Recursion Schemes, Discrete Differential Equations and Characterization of Polynomial Time Computations

Olivier Bournez
Laboratoire d'Informatique de l'X (LIX), CNRS, Ecole Polytechnique,
Institut Polytechnique de Paris, 91128 Palaiseau Cedex, France
bournez@lix.polytechnique.fr

Arnaud Durand
Université Paris Diderot, IMJ-PRG, CNRS UMR 7586, Case 7012, 75205 Paris cedex 13, France
durand@math.univ-paris-diderot.fr

Abstract

This paper studies the expressive and computational power of discrete Ordinary Differential Equations (ODEs). It presents a new framework using discrete ODEs as a central tool for computation and algorithm design. We present the general theory of discrete ODEs for computation theory, we illustrate this with various examples of algorithms, and we provide several implicit characterizations of complexity and computability classes.

The proposed framework presents an original point of view on complexity and computation classes. It unifies several constructions that have been proposed for characterizing these classes including classical approaches in implicit complexity using restricted recursion schemes, as well as recent characterizations of computability and complexity by classes of continuous ordinary differential equations. It also helps understanding the relationships between analog computations and classical discrete models of computation theory.

At a more technical point of view, this paper points out the fundamental role of linear (discrete) ordinary differential equations and classical ODE tools such as changes of variables to capture computability and complexity measures, or as a tool for programming many algorithms.

2012 ACM Subject Classification Theory of computation; Theory of computation → Models of computation; Theory of computation → Computability; Theory of computation → Recursive functions; Computer systems organization → Analog computers; Theory of computation → Complexity classes; Theory of computation → Complexity theory and logic; Mathematics of computing → Differential equations; Mathematics of computing → Ordinary differential equations; Mathematics of computing → Differential calculus

Keywords and phrases Implicit complexity, discrete ordinary differential equations, recursion scheme

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.23

Funding *Olivier Bournez*: Supported by RACAF Project from Agence National de la Recherche and Labex Digicosme Project ACDC.

Acknowledgements We would like to thank Sabrina Ouazzani for many scientific discussions about the results in this article.

1 Introduction

Since the beginning of its foundations, classification of the difficulty of problems, with various models of computation, either by their complexity or by their computability properties, is a thriving field of computer science. Nowadays, classical computer science problems also deal with continuous data coming from different areas and modeling involves the use of tools like numerical analysis, probability theory or differential equations. Thus new characterizations

related to theses fields have been proposed. On a dual way, the quest for new types of computers recently led to revisit the power of some models for analog machines based on differential equations, and to compare them to modern digital models. In both contexts, when discussing the related computability or complexity issues, one has to overcome the fact that today's (digital) computers are in essence discrete machines while the objects under study are continuous and naturally correspond to Ordinary Differential Equations (ODEs).

We consider here an original approach in between the two worlds: discrete oriented computation with differential equations.

ODEs appear to be a natural way of expressing properties and are intensively used, in particular in applied science. The theory of classical (continuous) ODEs has an abundant literature (see e.g. [1, 3, 9]) and is rather well understood under many aspects. We are interested here in a discrete counterpart of classical continuous ODEs: discrete ODEs. Its associated derivative notion, called *finite differences*, has been widely studied in numerical optimization for function approximation [12] and is reminiscent in *discrete calculus* [14, 13, 15, 20] for combinatorial analysis (remark that similarities between discrete and continuous statements have also been historically observed, under the terminology of *umbral* or *symbolic calculus* as early as in the 19th century). However, even if the underlying computational content of finite differences theory is clear and has been pointed out many times, no fundamental connections with algorithms and complexity have been exhibited so far.

In this article, our goal is to demonstrate that discrete ODEs is a very natural tool for algorithm design and to prove that complexity and computability notions can be elegantly and simply captured using discrete ordinary differential equations. We illustrate this by providing a characterization of **FPTIME**, the class of polynomial time computable functions, and of its non deterministic analog **FNP**. To this aim, we will also demonstrate how some notions from the analog world such as linearity of differential equations or derivatives along some particular functions (i.e. changes of variables) are representative of a certain computational hardness and can be used to solve efficiently some (classical, digital) problems.

As far as we know, this is the first time that computations with discrete ODEs and their related complexity aspects are considered. By contrast, complexity results have been recently obtained about classical (continuous) ODEs for various classes of functions, mostly in the framework of computable analysis. The hardness of solving continuous ODEs has been intensively discussed: for example [19] establishes some bases of the complexity aspects of ODEs and more recent works like [18] or [10] establish links between complexity or effective aspects of such differential equations. We believe that investigating the expressive power of discrete ODE, can help to better understand complexity of computation for both the discrete and continuous settings. Indeed, on the one hand, our results offers a new machine independent perspective on classical discrete computations, i.e. computations that deal with bits, words, or integers. And, on the other hand, it relates classical (discrete) complexity classes to analog computations, i.e. computations over the reals, as analog computation have been related in various ways to continuous ordinary differential equations, and as discrete ordinary differential equations provide clear hints about their continuous counterparts. A mid-term goal of this line of research is also to bring insights from complexity theory to the problem of solving ODE (discrete and, hopefully also, numerical). A descriptive approach such as the one initiated in the paper could help classifying large classes of ODE by their computational hardness and bring some uniformity to methods of this field.

From restricted recursion scheme to discrete differential equations

Recursion schemes constitutes a major approach of computability theory and to some extent of complexity theory. A foundational result in that spirit is due to Cobham, who gave in [8] a characterization of functions computable in polynomial time through the notion of *bounded recursion on notations* (BRN, for short). A function f is defined by BRN from g, h_0, h_1, k if $f(0, \mathbf{y}) = g(\mathbf{y})$; $f(\mathbf{0}(x), \mathbf{y}) = h_0(f(x, \mathbf{y}), x, \mathbf{y})$ for $x \neq 0$; $f(\mathbf{1}(x), \mathbf{y}) = h_1(f(x, \mathbf{y}), x, \mathbf{y})$ with, for all $x, \mathbf{y}$: $f(x, \mathbf{y}) \leq k(x, \mathbf{y})$. Here $\mathbf{0}(.)$ and $\mathbf{1}(.)$ denote the successor functions defined by $\mathbf{0}(x) = 2.x$ and $\mathbf{1}(x) = 2.x + 1$. In this approach, the number of steps is controlled through the use of the alternative successors $\mathbf{0}(.)$ and $\mathbf{1}(.)$ (which increase the size of their argument by one in each round) and the size of objects through an explicit upper bound by a given function k. Later, notions such as safe recursion [2] or ramification ([22, 21] have allowed syntactical characterizations of polynomial time or other classes [23] that do not require the use of an explicit bound but at the expense of rather sophisticated schemes. These works have therefore been at the origin of the very vivid field of *implicit complexity* at the interplay of logic and theory of programming.

A discrete function can also be described by its derivative i.e. the value $\mathbf{f}(x+1, \mathbf{y}) - \mathbf{f}(x, \mathbf{y})$. It is straightforward to rewrite primitive recursion in this setting, though one may have to cope with possibly negative values. To capture more fine grained time and space measures we come up in the proposed context of discrete ODEs with at least two original concepts which are very natural in the continuous setting:

- deriving along a function; when such a function is suitably chosen this allows to control the number of steps in the computation;
- linearity that permits to control object sizes.

By combining these two approaches, we provide a characterization of **FPTIME** that does not require to specify an explicit bound in the recursion, in contrast to Cobham's work, nor to assign a specific role or type to variables, in contrast to safe recursion or ramification. The characterization happens to be very simple from a syntactical point of view using only natural notions from the world of ODE.

This characterization is also a first step to convince the reader that deep connections between complexity and ODE solving do exist and that these connections are worth to be further studied.

Related works on analog computations

As many historical or even possibly futuristic analog machines are naturally described by (continuous) ODEs, the quest of understanding how the computational power of analog models compare to classical digital ones have led to several results relating classical complexity to various classes of continuous ODEs. In particular, a series of papers has been devoted to study various classes of the so-called $\mathbb{R}$-recursive functions, after their introduction in [25] as a theoretical model for computations over the reals. At the complexity level, characterizations of complexity classes such as **PTIME** and **NPTIME** using $\mathbb{R}$-recursive algebra have been obtained [27], motivated in particular by the idea of transferring classical questions from complexity theory to the context of real and complex analysis [24, 27, 26]. But this has been done with the addition of limit schemata and with a rather different settings.

More recently, revisiting the model of General Purpose Analog Computer of Claude Shannon, it is has been proved that polynomial differential equations can be considered as a very simple and elegant model in which computable functions over the reals and polynomial time computable functions over the reals can be defined without any reference to concepts

from discrete computation theory [4, 29]. We believe the current work is a substantial step to understand the underlying power and theory of such analog models, by providing concepts, definitions and results relating the two worlds.

Refer to [5] for an up to date survey about various works on analog computations, in particular in a computation theory perspective.

Structure of the paper

In Section 2 a short introduction to discrete differentiability is given followed in Section 3 by an illustration, through examples, of the programming ability of discrete ODE. Formal definitions of discrete ODE schemas are given in Section 4 together with characterizations of classical computability classes. Our objective, from Section 2 to 4, is to expose in a simple way (using sometimes well-known concepts) the basics of the theory of discrete ODE and its computational content, before getting to more technical results. Section 5 introduces the notion of length-ODE which is central, together with the that of (essentially) linear differential equation, for the characterization of **FPTIME** (Section 6) and **FNP** (Sections 7). In Section 8 we discuss some extensions of the results.

2 Discrete differentiability and ODEs

In this section, we review some basic notions of discrete calculus to help intuition in the rest of the paper (refer to [16, 12] for a more complete review). Discrete derivatives are usually intended to concern functions over the integers of type $\mathbf{f} : \mathbb{N}^p \to \mathbb{Z}^d$, but the statements and concepts considered in our discussions are also valid more generally for functions of type $\mathbf{f} : \mathbb{Z}^p \to \mathbb{Z}^d$, for some integers p, d, or even functions $\mathbf{f} : \mathbb{R}^p \to \mathbb{R}^d$. The basic idea is to consider the following concept of derivative

▶ Remark 1. We first discuss the case where $p = 1$, i.e. functions $\mathbf{f} : \mathbb{N} \to \mathbb{Z}^d$. We will later on consider more general functions, with partial derivatives instead of derivatives.

▶ **Definition 2** (Discrete Derivative). *The discrete derivative of* $\mathbf{f}(x)$ *is defined as* $\Delta\mathbf{f}(x) = \mathbf{f}(x+1) - \mathbf{f}(x)$. *We will also write* $\mathbf{f}'$ *for* $\Delta\mathbf{f}(x)$ *to help to understand statements with respect to their classical continuous counterparts.*

Several results from classical derivatives generalize to this settings: this includes linearity of derivation $(a \cdot f(x) + b \cdot g(x))' = a \cdot f'(x) + b \cdot g'(x)$, formulas for products and division such as $(f(x) \cdot g(x))' = f'(x) \cdot g(x+1) + f(x) \cdot g'(x) = f(x+1)g'(x) + f'(x)g(x)$.

A fundamental concept is the following:

▶ **Definition 3** (Discrete Integral). *Given some function* $\mathbf{f}(x)$, *we write* $\int_a^b \mathbf{f}(x)\delta x$ *as a synonym for* $\int_a^b \mathbf{f}(x)\delta x = \sum_{x=a}^{x=b-1} \mathbf{f}(x)$ *with the convention that it takes value* 0 *when* $a = b$ *and* $\int_a^b \mathbf{f}(x)\delta x = -\int_b^a \mathbf{f}(x)\delta x$ *when* $a > b$.

The telescope formula yields the so-called Fundamental Theorem of Finite Calculus:

▶ **Theorem 4** (Fundamental Theorem of Finite Calculus). *Let* $\mathbf{F}(x)$ *be some function. Then,* $\int_a^b \mathbf{F}'(x)\delta x = \mathbf{F}(b) - \mathbf{F}(a)$.

As for classical functions, a given function has several primitives. These primitives are defined up to some additive constant. Several techniques from the classical settings generalize to the discrete settings: this includes the technique of integration by parts.

A classical concept in discrete calculus is the one of falling power defined as $x^{\underline{m}} = x \cdot (x-1) \cdot (x-2) \cdots (x-(m-1))$. This notion is motivated by the fact that it satisfies a derivative formula $(x^{\underline{m}})' = m \cdot x^{\underline{m-1}}$ similar to the classical one for powers in the continuous setting. In a similar spirit, we introduce the concept of falling exponential.

▶ **Definition 5** (Falling exponential)**.** *Given some function* $\mathbf{U}(x)$*, the expression* $\mathbf{U}$ *to the falling exponential* x*, denoted by* $\overline{2}^{\mathbf{U}(x)}$*, stands for* $\overline{2}^{\mathbf{U}(x)} = (1+\mathbf{U}'(x-1)) \cdots (1+\mathbf{U}'(1)) \cdot (1+\mathbf{U}'(0)) = \prod_{t=0}^{t=x-1}(1+\mathbf{U}'(t))$*, with the convention that* $\prod_0^0 = \mathbf{id}$*, where* $\mathbf{id}$ *is the identity (sometimes denoted* 1 *hereafter)*

This is motivated by the remarks that $2^x = \overline{2}^x$, and that the discrete derivative of a falling exponential is given by $\left(\overline{2}^{\mathbf{U}(x)}\right)' = \mathbf{U}'(x) \cdot \overline{2}^{\mathbf{U}(x)}$ for all $x \in \mathbb{Z}$.

We will focus in this article on discrete Ordinary Differential Equations (ODE) on functions with several variables, that is to say for example on equations of the (possibly vectorial) form:

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

where $\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial x}$ stands as expected for the derivative of functions $\mathbf{f}(x,\mathbf{y})$ considered as a function of x, when $\mathbf{y}$ is fixed i.e. $\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial x} = \mathbf{f}(x+1,\mathbf{y}) - \mathbf{f}(x,\mathbf{y})$. When some initial value $\mathbf{f}(0,\mathbf{y}) = \mathbf{g}(\mathbf{y})$ is added, this is called an *Initial Value Problem (IVP)* or a *Cauchy Problem*. An IVP can always be put in integral form

$$\mathbf{f}(x,\mathbf{y}) = \mathbf{f}(0,\mathbf{y}) + \int_0^x \mathbf{h}(\mathbf{f}(x,\mathbf{y}),x,\mathbf{y})\delta x.$$

Our aim here is to discuss total functions whose domain and range is either of the form $\mathcal{D} = \mathbb{N}$, $\mathbb{Z}$, or possibly a finite product $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_k$ where each $\mathcal{D}_i = \mathbb{N}$, $\mathbb{Z}$. By considering that $\mathbb{N} \subset \mathbb{Z}$, we assume that the range is always $\mathbb{Z}^d$ for some d. The concept of solution for such ODEs is as expected: given $h : \mathbb{Z}^d \times \mathbb{N} \times \mathbb{Z}^p \to \mathbb{Z}$ (or $h : \mathbb{Z}^d \times \mathbb{Z} \times \mathbb{Z}^p \to \mathbb{Z}$), a solution over $\mathcal{D}$ is a function $f : \mathcal{D} \times \mathbb{Z}^p \to \mathbb{Z}^d$ that satisfies the equations for all $x, \mathbf{y}$.

We will only consider well-defined ODEs such as above in this article (but variants with partially defined functions could be considered as well). Observe that an IVP of the form (1) always admits a (necessarily unique) solution over $\mathbb{N}$ since f can be defined inductively with $\mathbf{f}(0,\mathbf{y}) = \mathbf{g}(\mathbf{y})$ and $\mathbf{f}(x+1,\mathbf{y}) = \mathbf{f}(x,\mathbf{y}) + \mathbf{h}(\mathbf{f}(x,\mathbf{y}),x,\mathbf{y})$.

▶ Remark 6. Notice that this is not necessarily true over $\mathbb{Z}$: As an example, consider $f'(x) = -f(x)+1$, $f(0) = 0$. By definition of $f'(x)$, we must have $f(x+1) = 1$ for all x, but if $x = -1$, $f(0) = 1 \neq 0$.

▶ Remark 7 (Sign function). It is very instructive to realize that the solution of this IVP over $\mathbb{N}$ is the sign $\mathsf{sg}_{\mathbb{N}}(x)$ function defined by $\mathsf{sg}_{\mathbb{N}}(x) = 1$ if $x > 0$ and $\mathsf{sg}_{\mathbb{N}}(x) = 0$ in the other case.

Linear (also called *Affine*) ODEs will play a very important role in what follows, i.e. discrete ordinary differential equations of the form $\mathbf{f}'(x) = \mathbf{A}(x) \cdot \mathbf{f}(x) + \mathbf{B}(x)$.

▶ Remark 8. Recall that the solution of $f'(x) = a(x)f(x) + b(x)$ for classical continous derivatives turns out to be given by (usually obtained using the method of variation of parameters): $f(x) = f(0)e^{\int_0^x a(t)dt} + \int_0^x b(u)e^{\int_u^x a(t)dt}du$. This generalizes with our definitions to discrete ordinary differential equations, and this works even vectorially:

▶ **Lemma 9** (Solution of linear ODE)**.** *For matrices* $\mathbf{A}$ *and vectors* $\mathbf{B}$ *and* $\mathbf{G}$*, the solution of equation* $\mathbf{f}'(x,\mathbf{y}) = \mathbf{A}(x,\mathbf{y}) \cdot \mathbf{f}(x,\mathbf{y}) + \mathbf{B}(x,\mathbf{y})$ *with initial conditions* $\mathbf{f}(0,\mathbf{y}) = \mathbf{G}(\mathbf{y})$ *is* $\mathbf{f}(x,\mathbf{y}) = \left(\overline{2}^{\int_0^x \mathbf{A}(t,\mathbf{y})\delta t}\right) \cdot \mathbf{G}(\mathbf{y}) + \int_0^x \left(\overline{2}^{\int_{u+1}^x \mathbf{A}(t,\mathbf{y})\delta t}\right) \cdot \mathbf{B}(u,\mathbf{y})\delta u.$

▶ Remark 10. Notice that this can be rewritten as $\sum_{u=-1}^{x-1}\left(\prod_{t=u+1}^{x-1}(1+\mathbf{A}(t,\mathbf{y}))\right)\cdot\mathbf{B}(u,\mathbf{y})$ with the conventions that for any function $\kappa(\cdot)$, $\prod_{x}^{x-1}\kappa(x)=1$ and $\mathbf{B}(-1,\mathbf{y})=\mathbf{G}(\mathbf{y})$. Such equivalent expressions both have a clear computational content. They can be interpreted as an algorithm unrolling the computation of $\mathbf{f}(x+1,\mathbf{y})$ from the computation of $\mathbf{f}(x,\mathbf{y}),\mathbf{f}(x-1,\mathbf{y}),\ldots,\mathbf{f}(0,\mathbf{y})$. The next section will build on that remark through some examples.

3 Programming with discrete ODE

The computational dimension of calculus of finite differences has been widely stressed in mathematical analysis. However, no fundamental connection has been established with algorithmic and complexity. In this section, we show that several algorithms can actually be naturally expressed as discrete ODEs.

For now, we suppose that composition of functions, constant and the following basic functions can be used freely as functions from $\mathbb{Z}$ to $\mathbb{Z}$: arithmetic operations: $+$, $-$, $\times$; $\ell(x)$ returns the length of $|x|$ written in binary; $\mathsf{sg}(x):\mathbb{Z}\to\mathbb{Z}$ (respectively: $\mathsf{sg}_{\mathbb{N}}(x):\mathbb{N}\to\mathbb{Z}$) that takes value 1 for $x>0$ and 0 in the other case; From these basic functions, for readability, one may define useful functions as synonyms: $\bar{\mathsf{sg}}(x)$ stands for $\bar{\mathsf{sg}}(x)=(1-\mathsf{sg}(x))\times(1-\mathsf{sg}(-x))$: it tests if $x=0$ for $x\in\mathbb{Z}$; $\bar{\mathsf{sg}}_{\mathbb{N}}(x)$ stands for $\bar{\mathsf{sg}}_{\mathbb{N}}(x)=1-\mathsf{sg}_{\mathbb{N}}(x)$: it tests if $x=0$ for $x\in\mathbb{N}$. $\mathsf{if}(x,y,z)$ stands for $\mathsf{if}(x,y,z)=y+\bar{\mathsf{sg}}(x)\cdot(z-y)$ and $\mathsf{if}_{\mathbb{N}}(x,y,z)$ stands for $\mathsf{if}_{\mathbb{N}}(x,y,z)=y+\bar{\mathsf{sg}}_{\mathbb{N}}(x)\cdot(z-y)$. $\mathsf{if}(x<x',y,z)$ will be a synonym for $\mathsf{if}(\mathsf{sg}(x'-x+1),y,z)$ and $\mathsf{if}(x=x',y,z)$ will be a synonym for $\mathsf{if}(1-\bar{\mathsf{sg}}(x-x'),y,z)$.

First observe that discrete ODEs allow to express easily search functions:

▶ **Example 11** (Computing the minimum of a function)**.** The minimum of a function $\min f: x\mapsto min\{f(y):0\le y\le x\}$ is given by $F(x,x)$ where F is solution of the discrete ODE $F(0,x)=f(0)$; $\frac{\partial F(t,x)}{\partial t}=H(F(t,x),f(x),t,x)$, where $H(F,f,t,x)=0$ if $F<f$, $f-F$ if $F\ge f$. In integral form, we have: $F(x,y)=F(0)+\int_0^x H(F(t,y),t,y)\delta t$.

Conversely such an integral above (equivalently discrete ODE) can always be considered as a (recursive) algorithm: compute the integral from its definition as a sum. On this example, this corresponds basically to compute $F(x,x)$ recursively by $F(t+1,x)=\mathsf{if}(F(t,x)<f(x),F(t,x),f(x))$.

▶ Remark 12. Note that this algorithm is not polynomial in the length of its argument x, as it takes time x to compute $\min f$. Getting to polynomial algorithms will be at the heart of coming discussions in the two next examples and in Section 5.

As shown in the following two examples, discrete ODEs can express more sophisticated functions and turn out to be very natural in many other contexts, in particular non numerical ones, where they would probably not be expected.

▶ **Example 13** (Computing the integer part and divisions, going to Length-ODE)**.** Suppose that we want to compute $\lfloor\sqrt{x}\rfloor=\max\{y\le x:y\cdot y\le x\}$ and $\left\lfloor\frac{x}{y}\right\rfloor=\max\{z\le x:z\cdot y\le x\}$. It can be done by the following general method. Let f,h be some functions with h being non decreasing. We compute some_h with $\mathsf{some}_h(x)=y$ s.t. $|f(x)-h(y)|$ is minimal. When $h(x)=x^2$ and $f(x)=x$, it holds that: $\lfloor\sqrt{x}\rfloor=\mathsf{if}(\mathsf{some}_h(x)^2\le x,\mathsf{some}_h(x),\mathsf{some}_h(x)-1)$. The function some_h can be computed (in non-polynomial time) as a solution of an ODE as in Example 11. However, there is a more efficient (polynomial time) way to do it based on what one usually does with classical ordinary differential equations: performing a change of variable so that the search becomes logarithmic in x. Indeed, we can write $\mathsf{some}_h(x)=G(\ell(x),x)$ for some function $G(t,x)$ that is a solution of $G(0,x)=x$; $\frac{\partial G(t,x)}{\partial t}=E(G(t,x),t,x)$ where $E(G,t,x)$ takes value $2^{\ell(x)-t-1}$ whenever $h(G>f(x))$, 0 whenever $h(G)=f(x)$ and $-2^{\ell(x)-t-1}$ whenever $h(G)<f(x)$.

▶ **Example 14** (Computing suffixes with discrete ODEs)**.** The suffix function, $\mathsf{suffix}(x, y)$ takes as input two integers x and y and outputs the $\ell(y) = t$ least significant bits of the binary decomposition of x. We describe below a way to compute a suffix working over a parameter t, that is logarithmic in x. Consider the following unusual algorithm that can be interpreted as a fix-point definition of the function: $\mathsf{suffix}(x, y) = F(\ell(x), y)$ where $F(0, x) = x$; $F(t + 1, x) = \mathsf{if}(\ell(F(t, x)) = 1, F(t, x), F(t, x) - 2^{\ell(F(t,x))-1})$.This can be interpreted as a differential equation, whose solution is converging fast again (i.e. in polynomial time) to what we want. In other words, $\mathsf{suffix}(x, y) = F(\ell(x), x)$ using the solution of the IVP: $F(0, y) = x$, $\frac{\partial F(T,y)}{\partial T} = \mathsf{if}(\ell(F(t, x)) = 1, 0, -2^{\ell(F(t,x))-1})$.

4 Computability and Discrete ODEs

Here, we consider functions defined by ODEsunder the prism of computability. It serves as an introductory illustration of the close relationship between discrete ODEs and recursive schemata before moving to efficient algorithms and complexity classes characterizations with a finer approach. This part is clearly inspired by ideas from [6, 7], but adapted here for our framework of discrete ODEs. We refer to [30, 28] for basic definitions from computability.

▶ **Definition 15** ((Scalar) Discrete ODE schemata)**.** *Given $g : \mathbb{N}^p \to \mathbb{N}$ and $h : \mathbb{Z} \times \mathbb{N}^{p+1} \to \mathbb{Z}$, we say that f is defined by discrete ODE solving from g and h, denoted by $f = \mathrm{ODE}(g, h)$, if $f : \mathbb{N}^{p+1} \to \mathbb{Z}$ corresponds to the (necessarily unique) solution of Initial Value Problem*

$$\frac{\partial f(x,\mathbf{y})}{\partial x} = h(f(x, \mathbf{y}), x, \mathbf{y}), \qquad f(0, \mathbf{y}) = g(\mathbf{y}). \tag{2}$$

▶ Remark 16. To be more general, we could take $g : \mathbb{N}^p \to \mathbb{Z}$. However, this would be of no use in the context of this paper. We could also consider vectorial equations as in what follows which are very natural in the context of ODEs, and we would get similar statements.

It is clear that primitive recursion schemes can be reformulated as discrete ODE schemata. The restriction to Linear ODEs is very natural, in particular as this class of ODEs has a highly developed theory for the continuous setting. It is very instructive to realize that the class of functions definable by Linear ODEs is exactly the well-known class of elementary functions [17] (we will see later how additional requirements lead to a characterization of polynomial time functions).

▶ **Definition 17** (Linear ODE schemata)**.** *Given a vector $\mathbf{G} = (G_i)_{1\le i\le k}$ matrix $\mathbf{A} = (A_{i,j})_{1\le i,j\le k}$, $\mathbf{B} = (B_i)_{1\le i\le k}$ whose coefficients corresponds to functions $g_i : \mathbb{N}^p \to \mathbb{N}^k$, and $a_{i,j} : \mathbb{N}^{p+1} \to \mathbb{Z}$ and $b_{i,j} : \mathbb{N}^{p+1} \to \mathbb{Z}$ respectively, we say that $\mathbf{f}$ is obtained by linear ODE solving from g, A and B, denoted by $\mathbf{f} = \mathrm{LI}(\mathbf{G}, \mathbf{A}, \mathbf{B})$, if $f : \mathbb{N}^{p+1} \to \mathbb{Z}^k$ corresponds to the (necessarily unique) solution of Initial Value Problem*

$$\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial x} = \mathbf{A}(x, \mathbf{y}) \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}(x, \mathbf{y}), \qquad \mathbf{f}(0, \mathbf{y}) = \mathbf{G}(\mathbf{y}). \tag{3}$$

▶ **Theorem 18** (A discrete ODE characterization of elementary functions)**.** *The set of elementary functions $\mathcal{E}$ is the intersection with $\mathbb{N}^{\mathbb{N}}$ of the smallest set of functions that contains the zero functions $\mathbf{0}$, the projection functions π_i^p, the successor function $\mathbf{s}$, addition $+$, subtraction $-$, and that is closed under composition and discrete linear ODE schemata (respectively: scalar discrete linear ODE schemata)* LI.

By adding suitable towers of exponential, the above result can be generalized to characterize the various levels of the Grzegorczyk hierarchy. We can also reexpress Kleene's minimization:

▶ **Theorem 19** (Discrete ODE computability and classical computability are equivalent)**.** *A total function $f : \mathbb{N}^p \to \mathbb{N}$ is total recursive iff there exist some functions $h_1, h_2 : \mathbb{N}^{p+1} \to \mathbb{N}^2$ in the smallest set of functions that contains the zero functions $\mathbf{0}$, the projection functions π_i^p, the successor function $\mathbf{s}$, and that is closed under composition and discrete* ODE *schemata such that: for all $\mathbf{y}$, there exists some $T = T(\mathbf{y})$ with $h_2(T, \mathbf{y}) \neq 0$; $f(\mathbf{y}) = h_1(T, \mathbf{y})$ where T is the smallest such T.*

5 Restricted recursion and integration schemes

In order to talk about complexity instead of computability, we need to add some restrictions on the integration scheme. This follows from the following remark:

▶ **Example 20.** The solution of the (degree 2) polynomial ODE $y_1' = y_1$, $y_i' = y_{i-1}' y_i'$ for $i = 1, \dots d$ is growing faster than a tower of falling exponentials. Consequently, it is not polynomial time computable as just outputting its value in binary cannot be done in polynomial time.

We consequently propose to introduce the following variation on the notion of derivation: derivation along some function $\mathcal{L}(x, \mathbf{y})$.

▶ **Definition 21** ($\mathcal{L}$-ODE)**.** *Let $\mathcal{L} : \mathbb{N}^{p+1} \to \mathbb{Z}$. We write*

$$\frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \mathcal{L}} = \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \mathcal{L}(x, \mathbf{y})} = \mathbf{h}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}), \tag{4}$$

as 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 22. This is motivated by the fact that the latter expression is similar to classical formula for classical continuous ODEs:

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

This will allow us to simulate suitable changes of variables using this analogy. We will talk about $\mathcal{L}$-IVP when some initial condition is added. An important special case is when $\mathcal{L}(x, \mathbf{y})$ corresponds to the length $\mathcal{L}(x, \mathbf{y}) = \ell(x)$ function: we will call this special case length-ODEs.

5.1 General theory

The main result of this part illustrates one key property of the $\mathcal{L}$-ODE scheme from a computational point of view: its dependence on the number of distinct values of function $\mathcal{L}$.

▶ **Definition 23** ($Jump_{\mathcal{L}}$)**.** *Let $\mathcal{L} : \mathbb{N}^{p+1} \to \mathbb{Z}$ be some function. Fixing $\mathbf{y} \in \mathbb{N}^p$, let $Jump_{\mathcal{L}}(x, \mathbf{y}) = \{0 \leq i \leq x - 1 | \mathcal{L}(i+1, \mathbf{y}) \neq \mathcal{L}(i, \mathbf{y})\}$ be the set of positive integers less than x after which the value of $\mathcal{L}$ changes.*

We also write: $J_{\mathcal{L}} = |Jump_{\mathcal{L}}(x, \mathbf{y})|$ for its cardinality; $\alpha : [0..J_{\mathcal{L}} - 1] \to Jump_{\mathcal{L}}(x, \mathbf{y})$ for an increasing function enumerating the elements of $Jump_{\mathcal{L}}(x, \mathbf{y})$: If $i_0 < i_1 < i_2 < \cdots < i_{J_{\mathcal{L}}-1}$ denote all elements of $Jump_{\mathcal{L}}(x, \mathbf{y})$, then $\alpha(j) = i_j \in Jump_{\mathcal{L}}(x, \mathbf{y})$.

▶ **Lemma 24.** *Let $\mathbf{f} : \mathbb{N}^{p+1} \to \mathbb{Z}^d$ and $\mathcal{L} : \mathbb{N}^{p+1} \to \mathbb{Z}$ be some functions. Assume that* (4) *holds. Then $\mathbf{f}(x, \mathbf{y})$ is equal to:*

$$\mathbf{f}(0, \mathbf{y}) + \int_0^{J_{\mathcal{L}}} \Delta \mathcal{L}(\alpha(u), \mathbf{y}) \cdot \mathbf{h}(\mathbf{f}(\alpha(u), \mathbf{y}), \alpha(u), \mathbf{y}) \delta u.$$

▶ Remark 25. Note that the above result would still hold with $\mathcal{L}$ and h taking their images in $\mathbb{R}$.

Its proof is based on (and illustrates) some fundamental aspect of $\mathcal{L}$-ODE from their definition: for fixed $\mathbf{y}$, the value of $\mathbf{f}(x, \mathbf{y})$ only changes when the value of $\mathcal{L}(x, \mathbf{y})$ changes. If the previous hypotheses hold, there is then an alternative view to understand the integral, by using a change of variable, and by building a discrete ODE that mimics the computation of the integral.

▶ **Lemma 26** (Alternative view). *Let $k \in \mathbb{N}$, $f : \mathbb{N}^{p+1} \to \mathbb{Z}^d$, $\mathcal{L} : \mathbb{N}^{p+1} \to \mathbb{Z}$ be some functions and assume that (4) holds. Then $\mathbf{f}(x, \mathbf{y})$ is given by $\mathbf{f}(x, \mathbf{y}) = \mathbf{F}(\mathcal{L}(x, \mathbf{y}), \mathbf{y})$ where $\mathbf{F}$ is the solution of initial value problem*

$$\frac{\partial \mathbf{F}(t,\mathbf{y})}{\partial t} = \Delta\mathcal{L}(t, \mathbf{y}) \cdot \mathbf{h}(\mathbf{F}(t, \mathbf{y}), t, \mathbf{y}) \qquad \mathbf{F}(0, \mathbf{y}) = \mathbf{f}(\mathcal{L}(0, \mathbf{x}), \mathbf{y}).$$

Conversely, if there is such a function $\mathbf{F}$, then a discrete ODE of the type of (4) can easily be derived.

5.2 Linear length-ODEs

▶ Remark 27. In all previous reasonings, we considered that a function over the integers is polynomial time computable if it is in the length of all its arguments, as this is the usual convention. When not explicitly stated, this is our convention. As usual, we also say that some vectorial function (respectively: matrix) is polynomial time computable if all its components are. We need sometimes to consider also polynomial dependency directly in some of the variables and not on their length: This happens in the next fundamental lemma.

We write $\|\cdots\|$ for the sup norm: given some matrix $\mathbf{A} = (A_{i,j})_{1 \le i \le n, 1 \le j \le m}$, $\|A\| = \max_{i,j} A_{i,j}$.

▶ **Lemma 28** (Fundamental observation). *Consider the ODE*

$$\mathbf{f}'(x, \mathbf{y}) = \mathbf{A}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}) \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}). \tag{5}$$

Assume:

1. *The initial condition $\mathbf{G}(\mathbf{y}) =^{def} \mathbf{f}(0, \mathbf{y})$, as well as Matrix $\mathbf{A}$ and vector $\mathbf{B}$ are polynomial time computable.*
2. *$\ell(\|\mathbf{A}(f, x, \mathbf{y})\|) \le \ell(\|\mathbf{f}\|) + p_{\mathbf{A}}(x, \ell(\mathbf{y}))$ for some polynomial p_A*
3. *$\ell(\|\mathbf{B}(f, x, \mathbf{y})\|) \le \ell(\|\mathbf{f}\|) + p_{\mathbf{B}}(x, \ell(\mathbf{y}))$ for some polynomial p_B*

Then its solution $\mathbf{f}(x, \mathbf{y})$ is polynomial time computable in x and the length of $\mathbf{y}$.

Proof sketch. This comes from the explicit form of solutions provided by Lemma 9, considered then as an algorithm (see Remark 10). The point is then to analyse the size of all the involved quantities in order to state that this remains indeed polynomial time feasible. ◀

We now go to specific forms of linear ODEs.

▶ **Definition 29.** *A sg-polynomial expression $P(x_1, ..., x_h)$ is an expression built-on $+, -, \times$ (often denoted $\cdot$) and sg() functions over a set of variables $V = \{x_1, ..., x_h\}$ and integer constants. The degree $\deg(x, P)$ of a term $x \in V$ in P is defined inductively as follows:*

- $\deg(x, x) = 1$ *and for $x' \in X \cup \mathbb{Z}$ such that $x' \neq x$,* $\deg(x, x') = 0$
- $\deg(x, P + Q) = \max\{\deg(x, P), \deg(x, Q)\}$
- $\deg(x, P \times Q) = \deg(x, P) + \deg(x, Q)$
- $\deg(x, sg(P)) = 0$

A sg-polynomial expression P is essentially constant in x if $\deg(x, P) = 0$.

Compared to the classical notion of degree in polynomial expression, all subterms that are within the scope of a sign function contributes 0 to the degree. A vectorial function (resp. a matrix or a vector) is said to be a sg-polynomial expression if all its coordinates (resp. coefficients) are. It is said to be *essentially constant* if all its coefficients are.

▶ **Definition 30.** *A (possibly vectorial) sg-polynomial expression* $\mathbf{g}(\mathbf{f}(x,\mathbf{y}),x,\mathbf{y})$ *is essentially linear in* $\mathbf{f}(x,\mathbf{y})$ *if it is of the form* $\mathbf{g}(\mathbf{f}(x,\mathbf{y}),x,\mathbf{y}) = \mathbf{A}[\mathbf{f}(x,\mathbf{y}),x,\mathbf{y}]\cdot\mathbf{f}(x,\mathbf{y}) + \mathbf{B}[\mathbf{f}(x,\mathbf{y}),x,\mathbf{y}]$ *where* $\mathbf{A}$ *and* $\mathbf{B}$ *are sg-polynomial expressions essentially constant in* $\mathbf{f}(x,\mathbf{y})$.

▶ **Example 31.** The expression $P(x,y,z) = x\cdot\mathsf{sg}((x^2-z)\cdot y)+y^3$ is linear in x, essentially constant in z and not linear in y. The expression $P(x,2^{\ell(y)},z) = \mathsf{sg}(x^2-z)\cdot z^2 + 2^{\ell(y)}$ is essentially constant in x, essentially linear in $2^{\ell(y)}$ (but not essentially constant) and not essentially linear in z. The expression: $\mathsf{if}(x,y,z) = y+\bar{\mathsf{sg}}(x)\cdot(z-y) = y+(1-\mathsf{sg}(x))\cdot(z-y)$ is essentially constant in x and linear in y and z.

▶ **Definition 32.** *Function* $\mathbf{f}$ *is linear* $\mathcal{L}$*-ODE definable (from* $\mathbf{u}$, $\mathbf{g}$ *and* $\mathbf{h}$*) if it corresponds to the solution of* $\mathcal{L}$*-IVP*

$$\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \mathcal{L}} = \mathbf{u}(\mathbf{f}(x,\mathbf{y}),\mathbf{h}(x,\mathbf{y}),x,\mathbf{y}) \qquad f(0,\mathbf{y}) = \mathbf{g}(\mathbf{y}) \tag{6}$$

where $\mathbf{u}$ *is essentially linear in* $\mathbf{f}(x,\mathbf{y})$. *When* $\mathcal{L}(x,\mathbf{y}) = \ell(x)$, *such a system is called linear length-ODE.*

The previous statements lead to the following:

▶ **Lemma 33** (Key Observation for linear $\mathcal{L}$-ODE)**.** *Assume that* $\mathbf{f}$ *is the solution of* (6) *and that functions* $\mathbf{g},\mathbf{h},\mathcal{L}$ *and* $Jump_{\mathcal{L}}$ *are computable in polynomial time. Then,* $\mathbf{f}$ *is computable in polynomial time.*

6 A characterization of polynomial time

The above result shows that functions defined by linear length-ODE from functions computable in polynomial time, are indeed polynomial time. We are now ready to introduce a recursion scheme based on solving linear differential equations to capture polynomial time.

▶ Remark 34. Since the functions we define take their values in $\mathbb{N}$ and have output in $\mathbb{Z}$, composition is an issue. Instead of considering restrictions of these functions with output in $\mathbb{N}$ (which is always possible, even by syntactically expressible constraints), we simply admit that composition may not be defined in some cases. In other words, we consider that composition is a partial operator.

▶ **Definition 35** ($\mathbb{DL}$)**.** *Let* $\mathbb{DL}$ *be the smallest subset of functions, that contains* $\mathbf{0}$, $\mathbf{1}$, *projections* π_i^p, *the length function* $\ell(x)$, *the addition function* $x{+}y$, *the subtraction function* $x{-}y$, *the multiplication function* $x\times y$ *(often denoted* $x\cdot y$*), the sign function* $\mathsf{sg}(x)$ *and closed under composition (when defined) and linear length-ODE scheme.*

▶ Remark 36. As our proofs show, the definition of $\mathbb{DL}$ would remain the same by considering closure under any kind of $\mathcal{L}$-ODE with $\mathcal{L}$ satisfying the hypothesis of Lemma 33.

▶ **Example 37.** A number of natural functions are in $\mathbb{DL}$. Functions $2^{\ell(x)}$, $2^{\ell(x)\cdot\ell(y)}$, $\mathsf{if}(x,y,z)$, $\mathsf{suffix}(x,y)$, $\lfloor\sqrt{x}\rfloor$, $\lfloor\frac{x}{y}\rfloor$, $2^{\lfloor\sqrt{x}\rfloor}$ all belong to $\mathbb{DL}$ by some linear length-ODE easy to establish.

▶ **Theorem 38.** $\mathbb{DL} = \mathbf{FPTIME}$

sketch. $\boxed{\subseteq}$ Consequence of Lemma 33, on the fact that arithmetic operations that are allowed can be computed in polynomial time and that **FPTIME** is closed under composition.

$\boxed{\supseteq}$ A register machine program is a finite sequence of ordered labeled instructions acting on a finite set of registers R_0,...,R_k. Such a machine is equipped with assignment instructions ($R_j := R_j + R_i$, $R_j := R_j - R_i$, $R_j := i$ for $i \in \{0, 1\}$), control flow statement (if $R_j = 0$ goto p) and halt instruction.

Let $f : \mathbb{N}^p \longrightarrow \mathbb{N}$ be computable in polynomial time and M a k registers machine that computes f in time $\ell(\mathbf{x})^c$ for some $c \in \mathbb{N}$ (with $\mathbf{x}$ in registers $R_1, ..., R_p$, $p \leq k$, at startup and $f(\mathbf{x})$ in R_0 at the end). It is well known that polynomial time on register machines and on Turing machines is the same.

The computation of M is defined by simultaneous recursion scheme on length for functions $R_0(t, \mathbf{x}), ..., R_k(t, \mathbf{x})$ and $\mathsf{inst}(t, \mathbf{x})$: it gives, respectively, the values of each register and the label of the current instruction at time $\ell(t)$. It is shown to correspond to some length-linear ODE. As an example, we describe function $\mathsf{inst}(t, \mathbf{x})$ below.

Let next_l^I, next_l^h, $h \leq k$ describe the evolution of function inst and of register R_h between two instants of time after firing instruction with label l. If l is of the type $R_j := R_j - R_i$ (or is any assignment), then $\mathsf{next}_l^I = 1$ since $\mathsf{inst}(t + 1, \mathbf{x}) = \mathsf{inst}(t, \mathbf{x}) + 1$. If l if of the type if $R_j = 0$ goto p, $\mathsf{next}_l^I = \mathsf{if}(R_j(t, \mathbf{x}), p - \mathsf{inst}(t, \mathbf{x}), 1)$ since, in case $R_j(t, \mathbf{x}) = 0$ instruction number goes from $\mathsf{inst}(t, \mathbf{x})$ to p. The definition of each next_l^h is similar.

The value of $\mathsf{inst}(t + 1, \mathbf{x})$ can then be described from $\mathsf{inst}(t, \mathbf{x})$ and $R_h(t, \mathbf{x})$, $h \leq k$, by cases depending on the type of the current instruction. Expanded as an arithmetic expression, this gives:

$$\frac{\partial \mathsf{inst}}{\partial \ell}(t, \mathbf{x}) = \sum_{l=0}^{m} \Big(\prod_{i=0}^{l-1} \mathsf{sg}(\mathsf{inst}(t, \mathbf{x}) - i) \Big) \cdot \bar{\mathsf{sg}}(\mathsf{inst}(t, \mathbf{x}) - l) \cdot \mathsf{next}_l^I .$$

It is proved to be of the form

$$\frac{\partial R_j}{\partial \ell}(t, \mathbf{x}) = \sum_{l=0}^{m} \Big(\prod_{i=0}^{l-1} \mathsf{sg}(\mathsf{inst}(t, \mathbf{x}) - i) \Big) \cdot \bar{\mathsf{sg}}(\mathsf{inst}(t, \mathbf{x}) - l) \cdot \mathsf{next}_l^j$$

for some next_l^j encoding the various types of instructions.

The definition is similar (and easier) for each function $R_j(t, \mathbf{x})$. Observe that in the expression above, there is at most one occurrence of $\mathsf{inst}(t, \mathbf{x})$ and $R_j(t, \mathbf{x})$ that is not under the scope of an essentially constant function (i.e. the sign functions). Hence, the expressions are of the prescribed form.

We know M works in time $\ell(\mathbf{x})^c$ for some fixed $c \in \mathbb{N}$. Both functions $\ell(\mathbf{x}) = \ell(x_1) + \ldots + \ell(x_p)$ and $B(\mathbf{x}) = 2^{\ell(\mathbf{x}) \cdot \ell(\mathbf{x})}$ are in $\mathbb{DL}$. It is easily seen that : $\ell(\mathbf{x})^c \leq B^{(c)}(\ell(\mathbf{x})))$ where $B^{(c)}$ is the c-fold composition of function B. We conclude by setting $f(\mathbf{x}) = R_0(B^{(c)}(\max(\mathbf{x})), \mathbf{x})$. ◀

▶ **Definition 39** (Normal linear $\mathcal{L}$-ODE (N$\mathcal{L}$-ODE)). *Function* $\mathbf{f}$ *is definable by a normal linear* $\mathcal{L}$*-ODE if it corresponds to the solution of* $\mathcal{L}$*-IVP* $\frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \mathcal{L}} = \mathbf{u}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}), \quad \mathbf{f}(0, \mathbf{y}) = \mathbf{v}(\mathbf{y})$ *where* $\mathbf{u}$ *is essentially linear in* $\mathbf{f}(x, \mathbf{y})$ *and* $\mathbf{v}$ *is either the identity, a projection or a constant function.*

From the proof of Theorem 38 the result below can be easily obtained. It expresses that composition needs to be used only once as exemplified in the above definition.

▶ **Theorem 40** (Alternative characterization of **FPTIME**). *A function* $\mathbf{f} : \mathbb{N}^p \to \mathbb{Z}$ *is in* **FPTIME** *iff* $f(\mathbf{y}) = \mathbf{g}(\ell(\mathbf{y})^c, \mathbf{y})$ *for some integer* c *and some* $\mathbf{g} : \mathbb{N}^{p+1} \to \mathbb{Z}^k$ *solution of a normal linear length-ODE* $\frac{\partial \mathbf{g}(x, \mathbf{y})}{\partial \ell(x)} = \mathbf{u}(\mathbf{g}(x, \mathbf{y}), x, \mathbf{y})$.

▶ Remark 41. From similar arguments, **FPTIME** also corresponds to the class defined as $\mathbb{DL}$ but where linear length-ODE scheme is replaced by normal linear length-ode scheme: this forbids a function already defined with some ODE scheme to be used into some other ODE scheme.

7 A characterization of FNP

This can be extended to more general class such as **FNP**. A function can always seen by its graph Q: A relation Q is in **FNP** if and only if there is a deterministic time verifier, given an arbitrary input (x, y) determines whether $(x, y) \in Q$. Equivalently, Q is **FNP** if and only if there is a non deterministic polynomial time algorithm that, given an arbitrary input x, can find some y such that $(x, y) \in Q$. A function $w : \mathbb{N}^{p+1} \to \mathbb{Z}$ is bounded if for all $\mathbf{x} \in \mathbb{N}^p$, $w(\mathbf{x}) \leq \|\mathbf{x}\|$.

▶ **Definition 42** (Normal linear $\mathcal{L}$-ODE with parameter). *Let $\mathcal{L} : \mathbb{N}^{p+1} \to \mathbb{Z}$. Function $\mathbf{f} : \mathbb{N}^{p+1} \to \mathbb{Z}^k$ is definable by a $\mathcal{L}$-ODE with parameter if there exists a bounded function $w : \mathbb{N}^{p+1} \to \mathbb{Z}$ such that $\mathbf{f}$ is the solution of an equation $\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \mathcal{L}} = \mathbf{u}(\mathbf{f}(x, \mathbf{y}), w(x, \mathbf{y}), x, \mathbf{y})$ $\mathbf{f}(0, \mathbf{y}) = \mathbf{v}(\mathbf{y})$, where $\mathbf{u}$ is essentially linear in $\mathbf{f}(x, \mathbf{y})$ and $\mathbf{v}$ is either the identity, a projection or a constant function.*

▶ **Theorem 43** (Characterization of **FNP**). *A function $\mathbf{f} : \mathbb{N}^p \to \mathbb{N}^k$ is in **FNP** iff $f(\mathbf{y}) = \mathbf{g}(2^{\ell(\mathbf{y})^c}, \mathbf{y})$ for some integer c and some $\mathbf{g} : \mathbb{N}^{p+1} \to \mathbb{N}^k$, solution of a normal linear length-ODE with parameter $\frac{\partial \mathbf{g}(x,\mathbf{y})}{\partial \ell(x)} = \mathbf{u}(\mathbf{g}(x, \mathbf{y}), w(x, \mathbf{y}), x, \mathbf{y})$ for some bounded $w : \mathbb{N}^{p+1} \to \mathbb{N}$.*

8 Discussions and further works

Our aim in this article was to give the basis of a presentation of complexity theory based on discrete Ordinary Differential Equations and their basic properties. We demonstrated the particular role played by affine ordinary differential equations in complexity theory, as well as the concept of derivation along some particular function (i.e. change of variable) to guarantee a low complexity.

Previous ideas used here for **FPTIME** can also be extended to provide a characterization of other complexity classes: This includes the possibility of characterizing the class $\mathbf{P}_{[0,1]}$ of functions computable in polynomial time over the reals in the sense of computable analysis, or more general classes of classical discrete complexity theory such as **FPSPACE**. For the clarity of the current exposition, as this would require to introduce other types of schemata of ODEs, we leave this characterization (and improvement of the corresponding schemata to "the most natural and powerful form") for future work, but we believe the current article basically provides the key types of arguments to conceive that this is indeed possible.

More generally, it is also very instructive to revisit classical algorithmic under this viewpoint, and for example one may realize that even inside class **PTIME**, the Master Theorem (see e.g. [11, Theorem 4.1] for a formal statement) can be basically read as a result on (the growth of) a particular class of discrete time length ODEs. Several recursive algorithms can then be reexpressed as particular discrete ODEs of specific type.

References

1 V. I. Arnold. *Ordinary Differential Equations*. MIT Press, 1978.
2 S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity*, 2:97–110, 1992.

3 G. Birkhoff and G.-C. Rota. *Ordinary Differential Equations*. John Wiley & Sons, 4th edition, 1989.

4 O. Bournez, D. S. Graça, and A. Pouly. Polynomial Time corresponds to Solutions of Polynomial Ordinary Differential Equations of Polynomial Length. *Journal of the ACM*, 64(6):38:1–38:76, 2017. doi:10.1145/3127496.

5 Olivier Bournez and Amaury Pouly. A Survey on Analog Models of Computation. In Vasco Brattka and Peter Hertling, editors, *Handbook of Computability and Complexity in Analysis*. Springer. To appear, 2018.

6 Manuel L. Campagnolo. *Computational Complexity of Real Valued Recursive Functions and Analog Circuits*. PhD thesis, Universidade Técnica de Lisboa, 2001.

7 Manuel L. Campagnolo, Cristopher Moore, and José Félix Costa. An analog characterization of the Grzegorczyk hierarchy. *Journal of Complexity*, 18(4):977–1000, 2002.

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

9 E. A. Coddington and N. Levinson. *Theory of Ordinary Differential Equations*. Mc-Graw-Hill, 1955.

10 Pieter Collins and Daniel S Graça. Effective computability of solutions of ordinary differential equations the thousand monkeys approach. *Electronic Notes in Theoretical Computer Science*, 221:103–114, 2008.

11 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms (third edition)*. MIT press, 2009.

12 AO Gelfand. *Calcul des différences finies*. Dunod, 1963.

13 David Gleich. Finite calculus: A tutorial for solving nasty sums. *Stanford University*, 2005.

14 Ronald L Graham, Donald E Knuth, Oren Patashnik, and Stanley Liu. Concrete mathematics: a foundation for computer science. *Computers in Physics*, 3(5):106–107, 1989.

15 FA Izadi, N Aliev, and G Bagirov. *Discrete Calculus by Analogy*. Bentham Science Publishers, 2009.

16 Charles Jordan and Károly Jordán. *Calculus of finite differences*, volume 33. American Mathematical Soc., 1965.

17 L. Kalmár. Egyzzerü példa eldönthetetlen aritmetikai problémára. *Mate és Fizikai Lapok*, 50:1–23, 1943.

18 A. Kawamura. Lipschitz continuous ordinary differential equations are polynomial-space complete. In *2009 24th Annual IEEE Conference on Computational Complexity*, pages 149–160. IEEE, 2009.

19 Ker-I Ko. On the Computational Complexity of Ordinary Differential Equations. *Information and Control*, 58(1-3):157–194, 1983.

20 Gustavo Lau. Discrete calculus. URL: http://www.acm.ciens.ucv.ve/main/entrenamiento/material/DiscreteCalculus.pdf.

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

22 D. Leivant and J-Y Marion. Lambda Calculus Characterizations of Poly-Time. *Fundamenta Informatica*, 19(1,2):167,184, September 1993.

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

24 Bruno Loff, José Félix Costa, and Jerzy Mycka. The New Promise of Analog Computation. In *Computability in Europe 2007: Computation and Logic in the Real World.*, 2007.

25 Cristopher Moore. Recursion theory on the reals and continuous-time computation. *Theoretical Computer Science*, 162(1):23–44, August 1996.

26 Jerzy Mycka and José Félix Costa. What lies beyond the mountains? Computational systems beyond the Turing limit. *European Association for Theoretical Computer Science Bulletin*, 85:181–189, February 2005.

27 Jerzy Mycka and José Félix Costa. The $P \neq NP$ conjecture in the context of real and complex analysis. *Journal of Complexity*, 22(2):287–303, 2006.

28 P. Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and the foundations of mathematics*. North-Holland, April 1992.

29 Amaury Pouly. *Continuous models of computation: from computability to complexity*. PhD thesis, Ecole Polytechnique and Unidersidade Do Algarve, 2015. https://pastel.archives-ouvertes.fr/tel-01223284, Ackermann Award 2017.

30 H. E. Rose. *Subrecursion, Functions and Hierarchies*. Clarendon Press, Oxford, 1984.