Journal of Logic & Analysis 17:FDS5 (2025) 1–42
ISSN 1759-9008

Simulation of Turing machines with analytic discrete ODEs: Polynomial–time and space over the reals characterised with discrete ordinary differential equations

MANON BLANC
OLIVIER BOURNEZ

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 characterise functions computable in polynomial space over the reals.

While existing characterisations could only cover time complexity or were restricted to functions over the integers, here we deal with real numbers and space complexity. Furthermore, 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.

2020 Mathematics Subject Classification 03D15, 03D20, 68Q15, 68Q19, 39A05, 26E05 (primary)

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

1 Introduction and related work

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 [20] gave birth to the field of *implicit complexity* at the interplay of logic and theory of programming. Alternative characterisations based on safe recursion, by Bellantoni and Coook [1], or on ramification (see Leivant [30] and Leivant and Marion [31]) or for other classes [32]

Published: February 2025 https://doi.org/10.4115/jla.2025.17.FDS5

followed: see Clote [18] and Clote and Kranakis [19] for monographs. Recent results include Férée, Hainry, Hoyrup and Péchoux [23] and Hainry, Mazza and Péchoux [27].

Initially motivated to help understand 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, see Bournez and Pouly [14], the proof of the Turing-completeness of chemical reactions, see Fages, Le Guludec, Bournez and Pouly [21], or hardness of problems related to dynamical systems, see Graça and Zhong [26].

While the above results are easy to state, their proofs are mixing considerations about approximations, control of errors, and various constructions to emulate continuously some discrete processes, despite some recent attempts for a kind of programming language to help intuition, see Bournez [7].

Discrete ODEs, which we consider in this article, are an approach in-between born from the attempt of Bournez and Durand in [11, 12] 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 [11, 12] of polynomial time. Namely, 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 [20], nor an explicit assignment of a specific role or type to variables, in contrast to safe recursion or ramification, see Bellantoni and Cook in [1] and Leivant in [29]. Instead, they only assume the involved ODEs to be linear, implicitly capturing the role of the ramification. A characterisation, like ours, happens to be rather simple using only common notions from the world of ODEs. In particular, considering *linear* ordinary differential equations is very natural for ODEs.

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 discrete settings. The Leibniz rule is a very basic tool for establishing results over the continuum, using various stability properties, but similar statements cannot be established 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 $[f_1, f_2, \ldots, f_k; op_1, op_2, \ldots, op_\ell]$ denotes the smallest set of functions containing $f_1, f_2, \ldots, f_k$ that is closed under the operations op_1, op_2, $\ldots$ op_ℓ. Call *discrete function* a function of type $f: S_1 \times \cdots \times S_d \to S'_1 \times \ldots S'_{d'}$, 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. We give here general notations and do not the domain and the image of the functions. We will use intersections with sets of functions when we need to specify which class of functions we study. For example, $\mathbf{FPTIME} \cap \mathbb{N}^{\mathbb{N}}$ means the class of *discrete* functions computable in polynomial time.

It is important to observe that the literature considers two possible definitions for **FPSPACE**, according to whether functions with non-polynomial size values are allowed. In our case, we should add "whose outputs remain of polynomial size" to resolve the ambiguity.

Remark 2 Without this assumption, **FPSPACE** would not be closed by composition. This may be considered 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.

A main result of Bournez and Durand [11, 12] is the following ($\mathbb{LDL}$ stands for linear derivation on length):

Theorem 1.1 (Bournez and Durand) *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 1.2 (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–th 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 2.4, corresponds to defining a function from a linear ODE 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 $\mathcal{L}\colon \mathbb{N}^{p+1} \to \mathbb{Z}$ and in particular for the case where $\mathcal{L}(x,\mathbf{y}) = \ell(x)$,

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

is a formal synonym for

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

Remark 3 This concept introduced in Bournez and Durand [11, 12], 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 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 $f\colon S_1 \times \cdots \times S_d \to S'_1 \times \ldots S'_{d'}$, where each S_i, S'_i is either $\mathbb{R}$, $\mathbb{N}$ or $\mathbb{Z}$, with at least one S_i and one S'_j being $\mathbb{R}$. A natural question about the characterisation of **FPTIME** for real functions (in particular functions from $\mathbb{R}^{\mathbb{R}}$) arises, and not only discrete functions: we consider here computability over the reals in its most classical approach, namely computable analysis, see Weihrauch [39].

As a first step, the class

$$\mathbb{LDL}^{\bullet} = \left[\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \times, \overline{\mathrm{cond}}(x), \frac{x}{2}; \textit{composition, linear length ODE}\right]$$

has been considered in [2, 5] by Blanc and Bournez where the authors get some characterisation of **PTIME**, but only for functions from the integers to the reals (ie 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, covering not only time complexity but also space complexity. Consider

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

where $\frac{x}{2}\colon \mathbb{R} \to \mathbb{R}$ is the function dividing by 2 (similarly for $\frac{x}{3}$) and all other basic functions defined exactly as for $\mathbb{LDL}$, but are considered here as functions from the reals to reals.

Remark 4 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 this is done in [2, 5] by Blanc and Bournez, that composition may not be defined in some cases: it is a partial operator. For example, given $f\colon \mathbb{N} \to \mathbb{R}$ and $g\colon \mathbb{R} \to \mathbb{R}$, the composition of g and f is defined as expected, but f cannot be composed with a function such as $h\colon \mathbb{N} \to \mathbb{N}$.

First, we improve Theorem 1.1 by stating **FPTIME** over the integers can be characterised algebraically using linear length ODEs and only analytic functions (ie no need for sign function). Since $\mathbb{LDL}^\circ$ is about functions over the reals, and Theorem 1.1 is about functions over the integers, we need a way to compare these classes. Given a function $\mathbf{f}\colon \mathbb{R}^d \to \mathbb{R}^{d'}$ sending every integer $\mathbf{n} \in \mathbb{N}^d$ to the vicinity of some integer of $\mathbb{N}^d$, say at a 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 1.3 $\mathrm{DP}(\mathbb{LDL}^\circ) = \mathbf{FPTIME} \cap \mathbb{N}^{\mathbb{N}}$.

Second, we improve Blanc and Bournez [2]. Write $\overline{\mathbb{LDL}^\circ}$ for the class obtained by adding some effective limit operation similar to the one considered there. Namely, we introduce the operation *ELim* (standing for Effective Limit):

Definition 1.4 (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 $\mathbf{x} \in \mathbb{R}^d$, $X \in \mathbb{N}$, $\mathbf{x} \in \left[-2^X, 2^X\right]$, $\mathbf{m} \in \mathbb{N}^{d''}$, $n \in \mathbb{N}$, $\|\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^n) - \mathbf{f}(\mathbf{x}, \mathbf{m})\| \leq 2^{-n}$, then $\mathit{ELim}(\tilde{\mathbf{f}})$ is the (clearly uniquely defined) corresponding function $\mathbf{f}\colon \mathbb{R}^d \to \mathbb{R}^{d'}$.

We get a characterisation of functions over the reals (and not only sequences as in Blanc and Bournez [2]) computable in polynomial time.

Theorem 1.5 (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 2.7), we get a class $\mathbb{RLD}^{\circ}$ (this stands for robust linear derivation) with similar statements but for **FPSPACE**.

Theorem 1.6 $\mathrm{DP}(\mathbb{RLD}^{\circ}) = \mathbf{FPSPACE} \cap \mathbb{N}^{\mathbb{N}}$.

Theorem 1.7 (Generic functions over the reals) $\overline{\mathbb{RLD}^{\circ}} \cap \mathbb{R}^{\mathbb{R}} = \mathbf{FPSPACE} \cap \mathbb{R}^{\mathbb{R}}$. *More generally:* $\overline{\mathbb{RLD}^{\circ}} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}} = \mathbf{FPSPACE} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}}$.

There exist previous characterisations of **FPSPACE** without explicit bounds, but for very different models of computation, eg for term rewrite systems in [33] by Marion and Péchoux, and for λ–calculus in [24] by Gaboardi, Marion and Ronchi Della Rocca. As far as we know, this is the first time a characterisation of **FPSPACE** with discrete ODEs is provided for Turing machines.

If we forget the context of discrete ODEs, **FPSPACE** has been characterised in [38] by Thompson but using a bounded recursion scheme, ie requiring some explicit bound in the spirit of Cobham's statement [20]. 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 suitably simulated with analytic discrete ODEs. We believe our constructions could be applied to many other situations where programming with ODEs is needed.

More on related works In addition to the previous state-of-the-art discussions, we comment here on some aspects. Our ways of simulating Turing machines are reminiscent of similar constructions used in other contexts such as neural networks, see Siegelmann and Sontag [37] and Siegelmann [36]. In particular, we use a Cantor-like encoding set $\mathcal{I}$ with inspiration from these references. These references use some particular sigmoid function σ (called sometimes the *saturated linear function*) that values 0 when $x \leq 0$, x for $0 \leq x \leq 1$, 1 for $x \geq 1$. The latter corresponds to $\overline{\mathrm{cond}}(\frac{1}{4} + \frac{1}{2}x)$, for the function considered by Blanc and Bournez in [2, 5] and hence their constructions can be reformulated using the $\overline{\mathrm{cond}}$ function. We completely avoid this, by considering the tanh function, which is more natural in the context of formal neural networks. The models considered by Siegelmann and Sontag in [37, 36] rely on inductions, while our constructions are not inductive neural networks, and second, their models are restricted to live on the compact domain $[0, 1]$, which forbids getting functions from $\mathbb{R} \to \mathbb{R}$, while our settings allow more general functions. Our proofs also require functions taking some integer arguments that would be impossible to consider in their settings (unless at the price of an artificial recoding).

Remark 5 In some sense, our constructions can be seen as operators that map to a family of neural networks in the spirit of these models, instead of considering fixed recurrent neural networks, but also dealing with $\tanh$, and not requiring the saturated linear function.

The question of whether Turing machines can be simulated by recurrent neural networks of finite size, using the tanh activation function (and not the "exact" saturated linear function) is a well-known long-standing open problem. Despite some attempts, such as the one described in the monograph from Siegelmann [36], up to our knowledge, there remain some of the statements not yet formally proved, or at least not fully generally accepted, in the existing proofs. Our statements deal with the tanh activation function in some sense, but we avoid this open question by restricting it to finite space or time computations. By the way, our proofs state this is possible if the space or the time of the machine is bounded, up to some controlled error.

In the context of neural network models, there have been several characterisations of complexity classes over the discrete (see the monograph [36] by Sieglemann about the approach discussed above, but not only), as far as we know, the relation between formal neural networks with classes of computable analysis has never been established before.

If we do not restrict ourselves to neural network-related models, as in all these previous contexts, as far as we know, only a few papers have been devoted to characterisations of complexity and even computability classes in the sense of computable analysis. There have been some attempts using continuous ODEs, for example Bournez, Campagnolo, Graça and Hainry [8], that we already mentioned, or the so-called $\mathbb{R}$–recursive functions, see Bournez and Pouly [15] and Bournez, Gomaa and Hainry [13]. For discrete schemata, we only know Brattka [16] and Ng, Tavana and Yang [34], focusing on computability and not complexity. We also mention the references [22], by Férée, Gomaa and Hoyrup, discussing the complexity of operators in computable analysis. Notice that most of the classical complexity classes of the Blum–Shub–Smale model [6] have been characterised using the implicit complexity approach, see Bournez, Cucker, de Naurois and Marion [10, 9].

Organisation of the article 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 the above theorems.

In this article, when we say that a function $f\colon S_1 \times \cdots \times S_d \to \mathbb{R}^{d'}$ is (respectively, polynomial time or space) computable, it will always be in the sense of computable analysis: see eg Brattka Hertling Weihrauch [17] and Weihrauch [39]. As we did not find a reference where the case of functions mixing integer and real arguments is fully formalised, we proposed a formalisation in [2]. We assume the same formalisation here.

This is the journal version of an article accepted at MFCS'2023 [3].

2 Some concepts related to discrete ODEs

In this section, we recall some concepts and definitions from discrete ODEs, either well-known or established by Bournez and Durand [11, 12] or Blanc and Bournez [2].

We need to talk about tanh–polynomial expression. This is not exactly as in Bournez and Durand [11, 12], but similar. They use similar definitions with the sign function sg and Blanc and Bournez [2] with the piecewise affine function $\overline{\mathrm{cond}}$, which values 1 for $x > \frac{3}{4}$ and 0 for $x < \frac{1}{4}$, instead of tanh.

Definition 2.1 (adapted from Blanc and Bournez [2]) A tanh–polynomial expression $P(x_1, ..., x_h)$ is an expression built-on $+, -, \times$ (often denoted $\cdot$) and tanh functions over a set of variables $V = \{x_1, ..., x_h\}$ and rational constants.

We need to measure the degree, similarly to the classical notion of degree in a polynomial expression, but considering all the subterms within the scope of a tanh function contributes to 0 to the degree.

Definition 2.2 (adapted from Blanc and Bournez [2]) The degree $\deg(x, P)$ of a term of a variable $x \in V$ in a tanh–polynomial expression P is defined inductively as follows:

- Define $\deg(x, q) = 0$, when q is some rational.
- $\deg(x, y) = 1$ if $x = y$, and $\deg(x, y) = 0$ otherwise.
- $\deg(x, P + Q) = \max\{\deg(x, P), \deg(x, Q)\}$.
- $\deg(x, P \times Q) = \deg(x, P) + \deg(x, Q)$.
- $\deg(x, \tanh(P)) = 0$.

A tanh–polynomial expression P is *essentially constant* in x if $\deg(x, P) = 0$.

A vectorial function (respectively a matrix or a vector) is said to be a tanh–polynomial expression if all its coordinates (respectively coefficients) are, and *essentially constant* if all its coefficients are.

Definition 2.3 (Bournez and Durand [11, 12], Blanc and Bournez [2]) A tanh–polynomial expression $\mathbf{g}(\mathbf{f}(x,\mathbf{y}), \mathbf{h}(x,\mathbf{y}), x, \mathbf{y})$ is *essentially linear* in $\mathbf{f}$ 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 tanh–polynomial expressions essentially constant in $\mathbf{f}(x,\mathbf{y})$.

For example,

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

Definition 2.4 (Linear length ODE, Bournez and Durand [11, 12], Blanc and Bournez [2]) A function $\mathbf{f}$ is *linear length ODE* definable from $\mathbf{g}, \mathbf{h}, \mathbf{u}$, with $\mathbf{u}$ essentially linear in $\mathbf{f}(x, y)$ if it corresponds to the solution of

(2–1) $$f(0, \mathbf{y}) = \mathbf{g}(\mathbf{y}) \quad \text{and} \quad \frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \ell} = \mathbf{u}(\mathbf{f}(x,\mathbf{y}), \mathbf{h}(x,\mathbf{y}), x, \mathbf{y}).$$

A fundamental fact is that the derivation with respect to length provides a way to do some change of variables. This means $\mathbf{f}(x,\mathbf{y})$ depends only on the length of its first argument: $\mathbf{f}(x,\mathbf{y}) = \mathbf{f}(2^{\ell(x)}, \mathbf{y})$.

Lemma 2.5 (Bournez and Durand [11, 12]) *Assume that (2–1) 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:*

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

Then (2–2) can be seen as defining a function (with this latter property) by a recurrence of type

(2–3) $$\mathbf{f}(2^0, \mathbf{y}) = \mathbf{g}(\mathbf{y}), \quad \text{and} \quad \mathbf{f}(2^{t+1}, \mathbf{y}) = \overline{\mathbf{u}}(\mathbf{f}(2^t, \mathbf{y}), \mathbf{h}(2^t - 1, \mathbf{y}), 2^t, \mathbf{y})$$

for some $\overline{\mathbf{u}}$ is *essentially linear* in $\mathbf{f}(2^t, \mathbf{y})$. As recurrence (2–2) is basically equivalent to (2–1):

Corollary 2.6 (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* (2–3)*, for some* $\mathbf{g}$ *and* $\mathbf{h}$*, and* $\overline{\mathbf{u}}$*, essentially linear in* $\mathbf{f}(2^t, \mathbf{y})$.

We guess it is easier for our readers to deal with recurrences of the form (2–3) than with ODEs of the form (2–1). Consequently, this is how we will describe many functions from now on, starting with some basic functions, authorising compositions, and the above schemes. For example, $x \mapsto 2^{\ell(x)}$ can easily be defined that way. Consider

$$\begin{cases} 2^0 & = & 1 \\ 2^{t+1} & = & 2 \cdot 2^t = 2^t + 2^t \end{cases}$$

where $t = \ell(x)$. Similarly, we can produce $x \mapsto 2^{p(\ell(x))}$ for any polynomial p. Furthermore,

$$(x_1, \ldots, x_k) \to 2^{\ell(x_1)\ell(x_2)\ldots\ell(x_n)}$$

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 2.7 A bounded function $\mathbf{f}$ is *robustly linear* ODE definable from $\mathbf{g}, \mathbf{h}, \mathbf{u}$, with $\mathbf{u}$ essentially linear in $\mathbf{f}(x, y)$ if

(1) it corresponds to the solution of

$$\text{(2–4)} \qquad \mathbf{f}(0, \mathbf{y}) = \mathbf{g}(\mathbf{y}) \quad \text{and} \quad \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial x} = \mathbf{u}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}),$$

(2) where the schema (2–4) is polynomially numerically stable.

The main issue for space is the need to prove the schema given by Definition 2.7 guarantees $\mathbf{f}$ is in **FPSPACE** when $\mathbf{u}$, $\mathbf{g}$, and $\mathbf{h}$ are. Assuming condition (1) of Definition 2.7 would not be sufficient: the problem is that $\mathbf{f}(x, \mathbf{y})$ may polynomially grow too fast or have a modulus function that would grow too fast. The point is, in Definition 2.7, we assumed $\mathbf{f}$ to be both bounded and satisfying (2), ie 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 write $a =_n b$ for $\|a - b\| \leq 2^{-n}$ for conciseness. (2) means formally there exists some polynomial p such that, for all integer n, for $\epsilon(n) = p(n + \ell(\mathbf{y}))$, considering any solution of

$$\begin{cases} \tilde{\mathbf{y}} & =_{\epsilon(n)} \quad \mathbf{y} \\ \tilde{\mathbf{h}}(x, \tilde{\mathbf{y}}) & =_{\epsilon(n)} \quad \mathbf{h}(x, \tilde{\mathbf{y}}) \\ \tilde{\mathbf{f}}(0, \tilde{\mathbf{y}}) & =_{\epsilon(n)} \quad \mathbf{g}(\mathbf{y}) \\ \frac{\partial \tilde{\mathbf{f}}(x,\tilde{\mathbf{y}})}{\partial x} & =_{\epsilon(n)} \quad \mathbf{u}(\tilde{\mathbf{f}}(x, \tilde{\mathbf{y}}), \tilde{\mathbf{h}}(x, \tilde{\mathbf{y}}), x, \tilde{\mathbf{y}}) \end{cases}$$

then, we have

$$\tilde{\mathbf{f}}(x, \tilde{\mathbf{y}}) =_{\epsilon(n)} \mathbf{f}(x, \mathbf{y}).$$

This corresponds roughly to the concept of polynomially robust to precision considered in [4], and turns out to be a very natural concept for functions defined on a compact.

Remark 6 For linear length ODEs, we did not have to put explicitly numerical stability as a hypothesis, as it comes free from the fact that we consider solutions at most at some logarithmic value of their arguments. But this is required here to guarantee the computability of the solution (and even polynomial space computability).

Remark 7 Notice that, over the continuum, even computable ODEs may have no computable solution: see Pour-El and Richards [35]. Over the discrete, not all dynamics can be simulated, and numerical stability is indeed an issue.

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, ie $\{x\} = x - \lfloor x \rfloor$. These functions will be used to simulate the execution of the Turing machines in the following section. For many of them, we have functions parameterised by some m, providing a way to approximate some "ideal" function: the error between the function and the ideal function remains at less than 2^{-m}. This sometimes holds on a subdomain controlled by some parameter n.

Lemma 3.1 $|1 + \tanh x| \leq 2\exp(2|x|)$ *for* $x \in (-\infty, 0]$.

Proof For $x \leq 0$, $|1 + \tanh x| = 1 + \tanh x$, and $|x| = -x$. We have $f(x) = 2\exp(2x) - \tanh(x) - 1 = 2\exp(2x) - \frac{1-\exp(-2x)}{1+\exp(-2x)} - 1 = \frac{2\exp(4x)}{1+\exp(2x)} \geq 0$. □

Lemma 3.2 $|1 - \tanh x| \leq 2\exp(-2|x|)$ *for* $x \in [0, +\infty)$.

Proof This follows from Lemma 3.1, using the fact that tanh is odd. □

A first observation is that we can uniformly approximate the (very famous in deep learning context) $\mathrm{ReLU}(x) = \max(0, x)$ function using an essentially constant function:

Lemma 3.3 *Consider (see Figure 1):*

$$Y(x, 2^{m+2}) = \frac{1 + \tanh(2^{m+2}x)}{2}$$

For all integer m*, for all* $x \in \mathbb{R}$,

$$|\,\mathrm{ReLU}(x) - xY(x, 2^{m+2})| \leq 2^{-m}.$$

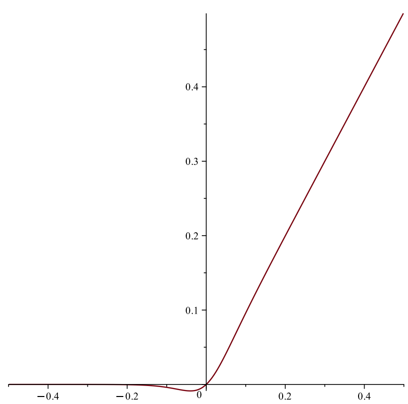


Figure 1: Graphical representation of $xY(x, 2^{2+2})$ obtained with Maple.

To prove Lemma 3.3, we start by the following basic facts about function tanh.

Proof of Lemma 3.3 Let $m \in \mathbb{N}$. Consider $Y(x, K) = \frac{1+\tanh(Kx)}{2}$, with $K > 0$.

For $0 \leq x$, $\mathrm{ReLU}(x) = x$, and $|\,\mathrm{ReLU}(x) - xY(x, K)| = \frac{x}{2}|1 - \tanh(Kx)| \leq x\exp(-2Kx)$ from Lemma 3.2.

For $x \leq 0$, $\mathrm{ReLU}(x) = 0$, and $|\,\mathrm{ReLU}(x) - xY(x, K)| = \frac{|x|}{2}|1 + \tanh(Kx)| \leq |x|\exp(-2K|x|)$ from Lemma 3.1, which is the same expression as above for $0 \leq x$.

Function $g(x) = x\exp(-2Kx)$ has its derivative $g'(x) = \exp(-2Kx)(1-2Kx)$. We deduce the maximum of this function $g(x)$ over $\mathbb{R}$ is in $\frac{1}{2K}$, and that the maximum value of $g(x)$ is $\frac{1}{e2K}$.

Consequently, if we take $K = 2^{m+2}$, then $g(x) \leq 2^{-m}$ for all x, and we conclude. □

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

$$|\,\mathcal{C}\text{–}\mathfrak{s}(m,a,b,x) - \mathfrak{s}(a,b,x)| \leq 2^{-m}.$$

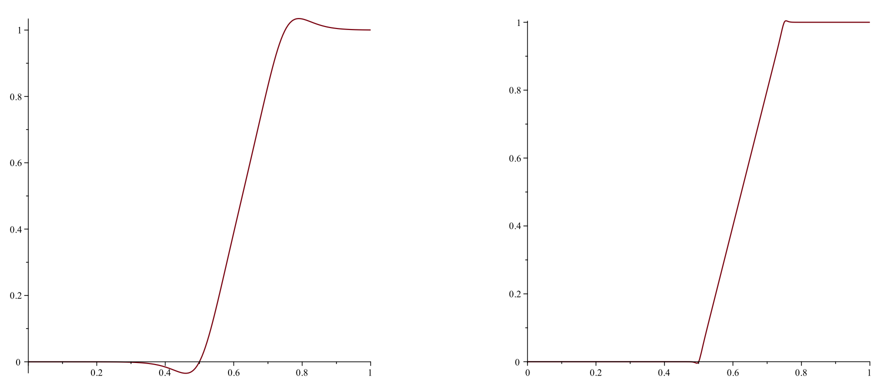


Figure 2: Graphical representations of $\mathcal{C}\text{–}\mathfrak{s}(2, \frac{1}{2}, \frac{3}{4}, x)$ and $\mathcal{C}\text{–}\mathfrak{s}(2^5, \frac{1}{2}, \frac{3}{4}, x)$ obtained with Maple.

Proof We can write $\mathfrak{s}(a,b,x) = \frac{\mathrm{ReLU}(x-a)-\mathrm{ReLU}(x-b)}{b-a}$. Consider $\mathcal{C}\text{–}\mathfrak{s}(z,a,b,x) = \frac{(x-a)Y(x-a,z2^{1+c})-(x-b)Y(x-b,z2^{1+c})}{b-a}$. 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 $\frac{1}{b-a} \leq 2^c$. □

We prove the existence of a continuous approximation of a threshold, written in $\mathbb{LDL}^{\circ}$. We will need it to express continuous approximations of discrete functions, such as the fractional part (Corollary 3.6), the floor function (Corollary 3.7), a function to express an adaptive barycenter (Corollary 3.8), the modulo 2 (Corollary 3), and the Euclidian division by 2 (Corollary 3.10).

Theorem 3.5 *There exists some function (illustrated by Figure 3) $\xi \colon \mathbb{N}^2 \to \mathbb{R}$ in $\mathbb{LDL}^\circ$ such that for all $n, m \in \mathbb{N}$ and $x \in [-2^n, 2^n]$, whenever $x \in [\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}]$,*

$$\left| \xi(2^m, 2^n, x) - \left\{ x - \frac{1}{8} \right\} \right| \le 2^{-m}.$$

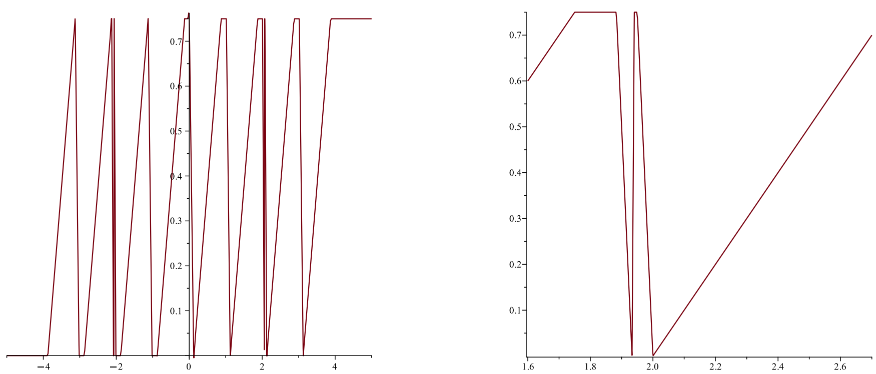


Figure 3: Graphical representations of $\xi(2, 4, x)$ obtained with Maple; some details on the right.

Proof If we take ξ' that satisfies the constraint only when $x \ge 0$, and that values 0 for $x \le 0$, then $\frac{3}{4} - \xi'(\cdot, \cdot, -x)$ would satisfy the constraint when $x \le 0$, but values $3/4$ for $x \ge 0$. So,

$$\xi(2^m, N, x) = \xi'(2^{m+2}, N, x) - \xi'(2^{m+2}, N, -x) + \frac{3}{4} - \frac{3}{4}\, \mathcal{C}\text{–}\mathfrak{s}(2^{m+2}, 0, \frac{1}{8}, x)$$

would work for all x. So it remains to construct ξ' such that for all $n \in \mathbb{N}$, $x \in [0, 2^n]$ and $m \in \mathbb{N}$, whenever $x \in [\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}]$, $|\xi'(2^m, 2^n, x) - \{x - \frac{1}{8}\}| \le 2^{-m}$, and $|\xi'(2^m, N, x) - 0] \le 2^{-m}$ for $x \le 0$.

Let $s(x) = \frac{3}{4}\, \mathfrak{s}(\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}, x)$. Over $[\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}]$, we have $s(x) = \{x - \frac{1}{8}\}$. Actually, we will even construct ξ' with the stronger properties that whenever $x \in [\lfloor x \rfloor + \frac{1}{8} - 2^{-m}, \lfloor x \rfloor + \frac{7}{8} + 2^{-m}]$, $|\xi'(2^m, 2^n, x) - s(x - \frac{1}{8})| \le 2^{-m}$.

It suffices to define ξ' by induction by

$$\begin{cases} \xi'(2^m, 2^0, x) &= \frac{3}{4}\, \mathcal{C}\text{–}\mathfrak{s}(2^m, \frac{1}{8}, \frac{7}{8}, x) \\ \xi'(2^m, 2^{n+1}, x) &= \xi'(2^{m+1}, 2^n, F(2^{m+1}, 2^n, x)) \end{cases}$$

where

$$F(2^{m+1}, K, x) = x - K.\, \mathcal{C}\text{–}\mathfrak{s}(2^{m+1}, K + \frac{1}{32}, K + \frac{3}{32}, x).$$

Let $I_{\lfloor x \rfloor}$ be $[\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}]$, $x \in I_{\lfloor x \rfloor}$, and let us first study the value of $F(2^{m+1}, 2^n, x)$:

- If $x \leq 2^n$, by definition of $\mathcal{C}\text{–}\mathfrak{s}$, $|F(2^{m+1}, 2^n, x) - x| \leq 2^{-(m+1)}$, with $x \in I_{\lfloor x \rfloor}$.
- The case $2^n < x < 2^n + \frac{1}{8}$ cannot happen as we assume $x \in I_{\lfloor x \rfloor}$.
- If $2^n + \frac{1}{8} \leq x$ then $|F(2^{m+1}, 2^n, x) - (x - 2^n)| \leq 2^{-(m+1)}$ with $x - 2^n \in I_{\lfloor x \rfloor - 2^n}$.

Now, the property is true by induction. Indeed, it is true for $n = 0$ by definition of $\xi'(2^m, 2^0, x)$. We now assume it is true for some $n \in \mathbb{N}$. We have $\xi'(2^m, 2^{n+1}, x) = \xi'(2^{m+1}, 2^n, F(2^{m+1}, 2^n, x))$. Thus, by induction hypothesis,

$$|\xi'(2^{m+1}, 2^n, F(2^{m+1}, 2^n, x)) - s(F(2^{m+1}, 2^n, x) - 1/8)| \leq 2^{-(m+1)}.$$

Now:

- If $x \leq 2^n$, by definition of $\mathcal{C}\text{–}\mathfrak{s}$, $|F(2^{m+1}, 2^n, x) - x| \leq 2^{-(m+1)}$, and as s is 1–Lipschitz, $|s(F(2^{m+1}, 2^n, x) - \frac{1}{8}) - s(x - \frac{1}{8})| \leq |F(2^{m+1}, 2^n, x) - x| \leq 2^{-(m+1)}$. Consequently, $|\xi'(2^m, 2^{n+1}, x) - s(x - \frac{1}{8})| \leq 2^{-m}$ and the property holds for $n + 1$.
- The case $2^n < x < 2^n + \frac{1}{8}$ cannot happen with our constraint $x \in I_{\lfloor x \rfloor}$.
- If $2^n + \frac{1}{8} \leq x$ then $|F(2^{m+1}, 2^n, x) - (x - 2^n)| \leq 2^{-(m+1)}$ and as s is 1–Lipschitz, $|s(F(2^{m+1}, 2^n, x) - \frac{1}{8}) - s(x - 2^n - \frac{1}{8})| \leq |F(2^{m+1}, 2^n, x) - x - 2^n| \leq 2^{-(m+1)}$. Consequently, $|\xi'(2^m, 2^{n+1}, x) - s(x - 2^n - \frac{1}{8})| \leq 2^{-m}$ and the property holds for $n + 1$.

There remains to prove that the function ξ' is in $\mathbb{LDL}^{\circ}$. Unfortunately, this is not clear from the recursive definition, but this can be written in another way, from which this follows. Indeed, we have from an easy induction that $\xi'(2^m, 2^n, x) = F(2^{m+n-1}, 2^0, F(2^{m+n-2}, 2^1, F(2^{m+n-3}, 2^2\ (\ldots, F(2^m, 2^{n-1}, x)))))$, if we define:

$$F(2^m, 2^0, x) = \xi'(2^m, 2^0, x) = \frac{3}{4}\,\mathcal{C}\text{–}\mathfrak{s}(2^m, \frac{1}{8}, \frac{7}{8}, x)$$

Then, we can obtain $\xi'(2^m, 2^n, x) = H(2^m, 2^{n-1}, 2^n, x)$ with:

$$\begin{aligned}
H(2^m, 2^0, 2^n, x) &= F(2^m, 2^{n-1}, x) \\
H(2^m, 2^{t+1}, 2^n, x) &= F(2^{m+t}, 2^{n-1-t}, H(2^m, 2^t, 2^n, x)) \\
&= H(2^m, 2^t, 2^n, x) - 2^{n-1-t}.\,\mathcal{C}\text{–}\mathfrak{s}(2^{m+t}, 2^{n-1-t}, 2^{n-1-t} \\
&\quad + \frac{1}{8}, H(2^m, 2^t, 2^n, x))
\end{aligned}$$

Such recurrence can be then seen as a linear length ordinary differential equation in the length of its first argument. It follows that ξ' is in $\mathbb{LDL}^{\circ}$. □

From the construction of the previous functions, we obtain a bestiary of various functions:

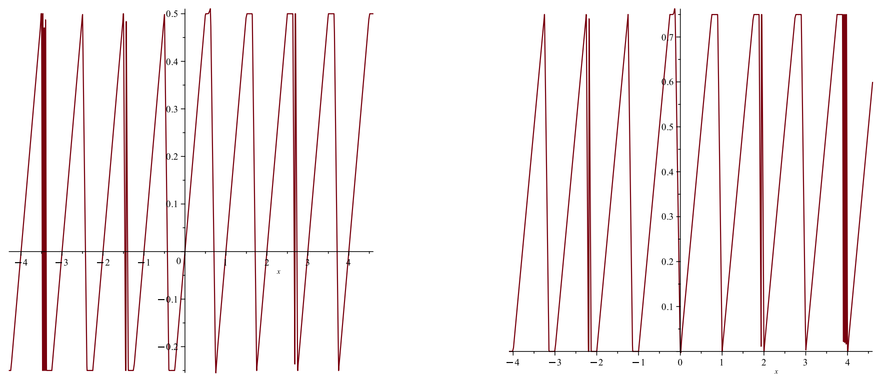


Figure 4: Graphical representation of $\xi_1(2,4,x)$ and $\xi_2(2,4,x)$ obtained with Maple.

Corollary 3.6 *There exists (see Figure 4)* $\xi_1,\xi_2\colon\ \mathbb{N}^2\times\mathbb{R}\mapsto\mathbb{R}\ \in\mathbb{LDL}^\circ$ *such that, for all* $n,m\in\mathbb{N}$, $\lfloor x\rfloor\in[-2^n+1,2^n]$,

- *whenever* $x\in\lfloor x\rfloor-\frac{1}{2},\lfloor x\rfloor+\frac{1}{4}]$, $\quad|\xi_1(2^m,2^n,x)-\{x\}|\leq 2^{-m}$, *and*
- *whenever* $x\in[\lfloor x\rfloor,\lfloor x\rfloor+\frac{3}{4}]$, $\quad|\xi_2(2^m,2^n,x)-\{x\}|\leq 2^{-m}$.

Proof Consider

$$\xi_1(2^m,N,x)=\xi(2^m,N,x-\frac{3}{8})-\frac{1}{2}$$

and

$$\xi_2(2^m,N,x)=\xi(2^m,N,x-\frac{7}{8}).$$

□

Corollary 3.7 *There exists (see Figure 5)* $\sigma_1,\sigma_2\colon\ \mathbb{N}^2\times\mathbb{R}\mapsto\mathbb{R}\ \in\mathbb{LDL}^\circ$ *such that, for all* $n,m\in\mathbb{N}$, $\lfloor x\rfloor\in[-2^n+1,2^n]$,

- *whenever* $x\in[\lfloor x\rfloor-\frac{1}{2},\lfloor x\rfloor+\frac{1}{4}]$, $\quad|\sigma_1(2^m,2^n,x)-\lfloor x\rfloor|\leq 2^{-m}$, *and*
- *whenever* $x\in I_2=[\lfloor x\rfloor,\lfloor x\rfloor+\frac{3}{4}]$, $\quad|\sigma_2(2^m,2^n,x)-\lfloor x\rfloor|\leq 2^{-m}$.

Proof Consider $\sigma_i(2^n,x)=x-\xi_i(2^n,x)$ with the function defined in Corollary 3.6. □

Corollary 3.8 *There exist (see Figure 6)* $\lambda\colon\ \mathbb{N}^2\times\mathbb{R}\mapsto[0,1]\in\mathbb{LDL}^\circ$ *such that for all* $m,n\in\mathbb{N}$, $\lfloor x\rfloor\in[-2^n+1,2^n]$,

- *whenever* $x\in[\lfloor x\rfloor+\frac{1}{4},\lfloor x\rfloor+\frac{1}{2}]$, $\quad|\lambda(2^m,2^n,x)-0|\ \leq 2^{-m}$, *and*
- *whenever* $x\in[\lfloor x\rfloor+\frac{3}{4},\lfloor x\rfloor+1]$, $\quad|\lambda(2^m,2^n,x)-1|\leq 2^{-m}$.

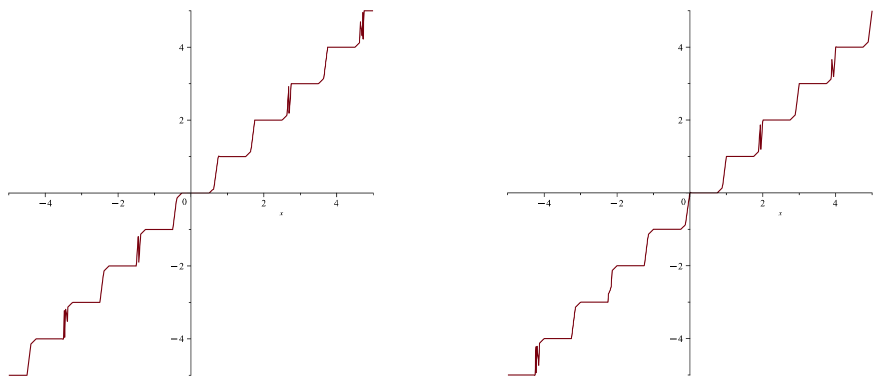


Figure 5: Graphical representation of $\sigma_1(2,4,x)$ and $\sigma_2(2,4,x)$ obtained with Maple.

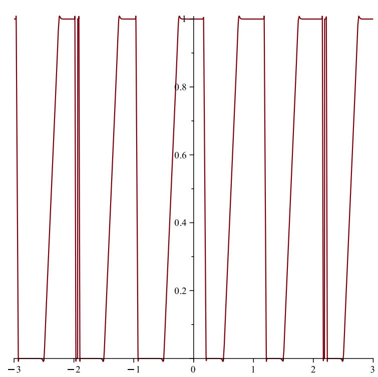


Figure 6: Graphical representation of $\lambda(2,4,x)$ obtained with Maple.

Proof Consider

$$\lambda(2^m, 2^n, x) = F(\xi(2^{m+1}, 2^n, x - 9/8))$$

where

$$F(x) = \mathcal{C}\text{–}\mathfrak{s}(2^{m+1}, 1/4, 1/2, x).$$

□

Corollary 3.9 *There exists (see Figure 7)* $\mathrm{mod}_2\colon\ \mathbb{N}^2 \times \mathbb{R} \mapsto [0,1] \in \mathbb{LDL}^\circ$ *such that for all* $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n+1, 2^n]$, *whenever* $x \in [\lfloor x \rfloor - \frac{1}{4}, \lfloor x \rfloor + \frac{1}{4}]$,

$$|\,\mathrm{mod}_2(2^m, 2^n, x) - \lfloor x \rfloor \bmod 2| \leq 2^{-m}.$$

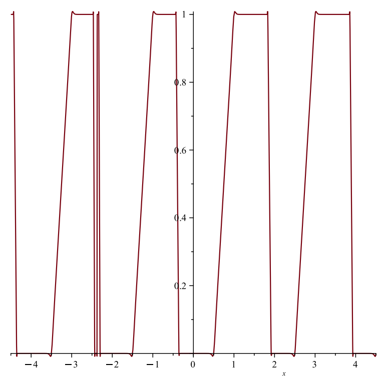


Figure 7: Graphical representation of $\mathrm{mod}_2(2,4,x)$ obtained with Maple.

Proof We can take

$$\mathrm{mod}_2(2^m, N, x) = 1 - \lambda(2^m, N/2, \frac{1}{2}x + \frac{7}{8})$$

where λ is the function given by Corollary 3.8. □

Corollary 3.10 *There exists (see Figure 8)* $\div_2\colon \mathbb{N}^2 \times \mathbb{R} \mapsto [0,1] \in \mathbb{LDL}^\circ$ *such that for all* $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n+1, 2^n]$, *whenever* $x \in [\lfloor x \rfloor - \frac{1}{4}, \lfloor x \rfloor + \frac{1}{4}]$,

$$|\div_2(2^m, 2^n, x) - \lfloor x \rfloor // 2| \leq 2^{-m}$$

where $//$ *is the integer division.*

Proof We can take

$$\div 2(2^m, N, x) = \frac{1}{2}(\sigma_1(2^m, N, x) - \mathrm{mod}_2(2^m, N, x))$$

where mod_2 is the function given by Corollary 3.9, and σ_2 is the function given by Corollary 3.7. □

3.1 Some properties of sigmoids

In this section, we encode a conditional function in $\mathbb{LDL}^\circ$. We observe that, for the function

$$\overline{\mathrm{if}}(d, \ell) = 4\,\mathfrak{s}(1, 2, 1/2 + d + \ell/4) - 2$$

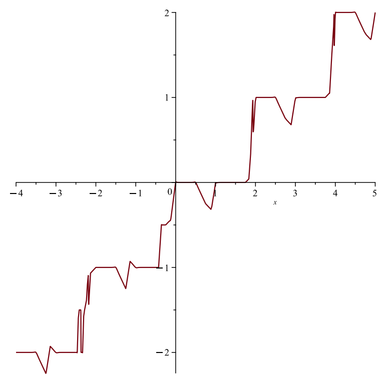


Figure 8: Graphical representation of $\div_2(2,4,x)$ obtained with Maple.

with $\ell \in [0,1]$ we have:

$$\begin{array}{rcl}\overline{\text{if}}(0,\ell) & = & 0\\ \overline{\text{if}}(1,\ell) & = & \ell\end{array}$$

Applying Lemma 3.4 on this sigmoid, we get:

Lemma 3.11 *There exists* $\mathcal{C}$–if $\in \mathbb{LDL}^\circ$ *such that, for* $\ell \in [0,1]$,

- *if we take* $|d'-0| \leq 1/4$, *then* $|\mathcal{C}\text{–if}(d',\ell)-0| \leq 2^{-m}$, *and*
- *if we take* $|d'-1| \leq 1/4$, *then* $|\mathcal{C}\text{–if}(d',\ell)-\ell| \leq 2^{-m}$.

We start by proving the following lemma, which is about some ideal sigmoids:

Lemma 3.12 *For* $\ell \in [0,1]$, *if* $d \in \left[-\frac{1}{4},\frac{1}{4}\right]$, *then* $\overline{\text{if}}(d,\ell)=0$, *and if* $d \in \left[\frac{3}{4},\frac{5}{4}\right]$, *then* $\overline{\text{if}}(d,\ell)=4(d-1)+\ell$.

Consider $\text{if}(d,\ell)=\overline{\text{if}}(\mathfrak{s}(1/4,3/4,x),\ell)$.

- *If we take* $|d'-0| \leq 1/4$, *then* $\text{if}(d',\ell)=0$, *and*
- *if we take* $|d'-1| \leq 1/4$, *then* $\text{if}(d',\ell)=\ell$.

Proof Just check that for $d \leq 1/4$, we have $1/2+d+\ell/4 \leq 1$, and hence $\mathfrak{s}(1,2,1/2+d+\ell/4)=0$, so $\overline{\text{if}}(d,\ell)=0$. For $3/4 \leq d \leq 5/4$, we have $5/4 \leq 1/2+d+\ell/4 \leq 2$, and hence $\mathfrak{s}(1,2,1/2+d+\ell/4)=d+\ell/4-1/2$, and hence $\overline{\text{if}}(1,\ell)=\ell$. The other observation follows. □

Then we go to versions using tanh:

Lemma 3.13 *Consider* $\overline{\mathcal{C}\text{–if}}(2^m, d, l) = 4\,\mathcal{C}\text{–}\mathfrak{s}(2^{m+2}, 1, 2, 1/2 + d + l/4) - 2$. *For* $\ell \in [0, 1]$, *we have* $|\overline{\mathcal{C}\text{–if}}(0, \ell) - 0| \leq 2^{-m}$, *and* $|\overline{\mathcal{C}\text{–if}}(1, \ell) - \ell| \leq 2^{-m}$.

If $d \in \left[-\frac{1}{4}, \frac{1}{4}\right]$, *then* $|\overline{\mathcal{C}\text{–if}}(2^m, d, \ell) - 0| \leq 2^{-m}$, *and if* $d \in \left[\frac{3}{4}, \frac{5}{4}\right]$, *then* $|\overline{\mathcal{C}\text{–if}}(2^m, d, \ell)$ $-4(d-1) + \ell| \leq 2^{-m}$.

Consider $\mathcal{C}\text{–if}(2^m, d, \ell) = \overline{\mathcal{C}\text{–if}}(2^{m+1}, \mathcal{C}\text{–}\mathfrak{s}(2^{m+1}, 1/4, 3/4, x), \ell)$.

- *If we take* $|d' - 0| \leq 1/4$, *then* $|\mathcal{C}\text{–if}(d', \ell) - 0| \leq 2^{-m}$, *and*
- *if we take* $|d' - 1| \leq 1/4$, *then* $|\mathcal{C}\text{–if}(d', \ell) - \ell| \leq 2^{-m}$.

Proof This is direct from the previous lemma and Lemma 3.4. □

Then Lemma 3.11 follows.

Lemma 3.14 *Let* $\alpha_1, \alpha_2, \ldots, \alpha_n$ *be some pairwise distinct integers, and* $V_1, V_2, \ldots, V_n$ *some constants. There is some function in* $\mathbb{LDL}^{\circ}$, *that we write*

$$\mathcal{C}\text{–send}(2^m, \alpha_i \mapsto V_i)_{i \in \{1,\ldots,n\}}$$

that maps any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ *to a real at a distance at most* 2^{-m} *of* V_i, *for all* $i \in \{1, \ldots, n\}$.

We prove the following lemma about some ideal sigmoids: define $\overline{\text{cond}}(x)$ as $\mathfrak{s}(1/4, 3/4, x)$ and $\overline{\mathcal{C}\text{–cond}}(2^m, x)$ as $\mathcal{C}\text{–}\mathfrak{s}(2^m, 1/4, 3/4, x)$.

Lemma 3.15 *Assume you are given some pairwise distinct integers* $\alpha_1, \alpha_2, \ldots,$ α_n, *and some constants* $V_1, V_2, \ldots, V_n$. *Then there is some function in* $\mathbb{LDL}^{\circ}$, *written* $\text{send}(\alpha_i \mapsto V_i)_{i \in \{1,\ldots,n\}}$, *that maps any* $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ *to* V_i, *for all* $i \in \{1, \ldots, n\}$.

Proof Sort the α_i so that $\alpha_1 < \alpha_2 < \ldots, \alpha_n$. Then consider $T_1 + \overline{\text{cond}}(x - \alpha_1)(T_2 - T_1) + \overline{\text{cond}}(x - \alpha_2)(T_3 - E_3) + \cdots + \overline{\text{cond}}(x - \alpha_{n-1})(T_n - T_{n-1})$. □

We can now go to versions with tanh.

Proof of Lemma 3.14 Sort the α_i so that $\alpha_1 < \alpha_2 < \ldots, \alpha_n$. Then consider $T_1 + \overline{\mathcal{C}\text{–cond}}(2^{m+c}, x - \alpha_1)(T_2 - T_1) + \overline{\mathcal{C}\text{–cond}}(2^{m+c}, x - \alpha_2)(T_3 - T_2) + \cdots + \overline{\mathcal{C}\text{–cond}}(2^{m+c}, x - \alpha_{n-1})(T_n - T_{n-1})$. for some constant c so that $n \max_j (T_j - T_{j+1}) \leq 2^c$. □

More generally:

Lemma 3.16 *Let $N \in \mathbb{N}$, $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{N}^n$, pairwise distinct, and, for $1 \leq i \leq n$ and $1 \leq j \leq N$, let $V_{i,j}$ be some constants. Then there exists some function*

$$\mathcal{C}\text{–send}(2^m, (\alpha_i, j) \mapsto V_{i,j})_{i \in \{1,\ldots,n\}, j \in \{1,\ldots,N\}}$$

in $\mathbb{LDL}^\circ$, such as, for all $i \in \{1,\ldots,n\}$, $j \in \{1,\ldots,N\}$, it maps any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ and any $y \in [j - 1/4, j + 1/4]$ to a real number at a distance at most 2^{-m} of $V_{i,j}$.

We start by proving the following lemma talking about some ideal sigmoids:

Lemma 3.17 *Let N be some integer. Assume some pairwise distinct integers $\alpha_1, \alpha_2, \ldots, \alpha_n$, and some constants $V_{i,j}$ for $1 \leq i \leq n$, and $0 \leq j < N$ are given. Then there is some function in $\mathbb{LDL}^\circ$, that we write $\mathrm{send}((\alpha_i, j) \mapsto V_{i,j})_{i \in \{1,\ldots,n\}, j \in \{0,\ldots,N-1\}}$, that maps any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ and $y \in [j - 1/4, j + 1/4]$ to $V_{i,j}$, for all $i \in \{1,\ldots,n\}$, $j \in \{0,\ldots,N-1\}$.*

Proof If we define the function

$$\mathrm{send}((\alpha_i, j) \mapsto V_{i,j})_{i \in \{1,\ldots,n\}, j \in \{1,\ldots,N\}}(x, y)$$

by $\mathrm{send}(N\alpha_i + j \mapsto V_{i,j})_{i \in \{1,\ldots,n\}, j \in \{1,\ldots,N\}}(Nx + y)$ this works when $x = \alpha_i$ for some i. Considering instead $\mathrm{send}(N\alpha_i + j \mapsto V_{i,j})_{i \in \{1,\ldots,n\}, j \in \{1,\ldots,N\}}$ $(N\,\mathrm{send}(\alpha_i \mapsto \alpha_i)_{i \in \{1,\ldots,n\}}(x) + y)$ works for any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$. □

We then go to versions with tanh:

Proof of Lemma 3.16 If we define the function

$$\mathcal{C}\text{–send}(2^m, (\alpha_i, j) \mapsto V_{i,j})_{i \in \{1,\ldots,n\}, j \in \{1,\ldots,N\}}(x, y)$$

by $\mathcal{C}\text{–send}(2^m, N\alpha_i + j \mapsto V_{i,j})_{i \in \{1,\ldots,n\}, j \in \{1,\ldots,N\}}(Nx + y)$ this works when $x = \alpha_i$ for some i. Considering instead

$$\mathcal{C}\text{–send}(2^m, N\alpha_i + j \mapsto V_{i,j})_{i \in \{1,\ldots,n\}, j \in \{1,\ldots,N\}}(N\,\mathcal{C}\text{–send}(2^{m+c}, \alpha_i \mapsto \alpha_i)_{i \in \{1,\ldots,n\}}(x) + y)$$

that works for any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$, for some constant c selected as above. □

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 [2] by Blanc and Bournez 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 0 is the blank symbol.

Remark 8 The reason for the choice of symbols 1 and 3 is that it enables us to have additional properties on the numbers written on the tape of the Turing machine. We explained it more accurately in Remark 9.

We assume $Q = \{0, 1, \dots, |Q| - 1\}$. Let $\dots l_{-k}l_{-k+1} \dots l_{-1}l_0r_0r_1 \dots r_n \dots$ 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: $\gamma_{word}\colon \Sigma^\omega \to \mathbb{R}$ for the function that maps a word $w = w_0w_1w_2 \dots$ to the dyadic $\gamma_{word}(w) = \sum_{n\geq 0} w_n 4^{-(n+1)}$.

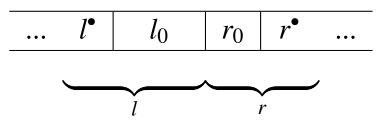
The idea is that such a configuration C can also be encoded by some element $\overline{C} = (q, \bar{l}, \bar{r}) \in \mathbb{N} \times \mathbb{R}^2$, by considering $\bar{r} = \gamma_{word}(r)$ and $\bar{l} = \gamma_{word}(l)$. In other words, we encode the configuration of a bi-infinite tape Turing 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 $\gamma_{word}\colon \Sigma^\omega \to \mathbb{R}$ by $\mathcal{I}$, this even lives in $Q \times \mathcal{I}^2$. 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 (ie of the form $n/4^S$ for some integer n).

The key point is to observe that we can write a function in $\mathbb{LDL}^\circ$ simulating a one-step reduction of the TM M.

Lemma 4.1 *Given a TM M, there exists some function $\overline{Next}$ in $\mathbb{LDL}^\circ$ simulating one step of M: for any configuration C of M, writing C' the configuration after one transition in M, for all $m \in \mathbb{N}$, $\|\overline{Next}(2^m, \overline{C}) - \overline{C}'\| \leq 2^{-m}$.*

Remark 9 Our problem with such formalism is that the involved expressions are sometimes discontinuous functions such as the floor function ($x \in \mathbb{R} \mapsto \lfloor x \rfloor = n \in \mathbb{N}$ such that $n \leq x < n + 1$) and the fractional part ($x \in \mathbb{R} \mapsto x - \lfloor x \rfloor$) functions, and we would rather have analytic (hence continuous) functions. However, 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 $[3,4]$.

Proof We can write $l = l_0 l^\bullet$ and $r = r_0 r^\bullet$, respectively the left part of the tape with respect to the head and the right part of the tape, 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 $l_{-1} l_{-2} \ldots$ and $r_1 r_2 \ldots$ respectively. Notice that l is written in reverse order on the tape.



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:

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

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

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

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

We introduce the following functions:

$$\begin{aligned} \rightarrow : Q \times \{0,1,3\} &\mapsto \{0,1\} \\ (q,a) &\mapsto 1 \quad \text{if } \delta(q,a) = (\cdot, \cdot, \rightarrow) \\ (q,a) &\mapsto 0 \quad \text{otherwise} \end{aligned}$$

correponding to the head moving to the right, and

$$\begin{aligned} \leftarrow : Q \times \{0,1,3\} &\mapsto \{0,1\} \\ (q,a) &\mapsto 1 \quad \text{if } \delta(q,a) = (\cdot, \cdot, \leftarrow) \\ (q,a) &\mapsto 0 \quad \text{otherwise} \end{aligned}$$

corresponding to the head moving to the left.

We also define $nextq^q_a = q'$ if $\delta(q,a) = (q',\cdot,\cdot)$, ie 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

$$\bar{l'} = \sum_{q,r_0} \left[\rightarrow (q,r_0) \left(\frac{\bar{l}}{4} + \frac{x}{4} \right) + \leftarrow (q,r_0) \left\{ 4\bar{l} \right\} \right]$$

and

$$\bar{r'} = \sum_{q,r_0} \left[\rightarrow (q,r_0) \left\{ 4\bar{r} \right\} + \leftarrow (q,r_0) \left(\frac{\{4r\}}{4^2} + \frac{x}{4^2} + \frac{\lfloor 4\bar{l} \rfloor}{4} \right) \right]$$

and, using notation of Lemma 3.16, $q' = \text{send}((q,r) \mapsto nextq^q_r)_{q \in Q, r \in \{0,1,3\}}(q, \lfloor 4\bar{r} \rfloor)$.

Then, following the intuition of Remark 9, we can 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 (eg 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 $\mathcal{C}$–$\mathfrak{s}(2^{m+c},\cdot,\cdot)$ 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 $(3+1/4^2)3|Q|2^{-c}2^{-m} \leq 2^{-m}$.

We can also replace every $\rightarrow (q,r)$ in above expressions for $\bar{l}'$ and $\bar{r}'$ by

$$\mathcal{C}\text{–send}(k, (q,r) \mapsto \rightarrow (q,r))(q, \sigma(4\bar{r}))$$

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 3.11: we can also write the above expressions as:

$$\overline{l'} = \sum_{q,r} \Big[\mathcal{C}\text{–if}\left(2^{m+c}, \mathcal{C}\text{–send}(2^2, (q,r) \mapsto \rightarrow (q,r))(q, \sigma(4\bar{r})), \frac{\bar{l}}{4} + \frac{x}{4}\right) + \mathcal{C}\text{–if}\left(2^{m+c}, \mathcal{C}\text{–send}(2^2, (q,r) \mapsto \leftarrow (q,r))(q, \sigma(4\bar{r})), \xi(4\bar{l})\right) \Big]$$

$$\overline{r'} = \sum_{q,r} \Big[\mathcal{C}\text{–if}\left(2^{m+c}, \mathcal{C}\text{–send}(2^2, (q,r) \mapsto \rightarrow (q,r))(q, \sigma(4\bar{r})), \xi(4\bar{r})\right) + \mathcal{C}\text{–if}\left(2^{m+c}, \mathcal{C}\text{–send}(2^2, (q,r) \mapsto \leftarrow (q,r))(q, \sigma(4\bar{r})), \frac{\xi(4r)}{4^2} + \frac{x}{4^2} + \frac{\sigma(4\bar{l})}{4}\right) \Big]$$

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 with the 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 the function *Next* again, 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 a 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, it will be amplified very soon, up to a level where we have no true idea or control about what becomes the value of the 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 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 a 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 the 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 3.7. We could also replace m by $m + 2S + 4$ to guarantee that $2^{-m} \leq 4^{-(S+2)}$. We get the following major improvement of the previous lemma:

Lemma 4.2 *For any TM M, there exists a function $\overline{Next} \in \mathbb{LDL}^\circ$ simulating one step of M, ie computing the $Next$ function sending any configuration $\overline{C}$ of M to the next configuration $\overline{C}'$, such as $\|Next(2^m, 2^S, \overline{C}) - \overline{C}'\| \leq 2^{-m}$.*

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.

We can deduce the following proposition, adding a time complexity constraint:

Proposition 4.3 *Consider some Turing machine M computing some function $f\colon \Sigma^* \to \Sigma^*$ in some time $T(\ell(\omega))$ on input ω. Then, there exists some function $\tilde{f}\colon \mathbb{N}^2 \times \mathbb{R} \to \mathbb{R}$ in $\mathbb{LDL}^\circ$ such that $\|\tilde{f}(2^m, 2^{T(\ell(\omega))}, \gamma_{word}(\omega)) - \gamma_{word}(f(\omega))\| \leq 2^{-m}$.*

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 lemmas by:

$$\begin{cases} \overline{Exec}(2^m, 0, 2^T, C) & = & C \\ \overline{Exec}(2^m, 2^{t+1}, 2^T, C) & = & \overline{Next}(2^m, 2^T, \overline{Exec}(2^m, 2^t, 2^T, C)) \end{cases}$$

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. □

To get **FPSPACE**, observe that we can also replace the linear length ODE with a linear ODE:

Proposition 4.4 *Consider some Turing machine M computing some function $f\colon \Sigma^* \to \Sigma^*$ in space $S(\ell(\omega))$, for some polynomial S, on input ω. There exists some function $\tilde{f} : \mathbb{N}^2 \times \mathbb{R} \to \mathbb{R}$ in $\mathbb{RLD}^\circ$ such that $\|\tilde{f}(2^m, 2^{S(\ell(\omega))}, \gamma_{word}(\omega)) - \gamma_{word}(f(\omega)) \leq 2^{-m}\|$.*

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 the previous lemma by:

$$\begin{cases} \overline{Exec}(2^m, 0, 2^S, C) & = & C \\ \overline{Exec}(2^m, t+1, 2^S, C) & = & \overline{Next}(2^m, 2^S, \overline{Exec}(2^m, t, 2^S, C)) \end{cases}$$

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.

We 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 $\overline{d} \in \mathcal{I} \cap [0, 1]$:

- If it is before the "decimal" point in d, digit 0 is encoded by 11 and digit 1 by 13.
- If it is after the "decimal" point in d, digit 0 is encoded by 31 and digit 1 by 33.

For example, for $d = 101.1$ in base 2, $\overline{d} = 0.13111333$ in base 4.

Lemma 5.1 (From $\mathbb{N}$ to $\mathcal{I}$) *There exists some function Decode* : $\mathbb{N}^2 \to \mathbb{R}$ *in* $\mathbb{LDL}^\circ$, *mapping* m *and* n *to some point* $\eta_{m,n}$ *such that* $\|\gamma_{word}(\overline{n}) - \eta_{m,n}\| \leq 2^{-m}$.

Proof Recall the functions provided by Corollaries 3.9 and 3.10. The idea is to iterate $\ell(n)$ 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 projects on the second argument.

This can be done in $\mathbb{LDL}^\circ$ by considering

$$\begin{aligned} F(2^m, 2^M, \overline{r_1}, \overline{l_2}) = \mathcal{C}\text{–send}(2^{m+1}, 0 \mapsto (\div_2 (2^{m+1}, 2^M, \overline{r_1}), (\overline{l_2} + 5)/4), \\ 1 \mapsto (\div_2(2^{m+1}, 2^M, \overline{r_1}), (\overline{l_2} + 7)/4))(\overline{r_1}) \end{aligned}$$

and then we define

$$Decode(2^m, n) = \pi_2^2(G(2^{m+\ell(n)}, 2^{\ell(n)+1}, 2^{\ell(n)}, n, 0))$$

with

$$\begin{cases} G(2^m, 2^M, 2^l, 2^0, n, 0) &= (n, 0) \\ G(2^m, 2^M, 2^{l+1}, r, l) &= F(2^m, 2^M, G(2^m, 2^l, r, l)). \end{cases}$$

The global error will be at most $2^{-m-\ell(n)} \times \ell(n) \leq 2^{-m}$. □

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. 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 5.2 (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.

Proof The idea is to do as in the proof of previous lemma, but considering

$$F(\overline{r_1}, \overline{l_2}, \lambda) = \begin{cases} (\sigma(16\overline{r_1}), 2\overline{l_2} + 0, \lambda) & \text{whenever } i(16\overline{r_1}) = 5 \\ (\sigma(16\overline{r_1}), 2\overline{l_2} + \lambda, \lambda) & \text{whenever } i(16\overline{r_1}) = 7 \\ (\sigma(16\overline{r_1}), (\overline{l_2} + 0)/2, \lambda) & \text{whenever } i(16\overline{r_1}) = 13 \\ (\sigma(16\overline{r_1}), (\overline{l_2} + \lambda)/2, \lambda) & \text{whenever } i(16\overline{r_1}) = 15 \end{cases}$$

iterated S times over suitable approximation of the rounding $\mathrm{round}_{S+1}(\gamma_{word}(\overline{d}), 0, \lambda))$, with σ and ξ constructed as an approximation of the integer and fractional part, as before. □

6 Proofs and applications

6.1 Some statements from [2]

We repeat here some statements from Blanc and Bournez [2] . They are motivated by repeating the arguments to prove Proposition 6.4, 1.

On the complexity of solving a linear length ODE We have to prove that all functions of $\mathbb{LDL}^\bullet$ are computable (in the sense of computable analysis) in polynomial time. The hardest part is to prove that the class of polynomial time computable functions is preserved by the linear length ODE schema, so we start with it.

Lemma 6.1 *The class of polynomial time computable functions is preserved by the linear length ODE schema.*

We write $|||\cdots|||$ for the sup norm of the floor function: given some matrix $\mathbf{A} = (A_{i,j})_{1\leq i\leq n, 1\leq j\leq m}$, $|||\mathbf{A}||| = \max_{i,j}\lceil A_{i,j}\rceil$. In particular, given a vector $\mathbf{x}$, it can be seen as a matrix with $m = 1$, and $|||\mathbf{x}|||$ is the sup norm of the floor of its components.

Before proving this lemma, we prove the following result:

Lemma 6.2 (Fundamental observation) *Consider the ODE*

$$\mathbf{F}'(x, \mathbf{y}) = \mathbf{A}(\mathbf{F}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \cdot \mathbf{F}(x, \mathbf{y}) + \mathbf{B}(\mathbf{F}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}). \tag{6–1}$$

Assume:

- *The initial condition* $\mathbf{G}(\mathbf{y}) =^{def} \mathbf{F}(0, \mathbf{y})$ *is polynomial time computable, and* $\mathbf{h}(x, \mathbf{y})$ *are polynomial time computable with respect to the value of* x*.*
- $\mathbf{A}(\mathbf{F}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})$ *and* $\mathbf{B}(\mathbf{F}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})$ *are* $\tanh$*–polynomial expressions essentially constant in* $\mathbf{F}(x, \mathbf{y})$*.*

Then, there exists a polynomial p *such that* $\ell(|||\mathbf{F}(x, \mathbf{y})|||) \leq p\left(x, \ell(|||\mathbf{y}|||)\right)$ *and* $\mathbf{F}(x, \mathbf{y})$ *is polynomial time computable with respect to the value of* x*.*

Proof The solution of ordinary differential equation (6–1) can be put in some explicit form (this follows from [11, 12]):

$$\mathbf{F}(x, \mathbf{y}) = \sum_{u=-1}^{x-1} \left(\prod_{t=u+1}^{x-1} (1 + \mathbf{A}(\mathbf{F}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y})) \right) \cdot \mathbf{B}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y}) \tag{6–2}$$

with the conventions that $\prod_{x}^{x-1} \kappa(x) = 1$ and $\mathbf{B}(\cdot, -1, \mathbf{y}) = \mathbf{G}(\mathbf{y})$.

This formula permits to evaluate $\mathbf{F}(x, \mathbf{y})$, using a dynamic programming approach, from the quantities $\mathbf{y}$, u, $\mathbf{h}(u, \mathbf{y})$, for $1 \leq u < x$, in several arithmetic steps that is polynomial in x. Indeed: for any $-1 \leq u \leq x$, $\mathbf{A}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y})$ and $\mathbf{B}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y})$ are matrices whose coefficients are $\tanh$–polynomial. Their coefficients involve finitely many arithmetic operations or $\tanh$ operations from their inputs. Once this is done, computing $\mathbf{F}(x, \mathbf{y})$ requires polynomially in x many arithmetic operations: once the values for $\mathbf{A}$ and $\mathbf{B}$ are known, we have to sum up $x + 1$ terms, each of them involving at most $x - 1$ multiplications.

We need to take care of not only the arithmetic complexity, which is polynomial in x, but also the bit complexity. We start by discussing the bit complexity of the integer parts. Each of the quantities $\mathbf{y}$, u, $\mathbf{h}(u, \mathbf{y})$, for $1 \leq u < x$, are computable in polynomial time, so their integer parts have a bit complexity remaining polynomial in x and $\ell(|||y|||)$. The bit complexity of a sum, product, etc is polynomial in the size of its arguments, it is sufficient to show that the growth rate of the function $\mathbf{F}(x, \mathbf{y})$ can be polynomially dominated. For this, recall that, for any $-1 \leq u \leq x$, coefficients of $\mathbf{A}(\mathbf{F}(u,\mathbf{y}),\mathbf{h}(u,\mathbf{y}),u,\mathbf{y})$ and $\mathbf{B}(\mathbf{F}(u,\mathbf{y}),\mathbf{h}(u,\mathbf{y}),u,\mathbf{y})$ are essentially constant in $\mathbf{F}(u,\mathbf{y})$. Hence, the size of the integer part of these coefficients does not depend on $\ell(\mathbf{F}(u,\mathbf{y}))$. Since, in addition, $\mathbf{h}$ is computable in polynomial time in x and $\ell(\mathbf{y})$, there exists a polynomial p_M such that:

$$\max(\ell(|||\mathbf{A}(\mathbf{F}(u,\mathbf{y}),\mathbf{h}(u,\mathbf{y}),u,\mathbf{y})|||),\ell(|||\mathbf{B}(\mathbf{F}(u,\mathbf{y}),\mathbf{h}(u,\mathbf{y}),u,\mathbf{y})||| \\ \leq p_M(u,\ell(|||\mathbf{y}|||))) \tag{6–3}$$

It then holds that:

$$\ell(|||\mathbf{F}(x+1,\mathbf{y})|||) \leq p_M(x,\ell(|||\mathbf{y}|||)) + \ell(|||\mathbf{F}(x,\mathbf{y})|||) + 2$$

It follows from an easy induction that we must have

$$\ell(|||\mathbf{F}(x,\mathbf{y})|||) \leq \ell(|||G(\mathbf{y})|||) + x \cdot (p_M(x,\ell(|||\mathbf{y}|||) + 2)$$

which gives the desired bound on the length of the integer part of the values for function $\mathbf{F}$, after observing that, since $\mathbf{G}$ is polynomial time computable, necessarily $\ell(|||G(\mathbf{y})|||)$ remains polynomial in $\ell(|||\mathbf{y}|||)$.

We now take care of the bit complexity of involved quantities to prove that $\mathbf{F}(x,\mathbf{y})$ is indeed polynomial time computable with respect to the value of x. Given $\mathbf{y}$, we can determine some integer Y such that $\mathbf{y} \in [2^{-Y}, 2^{Y}]$. We just need to prove that given n, we can provide some dyadic $\mathbf{z}_x$ approximating $\mathbf{F}(x,\mathbf{y})$ at precision n, ie with $\|\mathbf{F}(x,\mathbf{y}) - \mathbf{z}_x\| \leq 2^{-n}$, in a time polynomial in x and Y.

Basically, this follows from the possibility of evaluating $\mathbf{F}(x,\mathbf{y})$ using formula (6–2). This latter formula is made of a sum of $x+1$ terms, that we can call $\mathbf{T}_0,\mathbf{T}_1,\ldots,\mathbf{T}_x$, each of them $\mathbf{T}_i$ corresponding to a product of k matrices (or vectors) $\mathbf{T}_i = \mathbf{C}_1\mathbf{C}_2\ldots\mathbf{C}_{k(i)}$, with $k(i) \leq x+1$, were each $\mathbf{C}_j$ is either some $\mathbf{B}(\mathbf{F}(u,\mathbf{y}),\mathbf{h}(u,\mathbf{y}),u,\mathbf{y})$ for some u or some $(1+\mathbf{A}(\mathbf{F}(t,\mathbf{y}),\mathbf{h}(t,\mathbf{y}),t,\mathbf{y}))$ for some t.

To solve our problem, it is sufficient to be able to approximate the value of each $\mathbf{T}_i$ by some dyadic $\mathbf{d}_i$ with precision 2^{-n-m}, considering m with $x+1 \leq 2^m$. Indeed, taking $\mathbf{z}_x = \sum_{i=0}^{x} \mathbf{d}_i$ will guarantee an error on the approximation of $\mathbf{F}(x,\mathbf{y})$ less than $(x+1)2^{-n-m} \leq 2^{-n}$.

So we focus on the problem of estimating $\mathbf{T}_i = \mathbf{C}_1\mathbf{C}_2\ldots\mathbf{C}_{k(i)}$ with precision 2^{-n-m}.

If we write $\tilde{\mathbf{C}}_j$ for some approximation of $\mathbf{C}_j$, we can write:

$$\begin{aligned}
&\left\|\mathbf{C}_1\mathbf{C}_2\ldots\ \mathbf{C}_{k(i)} - \tilde{\mathbf{C}}_1\tilde{\mathbf{C}}_2\ldots\tilde{\mathbf{C}}_{k(i)}\right\| \\
&\qquad \leq \left\|\mathbf{C}_1\mathbf{C}_2\ldots\mathbf{C}_{k(i)-1}\left(\mathbf{C}_{k(i)} - \tilde{\mathbf{C}}_{k(i)}\right)\right\| \\
&\qquad + \left\|\mathbf{C}_1\mathbf{C}_2\ldots\mathbf{C}_{k(i)-2}\left(\mathbf{C}_{k(i)-1} - \tilde{\mathbf{C}}_{k(i)-1}\right)\tilde{\mathbf{C}}_{k(i)}\right\| \\
&\qquad + \left\|\mathbf{C}_1\mathbf{C}_2\ldots\mathbf{C}_{k(i)-3}\left(\mathbf{C}_{k(i)-2} - \tilde{\mathbf{C}}_{k(i)-2}\right)\tilde{\mathbf{C}}_{k(i)-1}\tilde{\mathbf{C}}_{k(i)}\right\| \\
&\qquad \vdots \\
&\qquad + \left\|\left(\mathbf{C}_1 - \tilde{\mathbf{C}}_1\right)\tilde{\mathbf{C}}_1\tilde{\mathbf{C}}_2\ldots\tilde{\mathbf{C}}_{k(i)}\right\|
\end{aligned} \tag{6–4}$$

We just need then to compute some approximation $\tilde{\mathbf{C}}_{k(i)}$ of $\mathbf{C}_{k(i)}$, guaranteeing the first term in (6–4) to be less than 2^{-n-m-m}, and then choose some approximation $\tilde{\mathbf{C}}_{k(i)-1}$ of $\mathbf{C}_{k(i)-1}$, guaranteeing the second term in (6–4) to be less than 2^{-n-m-m}, and so on. Doing so, we will solve our problem, as $\tilde{\mathbf{C}}_1\tilde{\mathbf{C}}_2\ldots\tilde{\mathbf{C}}_{k(i)}$ will provide an estimation of $\mathbf{T}_i$, with an error less than $(x+1)2^{-n-m-m} \leq 2^{-n-m}$ by (6–4). For the first term of (6–4), the point is that we know from a reasoning similar to previous computations (namely bounds such as (6–3)) that we have

$$|||\mathbf{C}_1\mathbf{C}_2\ldots\mathbf{C}_{k(i)-1}||| \leq 2^{p_1(x,Y)}$$

for some polynomial p_1.

Consequently, it is sufficient to take $\left\|\mathbf{C}_{k(i)} - \tilde{\mathbf{C}}_{k(i)}\right\| \leq 2^{-n-2m-p_1(x,Y)}$ to guarantee an error less than 2^{-n-m-m}.

A similar analysis applies for the second and other terms that involve finitely many terms. The whole approach takes a time that remains polynomial in x and Y. □

The previous statements lead to the following:

Lemma 6.3 (Intrinsic complexity of linear $\mathcal{L}$–ODE) *Let* $\mathbf{f}$ *be a solution of the linear* $\mathcal{L}$*–ODE:*

$$\begin{aligned}
\mathbf{f}(0,\mathbf{y}) &= \mathbf{g}(\mathbf{y}) \\
\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \ell} &= \mathbf{u}(\mathbf{f}(x,\mathbf{y}),\mathbf{h}(x,\mathbf{y}),x,\mathbf{y})
\end{aligned}$$

where $\mathbf{u}$ *is essentially linear in* $\mathbf{f}(x,\mathbf{y})$*. Assume that the functions* $\mathbf{u},\mathbf{g},\mathbf{h}$ *are computable in polynomial time. Then,* $\mathbf{f}$ *is computable in polynomial time.*

Proof From Lemma 2.5, $\mathbf{f}(x,\mathbf{y})$ can also be given by $\mathbf{f}(x,\mathbf{y}) = \mathbf{F}(\ell(x),\mathbf{y})$ where $\overline{\mathbf{F}}$ is the solution of the initial value problem:

$$\begin{aligned}
\overline{\mathbf{F}}(1,\mathbf{y}) &= \mathbf{g}(\mathbf{y}) \\
\frac{\partial \overline{\mathbf{F}}(t,\mathbf{y})}{\partial t} &= \mathbf{u}(\overline{\mathbf{F}}(t,\mathbf{y}),2^t-1,\mathbf{y}).
\end{aligned}$$

Functions $\mathbf{u}$ are tanh–polynomial expressions that are essentially linear in $\mathbf{f}(x,\mathbf{y})$. So there exist matrices $\mathbf{A}$, $\mathbf{B}$ that are essentially constants in $\mathbf{f}(t,\mathbf{y})$ such that

$$\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \ell} = \mathbf{A}(\mathbf{f}(x,\mathbf{y}),\mathbf{h}(x,\mathbf{y}),x,\mathbf{y}) \cdot \mathbf{f}(x,\mathbf{y}) + \mathbf{B}(\mathbf{f}(x,\mathbf{y}),\mathbf{h}(x,\mathbf{y}),x,\mathbf{y}).$$

In other words, it holds that

$$\mathbf{F}'(t,\mathbf{y}) = \overline{\mathbf{A}}(\mathbf{F}(t,\mathbf{y}),t,\mathbf{y}) \cdot \mathbf{F}(t,\mathbf{y}) + \overline{\mathbf{B}}(\mathbf{F}(t,\mathbf{y}),t,\mathbf{y})$$

by setting:

$$\begin{aligned} \overline{\mathbf{A}}(\mathbf{F}(t,\mathbf{y}),t,\mathbf{y}) &= \mathbf{A}(\overline{\mathbf{F}}(t,\mathbf{y}),\mathbf{h}(2^t-1,\mathbf{y}),2^t-1,\mathbf{y}) \\ \overline{\mathbf{B}}(\mathbf{F}(t,\mathbf{y}),t,\mathbf{y}) &= \mathbf{B}(\overline{\mathbf{F}}(t,\mathbf{y}),\mathbf{h}(2^t-1,\mathbf{y}),2^t-1,\mathbf{y}) \end{aligned}$$

The corresponding matrix $\overline{\mathbf{A}}$ and vector $\overline{\mathbf{B}}$ are essentially constant in $\mathbf{F}(t,\mathbf{y})$. Also, functions $\mathbf{g},\mathbf{h}$ are computable in polynomial time, more precisely polynomial in $\ell(x)$, hence in t, and $\ell(y)$. Given t, obtaining 2^t-1 is immediate. This guarantees that all hypotheses of Lemma 6.2 are true. We can then conclude by remarking, again, that $t = \ell(x)$. □

This proves Lemma 6.1.

6.2 Proof of the main result

Theorem 1.3 follows from point (1) of the next proposition for one inclusion, and previous simulation of Turing machines for the other.

Proposition 6.4 (1) *All functions of* $\mathbb{LDL}^\circ$ *are computable (in the sense of computable analysis) in polynomial time.*

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

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

The proof 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. The proof is similar to the statement about $\mathbb{LDL}^\bullet$ in [2] from Blanc and Bournez.

Proof of Proposition 6.4, 1. This is proved by induction. This is true for basis functions, from basic arguments from computable analysis. In particular, tanh is computable in polynomial time from standard arguments.

Now, the class of polynomial time computable functions is preserved by composition. This is proved in Ko [28]: in short, the idea of the proof for *composition*(f, g), is that by the induction hypothesis, there exists M_f and M_g two Turing machines computing in polynomial time $f\colon \mathbb{R} \to \mathbb{R}$ and $g\colon \mathbb{R} \to \mathbb{R}$. In order to compute *composition*$(f, g)(x)$ with precision 2^{-n}, we just need to compute $g(x)$ with a precision $2^{-m(n)}$, where $m(n)$ is the polynomial modulus function of f. Then, we compute $f(g(x))$, which, by definition of M_f takes a polynomial time in n. Thus, since polynomial time with an oracle in polynomial time is polynomial time, *composition*(f, g) is computable in polynomial time, so the class of polynomial time computable functions is preserved under composition. It only remains to prove that the class of polynomial time computable functions is preserved by the linear length ODE schema; this is Lemma 6.1. □

We now go to various applications of the proposition and our toolbox. First, we state a characterisation of **FPTIME** for general functions, covering both the case of a function $\mathbf{f}\colon \mathbb{N}^d \to \mathbb{R}^{d'}$, $\mathbf{f}\colon \mathbb{R}^d \to \mathbb{R}^{d'}$ as a special case; only the first type (sequences) was covered by [2] from Blanc and Bournez.

Theorem 6.5 (Theorem 1.5) *A function* $\mathbf{f}\colon \mathbb{R}^d \times \mathbb{N}^{d''} \to \mathbb{R}^{d'}$ *is computable in polynomial time iff there exists* $\tilde{\mathbf{f}}\colon \mathbb{R}^d \times \mathbb{N}^{d''+2} \to \mathbb{R}^{d'} \in \mathbb{LDL}^{\circ}$ *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 6.5 follows from Proposition 6.4, (1.) and arguments from computable analysis.

Proof of Theorem 6.5 Assume 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 $\mathbf{x} \in \mathbb{R}^d$, $X \in \mathbb{N}$, $\mathbf{x} \in \left[-2^X, 2^X\right]$, $\mathbf{m} \in \mathbb{N}^{d''}$, $n \in \mathbb{N}$, $\|\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^n) - \mathbf{f}(\mathbf{x}, \mathbf{m})\| \leq 2^{-n}$.

From Proposition 6.4, (1.), we know that $\tilde{\mathbf{f}}$ is computable in polynomial time (in the binary length of its arguments). Then $\mathbf{f}(\mathbf{x}, \mathbf{m})$ is computable: indeed, given $\mathbf{x}$, $\mathbf{m}$ and n, we can approximate $\mathbf{f}(\mathbf{x}, \mathbf{m})$ at precision 2^{-n} on $[-2^X, 2^X]$ as follows: approximate $\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^{n+1})$ at precision $2^{-(n+1)}$ by some rational q, and output q. We will then have:

$$\begin{aligned} \|q - \mathbf{f}(\mathbf{x}, \mathbf{m})\| \quad & \leq \|q - \tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^{n+1})\| + \|\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^{n+1}) - \mathbf{f}(\mathbf{x}, \mathbf{m})\| \\ & \leq 2^{-(n+1)} + 2^{-(n+1)} \\ & \leq 2^{-n} \end{aligned}$$

All of this is done in polynomial time in n and the size of $\mathbf{m}$, and hence, we get that $\mathbf{f}$ is polynomial time computable from definitions.

For the direct implication, for sequences, that is to say, functions of type $\mathbf{f}\colon \mathbb{N}^{d''} \to \mathbb{R}^{d'}$ (ie $d = 0$, the case considered in Blanc and Bournez [2]) we are almost done: reasoning componentwise, we only need to consider $f\colon \mathbb{N}^{d''} \to \mathbb{R}$ (ie $d' = 1$). As the function is polynomial time computable, this means that there is a polynomial time computable function $g\colon \mathbb{N}^{d''+1} \to \{1,3\}^*$ 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 $\mathbf{m}$.

Remark 10 This is basically what is done in Blanc and Bournez [2], except that we do it here with analytic functions. However, as already observed there, this cannot be done for the case $d \geq 1$, ie for example, for $f\colon \mathbb{R} \to \mathbb{R}$. 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 functions must be continuous, and there is no way to map continuously real numbers to finite words. This is where the approach of the article Blanc and Bournez [2] is stuck.

The problem is then to decode, compute and encode the result to produce this dyadic using our previous toolbox.

More precisely, from Proposition 4.3, we get $\tilde{g}$ with

$$|\tilde{g}(2^e, 2^{p(max(\mathbf{m},n))}, Decode(2^e, \mathbf{m}, n)) - \gamma_{word}(g(\mathbf{m},n))| \leq 2^{-e}$$

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

$$\tilde{\mathbf{f}}(\mathbf{m},n) = EncodeMul(2^e, 2^t, \tilde{g}(2^e, 2^t, Decode(2^e, \mathbf{m}, n)), 1)$$

where $t = p(max(\mathbf{m},n))$, $e = \max(p(max(\mathbf{m},n)), n)$ provides a solution such that $\|\tilde{\mathbf{f}}(\mathbf{m}, 2^n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$. □

To solve this, we use an adaptive barycentric technique. For simplicity and pedagogy, we discuss only the case of a polynomial time computable function $f\colon \mathbb{R} \times \mathbb{N} \to \mathbb{R}$. From standard arguments from computable analysis (see, eg Ko[28, Corollary 2.21]), the following holds, and the point is to be able to realise all this with functions from $\mathbb{LDL}^{\circ}$.

Lemma 6.6 *Assume $f\colon \mathbb{R}\times\mathbb{N}\to\mathbb{R}$ is computable in polynomial time. There exists some polynomial $m\colon \mathbb{N}^2\to\mathbb{N}$ and some $\tilde{f}\colon \mathbb{N}^4\to\mathbb{Z}$ computable in polynomial time such that for all $x\in\mathbb{R}$, $|2^{-n}\tilde{f}(\lfloor 2^{m(n,M)}x\rfloor, u, 2^M, 2^n) - f(x,u)| \le 2^{-n}$ 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 floor function given by Lemma 3.7. Then, given n,M, when $2^{m(n,M)}x$ falls in some suitable interval I_i for σ_i (see the statement of Lemma 3.7), 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 $Formula_i(x,u,M,n)$ of the form $EncodeMul(2^e, 2^t, \tilde{\tilde{f}}(2^2, 2^t, Decode(2^e, \sigma_i(2^e, 2^M, 2^{m(n,M)}x))), 2^{-n})$.

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 $Formula_i(x,u,\ M,n)$. 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 cover 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, $Formula_1(x,u,M,n)$ and $Formula_2(x,u,M,n)$ may differ, but they are both 2^{-n} approximations of $f(x)$.

The key is to compute some suitable "adaptive" barycenter, using function λ, provided by Corollary 3.8. Writing $\approx$ for the fact that two values are closed up to some controlled bounded error, observe from the statements of Corollary 3.8 and 3.7

- that whenever $\lambda(\cdot,2^n,x)\approx 0$, we know that $\sigma_2(\cdot,2^n,x)\approx\lfloor x\rfloor$;
- that whenever $\lambda(\cdot,2^n,x)\approx 1$ we know that $\sigma_1(\cdot,2^n,x)\approx\lfloor x\rfloor$; and
- that 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

$$\lambda(\cdot,2^n,x)Formula_1(x,u,M,n) + (1-\lambda(\cdot,2^n,n))Formula_2(x,u,M,n)$$

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 $Formula_i(x,u,M,n)$, this could be also be written as follows, ending the proof of Theorem 6.5.

$$
\begin{aligned}
&EncodeMul(2^e, 2^t, \tilde{\tilde{f}}(2^e, 2^t, Decode(2^e, \sigma_1(2^e, 2^M, 2^{m(n,M)}x))), \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad \lambda(2^e, 2^M, 2^{m(n,M)}x)2^{-n}) \\
&+EncodeMul(2^e, 2^t, \tilde{\tilde{f}}(2^e, 2^t, Decode(2^e, \sigma_2(2^e, 2^M, 2^{m(n,M)}x))), \\
&\qquad\qquad\qquad\qquad\qquad\qquad (1 - \lambda(2^e, 2^M, 2^{m(n,M)}x))2^{-n})
\end{aligned} \tag{6–5}
$$

Proof of Theorem 1.3 We know that a function $\mathbf{f}\colon \mathbb{R}^d \to \mathbb{R}^{d'}$ from $\mathbb{LDL}^\circ$ is polynomial time computable by Proposition 6.4, (1.). That means we can approximate it with arbitrary precision, in particular, precision $\frac{1}{4}$ in polynomial time. Given such an approximation $\mathbf{q}$, if we know it is some integer, it is easy to determine which integer it is: return (componentwise) the closest integer to $\mathbf{q}$.

Conversely, if we have a function $\mathbf{f}\colon \mathbb{N}^d \to \mathbb{N}^{d'}$ that is polynomial time computable, our previous simulations of Turing machines provide a function in $\mathbb{LDL}^\circ$ that computes it at any required precision, in particular $1/4$. □

Since we have the reverse direction in Theorem 6.5, it is natural to consider the operation that maps $\tilde{\mathbf{f}}$ to $\mathbf{f}$.

Theorem 6.7 *A continuous function* $\mathbf{f}$ *is computable in polynomial time if and only if all its components belong to* $\overline{\mathbb{LDL}^\circ}$, *where*

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

with ELim defined in Definition 1.4.

For the reverse direction, by induction, the only thing to prove is that the class of functions from to 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 1.5 as a corollary.

From the proofs, we also get a normal form theorem, namely formula (6–5). In particular:

Theorem 6.8 *Any function* $f\colon \mathbb{N}^d \times \mathbb{R}^{d''} \to \mathbb{R}^{d'}$ *can be obtained from the class* $\overline{\mathbb{LDL}^\circ}$ *using only one schema ELim.*

We obtain the statements for polynomial space computability (Theorems 1.6 and 1.7) replacing $\mathbb{LDL}^\circ$ by $\mathbb{RLD}^\circ$, using similar reasoning about space instead of time, considering point 2. instead of 1. of Proposition 6.4, and Proposition 4.4 instead of Proposition 4.3.

We now comment on the relations with formal neural network models. In this article, we are programming with tanh and sigmoids. We expressed the sigmoids in terms of the ReLU function through Lemma 3.3. Function tanh could be replaced by arctan: the key was to be able to approximate the ReLU function with tanh (Lemma 3.3), and this can be proved to hold also for arctan, using error bounds on arctan established in Graça [25].

Now, given some function $\mathbf{f}\colon \mathbb{R}^d \to \mathbb{R}^{d'}$, some error and some time t, our expressions provide explicit expressions in $\mathbb{LDL}^\circ$ of an approximation of what is computed by a Turing machine at time t uniformly over any compact domain.

Remark 11 The formula 6–5 can be seen as a function that generates uniformly a family of circuits/formal $\mathcal{C}$–$\mathfrak{s}$ approximating a given function at some given precision over some given domain. The functions we generate are the composition of essentially linear functions, which can be considered as layers of formal neural networks[1].

7 Conclusion

In this article, we proved that functions over the real computable in polynomial time can be characterised using discrete ordinary differential equations (ODE). It gave us an algebra for **FPTIME** $\cap\, \mathbb{R}^{\mathbb{R}}$ and **FPSPACE** $\cap\, \mathbb{R}^{\mathbb{R}}$.

Actually, our characterizations also cover **FPTIME** $\cap \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}}$ and **FPSPACE** $\cap \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}}$ for integers d and d'. In particular, this extends even the characterisations for sequences over the reals from Blanc and Bournez [2], ie the case $d' = 0$. A major improvement of our characterisation, in comparison to the one for sequences in Blanc and Bournez [2], is that it was obtained using only analytic functions (ie no need for sign function). It required some continuous approximation of several discontinuous or non-analytic functions (floor function, the Euclidean division...). Furthermore, it

[1] With a concept of neural network that is not assuming that the last layer of the network is made of neurons, and that result may be outputted by some linear combination of the neurons in the last layer.

required a barycentric method, inspired by some constructions of Bournez, Campagnolo, Graça and Hainry [8]. Eventually, our characterisations cover space complexity.

In a more abstract view, all of this was done by proving that we can simulate Turing machines with analytic discrete ordinary differential equations. We believe this result opens the way to many applications, as it opens the possibility of programming with (discrete time) ordinary differential equations with an underlying well-understood time and space complexity.

Regarding future work, our characterisations use discrete ODEs. We are investigating whether similar statements could be using classical continuous ODEs, which, from an ordinary differential point of view, are more natural.

References

[1] **S Bellantoni**, **S Cook**, *A new recursion-theoretic characterization of the poly-time functions*, Computational Complexity 2 (1992) 97–110; https://doi.org/10.1007/BF01201998

[2] **M Blanc**, **O Bournez**, *A Characterization of Polynomial Time Computable Functions from the Integers to the Reals Using Discrete Ordinary Differential Equations*, from: "Machines, Computations, and Universality - 9th International Conference, MCU 2022, Debrecen, Hungary, August 31 - September 2, 2022, Proceedings", (J Durand-Lose, G Vaszil, editors), Lecture Notes in Computer Science 13419, Springer (2022) 58–74MCU'22 Best Student Paper Award; https://doi.org/10.1007/BF01201998

[3] **M Blanc**, **O 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*, from: "48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France", (J Leroux, S Lombardy, D Peleg, editors), LIPIcs 272, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023) 21:1–21:15; https://doi.org/10.4230/LIPIcs.MFCS.2023.21

[4] **M Blanc**, **O Bournez**, *Measuring robustness of dynamical systems. Relating time and space to length and precision*, Technical report (2023); https://doi.org/10.48550/arXiv.2301.12723

[5] **M Blanc**, **O Bournez**, *A characterization of polynomial time computable functions from the integers to the reals using discrete ordinary differential equations*, International Journal of Foundations of Computer Science (2024), to appear. Journal version of [2]. https://arxiv.org/abs/2209.13599

[6] **L Blum**, **F Cucker**, **M Shub**, **S Smale**, *Complexity and Real Computation*, Springer (1998); https://doi.org/10.1007/978-1-4612-0701-6

[7] **O Bournez**, *Programming with Ordinary Differential Equations: Some First Steps Towards a Programming Language*, from: "Revolutions and Revelations in Computability - 18th Conference on Computability in Europe, CiE 2022, Swansea, UK, July 11-15, 2022, Proceedings", (U Berger, J N Y Franklin, F Manea, A Pauly, editors), Lecture Notes in Computer Science 13359, Springer (2022) 39–51; https://doi.org/10.1007/978-3-031-08740-0_4

[8] **O Bournez**, **M L Campagnolo**, **D Graça**, **E S Hainry**, *Polynomial differential equations compute all real computable functions on computable compact intervals*, Journal of Complexity 23 (2007) 317–335; https://doi.org/10.1016/j.jco.2006.12.005

[9] **O Bournez**, **F Cucker**, **P J de Naurois**, **J-Y Marion**, *Implicit Complexity over an Arbitrary Structure: Sequential and Parallel Polynomial Time*, Journal of Logic and Computation 15 (2005) 41–58; https://doi.org/10.1093/logcom/exh036

[10] **O Bournez**, **F Cucker**, **P J de Naurois**, **J-Y Marion**, *Implicit Complexity over an Arbitrary Structure: Quantifier Alternations*, Information and Computation 202 (2006) 210–230; https://doi.org/10.1016/j.ic.2005.07.005

[11] **O Bournez**, **A Durand**, *Recursion Schemes, Discrete Differential Equations and Characterization of Polynomial Time Computations*, from: "44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany", (P Rossmanith, P Heggernes, J Katoen, editors), LIPIcs 138, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019) 23:1–23:14; https://doi.org/10.4230/LIPIcs.MFCS.2019.23

[12] **O Bournez**, **A Durand**, *A Characterization of Functions over the Integers Computable in Polynomial Time Using Discrete Ordinary Differential Equations*, Computational Complexity 32 (2023); https://doi.org/10.1007/s00037-023-00240-1

[13] **O Bournez**, **W Gomaa**, **E Hainry**, *Algebraic Characterizations of Complexity-Theoretic Classes of Real Functions*, IJUC 7 (2011) 331–351

[14] **O Bournez**, **A Pouly**, *A Universal Ordinary Differential Equation*, from: "44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland", (I Chatzigiannakis, P Indyk, F Kuhn, A Muscholl, editors), LIPIcs 80, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017) 116:1–116:14; https://doi.org/10.4230/LIPIcs.ICALP.2017.116

[15] **O Bournez**, **A Pouly**, *A Survey on Analog Models of Computation*, from: "Handbook of Computability and Complexity in Analysis", (V Brattka, P Hertling, editors), Springer (2021); https://doi.org/10.1007/978-3-030-59234-9_6

[16] **V Brattka**, *Recursive characterization of computable real-valued functions and relations*, Theoretical Computer Science 162 (1996) 45–77; https://doi.org/10.1016/0304-3975(95)00249-9

[17] **V Brattka**, **P Hertling**, **K Weihrauch**, *A tutorial on computable analysis*, from: "New computational paradigms", Springer (2008) 425–491; https://doi.org/10.1007/978-0-387-68546-5_18

[18] **P Clote**, *Computational Models and Function Algebras*, from: "Handbook of Computability Theory", (E R Griffor, editor), North-Holland, Amsterdam (1998) 589–681; https://doi.org/10.1016/s0049-237x(99)80033-0

[19] **P Clote**, **E Kranakis**, *Boolean Functions and Computation Models*, Texts in Theoretical Computer Science. An EATCS Series, Springer (2002); https://doi.org/10.1007/978-3-662-04943-3

[20] **A Cobham**, *The intrinsic computational difficulty of functions*, from: "Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science", (Y Bar-Hillel, editor), North-Holland, Amsterdam (1962) 24–30

[21] **F Fages**, **G L Guludec**, **O Bournez**, **A Pouly**, *Strong Turing Completeness of Continuous Chemical Reaction Networks and Compilation of Mixed Analog-Digital Programs*, from: "Computational Methods in Systems Biology - 15th International Conference, CMSB 2017, Darmstadt, Germany, September 27-29, 2017, Proceedings", (J Feret, H Koeppl, editors), Lecture Notes in Computer Science 10545, Springer (2017) 108–127; https://doi.org/10.1007/978-3-319-67471-1_7

[22] **H Férée**, **W Gomaa**, **M Hoyrup**, *Analytical properties of resource-bounded real functionals*, Journal of Complexity 30 (2014) 647–671; https://doi.org/10.1016/j.jco.2014.02.008

[23] **H Férée**, **E Hainry**, **M Hoyrup**, **R Péchoux**, *Interpretation of stream programs: characterizing type 2 polynomial time complexity*, from: "Algorithms and Computation: 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I 21", Springer (2010) 291–303; https://doi.org/10.1007/978-3-642-17517-6_27

[24] **M Gaboardi**, **J Marion**, **S R D Rocca**, *A logical account of pspace*, from: "Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008", (G C Necula, P Wadler, editors), ACM (2008) 121–131; https://doi.org/10.1145/1328438.1328456

[25] **D S Graça**, *Computability with Polynomial Differential Equations*, PhD thesis, Instituto Superior Técnico (2007)

[26] **D S Graça**, **N Zhong**, *Handbook of Computability and Complexity in Analysis*, Springer. (2018)

[27] **E Hainry**, **D Mazza**, **R Péchoux**, *Polynomial time over the reals with parsimony*, from: "Functional and Logic Programming: 15th International Symposium, FLOPS 2020, Akita, Japan, September 14–16, 2020, Proceedings 15", Springer (2020) 50–65; https://doi.org/10.1007/978-3-030-59025-3_4

[28] **K Ko**, *Complexity theory of real functions*, volume 3 of *Progress in theoretical computer science*, Birkhäuser, Boston (1991)

[29] **D Leivant**, *Intrinsic Theories and Computational Complexity*, from: "Logical and Computational Complexity. Selected Papers. Logic and Computational Complexity, International Workshop LCC '94, Indianapolis, Indiana, USA, 13-16 October 1994", (D Leivant, editor), Lecture Notes in Computer Science 960, Springer (1994) 177–194; https://doi.org/10.1007/3-540-60178-3_84

[30] **D Leivant**, *Predicative recurrence and computational complexity I: Word recurrence and poly-time*, from: "Feasible Mathematics II", (P Clote, J Remmel, editors), Birkhäuser (1994) 320–343; https://doi.org/10.1007/978-1-4612-2566-9_11

[31] **D Leivant**, **J Marion**, *Lambda Calculus Characterizations of Poly-Time*, Fundam. Informaticae 19 (1993) 167–184; https://doi.org/10.3233/FI-1993-191-207

[32] **D Leivant**, **J Marion**, *Ramified Recurrence and Computational Complexity II: Substitution and Poly-Space*, from: "Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers", (L Pacholski, J Tiuryn, editors), Lecture Notes in Computer Science 933, Springer, Kazimierz, Poland (1994) 486–500; https://doi.org/10.1007/BFb0022277

[33] **J Marion**, **R Péchoux**, *Characterizations of polynomial complexity classes with a better intensionality*, from: "Proceedings of the 10th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 15-17, 2008, Valencia, Spain", (S Antoy, E Albert, editors), ACM (2008) 79–88; https://doi.org/10.1145/1389449.1389460

[34] **K M Ng**, **N R Tavana**, **Y Yang**, *A recursion theoretic foundation of computation over real numbers*, Journal of Logic and Computation 31 (2021) 1660–1689; https://doi.org/10.1093/logcom/exab042

[35] **M B Pour-El**, **J I Richards**, *A computable ordinary differential equation which possesses no computable solution*, Annals of Mathematical Logic 17 (1979) 61–90; https://doi.org/10.1016/0003-4843(79)90021-4

[36] **H T Siegelmann**, *Neural networks and analog computation - beyond the Turing limit*, Progress in theoretical computer science, Birkhäuser (1999); https://doi.org/10.1007/978-1-4612-0707-8_12

[37] **H T Siegelmann**, **E D Sontag**, *On the Computational Power of Neural Nets*, Journal of Computer and System Sciences 50 (1995) 132–150; https://doi.org/10.1006/jcss.1995.1013

[38] **D B Thompson**, *Subrecursiveness: Machine-independent notions of computability in restricted time and storage*, Mathematical Systems Theory 6 (1972) 3–15; https://doi.org/10.1007/BF01706069

[39] **K Weihrauch**, *Computable Analysis - An Introduction*, Texts in Theoretical Computer Science. An EATCS Series, Springer (2000); https://doi.org/10.1007/978-3-642-56999-9

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

Manon.Blanc@lix.polytechnique.fr, Olivier.Bournez@polytechnique.fr

https://perso.crans.org/mblanc/
https://www.lix.polytechnique.fr/~bournez/

Received: 20 July 2023 Revised: 6 January 2025