Polynomial differential equations compute all real computable functions on computable compact intervals ⋆

Olivier Bournez [a,b,*] , Manuel L. Campagnolo [c,d] ,
Daniel S. Graça [e,d] , Emmanuel Hainry [f,b]

[a] *Inria Lorraine, France*

[b] *LORIA (UMR 7503 CNRS-INPL-INRIA-Nancy2-UHP), Campus scientifique, BP 239, 54506 Vandœuvre-Lès-Nancy, France*

[c] *DM/ISA, Technical University of Lisbon, 1349-017 Lisboa, Portugal*

[d] *SQIG/IT, Technical University of Lisbon, 1049-001 Lisboa, Portugal*

[e] *DM/FCT, Universidade do Algarve, C. Gambelas, 8005-139 Faro, Portugal*

[f] *Institut National Polytechnique de Lorraine, France*

Abstract

In the last decade, the field of analog computation has experienced renewed interest. In particular, there have been several attempts to understand which relations exist between the many models of analog computation. Unfortunately, most models are not equivalent.

It is known that Euler's Gamma function is computable according to computable analysis, while it cannot be generated by Shannon's General Purpose Analog Computer (GPAC). This example has often been used to argue that the GPAC is less powerful than digital computation.

However, as we will demonstrate, when computability with GPACs is not restricted to real-time generation of functions, we obtain two equivalent models of analog computation.

Using this approach, it has been shown recently that the Gamma function becomes computable by a GPAC [1]. Here we extend this result by showing that, in an appropriate framework, the GPAC and computable analysis are actually equivalent from the computability point of view, at least in compact intervals. Since GPACs are equivalent to systems of polynomial differential equations then we show that all real computable functions over compact intervals can be defined by such models.

Preprint submitted to Elsevier Science *14 October 2008*

1 Introduction

According to the Church-Turing thesis, all "reasonable models" of digital computation, based on the intuitive notion of algorithm, are computationally equivalent to the Turing machine.

No similar result is known for analog computation. While many analog models have been studied including the BSS model [2], Moore's $\mathbb{R}$-recursive functions [3], neural networks [4], or computable analysis [5–7], but none was able to affirm itself as "universal". In part, this is due to the fact that few relations between them are known. Moreover some of the known results assert that these models are not equivalent, making the idea of a Church-Turing thesis for analog models an apparently unreachable goal. For example the BSS model allows discontinuous functions while only continuous functions can be computed in the framework of computable analysis [7].

Here, we will show that this goal may not be as far as those results suggest. Indeed, we will prove the equivalence of two models of real computation that were previously considered nonequivalent: computable analysis and Shannon's General Purpose Analog Computer (GPAC). However, this result is only true when considering a variant of the GPAC that is nevertheless defined in a natural way.

The GPAC was introduced in 1941 by Shannon [8] as a mathematical model of an analog device: the Differential Analyzer [9]. The Differential Analyzer was used from the 1930s to the early 60s to solve numerical problems. For example, differential equations were used to solve ballistics problems. These devices were first built with mechanical components and later evolved to electronic versions. A GPAC may be seen as a circuit built of interconnected black boxes, whose behavior is given by Figure 1, where inputs are functions of an independent variable called the *time* (in an electronic Differential Analyzer, inputs usually correspond to electronic voltages). These black boxes add or multiply two inputs, generate a constant, or solve a particular kind of Initial Value Problem defined with an Ordinary Differential Equation (ODE for short).

While many of the usual real functions are known to be generated by a GPAC,

---

⋆ Expanded version of the article "The General Purpose Analog Computer and Computable Analysis are two equivalent paradigms of analog computation" presented in the Theory and Applications of Models of Computation conference (TAMC06).

* Corresponding author.
Email addresses: Olivier.Bournez@loria.fr (Olivier Bournez), mlc@math.isa.utl.pt (Manuel L. Campagnolo), dgraca@ualg.pt (Daniel S. Graça), Emmanuel.Hainry@loria.fr (Emmanuel Hainry).

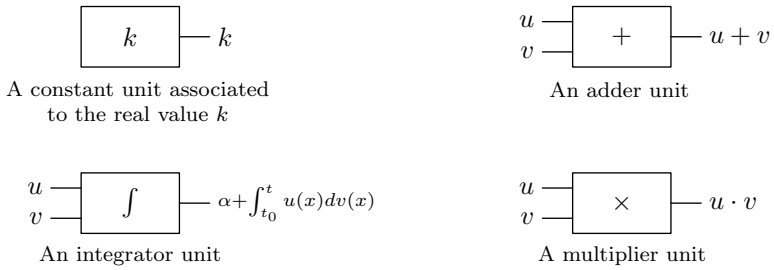


Fig. 1. Different types of units used in a GPAC.

a notable exception is the Gamma function $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt$ [8]. If we have in mind that this function is known to be computable under the computable analysis framework [5], the previous result has long been interpreted as evidence that the GPAC is a somewhat weaker model than computable analysis.

However, we believe that this limitation is due to the notion of GPAC-computability rather than the model itself.

Indeed, one assumes usually that GPAC computes in "real time" - a very restrictive form of computation. But if we change this notion of computability to the kind of "converging computation" used in recursive analysis, then it has been shown recently that the Γ function becomes computable [1]. Notice that this "converging computation" with GPACs corresponds to a particular class of $\mathbb{R}$-recursive functions [3,10,11]. As in [11] we only consider a Turing-computable subclass of $\mathbb{R}$-recursive functions, but here we restrict our focus to functions that can be defined as limits of solutions of polynomial differential equations.

In the present paper, we further strengthen this result and show that actually every *computable function* defined over a compact interval can be computed by a GPAC in the above sense.[1] Reciprocally, we show that under some reasonable hypotheses, the converse is also true.

In other words, we prove that (non-real time) GPAC computability coincides with computability according to recursive analysis, over computable compact domains. This is a significantly stronger result than Shannon's approximation of real continuous functions with the GPAC [8].

It is worth noting that it was shown in [12] that Turing machines can be simulated by GPACs. Since real computable functions are those computed by function-oracle Turing machines [6], this paper also shows that the result in [12] can be extended to oracle Turing machines.

The outline of this paper is as follows. In Section 2 we describe the GPAC and

[1] If not otherwise stated, the expression "computable function" is interpreted in the computable analysis sense.

we recall that GPACs are equivalent to systems of polynomial ordinary differential equations. The constructions in the remainder of the paper will rely on such systems, which are explicitly continuous-time in nature. Then, we define a notion of "converging computation", which we call GPAC-computability in opposition to the original "real-time" notion of GPAC-generability. We also recall the definition of computable real functions according to Computable Analysis. To conclude the preliminaries, we review how Turing machines can be simulated with ODEs. In Section 3 we state the main result of the paper on the equivalence between computable real functions and GPAC-computable functions over compact domains. In Section 4 we present some preliminary results. In particular, we show that GPACs can simulate oracle Turing machines. Finally, in Sections 5 and 6, we prove the main result of the paper. Since this result is about an equivalence, Section 5 proves the "only if" direction, while Section 6 proves the "if" direction.

2 Preliminaries

2.1 The GPAC

The GPAC was originally introduced by Shannon in [8], and further refined in [13–15,1]. The model basically consists of families of circuits built with the basic units presented in Figure 1, not all kinds of interconnections are allowed since this may lead to undesirable behavior (*e.g.* non-unique outputs. For further details, refer to [15]).

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.

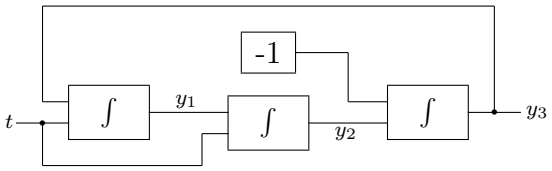
Definition 1 *A unary function y is differentially algebraic (d.a.) on the interval I if there exists an $n \in \mathbb{N}$ and a nonzero polynomial p with real coefficients such that*

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

As a corollary, and noting that the Gamma function $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt$ is not d.a. [16], we get that

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

However, Shannon's proof relating functions generated by GPACs with d.a.



$$\begin{cases} y_1' = y_3 & \& \ y_1(0) = 1 \\ y_2' = y_1 & \& \ y_2(0) = 0 \\ y_3' = -y_1 & \& \ y_3(0) = 0 \end{cases}$$

Fig. 2. Generating cos and sin via a GPAC: circuit version on the left and ODE version on the right. One has $y_1 = \cos$, $y_2 = \sin$, $y_3 = -\sin$.

functions was incomplete (as pointed out and partially corrected in [13,14]). Actually, as pointed out in [15], the original GPAC model suffers from several robustness problems. However, for the more robust class of GPACs defined in [15] by restricting the possible layout of a GPAC, the following stronger property holds:

Proposition 3 *A scalar function $f : \mathbb{R} \to \mathbb{R}$ is generated by a GPAC iff it is a component of the solution of a system*

$$y' = p(y, t), \tag{2}$$

where p is a vector of polynomials. A function $f : \mathbb{R} \to \mathbb{R}^k$ is generated by a GPAC iff all of its components are.

For a concrete example of Proposition 3, see Figure 2. From now on, we will mostly talk about GPACs as being systems of ODEs of the type (2). Functions generated by GPACs have a number of interesting properties. The following results are taken from [17] and will be used later, in an explicit or implicit manner.

Proposition 4 ([17]) *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 ([17]) *Consider the initial value problem (IVP)*

$$\begin{cases} x' = f(t, x), \\ x(t_0) = x_0, \end{cases} \tag{3}$$

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 (3) is given by the first n components of $y = (y_1, ..., y_m)$, where y is the solution of the polynomial

IVP

$$\begin{cases} y' = p(t, y), \\ y(t_0) = y_0. \end{cases} \tag{4}$$

We now set some useful notations.

Definition 6 *In the conditions of Proposition 3, we say that t is the input of the GPAC and that the components $y_1, \ldots, y_n$ of the solution are the outputs of the GPAC.*

Notice that GPAC generable functions are obviously d.a.. Another interesting consequence is the following (recall that solutions of analytic ODEs are always analytic – *cf.* [18]):

Corollary 7 *If f is a function generated by a GPAC, then it is real analytic.*

As one can see, GPAC generation refers to a notion of "real-time" computation. To follow the idea of a "converging" computation, as the one used in recursive analysis, we now introduce the idea of GPAC computability from [1] (Notice that in Shannon's original definition of the GPAC nothing is assumed about the constants and initial conditions of the ODE (2). In particular, there can be non-computable reals. This kind of GPAC can trivially lead to super-Turing computations. To avoid this, the model of [1] is actually reinforced here):

Definition 8 *A function $f : [a, b] \to \mathbb{R}$ is GPAC-computable*[2] *iff there exist some computable polynomials*[3] *$p : \mathbb{R}^{n+1} \to \mathbb{R}^n$, $p_0 : \mathbb{R} \to \mathbb{R}$, 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 $t \in [0, +\infty)$. [4]

We remark that $\alpha_1, \ldots, \alpha_{n-1}$ are auxiliary parameters needed to compute f. We will also use the following notation. If we are given the GPAC (2), with initial condition $(\alpha_1, ..., \alpha_{n-1}, p_0(x))$ set at time $t_0 = 0$, where $\alpha_1, ..., \alpha_{n-1}$ are

[2] Note that in this paper, the term GPAC-computability refers to this particular notion. The expression "generated by a GPAC" corresponds to Shannon's notion of computability.
[3] The notion of computable function and computable real will be provided in the next section.
[4] 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.

fixed computable values, and where x may vary for each computation, as in Definition (8), we say that the *initial condition x sets the output of the GPAC* (2).

Proposition 9 ([1]) *The Gamma function Γ is GPAC-computable.*

In this paper, we show that for compact domains, GPAC-computable functions are precisely the computable functions in the sense of computable analysis.

2.2 Computable Analysis

Recursive analysis, or *computable analysis*, was introduced by Turing [19], Grzegorczyk [20], and Lacombe [21].

The idea underlying computable analysis is to extend the classical computability theory so that it can deal with real quantities. See [7] for an up-to-date monograph of computable analysis from the computability point of view, or [6] for a presentation from a complexity point of view.

In this approach, informally, a function $f : \mathbb{R} \to \mathbb{R}$ is computable if there is a computer program that does the following. Let $x \in \mathbb{R}$ be an arbitrary element in the domain of f. Given an output precision 2^{-n}, the program has to compute a rational approximation of $f(x)$ with precision 2^{-n}.

To formalize this notion, we need oracle TMs. We say that M is an oracle TM if, at any step of the computation of M using oracle $\phi : \mathbb{N} \to \mathbb{N}^k$, M is allowed to query the value $\phi(n)$ for any n written on its tape.

The following definition is from [5,7], but is slightly adapted to match the approach of Ko91, so that we can make use of oracle TMS, as described in Section 9.4 of [7].

Definition 10 *(1) A sequence $\{r_n\}$ of rational numbers is called a ρ-name of a real number x if there are three functions a, b and c from $\mathbb{N}$ to $\mathbb{N}$ such that for all $n \in \mathbb{N}$, $r_n = (-1)^{a(n)} \frac{b(n)}{c(n)+1}$ and*

$$|r_n - x| \leq \frac{1}{2^n}. \tag{5}$$

(2) A real number x is called computable if it has a computable ρ-name, i.e., if a, b and c in (5) are computable (recursive) functions.
(3) A sequence $\{x_k\}_{k\in\mathbb{N}}$ of real numbers is computable if there are three com-

putable functions a, b, c from $\mathbb{N}^2$ to $\mathbb{N}$ such that, for all $k, n \in \mathbb{N}$,

$$\left|(-1)^{a(k,n)}\frac{b(k,n)}{c(k,n)+1} - x_k\right| \leq \frac{1}{2^n}.$$

The notion of the ρ-name can be extended to points in $\mathbb{R}^l$ as follows: a sequence $\{(r_{1n}, r_{2n}, \ldots, r_{ln})\}_{n\in\mathbb{N}}$ of rational vectors is called a ρ-name of $x = (x_1, x_2, \ldots, x_l) \in \mathbb{R}^l$ if $\{r_{jn}\}_{n\in\mathbb{N}}$ is a ρ-name of x_j, $1 \leq j \leq l$. Similarly, one can define computable points and sequences over $\mathbb{R}^l$, $l > 1$, by assuming that each component is computable. Next we present a notion of computability for open and closed subsets of $\mathbb{R}^l$ (cf. [7], Definition 5.1.15).

Definition 11 *(1) An open set $E \subseteq \mathbb{R}^l$ is called recursively enumerable (r.e. for short) open if there are computable sequences $\{a_n\}$ and $\{r_n\}$, $a_n \in E \cap \mathbb{Q}^l$ and $r_n \in \mathbb{Q}$ such that*

$$E = \cup_{n=0}^{\infty} B(a_n, r_n).$$

Without loss of generality one can also suppose that for any $n \in \mathbb{N}$, the closure of $B(a_n, r_n)$, denoted as $\overline{B(a_n, r_n)}$, is contained in E, where $B(a_n, r_n) = \{x \in \mathbb{R}^l : |x - a_n| < r_n\}$.

(2) A closed subset $K \subseteq \mathbb{R}^l$ is called r.e. closed if there exist computable sequences $\{b_n\}$ and $\{s_n\}$, $b_n \in \mathbb{Q}^l$ and $s_n \in \mathbb{Q}$, such that $\{B(b_n, s_n)\}_{n\in\mathbb{N}}$ enumerates all rational open balls intersecting K.

(3) An open set $E \subseteq \mathbb{R}^l$ is called computable (or recursive) if E is r.e. open and its complement E^c is r.e. closed. Similarly, a closed set $K \subseteq \mathbb{R}^l$ is called computable (or recursive) if K is r.e. closed and its complement K^c is r.e. open.

It is well known [7, Example 5.1.17] that an open interval $(\alpha, \beta) \subseteq \mathbb{R}$ or a closed interval $[\alpha, \beta]$ is computable if and only if α and β are computable real numbers. Having defined the notion of recursive and r.e. sets, we are now ready to introduce the notion of computable functions defined on those sets [5–7].

Definition 12 *Let $A \subseteq \mathbb{R}^l$ be either a r.e. open set or a r.e. closed set. A function $f : A \to \mathbb{R}^m$ is computable if there is an oracle Turing machine such that for any input $n \in \mathbb{N}$ (accuracy) and any ρ-name of $x \in A$ given as an oracle, the machine outputs a rational vector r satisfying $|r - f(x)| \leq 2^{-n}$.*

The following result is a straightforward adaptation of Corollary 2.14 from [6].

Proposition 13 *A real function $f : [a, b] \to \mathbb{R}$, with a and b computable, is computable iff there exist three computable functions $m : \mathbb{N} \to \mathbb{N}$, $sgn, abs : \mathbb{N}^4 \to \mathbb{N}$ such that:*

(1) m is a modulus of continuity for f, i.e. for all $n \in \mathbb{N}$ and all $x, y \in [a, b]$, one has

$$|x - y| \leq 2^{-m(n)} \quad \Longrightarrow \quad |f(x) - f(y)| \leq 2^{-n}$$

(2) For all $(i, j, k) \in \mathbb{N}^3$ such that $(-1)^i j/2^k \in [a, b]$, and all $n \in \mathbb{N}$,

$$\left|(-1)^{sgn(i,j,k,n)} \frac{abs(i, j, k, n)}{2^n} - f\left((-1)^i \frac{j}{2^k}\right)\right| \leq 2^{-n}.$$

2.3 Simulating TMs with ODEs

To prove the main result of this paper, we need to simulate a TM with differential equations. To this end, we recall in this section some results from [12].

For simplicity, and without loss of generality, we only consider Turing machines using 10 symbols and set the following coding. Let M be some one tape Turing machine, with m states and 10 symbols. Then to each state we associate a number in $\{1, ..., m\}$ and to each symbol we associate a number in $\{0, ..., 9\}$, assuming that the blank symbol corresponds to the 0. If

$$...B\ B\ B\ a_{-k}\ a_{-k+1} ...\ a_{-1}\ a_0\ a_1 ...\ a_n\ B\ B\ B ...$$

is the tape content of M, where a_0 is the currently scanned symbol, then it can be coded in the following two integers:

$$y_1 = a_0 + a_1 10 + ... + a_n 10^n, \qquad y_2 = a_{-1} + a_{-2} 10 + ... + a_{-k} 10^{k-1}. \tag{6}$$

The configuration of M is then given by its state s, and two integers y_1 and y_2. The transition function of M corresponds therefore to a function $\psi_M : \mathbb{N}^3 \to \mathbb{N}^3$ (we consider that if $x_0 \in \mathbb{N}^3$ is an halting configuration, then $\psi(x_0) = x_0$, i.e. x_0 is a fixed point). In [12] it is shown that ψ_M admits a *robust* extension to $\mathbb{R}^3$ and that this extension can be written as the composition of polynomials, the exponential, the trigonometric functions, and their inverses (such a function will be termed closed-form). In what follows, $\| \cdot \|_\infty$ stands for the sup-norm: $\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$ and $\psi^{[j]}(x_0)$ stands for jth iteration of function ψ on x_0: $\psi^{[0]}(x_0) = x_0$, and $\psi^{[j+1]}(x_0) = \psi(\psi^{[j]}(x_0))$.

Proposition 14 ([12]) *Let $\psi_M : \mathbb{N}^3 \to \mathbb{N}^3$ be the transition function of a Turing machine M, under the encoding described above and let $0 < \varepsilon < 1/2$. Then ψ_M admits a computable closed-form extension $\Psi_M : \mathbb{R}^3 \to \mathbb{R}^3$, robust to perturbations in the following sense: for all $j \in \mathbb{N}$, and for all $\bar{x}_0 \in \mathbb{R}^3$*

satisfying $\|\bar{x}_0 - x_0\|_\infty \leq \varepsilon$, *where* $x_0 \in \mathbb{N}^3$ *represents an initial configuration,*

$$\left\|\Psi^{[j]}(\bar{x}_0) - \psi^{[j]}(x_0)\right\|_\infty \leq \varepsilon.$$

More generally, if M has l tapes, then its transition function is defined over $\mathbb{N}^{2l+1}$ and also admits a closed-form robust extension to $\mathbb{R}^{2l+1}$.

The following result is an adaptation of Theorem 4 from [12] and shows that GPACs can iterate the transition function of a given Turing machine.

Proposition 15 ([12]) *Suppose that* $\psi_M : \mathbb{N}^{2l+1} \to \mathbb{N}^{2l+1}$ *is the transition function of a Turing machine M, under the encoding presented in Equation (6),* $x_0 \in \mathbb{N}^{2l+1}$ *represents an initial configuration and* $\varepsilon, \delta > 0$ *are constants satisfying* $\varepsilon + \delta < 1/2$. *Then there is a computable polynomial* p *and some computable value* $\alpha \in \mathbb{R}^n$

$$z' = p(z,t), \qquad z(0) = (\tilde{x}_0, \alpha)$$

such that for all $\tilde{x}_0 \in \mathbb{R}^{2l+1}$ *satisfying* $\|\tilde{x}_0 - x_0\|_\infty \leq \varepsilon$, *one has*[5]

$$\left\|z_1(t) - \psi_M^{[j]}(x_0)\right\|_\infty \leq \delta.$$

for all $j \in \mathbb{N}$ *and for all* $t \in [j, j + 1/2]$.

If there exists some computable value $\alpha \in \mathbb{R}^n$ such that $z' = p(z,t)$ has the properties described in Proposition 15, we say that the GPAC $z' = p(z,t)$ simulates the Turing machine M on input x.

Later we will use this result and, for that reason, it is convenient to survey some ideas underlying its proof. In particular, let $\psi : \mathbb{N} \to \mathbb{N}$ be a function over the integers that admits a closed-form extension $\Psi : \mathbb{R} \to \mathbb{R}$ to the reals. We would like to iterate ψ with a system of analytic ODEs. In [12] it is shown that this can be done (a more detailed analysis can be found in [17]) with a system of the type

$$\begin{aligned} y_1' &= f_1(\Psi(z_1), y_1, t) \\ z_1' &= g_1(y_1, z_1, t) \end{aligned} \tag{7}$$

where $f_1, g_1 : \mathbb{R}^3 \to \mathbb{R}$ are computable closed-form functions and Ψ is supposed to be robust in the following manner: for all $n \in \mathbb{N}$

$$|x - n| < \delta \quad \Rightarrow \quad |\Psi(x) - \Psi(n)| < \delta.$$

[5] For simplicity, we denote the solution z of the initial-value problem by (z_1, z_2), where $z_1 \in \mathbb{R}^{2l+1}$ and $z_2 \in \mathbb{R}^n$.

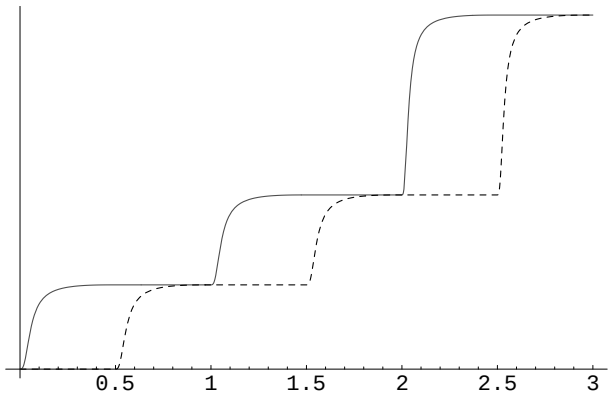


Fig. 3. Simulation of the iteration of a map ψ via ODEs. The solid line represents the variable y_1 and the dashed line represents z_1.

The ODE (7) can be shown to be equivalent to a (larger) polynomial ODE (cf. Proposition 5). Ideally, the variables y_1, z_1 have the following behavior on an interval $[n, n+1]$, where $n \in \mathbb{N}$ (cf. Fig. 3). On $[n, n+1/2]$, variable z_1 is kept constant to the value $\psi^{[n]}(x_0)$. This kind of "memory" is then used by the first equation of (7) to update the variable y_1 to the value $\psi^{[n+1]}(x_0)$. In the following next half-interval $[n+1/2, n+1]$, the roles of y_1 and z_1 are switched: y_1 is kept constant to the value $\psi^{[n+1]}(x_0)$ and z_1 is updated to this value.

However, this is only an ideal behavior since real analytic functions cannot be constant in an interval without be constant everywhere. Instead, the functions f_1, g_1 are defined so that the derivatives of y_1 and z_1 are kept sufficiently close to zero when their respective values should be kept constant. These functions can be defined to be computable and closed-form [12], [17]. A similar result can be obtained for the case of an $2l+1$-dimensional map ψ and, in particular, to the case of Turing machines (use the map Ψ given by Proposition 14).

Finally, we want to read the value $\psi^{[n]}(x_0)$ from (7) with precision bounded by δ. As we mentioned earlier, in the time interval $[n, n+1/2]$, z_1 is kept close to the value $\psi^{[n]}(x_0)$. In particular, it can be shown in the constructions from [12] that there is some $\eta < 1/2$ such that, for $t \in [n, n+1/2]$, $\|z_1(t) - \psi^{[n]}(x_0)\| \leq \eta$. Using the error-contracting function σ defined in the following result of [12]

Proposition 16 *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be defined by*

$$\sigma(x) = x - 0.2\sin(2\pi x). \tag{8}$$

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

We see that it is enough to apply σ a fixed number of times k to the variable z_1 to get the desired accuracy in the interval $[n, n+1/2]$ (just pick some k satisfying $\sigma^{[k]}(\eta) \leq \delta$). This still can be obtained as the solution of a polynomial

ODE.

Refer to [12] for full details.

3 The result

The main result of this paper relates computable analysis with the GPAC, showing their equivalence in the framework described in the previous section.

Theorem 17 (Main result) *Let a and b be computable reals. A function $f : [a, b] \to \mathbb{R}$ is computable iff it is GPAC-computable.*

We postpone the proof of this result to Sections 5 and 6.

4 Simulating Type-2 machines with GPACs

We present in this section a result that shows that GPACs can simulate oracle Turing machines, under a suitable encoding. This will be necessary to prove Theorem 17.

From Proposition 15, we know how to simulate a Turing machine. However, the error of the output is bounded by some fixed quantity $\varepsilon > 0$, whereas in Type-2 machines we would like that the output is given with error bounded by 2^{-n}, where n is one of the inputs of the machine. The next theorem shows how this can be done with a GPAC.

Theorem 18 *Let $f : [a, b] \to \mathbb{R}$ be a computable function. Then there exists a GPAC and some index i such that if we set the initial conditions $(x, \bar{n}) \in [a, b] \times \mathbb{R}$, where $|\bar{n} - n| \leq \varepsilon < 1/2$, with $n \in \mathbb{N}$, there exists some $T \geq 0$ such that the output y_i of the GPAC satisfies $|y_i(t) - f(x)| \leq 2^{-n}$ for all $t \geq T$.*

Before giving the proof of the theorem, we provide some preliminary lemmas. To compute $f(x)$ with a GPAC, we want to use the hypothesis that f is computable. Hence, it would be useful to get a GPAC that, when we set the initial condition $x \in [a, b]$, outputs a succession of rationals converging to x. This succession could then be used to compute approximations of $f(x)$, as in condition 2 of Proposition 13. The problem is, given x, to get integers i, j, and 2^k such that $(-1)^i j/2^k$ approximates x enough to compute $f(x)$ with precision 2^{-n}, and to compute the values $sgn(i, j, k, n)$ and $abs(i, j, k, n)$.

We assume first in what follows that $[a, b] \subseteq \mathbb{R}^+$, so that x is always positive. It follows that i can be considered as constant 0. Now, from Proposition 13,

this is sufficient to take $k = m(n)$ (where m is a modulus of continuity), and $j \simeq x2^{m(n)}$.

In the following lemma, the function $g(j, n)$ is intended to represent function $abs(0, j, m(n), n)$.

By a barycenter of $x, y \in \mathbb{R}$ we mean a value of the form $tx + (1-t)y$, for $t \in [0, 1]$. By other words, a barycenter is a point in the segment of line joining x to y.

Lemma 19 *Let $g : \mathbb{N}^2 \to \mathbb{N}$ be a recursive function, $[a, b] \subseteq \mathbb{R}^+$ be a bounded interval, $m : \mathbb{N} \to \mathbb{N}$ be a recursive function, and ε be a real number satisfying $0 < \varepsilon < 1/4$. Then there is a GPAC with the following property: for all $x \in [a, b]$ and all j, $n \in \mathbb{N}$ satisfying $j \leq x2^{m(n)} < j + 1$, there exists some $T > 0$ and some index i such that, when we set initial conditions $\bar{n}$, $\overline{x2^{m(n)}}$, where $|n - \bar{n}| < \varepsilon$ and $\left|\overline{x2^{m(n)}} - x2^{m(n)}\right| < \varepsilon$, the output y_i of the GPAC satisfies $|y_i(t) - c| \leq \varepsilon$ for all $t \geq T$, where c is a barycenter of $g(j, n)$ and $g(j + 1, n)$.*

Proof. By Proposition 15 there is a GPAC $\mathcal{G}$ that when set on initial condition $\overline{k}$, $\bar{n}$, where $\left|k - \overline{k}\right|$, $|n - \overline{n}| < 1/3$ (the reason why we use $1/3$ will be clear later) and $k, n \in \mathbb{N}$, ultimately (i.e. at any time $t \geq T$ for some $T > 0$) outputs $g(k, n)$ with an error less than or equal to $\varepsilon/2$ (k is intended to be j or $j + 1$). Define $\overline{k}_1 = \overline{x2^{m(n)}}$. We would like to use $\overline{k} = \overline{k}_1$ as an initial condition to GPAC $\mathcal{G}$. However, $\overline{k}_1$ is not guaranteed to be close to an integer (i.e. within distance $1/3$). We show now how to overcome this. Let us consider two cases:

(1) If $\overline{k}_1 \in [l - 1/4, l + 1/4]$, for some $l \in \{j, j + 1\}$, then we can set $\overline{k} = \overline{k}_1$. With this initial condition, the output of GPAC $\mathcal{G}$ (let us call it y_1) will be $g(j, n)$ or $g(j + 1, n)$, plus an error not exceeding $\varepsilon/2$. Therefore, the output satisfies the conditions imposed by the lemma. Notice that this reasoning extends for the case $\overline{k}_1 \in [l - 1/3, l + 1/3]$, because (integer) initial conditions of the GPAC can be perturbed by an amount bounded by $1/3$;
(2) If $\overline{k}_1 \in [j+1/4, j+3/4]$, then we can set the initial condition $\overline{k} = \overline{k}_1 - 1/2$. With this initial condition, the output of GPAC $\mathcal{G}$ (let us call it y_2) will be $g(j, n)$ plus an error not exceeding $\varepsilon/2$. Therefore, the output satisfies the conditions imposed by the lemma. Notice that this reasoning extends for the case $\overline{k}_1 \in [l + 1/6, l + 5/6]$.

The real problem here is to implement both cases in a single GPAC. Since GPACs do not allow the existence of discontinuous functions that might work like a "case checker," we have to resort to a different approach.

From the study of the previous cases, we know the following: there is a GPAC

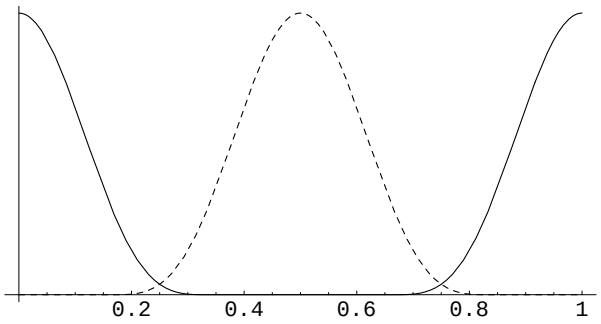


Fig. 4. Functions ω_1 and ω_2. The solid line represents ω_1, while the dashed line represents ω_2.

$\mathcal{G}$ such that on initial conditions $\overline{k}_1$ or $\overline{k}_1 - 1/2$, outputs y_1 or y_2, respectively. We now consider a new GPAC, obtained with two copies of $\mathcal{G}$, but where one copy has initial condition set to $\overline{k}_1$, and the other has initial condition set to $\overline{k}_1 - 1/2$ (from the comments following Definition 8, both cases are covered by the expression "each copy has initial condition set to $\overline{k}_1$"). Hence, this GPAC outputs both y_1 and y_2. We now combine these outputs to get the desired result.

Assume we had two periodic functions ω_1 and ω_2, with period 1 and graphs similar to the ones depicted in Fig. 4. We do not explicitly define ω_1 and ω_2, but rather state their most important properties: (i) for every $t \in \mathbb{R}$, $\omega_1(t) \geq 0$, $\omega_2(t) \geq 0$, and $\omega_1(t)+\omega_2(t) > 0$, (ii) $\omega_1(t) > 0$ implies that $t \in (a-1/3, a+1/3)$ for some $a \in \mathbb{N}$, and (iii) $\omega_2(t) > 0$ implies that $t \in (a + 1/6, a + 5/6)$ for some $a \in \mathbb{N}$. Remark that, for all $a \in \mathbb{N}$, (ii) implies $\omega_1(t) = 0$ for all $t \in (a + 1/3, a + 2/3)$, and (iii) implies $\omega_2(t) = 0$ for all $t \in (a - 1/6, a + 1/6)$. Then, taking into account the previous two cases described above, one sees that we could output the value

$$\bar{y} = \frac{\omega_1(\overline{k}_1)y_1 + \omega_2(\overline{k}_1)y_2}{\omega_1(\overline{k}_1) + \omega_2(\overline{k}_1)}. \tag{9}$$

that would be correct in any of the two cases above. Indeed, the only case where both $\omega_1(\overline{k}_1)$ and $\omega_2(\overline{k}_1)$ are non-null is whenever $\overline{k}_1 \in [l + 2/3, l + 5/6]$, where both outputs y_1 and y_2 are valid, and the result is a barycenter of y_1 and y_2, i.e. a barycenter of $g(j, n)$ and $g(j + 1, n)$ plus an error not exceeding $\varepsilon/2$.

However, ω_1 and ω_2 are not GPAC-generable since they are not analytic. Alternatively, we will use closed-form functions that approximate ω_1 and ω_2. In particular, we use the function l_2 defined in [12] as below, with the following property.

Proposition 20 ([12]) *Let $l_2 : \mathbb{R}^2 \to \mathbb{R}$ be given by $l_2(x, y) = \frac{1}{\pi} \arctan(4y(x - 1/2)) + \frac{1}{2}$. For $y > 0$ one has:*

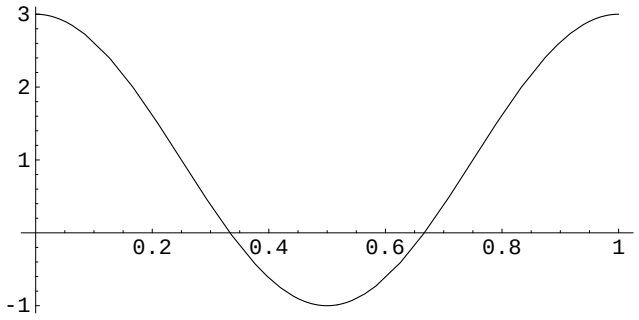


Fig. 5. Function Υ.

(1) If $x \leq 1/4$, *then* $0 < l_2(x, y) < 1/y$;
(2) If $x \geq 3/4$, *then* $1 - 1/y < l_2(\bar{a}, y) < 1$.

We also use the periodic function $\Upsilon : \mathbb{R} \to \mathbb{R}$ defined by

$$\Upsilon(x) = 1 + 2\sin 2\pi(x + 1/4)$$

with period 1 and whose graph is depicted in Fig. 5. Notice that for $x \in [1/3, 2/3]$, $\Upsilon(x) \leq 0$, and for $x \in [-1/4, 1/4]$, $\Upsilon(x) \geq 1$. Therefore, we can take

$$\overline{\omega_1}(x) = l_2(\Upsilon(x), 1/\delta) \simeq \omega_1(x), \quad \overline{\omega_2}(x) = l_2(\Upsilon(x - 1/2), 1/\delta) \simeq \omega_2(x)$$

since $|\omega_1(x) - \overline{\omega_1}(x)| \leq \delta$, for $x \in [a + 1/3, a + 2/3]$, where $a \in \mathbb{Z}$, and similarly for ω_2. Moreover, $|\overline{\omega_1}(x) - 1| \leq \delta$ for $x \in [a - 1/4, a + 1/4]$, and similarly for ω_2, which implies that

$$\overline{\omega_1}(t) + \overline{\omega_2}(t) > 1 - 2\delta \gg 0,$$

for all $t \in \mathbb{R}$ (i.e. the magnitude of $\overline{\omega_1}(t) + \overline{\omega_2}(t)$ is not comparable to that of δ, to avoid problems). Now, we just have to substitute (9) by

$$\bar{y} \simeq \frac{\overline{\omega_1}(\overline{k}_1)y_1 + \overline{\omega_2}(\overline{k}_1)y_2}{\overline{\omega_1}(\overline{k}_1) + \overline{\omega_2}(\overline{k}_1)}. \tag{10}$$

If we pick $1/\delta = \gamma(y_1 + y_2 + 1)$, for some $\gamma > 0$ (the value 1 is to avoid a singularity for $y_1 = y_2 = 0$), we conclude that $l_2(\Upsilon(\overline{k}_1), 1/\delta)y_1$ approaches $\omega_1(\overline{k}_1)y_1$ with error bounded by γ, and similarly for the other term. Moreover, $\Upsilon(\overline{k}_1) + \Upsilon(\overline{k}_1 - 1/2) > 1 - 2\gamma$. This implies that $\bar{y}$ in (10) is computed with error bounded by $2\gamma/(1 - 2\gamma) < \varepsilon/2$ for γ sufficiently small. By other words, this yields a GPAC with an output y_i such that for some $T > 0$, $|y_i(t) - c| \leq \varepsilon$ for all $t \geq T$, where c is a barycenter of $g(j, n)$ and $g(j + 1, n)$. ■

In many occasions it will be useful to switch the behavior of a GPAC upon some "control function" $y : \mathbb{R} \to \mathbb{R}$ which is also the output of some GPAC. Ideally, we would like to have the situation pictured in Fig. 6, which illustrates

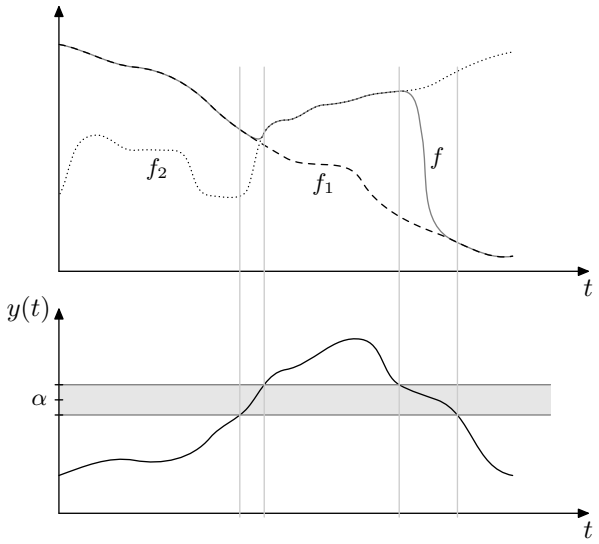


Fig. 6. Switching functions. Functions f_1 and f_2 are represented in the first graph by the dashed and dotted line, respectively. The resulting function is represented in gray. The second graph displays the control function y, where α is the threshold.

a coupled system that behaves like a "switch". There one can see on the above graph two functions f_1 and f_2, generated by GPACs. The graph below represents the control function and a value $\alpha \in \mathbb{R}$ called the threshold value. Then we would like to have a GPAC with output z such that, if $y(t) < \alpha$, then $z(t) = f_1(t)$, and $z(t) = f_2(t)$ otherwise.

Of course, the previous idea cannot be implemented with a GPAC, since we allow immediate transitions between two distinct functions, which would yield a discontinuous function. To remedy that, we allow some transition zone around the threshold value (in gray in the second graph of Fig. 6).

The construction of switching functions has to be further relaxed to cope with the fact that only analytic functions can be used. Therefore, function z in the following lemma will just be an approximation of f_1 and f_2.

Lemma 21 (Switching functions) *Let $y, f_1, f_2 : \mathbb{R} \to \mathbb{R}$ be three functions generated by GPACs (y is called the control function) and $\varepsilon > 0$. Then there is a function $f : \mathbb{R} \to \mathbb{R}$ generated by a GPAC with the following property: for all $t \in \mathbb{R}$,*

$$\begin{cases} |f(t) - f_1(t)| \leq \varepsilon \quad \textit{if} \quad y(t) \leq \alpha - 1/4 \\ |f(t) - f_2(t)| \leq \varepsilon \quad \textit{if} \quad y(t) \geq \alpha + 1/4. \end{cases}$$

Proof. This can be done in a quite straightforward way using the function l_2 introduced in the proof of Lemma 19. It suffices to take

$$f = f_1.l_2\left(\alpha + 1/2 - y(t), \frac{f_1 + f_2}{\varepsilon/2}\right) + f_2.l_2\left(y(t) - \alpha + 1/2, \frac{f_1 + f_2}{\varepsilon/2}\right)$$

It is easy to see that f satisfies the given conditions. ■

We will mainly use this lemma to switch between dynamics simulating different Turing machines. Suppose that the GPACs $z_1' = p_1(t, z_1)$ and $z_2' = p_2(t, z_2)$ simulate two Turing machines TM_1 and TM_2 as in Proposition 15, respectively. Then if we have a control function $y_i(t)$, provided by the output of a third GPAC $y' = p(t, y)$, we can build a function f that switches between $f_1 = p_1$ and $f_2 = p_2$, yielding a new system $y'(t) = p(t, y(t))$, $z'(t) = f(t, z(t))$, simulating the transition function of TM_1 and TM_2, according to the value of $y_i(t)$. From Proposition 5, this corresponds to a GPAC.

An useful application will be to simulate two Turing machines TM_1 and TM_2 working in series, i.e. where the output of TM_1 is be used as the input of TM_2.

Indeed, when simulating TM_1 with a GPAC as in Proposition 15 we can suppose that the states are coded by the integers $1, \ldots, m$, where state m corresponds to the halting state, and that there is a variable y_i of the GPAC giving the current state of TM_1 in the simulation with error bounded by $1/4$ for every $t \in [n, n+1/2]$. Therefore, if we set $\alpha = m - 1/2$ and $y = y_i$ in Lemma 21, we have a way of switching between dynamic of a GPACs simulating TM_1 and another one simulating TM_2 upon the value of y_i, i.e. depending on whether TM_1 is still running, or already halted.

We can obtain a GPAC having the desired property in the following manner. The initial condition sets the input of TM_1 and this GPAC simulates TM_1 until it halts. When this happens, the variables coding the tape contents have the input of TM_2. Then we switch the evolution law of this GPAC so that now it simulates TM_2, thus giving the desired output (to avoid interference problems, the control function should be given by a separated GPAC that just simulates TM_1 and that stays in an halting configuration after this Turing machine halts).

From the robustness conditions of Proposition 15, by choosing ε sufficiently small in the lemma above, one can ensure that errors will stay controlled at each step, so that a correct simulation of TM_1 and then TM_2 will happen.

In some cases, we will need to switch to a dynamic that sets one variable to some value. This can be done with the following lemma, taken from [17].

Lemma 22 ([17] Resetting configurations) *Let $\varepsilon \in \mathbb{R}$ satisfy $1/4 \geq \varepsilon > 0$. Let y be some GPAC generated function and $t_0 < t_1$ be some reals.*

There is a polynomial p such that the solution of $z' = p(t, z, y)$ with some fixed initial condition at t_0 satisfies $\|z(t_1) - k\|_\infty < \varepsilon$, whenever $\|y(t) - k\|_\infty \leq \varepsilon$ for all $t \in [t_0, t_1]$ for some vector $k \in \mathbb{N}^m$.

Proof of Theorem 18. For simplicity, let us suppose that $a > 0$ so that we don't have to care about sign of $x \in [a, b]$. This is never problematic since, if $a < 0$, we can always shift $[a, b]$ by an amount $k \in \mathbb{N}$ such that $a+k > 0$, to define a new computable function $h : [a+k, b+k] \to \mathbb{R}$ satisfying $h(x) = f(x-k)$.

Suppose also that $f(x)$ always takes the same sign for all $x \in [a, b]$. The case where the sign of $f(x)$ switches can be reduced to this one. Indeed, since $[a, b]$ is compact, there is some $l \in \mathbb{Z}$ such that $g(x) = l + f(x) > 0$, for all $x \in [a, b]$. Once we have a GPAC computing $g(x)$, we just have to subtract l to the output to obtain a GPAC computing $f(x)$. So, to fix ideas, let us suppose that $f(x)$ always takes positive values.

Let us now proceed with the proof. Since f is computable, according to Proposition 13, and previous discussions, there are recursive functions $m : \mathbb{N} \to \mathbb{N}$, abs (the function sgn is no longer needed since $f(x)$ takes positive values) such that given $x \in [a, b]$ and non-negative integers j, n satisfying

$$\left| j/2^{m(n)} - x \right| < 2^{-m(n)}, \tag{11}$$

one has

$$\left| \frac{abs(0, j, m(n), n)}{2^n} - f(x) \right| < \frac{2}{2^n}.$$

We will design a GPAC with an output y_i such that, for some $T > 0$, for all $t \geq T$, $y_i(t)$ is always close to

$$\frac{abs(0, j, m(n), n)}{2^n}, \tag{12}$$

the error between two values being bounded by 2^{-n}. This will be sufficient to prove the theorem.

Let us show how we can compute (12) with a GPAC. Let TM_0 and TM_1 be Turing machines computing 2^n and $2^{m(n)}$ on input n. From Proposition 15, there are GPACs simulating these Turing machines. This yields two GPACs with outputs y_1 and y_2, so that on initial condition $\bar{n}$ close to n, one has $|y_1(t) - 2^n| \leq \varepsilon$ and $|y_2(t) - 2^{m(n)}| \leq \varepsilon$ for all $t \geq T_2$ for some T_2. Moreover, since $[a, b]$ is bounded, we can suppose that $|xy_2(t) - x2^{m(n)}| \leq \varepsilon$ (if necessary, apply σ a fixed number of times, independent of n, to y_2). The values $\bar{n}$ and xy_2 can then be used to feed the GPAC described in Lemma 19, with $g(j, n) = abs(0, j, m(n), n)$. This GPAC $\mathcal{U}_3$ has an output y_3, which, after some time T_3, yields a barycenter of $abs(0, j, m(n), n)$ and $abs(0, j+1, m(n), n)$ plus an error bounded by ε. Since m is a modulus of continuity,

$$|abs(0, j, m(n), n) - abs(0, j+1, m(n), n)| \leq 1.$$

This implies that the output of $\mathcal{U}_3$, let it call $\overline{abs(j,n)}$, satisfies

$$|y_3(t) - abs(0, j, m(n), n)| \leq 1 + \varepsilon$$

for all $t \geq T_3$. Since $[a, b]$ is bounded, there is some $\eta \in \mathbb{N}$ such that one has $abs(0, j, m(n), n) \leq 2^n\eta$ for all $n \in \mathbb{N}$ and all j satisfying (11) for every $x \in [a, b]$. Therefore

$$\left|\frac{y_3(t)}{y_1(t)} - \frac{abs(0, j, m(n), n)}{2^n}\right| \leq \frac{\varepsilon.abs(0, j, m(n), n) + 2^n(1 + \varepsilon)}{2^n(2^n - \varepsilon)} \leq$$
$$\leq \frac{\varepsilon\eta + 1 + \varepsilon}{2^n 2^{-1}} \leq \lambda 2^{-n} \tag{13}$$

with $\lambda = 2(\varepsilon\eta + 1 + \varepsilon)$ independent of j, n, for all $t \geq T_3$, thus giving an appropriate approximation of $abs(0, j, m(n), n)/2^n$, with error bounded by $\lambda 2^{-n}$, with λ independent of j, n. This proves the theorem. ■

Remark 23 *Notice that after the time T referred to in Theorem 18, the corresponding GPAC continues to output forever an approximation of $f(x)$ with error bounded by 2^{-n}. This is because, as assumed in Section 2.3, the halting configuration of the Turing machine simulated by the GPAC is a fixed point (modulo some error bounded by ε).*

5 Proof of the "only if" direction of Theorem 17

Our idea is the following. From Theorem 18, we already know how to generate $f(x)$ with precision 2^{-n} with a GPAC $\mathcal{G}$ fed with approximations of n and $x2^{m(n)}$ as initial conditions, say in components y_1 and y_2, respectively. Hence, to get a GPAC with an output converging to $f(x)$ in the limit, it suffices to implement the previous theorem in a cyclic way: start the computation with $n = 0$. When the computation finishes, increment n, repeat the computation, and so on.

To do so, we need to address several problems. The first one is to know when the computation with the current n is over. Indeed, Theorem 18 tells us that this happens after some time T, but does not give us any procedure to compute this instant T.

This can be solved by building another GPAC that provides a corresponding control function. Indeed, consider a clocking Turing machine TM_0 that basically simulates all involved Turing machines in the constructions of Theorem 18 on all possible arguments $x \in [a, b]$ of type $k2^n$ (that are finitely many). This guarantees that whenever TM_0 terminates on input n, we are sure that all involved Turing machines in the constructions of Theorem 18 have had

enough time to do their computations in GPAC $\mathcal{G}$, and so that the output is correct. The description of TM_0, on input $n \in \mathbb{N}$, is as follows:

(1) Compute the first $k_1 \in \mathbb{Z}$ such that $k_1/n \geq a$, and the last $k_2 \in \mathbb{Z}$ such that $k_2/n \leq b$
(2) For $i = k_1$ to k_2 simulate Turing machines involved in the proof of Theorem 18 on input (i, n).

Observe that Step 1 can be implemented because, by hypothesis, a and b are computable constants. The Turing machine TM_0 can be simulated by a GPAC (independent from GPAC $\mathcal{G}$). The output y_i of this GPAC that encodes the state of TM_0 can be used as a control function. Indeed, whenever it becomes greater than $m_0 - 1/2$, where m_0 is the number of states of TM_0, this means that TM_0 halted, and hence the output of $\mathcal{G}$ is correct. This solves our first problem.

Actually, we need to simulate $\mathcal{G}$ on increasing n. So, more precisely, we consider a Turing machine TM_1 that does the following:

(1) Start with $n = 0$
(2) Simulate TM_0 with input n
(3) Increment n and go to Step 2.

Suppose, without loss of generality, that this Turing machine has m_1 states, where m_1 is the halting state (that is never reached), and $m_1 - 1$ is a special state, only reached in the transition of Step 3 to Step 1. Value $m_1 - 3/2$ can then be used a threshold. Whenever the output encoding the state of TM_1, call it y_i, is higher than $m_1 - 3/2$, we know that the computation of $\mathcal{G}$ is over for corresponding n.

We can then use Lemmas 21 and 22 so that control function y_i resets the value of y_1 and y_2 to approximations of $n + 1$ and $x2^{m(n+1)}$, respectively, and begins a new cycle by allowing again the simulation of $\mathcal{G}$ on these new values for y_1 and y_2.

So far we have seen that while y_1 and y_2 are respectively approximations of n and $x2^m(n)$, one of the components of the system, say y_j, approaches $f(x)$ with error 2^{-n}. However, during the following time period, when n is incremented, y_j fluctuates before converging to $f(x)$ with error $2^{-(n+1)}$. Therefore, y_j doesn't match condition 2 of Definition 8 for all times.

To define a component of the system that converges to $f(x)$ with time, we use the components of the system that encode the values of $abs(0, j, m(n), n)$ and 2^n in each time period. We create a pair of components, say z_1, z_2, that are reset to the values of $abs(0, j, m(n), n)$ and 2^n respectively, at the end of each time period. This can be done with Lemma 22 using the control function y_i. More

precisely, when y_i is above the threshold, z_1, z_2 approach $abs(0, j, m(n), n)$ and 2^n. During the following time period, i.e. while y_i is below the threshold, z_1, z_2 are kept approximately constant. This can be done with Lemma 21 by switching the dynamics from our current system to a GPAC that simulates a Turing machine in a fixed configuration. Again, we use y_i as the control function.

Hence, a sufficient approximation of $f(x)$ is then given by z_1/z_2, and a bound on current error on $f(x)$, as required in Definition 8, is given by $1/z_2$ both values being valid at any time.

6 Proof of the "if" direction of Theorem 17

We now proceed with the proof of the "if" direction of Theorem 17. Let $f : [a, b] \to \mathbb{R}$ be a GPAC-computable function. We want to show that f is computable in the sense of computable analysis. By definition, we know that there is a computable polynomial ODE

$$\begin{cases} y' = p(t, y) \\ y(0) = (\alpha, x) \end{cases} \tag{14}$$

whose solution has two components $y_i : \mathbb{R}^2 \to \mathbb{R}$ and $y_j : \mathbb{R}^2 \to \mathbb{R}$ such that

$$|f(x) - y_i(x, t)| \leq y_j(x, t) \text{ and } \lim_{t \to \infty} y_j(x, t) = 0 \tag{15}$$

Since we have to study the computability of functions defined by ODEs, we resort to the following result, taken from [22].

Theorem 24 *Let $E \subseteq \mathbb{R}^{m+1}$ be a r.e. open set and $f : E \to \mathbb{R}^m$ be a computable analytic function. Let (α, β) be the maximal interval of existence of the solution $x(t)$ of the initial-value problem*

$$\begin{cases} \dot{x} = f(t, x), \\ x(t_0) = x_0, \end{cases} \tag{16}$$

where (t_0, x_0) is a computable point in E. Then (α, β) is a r.e. open interval and x is a computable function on (α, β). In particular, there is an oracle TM that on inputs (t_0, x_0) outputs the solution x.

Since for (14) $E = \mathbb{R}^{m+1}$ is r.e. and p is computable, we conclude that the solution is computable over the maximal interval, which is assumed in Definition 8 to include $(0, +\infty)$. It follows that y_i and y_j are computable in $(0, \infty)$.

Suppose that we want to compute $f(x)$ with precision 2^{-n}. Then proceed with the following algorithm.

(1) Set $t = 1$
(2) Compute an approximation $\bar{y}_j$ of $y_j(x, t)$ with precision $2^{-(n+2)}$
(3) If $\bar{y}_j > 2^{-(n+2)}$ then set $t := t + 1$ and go to Step 2
(4) Compute $y_i(x, t)$ with precision $2^{-(n+1)}$ and output the result

Steps 1, 2 and 3 are used to determine an integer value of t for which $|y_j(x, t)| \leq 2^{-(n+1)}$. Once this value is obtained, an approximation of $y_i(x, t)$ with precision $2^{-(n+1)}$ will provide an approximation of $f(x)$ with error 2^{-n}, due to (15), thus providing the desired output. This proves the result.

7 Conclusion

In this paper we established some links between computable analysis and Shannon's General Purpose Analog Computer. In particular, we showed that contrarily to what was previously suggested, the GPAC and computable analysis can be made equivalent from a computability point of view, as long as we take an adequate notion of computation for the GPAC. In addition to those results it would be interesting to answer the following questions. Is it possible to have similar results, but at a complexity level? For instance, using the framework of [6], is it possible to relate polynomially-time computable functions to a class of GPAC-computable functions where the error ε is given as a function of a polynomial of t? And if this is true, can this result be generalized to other classes of complexity? From the computability perspective, our results suggest that polynomial ODEs and GPACs are very natural continuous-time counterparts to Turing machines.

Acknowledgments. This work was partially supported by *Fundação para a Ciência e a Tecnologia* and EU FEDER POCTI/POCI via CLC, project ConTComp POCTI/MAT/45978 /2002, grant SFRH/BD/17436/2004 (DG), and within the initiative RealNComp of SQIG - IT, and by French Ministery of Research through ANR Project SOGEA. Additional support was also provided by the *Fundação Calouste Gulbenkian* through the *Programa Gulbenkian de Estímulo à Investigação*, and by EGIDE and GRICES under the Program *Pessoa* through the project *Calculabilité et complexité des modèles de calculs à temps continu.*

References

[1] D. S. Graça, Some recent developments on Shannon's General Purpose Analog Computer, Math. Log. Quart. 50 (4-5) (2004) 473–485.

[2] L. Blum, M. Shub, S. Smale, On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines, Bull. Amer. Math. Soc. 21 (1) (1989) 1–46.

[3] C. Moore, Recursion theory on the reals and continuous-time computation, Theoret. Comput. Sci. 162 (1996) 23–44.

[4] H. T. Siegelmann, Neural Networks and Analog Computation: Beyond the Turing Limit, Birkhäuser, 1999.

[5] M. B. Pour-El, J. I. Richards, Computability in Analysis and Physics, Springer, 1989.

[6] K.-I. Ko, Computational Complexity of Real Functions, Birkhäuser, 1991.

[7] K. Weihrauch, Computable Analysis: an Introduction, Springer, 2000.

[8] C. E. Shannon, Mathematical theory of the differential analyzer, J. Math. Phys. MIT 20 (1941) 337–354.

[9] V. Bush, The differential analyzer. A new machine for solving differential equations, J. Franklin Inst. 212 (1931) 447–488.

[10] J. Mycka, J. F. Costa, Real recursive functions and their hierarchy, J. Complexity 20 (6) (2004) 835–857.

[11] O. Bournez, E. Hainry, Recursive analysis characterized as a class of real recursive functions, to appear in Fund. Inform.

[12] D. S. Graça, M. L. Campagnolo, J. Buescu, Robust simulations of Turing machines with analytic maps and flows, in: S. B. Cooper, B. Löwe, L. Torenvliet (Eds.), CiE 2005: New Computational Paradigms, LNCS 3526, Springer, 2005, pp. 169–179.

[13] M. B. Pour-El, Abstract computability and its relations to the general purpose analog computer, Trans. Amer. Math. Soc. 199 (1974) 1–28.

[14] L. Lipshitz, L. A. Rubel, A differentially algebraic replacement theorem, and analog computability, Proc. Amer. Math. Soc. 99 (2) (1987) 367–372.

[15] D. S. Graça, J. F. Costa, Analog computers and recursive functions over the reals, J. Complexity 19 (5) (2003) 644–664.

[16] L. A. Rubel, A survey of transcendentally transcendental functions, Amer. Math. Monthly 96 (9) (1989) 777–788.

[17] D. S. Graça, M. L. Campagnolo, J. Buescu, Computability with polynomial differential equations, submitted for publication.

[18] V. I. Arnold, Ordinary Differential Equations, MIT Press, 1978.

[19] A. M. Turing, On computable numbers, with an application to the Entscheidungsproblem, Proc. London Math. Soc. 2 (42) (1936) 230–265.

[20] A. Grzegorczyk, On the definitions of computable real continuous functions, Fund. Math. 44 (1957) 61–71.

[21] D. Lacombe, Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles III, Comptes Rendus de l'Académie des Sciences Paris 241 (1955) 151–153.

[22] D. Graça, N. Zhong, J. Buescu, Maximal intervals of computable IVPs are not necessarily computable, trans. Amer. Math. Soc., to appear.