Computability, Complexity and Programming with Ordinary Differential Equations

Olivier Bournez

To cite this version:

Olivier Bournez. Computability, Complexity and Programming with Ordinary Differential Equations. 37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020), 2020, Montpellier, France. 10.4230/LIPIcs.STACS.2020.3 . hal-03668004

Computability, Complexity and Programming with Ordinary Differential Equations

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

Abstract

Ordinary Differential Equations (ODEs) appear to be a universally adopted and very natural way for expressing properties for continuous time dynamical systems. They are intensively used, in particular in applied sciences. There exists an abundant literature about the hardness of solving ODEs with numerical methods.

We adopt a dual view: we consider ODEs as a way to program or to describe our mathematical/computer science world. We survey several results considering ODEs under this computational perspective, with a computability and complexity theory point of view. In particular, we provide various reasons why polynomial ODEs should be considered as the continuous time analog of Turing machines for continuous-time computations, or should be used as a way to talk about mathematical logic.

This has already applications in various fields: determining whether analog models of computation can compute faster than classical digital models of computation; solving complexity issues for computations with biochemical reactions in bioinformatics; machine independent characterizations of various computability and complexity classes such as PTIME or NPTIME, or proof of the existence of a universal polynomial ordinary differential equation whose solutions can approximate any continuous function if provided with a suitable well-chosen initial condition.

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

Keywords and phrases Ordinary differential equations, Models of computation, Analog Computations, discrete ordinary differential equations, Implicit complexity, recursion scheme

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.3

Category Invited Talk

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

Acknowledgements This material is based upon work supported by the RACAF Project from Agence National de la Recherche.

1 Back to the history of analog models of computation

In 1941, Claude Shannon introduced in [42] the *General Purpose Analog Computer* (GPAC) model as a model for the Differential Analyzer [13], on which he worked as an operator.

Differential Analysers were mechanical (and later on electronics) continuous time analog machines. First ever built machine was built under the supervision of V. Bush 1931 at MIT: Applications were from gunfire control up to aircraft design. First differential analyzers, such as the ones at the time of Shannon, were mechanical. Electronic versions were used from late 40s until 70s. Nowadays, company *Analog paradigm* is selling some modern differential analyzers based on operational amplifiers.

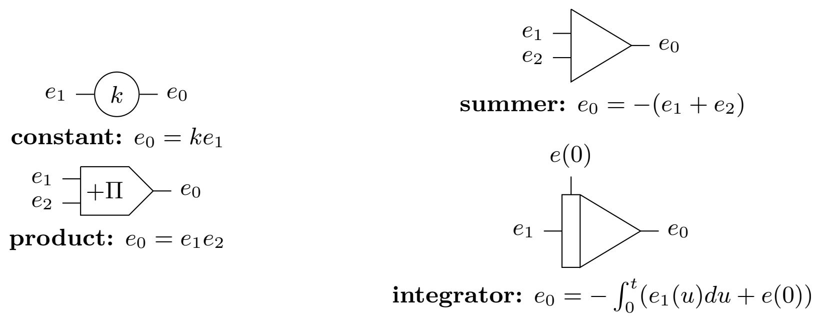

Figure 1 Presentation of the 4 types of units: constant, adder, multiplier, and integrator.

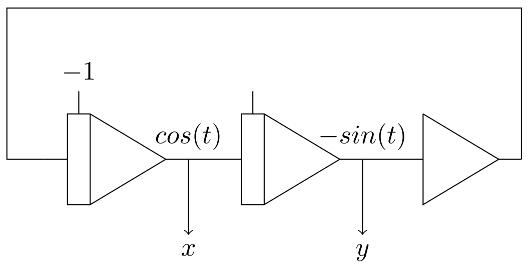

Figure 2 Example of GPAC circuit: computing sine and cosine with two variables.

Basically, a GPAC is any circuit that can be build from the 4 basic units of Figure 1, that is to say from basic units realizing constants, additions, multiplications and integrations, all of them working over analog real quantities (that were corresponding to angles in the mechanical Differential Analysers, and later on to voltage in the electronic versions).

Actually, not all kinds of interconnections must be allowed since this may lead to some undesirable behavior (*e.g.* non-unique outputs. For further details, refer to [29]).

Figures 2 illustrates for example how the sine function can generated using two integrators, with suitable initial state, as being the solution of ordinary differential equation

$$\begin{cases} y'(t) = z(t) \\ z'(t) = -y(t) \end{cases}$$

with suitable initial conditions.

Shannon, in his original paper, already mentioned that the GPAC generates polynomials, the exponential function, the usual trigonometric functions, their inverses, and their composition. More generally, Shannon claimed that all functions generated by a GPAC are differentially algebraic in the sense of the following definition.

▶ **Definition 1.** *A unary function y is differentially algebraic (d.a.) on the interval I if there exists an $n \in \mathbb{N}$ and a nonzero polynomial p with real coefficients such that*

$$p\left(t, y, y', ..., y^{(n)}\right) = 0, \qquad \text{on } I. \tag{1}$$

As a corollary, and noting that the Gamma function $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt$ is not d.a. [38], we get that

▶ **Proposition 2.** *The Gamma function cannot be generated by a GPAC.*

Another famous example of not d.a. function is Riemann's Zeta function $\zeta(x) = \sum_{k=0}^{\infty} \frac{1}{k^x}$.

2 Polynomial Ordinary Differential Equations (pODEs)

Following corrections and refinements [37, 34, 29, 27] of some statements from Shannon, GPACs can be identified with polynomial Ordinary Differential Equations (pODE), i.e. dynamics of type $y' = p(y)$ where $y(t) = (y_1(t), \dots, y_d(t)) \in \mathbb{R}^d$ is a (vectorial) function of time, and $p : \mathbb{R}^d \to \mathbb{R}^d$ is a (vector of) polynomials, where d is some integer.

Hence, one can consider that a GPAC is a polynomial ordinary differential equation of *fixed* dimension, or a polynomial initial value (pIVP) if some initial condition $y(t_0) = y_0$ is added.

Following Shannon,

▶ **Definition 3** ([42]). *A function $f : \mathbb{R} \to \mathbb{R}$ is* GPAC-generable *if it corresponds to some (coordinate) projection of a solution of an pIVP: that is there exists a pODE (i.e. some p, t_0, $i_0 \in \{1, 2, \dots, d\}$ and y_0 over $\mathbb{R}^d$ for some d) such that $f(t) = y_{i_0}(t)$ for all $t \in \mathbb{R}$.*

The concept can naturally be extended to function $f : \mathbb{R}^m \to \mathbb{R}^m$ [42, 2]. From definition, a GPAC-generable function is necessarily (real) analytic.

In modern terms, this concept is strongly related to the so-called class of *Noetherian functions* [36] (and hence also to *Pfaffian* or *differentially algebraic functions*: see e.g. [24]).

The class of generable functions is particularly robust: Several variations on the GPAC circuits were explored and proven to be all equivalent. This essentially shows that the above notion is probably the right definition to be considered [27]. Furthermore, it has very strong closure properties: For example.

▶ **Proposition 4** ([28]). *The class of functions generated by GPACs is closed under the operations $+, -, \times, \div$, under composition, derivation, and compositional inverses (i.e. if f is generated by a GPAC, then so is f^{-1}).*

The following result states that the solution of an initial-value problem defined with functions generated by GPACs is also generated by a GPAC.

▶ **Proposition 5** ([28]). *Consider the initial value problem (IVP)*

$$\begin{cases} x' = f(t, x), \\ x(t_0) = x_0, \end{cases} \tag{2}$$

where $f : \mathbb{R}^{n+1} \to \mathbb{R}^n$ and each component of f is a composition of polynomials and functions generated by GPACs. Then there exist $m \geq n$, a polynomial $p : \mathbb{R}^{m+1} \to \mathbb{R}^m$ and a $y_0 \in \mathbb{R}^m$ such that the solution of (2) is given by the first n components of $y = (y_1, \dots, y_m)$, where y is the solution of the polynomial IVP

$$\begin{cases} y' = p(t, y), \\ y(t_0) = y_0. \end{cases} \tag{3}$$

A digression in order to possibly help with intuition: Let us authorize ourselves in this single paragraph to leave the world of formal mathematical statements, to possibly help for intuition: It follows from all these closure properties that *"most" "common"* analytic functions fall in the class of generable functions and turn out to be GPAC-generable, i.e. generable with a differential analyzer. Of course, this last statement is not referring to a well-defined concept of *"most"* and *"common"*, but to some well-known fact to engineers at the time of analog machines: Repeating Shannon [42], *"When the Differential Analyzers was first built it was thought that all functional relationships between terms of the equation* *being solved would have to be introduced into the machine by means of input tables. Soon it was found that practically all the important simple functions could be 'generated' using only integrators and adders.".* This phenomenon has similarities with the class of computable functions in classical computability which include all *"most" "common"* functions, even if we can build some non-computable functions (e.g. by diagonalization).

Back to the mathematical world: From Shannon's analysis, as we know some (analytic) Turing computable (in the sense of computable analysis) *non-differentially algebraic* functions (e.g. Euler's ζ and Riemann's Γ) it was deduced that the model is not Turing complete, and the model was mainly forgotten.

▶ Remark 6. Observe that the proofs of the non-differential algebraicity of ζ and Γ by respectively Hilbert and Hölder [31] can consequently be seen as a proof of uncomputability (ungenerability to be more precise) really different from usual methods of (un)computability such as diagonalization.

3 Revisiting computability with polynomial ordinary differential equations

We proved in 2007 [6] that this statement about the limitation of the computational power of the GPAC model comes mainly from a misconception of what was called *computable* at the time of Shannon with respect to what would be called such today: If the input is given as some initial data, and if we allow the machine not to produce its output in real-time, but after *some delay* (i.e. after some *computation time* (encoded by the delay required for y_j to be less than required precision in Theorem 9 which follows)), as in all modern models of computations, then the model can compute any function that is classically (Turing) computable.

Call this notion *GPAC-computability* by opposition to *GPAC-generability*: GPAC-computability does correspond to classical (i.e. Turing) computability [5]. Consequently, the class of GPAC-computable does inherit from all the closure and robustness properties of computable functions and does include all GPAC-generable functions, as well as functions such as Γ, ζ and many others, as soon as we know that there are (Turing) computable.

Formally (see Figure 3 for illustration).

▶ **Definition 7.** *A function $f : [a, b] \to \mathbb{R}$ is GPAC-computable iff there exist some polynomials with polynomial coefficients $p : \mathbb{R}^{n+1} \to \mathbb{R}^n$, $p_0 : \mathbb{R} \to \mathbb{R}$, and $n-1$ computable real values $\alpha_1, ..., \alpha_{n-1}$ such that:*

1. *$(y_1, ..., y_n)$ is the solution of the Cauchy problem $y' = p(y, t)$ 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 $t \in [0, +\infty)$.*

We remark that $\alpha_1, \ldots, \alpha_{n-1}$ are auxiliary parameters needed to compute f.

The following is true with this notion of computation:

▶ **Proposition 8** ([27]). *The Gamma function Γ is GPAC-computable.*

And more generally:

▶ **Theorem 9.** *Let a and b be computable reals. A function $f : [a, b] \to \mathbb{R}$ is computable in the sense of computable analysis iff it is GPAC-computable.*

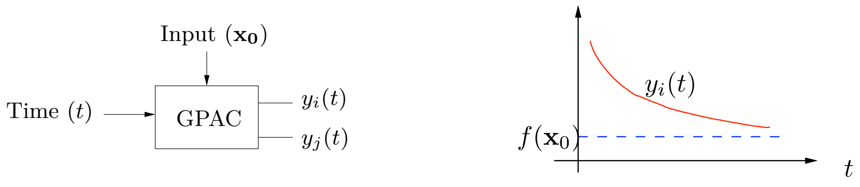

Figure 3 Graphical illustration of Definition 7.

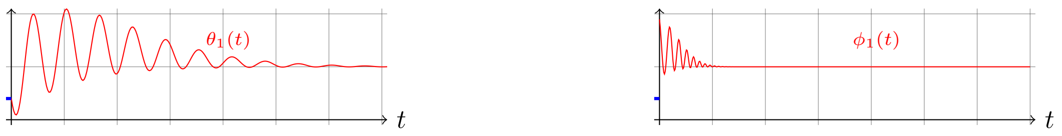

Figure 4 A continuous system before and after an exponential time speed-up.

More importantly, to prove so, we developed some *ad hoc* techniques to program with pODEs: how to realize *assignments*, *iterations*, *loops,* etc. with differential equations. This led us to understand that pODEs play a very particular role, in particular, if one wants to go to complexity theory, and not only computability theory.

4 Complexity theory with polynomial ordinary differential equations

This connection between Turing computability and the GPAC was recently refined [9, 8] at the level of complexity, with a characterization of the class PTIME and polynomial time computable real functions.

Defining a robust time complexity notion for continuous time systems was a well-known open problem [4], 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” (here *time-contraction*). For example, consider a continuous system defined by an ODE $y' = f(y)$ where $f : \mathbb{R} \to \mathbb{R}$ and with solution $\theta : \mathbb{R} \to \mathbb{R}$. Now consider the following system $y' = f(y)z$, $z' = z$ with solution $\phi : \mathbb{R}^2 \to \mathbb{R}^2$. This system re-scales the time variable and that its solution $\phi = (\phi_1, \phi_2)$ is given by $\phi_2(t) = e^t$ and $\phi_1(t) = \theta(e^t)$ (see Figure 2). Therefore, the second ODE simulates the first ODE, with an exponential acceleration. Similarly, it is also possible to present an ODE which has a solution with a component $\varphi_1 : \mathbb{R} \to \mathbb{R}$ such that $\varphi_1(t) = \phi(\tan t)$, i.e. it is possible to contract the whole real line into a bounded set. Thus any language computable by the first system (or, in general, by a continuous system) can be computed by another continuous system in time $O(1)$. This problem appears not only for PIVPs (or, equivalently, GPACs), but also for many continuous models (see e.g. [40], [41], [35], [3], [14], [21], [18], [19]). We illustrated here *time contraction*, but a *space contraction* (or both simultaneously) is also possible, using a change of variable on space variable y.

We solved the above mentioned open problem for time complexity and proved that a robust notion of *computation time* for a pIVP is provided by *the length* of the solution (up to polynomial time) in [9, 8].

This has been established by proving that a time t Turing machine can be simulated (we established this direction using ad hoc suitable pODE programming) in a length $poly(t)$, and conversely that pODE can be solved (we established this direction using an original numerical method) by a Turing machine in a time polynomial in the length of the solution.

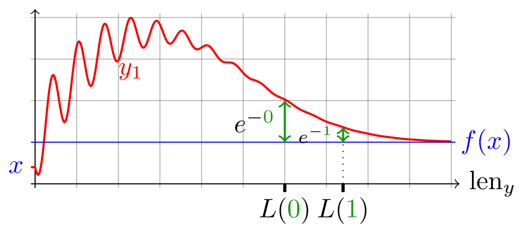

Figure 5 Graphical representation of Theorem 2: on input x, starting from initial condition $(x, 0, \ldots, 0)$, the pODE $y' = p(y)$ ensures that $y_i(t)$ gives $f(x)$ with accuracy better than $e^{-\mu}$ as soon as the length of y (from 0 to t) is greater than $L(\mu)$. Other variables $y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_n$ are not plotted and the horizontal axis measures the length of y (instead of time t).

Formally: We recall that the length of a curve $y \in C^1(I, \mathbb{R}^n)$ defined over some interval $I = [a, b]$ is given by $\text{len}_y(a, b) = \int_I \|y'(t)\|_2 \, dt$.

▶ **Theorem 10** ([8], Equivalence between GPAC and CA (Complexity)). *A function $f : [a, b] \to \mathbb{R}$ is computable in polynomial time (in the sense of Computable Analysis) if and only if there exists a polynomial $L : \mathbb{R}_+ \to \mathbb{R}_+$, an integer d, $i \in \{1, 2, \ldots, d\}$ and a vector of polynomials $p : \mathbb{R}^d \to \mathbb{R}^d$ with coefficients in $\mathbb{Q}$, such that for any $x \in [a, b]$, the unique solution $y : \mathbb{R}_+ \to \mathbb{R}^d$ to*

$$y(0) = (x, 0, \ldots, 0), \qquad y'(t) = p(y(t))$$

satisfies for all $t \in \mathbb{R}_+$:
- *for any $\mu \in \mathbb{R}_+$, if $\text{len}_y(0, t) \geqslant L(\mu)$ then $|f(x) - y_i(t)| \leqslant e^{-\mu}$,*
- *$\|y'(t)\| \geqslant 1$.*

In other words, the precision of y_i increases with the length of the curve. More precisely, as soon as the length between 0 and t is at least $L(\mu)$, the precision is at least $e^{-\mu}$. Notice how rescaling the curve would not help here since it does not change the length of y. The second condition on the derivative of y prevents some pathological cases and ensures that curve has infinite length, and thus that y_i indeed converges to $f(x)$. It is possible to extend this equivalence to multivariate functions and unbounded input domains such as $\mathbb{R}$, by making L take into account the norm of x.

Previous statement is a characterization of functions computable in polynomial time, i.e. $\text{FPTIME}_{\mathbb{R}}$ over the reals. If one wants to talk about decision problems over alphabet $\{0, 1\}$, it possible to define the class PTIME directly in terms of differential equations, by encoding words with rational numbers. Again the length plays a crucial role, but since a differential equation does not "stop", the component y_i is used to signal that it accepts ($y_i \geqslant 1$) or rejects ($y_i \leqslant -1$).

▶ **Theorem 11** ([8], Analog characterization of PTIME). *A language $\mathcal{L} \subseteq \{0, 1\}^*$ belongs to* PTIME*, the class of polynomial time decidable languages, if and only if there exist a polynomial $L : \mathbb{N} \to \mathbb{N}$, an integer d, $i \in \{1, 2, \ldots, d\}$ and a vector of polynomials $p : \mathbb{R}^d \to \mathbb{R}^d$ with coefficients in $\mathbb{Q}$, such that for all words $w \in \{0, 1\}^*$, the unique solution $y : \mathbb{R}_+ \to \mathbb{R}^d$ to*

$$y(0) = (0, \ldots, 0, |w|, \psi(w)), \qquad y'(t) = p(y(t))$$

where $\psi(w) = \sum_{i=1}^{|w|} w_i 2^{-i}$, satisfies for all $t \in \mathbb{R}_+$:
- *if $|y_i(t)| \geqslant 1$ then $|y_i(u)| \geqslant 1$ for all $u \geqslant t \geqslant 0$ (and similarly for $|y_i(t)| \leqslant -1$),*
- *if $w \in \mathcal{L}$ (resp. $\notin \mathcal{L}$) and $\text{len}_y(0, t) \geqslant L(|w|)$ then $y_i(t) \geqslant 1$ (resp. $\leqslant -1$),*
- *$\|y'(t)\| \geqslant 1$.*

One clear interest of the previous statements is that they provide a way to define classical concepts from the theory of computation (computable function, polynomial time computable functions) only using concepts from analysis, namely polynomial ordinary differential equations.

This has many consequences: This provides a way to define classical concepts from the theory of computation using only concepts from analysis: Theorem 10 provides a definition of computability for functions more natural than the classical digression through type 2 computability (see e.g. [43]), in particular when presenting the concept to people working in analysis. This transforms the question whether analog models can solve faster problems (even about the discrete) than classical means to a question about lengths of solutions. This proves that pODE solving is (PTIME-)complete in its length and solves the issue of avoidance of the so-called Zeno's phenomenon in the verification community in models of hybrid and cyberphysical systems [8]. This also brings new perspectives on some philosophical concepts such as time for systems of our physical world.

5 Some applications of pODE programming

Actually, more generally, ODEs is *universal language* in many domains, in particular in applied fields (physics, biology, ...). We already used our *"ODE programming technology"* to solve several open problems in various fields. We illustrate this with several examples in the rest of this article.

5.1 A universal ordinary differential equation

In computer algebra, a well-known surprising result is due to L. A. Rubel who proved in 1981 in [39] that there exists a fixed non-trivial fourth-order polynomial differential algebraic equation (DAE)

$$p(y, y', \dots, y^d) = 0$$

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. Lee A. Rubel provided an explicit example of such a polynomial DAE [39]. Other examples of universal DAE have later been proposed by other authors.

However, while this result may seem very surprising, its proof is quite simple, is frustrating, and is mainly an indication that DAE is too loose a model for modeling purpose:

- First, the involved notions of universality is far from usual notions of universality in computability theory because the proofs heavily rely on the fact that constructed DAE does not have unique solutions for a given initial data. Indeed, in general a DAE may not have a unique solution, given some initials conditions. But Rubel's DAE *never* has a unique solution, even with a countable number of conditions of the form $y^{(k_i)}(a_i) = b_i$. This is very different from usual notions of universality where one would expect that there is clear unambiguous notion of evolution for a given initial data, for example as in computability theory.
- Second, the proofs usually rely on solutions that are piecewise defined. Hence they cannot be analytic, while analycity is often a key expected property in experimental sciences.
- Third, the proofs of these results can be interpreted more as the fact that (fourth-order) polynomial algebraic differential equations is a too loose a model compared to classical ordinary differential equations. In particular, one may challenge whether the result is really a universality result.

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 [39, page 2], [1, Conjecture 6.2]. In [10], using ODE programming, we solved it and showed that Rubel's statement holds for polynomial ordinary differential equations, and since polynomial ODEs have a unique solution given an initial data, this positively answers Rubel's open problem:

▶ **Theorem 12** ([11], Universal PIVP). *There exists a* fixed *polynomial vector p in d variables with rational coefficients such that for any functions $f \in C^0(\mathbb{R})$ and $\varepsilon \in C^0(\mathbb{R}, \mathbb{R}^*_+)$, there exists $\alpha \in \mathbb{R}^d$ such that there exists a unique solution $y : \mathbb{R} \to \mathbb{R}^d$ to $y(0) = \alpha$, $y' = p(y)$. Furthermore, this solution satisfies that $|y_1(t) - f(t)| \leqslant \varepsilon(t)$ for all $t \in \mathbb{R}$, and it is analytic.*

Furthermore, α can be computed from f and ε (in the sense of Computable Analysis).

Using the fact that polynomial ODEs can be transformed into differentially algebraic equations (see [15]), the following then follows.

▶ **Theorem 13** ([11], Universal DAE). *There exists a* fixed *polynomial p in $d+1$ variables with rational coefficients such that for any functions $f \in C^0(\mathbb{R})$ and $\varepsilon \in C^0(\mathbb{R}, \mathbb{R}^*_+)$, there exists $\alpha_0, \dots, \alpha_{d-1} \in \mathbb{R}$ such that there exists a unique* analytic *solution $y : \mathbb{R} \to \mathbb{R}$ to $y(0) = \alpha_0, y'(0) = \alpha_1, \dots, y^{(d-1)}(0) = \alpha_{d-1}$, $p(y, y', \dots, y^d) = 0$. Furthermore, this solution satisfies that $|y(t) - f(t)| \leqslant \varepsilon(t)$ for all $t \in \mathbb{R}$.*

Furthermore, α can be computed from f and ε (in the sense of Computable Analysis).

5.2 Strong Turing completeness of biochemical reactions

Ordinary differential equations is a well-used models for modeling the dynamics of chemical reactions. In particular, the Turing completeness of kinetic reactions was an open problem (see e.g. [17, Section 8]) that we solved using previous polynomial ODE programming technology: we prove the Turing completeness of chemical reaction networks over a finite set of molecular species under the differential semantics in [23].

A biochemical reaction system is a *positive* dynamical system living in the cone $\mathbb{R}^n_+$, where the state is defined by the positive concentration values of the molecular species. We restrict to elementary reaction systems, governed by the mass-action-law kinetics and where each reaction has at most two reactants.

In short, let $\mathcal{M}$ be a *finite set* of n molecular species $\{y_1, \dots, y_n\}$.

▶ **Definition 14** ([22]). *A* reaction *is a triple (R, P, f), where $R : \mathcal{M} \to \mathbb{N}$ is a multiset of reactants, $P : \mathcal{M} \to \mathbb{N}$ is a multiset of products and $f : \mathbb{R}^n_+ \to \mathbb{R}_+$, called the* rate function, *is a partially differentiable function verifying $R(y_i) > 0$ iff $\frac{\partial f}{\partial y_i}(y) > 0$ for some $y \in \mathbb{R}^n_+$.*

A reaction system *is a finite set of reactions.*

A mass-action-law reaction *is a reaction in which the rate function f is a monomial of the form $k * \Pi_{y \in \mathcal{M}} y^{R(y)}$ where k is called the* rate constant.

An elementary reaction *is a mass-action-law reaction with at most two reactants.*

▶ **Definition 15** ([22]). *The differential semantics of a reaction system $\{(R_i, P_i, f_i)\}_{i \in I}$ is the ODE system*

$$\{y' = \Sigma_{i \in I}(R_i(y) - Pi(y)) * f_i\}_{y \in \mathcal{M}}.$$

The dynamics given by the law of mass action leads to a polynomial ODE system of the form $y'(t) = p(y(t))$ with $p(y)_i = \sum_j (P_j(y_i) - R_j(y_i) * k_j * \Pi_{i=1}^n {y_i}^{R_j(y_i)}$. There are thus additional constraints, compared to general polynomial ordinar differential equations: the components y_i must always be positive, and the monomials of p_i whose coefficient is negative must have a non-zero y_i exponent. These constraints are necessary conditions for the existence of a set of biochemically realizable reactions that react according to the dynamics $y' = p(y)$.

Interestingly, we prove that the previous computability and complexity results can be generalized to elementary biochemical reaction systems. First, the restriction to positive systems is shown complete, by encoding each component y_i by the difference between two positive components y_i^+ and y_i^-, which can be normalized by a mutual annihilation reaction, $y_i^+ + y_i^- \Rightarrow \_$, so that one variable is null.

▶ **Definition 16.** *A function $f : \mathbb{R}_+ \to \mathbb{R}_+$ is* chemically-computable *if there exist a mass-action-law reaction system $\{(R_i, P_i, f_i)\}_{i\in I}$ over some molecular species $\{y_1, ..., y_n\}$, and a polynomial $q \in {\mathbb{R}_+}^n[\mathbb{R}_+]$ defining the initial concentration values, such that f is GPAC-computed by q and its (polynomial) differential semantics $p \in {\mathbb{R}_+}^n[{\mathbb{R}_+}^n]$.*

A function $f : \mathbb{R}_+ \to \mathbb{R}$ is chemically-computable *if there exists a chemically computable function $f^+ : \mathbb{R}_+ \to {\mathbb{R}_+}^2$ over $\{y_1^+, ..., y_n^+, y_1^-, ..., y_n^-\}$ such that $f = f_1^+ - f_2^-$.*

In this definition, to compute $f(x)$, one has thus to design a reaction system over a finite set of molecular species, initialized to some values defined by a vector of polynomials $q(x)$ (e.g. following [12, 16]), which guarantees that the result is obtained in the concentration of one distinguished molecular species, with a precision indicated by another distinguished molecular species (see Definition 7).

We prove.

▶ **Theorem 17** ([23]). *Any GPAC-computable function can be computed by a mass-action-law reaction system under the differential semantics preserving the polynomial length complexity.*

▶ **Theorem 18** ([23]). *Elementary reaction systems on finite universes of molecules are Turing-complete under the differential semantics.*

5.3 Revisiting classical computability theory with discrete ODEs

We also came to 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 [25] and is reminiscent in *discrete calculus* [30, 26, 32, 33] for combinatorial analysis. 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.

We started 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 illustrated 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 also demonstrated 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.

More concretely, we focus on functions over the integers of type $f : \mathbb{N}^p \to \mathbb{Z}^d$, or functions of type $f : \mathbb{Z}^p \to \mathbb{Z}^d$, for some integers p, d. The basic idea is to consider the following concept of derivative, introduced here for the case where $p = 1$. We will later on consider more general functions, with partial derivatives instead of derivatives (defined then as expected).

▶ **Definition 19** (Discrete Derivative). *The discrete derivative of $f(x)$ is defined as $\Delta f(x) = f(x+1) - f(x)$. We will also write f' for $\Delta f(x)$ to help to understand statements with respect to their classical continuous counterparts.*

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

▶ **Definition 20** ($\mathcal{L}$-ODE). *Let $\mathcal{L} : \mathbb{N}^{p+1} \to \mathbb{Z}$. We write*

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

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

▶ Remark 21. This is motivated by the fact that the latter expression is similar to classical formula for classical continuous ODEs:

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

This allows us to simulate suitable changes of variables using this analogy. We talk about $\mathcal{L}$-Initial Value Problem (IVP) when some initial condition is added. An important special case is when $\mathcal{L}(x, y)$ corresponds to the bit length $\mathcal{L}(x, y) = \ell(x)$ function: we call this special case length-ODEs. Let $\mathsf{sg}(x) : \mathbb{Z} \to \mathbb{Z}$ denotes the function that takes value 1 for $x > 0$ and 0 in the other case.

▶ **Definition 22.** *A sg-polynomial expression $P(x_1, ..., x_h)$ is an expression built-on $+, -, \times$ (often denoted $\cdot$) and $\mathsf{sg}()$ 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 $x' \neq x$,* $\deg(x, x') = 0$
- $\deg(x, P + Q) = \max\{\deg(x, P), \deg(x, Q)\}$
- $\deg(x, P \times Q) = \deg(x, P) + \deg(x, Q)$
- $\deg(x, \mathsf{sg}(P)) = 0$

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

▶ **Example 24.** 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 25.** *Function f is linear $\mathcal{L}$-ODE definable (from u, g and h) if it corresponds to the solution of $\mathcal{L}$-IVP*

$$\frac{\partial f(x,y)}{\partial \mathcal{L}} = u(f(x,y), h(x,y), x, y) \qquad f(0,y) = g(y) \tag{5}$$

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

▶ **Definition 26** ($\mathbb{DL}$)**.** *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.*

▶ **Theorem 27** ([7])**.** $\mathbb{DL} = \text{FPTIME}$

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 FNP as we did in [7]. Our current investigations concern $\text{PTIME}_{[0,1]}$ 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.

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. [20, 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 Michael Boshernitzan. Universal formulae and universal differential equations. *Annals of mathematics*, 124(2):273–291, 1986.

2 Daniel Bournez, Olivierand Graça and Amaury Pouly. On the Functions Generated by the General Purpose Analog Computer. *Information and Computation*, 257:34–57, 2017. `doi:10.1016/j.ic.2017.09.015`.

3 Olivier Bournez. Achilles and the Tortoise climbing up the hyper-arithmetical hierarchy. *Theoretical Computer Science*, 210(1):21–71, 1999.

4 Olivier Bournez and Manuel L. Campagnolo. *New Computational Paradigms. Changing Conceptions of What is Computable*, chapter A Survey on Continuous Time Computations, pages 383–423. Springer-Verlag, New York, 2008.

5 Olivier Bournez, Manuel L. Campagnolo, Daniel Graça, and Emmanuel S. Hainry. Polynomial differential equations compute all real computable functions on computable compact intervals. *Journal of Complexity*, 23(3):317–335, 2007.

6 Olivier Bournez, Manuel L. Campagnolo, Daniel S. Graça, and Emmanuel Hainry. Polynomial differential equations compute all real computable functions on computable compact intervals. *jcomp*, 23(3):317–335, 2007.

7 Olivier Bournez and Arnaud Durand. Recursion schemes, discrete differential equations and characterization of polynomial time computation. In *MFCS 2019: 44th International Symposium on Mathematical Foundations of Computer Science*, 2019.

8 Olivier Bournez, Daniel S. Graça, and Amaury 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`.

9 Olivier Bournez, Daniel S. Graça, and Amaury Pouly. Polynomial Time corresponds to Solutions of Polynomial Ordinary Differential Equations of Polynomial Length. The General Purpose Analog Computer and Computable Analysis are two efficiently equivalent models of computations. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 109:1–109:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

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

11 Olivier Bournez and Amaury Pouly. A universal ordinary differential equation. *To appear in Logical Methods in Computer Science*, abs/1702.08328, 2020. `arXiv:1702.08328`.

12 H. J. Buisman, H. M. M. ten Eikelder, P. A. J. Hilbers, and A. M. L. Liekens. Computing algebraic functions with biochemical reaction networks. *Artificial Life*, 15(1):5–19, 2009. doi:10.1162/artl.2009.15.1.15101.

13 Vannevar Bush. The differential analyzer. A new machine for solving differential equations. *J. Franklin Inst.*, 212:447–488, 1931.

14 C. S. Calude and B. Pavlov. Coins, quantum measurements, and Turing's barrier. *Quantum Information Processing*, 1(1-2):107–127, 2002.

15 David C. Carothers, G. Edgar Parker, James S. Sochacki, and Paul G. Warne. Some properties of solutions to polynomial systems of differential equations. *Electronic Journal of Differential Equations*, 2005(40):1–17, 2005.

16 Ho-Lin Chen, David Doty, and David Soloveichik. Rate-independent computation in continuous chemical reaction networks. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science*, ITCS '14, pages 313–326, New York, NY, USA, 2014. ACM. doi:10.1145/2554797.2554827.

17 Matthew Cook, David Soloveichik, Erik Winfree, and Jehoshua Bruck. Programmability of chemical reaction networks. In *Algorithmic Bioprocesses*, pages 543–584. Springer, 2009.

18 B. Jack Copeland. Even Turing machines can compute uncomputable functions. In C.S. Calude, J. Casti, and M.J. Dinneen, editors, *Unconventional Models of Computations.* Springer-Verlag, 1998.

19 B. Jack Copeland. Accelerating Turing machines. *Minds and Machines*, 12:281–301, 2002.

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

21 E. B. Davies. Building infinite machines. *The British Journal for the Philosophy of Science*, 52:671–682, 2001.

22 François Fages, Steven Gay, and Sylvain Soliman. Inferring reaction systems from ordinary differential equations. *Theoretical Computer Science*, 599:64–78, 2015. doi:10.1016/j.tcs.2014.07.032.

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

24 Andrei Gabrielov and Nicolai Vorobjov. Complexity of computations with pfaffian and noetherian functions. *Normal forms, bifurcations and finiteness problems in differential equations*, pages 211–250, 2004.

25 Aleksandr Gelfond. *Calculus of finite differences.* Hindustan Publ. Corp., 1971.

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

27 Daniel S. Graça. Some recent developments on Shannon's General Purpose Analog Computer. *Mathematical Logic Quaterly*, 50(4-5):473–485, 2004.

28 Daniel S. Graça, Manuel L. Campagnolo, and J. Buescu. Computability with polynomial differential equations. *Advances in Applied Mathematics*, 2007. To appear.

29 Daniel S. Graça and José Félix Costa. Analog computers and recursive functions over the reals. *Journal of Complexity*, 19(5):644–664, 2003.

30 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.

31 Otto Hölder. Über die eigenschaft der gamma funktion keiner algebraische differentialgleichung zu genügen. *Math. Ann.*, 28:1–13, 1887.

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

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

34 Lenoard. Lipshitz and Lee A. Rubel. A differentially algebraic replacement theorem, and analog computability. *Proceedings of the American Mathematical Society*, 99(2):367–372, 1987.

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

36 André Platzer and Yong Kiam Tan. Differential equation axiomatization: The impressive power of differential ghosts. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 819–828. ACM, 2018.

37 Marian B. Pour-El. Abstract computability and its relation to the general purpose analog computer (some connections between logic, differential equations and analog computers). *Transactions of the American Mathematical Society*, 199:1–28, 1974.

38 L. A. Rubel. A survey of transcendentally transcendental functions. *American Mathematical Monthly*, 96(9):777–788, 1989.

39 Lee A. Rubel. A universal differential equation. *Bulletin of the American Mathematical Society*, 4(3):345–349, 1981.

40 Keijo Ruohonen. Undecidability of event detection for ODEs. *Journal of Information Processing and Cybernetics*, 29:101–113, 1993.

41 Keijo Ruohonen. Event detection for ODEs and nonrecursive hierarchies. In *Proceedings of the Colloquium in Honor of Arto Salomaa. Results and Trends in Theoretical Computer Science (Graz, Austria, June 10-11, 1994)*, volume 812 of *Lecture Notes in Computer Science*, pages 358–371. Springer-Verlag, Berlin, 1994. URL: `http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743&volume=812&spage=358`.

42 Claude E. Shannon. Mathematical theory of the differential analyser. *Journal of Mathematics and Physics MIT*, 20:337–354, 1941.

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