Programming with Ordinary Differential Equations: Some First Steps Towards a Programming Language

Olivier Bournez

Programming with Ordinary Differential Equations: Some First Steps Towards a Programming Language

Olivier Bournez

Ecole Polytechnique, LIX, 91128 Palaiseau Cedex, France

Abstract. Various open problems have been recently solved using Ordinary Differential Equation (ODE) programming: basically, ODEs are used to implement various algorithms, including simulation over the continuum of discrete models such as Turing machines, or simulation of discrete time algorithms working over continuous variables. Applications include: Characterization of computability and complexity classes using ODEs [1–4]; Proof of the existence of a universal (in the sense of Rubel) ODE [5]; Proof of the strong Turing completeness of biochemical reactions [6], or more generally various statements about the completeness of reachability problems (e.g. PTIME-completeness of bounded reachability) for ODEs [7].
It is rather pleasant to explain how this ODE programming technology can be used in many contexts, as ODEs are in practice a kind of universal language used by many experimental sciences, and how consequently we got to these various applications.
However, when going to say more about proofs, their authors including ourselves, often feel frustrated: Currently, the proofs are mostly based on technical lemmas and constructions done with ODEs, often mixing both the ideas behind these constructions, with numerical analysis considerations about errors and error propagation in the equations. We believe this is one factor hampering a more widespread use of this technology in other contexts.
The current article is born from an attempt to popularize this ODE programming technology to a more general public, and in particular master and even undergraduate students. We show how some constructions can be reformulated using some notations, that can be seen as a pseudo programming language. This provides a way to explain in an easier and modular way the main intuitions behind some of the constructions, focusing on the algorithm design part. We focus here, as an example, on how the proof of the universality of polynomial ODEs (a result due to [8], and fully developed in [2]) can be reformulated and presented.

1 Introduction

It has been understood quite recently that it is possible to program with Ordinary Differential Equations (ODEs). This actually was obtained as a side effect from attempts to relate the computational power of analog computational

models to classical computability. Refer to [9–11] for surveys on analog computation with a point of view based on computation theory aspects, or to [12–14] for surveys discussing historical and technological aspects. In particular, several authors revisited the *General Purpose Analog Computer* (GPAC) model of Shannon [15]. Following [16], this model can essentially be abstracted as corresponding to (vectorial) polynomial ordinary differential equations: That is to say, as dynamics over $\mathbb{R}^d$ corresponding to solutions of ODEs of the form $\mathbf{y}' = \mathbf{p}(\mathbf{y})$ where $\mathbf{y}(t) \in \mathbb{R}^d$ is some function of time, and $\mathbf{p} : \mathbb{R}^d \to \mathbb{R}^d$ is (componentwise) polynomial, and d is some integer. If some initial condition is added, this is also called a polynomial Initial Value Problems (pIVP).

We do not intend to repeat here the full story, but in short, two main notions of computations by pIVP have been introduced: the notion of GPAC-generated function, corresponding to the initial notion from Shannon in [15], and the notion of GPAC-computable function. This latter notion of computability is now known to be equivalent to classical computability [17]. It is also possible to talk about complexity theory in such models: It has been established that this is indeed possible, if measuring time of computation as the length of the solution [7]. This has been recently extended to space complexity in [18], or to exponential time and the Grzegorczyk hierarchy[19].

All these statements have been obtained by realizing that continuous time processes defined by ODEs, and even defined by polynomial ODEs, can simulate various discrete time processes. They hence can be used to simulate models such as Turing machines [20, 2], and even some more exotic models working with a discrete time but over continuous data. This is based on various refinements of constructions done in [20, 2, 1, 3, 4]. We call this *ODE programming*, as this is indeed some kind of programing with various continuous constructions.

Forgetting analog machines or models of computation, it is important to realize that ODEs is a kind of universal language of mathematics that is used in many, if not all, experimental sciences: Physics, Biology, Chemistry, ... Consequently, once it is known that one can program with ODEs, many questions asking about universality, or computations, in experimental contexts can be solved. This is exactly what has been done by several authors, including ourselves, to solve various open problems, in various contexts such as applied maths, computer algebra, biocomputing... We do not intend here to be exhaustive about various applications, but we describe some of them.

Some applications of ODE programming:

– **Implicit complexity: computability:** Concepts such as being computable for a function over the reals (in the sense of computable analysis) can be described using ODEs only, i.e., with concepts from analysis only, and no-reference to computational models such as Turing machines.

Theorem 1 ([17]). *Let a and b be computable reals. A function $f : [a, b] \to \mathbb{R}$ is computable in the sense of computable analysis iff there exist some polynomials with rational coefficients $\mathbf{p} : \mathbb{R}^{n+1} \to \mathbb{R}^n$, some polynomial p_0 :*

$\mathbb{R} \to \mathbb{R}$ *with rational coefficients, and* $n-1$ *computable real values* $\alpha_1, ..., \alpha_{n-1}$ *such that:*

1. $(y_1, ..., y_n)$ *is the solution of the initial value problem*[1] $\mathbf{y}' = \mathbf{p}(\mathbf{y}, t)$ *with initial condition* $(\alpha_1, ..., \alpha_{n-1}, p_0(x))$ *set at time* $t_0 = 0$
2. *There are* $i, j \in \{1, ..., n\}$ *such that* $\lim_{t\to\infty} y_j(t) = 0$ *and* $|f(x) - y_i(t)| \leq y_j(t)$ *for all* $x \in [a, b]$ *and all* $t \in [0, +\infty)$.

Condition 2. basically says that some component, namely y_i is converging over time t toward $f(x)$, with error given by some other component, namely $y_j(t)$.

- **Robust complexity theory for continuous-time systems:** Defining a robust time complexity notion for continuous time systems was a well-known open problem [10], with several attempts, but with no generic solution provided. In short, the difficulty is that the naive idea of using the time variable of the ODE as a measure of "time complexity" is problematic, since time can be arbitrarily contracted in a continuous system due to the "Zeno phenomena". This was solved by establishing that the length of the solutions for pODEs provides a robust complexity, that corresponds to classical complexity [7]. This also provides some implicit characterization of PTIME, and completeness of bounded time reachability.
- **Characterization of other computability and complexity classes:** This has been extended very recently to space complexity, with a characterization of PSPACE in [18], and of EXPTIME in [19], and to the Grzegorczyk hierarchy [4].
- **A universal ordinary differential equation:** Following [21], there exists a fixed non-trivial fourth-order polynomial differential algebraic equation (DAE) $p(y, y', \ldots, y^d) = 0$ such that for any continuous positive function φ on the reals, and for any continuous positive function $\epsilon(t)$, it has a $\mathcal{C}^\infty$ solution with $|y(t) - \varphi(t)| < \epsilon(t)$ for all t. The question whether one can require the solution that approximates φ to be the unique solution for a given initial data was a well-known open problem [21, page 2], [22, Conjecture 6.2]. It has been solved using ODE programming.

  Theorem 2 ([5], Universal PIVP). *There exists a* fixed *polynomial vector* $\mathbf{p}$ *in* d *variables with rational coefficients such that for any functions* $f \in C^0(\mathbb{R})$ *and* $\varepsilon \in C^0(\mathbb{R}, \mathbb{R}^{>0})$*, there exists* $\alpha \in \mathbb{R}^d$ *such that there exists a unique solution* $\mathbf{y} : \mathbb{R} \to \mathbb{R}^d$ *to* $\mathbf{y}(0) = \alpha$*,* $\mathbf{y}' = \mathbf{p}(\mathbf{y})$*. Furthermore, this solution satisfies that* $|y_1(t) - f(t)| \leqslant \varepsilon(t)$ *for all* $t \in \mathbb{R}$*.*

- **Strong Turing completeness of biochemical reactions:** Ordinary differential equations are a well-used models for modeling the dynamics of the kinetic of reactions, and in particular of biochemical reactions between proteins. Their Turing completeness was an open problem (see e.g. [23, Section 8]) solved in [6] using ODE programming.

[1] We suppose that $y(t)$ is defined for all $t \geq 0$. This condition is not necessarily satisfied for all polynomial ODEs, and we restrict our attention only to ODEs satisfying this condition.

However, one difficulty when one wants to present ideas of the proofs, is that currently the proofs are based on technical lemmas and constructions done with ODEs, often mixing both the ideas behind these constructions, with numerical analysis considerations about errors and error propagation in the equations. We believe this is one factor hampering a more widespread use of this technology in other contexts. This is also clearly a difficulty for newcomers in the field.

The current document is a preliminary step towards solving this, by presenting through an example a pseudo-programming language. Actually, this document follows from an attempt to popularize this ODE programming technology to a more general public, and in particular master and even undergraduate students. We show how some constructions can be reformulated using some notations, that can be seen as a pseudo programming language. This provides a way to explain in an easier and modular way the main intuitions behind some of the constructions, focusing on the algorithm design part.

From our experiments, it can be done more generally for all of the constructions of the references above. By lack of space, we only take an example, namely the main result of [8], fully developed in [2], establishing the universality of ODEs. We agree that for experts, this might be, or it is, at the end the same proofs, but we believe, that presented that way, with this pseudo-programming language the intuition is easier to grasp for newcomers. One motivation of popularizing ODE programming is that this may then help to solve other various problems in particular in experimental sciences.

We also believe that this example is good to make our reader feel what ODE programming is at the end, and the kind of techniques that are used in all the mentioned references to establish all these results.

2 Computing with pIVPs

Proving a result such as Theorem 1, is done in one direction by simulating some Turing machine using some pIVP. The purpose in this section is to provide the intuition on how this is indeed possible to do so. We believe this provides a good intuition on how this is possible to program with ODEs, and more generally solve various discrete problems in some continuous way.

Remark 1. The other direction of the theorem, that we will not discuss in this article, is obtained by proving that one can solve the involved ordinary differential equations by some numerical method, and hence a Turing machine. Notice that this is not as obvious as it may seem: All the ordinary differential equations cannot be solved easily (see famous counterexample [24]), and we use the fact that we restrict to some particular (polynomial) ordinary differential equations: See for example [25] for some conditions and methods to guarantee effectivity for more general ordinary differential equations.

Remark 2. Notice that our example of simulation of a Turing machine by an ordinary differential equation may be however misleading, as it may be thought that, as Turing machines are universal for classical computability, this is the

end of the story. But, actually some results (e.g. Theorem 2) are obtained by simulating a discrete time model not directly covered by classical computability, as working over continuous variables. Understanding the suitable underlying models is a fascinating question from our point of view, which remains to be fully explored and understood.

2.1 Discrete time computations

Encoding a Turing machine configuration with a triplet Consider without loss of generality some Turing machine M using the ten symbols $0, 1, \ldots, 9$, where $B = 0$ is the blank symbol. Let $\ldots BBBa_{-k}a_{-k+1}\ldots a_{-1}a_0a_1\ldots a_nBBB\ldots$ denotes the content of the tape of the Turing machine M. In this representation, the head is in front of symbol a_0, and $a_i \in \{1, \ldots, 9\}$ for all i. Suppose that M has m internal states, corresponding to integers 1 to m. Such a configuration C can be encoded by some element $\gamma(C) = (y_1, y_2, q) \in \mathbb{N}^3$, by taking

$$\begin{aligned} y_1 &= a_0 + a_1 10 + \cdots + a_n 10^n, \\ y_2 &= a_{-1} + a_{-2} 10 + \cdots + a_{-k} 10^{k-1}, \end{aligned}$$

and where q denotes the internal state of M.

Let $\theta : \mathbb{N}^3 \to \mathbb{N}^3$ be the transition function of M: Function θ maps the encoding $\gamma(C)$ of a configuration C to the encoding $\gamma(C_{next})$ of its successor configuration.

Determining next configuration Next step is to observe that one can construct some function $\mathbf{f} : \mathbb{R}^3 \to \mathbb{R}^3$ that realizes some analytic extension of θ: i.e. that coincides with θ on $\mathbb{N}^3$. To do so, the problems to solve are the following:

1. **Determine the symbol being read.** The symbol a_0 in front of the head of the machine is given by $a_0 = \mod_{10}(y_1)$, where $\mod_{10}(n)$ is the remainder of the division of integer n by 10. Since we want to consider some analytic functions, we can instead consider

$$a_0 = \omega(y_1), \tag{1}$$

where ω is some analytic extension of $\mod_{10}$. For example, it can be taken of the form

$$\omega(x) = a_0 + a_5 \cos(\pi x) + \left(\sum_{j=1}^{4} a_j \cos\left(\frac{j\pi x}{5}\right) + b_j \sin\left(\frac{j\pi x}{5}\right) \right), \tag{2}$$

where $a_0, \ldots, a_4, b_1, \ldots, b_4$ are some (computable) coefficients that can be obtained by solving some system of linear equations: write $\omega(i) = i$, for $i = 0, 1, ..., 9$.

2. **Determine the next state.** The function that returns the next state can be defined by a Lagrange interpolation polynomial as follows: Let $y = \omega(y_1) = a_0$ be the symbol being currently read and q the current state. Take

$$q_{next} = \sum_{i=0}^{9} \sum_{j=1}^{m} \left(\prod_{r=0, r\neq i}^{9} \frac{(y-r)}{(i-r)} \right) \left(\prod_{s=1, s\neq j}^{m} \frac{(q-s)}{(j-s)} \right) q_{i,j}, \tag{3}$$

   where $q_{i,j}$ is the state that follows symbol i and state j.
3. **Determine the symbol to be written on the tape.** The symbol to be written, s_{next}, can be obtained by some similar interpolation polynomial.
4. **Determine the direction of the move for the head.** Let h denote the direction of the move of the head, where $h = 0$ denotes a move to the left, $h = 1$ denotes a "no move", and $h = 2$ denotes a move to the right. Then, again, the "next move" h_{next} can be obtained by some interpolation polynomial.
5. **Update the tape contents.** Define functions P_1, P_2, P_3, that provides the tape contents after the head moves left, does not move, or moves right, respectively. Then, the next value of y , denoted by y_1^{next}, can be obtained by

$$y_1^{next} = P_1 \frac{(1-H)(2-H)}{2} + P_2 H(2-H) + P_3 \frac{H(H-1)}{2}, \tag{4}$$

   With $P_1 = 10\,(y_1 + s_{next} - y) + \omega\,(y_2)$, $P_2 = y_1 + s_{next} - y$ and $P_3 = \frac{y_1 - y}{10}$, where $H = h$ is the direction of the move given by previous item, and still $y = \omega(y_1) = a_0$. We can do something similar to get y_2^{next}.

Consequently, it is sufficient to take $\mathbf{f}(y_1, y_2, q) = (y_1^{next}, y_2^{next}, q_{next})$. It follows that if one succeeds to repeat in a loop the instruction

$$\mathbf{x} \leftarrow \mathbf{f}(\mathbf{x}),$$

where $\leftarrow$ denotes an assignment, that is to say replacing the value of $\mathbf{x}$ by $\mathbf{f}(\mathbf{x})$, one will simulate Turing machine M: starting from $\mathbf{x}(0) = \gamma(C_0)$, encoding the initial configuration of the Turing machine M, then $\mathbf{x}(t)$ will be $\gamma(C(t))$, where $C(t)$ is the configuration of the machine at time t.

2.2 Computations with a continuous time

Write $instruction_1; instruction_2$ to denote the fact of doing first $instruction_1$ and then $instruction_2$. Actually, if we succeed to repeat in a loop $\mathbf{x}_2 \leftarrow \mathbf{f}(\mathbf{x})\,; \mathbf{x} \leftarrow \mathbf{x}_2$ that would be sufficient: We will do the same, but two times slower, in the sense that starting from $\mathbf{x}(0) = \gamma(C_0)$, then now $\mathbf{x}(2t)$ will be $\gamma(C(t))$. If we want to preserve speed, it would be sufficient to repeat in a loop

$$\mathbf{x}_2 \leftarrow_{1/2} \mathbf{f}(\mathbf{x}); \mathbf{x} \leftarrow_{1/2} \mathbf{x}_2 \tag{5}$$

when $\mathbf{x} \leftarrow_{1/2} \mathbf{y}$ means an assignment done in time 1/2, i.e. doing $\mathbf{x}(t+\frac{1}{2}) = \mathbf{y}(t)$ if executed at time t . That way, $\mathbf{x}(t)$ will still be $\gamma(C(t))$.

To do so with ODEs, one needs to be able to do this operation $\leftarrow_{1/2}$. A construction, due to Branicky in [20], is based on the following remark: One can construct some ODE that does some assignment, and even this particular assignment. More precisely, if one takes some ODE of the form

$$\mathbf{y}' = c(\mathbf{g} - \mathbf{y})^3 \phi(t), \tag{6}$$

where c is some real constant, one can check by some simple arguments from analysis, that whatever the value of $\mathbf{y}(0)$ is, the solution will converge very fast to $\mathbf{g}$. Even uniformly, in the sense that for any function $\phi(t)$ of positive integral, for any precision $\epsilon > 0$, real constant $c > 0$ can be fixed sufficient big, such that *for any* $\mathbf{y}(0)$*, it is certain that* $\mathbf{y}(\frac{1}{2})$ *is at a distance less than* ϵ *of* $\mathbf{g}$. In other words, looking what is written *in italic like this*, this essentially means *realizing the equivalent of assignment* $\mathbf{y} \leftarrow_{1/2} \mathbf{g}$ (possibly with some error, but less than ϵ).

Remark 3 (Notation). In order to help, we use the following notation: we write $\boxed{\mathbf{y} \leftarrow_{1/2} \mathbf{g}}_{\epsilon}{}'$ for $c(\mathbf{g} - \mathbf{y})^3 \phi(t)$ with the real constant c associated to that ϵ. Consequently, writing ODE

$$\mathbf{y}' = \boxed{\mathbf{y} \leftarrow_{1/2} \mathbf{g}}_{\epsilon}{}'$$

is the same as dynamics (6), i.e. some ODE doing $\mathbf{y} \leftarrow_{1/2} \mathbf{g}$ with an error less than ϵ.

The intuition about these notations is that $\boxed{\textit{operation}}$ is about a function doing some exact operation, whereas $\boxed{\textit{operation}}$ is about some function that is intending to do something similar, but possibly introducing some errors.

Once we understand this, we can solve our problem: Consider a function $\mathbf{r}$ that does some rounding componentwise, say on every component $r(x) = j$ for $x \in [j - 1/4, j + 1/4]$, for $j \in \mathbb{Z}$. We will use some function $\theta(u)$ that we will also write $\boxed{\textit{when } u \geq 0}$. Observing that $\sin(2\pi t)$ is alternatively positive and negative, and of period 1, we consider dynamic:

$$\begin{cases} \mathbf{x}' = \boxed{\textit{when } -\sin(2\pi t) \geq 0} \cdot \boxed{\mathbf{x} \leftarrow_{1/2} \mathbf{r}(\mathbf{x}_2)}_{1/4}{}' \\ \mathbf{x}_2' = \boxed{\textit{when } \sin(2\pi t) \geq 0} \cdot \boxed{\mathbf{x}_2 \leftarrow_{1/2} \boxed{\mathbf{f}}(\mathbf{r}(\mathbf{x}))}_{1/4}{}' \end{cases} \tag{7}$$

with $\mathbf{x}_1(0) = \mathbf{x}_2(0) = \mathbf{x}_0$.

Written in another way, this is the dynamic:

$$\begin{cases} \mathbf{x}' = c_1(\mathbf{r}(\mathbf{x}_2) - \mathbf{x})^3 \theta(-\sin(2\pi t)) \\ \mathbf{x}_2' = c_2(\boxed{\mathbf{f}}(\mathbf{r}(\mathbf{x})) - \mathbf{x}_2)^3 \theta(\sin(2\pi t)) \end{cases} \tag{8}$$

We select the function $\theta(x) = \boxed{\textit{when } x \geq 0}$ so that it is 0 for $x \leq 0$, and positive for $x > 0$ (say x^2 for $x > 0$). The idea is that $\sin(2\pi t)$ is alternatively positive for $t \in [j, j + 1/2]$, then negative for $t \in [j + 1/2, j]$ when t increases, where t is some integer. Consequently, $\theta(\sin(2\pi t))$ is alternatively positive and null. Consequently, $\mathbf{x}_2'$ alternates between the right hand side of ODE (6) with $\mathbf{g} = \boxed{\mathbf{f}}(\mathbf{r}(\mathbf{x}))$ (for some function ϕ) and exactly 0. In other words, alternatively $\mathbf{x}_2$ evolves in order to do $\mathbf{x}_2 \leftarrow_{1/2} \boxed{\mathbf{f}}(\mathbf{r}(\mathbf{x}))$, then stay fixed, and this process is repeated for ever.

In a symmetric way, $\mathbf{x}'$ alternates between exactly 0 and exactly the right hand side of ODE (6) with $\mathbf{g} = \mathbf{r}(\mathbf{x}_2)$ (for some function ϕ). In other words, $\mathbf{x}$ evolves between instants where it is fixed, and where one does $\mathbf{x} \leftarrow_{1/2} \mathbf{r}(\mathbf{x}_2)$).

Clearly, if one starts from a point $\mathbf{x}_0$ with integer coefficients, and since $\boxed{\mathbf{f}}$ preserves the integers, this will do exactly what intended, that is to say repeat in a loop (5): In order to get this working, it is sufficient to consider $\epsilon < 1/4$ and select c_1 and c_2 sufficiently big, by the property of the ODE (6) above.

Indeed, each time t multiple of $1/2$ is starting to do some assignment of type $\mathbf{y} \leftarrow_{1/2} \mathbf{r}(\mathbf{g})$. Actually, this does not do this exactly: $\mathbf{y}$ is not set to the precise value $\mathbf{r}(\mathbf{g})$ at next multiple of $\frac{1}{2}$, but with these hypotheses, we (by a simple induction) are always in the case where we know that $\mathbf{r}(\mathbf{g})$ has integer coordinates, and since $\epsilon < 1/4$, $\mathbf{r}(\mathbf{y})$ will indeed be the correct integer after time $\frac{1}{2}$. In other words, by recurrence, if we consider $\mathbf{r}(\mathbf{x})$ and $\mathbf{r}(\mathbf{x}_2)$ at some time multiple of $\frac{1}{2}$, this does exactly the same as repeating in a loop (5).

This works perfectly fine, and provides a way to simulate some Turing machine by some ODE, and more generally the iterations of some functions over the integers.

However, we want a stronger property: we want to simulate some Turing machine by some *analytic* dynamic. We know from the theory of analytic functions, that some analytic function that is constant in some interval is necessarily constant. Consequently, our functions θ and $\mathbf{r}$ above cannot be analytic.

The idea is then to replace ideal $\boxed{\textit{when } u \geq 0}$ by $\boxed{\textit{when } u \geq 0}_{\epsilon'}$ that would approximate the first.

2.3 Some useful functions to correct errors

A function such as $\sigma(x) = x - 0.2\sin(2\pi x)$ is a contraction on the vicinity of integers:

Lemma 1. *Let $n \in \mathbb{Z}$, and let $\epsilon \in [0, 1/2)$. Then there is some contracting factor $\lambda_\epsilon \in (0, 1)$ such that $\forall \delta \in [-\epsilon, \epsilon]$, $|\sigma(n + \delta) - n| < \lambda_\epsilon \delta$.*

This function can be used to do some static error correction. To help readability, we will also write $\boxed{x}$ for $\sigma(x)$. We will also write $\boxed{x}_{[n]}$ for $\sigma^{[n]}(x)$, i.e. n fold composition of function σ. We do so, as when x is close to some integer,

these values are basically close to x: Just ignore rounded corners in our notations for intuition.

It is also possible to do some dynamic error correction: We construct a function $\hookrightarrow\{0,1\}$. It takes two arguments, $\overline{a}$ and y, and its value, that we will write $\overline{a}\hookrightarrow\{0,1\}_y$, values a with error at most $1/y$ when $\overline{a}$ is close to a, with $a\in\{0,1\}$ and we have some positive $y>0$. Formally:

Lemma 2 ([2, Lemma 4.2.5]). *One can create* $\hookrightarrow\{0,1\}:\mathbb{R}^2\to\mathbb{R}$ *such that for all* $y>0$*, and for all* $\overline{a}\in\mathbb{R}$*, as soon as* $|a-\overline{a}|\leq 1/4$ *with* $a\in\{0,1\}$*, then* $\overline{a}\hookrightarrow\{0,1\}_y\in\left]a-\frac{1}{y},a+\frac{1}{y}\right[$.

Proof. Consider $x\hookrightarrow\{0,1\}_y=\frac{1}{\pi}\arctan(4y(x-1/2))+\frac{1}{2}$, and observe that $\left|\frac{\pi}{2}-\arctan x\right|<\frac{1}{x}$ for $x\in(0,\infty)$ and $\left|\frac{\pi}{2}+\arctan x\right|<\frac{1}{|x|}$ for $x\in(-\infty,0)$.

In $x\hookrightarrow\{0,1\}_y$, the argument x is expected to take essentially only two values. We can construct a version with 3 values, namely 0, 1 and 2.

Lemma 3 ([2, Lemma 4.2.7]). *Fix* $\epsilon>0$*. One can create* $\hookrightarrow\{0,1,2\}:\mathbb{R}^2\to\mathbb{R}$ *such that for all* $y\geq 2$*, and for all* $\overline{a}\in\mathbb{R}$*, as soon as* $|a-\overline{a}|\leq\epsilon$ *with* $a\in\{0,1,2\}$*, then* $\overline{a}\hookrightarrow\{0,1,2\}_y\in\left]a-\frac{1}{y},a+\frac{1}{y}\right[$.

Proof. Take $\left(\left(x\right)_{[d+1]}-1\right)^2\hookrightarrow\{0,1\}_{3y}\cdot\left(2\cdot\left(x\right)_{[d]}/2\hookrightarrow\{0,1\}_{3y}-1\right)+1$, where $d=0$ if $\epsilon\leq 1/4$ and $d=\lceil-\log(4\epsilon)/\log\lambda_\epsilon\rceil$ otherwise.

2.4 And hence, how to so with analytic functions?

We would like to do some dynamic close to the one of (7), but using only analytic functions. We intend to replace $\phi(t)=\theta(\sin 2\pi t)=$ *when* $\sin 2\pi t\geq 0$ in the dynamic $\zeta(t)$, by some analytic function $\zeta:\mathbb{R}\to\mathbb{R}$.

An idea to get such a function is to consider $\zeta_\epsilon(t)=\vartheta(t)\hookrightarrow\{0,1\}_{1/\epsilon}$, with $\vartheta(t)=\frac{1}{2}\left(\sin^2(2\pi t)+\sin(2\pi t)\right)$. Using the notation $\textit{when } u\geq 0_{1/\epsilon}=u\hookrightarrow\{0,1\}_{1/\epsilon}$, we can also write $\zeta_\epsilon(t)=\textit{when } \vartheta(t)\geq 0_{1/\epsilon}$.

We would then like to replace dynamic (7) by something like

$$\begin{cases}\mathbf{x}'=\textit{when } \vartheta(-t)\geq 0_{1/\epsilon}\cdot\left(\mathbf{x}\leftarrow_{1/2}\mathbf{r}(\mathbf{x}_2)\right)_\epsilon'\\ \mathbf{x}_2'=\textit{when } \vartheta(t)\geq 0_{1/\epsilon}\cdot\left(\mathbf{x}_2\leftarrow_{1/2}\mathbf{f}(\mathbf{r}(\mathbf{x}))\right)_\epsilon'\end{cases}\tag{9}$$

We however still need to get rid of functions $\mathbf{r}(r)$ doing some exact rounding, that cannot be analytic. The ideas is to consider that $\sigma^{[n]}(x)=\left(x\right)_{[n]}$ does the

job, if integer n is big enough, and if $\mathbf{f}$ commits an error that remains small and uniform, in the vicinity of integers.

How to control errors on f? We basically replace $\mathbf{f}$ by some function $\mathbf{f}$ that is built exactly with the same ideas, but keeping errors under control.

Theorem 3 ([2, Theorem 4.4.1]). *Let $\theta : \mathbb{N}^3 \to \mathbb{N}^3$ be the transition function of some Turing machine. Then, given $0 \leq \epsilon < 1/2$, θ has some analytic expansion $\mathbf{f} : \mathbb{R}^3 \to \mathbb{R}^3$ such that*

$$\|(y_1, y_2, q) - (\overline{y}_1, \overline{y}_2, \overline{q})\| \leq \epsilon \Rightarrow \left\| \theta(y_1, y_2, q) - \mathbf{f}(\overline{y}_1, \overline{y}_2, \overline{q}) \right\| \leq \epsilon$$

where $(y_1, y_2, q) \in \mathbb{N}^3$ encodes a configuration of M.

Proof. To get so, using previous ideas, replace equation (1), by $a_0 = \overline{y} = \omega\left(\overline{y_1}_{[l]}\right)$, and the polynomial interpolations such as equation (3) to determine the next state, the next symbol and the direction of the move, by the equivalent expression where each variable has been "rounded cornered". For example, instead of (3), write

$$q_{next} = \sum_{i=0}^{9} \sum_{j=1}^{m} \left(\prod_{r=0, r\neq i}^{9} \frac{\left(\overline{y}_{[n]} - r\right)}{(i-r)} \right) \left(\prod_{s=1, s\neq j}^{m} \frac{(\overline{q}_{[n]} - s)}{(j-s)} \right) q_{i,j},$$

If l and n are chosen sufficiently big, the approximation of the interpolation will be uniform, and as small as desired.

The difficulty is on equation (4), since the error is not uniform if this is done in a too naive way. But actually, instead of taking $H = h$, the idea is to take some dynamic approximation sufficiently precise to correct errors.

More concretely: We still intend to define some functions $\overline{P}_1$, $\overline{P}_2$, $\overline{P}_3$, which intend to approximate the content of the tapes if the head respectively move left, don't move or move right. Let H some "sufficiently big" approximation of h, still to be determined. Then y_1^{next} can be approximated by

$$\overline{y}_1^{next} = \overline{P}_1 \frac{1}{2}(1-H)(2-H) + \overline{P}_2 H(2-H) + \overline{P}_3(-\frac{1}{2})H(1-H), \qquad (10)$$

With $\overline{P}_1 = 10\left(\overline{y_1}_{[j]} + \overline{s_{next}}_{[j]} - \overline{y}_{[j]}\right) + \overline{\omega\left(\overline{y_2}_{[j]}\right)}_{[j]}$, $\overline{P}_2 = \overline{y_1}_{[j]} + \overline{s_{next}}_{[j]} - \overline{y}_{[j]}$, and $\overline{P}_3 = \frac{\overline{y_1}_{[j]} - \overline{y}_{[j]}}{10}$.

This is at the end, once again exactly similar to expressions in (4), but where all variables have been "rounded cornered".

The difficulty is that in that case, $\overline{P}_1$ depends on $\overline{y}_1$, which is not a bounded value. So if we take as before $\overline{H} = h = \overline{h}_{next}$, the error on term $(1-H)(2-H)/2$ can be arbitrarily amplified when this term is multiplied by $\overline{P}_1$. But one can take some error proportional to $\overline{y}_1$. One can then check that $H = \boxed{\overline{h}_3 \hookrightarrow \{0,1,2\}}_{10000(\overline{y}_1+1/2)+2}$ is fine.

Use same arguments on $\overline{P}_2$ and $\overline{P}_3$ to define $|\overline{y}_1^{next} - y_1^{next}| < \epsilon$. And does something similar for the other part of the tape, to define $\overline{y}_2^{next}$ such that $|\overline{y}_2^{next} - y_2^{next}| < \epsilon$.

At the end, consider $\boxed{\mathbf{f}} : \mathbb{R}^3 \to \mathbb{R}^3$ defined by

$$\boxed{\mathbf{f}}(\overline{y}_1, \overline{y}_2, \bar{q}) = (\overline{y}_1^{next}, \overline{y}_2^{next}, \bar{q}_{next}).$$

Coming back to the simulation So at the end, we have replaced the dynamic of (7) by a dynamic of the form:

$$\begin{cases} \mathbf{x}' = \boxed{when\ \vartheta(-t) \geq 0}_{1/\epsilon} \cdot \boxed{\mathbf{x} \leftarrow_{1/2} \boxed{\mathbf{x}_2}_{[n]}}_{\epsilon'}{}' \\ \mathbf{x}_2' = \boxed{when\ \vartheta(t) \geq 0}_{1/\epsilon} \cdot \boxed{\mathbf{x}_2 \leftarrow_{1/2} \boxed{\mathbf{f}}\left(\boxed{\mathbf{x}}_{[m]}\right)}_{\epsilon'}{}' \end{cases} \tag{11}$$

with $\boxed{when\ u \geq 0}_{1/\epsilon} = \boxed{u \hookrightarrow \{0,1\}}_{1/\epsilon}$.

We get consequently to some functions that are indeed analytic. It remains to check that each of the parameters n, m, ϵ, ϵ', ... can be fixed sufficiently small so that the dynamic will never leave a vicinity of the dynamic that we intend to simulate. This is basically developing all the previous arguments, without true difficulties, using basic analysis: refer to [2] for details.

2.5 Going to pODEs

The ODE that we obtained is not polynomial: It uses sin, arctan, ... for example. But it can be transformed into some pODE. The idea is that many ODEs can be transformed as such using a simple process: Introduce some variables that are needed, express their derivative in terms of already present variables, using usual relations on derivatives, and repeat this process until it terminates. It terminates in practice very quickly for usual functions.

Maybe an example is more clear than a long and formal discussion. Suppose that you want to program the equation
$\begin{cases} y_1' = \sin^2 y_2 \\ y_2' = y_1 \cos y_2 - e^{e^{y_1}+t} \end{cases}$ with the initial condition $y_1(0) = 0$ and $y_2(0) = 0$.

Since y_1' is expressed as the square of the sinus of y_2, and this is not a polynomial, we introduce $y_3 = \sin y_2$, and we write $y_1' = y_3^2$ with $y_3(0) = 0$. Similarly, since $y_1 \cos y_2 - e^{e^{y_1}+t}$ is not polynomial, we introduce $y_4 = \cos y_2$ and $y_5 = e^{e^{y_1}+t}$, and we write $y_2' = y_1 y_4 - y_5$.

We then consider the derivatives of the variables that have been introduced. We compute the derivative of y_3, and we realize that this is $y_2' \cos(y_2)$. We already know y_2', and $\cos(y_2)$ is y_4. We can hence write $y_3' = y_4(y_1 y_4 - y_5)$ that is a polynomial. We now focus on the derivative of y_4 that values $-y_2' \sin(y_2)$, which can be written $y_4' = -y_3(y_1 y_4 - y_5)$. We next go to the derivative of y_5 that writes $y_5' = y_5(y_6 y_3^2 + 1)$ if we introduce $y_6 = e^{y_1}$ to keep it polynomial, writing $y_5(0) = e$. We then go to the derivative of y_6 that values $y_1' e^{y_1} = y_6 y_3^2$ which is polynomial and we write $y_6(0) = 1$.

We have obtained that we can simulate (by just projecting) the previous system by the pIVP

$$\begin{cases} y_1' = y_3^2 \\ y_2' = y_1 y_4 - y_5 \\ y_3' = y_4(y_1 y_4 - y_5) \\ y_4' = -y_3(y_1 y_4 - y_5) \\ y_5' = y_5(y_6 y_3^2 + 1) \\ y_6' = y_6 y_3^2 \end{cases} \text{ with } \mathbf{y}(0) = (y_1, \ldots, y_6)(0) = (0, 0, 0, 1, e, 1).$$

Coming back to previous dynamics (11): Just do the similar process on the analytic dynamic. This will necessarily terminates (from known closure properties established in [2]) and this will provide a pIVP.

Of course not all the ingredients of ODE programming are covered by this example, and we miss constructions such as using change of variables, or mixing results, etc... by lack of space. But it is however quite representative of ODE programming technology, and of our pseudo programming language.

References

1. Hainry, E.: Modèles de calculs sur les réels. Résultats de Comparaisons. PhD thesis, LORIA (7 Décembre 2006)
2. Graça, D.S.: Computability with Polynomial Differential Equations. PhD thesis, Instituto Superior Técnico (2007)
3. Pouly, A.: Continuous models of computation: from computability to complexity. PhD thesis, Ecole Polytechnique and Unidersidade Do Algarve (Defended on July 6, 2015. 2015) https://pastel.archives-ouvertes.fr/tel-01223284, Prix de Thèse de l'Ecole Polyechnique 2016, Ackermann Award 2017.
4. Gozzi, R.: Analog Characterization of Complexity Classes. PhD thesis, Instituto Superior Técnico, Lisbon, Portugal and University of Algarve, Faro, Portugal (2022)
5. Bournez, O., Pouly, A.: A universal ordinary differential equation. Logical Methods in Computer Science **16**(1) (2020)
6. Fages, F., Le Guludec, G., Bournez, O., Pouly, A.: Strong turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs. In: Computational Methods in Systems Biology-CMSB 2017. (2017)
7. Bournez, O., Graça, D.S., Pouly, A.: Polynomial Time corresponds to Solutions of Polynomial Ordinary Differential Equations of Polynomial Length. Journal of the ACM **64**(6) (2017) 38:1–38:76
8. Graça, D.S., Campagnolo, M.L., Buescu, J.: Computability with polynomial differential equations. Adv. Appl. Math. **40**(3) (2008) 330–349

9. Bournez, O., Pouly, A.: A survey on analog models of computation. In: Handbook of Computability and Complexity in Analysis. Springer (2021) 173–226
10. Bournez, O., Campagnolo, M.L.: New computational paradigms. changing conceptions of what is computable. Springer-Verlag, New York (2008) 383–423
11. Orponen, P.: A survey of continous-time computation theory. In Du, D.Z., Ko, K.I., eds.: Advances in Algorithms, Languages, and Complexity, Kluwer Academic Publishers (1997) 209–224
12. Ulmann, B.: Analog and hybrid computer programming. De Gruyter Oldenbourg (2020)
13. Ulmann, B.: Analog computing. Walter de Gruyter (2013)
14. MacLennan, B.J.: Analog computation. In: Encyclopedia of complexity and systems science. Springer (2009) 271–294
15. Shannon, C.E.: Mathematical theory of the differential analyser. Journal of Mathematics and Physics MIT **20** (1941) 337–354
16. Graça, D.S., Costa, J.F.: Analog computers and recursive functions over the reals. Journal of Complexity **19**(5) (2003) 644–664
17. Bournez, O., Campagnolo, M.L., Graça, D., S. Hainry, E.: Polynomial differential equations compute all real computable functions on computable compact intervals. Journal of Complexity **23**(3) (2007) 317–335
18. Bournez, O., Gozzi, R., Graça, D.S., Pouly, A.: A continuous characterization of PSPACE using polynomial ordinary differential equations. (2022)
19. Gozzi, R., Graça, D.S.: Characterizing time computational complexity classes with polynomial differential equations. Submitted (2022)
20. Branicky, M.S.: Universal computation and other capabilities of hybrid and continuous dynamical systems. Theoretical Computer Science **138**(1) (6 February 1995) 67–100
21. Rubel, L.A.: A universal differential equation. Bulletin of the American Mathematical Society **4**(3) (May 1981) 345–349
22. Boshernitzan, M.: Universal formulae and universal differential equations. Annals of mathematics **124**(2) (1986) 273–291
23. Cook, M., Soloveichik, D., Winfree, E., Bruck, J.: Programmability of chemical reaction networks. In: Algorithmic Bioprocesses. Springer (2009) 543–584
24. Pour-El, M.B., Richards, J.I.: A computable ordinary differential equation which possesses no computable solution. Ann. Math. Logic **17** (1979) 61–90
25. Collins, P., Graça, D.S.: Effective computability of solutions of ordinary differential equations the thousand monkeys approach. Electronic Notes in Theoretical Computer Science **221** (2008) 103–114