real function* a function $f : S_1 \times \dots \times S_d \rightarrow$
88 $S'_1 \times \dots \times S'_d$, where each S_i, S'_i is either \mathbb{R}, \mathbb{N} or \mathbb{Z} . A natural question about the characterisation
89 of **FPTIME** for real functions arises, and not only discrete functions: we consider here
90 computability over the reals in its most classical approach, namely computable analysis [19].

91 As a first step, the class $\mathbb{L}\mathbb{D}\mathbb{L}^\bullet = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \times, \overline{\text{cond}}(x), \frac{x}{2}; \text{composition, linear}$
92 $\text{length ODE}]$ has been considered in [3, 2] where some characterisation of **P**TIME was
93 obtained, but only for functions from the integers to the reals (i.e. sequences) while it would
94 be more natural to characterise functions from the reals to the reals. More importantly, this
95 was obtained by assuming that some **non-analytic exact function** is among the basic
96 available functions to simulate a Turing machine: $\overline{\text{cond}}$ 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

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

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

100 ► Remark 6. This class is $\mathbb{L}\mathbb{D}\mathbb{L}$ but without the $\text{sg}(x)$ function, nor the multiplication function,
101 or $\mathbb{L}\mathbb{D}\mathbb{L}^\bullet$ but without the $\overline{\text{cond}}$ function, nor the multiplication. This is done by adding the
102 analytic \tanh functions as a substitute (and adding $x/3$).

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

108 First, we improve Theorem 3 by stating **FPTIME** over the integers can be characterised
109 algebraically using linear length ODEs and only analytic functions (i.e. no need for sign
110 function). Since $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$ is about functions over the reals, and Theorem 3 is about functions
111 over the integers, we need a way to compare these classes. Given a function $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$
112 sending every integer $\mathbf{n} \in \mathbb{N}^d$ to the vicinity of some integer of $\mathbb{N}^{d'}$, say at distance less than
113 $1/4$, we write $\text{DP}(f)$ for its discrete part: this is the function from $\mathbb{N}^d \rightarrow \mathbb{N}^{d'}$ mapping $\mathbf{n} \in \mathbb{N}^d$
114 to the integer rounding of $\mathbf{f}(n)$. Given a class \mathcal{C} of such functions, we write $\text{DP}(\mathcal{C})$ for the
115 class of the discrete parts of the functions of \mathcal{C} .

116 ► **Theorem 8.** $\text{DP}(\mathbb{L}\mathbb{D}\mathbb{L}^\circ) = \mathbf{FPTIME} \cap \mathbb{N}^{\mathbb{N}}$.

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

19:4 Simulation of Turing machines with analytic discrete ODEs. Applications.

120 ► **Theorem 9** (Generic functions over the reals). $\overline{\text{LDL}}^\circ \cap \mathbb{R}^{\mathbb{R}} = \mathbf{FPTIME} \cap \mathbb{R}^{\mathbb{R}}$

121 *More generally:* $\overline{\text{LDL}}^\circ \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}} = \mathbf{FPTIME} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}}.$

122 We also prove that, by adding a robust linear ODE scheme (Definition 18), we get a
 123 class RLD° (this stands for robust linear derivation) with the similar statements but for
 124 **FSPACE**.

125 ► **Theorem 10.** $\text{DP}(\text{RLD}^\circ) = \mathbf{FSPACE} \cap \mathbb{N}^{\mathbb{N}}.$

126 ► **Theorem 11** (Generic functions over the reals). $\overline{\text{RLD}}^\circ \cap \mathbb{R}^{\mathbb{R}} = \mathbf{FSPACE} \cap \mathbb{R}^{\mathbb{R}}$

127 *More generally:* $\overline{\text{RLD}}^\circ \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}} = \mathbf{FSPACE} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}}.$

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

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

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

2 Some concepts related to discrete ODEs

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

147 ► **Definition 12** ([3]). A \mathbf{tanh} -polynomial expression $P(x_1, \dots, x_h)$ is an expression built-on
 148 $+, -, \times$ (often denoted \cdot) and \mathbf{tanh} functions over a set of variables $V = \{x_1, \dots, x_h\}$ and
 149 integer constants.

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

153 ► **Definition 13** ([3]). The degree $\text{deg}(x, P)$ of a term P in $x \in V$ is defined inductively as
 154 follows: $\text{deg}(x, x) = 1$ and for $x' \in V \cup \mathbb{Z}$ such that $x' \neq x$, $\text{deg}(x, x') = 0$; $\text{deg}(x, P + Q) =$
 155 $\max\{\text{deg}(x, P), \text{deg}(x, Q)\}$; $\text{deg}(x, P \times Q) = \text{deg}(x, P) + \text{deg}(x, Q)$; $\text{deg}(x, \mathbf{tanh}(P)) = 0$.
 156 A \mathbf{tanh} -polynomial expression P is essentially constant in x if $\text{deg}(x, P) = 0$.

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

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

162 For example, the expression $P(x, y, z) = x \cdot \tanh(x^2 - z) \cdot y + y^3$ is essentially linear in
 163 x , essentially constant in z and not linear in y . The expression: $z + (1 - \tanh(x)) \cdot (1 -$
 164 $\tanh(-x)) \cdot (y - z)$ is essentially constant in x and linear in y and z .

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

$$167 \quad \mathbf{f}(0, \mathbf{y}) = \mathbf{g}(\mathbf{y}) \quad \text{and} \quad \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \ell} = \mathbf{u}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}). \quad (2)$$

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

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

$$172 \quad \mathbf{F}(1, \mathbf{y}) = \mathbf{g}(\mathbf{y}), \quad \text{and} \quad \frac{\partial \mathbf{F}(t, \mathbf{y})}{\partial t} = \mathbf{u}(\mathbf{F}(t, \mathbf{y}), \mathbf{h}(2^t - 1, \mathbf{y}), 2^t - 1, \mathbf{y}). \quad (3)$$

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

$$176 \quad \mathbf{f}(2^0, \mathbf{y}) = \mathbf{g}(\mathbf{y}), \quad \text{and} \quad \mathbf{f}(2^{t+1}, \mathbf{y}) = \bar{\mathbf{u}}(\mathbf{f}(2^t, \mathbf{y}), \mathbf{h}(2^t - 1, \mathbf{y}), 2^t, \mathbf{y}). \quad (4)$$

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

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

181 We assume it is easier for our reader to deal with recurrences of the form (4) than with
 182 ODEs of the form (2). Consequently, this is how we will describe many functions from now on,
 183 starting with some basic functions, authorising compositions, and the above schemes. As an
 184 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
 185 we can produce $n \mapsto 2^{p(n)}$ for any polynomial p . For example, $(n_1, \dots, n_k) \rightarrow 2^{n_1 n_2 \dots n_k}$ can
 186 be obtained, using k such schemes in turn, providing the case of the polynomial $p(n) = n^k$.

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

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

192 1. *it corresponds to the solution of*

$$193 \quad \mathbf{f}(0, \mathbf{y}) = \mathbf{g}(\mathbf{y}) \quad \text{and} \quad \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial x} = \mathbf{u}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}), \quad (5)$$

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

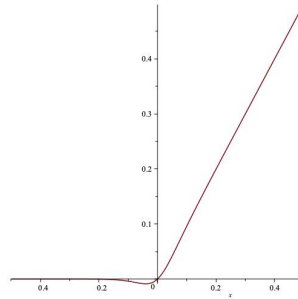
195 Here, writing $a \approx_n b$ for $\|a - b\| \leq 2^{-n}$ for conciseness, **2.** means formally there
 196 exists some polynomial p such that, for all integer n , writing $\epsilon(n) = p(n + \ell(\mathbf{y}))$, if you
 197 consider any solution of $\tilde{\mathbf{y}} =_{\epsilon(n)} \mathbf{y}$ 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
 198 $\frac{\partial \tilde{\mathbf{f}}(x, \tilde{\mathbf{y}})}{\partial x} =_{\epsilon(n)} \mathbf{u}(\tilde{\mathbf{f}}(x, \tilde{\mathbf{y}}), \tilde{\mathbf{h}}(x, \tilde{\mathbf{y}}), x, \tilde{\mathbf{y}})$ then $\tilde{\mathbf{f}}(x, \tilde{\mathbf{y}}) =_{\epsilon(n)} \mathbf{f}(x, \mathbf{y})$.

199 **3 Some results about various functions**

200 A key part of our proofs is the construction of very specific functions in $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$: we write
 201 $\{x\}$ for the fractional part of the real x , i.e. $\{x\} = x - \lfloor x \rfloor$. We provide some graphical
 202 representations of some of them to show that these functions are sometimes highly non-trivial
 203 (see for e.g. Figures 3 or 6).

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

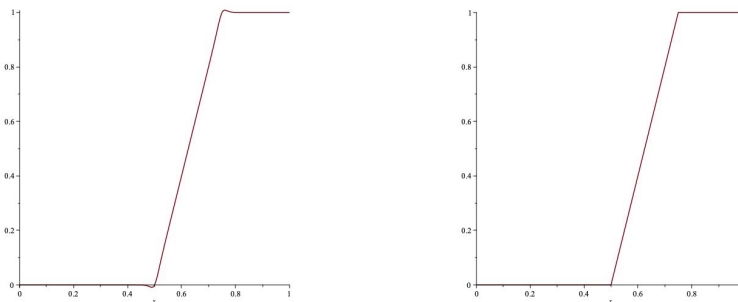
206 **► Lemma 19.** We denote by $Y(x, 2^{m+2})$ the function $Y(x, 2^{m+2}) = \frac{1+\tanh(2^{m+2}x)}{2}$ (illustrated
 207 by Figure 1). For all integer m , for all $x \in \mathbb{R}$, $|\text{ReLU}(x) - xY(x, 2^{m+2})| \leq 2^{-m}$.



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

208 We deduce we can uniformly approximate the continuous sigmoid functions (when $1/(b-a)$
 209 is in $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$) 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
 210 whenever $b \leq x$.

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

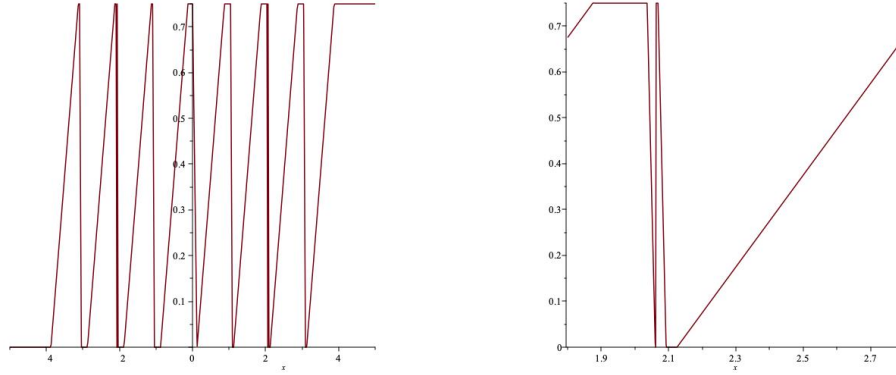


■ **Figure 2** Graphical representation of $\mathcal{C}\text{-}\mathfrak{s}(2, \frac{1}{2}, \frac{3}{4}, x)$ and $\mathcal{C}\text{-}\mathfrak{s}(2^5, \frac{1}{2}, \frac{3}{4}, x)$ obtained with maple.

214 **Proof.** We can write $\mathfrak{s}(a, b, x) = \frac{\text{ReLU}(x-a) - \text{ReLU}(x-b)}{b-a}$. Thus, $|\mathcal{C}\text{-}\mathfrak{s}(m+1+c, a, b, x) -$
 215 $\mathfrak{s}(a, b, x)| \leq \frac{2 \cdot 2^{-m-1-c}}{b-a}$, using the triangle inequality. Take c such that $\frac{1}{b-a} \leq 2^c$. ◀

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

218 ► **Theorem 21.** *There exists some function $\xi : \mathbb{N}^2 \rightarrow \mathbb{R}$ in LDL° (illustrated in Figure*
 219 *3) such that for all $n, m \in \mathbb{N}$ and $x \in [-2^n, 2^n]$, whenever $x \in [\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}]$,*
 220 *$|\xi(2^m, 2^n, x) - \{x - \frac{1}{8}\}| \leq 2^{-m}$.*

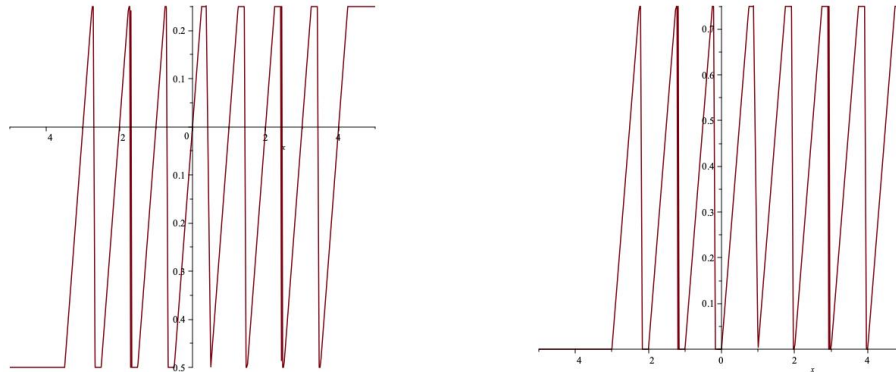


■ **Figure 3** Graphical representations of $\xi(2, 4, x)$ obtained with maple: some details on the right.

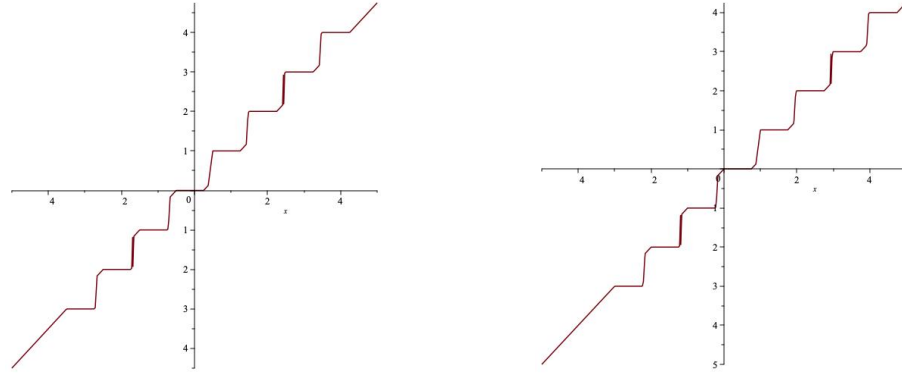
221 The main idea of the proof is, by parity, to reduce the problem to construct an auxiliary
 222 function ξ' that works for $x \geq 0$, writing $\xi(2^m, N, x) = \xi'(2^{m+2}, N, x) - \xi'(2^{m+2}, N, -x) + \frac{3}{4} -$
 223 $\frac{3}{4} \mathcal{C}\text{-s}(2^{m+2}, 0, \frac{1}{8}, x)$, and then proving that ξ' is definable in LDL° , using an adhoc recursive
 224 (in n) definition of it.

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

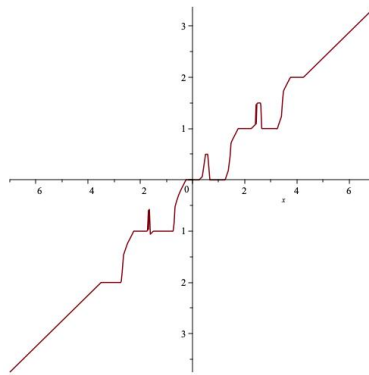
- 226 1. $\xi_1, \xi_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto \mathbb{R} \in \text{LDL}^\circ$ such that, for all $n, m \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever
 227 $x \in [\lfloor x \rfloor - \frac{1}{2}, \lfloor x \rfloor + \frac{1}{4}]$, $|\xi_1(2^m, 2^n, x) - \{x\}| \leq 2^{-m}$, and whenever $x \in [\lfloor x \rfloor, \lfloor x \rfloor + \frac{3}{4}]$,
 228 $|\xi_2(2^m, 2^n, x) - \{x\}| \leq 2^{-m}$ (see Figure 4).
- 229 2. $\sigma_1, \sigma_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto \mathbb{R} \in \text{LDL}^\circ$ such that, for all $n, m \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever
 230 $x \in [\lfloor x \rfloor - \frac{1}{2}, \lfloor x \rfloor + \frac{1}{4}]$, $\sigma_1(2^m, 2^n, x) - \lfloor x \rfloor \leq 2^{-m}$, and whenever $x \in [\lfloor x \rfloor, \lfloor x \rfloor + \frac{3}{4}]$,
 231 $|\sigma_2(2^m, 2^n, x) - \lfloor x \rfloor| \leq 2^{-m}$ (see Figure 5).
- 232 3. $\lambda : \mathbb{N}^2 \times \mathbb{R} \mapsto [0, 1] \in \text{LDL}^\circ$ such that for all $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever
 233 $x \in [\lfloor x \rfloor + \frac{1}{4}, \lfloor x \rfloor + \frac{1}{2}]$, $|\lambda(2^m, 2^n, x) - 0| \leq 2^{-m}$, and whenever $x \in [\lfloor x \rfloor + \frac{3}{4}, \lfloor x \rfloor + 1]$,
 234 $|\lambda(2^m, 2^n, x) - 1| \leq 2^{-m}$.
- 235 4. $\text{mod}_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto [0, 1] \in \text{LDL}^\circ$ such that for all $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever
 236 $x \in [\lfloor x \rfloor - \frac{1}{4}, \lfloor x \rfloor + \frac{1}{4}]$, $|\text{mod}_2(2^m, 2^n, x) - \lfloor x \rfloor \bmod 2| \leq 2^{-m}$.



■ **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.

237 **5.** $\div_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto [0, 1] \in \mathbb{L}\mathbb{D}\mathbb{L}^\circ$ such that for all $m, n \in \mathbb{N}$, $[x] \in [-2^n + 1, 2^n]$, whenever
 238 $x \in [[x] - \frac{1}{4}, [x] + \frac{1}{4}]$, $|\div_2(2^m, 2^n, x) - [x]/2| \leq 2^{-m}$, with // the integer division (see
 239 *Figure 6*).

240 **Proof.** Take $\xi_1(M, N, x) = \xi(M, N, x - \frac{3}{8}) - \frac{1}{2}$, $\xi_2(M, N, x) = \xi(N, x - \frac{7}{8})$, $\sigma_i(M, N, x) =$
 241 $x - \xi_i(M, N, x)$, $\lambda(M, N, x) = \mathcal{C}\text{-}\mathfrak{s}(2M, 1/4, 1/2, \xi(2M, N, x - 9/8))$, $\text{mod}_2(M, N, x) = 1 -$
 242 $\lambda(M, N/2, \frac{1}{2}x + \frac{7}{8})$, $\div_2(M, N, x) = \frac{1}{2}(\sigma_1(M, N, x) - \text{mod}_2(M, N, x))$. ◀

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

245 ▶ **Lemma 23.** *There exists $\mathcal{C}\text{-if} \in \mathbb{L}\mathbb{D}\mathbb{L}^\circ$ such that, $l \in [0, 1]$, if we take $|d' - 0| \leq 1/4$, then*
 246 $|\mathcal{C}\text{-if}(d', l) - 0| \leq 2^{-m}$, and if we take $|d' - 1| \leq 1/4$, then $|\mathcal{C}\text{-if}(d', l) - l| \leq 2^{-m}$.

247 ▶ **Lemma 24.** *Let $\alpha_1, \alpha_2, \dots, \alpha_n$ be some integers, and V_1, V_2, \dots, V_n some constants. We*
 248 *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 ,*
 249 *for all $i \in \{1, \dots, n\}$.*

250 *There is some function in $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$, that we write $\mathcal{C}\text{-send}(2^m, \alpha_i \mapsto V_i)_{i \in \{1, \dots, n\}}$, that maps*
 251 *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 $i \in \{1, \dots, n\}$.*

252 ▶ **Lemma 25.** *Let N be some integer. Let $\alpha_1, \alpha_2, \dots, \alpha_n$ be some integers, and $V_{i,j}$ for $1 \leq$
 253 $i \leq n$ some constants, with $0 \leq j < N$. We write $\text{send}((\alpha_i, j) \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}}$*

254 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
 255 all $i \in \{1, \dots, n\}$, $j \in \{0, \dots, N - 1\}$.

256 There is some function in $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$, that we write $\mathcal{C}\text{-send}(2^m, (\alpha_i, j) \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}}$,
 257 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
 258 2^{-m} of $V_{i,j}$, for all $i \in \{1, \dots, n\}$, $j \in \{0, \dots, N - 1\}$.

259 4 Simulating Turing machines with functions of $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$

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

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

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

266 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
 267 of the tape of the Turing machine M . In this representation, the head is in front of symbol
 268 r_0 , and $l_i, r_i \in \{0, 1, 3\}$ for all i . Such a configuration C can be denoted by $C = (q, l, r)$,
 269 where $l, r \in \Sigma^\omega$ are words over alphabet $\Sigma = \{0, 1, 3\}$ and $q \in Q$ denotes the internal state of
 270 M . Write: $\gamma_{word} : \Sigma^\omega \rightarrow \mathbb{R}$ for the function that maps a word $w = w_0 w_1 w_2 \dots$ to the dyadic
 271 $\gamma_{word}(w) = \sum_{n \geq 0} w_n 4^{-(n+1)}$.

272 The idea is that such a configuration C can also be encoded by some element $\bar{C} =$
 273 $(q, \bar{l}, \bar{r}) \in \mathbb{N} \times \mathbb{R}^2$, by considering $\bar{r} = \gamma_{word}(r)$ and $\bar{l} = \gamma_{word}(l)$. In other words, we encode
 274 the configuration of a bi-infinite tape Turing machine M by real numbers using their radix
 275 4 encoding, but using only digits 1,3. Notice that this lives in $Q \times [0, 1]^2$. Denoting the
 276 image of $\gamma_{word} : \Sigma^\omega \rightarrow \mathbb{R}$ by \mathcal{I} , this even lives in $Q \times \mathcal{I}^2$.

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

280 ▶ **Lemma 28.** We can construct some function \overline{Next} in $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$ that simulates one step of
 281 M : given a configuration C , writing C' for the next configuration, we have for all integer m ,
 282 $\|\overline{Next}(2^m, \bar{C}) - \bar{C}'\| \leq 2^{-m}$.

283 **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
 284 l^\bullet and r^\bullet corresponding to the (possibly infinite) word $l_{-1} l_{-2} \dots$ and $r_1 r_2 \dots$ respectively.

$$\begin{array}{ccccccc}
 \dots & l^\bullet & | & l_0 & | & r_0 & | & r^\bullet & \dots \\
 \hline
 & & & \underbrace{\hspace{2cm}} & & \underbrace{\hspace{2cm}} & & & \\
 & & & l & & r & & &
 \end{array}$$

286 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
 287 a definition by case of type:

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

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

$$\begin{aligned}
 \overline{Next}(q, \bar{l}, \bar{r}) &= \overline{Next}(q, \bar{l}^\bullet \bar{l}_0, \bar{r}_0 \bar{r}^\bullet) = (q', \bar{l}', \bar{r}') \\
 &= \begin{cases} (q', \bar{l}^\bullet \bar{l}_0 x, \bar{r}^\bullet) & \text{whenever } \delta(q, r_0) = (q', x, \rightarrow) \\ (q', \bar{l}^\bullet, \bar{l}_0 x \bar{r}^\bullet) & \text{whenever } \delta(q, r_0) = (q', x, \leftarrow) \end{cases}
 \end{aligned}$$

294

- in the first case “ \rightarrow ” : $\bar{l} = 4^{-1}\bar{l} + 4^{-1}x$ and $\bar{r}' = \bar{r}^\bullet = \{4\bar{r}\}$
- in the second case “ \leftarrow ” : $\bar{l} = \bar{l}^\bullet = \{4\bar{l}\}$ and $\bar{r}' = 4^{-2}\{4\bar{r}\} + 4^{-2}x + \lfloor 4\bar{l} \rfloor / 4$

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 $nextq_a^q = q'$ if $\delta(q, a) = (q', \cdot, \cdot)$, i.e. values (q', x, m) for some x and $m \in \{\leftarrow, \rightarrow\}$.

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

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 $\lfloor 4\bar{r} \rfloor$, either it values 0 (this is the specific case where there remain only blanks in r), or that $4\bar{r}$ lives in an interval $[1, 2]$ or in interval $[3, 4]$. That means that we could replace $\lfloor 4\bar{r} \rfloor$ by $\sigma(4\bar{r})$ 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 $\lfloor 4\bar{r} \rfloor$ 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 $\xi(4\bar{r})$ would be the same as $\{4\bar{r}\}$: that is, considering $r_0 = \sigma(4\bar{r})$, 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 $(3 + 1/4^2)3|Q| \leq 2^c$. This would just introduce some error at most $(3 + 1/4^2)3|Q|2^{-c}2^{-m} \leq 2^{-m}$.

► Remark 29. We could also replace every $\rightarrow(q, r)$ in above expressions for \bar{l}' and \bar{r}' by $\mathcal{C}\text{-}\text{send}(k, (q, r) \mapsto \rightarrow(q, r))(q, \sigma(4\bar{r}))$, 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

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

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 \bar{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{l} at distance less than $4^{-(S+2)}$ from a point $\bar{l} \in \mathcal{I}_{S+1}$, $\text{round}_{S+1}(\tilde{l}) = \bar{l}$. In other words, round_{S+1} “deletes” errors of order $4^{-(S+2)}$. Consequently, we can replace every \bar{l} in above expressions by $\sigma_1(2^{2S+4}, 2^{2S+3}, 4^{S+1} \tilde{l}) / 4^{S+1}$, 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 $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$ that simulates one step of M , i.e. that computes the Next function sending a configuration \bar{C} of Turing machine M to \bar{C}' , where C' is the next one: $\|Next(2^m, 2^S, \bar{C}) - \bar{C}'\| \leq 2^{-m}$. Furthermore, it is robust to errors on its input, up to space S : considering $\|\tilde{C} - \bar{C}\| \leq 4^{-(S+2)}$, $\|Next(2^m, 2^S, \tilde{C}) - \bar{C}'\| \leq 2^{-m}$ remains true.*

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

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 $\overline{Exec}(2^m, 2^{T(\ell(\omega))}, 2^{T(\ell(\omega))}, C_{\text{init}})$ on input ω , considering $C_{\text{init}} = (q_0, 0, \gamma_{\text{word}}(\omega))$. 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 $f : \Sigma^* \rightarrow \Sigma^*$ in some polynomial space $S(\ell(\omega))$ on input ω . One can construct some function $\tilde{f} : \mathbb{N}^2 \times \mathbb{R} \rightarrow \mathbb{R}$ in $\mathbb{R}\mathbb{L}\mathbb{D}^\circ$ that does the same: we have $\tilde{f}(2^m, 2^{S(\ell(\omega))}, \gamma_{\text{word}}(\omega))$ that is at most 2^{-m} far from $\gamma_{\text{word}}(f(\omega))$.*

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).

373 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
 374 ω , considering $C_{init} = (q_0, 0, \gamma_{word}(\omega))$. By applying some projection, we get the following
 375 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. ◀

376 5 Converting integers and dyadics to words, and conversely

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

380 Fix the following encoding: every digit in the binary expansion of d is encoded by a pair
 381 of symbols in the radix 4 expansion of $\bar{d} \in \mathcal{I} \cap [0, 1]$: digit 0 (respectively: 1) is encoded by
 382 11 (resp. 13) if before the “decimal” point in d , and digit 0 (respectively: 1) is encoded by 31
 383 (resp. 33) if after. For example, for $d = 101.1$ in base 2, $\bar{d} = 0.13111333$ in base 4.

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

$$F(\bar{r}_1, \bar{l}_2) = \begin{cases} (\div_2(\bar{r}_1), (\bar{l}_2 + 5)/4) & \text{whenever } \text{mod}_2(\bar{r}_1) = 0 \\ (\div_2(\bar{r}_1), (\bar{l}_2 + 7)/4) & \text{whenever } \text{mod}_2(\bar{r}_1) = 1. \end{cases}$$

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

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

387 This technique can be extended to consider decoding of tuples: there is a function
 388 $Decode : \mathbb{N}^{d+1} \rightarrow \mathbb{R}$ in LDL° that maps m and \mathbf{n} to some point at distance less than 2^{-m}
 389 from $\gamma_{word}(\bar{\mathbf{n}})$, with $\bar{\mathbf{n}}$ defined componentwise.

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

393 ▶ **Lemma 34** (From \mathcal{I} to \mathbb{R} , and multiplying in parallel). *We can construct some function
 394 $EncodeMul : \mathbb{N}^2 \times [0, 1] \times \mathbb{R} \rightarrow \mathbb{R}$ in LDL° that maps m , 2^S , $\gamma_{word}(\bar{d})$ and (bounded) λ
 395 to some real at distance at most 2^{-m} from λd , whenever \bar{d} is of length less than S .*

396 6 Proofs and applications

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

403 ▶ **Proposition 35. 1.** *All functions of LDL° are computable (in the sense of computable
 404 analysis) in polynomial time.*
 405 **2.** *All functions of RLD are computable (in the sense of computable analysis) in polynomial
 406 space.*

407 The proof of the proposition consists in observing this holds for the basic functions and
 408 that composition preserves polynomial time (respectively: space) computability and also
 409 by linear length ODEs. This latter fact is established by computable analysis arguments,

410 reasoning on some explicit formula giving the solution of linear length ODE. Regarding space,
 411 the main issue is the need to prove the schema given by Definition 18 guarantees \mathbf{f} is in
 412 **FPSPACE**, when \mathbf{u} , \mathbf{g} , and \mathbf{h} are. Assuming condition 1. of Definition 18 would not be
 413 sufficient: the problem is $\mathbf{f}(x, \mathbf{y})$ may polynomially grow too fast or have a modulus function
 414 that would grow too fast. The point is, in Definition 18, we assumed \mathbf{f} to be both bounded
 415 and satisfying **2.**, i.e. polynomial numerical robustness. With these hypotheses, it is sufficient
 416 to work with the precision given by this robustness condition and these conditions guarantee
 417 the validity of computing with such approximated values.

418 We now go to various applications of it and of our toolbox. First, we state a characterisation
 419 of **FPTIME** for general functions, covering both the case of a function $\mathbf{f} : \mathbb{N}^d \rightarrow \mathbb{R}^{d'}$,
 420 $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ as a special case: only the first type (sequences) was covered by [3].

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

424 The reverse implication of Theorem 36 follows from Proposition 35, (1.) and arguments
 425 from computable analysis. For the direct implication, for sequences, that is to say, functions
 426 of type $\mathbf{f} : \mathbb{N}^{d''} \rightarrow \mathbb{R}^{d'}$ (i.e. $d = 0$, the case considered in [3]) we are almost done: reasoning
 427 componentwise, we only need to consider $f : \mathbb{N}^{d''} \rightarrow \mathbb{R}$ (i.e. $d' = 1$). As the function is
 428 polynomial time computable, this means that there is a polynomial time computable function
 429 $g : \mathbb{N}^{d''+1} \rightarrow \{1, 3\}^*$ so that on $\mathbf{m}, 2^n$, it provides the encoding $\phi(\mathbf{m}, n)$ of some dyadic
 430 $\phi(\mathbf{m}, n)$ with $\|\phi(\mathbf{m}, n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$ for all \mathbf{m} . The problem is then to decode, compute
 431 and encode the result to produce this dyadic, using our previous toolbox.

432 More precisely, from Proposition 31, we get \tilde{g} with $|\tilde{g}(2^e, 2^{p(\max(\mathbf{m}, n))}, \text{Decode}(2^e, \mathbf{m}, n)) -$
 433 $\gamma_{\text{word}}(g(\mathbf{m}, n))| \leq 2^{-e}$ for some polynomial p corresponding to the time required to compute
 434 g , and $e = \max(p(\max(\mathbf{m}, n)), n)$. Then we need to transform the value to the correct dyadic:
 435 we mean $\tilde{\mathbf{f}}(\mathbf{m}, n) = \text{EncodeMul}(2^e, 2^t, \tilde{g}(2^e, 2^t, \text{Decode}(2^e, \mathbf{m}, n)), 1)$, where $t = p(\max(\mathbf{m}, n))$,
 436 $e = \max(p(\max(\mathbf{m}, n)), n)$ provides a solution such that $\|\tilde{\mathbf{f}}(\mathbf{m}, 2^n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$.

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

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

448 ► **Lemma 38.** *Assume $f : \mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R}$ is computable in polynomial time. There exists some*
 449 *polynomial $m : \mathbb{N}^2 \rightarrow \mathbb{N}$ and some $\tilde{f} : \mathbb{N}^4 \rightarrow \mathbb{Z}$ computable in polynomial time such that for*
 450 *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 $\frac{x}{2^{m(n, M)}} \in [-2^M, 2^M]$.*

451 Assume we consider an approximation σ_i (with either $i = 1$ or $i = 2$) of the integer part
 452 function given by Lemma 22. Then, given n, M , when $2^{m(n, M)} x$ falls in some suitable interval
 453 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
 454 some distance upon control from $\lfloor 2^{m(n, M)} x \rfloor$. Consequently, $2^{-n} \tilde{f}(\sigma_i(2^{m(n, M)+X+1}, 2^{m(n, M)}$
 455 $x), u, 2^M, 2^n)$ provides some 2^{-n} -approximation of $f(x, u)$, up to some error upon control.

456 When this holds, we then use an argument similar to what we describe for sequences: using
 457 functions from $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$, we can decode, compute, and encode the result to provide this dyadic.
 458 It is provided by an expression $Formula_i(x, u, M, n)$ of the form $EncodeMul(2^e, 2^t, \tilde{f}(2^2, 2^t,$
 459 $Decode(2^e, \sigma_i(2^e, 2^M, 2^{m(n,M)}x))), 2^{-n})$.

460 The problem is that it might also be the case that $2^{m(n,M)}x$ falls in the complement
 461 of the intervals $(I_i)_i$. In that case, we have no clear idea of what could be the value of
 462 $\sigma_i(2^e, 2^{m(n,M)+X+1}, 2^{m(n,M)}x)$, and consequently of what might be the value of the above
 463 expression $Formula_i(x, u, M, n)$. But the point is that when it happens for an x for σ_1 , we
 464 could have used σ_2 , and this would work, as one can check that the intervals of type I_1 covers
 465 the complements of the intervals of type I_2 and conversely. They also overlap, but when x is
 466 both in some I_1 and I_2 , $Formula_1(x, u, M, n)$ and $Formula_2(x, u, M, n)$ may differ, but they
 467 are both 2^{-n} approximation of $f(x)$.

468 The key is to compute some suitable "adaptive" barycenter, using function λ , provided by
 469 Corollary 22. Writing \approx for the fact that two values are closed up to some controlled bounded
 470 error, observe from the statements of Lemma 22 that whenever $\lambda(\cdot, 2^n, x) \approx 0$, we know
 471 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
 472 $\lambda(\cdot, 2^n, x) \in (0, 1)$, 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
 473 that if we consider $\lambda(\cdot, 2^n, x)Formula_1(x, u, M, n) + (1 - \lambda(\cdot, 2^n, x))Formula_2(x, u, M, n)$ we are
 474 sure to be close (up to some bounded error) to some 2^{-n} approximation of $f(x)$. There remains
 475 that this requires some multiplication with λ . But from the form of $Formula_i(x, u, M, n)$,
 476 this could be also be written as follows, ending the proof of Theorem 36.
 477

$$478 \quad EncodeMul(2^e, 2^t, \tilde{f}(2^e, 2^t, Decode(2^e, \sigma_1(2^e, 2^M, 2^{m(n,M)}x))), \lambda(2^e, 2^M, 2^{m(n,M)}x)2^{-n}) +$$

$$479 \quad EncodeMul(2^e, 2^t, \tilde{f}(2^e, 2^t, Decode(2^e, \sigma_2(2^e, 2^M, 2^{m(n,M)}x))), (1 - \lambda(2^e, 2^M, 2^{m(n,M)}x))2^{-n})$$

(7)

480 From the fact that we have the reverse direction in Theorem 36, it is natural to consider
 481 the operation that maps $\tilde{\mathbf{f}}$ to \mathbf{f} . Namely, we introduce the operation $ELim$ ($ELim$ stands
 482 for Effective Limit):
 483

484 ► **Definition 39** (Operation $ELim$). Given $\tilde{\mathbf{f}} : \mathbb{R}^d \times \mathbb{N}^{d''} \times \mathbb{N} \rightarrow \mathbb{R}^d \in \mathbb{L}\mathbb{D}\mathbb{L}^\circ$ such that for all
 485 $\mathbf{x} \in \mathbb{R}^d$, $X \in \mathbb{N}$, $\mathbf{x} \in [-2^X, 2^X]$, $\mathbf{m} \in \mathbb{N}^{d''}$, $n \in \mathbb{N}$, $\|\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^n) - \mathbf{f}(\mathbf{x}, \mathbf{m})\| \leq 2^{-n}$, then
 486 $ELim(\tilde{\mathbf{f}})$ is the (clearly uniquely defined) corresponding function $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^d$.

487 ► **Theorem 40.** A continuous function \mathbf{f} is computable in polynomial time if and only if all its
 488 components belong to $\overline{\mathbb{L}\mathbb{D}\mathbb{L}^\circ}$, where $\overline{\mathbb{L}\mathbb{D}\mathbb{L}^\circ} = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \overline{cond}(x), \frac{x}{2}, \frac{x}{3}; \text{composition},$
 489 $\text{linear length ODE}, ELim]$.

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

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

499 ——— References ———

- 500 1 Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the
 501 poly-time functions. *Computational Complexity*, 2:97–110, 1992.

- 502 2 Manon Blanc and Olivier Bournez. A characterization of polynomial time computable functions
503 from the integers to the reals using discrete ordinary differential equations. Submitted. Journal
504 version of [3]. Preliminary version available on <https://arxiv.org/abs/2209.13599>.
- 505 3 Manon Blanc and Olivier Bournez. A characterization of polynomial time computable functions
506 from the integers to the reals using discrete ordinary differential equations. In Jérôme Durand-
507 Lose and György Vaszil, editors, *Machines, Computations, and Universality - 9th International*
508 *Conference, MCU 2022, Debrecen, Hungary, August 31 - September 2, 2022, Proceedings*,
509 volume 13419 of *Lecture Notes in Computer Science*, pages 58–74. Springer, 2022. MCU'22
510 Best Student Paper Award. doi:10.1007/978-3-031-13502-6_4.
- 511 4 Olivier Bournez and Arnaud Durand. Recursion schemes, discrete differential equations and
512 characterization of polynomial time computation. In Peter Rossmanith, Pinar Heggernes, and
513 Joost-Pieter Katoen, editors, *44th Int Symposium on Mathematical Foundations of Computer*
514 *Science, MFCS*, volume 138 of *LIPICs*, pages 23:1–23:14. Schloss Dagstuhl - Leibniz-Zentrum
515 für Informatik, 2019.
- 516 5 Olivier Bournez and Arnaud Durand. A characterization of functions over the integers
517 computable in polynomial time using discrete ordinary differential equations. *Computational*
518 *Complexity*, 32(2):7, 2023.
- 519 6 Olivier Bournez and Amaury Pouly. A universal ordinary differential equation. In *International*
520 *Colloquium on Automata Language Programming, ICALP'2017*, 2017.
- 521 7 Vasco Brattka, Peter Hertling, and Klaus Weihrauch. A tutorial on computable analysis. In
522 *New computational paradigms*, pages 425–491. Springer, 2008.
- 523 8 P. Clote. Computational models and function algebras. In Edward R. Griffor, editor, *Handbook*
524 *of Computability Theory*, pages 589–681. North-Holland, Amsterdam, 1998.
- 525 9 Peter Clote and Evangelos Kranakis. *Boolean functions and computation models*. Springer
526 Science & Business Media, 2013.
- 527 10 Alan Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor,
528 *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*,
529 pages 24–30. North-Holland, Amsterdam, 1962.
- 530 11 Francois Fages, Guillaume Le Guludec, Olivier Bournez, and Amaury Pouly. Strong turing
531 completeness of continuous chemical reaction networks and compilation of mixed analog-digital
532 programs. In *Computational Methods in Systems Biology-CMSB 2017*, 2017. CMSB'2017 Best
533 Paper Award.
- 534 12 Daniel S. Graça and Ning Zhong. *Handbook of Computability and Complexity in Analysis*,
535 chapter Computability of Differential Equations. Springer., 2018.
- 536 13 Ker-I Ko. *Complexity Theory of Real Functions*. Progress in Theoretical Computer Science.
537 Birkhäuser, Boston, 1991.
- 538 14 D. Leivant. Intrinsic theories and computational complexity. In *LCC'94*, number 960 in
539 *Lecture Notes in Computer Science*, pages 177–194, 1995.
- 540 15 Daniel Leivant. Predicative recurrence and computational complexity I: Word recurrence
541 and poly-time. In Peter Clote and Jeffery Remmel, editors, *Feasible Mathematics II*, pages
542 320–343. Birkhäuser, 1994.
- 543 16 Daniel Leivant and Jean-Yves Marion. Lambda calculus characterizations of Poly-Time.
544 *Fundamenta Informatica*, 19(1,2):167,184, 1993.
- 545 17 Daniel Leivant and Jean-Yves Marion. Ramified recurrence and computational complexity II:
546 substitution and poly-space. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic,*
547 *8th Workshop, CSL'94*, volume 933 of *Lecture Notes in Computer Science*, pages 369–380,
548 Kazimierz, Poland, 1995. Springer.
- 549 18 David B Thompson. Subrecursiveness: Machine-independent notions of computability in
550 restricted time and storage. *Mathematical Systems Theory*, 6(1-2):3–15, 1972.
- 551 19 Klaus Weihrauch. *Computable Analysis: an Introduction*. Springer, 2000.