Computability via Analog Circuits

Daniel Silva Graça
CLC and ADM/FCT, Universidade do Algarve, C. Gambelas,
8000-062 Faro, Portugal

July 11, 2003

Abstract

In this paper we are interested in a particular model of analog computation, the General Purpose Analog Computer (GPAC). In particular, we provide more solid foundations for this model and we show that it can be used to introduce a notion of computability for smooth continuous dynamical systems over $\mathbb{R}^n$. We also show that hierarchies over these dynamical systems can be established, thereby defining a notion of relative computability.

1 Introduction

The General Purpose Analog Computer (GPAC) was introduced in 1941 by Shannon [19] as a mathematical model of an analog device, the Differential Analyzer [2]. This device was one of the first (analog) computers to appear and was intended to solve numerical problems, especially differential equations.

Unlikely to the approach in computable analysis [15, 11, 22], the GPAC is not directly based on the Turing machine, neither on some effective procedures. The model basically consists of circuits composed of 'black boxes' as indicated in Fig. 1.1 (the so-called *analog units.* These are not the units originally used by Shannon, but they are equivalent). It is required that two inputs and two outputs can never be interconnected. It is also required that each input is connected to, at most, one output.

Another different characterization for the GPAC has been given by Pour-El [14]. However, this last model still have some deficiencies (see [7] for details).

We take inputs $x_1, ..., x_k$ for the GPAC as real unary functions (i.e. these functions depend on the value of a variable, the time) that are applied to every input of a unit that is not connected to the output of some other unit. Then an output for the GPAC will consist of outputs of some units and/or some inputs of the GPAC. Notice the existence of a parameter α in the integrator unit. This corresponds to an initial setting that will settle the output for the integrator.

More formally, each output of an unit (and, hence, of a GPAC) will be a real unary function. This output can be obtained by solving a set of equations. For instance, if $\mathcal{U}$ is a GPAC consisting of only one adder with inputs x_1 and x_2, then the output of the adder will be the solution of the equation $y = x_1 + x_2$. In general, if we want to determinate the output of some GPAC with n units, we have to solve a set of n equations.

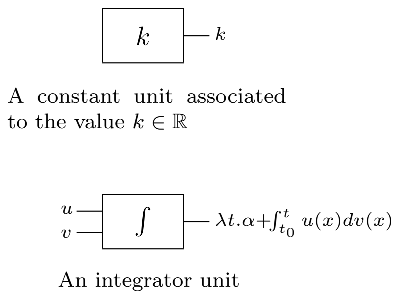
k
k
A constant unit associated to the value $k \in \mathbb{R}$
u
v
$+$
$u+v$
An adder unit
u
v
$\int$
$\lambda t.\alpha + \int_{t_0}^{t} u(x)dv(x)$
An integrator unit
u
v
$\times$
uv
A multiplier unit

Figure 1.1: Representations of different types of units in a GPAC.

Of course, the solution of a system of equations may not be unique or can even not exist (and it is not difficult to find examples - cf. [7]). In the next section, we restrict Shannon's model in order to avoid these problems.

Another important question (already reported in [14, 17]) is what happens if we allow other types of black boxes beside those indicated in Fig. 1.1. An answer for this question will be supplied by using the framework introduced in the next section. This approach will also enable us to present connections with a hierarchy consisting of smooth continuous dynamical systems defined in $\mathbb{R}^n$.

We believe that the model presented in this paper might provide a complementary theory to computable analysis. An argument for this will be provided in section 5. It is also worthwhile to notice that this theory is closely related to the field of control theory [20].

The style of the text is rather informal. In general, we will be more concerned with the ideas behind the concepts than with a detailed description of notions and proofs.

2 The basic model

In this section we present the basic model that will be used in what follows. It is essentially a restricted form of Shannon's GPAC.[1]

For the matters of this paper it is only necessary to consider one input for this model (although it is possible to adapt the model for the case of several inputs). The first idea is to use the units in Fig 1.1, except integrators, to

[1]The approach that we use in this paper is slightly different from the one that is used in [7]. However, it is possible to show that these approaches are equivalent. The author will provide a proof of that result in a forthcoming paper.

construct acyclic circuits that compute polynomials (polynomial circuits).[2] The second idea is to use these circuits as building blocks for more complex GPAC. These more complex GPACs are constructed in the following manner. Take n integrators $\mathcal{U}_1, ..., \mathcal{U}_n$. Then use polynomial circuits such that the following three conditions hold:

1. Each input of a polynomial circuit is the input of the GPAC or the output of an integrator;
2. Each integrand input of an integrator is the output of a polynomial circuit;
3. Each variable of integration input of an integrator is the input of the GPAC.

Formally, a polynomial circuit is defined as follows.

Definition 1 *A* polynomial circuit *is an acyclic GPAC built only with adders, constants units, and multipliers in the following inductive way:*

1. *A constant unit is a polynomial circuit with zero inputs and one output;*
2. *An adder is a polynomial circuit with two inputs and one output;*
3. *A multiplier is a polynomial circuit with two inputs and one output;*
4. *If $\mathcal{A}$ and $\mathcal{B}$ are polynomial circuits and if we connect the outputs of $\mathcal{A}$ and $\mathcal{B}$ to an adder, then the resulting circuit is a polynomial circuit in which the inputs are the inputs of $\mathcal{A}$ and $\mathcal{B}$, and the output is the output of the adder.*
5. *If $\mathcal{A}$ and $\mathcal{B}$ are polynomial circuits and if we connect the outputs of $\mathcal{A}$ and $\mathcal{B}$ to a multiplier, then the resulting circuit is a polynomial circuit in which the inputs are the inputs of $\mathcal{A}$ and $\mathcal{B}$, and the output is the output of the multiplier.*

The proof of the following proposition will be left to the reader.

Theorem 2 *If $x_1, ..., x_n$ are the inputs of a polynomial circuit, then the output of the circuit will be $y = p(x_1, ..., x_n)$, where p is a polynomial. Reciprocally, if $y = p(x_1, ..., x_n)$, where p is a polynomial, then there is a polynomial circuit with inputs $x_1, ..., x_n$, and output y.*

Definition 3 *Consider a GPAC $\mathcal{U}$ with n integrators $\mathcal{U}_1, ..., \mathcal{U}_n$ and one input x. Suppose that to each integrator $\mathcal{U}_i$, $i = 1, ..., n$, we can associate a polynomial circuit $\mathcal{A}_i$ with the property that the integrand input of $\mathcal{U}_i$ is connected to the output of $\mathcal{A}_i$. Suppose that each input of $\mathcal{A}_i$ is connected to the output of an integrator or to the input x. Suppose also that the variable of integration input of each integrator is connected to the input x. In these conditions we say that $\mathcal{U}$ is a polynomial GPAC (PGPAC) with input x. (cf. Fig. 2.1.)*

[2]Notice that multipliers could be replaced by integrators and adders (cf. [14, p. 11]). However, this is not very relevant to our results.

Before continuing with our work, we have to set up more conditions on this model. We have admitted that the inputs $x_1, ..., x_m$ of a GPAC are functions of a parameter t, but we didn't make any assumption on these functions (about computability, differentiability, or whatever). When we consider the integrator units one problem still arises: if $I = [a, b]$ is a closed interval, the Riemann-Stieltjes integral $\int_I \varphi(t) d\psi(t)$ is not defined for every pair of functions φ, ψ, even if they are continuous [21].

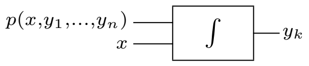

Figure 2.1: Schema of inputs and outputs of $\mathcal{U}_k$ in the PGPAC $\mathcal{U}$. p denotes a polynomial.

However, it is possible to show that if φ, ψ are continuously differentiable on I, then $\int_I \varphi(t) d\psi(t)$ is defined. So, from now on, we will always assume that the inputs are continuously differentiable functions of time.

3 Properties of the model

Provided with the former definitions, we can show that the output of a PGPAC exists and is unique, in contrast to Shannon's GPAC.[3]

Theorem 4 *Suppose that the input of a PGPAC is of class C^r on some interval I, for some $r \geq 1$, possibly ∞. Then the outputs are also of class C^r on I.*

Theorem 5 *Suppose that $\mathcal{U}$ is a PGPAC with one input x, of class C^1 on an interval $[t_0, t_f)$, where t_f may possibly be ∞. Then there exists an interval $[t_0, t^*)$ (with $t^* \leq t_f$) where each output exists and is unique. Moreover, if $t^* < t_f$, then there exists an integrator with output y such that $y(t)$ is unbounded as $t \to t^*$.*

Theorem 6 *If y is generated on some non-trivial interval I by a PGPAC with n integrators and one input x, then there is a nonzero polynomial p with real coefficients such that*

$$p\left(x, y, y', ..., y^{(n)}\right) = 0, \qquad \text{on } I. \tag{1}$$

Definition 7 *The unary function y is differentially algebraic if there exists a nonzero polynomial p with real coefficients such that (1) holds.*

Theorem 8 *Suppose that y is differentially algebraic on some non-trivial interval I. Then there is a closed subinterval $I' \subseteq I$ with non-empty interior such that y can be generated by a PGPAC on I'.*

[3]Shannon says in its paper [19, p. 338] the following: 'We shall assume that all ordinary differential equations have unique solutions and that formal processes of differentiation, integration, etc. are valid in the region of interest'. However, he does not provide a criteria for deciding when these conditions hold, in contrast to the PGPAC model.

The proofs of these results are rather technical and can be found in [8]. The last two theorems assert a classical result on the literature about the GPAC [19, 14, 12]: unary functions generated by (P)GPACs are, in essence, differentially algebraic functions.

This result indicates that a large class of functions, such as polynomials, trigonometric functions, elliptic functions, etc., can actually be generated by a PGPAC. As a corollary, some functions such as the Gamma function,

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt,$$

cannot be generated because they are not differentially algebraic functions [16].

Although this may appear problematic in the context of computable analysis (for instance, in [15, p. 27] it is indicated that Γ is computable), we believe that the problem is apparent. We will refer to this in section 5.

4 An extension of the model

In the previous models (GPAC, PGPAC) we always worked with the units indicated in Fig. 1.1. But what happens if other types of units are allowed?

Consider an indexed set $\mathcal{F} = (f_i)_{i \in I}$ such that $f_i : \mathbb{R}^{k_i} \to \mathbb{R}$ is a C^1 function, for $k_i \in \mathbb{N}$ and each $i \in I$. Now consider a new kind of PGPAC that is obtained by allowing only units associated to functions in $\mathcal{F}$ or integrators.

Definition 9 *A $\mathcal{F}$-circuit is a circuit built inductively like a polynomial circuit, but using units associated to functions in $\mathcal{F}$ instead of those indicated in Fig. 1.1.*

Definition 10 *An $\mathcal{F}$-integrating circuit ($\mathcal{F}$-IC) is a circuit built like a PGPAC, where $\mathcal{F}$-circuits are used instead of polynomial circuits.*

In recursion theory one defines classes of functions by considering algebras of functions. Formally,

Definition 11 *Let χ be a set of functions and OP a collection of operators. Then $[OP; \chi]$ denotes the smallest set of functions containing χ and closed under the operations of OP. The set $[OP; \chi]$ is called a* function algebra.

The notation will not be very rigorous (e.g. $[OP, G; \chi]$ means $[OP \cup \{G\}; \chi]$, $[OP; \chi_1, \chi_2]$ means $[OP; \chi_1 \cup \chi_2]$, etc.), but the context will be enough to clarify all situations. We shall consider the following functions and operators.

1. The projections. Let A be a set. For each $n, i \in \mathbb{N}$, where $1 \leq i \leq n$, $U_i^n : A^n \to A$ is called projection (over A) and is defined by $U_i^n(x_1, ..., x_n) = x_i$;

2. The constant functions. For each $k \in \mathbb{R}$, $f_k : \mathbb{R} \to \mathbb{R}$ is defined by $f_k(x) = k$;

3. Composition: Suppose that g is an p-ary function, with $p \geq 1$, and that $f_1, ..., f_p$ are n-ary functions. Then the composition operator applied to these functions by that order yields the n-ary function h given by $h(\mathbf{x}) = g(f_1(\mathbf{x}), ..., f_p(\mathbf{x}))$. We write $h = C(g; f_1, ..., f_p)$.

We set the following notation

$$U = \{U_i^n : n, i \in \mathbb{N} \text{ and } 1 \leq i \leq n\},$$
$$C_{\mathbb{R}} = \{f_k : k \in \mathbb{R}\}.$$

Notice that a PGPAC is simply a $(C_{\mathbb{R}}, +, \times)$-IC. The following result, as we will see, provides fundamental links between the IC model and the theory of continuous dynamical systems.

Theorem 12 *Let $\mathcal{U}$ be a $\mathcal{F}$-IC with n integrators and one input x. Then there exist n $(n+1)$-ary functions $h_1, ..., h_n \in [C; U, \mathcal{F}]$ such that $(\psi_1, ..., \psi_j)$ is an output of $\mathcal{U}$ iff there exist n unary functions $y_1, ..., y_n$ such that:*

1. *$\partial_t y_i = h_i(t, y_1, ..., y_n)$ and $y_i(x(0)) = \alpha_i$, where $\alpha_i \in \mathbb{R}$;*
2. *There exist j $(n+1)$-ary functions $g_1, ..., g_j \in [C; U, \mathcal{F}]$ such that $\psi_i = g_i(x, y_1 \circ x, ..., y_n \circ x)$, for $i = 1, ..., j$.*

Proof. It is possible to show that if $\bar{y}_1, ..., \bar{y}_n$ are the outputs of the integrators, then the output of each $\mathcal{F}$-circuit is given by $f(x, \bar{y}_1, ..., \bar{y}_n)$, where $f \in [C; U, \mathcal{F}]$ (simply generalize theorem 2). Therefore, each output $\bar{y}_i$ of an integrator satisfies

$$\bar{y}_i(t) = \alpha_i + \int_0^t h_i(x(t), \bar{y}_1(t), ..., \bar{y}_n(t)) dx(t),$$

where $\alpha_i \in \mathbb{R}$ and $h_i \in [C; U, \mathcal{F}]$. Then there are unary functions $y_1, ..., y_n$ satisfying

$$y_i(w) = \alpha_i + \int_{x(0)}^w h_i(w, y_1(w), ..., y_n(w)) dw, \tag{2}$$

and $\bar{y}_i(t) = y_i \circ x(t)$. In fact, if $\varphi : \mathbb{R} \to \mathbb{R}$ is a C^1 function, then

$$\int_{\varphi(a)}^{\varphi(b)} f(s) ds = \int_a^b f \circ \varphi(t)\ \varphi'(t) dt = \int_a^b f \circ \varphi(t) d\varphi(t).$$

The first equality comes from the substitution formula for definite integration [4, p. 265]. The second equality comes from a well known result for Riemann-Stieltjes integrals [21, formula (1.2.3)]. Part 1 of the theorem follows by differentiating equation (2).

Part 2 of the theorem follows from the fact that each output is the input x, the output of some integrator, or a single output of a $\mathcal{F}$-circuit.

Reciprocally, if conditions 1 and 2 are satisfied, then it is not difficult to construct a $\mathcal{F}$-IC $\mathcal{U}$ with input x, n integrators, and output $(\psi_1, ..., \psi_j)$. ■

Corollary 13 *y is generated by a PGPAC iff it is a component of the solution $\mathbf{y} = (y_1, ..., y_n)$ of a differential equation $\frac{d\mathbf{y}}{dx} = \mathbf{p}(\mathbf{y}, x)$, where $\mathbf{p}$ is a vector of polynomials.*

Proof. The PGPAC uses as basic functions elements of $(C_{\mathbb{R}}, +, \times)$. But $[C; U, C_{\mathbb{R}}, +, \times]$ is the set of all polynomials. Then part 1 of theorem 12 gives us $y_i' = p_i(t, y_1, ..., y_n)$, where p_i is a polynomial. Moreover, by using part 2 of that theorem, we conclude that each g_i is a polynomial. Hence, it can be written as $g_i' = q_i$, where q_i is a polynomial. Therefore, for the special case of polynomials, part 1 and 2 of theorem 12 can be condensed in a single system $\frac{d\mathbf{y}}{dx} = \mathbf{p}(\mathbf{y}, x)$, where x is the input. (Note that some of the y_i's actually represent the functions g_l.) $\blacksquare$

5 Dynamical systems

In this section we explain the importance of theorem 12 and corollary 13. One of their consequences is that although the IC model and standard computable analysis theory may appear quite different, from the dynamical systems point of view, they complete each other (at least, in $\mathbb{R}^n$).

We first start with a few basic definitions.

Definition 14 *A dynamical system defined on the topological space S over $\mathbb{A} = \mathbb{R}_0^+$ ($\mathbb{A} = \mathbb{N}$) is a triple $(S, \mathbb{A}, \phi)$, where $\phi : S \times \mathbb{A} \to S$ is a function satisfying*

1. *Initial condition: $\phi(p, 0) = p$ for any $p \in S$;*
2. *Continuity on both arguments;*
3. *Semigroup property:*

$$\phi(\phi(p, t_1), t_2) = \phi(p, t_1 + t_2),$$

for any point $p \in S$ and any $t_1, t_2 \in \mathbb{A}$.

When $\mathbb{A} = \mathbb{R}_0^+$, we say that we have a *continuous dynamical system.* On the other side, when $\mathbb{A} = \mathbb{N}$, we say that we have a *discrete dynamical system. S* is the *state space* and points in S are called *states.*

Notice that in the above definition we could replace $\mathbb{R}_0^+$ by $\mathbb{R}$ and $\mathbb{N}$ by $\mathbb{Z}$. In these cases we say that we have *reversible dynamical systems.*

Let us analyze the reasons that motivated our dynamical systems approach to computability (notice that this already appears in some papers, e.g. [13, 10, 1, 18]).

One of the reasons is tied up to the notion of time. It seems natural that a computer carries out a sequence of instructions along time. For example, one of the most important measures of computational complexity is time complexity. In dynamical systems, time is handled by the semigroup $\mathbb{A}$.

Another important aspect is the existence of some space where the process of computation is carried out. For instance, a Turing machine has one or more tapes and also some states. This is modelled by the state space.

Finally, a computer must follow some 'mechanical process' while it is computing. This is given by the function ϕ. It is also natural that, given a computer and some initial state, and for time 0, we should have as current state the given initial state (cf. point 1 of definition 14). Moreover, if we compute from an initial state p_0 for t_1 time units, obtaining a state p_1, and then start computing t_2 time units from state p_1, we should get as final state the same state that we would obtain starting from p_0 and computing for $t_1 + t_2$ time units. This explains the semigroup property. Finally, condition 2 says that we do not want functions ϕ 'too strange'.

As a concrete example, each Turing machine is a discrete dynamical system in $\mathbb{Z}^2$ [1, proposition 5.1]. Taking in account all the facts presented above we could perhaps say that a computer is simply a dynamical system. Then a notion of computability can be defined as follows.

Take the class of functions that can be generated by a model of computation. This is defined to be the class of computable functions. Notice that in the classical case, and also in computable analysis, some object is computable if it is the limit object obtained by carrying out a sequence of steps in the respective model, perhaps through infinite time. For instance, if we take as model the Turing machine, then the computable functions correspond to recursive functions. If we take Type-2 Machines [22] and a represented space $(\mathbb{R}, \delta_{\mathbb{R}})$, then the computable functions correspond to $(\delta_{\mathbb{R}}, \delta_{\mathbb{R}})$-computable functions.

Hence, if the computational model is seen as a dynamical system, then computable objects do not correspond to dynamical systems, but instead to their *attractors*. In this manner a continuous object can be generated by a model associated to a discrete dynamical system, working with a discrete state space. It is important to remark that, in this case, dynamical systems are not directly related to computable objects, but instead to the underlying model of computation and, indirectly, to these computable objects.

In opposition to the previous case, when using the PGPAC model, computable objects are no longer objects obtained through limit operations. Although it is possible to use this approach (and to show that, e.g. points and many chaotic attractors are computable) it does not reflect the type of computation carried out by many continuous time devices, such as the Differential Analyzer, where the output is obtained in 'real time'. Hence, the work presented in this paper differs from the computable analysis approach in two aspects: (i) it uses a continuous time model of computation, instead of using a discrete time model (ii) the outputs are obtained in 'real time', instead of being successively approximated. Under these assumptions, the PGPAC-computable functions are given by corollary 13.

Moreover, by adding more power to the model, we can generate a full hierarchy having as bottom the class of computable functions. For example, in the classical case, we can consider the *jump* operation [6]. This is done by considering Turing machines with oracles. It is well known that the Halting Problem is not computable by a Turing machine and, hence, if we use an oracle that solves the problem, we will get more power. This defines the class next to the computable functions. By taking a similar approach (use an oracle that solves the Halting Problem for the previous class), we get a non-collapsing hierarchy having as bottom the computable functions.

The same thing can be done with ICs. The bottom of the hierarchy would be given by corollary 13. Then, because Γ does not belong to this class, we could add a unit generating Γ, obtaining $(C_{\mathbb{R}}, +, \times, \Gamma)$-ICs. In this manner we can define the next class. Then we might add to the model a function not generated by $(C_{\mathbb{R}}, +, \times, \Gamma)$-ICs (the class of functions generated by this model is given by theorem 12), therefore defining a new larger class and so on.

The following question can be put about the last procedure: Is there always a next class? By other words, does the hierarchy collapses? We know that there is a class that properly contains the functions generated by PGPACs. But what about for further classes?

We don't have an answer for this yet but we strongly believe that there is a non-collapsing hierarchy of C^1 functions that can be defined as suggested above.

Relatively to the issue of continuous time dynamics, it is worthwhile to mention that a continuous dynamical system working on the euclidean space S is associated to an ordinary differential equation

$$\mathbf{x}' = \mathbf{f}(\mathbf{x}), \tag{3}$$

where $\mathbf{x}$ is a unary function with $\mathbf{x}(t) \in S$ (provided ϕ is of class C^1) [9, p. 160]. There is also a converse result.

It might appear that the last result is only true for autonomous equations like (3) (the equation does not depend on t), but we can always reduce a differential equation

$$\mathbf{x}' = f(\mathbf{x}, t)$$

in $\mathbb{R}^n$ to an autonomous differential equation in $\mathbb{R}^{n+1}$. This is done by considering the system

$$\begin{cases} \mathbf{y}' = f(\mathbf{y}, z) \\ z' = 1. \end{cases}$$

It is easily seen that $\mathbf{y} = \mathbf{x}$, thus showing the result.

Therefore, C^1 continuous dynamical systems in $\mathbb{R}^n$ can be identified with differential equations like (3). Hence, by theorem 12, it seems that the IC is especially well suited to deal with continuous dynamical systems over $\mathbb{R}^n$.

This is the reason for the assertion in the beginning of this section: the IC model and computable analysis are complementary while studying computability issues in $\mathbb{R}^n$. Indeed, the first uses continuous dynamical systems to define a notion of computability, while the second uses models related to discrete dynamical systems to achieve its purposes.

We believe that this is meaningful, because we provide a tool for dealing with computability in continuous time dynamics.

For example, although modern computers are best described using discrete procedures, as physical devices they compute in continuous time. Moreover, discrete procedures are not very well designed to deal with processes that work in 'real time'. These processes are common, for example, in control theory.

Therefore, this continuous time dynamics approach is needed if one wants to study the computational capabilities of natural phenomena. We think that only this kind of approach can answer questions such as: 'Is there some realistic physical device with super-Turing power?'.

So, we believe that the IC has an important role to play in the description of continuous dynamical systems, perhaps similar to the one that is played by the Turing machine in discrete dynamical systems.

6 Conclusion

We have presented a new model (PGPAC) based on Shannon's GPAC and we have shown that this model presents some characteristics that make it more suitable than Shannon's and Pour-El's GPAC. Moreover, we have extended this model (to the IC model) and also established links with the theory of continuous dynamical systems.

We have also explained how the IC model can provide a complementary theory relatively to computable analysis, by using the framework of dynamical systems. In this manner, the traditional point of view in computable analysis is to introduce a notion of computability via discrete dynamical systems, while the IC gives a notion of computability via continuous dynamical systems.

It is worthwhile to explore relations between those two approaches. For instance, one could ask under which assumptions PGPACs lead to computable functions, in the sense of computable analysis. In [3, p. 3] it is pointed out that any analytic differentially algebraic function (functions generated by the PGPAC are of this type) satisfies an equation like (1), where p has integer coefficients and, therefore, the computational power of the PGPAC does not rely on its capacity to use the infinite amount of information which can be encoded in a real number. Moreover, Campagnolo also presents a conjecture [3, conjecture 3.5.1], where functions computable by PGPACs that only have access to rational constants in its initial conditions and parameters, are expected to have primitive recursive upper bounds and, therefore, be computable.

An alternative direction that might provide a bridge between these two distinct areas can also be found in [8]. Here it is presented a sketch of 'effective GPACs', where initial parameters can only take values in some set $A \subseteq \mathbb{R}$. Then, by using an 'initialization procedure', it is possible to generate other constants, and therefore to bound the computational power of the model. For instance, if we pick $A = \{-1, 0, 1\}$, the corresponding model will only access a countable number of constants in opposition to the standard model that has access to all real constants. Among these constants are the rational numbers, e, π, and $\sqrt{2}$. However, none of the previous approaches have been further developed.

Some directions for further research on the IC model can be pointed out. Let us present some of them.

1. The Turing machine is usually seen as an adequate model of computability for discrete dynamical systems. Can we say that the IC model has the same role for processes working in continuous time?

2. Can we extend the class of functions generated by a PGPAC in a realistic manner? By other words, the class of all dynamical systems associated to a differential equation $\mathbf{y}' = \mathbf{p}(\mathbf{y}, x)$, where $\mathbf{p}$ is a vector of polynomials, should be taken as the bottom of the hierarchy described in section 5, or should we consider a larger class?

3. How can we precisely define a notion of complexity for the PGPAC? And which are its connections with the theory of dynamical systems? For instance, in [5] it is shown that restricted forms of integration lead to a hierarchy of continuous time systems related to the Grzegorczyk hierarchy.

4. Is it possible to establish connections with computable analysis? In particular, is it possible to further exploit the ideas presented above?

Acknowledgments. The author would like to thank Manuel Campagnolo, José Félix Costa, Vasco Brattka, and the anonymous referee for their helpful comments on this paper. This work was partially supported by *Fundação para a Ciência e a Tecnologia* (FCT) and FEDER via the Center for Logic and Computation - CLC, and is inserted in the FCT project ConTComp POCTI/MAT/4597 8/2002. Part of this work appeared in the author's MSc dissertation supervised by José Félix Costa.

References

[1] M. S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science,* 138(1):67-100, 1995.

[2] V. Bush. The differential analyzer. A new machine for solving differential equations. *Journal of the Franklin Institute*, 212:447-488, 1931.

[3] M. L. Campagnolo. *Computational Complexity of Real Valued Recursive Functions and Analog Circuits.* PhD thesis, IST/UTL, 2002.

[4] R. Courant and F. John. *Introduction to Calculus and Analysis,* volume I, Springer, 1989.

[5] M. L. Campagnolo, C. Moore, and J. F. Costa. An analog characterization of the Grzegorczyk hierarchy. *Journal of Complexity*, 18(4):977-1000, 2002.

[6] N. J. Cutland. *Computability: An introduction to Recursive Function Theory,* Cambridge University Press, 1980.

[7] D. S. Graça and J. F. Costa. Analog computers and recursive functions over the reals. *Journal of Complexity*, to appear.

[8] D. S. Graça. *The General Purpose Analog Computer and Recursive Functions over the Reals*, MSc thesis, IST/UTL, 2002. (available online at `http://w3.ualg.pt/~dgraca`.)

[9] M. W. Hirsch and S. Smale. *Differential Equations, Dynamical Systems, and Linear Algebra,* Academic Press, 1974.

[10] P. Koiran, M. Cosnard, and M. Garzon. Computability with low-dimensional dynamical systems. *Theoretical Computer Science*, 132:113-128, 1994.

[11] Ker-I-Ko. *Computational Complexity of Real Functions*, Birkhäuser, 1991.

[12] L. Lipshitz and L. A. Rubel. A differentially algebraic replacement theorem, and analog computation. *Proceedings of the AMS*, 99(2):367-372, 1987.

[13] C. Moore. Unpredictability and undecidability in dynamical systems. *Physical Review Letters,* 64(20):2354-2357, 1990.

[14] M. B. Pour-El. Abstract computability and its relations to the general purpose analog computer. *Transactions of the AMS*, 199:1-28, 1974.

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

[16] L. A. Rubel. A survey of transcendentally transcendental functions. The *American Mathematical Monthly*, 96:777-788, 1989.

[17] L.A. Rubel. The extended analog computer. *Advances in Applied Mathematics*, 14:39-50, 1993.

[18] H. T. Siegelmann and S. Fishman. Analog computation with dynamical systems. *Physica D*, 120:214-235, 1998.

[19] C. Shannon. Mathematical theory of the differential analyzer. *Journal of Mathematics and Physics*, 20:337-354, 1941.

[20] E. D. Sontag. *Mathematical Control Theory*, Springer, 2nd Edition, 1998.

[21] D. W. Stroock. *A Concise Introduction to the Theory of Integration*, Birkhäuser, 3rd edition, 1999.

[22] K. Weihrauch. *Computable Analysis: An Introduction*, Springer, 2000.