UNIVERSIDADE TÉCNICA DE LISBOA

INSTITUTO SUPERIOR TÉCNICO

The General Purpose Analog Computer and Recursive Functions Over the Reals

Daniel da Silva Graça

MSc thesis
(submitted)

July 2002

O Computador Analógico e Funções Reais Recursivas

Nome: Daniel da Silva Graça
Curso de Mestrado em: Matemática Aplicada
Orientador: Doutor José Félix Gomes da Costa
Provas concluídas em:

Resumo: Pretende-se analisar na presente dissertação diversos modelos matemáticos de computação analógica.

Começa-se por analisar o primeiro modelo conhecido deste tipo, o Computador Analógico (GPAC, abreviatura do inglês). São descritos os principais resultados existentes para este modelo, sendo também apresentada uma abordagem alternativa. É mostrado que esta nova abordagem origina um modelo mais robusto que o GPAC, mantendo, no entanto, as suas principais propriedades, tais como a equivalência com funções diferencialmente algébricas. Introduzem-se também novos conceitos que julgamos relevantes, tais como o procedimento de inicialização e a noção de GPAC efectivo.

Seguidamente, o nosso estudo incide sobre a teoria das funções reais recursivas, uma teoria análoga à teoria clássica das funções recursivas, em que as funções são consideradas sobre o conjunto dos reais, em vez do conjunto dos naturais. Propõem-se novas classes de funções, relacionando-se estas com as principais classes da teoria clássica, incluindo a Hierarquia Aritmética. Além disso, mostram-se ainda relações entre as funções reais recursivas e funções geradas por modelos semelhantes ao GPAC.

Palavras-chave: Computação analógica, computador analógico, equações diferenciais, sistemas dinâmicos, funções recursivas, hierarquia aritmética.

The General Purpose Analog Computer and Recursive Functions Over the Reals

Abstract: The purpose of the present dissertation is to analyze various mathematical models of analog computation.

This work starts by analyzing the first known model of this kind, the General Purpose Analog Computer (GPAC). We present the existing results for the GPAC and propose an alternative approach. We show that this approach originates a more robust model than the GPAC, maintaining its principal properties such as the equivalence with differentially algebraic functions. We also introduce new concepts such as the initialization procedure and effective GPACs that we think to be relevant.

We pursue our study with the theory of recursive functions over the reals, a similar theory to the classical theory of recursive functions, proposing new classes of functions and relating them with the main classes of the classical theory, including the Arithmetical Hierarchy. We also show relations between recursive functions over the reals and functions generated by models similar to the GPAC.

Key-words: Analog computation, general purpose analog computer, differential equations, dynamical systems, recursive functions, arithmetical hierarchy.

Acknowledgments

Firstly, I want to thank my advisor, J. Félix Costa, for its dedication and guidance during the elaboration of this thesis. The constant support and motivation that he provided were essential for the determination of the course of this work. And with its energy and enthusiasm, he was also able to present me a more realistic perspective of research that I had, certainly improving my skills in this domain.

I would also like to thank the Section of Computer Science of the Department of Mathematics at IST/UTL for the assistance and support provided whenever needed. I also wish to thank my colleagues at the Department of Mathematics of the University of Algarve for their help in several occasions.

I must also express my gratitude to the Center for Logic and Computation and, in particular, to Amílcar Sernadas, for the opportunities that I had to expose my work in the Theory of Computation Seminar.

Many people helped me with useful discussions and suggestions, while doing this work. In particular, I wish to thank Manuel Campagnolo, Stefan Samko, J. Sousa Ramos, Hermenegildo Borges, Celestino Coelho, and Cristopher Moore. I also wish to thank Diana Rodelo for the improvements done in some parts of this text.

Finally, I wish to give very special thanks to my family, for their presence and unconditional support.

Contents

Introduction

The quest for understanding nature has long been one of the biggest challenges of human kind. The search for answers has lead to the creation of a diversity of models trying to explain and model different aspects of nature.

Meanwhile, a tremendous effort has been made with the objective of controlling natural phenomena in order to serve human needs. Nowadays, more than ever, technology is a key factor within human society.

Hence, the demand for machines that could simulate some aspects of the physical world appeared naturally, in order to understand and model it. Several computational models appeared to perform this task, but discrete (digital) models gained a rapid prominence.

Digital computation has been, since the thirties, the most important computational model, mainly due to the unifying work of Turing. Turing clarified the notion of *algorithm*, giving it a precise meaning, and introduced a coherent framework for discrete computation. In a short time, new results showing the relations of his model with other approaches, such as recursive functions (in the sense of Kleene), originated in a natural way consistent theoretical basis to standard computation theory. Meanwhile, with the rapidly growing needs of various fields such as physics, engineering, etc., to make enormous quantities of calculations and information processing, many times beyond human capabilities, new computing devices were developed and improved. With these new technologies, digital computers improved dramatically in speed, size and accuracy, until the present date. Hence, it is not difficult to understand why discrete computation became today's main computational paradigm.

Nevertheless, computers need not to be digital. In fact, the first computers were *analog computers*. In an analog computer, the internal states are continuous, rather than discrete as in digital computation. The first analog computers were especially well suited to solve ordinary differential equations and were effectively used to solve many military and civilian problems during World War II (e.g. gunfire control) and in the fifties and sixties (e.g. aircraft design). Unfortunately, because of the inexistence of a coherent theoretical basis to analog computation and the fact that analog computers technology almost didn't improve when compared with its digital counterpart in the last half century, analog computation was about to be forgotten with the emergence of digital computation.

Despite this period of oblivion, analog computation is regaining again interest. The search for new models that could provide an adequate notion of computation and complexity for the dynamical systems that are currently used to model the physical world contributed to change the situation. However, relatively little work exists on a general theory for analog computation, and this still seems far away. Nevertheless, this is a potentially rich and fertile field. In analog computers, each real is handled exactly and is considered to be an intrinsic quantity, whereas in digital computers it is represented (and approximated) by strings of digits. Hence, it seems that analog computation is better suited for studying notions of computability and complexity for continuous dynamical systems.

Generally speaking, any computational model can be seen as a dynamical system. The main property that distinguishes analog models from digital ones is the use of a continuous state space [Sie99].[1] Besides this feature there is no agreement upon the properties that characterize an analog model of computation. However, in this dissertation, we will say that a model of computation is *digital* if its space of states is discrete and *analog* otherwise.

Recent research [Moo90, Koi93, KCG94, KM99] shows that Turing machines, when converted into discrete dynamical systems, can be embedded in analog systems. Hence, we could see analog computation as an extension of digital computation. Moreover, in this fashion, we get a physical meaning to the Turing machine that cannot be obtained with the classical description.

We can also find common paradigms both to digital and analog computation, such as neural networks. Neural networks first appeared as discrete models [MP43], proposed by McCulloch and Pitts, but we can now find significant research in analog neural networks. For instance, Siegelmann and Sontag [SS94] were able to show how to simulate Turing machines with analog neural networks.

Although analog computation is certainly in its infancy, current research suggests some lines of work. Some analog models may be seen as high dimensional dynamical systems (highly parallel models), e.g. neural networks, and others may be seen as low dimensional dynamical systems (e.g. [Moo90, KCG94, KM99]). On the other hand, we may classify analog models as discrete time models (e.g. [Sie99]) or as continuous time models (e.g. [Orp95]). However, this is not a rigid characterization and it is possible to find hybrid models (e.g. [Bra95]). In this thesis we will be mainly concerned with continuous time models.

We will now briefly refer the contents of this dissertation. The initial purpose was to develop and explain the ideas presented in [Moo96]. In his paper, Moore introduced a recursion theory over the reals and presented some results establishing links with the classical recursion theory and also with a continuous time model of analog computation, the General Purpose Analog Computer (GPAC).

Unfortunately, these results presented some gaps and, hence, the primary goal of this thesis is precisely to provide an adequate framework to obtain, whereas possible, similar results to those presented by Moore. In particular,

[1] Note that we can have analog models with discrete time.

we are interested in presenting connections between recursive functions over the reals and functions generated by the GPAC. Nevertheless, while working on this topic, some problems not referred in existing literature (at least to our knowledge) appeared and a major revision of the GPAC was needed for our purposes. Therefore, this dissertation is roughly divided in two parts. The first part (corresponding to chapters 1 and 2) is dedicated to the study of the GPAC and the second part (chapter 3) is dedicated to recursive functions over the reals. The contents of each chapter are as follows.

In Chapter 1 we introduce the reader to the classical theory about the GPAC. We explain its underlying motivations and present an outline of the previous work done on the GPAC. This is basically an introductory chapter to the next chapter.

In chapter 2 we start by analyzing some problems that appear in the scope of the GPAC and we present an alternative model for it (the feedforward GPAC: FF-GPAC) that solves most of the problems referred for the GPAC. The rest of the chapter is devoted to the task of showing that the FF-GPAC model is more robust than the GPAC, but that still preserves the fundamental properties (equivalence with differentially algebraic functions). The last two sections may appear uninteresting but they will be important for the main result that we present in Chapter 3.

The third chapter is devoted to recursion theory over the reals. It is a mixture of existing theory (mainly contributions from C. Moore, M. L. Campagnolo, and J. F. Costa) and of new results that we propose in this thesis. It may be used as an introduction to the topic of recursive functions over the reals. The importance of results in this section depends on the point of view. In sections 3.1, 3.2, and 3.3, the existing theory is present and new results are added. Several connections between recursive functions over the naturals and recursive functions over the reals are established. If the reader is mainly interested in recursion theory, these are probably the most interesting sections. In sections 3.4 and 3.5, we establish full connections between some particular subclasses of recursive functions over the reals and the FF-GPAC model. We believe that this is an important result because it shows that it is possible to establish connections between recursive functions over the reals and a model based on circuits (the FF-GPAC).

Chapter 1

Preliminaries

1.1 Shannon's Work

In this section we will go to the origins of analog computation with the pioneer work of Claude Shannon on the so-called *General Purpose Analog Computer* (GPAC).[1] In essence, this is a mathematical model of an analog device, the differential analyzer. The fundamental principles for this device were first realized by Lord Kelvin in 1876 [Bus31]. Working with an integrating device conceived by his brother James Thomson, he had the idea of connecting integrators to solve differential equations.

A mechanical version of the differential analyzer was first developed by V. Bush and his associates at MIT in 1931 [Bus31]. Many mechanical difficulties were overcome by ingenious solutions. However, the inherent difficulties of the mechanical devices, due to mechanical restrictions, limited the speed of these differential analyzers and originated colossal machines.

The introduction of electronic differential analyzers was able to overcome these problems to a great extent. But this was not enough to compete with emergent digital computers.

Differential analyzers were typically used in the thirties and forties to compute functions, especially to solve ordinary differential equations (ODEs). Their continuous nature allowed the direct simulation of a solution from a given system of ODEs.[2] Hence, comparatively to standard procedures that basically use a discrete version of time (i.e., the independent variable) in order to approximate the solution (with the inherent errors introduced in these processes), differential analyzers permitted faster solutions with increased precision at that time.

[1]This name is somewhat misleading. This is not a general purpose device capable of handling all kinds of analog computations in a similar way as a universal Turing machine do. And it is not known whether it exists some general machine that can do all the computations that every GPAC can. Nevertheless, this is the usual designation in the field and we will hence keep it.

[2]In general, this system of ODEs must satisfy some conditions in order to be simulated by a differential analyzer. However, experience rapidly demonstrated that a differential analyzer could solve many useful systems of ODEs.

As an example, the Rockeffeler Differential Analyzer (designed by V. Bush and operational for the war effort in 1942) could solve an ODE up to order 18, being able to produce results to at least 0.1 percent accuracy [Bow96, p. 8].

A differential analyzer may be idealized as a General Purpose Analog Computer that basically is a model consisting of several (finite number) interconnected units, i.e., a circuit.[3]

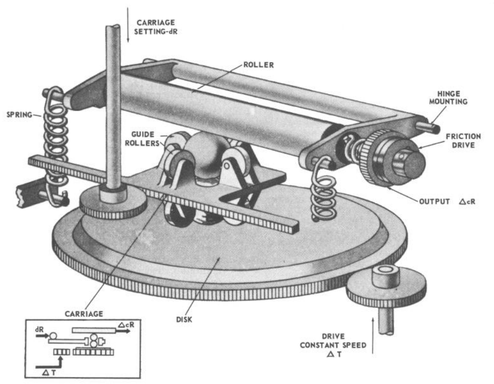


Figure 1.1.1: A disc type integrator device. Figure taken from [Bur71] with permission of Dover Publications, Inc.

There are some restrictions in the way in which these units are connected, but in general it is a quite liberal model. After having the circuit properly set up (i.e., after connecting in an appropriate way all the units), the computation can be started. The inputs of the circuit will be applied to the inputs of units not connected to any output.

When we introduce some input into the circuit, each unit will respond correspondingly to this input, generating some kind of output. We may pick some of these outputs to be the result of the computation (that we will refer as the functions *generated* or *computed* by the GPAC). In mechanical differential analyzers, the quantities involved in the computation are usually represented by the rotation of shafts, and in a electronic differential analyzer they are usually represented by electric voltages.

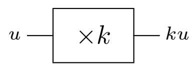
The units used in a GPAC are the following:

- *Integrator*: a two-input, one-output unit with a setting for initial condi-

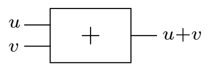
[3]More formally, it is sufficient for our purposes to consider a circuit as being a 7-tuple $(V, X, Y, E, S, \sigma, h)$, where V is a non-empty set (set of units), X is a set satisfying $X \cap V = \varnothing$ (set of inputs), $Y \subseteq V \cup X$ (Y denotes the set of outputs), S is a set satisfying $S \cap (V \cup X \cup E) = \varnothing$ (set indicating the type of the units), $\sigma : V \to S$ is a function such that $dom(\sigma) = V$ (it assigns each unit to its type), $h : S \to \mathbb{N}$ (we consider $0 \in \mathbb{N}$) is a function such that $dom(h) = S$ (it indicates the number of inputs associated to each type of unit), and $E \subseteq (V \cup X) \times V \times (\mathbb{N} - \{0\})$ satisfies the following conditions: if $(a, b, n) \in E$ then $n \leq h(\sigma(b))$ and if $(a_1, b, n), (a_2, b, n) \in E$ then $a_1 = a_2$. If $(a, b, n) \in E$ we say that the output of a is *connected* to the nth input of b. We consider units only with one output because, for our purposes, we may replace a unit with k outputs by k units, each with one output (keeping the same number of inputs).

tion. If the inputs are unary functions u, v, then the output is the Riemann-Stieljes integral $\lambda t. \int_{t_0}^{t} u(x)dv(x) + a$, where a and t_0 are constants defined by the initial settings of the integrator.

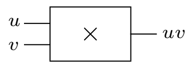
- *Constant multiplier*: a one-input, one-output unit associated to a real number. If u is the input of a constant multiplier associated to the real number k, then the output is ku.
- *Adder*: a two-input, one-output unit. If u and v are the inputs, then the output is $u + v$.
- *Multiplier*: a two-input, one-output unit. If u and v are the inputs, then the output is uv.
- *Constant function*: a zero-input, one-output unit. The value of the output is always 1.



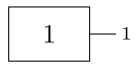
A constant multiplier unit associated to the value k



An adder unit



A multiplier unit



A constant function unit

Figure 1.1.2: Representations of different types of units in a GPAC.

We will also take from here the following conventions: except indicated otherwise, the inputs of a unit appear on the left and the outputs appear on the right (see figures 1.1.2 and 1.1.3); in an integrator the input corresponding to the integrand is on the top and the input corresponding to the variable of integration is on the bottom (see fig. 1.1.3).

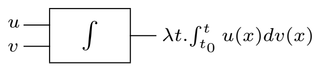


Figure 1.1.3: An integrator unit

As in the differential analyzer, we require that two inputs and two outputs can never be interconnected. We also demand that each input is connected to, at most, one output (but each output can be connected to several inputs).

These are natural restrictions that still allow considerable freedom to make connections. We can make circuits with feedback, generating in this way a rich set of functions. Of course, we could add other types of units (and this was actually done in practice), obtaining possibly different behaviors. But circuits are, in general, difficult models to deal with. So, we won't take this approach here and we will only analyze the circuits described above. Nevertheless, we think that it is a very interesting question to see what happens when new units are added. Circuits are widespread used, e.g. in engineering, and even digital computers are designed using circuits instead of thinking them as Turing machines.

Remark 1.1.1 Notice that an output of a GPAC can be considered as being the output of an integrator. In fact, suppose that the output y is the output of some unit $\mathcal{A}_k$. Hence, we may introduce a new integrator and connect the input associated to the variable of integration to the output of $\mathcal{A}_k$, and the input associated to the integrand to a constant function unit. If we set the initial condition of the output of the integrator to be $y(t_0)$, the output of this integrator will be precisely y.

Remark 1.1.2 Note that we do not need the multiplier unit. In fact, using the formula of integration by parts (for definite integration), we obtain

$$u(t)v(t) = \int_{t_0}^{t} u(x)dv(x) + \int_{t_0}^{t} v(x)du(x) + u(t_0)v(t_0).$$

So, uv can be computed by the circuit of figure 1.1.4.

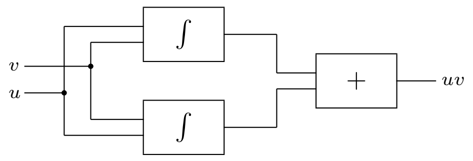


Figure 1.1.4: A circuit that computes uv.

In the circuit, we can take as initial conditions $y_1(t_0) = u(t_0)v(t_0)$ and $y_2(t_0) = 0$ for the outputs of the integrators at the bottom and at the top, respectively. Note that this actually only works when the Riemann-Stieljes integrals are defined. So, if we add other types of units as building components of the GPAC, this approach may fail.

Notice that in the above description of the units, a parameter t appears. Although it usually appears when defining these kind of circuits, its function is somehow unclear. In his seminal paper [Sha41], Shannon considers t to be some independent variable that is inputted into the circuit. He also considers that more than one independent variable can be inputted, say $t_1, ..., t_n$. He even manages to state some results for this case. However, these results are based in the assumption that the variables are independent, which appear not to be the case for differential analyzers. If we think in terms of a physical device, all the inputs are functions of time and they don't have to be independent (even if we don't know their actual relationships). So, we believe that this is not an appropriate model for working with differential analyzers with more than one input.

In the description of differential analyzers given in [Pou74] we find the ideas reported above: *"The functions generated by an electronic differential analyzer are functions of time..."* (p. 9) and *"The functions generated by a mechanical differential analyzer are also functions of time"* (also p. 9). Nevertheless, when Pour-El defines the GPAC she says: *"Note that our functions are not necessarily functions of time"* (p. 10) and in a footnote of page 11, she continues: *"An obvious modification of this can be made when dealing with functions of more than one variable. Simply require that precisely one of the independent variables be applied to each input not connected to an output."* So, when dealing with more than one variable, she actually works with independent variables.

However, even if time is our independent variable, we cannot input it directly to, e.g. an electronic differential analyzer. What we actually input is some function $x = \lambda t.x(t)$. For the one variable case, this actually does not make much difference: simply take x as a new independent variable. But for the case of more than one variable, the same does not happen. We are interested to clarify this point to present some results for the case of more than one variable. Little has been made in this domain and, as far as we know, the only results about this case appear in [Sha41].

In order to clarify the problem indicated above, we will suppose that a GPAC can have more than one input, but each input depends on a parameter t that we call the *time* (although this may not actually correspond physically to time but, e.g. to displacement). Hence, if $x_1, ..., x_n$ represent the inputs, there exist unary functions (and we are not assuming any special property for them) $\varphi_1, ..., \varphi_n$ such that $x_1 = \varphi_1, ..., x_n = \varphi_n$. If we only have one variable, we will often work with it as it were an independent variable.

This approach is similar to Turing machines, where the computations are done with the increment of one discrete variable, the running time (that usually corresponds to the number of steps that were already performed in the computation).

Although the work presented in [Sha41] contained some gaps, it introduced the main results and tools of the area. In particular, Shannon was the first to note an important connection that we will state in what follows.

Definition 1.1.3 *A unary function y is differentially algebraic (d.a.) on some set I if there exists some natural number n, where $n > 0$, and some $n+1$-ary nonzero polynomial P with real coefficients such that*

$$P(x, y(x), ..., y^{(n)}(x)) = 0,$$

for every $x \in I$.

We now present the first version of an important result, establishing fundamental links with other fields of mathematics, that was first presented in [Sha41].

Claim 1.1.4 (Shannon) *A unary function can be generated by a GPAC if and only if the function is differentially algebraic.*

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

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

could not be generated (cf. [Car77, pp. 49,50]). Nevertheless, the claim 1.1.4 gives us a nice characterization of the class of functions computed by the GPAC. Unfortunately, the original proof of claim 1.1.4 contained some gaps, as indicated on pp. 13-14 of [Pou74].

1.2 Examples

In order to familiarize the reader with the GPAC, we will give three examples in this section.

Example 1: We will start by analyzing the circuit represented in figure 1.2.1 (taken from [Cam02]).

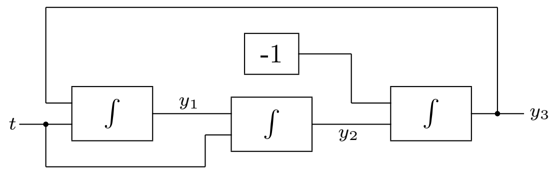


Figure 1.2.1: A circuit that computes both sin and cos.

The box with -1 inside represents a circuit that outputs the value -1. It can be obtained by connecting a constant function unit to a constant multiplier unit associated to the value -1. We can easily see that y_1, y_2, and y_3 are functions of t satisfying the following system of equations

$$\begin{cases} y_1' = y_3, \\ y_2' = y_1, \\ y_3' = -y_2'. \end{cases}$$

Solving it, we get the solutions expressed by

$$\begin{cases} y_1 = c_1 \cos + c_2 \sin, \\ y_2 = c_1 \sin - c_2 \cos + c_3, \\ y_3 = -c_1 \sin + c_2 \cos. \end{cases}$$

In particular, if we set the initial conditions of the integrators to $y_1(0) = 1$, $y_2(0) = 0$, and $y_3(0) = 0$, we have $y_1 = \cos$, $y_2 = \sin$, and $y_3 = -\sin$. Hence, we can compute the functions cos and sin.

Note that, as indicated in claim 1.1.4, sin and cos are d.a. functions. In fact, sin and cos are solutions of

$$y'' + y = 0.$$

Example 2: Consider the circuit pictured in figure 1.2.2.

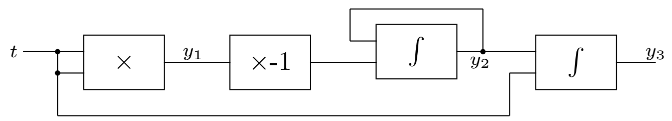


Figure 1.2.2: A circuit that computes the function with expression $\int_0^t e^{-v^2} dv$.

Associated to it, we have the following system of equations

$$\begin{cases} y_1 = \lambda t.t^2, \\ y_2' = -y_2 y_1', \\ y_3' = y_2. \end{cases}$$

If we take $y_2(0) = 1$ and $y_3(0) = 0$, we get the following expressions for the solutions: $y_1(t) = t^2$, $y_2(t) = e^{-t^2}$, and $y_3(t) = \int_0^t e^{-v^2} dv$. Note that y_3 is a d.a. function because, for each $t \in \mathbb{R}$,

$$2ty_3'(t) + y_3''(t) = 0.$$

Note also that y_3 is not an elementary function in the following sense (see [CJ89, p. 261], [Zwi96, p. 352]): its expression cannot be obtained by repeated use of addition, multiplication, division, and composition of the usual functions (power, exponential, trigonometric, and rational functions and their inverses) in a finite number of times. Hence, if we want to calculate the value of y_3 for some point t in actual computers, we must apply some procedure such as Simpson's rule to approximate numerically its value. Nevertheless, as we have just seen, this function can be computed *exactly* by a GPAC.

Although a primitive of $\lambda v.e^{-v^2}$ is not an elementary function, it is not difficult to show (using polar coordinates) that $\int_{-\infty}^0 e^{-v^2} dv = \frac{\sqrt{\pi}}{2}$. Then we can easily adapt the circuit of figure 1.2.2 to compute the integral $\int_{-\infty}^t e^{-v^2} dv$ associated to the Gaussian distribution. This distribution finds many applications in the theory of probability and also in physics (e.g. the Maxwell-Boltzmann distribution of velocities).

Example 3: Consider the circuit represented in figure 1.2.3.

It is not difficult to see that x, y, and z are solutions of the following system of ODEs

$$\begin{cases} x' = -10x + 10y, \\ y' = 28x - y - xz, \\ z' = -\frac{8}{3}z + xy. \end{cases}$$

This system describes a simplified model of thermal convection. It was obtained by E. Lorenz in 1963 while studying a rudimentary model of weather. Associated to this system of ODEs, a chaotic dynamical system with a sensitive dependence on initial conditions occurs [McC94, p. 45]. By other words, if we pick two nearby initial conditions, the respective solutions will drive away very fast with the increment of t.

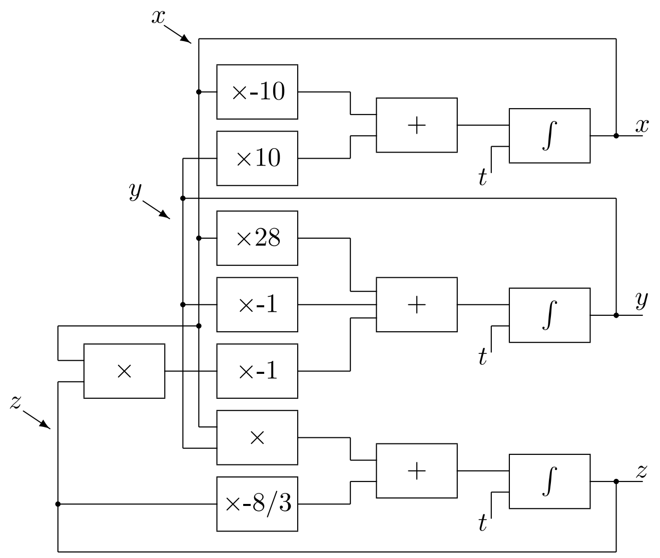


Figure 1.2.3: A circuit that computes the solutions of Lorenz's system of equations. This system is chaotic and has a sensitive dependence on initial parameters.

Hence, if we would like to simulate this system in a computer, we must deal with numbers exactly instead of approximations, as it is done in usual practice, in order to avoid the fast increase of error (because of errors introduced during the calculations). This cannot be done by standard methods in Turing machines (at least without using some unknown properties), but can be implemented in a GPAC.

Then, Turing machines are unable to simulate properly some dynamical systems that can be simulated by the GPAC and can only approximate them up to a certain point in time. Of course, this advantage doesn't exist in practice: in a physical system there are always some perturbations that will prevent the computation to be exact. Nevertheless, this problem only arises at the practical level, not at a theoretic one, as it appears to be in Turing machines.

1.3 Further Developments

Although the previous notion of GPAC seems fairly intuitive and natural, this is not the notion that people actually refer when talking about the GPAC. The accepted definition is due to Pour-El and was introduced in [Pou74]. As she claims in her paper, Shannon's proof of claim 1.1.4 is rather incomplete: *"A statement somewhat similar to this [claim 1.1.4] appears as part of theorem II [Sha41, p. 342]. We believe that there is a serious gap in the brief proof which appears on the top of p. 343"* (footnote in p. 13). And she continues: *"For this reason we have found it necessary to proceed along entirely different lines"* (footnote in p. 13). *"To fix this gap it was not only necessary to change Shannon's proof but to introduce the previously mentioned definition..."* (footnote in p. 4). So, the main reason for a new definition for the GPAC is to keep some relations of the type indicated in claim 1.1.4.

Let us now present the new definition of Pour-El that we will denominate as *theoretic GPAC* (or simply T-GPAC). This is not a usual notation, but we prefer to introduce it in order to distinguish this model from the previous one. In the following I will denote a closed bounded interval with non-empty interior. We now introduce the concept of function generated by a T-GPAC for functions of one variable.

Definition 1.3.1 *The unary function y is generated by a T-GPAC on I if there exists a set of unary functions $y_1, ..., y_n$ and a set of initial conditions $y_i(a) = y_i^*$, $i = 1, ..., n$, where $a \in I$, such that:*

1. *$\mathbf{y} = (y_1, ..., y_n)$ is the unique solution on I of a system of ODEs of the form*
$$A(x, \mathbf{y}) \frac{d\mathbf{y}}{dx} = b(x, \mathbf{y}) \tag{1.1}$$
satisfying the initial conditions, where $A(x, \mathbf{y})$ and $b(x, \mathbf{y})$ are $n \times n$ and $n \times 1$ matrices, respectively. Furthermore, each entry of A and b must be linear in $1, x, y_1, ..., y_n$.

2. *For some $i \in \{1, ..., n\}$, $y = y_i$ on I.*

3. *$(a, y_1^*, ..., y_n^*)$ has a domain of generation with respect to the equation (1.1), i.e., there are closed intervals $J_0, J_1, ..., J_n$ (with non-empty interiors) such that $(a, y_1^*, ..., y_n^*)$ is an interior point of $J_0 \times J_1 \times ... \times J_n$ and, furthermore, whenever $(b, z_1^*, ..., z_n^*) \in J_0 \times J_1 \times ... \times J_n$ there exists unary functions $z_1, ..., z_n$ such that:*
(i) $z_i(b) = z_i^*$ for $i = 1, ..., n$;
(ii) $(z_1, ..., z_n)$ satisfy (1.1) on some interval I^* with non-empty interior such that $b \in I^*$;
(iii) $(z_1, ..., z_n)$ is unique on I^*.

The existence of a domain of generation indicates that the solution of (1.1) remains unique for sufficiently small changes on the initial conditions. Provided with this definition, Pour-El shows the following:

Theorem 1.3.2 (Pour-El) *Let y be a differentially algebraic function on I. Then there exists a closed subinterval $I' \subseteq I$ with non-empty interior such that, on I', y can be generated by a T-GPAC.*

She also proves, although with some corrections made by Lipshitz and Rubel in [LR87], the following result:

Theorem 1.3.3 (Pour-El, Lipshitz, Rubel) *If y is generable on I by a T-GPAC, then there is a closed subinterval $I' \subseteq I$ with non-empty interior such that, on I', y is differentially algebraic.*

These results present a valid variation of claim 1.1.4. Hence, because of this important connection, the T-GPAC became a significant model. But why people actually replace the GPAC by the T-GPAC? Let us quote Pour-El: *"At first sight the relation between our definition and existing analog computers may seem a bit strange. In order to see that this is a natural generalization which includes existing GPAC we proceed as follows. First we give our definition of [T-GPAC]... This is followed by a discussion of a preliminary definition [of the GPAC]... Finally we relate our preliminary definition to the final definition..."* (pages 7-8). She actually indicates the following result ([Pou74], proposition 1):

Claim 1.3.4 *If a function y is generated on I by a GPAC, it is generated on I by a T-GPAC.*

This result is the main reason why people talk mainly in the T-GPAC instead of the GPAC. According to Pour-El, T-GPAC includes GPAC, being possibly a broader model. Moreover, the T-GPAC has suitable properties shown by theorems 1.3.2 and 1.3.3.

In that proposition she, in fact, presents an argument where she states that if y is generated by a GPAC, then there exists functions $(y_1, ..., y_n)$ satisfying equation (1.1) and condition 2 of definition 1.3.1. But she never shows that $(y_1, ..., y_n)$ is the unique solution of (1.1) and she also does not show condition 3. She only gives a physical argument to condition 3 on p. 12 when defining domain of generation, but does not give any formal proof. Hence, we believe that this argument is rather incomplete and consequently we will not consider claim 1.3.4 as a valid result. This is the reason why we have used distinct definitions: GPAC and T-GPAC.

Furthermore, the T-GPAC seems somehow problematic when dealing with several variables. Pour-El does not even give an appropriate definition to this case saying only that *"It ought to be remarked that Definition [1.3.1] can be extended to cover the case in which* $[y_1, ..., y_n]$ *are functions of more than one variable in an obvious way"* (p. 13). This is not so obvious! There is no known connections with circuits for this case (at least for the one variable case, Pour-El states the claim 1.3.4) that gives "an obvious generalization." The only definition to this case, as far as we know, was presented in [CMC00] where $\frac{d\mathbf{y}}{dx}$ is substituted by the jacobian matrix. Nevertheless, this model has the disadvantage of not having some physical counterpart as the GPAC and being consequently less natural.

In order to complete this survey, we must talk about Rubel's *Extended Analog Computer* (EAC) [Rub93]. This model is similar to the GPAC, but we now also allow, in addition, other types of units, e.g. units that solve boundary value problems (here we allow several independent variables because Rubel is not seeking any equivalence with existing models). “Roughly speaking, the EAC permits all the operations of ordinary analysis, except the unrestricted taking of limits” [Rub93, p. 41]. The new units add an extended computational power relatively to the T-GPAC. For example, the EAC can solve the Dirichlet problem for Laplace’s equation in the disk (it is known that the T-GPAC cannot solve this problem [Rub88]) and can generate the Γ function. In fact, the EAC seems too broad to be implemented in practice. As the author himself states, it is not known if it exists a physical version of the EAC. Hence, this is an unpopular model.

So far we have introduced the existing theory concerning the GPAC together with main results and problems. One objective of this dissertation is to clarify and give answers to some of the problems presented. This is going to be done in the following. We will give an alternative approach to the T-GPAC, keeping circuits as intuitive model, and maintaining theorems 1.3.2 and 1.3.3 as valid results. We will also be able to manage several inputs when they all depend on time.

Chapter 2

GPAC Circuits

2.1 Problems of the GPAC

As we had occasion to discuss on the previous chapter, the GPAC suffers from a severe disadvantage with respect to the T-GPAC: we don't have any known relationship between GPAC computable functions and d.a. functions.[1] Unfortunately, this is not the only important question that remains unsolved. In this section we will talk about other unsolved problems.

1. Given a GPAC and its inputs, is the output of every unit unique? This may seem a silly question. If we think in terms of a differential analyzer, we know that, due to physical restrictions, these outputs must be unique. But the GPAC is an idealization and we don't know the answer for this question.[2] Notice that we can associate to each unit an equation. For example, if x, y are the inputs associated to the variable of integration and to the integrand of an integrator unit, respectively, and z is the output, then the equations associated to the integrator are

$$\begin{cases} z' = yx', \\ z(t_0) = z_0 \end{cases}$$

   (we are supposing that x' exists). Proceeding in this way with respect to all units, we get a system of equations to be satisfied by all outputs. The particular type of equations depends on the units used. On the case of the GPAC, the system of equations will be mixed, where some equations are differential (with initial conditions).[3]
   So, we can associate a system of equations to every GPAC. The question is the following: is the solution of this system unique? It is not possible to answer this question directly. We even don't know if such a solution exists

[1]Except Shannon's claim 1.1.4.
[2]However, we will be able to supply the answer for a submodel of the GPAC.
[3]Hence, we may see a function generated by a GPAC as a solution of a system of equations.

(Notice that for the case of the T-GPAC we don't have this problem: Pour-El avoids the problem by assuming *a priori* the uniqueness of a solution).

2. If the inputs of a GPAC are some unary functions $\varphi_1, ..., \varphi_n$ of class C^k on some interval I, for $k \in \mathbb{N}$, what will happen to the outputs?[4] Will they remain in the same class?

3. Does a GPAC have a domain of generation in the sense of Pour-El (see definition 1.3.1), i.e., for sufficiently small changes in the initial conditions, does the outputs of every unit remain unique?[5]

4. What is the influence of small perturbations in the GPAC? With this question we are, of course, interested in knowing what is the influence of noise in this model. We would also like to know if we can compute with any preassigned precision. We can do this with a digital computer: simply build a circuit that handles with a sufficient number of bits.[6] But with the GPAC we cannot take this approach. In fact, in a differential analyzer we can improve materials and units to some extent, but not beyond a certain limit, and usually at a great cost. This is certainly an interesting and important question.

We will now give a further insight into some of the questions presented. We begin with the problem of non-uniqueness of an output of a GPAC.

Consider the circuit represented on figure 2.1.1.

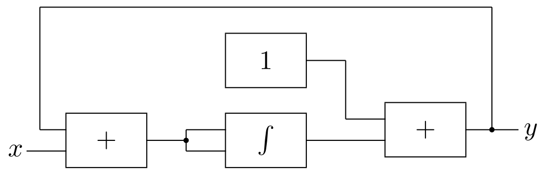


Figure 2.1.1: A circuit that admits two distinct solutions as outputs.

The output that we will analyze is the output of the adder at the bottom that we will call y. Similarly to the examples analyzed in the previous chapter, it is not difficult to see that y at time t is given by

[4]In our notation, $0 \in \mathbb{N}$.

[5]We suppose that the original initial conditions yield unique outputs.

[6]This is partially true. Although this happens for several important algorithms, there are exceptions such as the finite difference methods for solving ODEs [Atk89]. In these methods approximate values are obtained for the solution at a set of grid points

$$x_0 < x_1 < ... < x_n < ...$$

and the approximate value at each x_n is obtained by using some of the values obtained in previous steps. Because we are considering a discrete set of points, an error is introduced in the approximations even if we work with exact numbers. However, if we solve an ODE with a GPAC the same does not happen because we do not use these methods. This is an advantage towards standard numerical methods.

$$y(t) = 1 + \int_0^t (y(x) + x) d(y(x) + x).$$

We are supposing that $t_0 = 0$ and that the initial output of the integrator is 0. When we start the computation, we get two possible solutions:

$$y_\pm = \lambda t.1 \pm \sqrt{-2t} - t.$$

In fact,

$$\begin{aligned} 1 + \int_0^t (y_\pm(x) + x) d(y_\pm(x) + x) = \\ = 1 + \int_0^t (1 \pm \sqrt{-2x} - x + x) d(1 \pm \sqrt{-2x} - x + x) = \\ = 1 \pm \sqrt{-2t} - t. \end{aligned}$$

Note that y' is not defined for the initial value ($t_0 = 0$), being one of the reasons allowing the two solutions. So, we have a non-deterministic circuit.[7]

From the mathematical point of view, this seems not to be problematic. The circuit is a specification of the functions generated. Namely, it may be seen as a way to represent graphically a system of ODEs. But we have no reason to assume neither the existence nor the uniqueness of solutions for this system.

The non-determinism of a GPAC is problematic when we take the physical point of view. If we implement the circuit indicated above in a differential analyzer, what would happen? The circuit will only be able to output *one* solution, so which solution will be picked up, if any? We do believe that the non-determinism will be eradicated when we subject the above differential analyzer not only to the conditions indicated in figure 2.1.1, but also to the known physical laws. In fact, when we deal, for example, with interconnected shafts, there are situations where they can all be blocked up. This situation is not taken in account in the GPAC (as an idealization of mechanical differential analyzers). Hence, we believe that if we implement the circuit of figure 2.1.1 in a mechanical differential analyzer, the resulting output will be 1 because we will be unable to turn the input shaft (it will be blocked up). We will be naturally concerned in filtering this kind of behavior.

Another situation not referred above is presented in figure 2.1.2. The inputs are x, y, with $x(t_0) = y(t_0) = 0$. We also take the output of the integrator to be 0 at $t = t_0$. We may take, without loss of generality, $t_0 = 0$ (simply compose the functions defining x and y with $t - t_0$). Suppose that the computation runs for values of time between 0 and 1. Suppose also that $x(1) = y(1) = 1$. Then, when the computation is over, we have

$$z(1) = \int_0^1 \left(x(t)^2 + y(t)\right) d(x(t) + y(t)).$$

[7]In the sense that we have two distinct outputs.

Now, if we apply the functions $\varphi_1 = \lambda t.t$ and $\varphi_2 = \lambda t.t^2$ to the inputs x and y, respectively, during the interval of time $[0, 1]$, we get

$$\begin{aligned} z(1) &= \int_0^1 (t^2 + t^2)dt + \int_0^1 (t^2 + t^2)2tdt = \\ &= \frac{5}{3}. \end{aligned}$$

On the other side, if we now apply φ_1 to y and φ_2 to x, we get

$$z(1) = \int_0^1 (t^4 + t)d(t^2 + t) = \frac{17}{10}.$$

But $\frac{5}{3} \neq \frac{17}{10}$!

So, in this circuit, a path dependent behavior occurred: the value of an output depends not only on the present values of the inputs, but also on the path followed by them. So, if we take several inputs in a GPAC, the approaches of taking them as independent variables, or taking them as functions of time are different. Here, we cannot say that $z = \lambda xy.z(x, y)$.

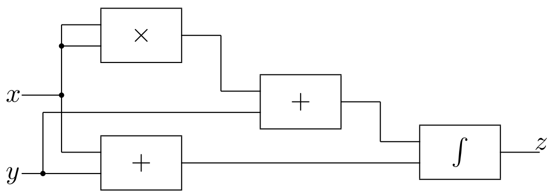


Figure 2.1.2: A circuit with a path dependent behavior.

We also see that in the previous case it is more correct to say that "z is computable by the circuit of figure 2.1.2, relatively to the functions φ_1 and φ_2" (or φ_2 and φ_1, depending on the situation).

2.2 FF-GPAC Circuits

In this section we will introduce a new model inspired by the GPAC. This model has many desirable properties and permits us to tackle questions raised on the previous section.

We had occasion to see that the output of a unit in a GPAC can be non-unique. We believe that this is due to the lack of restrictions of the GPAC, that allows connections with arbitrarily feedback. As we have observed, feedback is desirable since, without it, we could get an uninteresting model. However, we believe that too much feedback can be harmful.[8] So, in this new model, we will restrict the kind of feedback allowed.

[8]In particular, we believe that this is the cause for the abnormal behavior of circuit 2.1.1.

Let us first give a brief overview of the standard procedure used to deal with the GPAC (with one independent variable x). As we have already noticed in remark 1.1.2, we can remove multipliers from the definition of the GPAC. So, every GPAC can be built only with adders, constant multipliers, constant function units, and integrators. We can also consider that every output of a GPAC is the output of an integrator (see remark 1.1.1 on p. 8). Proceeding in a similar way as in [Sha41] (or [Pou74]), let $\mathcal{U}$ be a GPAC with $n-1$ integrators $\mathcal{U}_2, ..., \mathcal{U}_n$, having as outputs $y_2, ..., y_n$, respectively. Let $y_0 = \lambda x.1$ and let $y_1 = \lambda x.x$. Then, each input of an integrator must be the output of one of the following: an adder, an integrator, a constant function unit, or a constant multiplier. Hence, the integrand of $\mathcal{U}_k$ can be expressed as $\sum_{i=0}^{n} c_{ki}^{*} y_i$ and the variable of integration as $\sum_{i=0}^{n} c_{ki}^{**} y_i$, for some suitable constants c_{ki}^{*}, c_{ki}^{**}. Thus, the output of $\mathcal{U}_k$, y_k, can be expressed as

$$y_k = \int_{x_0}^{x} \sum_{i=0}^{n} c_{ki}^{*} y_i d\left(\sum_{j=0}^{n} c_{kj}^{**} y_j\right) + c_k, \quad \text{or}$$

$$y_k = \int_{x_0}^{x} \sum_{i,j=0}^{n} c_{ki}^{*} c_{kj}^{**} y_i dy_j + c_k.$$

We may simplify the last expression by taking $c_{ij}^{k} = c_{ki}^{*} c_{kj}^{**}$. It follows that

$$y_k' = \sum_{i,j=0}^{n} c_{ij}^{k} y_i dy_j, \qquad k = 2, ..., n. \tag{2.1}$$

In this way we get a system of the form (1.1) (p. 13). Hence, we could assert that it is equivalent to say that y is generated by a GPAC (in the sense that it is a solution for a system of equations where each equation is associated to a unit) and that y is a solution of some system of differential equations with the particular structure of (2.1). Although this seems a clear and natural procedure, we believe that we have to be more careful with it. For instance, why should we consider that an input of the integrator $\mathcal{U}_k$ could be expressed as $\sum_{i=0}^{n} c_{ki} y_i$ for some constants c_{ki}? We could have as input the output of a circuit like

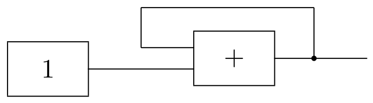


Figure 2.2.1: A circuit that admits no solutions as outputs.

This circuit follows the definition of a GPAC, but we cannot say that its output is a linear combination of the input, because it does not exist. But we would like to keep relations as (2.1). To do so, we will introduce the concept of linear circuit as follows.

Definition 2.2.1 *A* linear circuit *is a GPAC built only with adders, constant multipliers, and constant function units in the following inductive way:*

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

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

Theorem 2.2.2 *If $x_1, ..., x_n$ are the inputs of a linear circuit, then the output of the circuit will be $y = c_0 + c_1x_1 + ... + c_nx_n$, where $c_0, c_1, ..., c_n$ are appropriate constants. Reciprocally, if $y = c_0 + c_1x_1 + ... + c_nx_n$, then there is a linear circuit with inputs $x_1, ..., x_n$ and output y.*

We next introduce a new type of unit that will be necessary on what follows.[9]

- *Input unit* : A zero-input, one-output unit.

The input units may be considered as interface units for the inputs from outside world in order that they can be used by a GPAC (although we may pick an output directly from the GPAC).

We now present the main definition of this section.

Definition 2.2.3 *Consider a GPAC $\mathcal{U}$ with n integrators $\overline{\mathcal{U}}_1, ..., \overline{\mathcal{U}}_n$. Suppose that to each integrator $\overline{\mathcal{U}}_i$, $i = 1, ..., n$, we can associate two linear circuits, $\mathcal{A}_i$ and $\mathcal{B}_i$, with the property that the integrand and the variable of integration inputs of $\overline{\mathcal{U}}_i$ are connected to the outputs of $\mathcal{A}_i$ and $\mathcal{B}_i$, respectively. Suppose also that each input of the linear circuits $\mathcal{A}_i$ and $\mathcal{B}_i$ is connected to one of the following: the output of an integrator or to an input unit. $\mathcal{U}$ is said to be a feedforward GPAC (FF-GPAC) iff there exists an enumeration of the integrators of $\mathcal{U}$, $\mathcal{U}_1, ..., \mathcal{U}_n$, such that the variable of integration of the kth integrator can be expressed as*

$$c_k + \sum_{j=1}^{m} c_{kj}x_j + \sum_{i=1}^{k-1} \bar{c}_{ki}y_i, \qquad \textit{for all } k = 1, ..., n, \tag{2.2}$$

(see fig. 2.2.2) where y_i is the output of $\mathcal{U}_i$, for $i = 1, ..., n$, x_j is the input associated to the jth input unit, and $c_k, c_{kj}, \bar{c}_{ki}$ are suitable constants, for all $k = 1, ..., n$, $j = 1, ..., m$, and $i = 1, ..., k-1$.

[9]These units correspond to the elements of X introduced in the definition given on the footnote of p. 6.

Remark 2.2.4 In a FF-GPAC we can link the output of $\mathcal{U}_k$ directly to the input of $\mathcal{U}_r$ (as long as (2.2) is satisfied) because this is equivalent to have a constant multiplier associated to the value 1 between them.

We can also have a notion of function generated by a FF-GPAC using a straightforward adaptation of the homonymous definition for the case of the GPAC (see p. 6). We can also suppose that each function generated by a FF-GPAC is the output of an integrator (see remark 1.1.1).

Remark 2.2.5 When considering an enumeration $\mathcal{U}_1, ..., \mathcal{U}_n$ of integrators of a FF-GPAC $\mathcal{U}$, we will always assume that it satisfies condition (2.2).

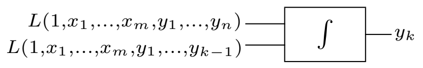

$L(1,x_1,...,x_m,y_1,...,y_n)$
$L(1,x_1,...,x_m,y_1,...,y_{k-1})$
$\int$
y_k

Figure 2.2.2: Schema of the inputs and outputs of $\mathcal{U}_k$ in the FF-GPAC $\mathcal{U}$. $L(z_1, ..., z_r)$ denotes a linear combination (with real coefficients) of $z_1, ..., z_r$.

As examples of FF-GPACs we have all the circuits presented in section 1.2. For the circuits of figures 1.2.1 and 1.2.2, we can take enumerations of integrators starting from the right and going to the left to see that they are FF-GPACs.[10] A similar argument applies to the circuit of figure 1.2.3.

It is clear that if a unary function y is generated by a FF-GPAC, it is generated by a GPAC (because each FF-GPAC is a GPAC) and it satisfies some system of equations with structure similar to (2.1). The converse relation does not hold. Take, for example, the GPAC represented in figure 2.1.1. Applying the standard procedure for a GPAC (see p. 21), we conclude that every output of the circuit will satisfy some system of differential equations similar to (2.1). But this circuit is not a FF-GPAC. We will see that the subclass of the GPAC constituted by FF-GPACs still preserves the desirable properties of the T-GPAC.

Remark 2.2.6 If we consider a FF-GPAC $\mathcal{U}$ with n integrators $\mathcal{U}_1, ..., \mathcal{U}_n$, corresponding to the outputs $z_1, ..., z_n$, respectively, and having inputs $x_1, ..., x_m$, we define $y_0, ..., y_{n+m}$ by

$$y_i = \begin{cases} \lambda x.1 & \text{if } i = 0, \\ x_i & \text{if } 1 \leq i \leq m, \\ z_{i-m} & \text{if } m < i \leq m + n. \end{cases}$$

Then (2.2) can be written as

$$\sum_{i=0}^{k-1} c_{ki} y_i, \qquad \text{for all } k = m+1, ..., m+n,$$

for suitable constants c_{ki}.

[10] In a FF-GPAC, we do not take multiplier units. However, a circuit like the one of figure 1.1.4 may substitute the multiplier unit of circuit 1.2.2.

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, smoothness, or whatever). When we consider the integrator units, one problem still arises: if $I = [a, b]$ is a closed interval, the Riemann-Stieljes integral $\int_I \varphi(t) d\psi(t)$ is not defined for every pair of functions φ, ψ, even if they are continuous.

It is possible to show [Str99, pp. 7,9] that it suffices to consider that ψ is continuously differentiable on I. Then

$$\int_I \varphi(x) d\psi(x) = \int_I \varphi(x) \psi'(x) dx,$$

and

$$\left| \int_I \varphi(x) d\psi(x) \right| \leq K_\psi \left\| \varphi \right\|_\infty ,$$

where $K_\psi = \left\| \psi' \right\|_\infty (b - a)$. So, from now on, we will always assume that the inputs are continuously differentiable functions of the time. And if the outputs of all units are defined for all $t \in I$, where I is an interval, then we will also assume that they are continuous in that interval.[11] This is needed for the following results and may be seen as physical constraints to which all units are subjected.

2.3 Basic Properties

In this section we show that some of the problems referred on section 2.1 can be solved for the FF-GPAC. We first prove a theorem that guarantees that if the inputs are of class C^r, $r \geq 1$, then the outputs are also of class C^r.

Theorem 2.3.1 *Suppose that the input functions of a FF-GPAC are 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.*

Proof. Suppose that we have a FF-GPAC $\mathcal{U}$ with m inputs and n integrators. To show our result we only have to prove that the output of every integrator is of class C^r. We will first show the result for $r < \infty$ using induction on r. We use the notation indicated in remark 2.2.6. Then

$$y_k = \int_{t_0}^{t} \left(\sum_{i=0}^{n+m} c_{ki}^* y_i \right) d \left(\sum_{j=0}^{k-1} c_{kj}^{**} y_j \right) + c_k, \quad k > m,$$

[11]It is not a difficult task to show that this condition can be replaced by the following: if I is closed and if y is the output of an integrator, then there exists some $L > 0$ such that, for $t \in I$, $|y(t)| < L$.

where $c^*_{ki}, c^{**}_{kj}, c_k$, for $i = 0, 1, ..., n+m$, $k = m+1, ..., n+m$, and $j = 0, 1, ..., k-1$, are suitable constants. We begin with $r = 1$. Then

$$y_{m+1} = \int_{t_0}^{t} \left(\sum_{i=0}^{n+m} \sum_{j=1}^{m} c^*_{m+1i} c^{**}_{m+1j} y_i y'_j \right) dt + c_{m+1}.$$

Let $c^k_{ij} = c^*_{ki} c^{**}_{kj}$. The integrand part is continuous and we can therefore apply the Fundamental Theorem of Calculus to conclude that y_{m+1} is differentiable and that

$$y'_{m+1} = \sum_{i=0}^{n+m} \sum_{j=1}^{m} c^{m+1}_{ij} y_i y'_j.$$

So, y_{m+1} is of class C^1. Now suppose that the result is true for all k such that $m < k < p$, for some $p \leq n+m$. Then, by similar arguments, we can show that y_p is differentiable and that

$$y'_p = \sum_{i=0}^{n+m} \sum_{j=1}^{p-1} c^p_{ij} y_i y'_j.$$

Therefore, $y_{m+1}, ..., y_{m+n}$ are all of class C^1. Now suppose that $r > 1$. If the inputs $x_1, ..., x_m$ are of class C^r, then they are also C^{r-1} functions. So, by induction hypothesis, $y_{m+1}, ..., y_{m+n}$ are all of class C^{r-1}. We now show that y_{m+1} is a function of class C^r. We already know that y_{m+1} is differentiable and that

$$y'_{m+1} = \sum_{i=0}^{n+m} \sum_{j=1}^{m} c^{m+1}_{ij} y_i y'_j.$$

But all the functions at the right side are of class C^{r-1}. So, we can differentiate the right expression $r-1$ times obtaining a continuous function. Then y_{m+1} is of class C^r. A similar argument shows that the functions $y_{m+1}, ..., y_{m+n}$ are all of class C^r. Finally, if $r = \infty$, then all the derivatives of $y_{m+1}, ..., y_{m+n}$ exist and, hence, these functions are of class C^∞. ■

Next we will focus to the problem of existence and uniqueness of outputs. We will need the following theorem (theorem 6.11 of [BR89]).

Theorem 2.3.2 *Let $\mathbf{f}(\mathbf{x}, t)$ be defined and of class C^1 in an open region $\mathcal{R}$ of $\mathbb{R}^{n+1}$. For any $(\mathbf{c}, a)$ in the region $\mathcal{R}$, the differential equation*

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

has a unique solution $\mathbf{x}(t)$ satisfying the initial condition $\mathbf{x}(a) = \mathbf{c}$ and defined for an interval $a \leq t < b$ (where b may eventually be ∞) such that, if $b < \infty$, either $\mathbf{x}(t)$ approaches the boundary of the region, or $\mathbf{x}(t)$ is unbounded as $t \to b$.

Remark 2.3.3 If we apply the previous theorem to the function $\mathbf{g} = \lambda \mathbf{x} t.\mathbf{f}(\mathbf{x}, -t)$ and the theorem 6.8 of [BR89], we can prove a similar result, but for an interval (b_0, b_1), where b_0, b_1 are constants such that $b_0 < a < b_1$.

Next, we show a theorem that guarantees the existence and the uniqueness of outputs for FF-GPACs with *only one input*.

Theorem 2.3.4 *Suppose that we have a FF-GPAC with only 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 $t \to t^*$.*

Proof. Suppose that we have a FF-GPAC $\mathcal{U}$ with $n-1$ integrators. Using the notation of theorem 2.3.1 and proceeding in a similar way, we conclude that

$$y_k' = \sum_{i=0}^{n} \sum_{j=1}^{k-1} c_{ij}^k y_i y_j', \qquad k = 2, ..., n,$$

where c_{ij}^k are suitable constants. We can write this as

$$y_k' - \sum_{j=2}^{k-1} \left(\sum_{i=0}^{n} c_{ij}^k y_i \right) y_j' = \sum_{i=0}^{n} c_{i1}^k y_i, \qquad k = 2, ..., n.$$

This system may be written in the following way

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ -\sum_{i=0}^n c_{i2}^3 y_i & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -\sum_{i=0}^n c_{i2}^n y_i & -\sum_{i=0}^n c_{i3}^n y_i & \cdots & 1 \end{bmatrix} \begin{bmatrix} y_2' \\ y_3' \\ \vdots \\ y_n' \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^n c_{i1}^2 y_i \\ \sum_{i=0}^n c_{i1}^3 y_i \\ \vdots \\ \sum_{i=0}^n c_{i1}^n y_i \end{bmatrix}$$

or simply

$$A\mathbf{y}' = \mathbf{b}, \qquad \text{where } \mathbf{y} = \begin{bmatrix} y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

It is easily seen that $\det A = 1$. Hence A is invertible and we have

$$A^{-1} = \frac{1}{\det A} A_{cof},$$

where A_{cof} is the transpose of the matrix in which each entry is its respective cofactor with respect to A (cf. [Str88]). So, each entry in A^{-1} is a polynomial in $x, y_2, ..., y_n$. The same happens for $\mathbf{b}$. We know that

$$\mathbf{y}' = A^{-1}\mathbf{b}.$$

Then we may write

$$\mathbf{y}' = \mathbf{p},$$

where each component of $\mathbf{p}$ is a polynomial in $x, y_2, ..., y_n$, defined on all $\mathbb{R}^{n+1}$. Hence, by remark 2.3.3, if $x(t_0) = x_0$, then there exists an interval (a, b), with $a < x_0 < b$, where the solution of the previous system associated to the initial condition $\mathbf{y}(x_0) = \mathbf{y}_0$ exists and is unique. Hence, because $\mathcal{R} = \mathbb{R}^{n+1}$, we have $b = \infty$ or $\mathbf{y}(x)$ is unbounded as $x \to b$. A similar result holds for a.

Let

$$A_0 = \{t \in [t_0, t_f) : a < x(t) < b\}.$$

Because we assumed that the inputs were continuous functions of the time and that (a, b) is an open set, we conclude that A_0 is also open (in order to the subspace topology of $[t_0, t_f)$. For the topological results indicated here, refer to [Mun00, Lip65]). But $[t_0, t_f)$ is locally connected and, hence, each component of A_0 is open on $[t_0, t_f)$ (theorem 25.3 from [Mun00]). Let A be the component of A_0 such that $t_0 \in A$. A must be open. Hence $A = [t_0, t^*)$, where t^* is possibly t_f. If $t^* < t_f$ then, because $t^* \notin A$ and A is a component, we conclude that $x(t^*) \notin (a, b)$. But if $\{t_n\}_{n\in\mathbb{N}}$ is a sequence in A that converges to t^*, then $x(t_n) \in (a, b)$. Because x is continuous, it must be $x(t^*) = a$ or $x(t^*) = b$. So, when $\{t_n\}_{n\in\mathbb{N}}$ is a sequence in A that converges to t^* (and $t^* < \infty$), we conclude that $x(t_n)$ converges to a or to b. Suppose, without loss of generality, that it converges to b. Then $b < \infty$ because $x(t)$ is defined for $t \in [t_0, t_f)$ and $b = x(t^*)$, with $t^* \in [t_0, t_f)$. But if $b < \infty$, then $\mathbf{y}(x)$ is unbounded as $x \to b$. Hence, from

$$\lim_{n\to\infty} x(t_n) = b$$

we conclude that $\mathbf{y}(t_n) \equiv \mathbf{y}(x(t_n)))$ is unbounded as $t_n \to t^*$. ■

The previous result applied only to FF-GPACs with one input. Next we present a slightly weaker result for FF-GPACs with more than one input.

Theorem 2.3.5 *Consider a FF-GPAC with m inputs $x_1, ..., x_m$ of class C^2 on an interval $[t_0, t_f)$, where t_f may possibly be ∞. Then there exists an interval $[t_0, t^*)$ (with $t^* \le t_f$) where each output exists and is unique. Moreover, if $t^* < t_f$ then there exists an integrator with output y such that y is unbounded as $t \to t^*$.*

Proof. Suppose that we have a FF-GPAC $\mathcal{U}$ with n integrators $\mathcal{U}_1, ..., \mathcal{U}_n$, with outputs $y_1, ..., y_n$, respectively. Then, for $k = 1, ..., n$, we have

$$y_k = \int_{t_0}^{t} \left(\sum_{i=0}^{m} c_{ki}^* x_i + \sum_{j=1}^{n} c_{kj} y_j \right) d\left(\sum_{r=0}^{m} b_{kr}^* x_r + \sum_{s=1}^{k-1} b_{ks} y_s \right) + c_k,$$

where $c_{ki}^*, c_{kj}, b_{kr}^*, b_{ks}, c_k$ are real constants. Let

$$\varphi_k = \sum_{i=0}^{m} c_{ki}^* x_i, \quad \psi_k = \sum_{r=0}^{m} b_{kr}^* x_r, \quad k = 1, ..., n.$$

By the Fundamental Theorem of Calculus, we get

$$y_k' = \left(\varphi_k + \sum_{j=1}^{n} c_{kj} y_j \right) \left(\psi_k' + \sum_{s=1}^{k-1} b_{ks} y_s' \right),$$

for $k = 1, ..., n$. Rewriting the last expression, we obtain

$$y'_k - \sum_{s=1}^{k-1} \left(\varphi_k + \sum_{j=1}^{n} c_{kj} y_j \right) b_{sk} y'_s = \left(\varphi_k + \sum_{j=1}^{n} c_{kj} y_j \right) \psi'_k \tag{2.3}$$

for $k = 1, ..., n$. Let

$$A = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -\left(\varphi_2 + \sum_{j=1}^{n} c_{2j} y_j\right) b_{12} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -\left(\varphi_n + \sum_{j=1}^{n} c_{nj} y_j\right) b_{1n} & -\left(\varphi_n + \sum_{j=1}^{n} c_{nj} y_j\right) b_{2n} & \cdots & 1 \end{bmatrix}$$

and

$$\mathbf{b} = \begin{bmatrix} \left(\varphi_1 + \sum_{j=1}^{n} c_{1j} y_j\right) \psi'_1 \\ \vdots \\ \left(\varphi_n + \sum_{j=1}^{n} c_{nj} y_j\right) \psi'_n \end{bmatrix}.$$

Then, the system (2.3) may be written as

$$A\mathbf{y}' = \mathbf{b}.$$

Note that, because $x_1, ..., x_m, y_1, ..., y_n$ are C^2-functions, each component of $\mathbf{b}$ is continuously differentiable (in order to t) and each element of A is of class C^2. We also have $\det(A) = 1$. Hence A is invertible and each entry of A^{-1} can be obtained from $x_1, ..., x_m, y_1, ..., y_n$ using only products and additions (remember that φ_k and ψ_k, $k = 1, ..., n$, are obtained in this way). Then each component of A^{-1} is of class C^2 and

$$\mathbf{y}' = A^{-1}\mathbf{b} = \mathbf{f},$$

where each component of $\mathbf{f}$ is continuously differentiable. Now we could apply theorem 2.3.2 to conclude the result. But one condition fails: the domain of $\mathbf{f}$, $[t_0, t_f)$, is not open. Nevertheless, if we extended the inputs $x_1, ..., x_n$ to the interval $(-\infty, t_f)$, maintaining the condition of being twice differentially continuous, we have our problem solved. Therefore we only have to extend the inputs. We may do this by picking

$$\bar{x}_k = \lambda t. \begin{cases} c_0 + c_1(t - t_0) + c_2(t - t_0)^2, & \text{if } t \in (-\infty, t_0), \\ x_k, & \text{if } t \in [t_0, t_f), \end{cases}$$

for $k = 1, ..., m$, where $c_0 = x_k(t_0)$, $c_1 = x'_k(t_0)$, and $c_2 = \frac{x''_k(t_0)}{2}$. It is a trivial exercise to verify that $\bar{x}_k \in C^2((-\infty, t_f))$ for $k = 1, ..., m$. ■

The previous theorems applied on an interval of time $[t_0, t_f)$, where t_0 is the initial value. This correspond to the physical notion of time, where we cannot, in principle, go back in time (irreversible systems). But as we already referred, we consider the parameter time as a broader concept, only assuming that it is an independent variable. It could correspond to electric voltage, displacement, etc. Hence, it is natural to consider the case in which t belongs to an open interval I, where $t_0 \in I$.

It is possible to adapt the previous arguments in order to show the following results.

Theorem 2.3.6 *Suppose that we have a FF-GPAC with only one input x, of class C^1 on an interval (t_i, t_f), with $t_0 \in (t_i, t_f)$, where t_0 is the initial value and t_i, t_f may possibly be $-\infty, \infty$, respectively. Then there exists an interval (t^*, t^{**}), with $t_i \leq t^*$, $t^{**} \leq t_f$, and $t^* < t_0 < t^{**}$, where each output exists and is unique. Moreover, if $t^{**} < t_f$ $(t^* > t_i)$, then there exists an integrator with output y such that y is unbounded as $t \to t^{**}$ $(t \to t^*)$.*

Theorem 2.3.7 *Consider a FF-GPAC with m inputs $x_1, ..., x_m$, of class C^2 on an interval (t_i, t_f), with $t_0 \in (t_i, t_f)$, where t_0 is the initial value and t_i, t_f may possibly be $-\infty, \infty$, respectively. Then there exists an interval (t^*, t^{**}), with $t_i \leq t^*$, $t^{**} \leq t_f$, and $t^* < t_0 < t^{**}$, where each output exists and is unique. Moreover, if $t^{**} < t_f$ $(t^* > t_i)$, then there exists an integrator with output y such that y is unbounded as $t \to t^{**}$ $(t \to t^*)$.*

The results shown above indicate that our model is "well behaved." This is an advantage towards the GPAC, but this is not enough. We would like to know which are the computing capabilities of this model. This is going to be done in the next section.

2.4 Connections Between Models

In this section we analyze the connections between the FF-GPAC and other models of analog computation. In particular, we will establish links with the T-GPAC and with the differentially algebraic functions. In the following I will be a closed, bounded interval with non-empty interior. We will only consider a FF-GPAC with one input x, the independent variable, and we will also only consider functions of class C^∞.

Definition 2.4.1 *$\mathbb{R}[x_1, ..., x_n]$ denotes the set of n-ary polynomials with real coefficients, for $n \in \mathbb{N}$.*

Definition 2.4.2 *Let y be a differentially algebraic function on J. For each $n \in \mathbb{N}$, let*

$$Pol_n(y) = \{P \in \mathbb{R}[x_1, ..., x_{n+2}] : P \neq 0 \text{ and } \forall x \in J,\ P(x, y(x), ..., y^{(n)}(x)) = 0\}.$$

The order *of y on J is given by*

$$order(y) = \inf\{n \in \mathbb{N} : Pol_n(y) \neq \varnothing\}.$$

Theorem 2.4.3 *If y is generable on I by a FF-GPAC with n integrators, then there exists a nonzero polynomial p with real coefficients such that*

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

Proof. Suppose that we have a FF-GPAC $\mathcal{U}$ with n integrators $\mathcal{U}_1, ..., \mathcal{U}_n$ in an appropriate order of enumeration, with outputs $y_1, ..., y_n$, respectively. We only have to show that the functions $y_1, ..., y_n$ are d.a. on I. We will show that y_1 is d.a.. Proceeding along similar lines to the proof of theorem 2.3.4, we conclude that

$$\begin{aligned} y_1' &= \bar{P}_1 \\ &\vdots \\ y_n' &= \bar{P}_n \end{aligned} \tag{2.4}$$

where each $\bar{P}_i$ is a polynomial in $x, y_1, ..., y_n$, for $i = 1, ..., n$. Differentiating $y_1' = \bar{P}_1$ with respect to x and using (2.4), we get

$$y_1^{(k)} = P_k, \qquad k = 1, ..., n, \tag{2.5}$$

where each P_k is a polynomial in $x, y_1, ..., y_n$. Now consider the field of rational functions in $x_1, ..., x_n$ over $\mathbb{R}$,

$$\mathbb{R}(x_1, ..., x_n) = \left\{ \frac{p(x_1, ..., x_n)}{q(x_1, ..., x_n)} : p, q \in \mathbb{R}[x_1, ..., x_n], \text{ and } q \neq 0 \right\},$$

(for the results on algebra, cf. [Hun96, pp. 311-317]). It is easy to see that $x, y_1 \in \mathbb{R}(x, y_1, ..., y_n)$ and that $P_k \in \mathbb{R}(x, y_1, ..., y_n)$, for $k = 1, ..., n$. But a transcendence base of $\mathbb{R}(x, y_1, ..., y_n)$ can only have $n+1$ elements. Hence, the $n+2$ polynomials x, y_1, P_k, $k = 1, ..., n$ must be algebraically dependent, i.e., there exists a nonzero polynomial $p \in \mathbb{R}[x_1, ..., x_{n+2}]$ such that $p(x, y, P_1, ..., P_n) = 0$. Using (2.5), we get

$$p\left(x, y_1, y_1' ..., y_1^{(n)}\right) = 0,$$

as we wanted to prove. ■

Corollary 2.4.4 *If y is generable on I by a FF-GPAC, then y is differentially algebraic on I.*

Corollary 2.4.5 *Suppose that y is generable on I by a FF-GPAC. Then there exists a closed subinterval $I' \subseteq I$ with non-empty interior such that, on I', y can be generated by a T-GPAC.*

Proof. This result is obtained by a straightforward application of theorem 1.3.2. ■

Theorem 2.4.6 *Suppose that y is differentially algebraic on I. Then there is a closed subinterval $I' \subseteq I$ with non-empty interior such that, on I', y can be generated by a FF-GPAC.*

Proof. This proof follows much along the lines of theorem 4 in [Pou74]. Let n be the order of y and let $P = \lambda x_1 ... x_{n+2}.P(x_1, ..., x_{n+2})$ be a polynomial of lowest degree in x_{n+2} such that

$$P(x, y, y', ..., y^{(n)}) \equiv 0$$

on I. Differentiating formally the last equation (in order to x), we get

$$\frac{\partial P}{\partial x} + \frac{\partial P}{\partial y} y' + \frac{\partial P}{\partial y'} y'' + ... + \frac{\partial P}{\partial y^{(n)}} y^{(n+1)} \equiv 0, \quad \text{or}$$

$$R(x, y, y', ..., y^{(n)}) y^{(n+1)} - Q(x, y, y', ..., y^{(n)}) \equiv 0 \tag{2.6}$$

on I, where R and Q are $n+2$-ary polynomials having the property that R is of lower degree than P in the last variable. Therefore, we must have $R(x, y, y', ..., y^{(n)}) \not\equiv 0$ on I, i.e., there exists some $a \in I$ that satisfies the condition $R(a, y(a), ..., y^{(n)}(a)) \neq 0$. Then we may pick some closed, bounded intervals $J, J_1, ..., J_n$ (with non-empty interiors) such that

(i) $J \subseteq I$,

(ii) $(a, y(a), ..., y^{(n)}(a)) \in J \times J_1 \times ... \times J_n$,

(iii) If $(c_0, ..., c_n) \in J \times J_1 \times ... \times J_n$ then $R(c_0, ..., c_n) \neq 0$,

(iv) If $x \in J$ then $(x, y(x), ..., y^{(n)}(x)) \in J \times J_1 \times ... \times J_n$.

We can rewrite (2.6) as a system of $n+1$ first-order equations, with $n+1$ variables defined on $J \times J_1 \times ... \times J_n$. Hence, this system satisfies a Lipschitz condition in that interval and the solution is unique (cf. with theorem 6.1 and the lemma preceding it in [BR89]). So, y is the unique solution possessing initial conditions $a, y(a), y'(a), ..., y^{(n)}(a)$ which satisfies (2.6) on J. Next, we are going to prove that y can be generated by a FF-GPAC on J. We begin by finding the equations that define the corresponding FF-GPAC. Solving for $y^{(n+1)}$ in (2.6), we get

$$y^{(n+1)} = \frac{Q(x, y, y', ..., y^{(n)})}{R(x, y, y', ..., y^{(n)})}.$$

Introducing the variables $y_1 = y, y_2 = y', ..., y_{n+1} = y^{(n)}, y_{n+2} = y^{(n+1)}$, the previous equation becomes

$$y'_k = y_{k+1}, \quad k = 1, ..., n+1, \tag{2.7}$$
$$y_{n+2} = \frac{Q(x, y_1, ..., y_{n+1})}{R(x, y_1, ..., y_{n+1})}.$$

It is not difficult to show that y can be generated by a FF-GPAC having only one independent variable x iff there are functions $z_2, ..., z_g$ satisfying the relations

$$z'_k = \sum_{i=0}^{g} \sum_{j=1}^{k-1} c_{ki} c_{kj} z_i z'_j, \qquad k = 2, ..., g. \tag{2.8}$$

where c_{ki}, c_{kj} are reals, and $z_0 = \lambda x.1$, $z_1 = \lambda x.x$, with $y = z_i$ for some $i = 2, ..., g$. Although the first $n+1$ equations of (2.7) satisfy (2.8), the same does not happen for the last equation (y_{n+2} is expressed as the quotient of two polynomials). Hence, we will substitute the last equation of (2.7) by a system of equations satisfying (2.8).

Consider the polynomial Q. Each term of Q that is not a constant is of the form $bv_1...v_r$, where each v_i is x or one of the variables y_j and b is a real number. Suppose that the first term is $bv_1...v_r$. Taking

$$y'_{n+3} = bv_1v'_2 + bv_2v'_1$$

with initial condition $y_{n+3}(a) = bv_1(a)v_2(a)$, we have $y_{n+3} = bv_1v_2$. In a similar way, taking

$$y'_{n+4} = v_3y'_{n+3} + y_{n+3}v'_3$$

with initial conditions $y_{n+4}(a) = v_3(a)y_{n+3}(a)$, we get $y_{n+4} = bv_1v_2v_3$. Continuing with this procedure we will have $y_{n+r+1} = bv_1v_2...v_r$. We can apply this method to all non-constant terms of Q and R, obtaining new equations. Let the y's corresponding to the non-constant terms of Q be $w_1, ..., w_s$ and let $w_1^*, ..., w_t^*$ be those corresponding to the non-constant terms of R. Then

$$y_{n+2} = \frac{\left(\sum_{i=1}^s w_i + c\right)}{\left(\sum_{i=1}^t w_i^* + c^*\right)}, \tag{2.9}$$

where c and c^* are real constants. Now, we must reduce the last equation in a form that fits (2.8). Suppose that the last y, w_t^*, was y_{p-1}. Let

$$\begin{aligned} y'_p &= -y_{p+1}\sum_{i=1}^t (w_i^*)', \\ y'_{p+1} &= 2y_py'_p, \end{aligned} \tag{2.10}$$

with initial conditions

$$y_p(a) = \left(\sum_{i=1}^t w_i^*(a) + c^*\right)^{-1}, \quad y_{p+1}(a) = y_p^2(a).$$

Then $y_p = \left(\sum_{i=1}^t w_i^* + c^*\right)^{-1}$, $y_{p+1} = y_p^2$, and we may replace (2.9) by three equations: equations (2.10) and

$$y'_{n+2} = y_p\sum_{i=1}^s (w_i)' + \left(\sum_{i=1}^s w_i + c\right)y'_p,$$

where $y_{n+2}(a) = \left(\sum_{i=1}^{s} w_i(a) + c\right).y_p(a)$. We conclude that (2.7) can be replaced by the following system of equations

$$\begin{aligned}
y_k' &= y_{k+1}, \quad k = 1, ..., n+1 \\
y_{n+3}' &= bv_1v_2' + bv_2v_1', \quad \text{where } v_1, v_2 \in \{x, y_1, ..., y_{n+1}\} \\
&\vdots \\
y_{p-1}' &= y_{p-2}v_m' + v_my_{p-2}', \quad \text{where } v_m \in \{x, y_1, ..., y_{n+1}\} \\
y_p' &= -\sum_{i=1}^{t} y_{p+1}(w_i^*)' \\
y_{p+1}' &= 2y_py_p' \\
y_{n+2}' &= \sum_{i=1}^{s} y_p.w_i' + \left(\sum_{i=1}^{s} w_i + c\right) y_p'
\end{aligned}$$

It is easily seen that this system in on the form of (2.8) (the respective sequence of $z_2, ..., z_g$ is $y_1, ..., y_{n+1}, y_{n+3}, ..., y_{p+1}, y_{n+2}$) and that $y = y_1$ on J. So, y is computable by a FF-GPAC on J. ■

Corollary 2.4.7 *If y is generable on I by a T-GPAC, then there is a closed subinterval $I' \subseteq I$ with non-empty interior such that y is generable by a FF-GPAC on I'.*

Proof. This follows immediately from theorem 1.3.3. ■

A question naturally arises. Can we extend the result presented in theorem 2.4.6 in order to have $I' = I$? This is an interesting question that certainly deserves more attention. On one hand, we believe that a different technic from the one used here is necessary in order to prove that $I' = I$ (if such a result holds). On the other hand, we could perhaps improve this result by using analytic functions and their special properties (including algebraic results), but we didn't touch this topic.

The results of this section show that functions generated by a FF-GPAC with one input are essentially smooth d.a. functions (in the specific sense of subsets). Moreover, functions generated by the FF-GPAC were also shown to be equivalent (also in the specific sense of subsets) to smooth functions generated by the T-GPAC. Hence, although it was believed that the T-GPAC was a more general model than the GPAC [Pou74, p. 7], it seems that it is the opposite case that holds.

The results shown above present many desirable properties and relations. We may also introduce notions of standard computation like complexity, limiting the number of integrators, the kind of connections, etc. For instance, suppose that some unary d.a. function y has order n. Then, by theorem 2.4.3, y cannot be generated by a FF-GPAC with one input using less than n integrators.

Not all functions can be generated by a FF-GPAC. For instance, Γ cannot be generated because it is not a d.a. function. But the following result (an adaptation of theorem V from [Sha41]) permit us to approximate functions of a broader class than d.a. functions.

Theorem 2.4.8 *Let S be a closed, bounded subset of $\mathbb{R}^n$ and let f be a continuous function from S to $\mathbb{R}$. Let $x_1, ..., x_n \in C^1(J)$, where J is an interval such that for every $t \in J$, $(x_1(t), ..., x_n(t)) \in S$. Then $\lambda t.f(x_1(t), ..., x_n(t))$ can be approximated to within any pre-specified error $\varepsilon > 0$, on J, by a FF-GPAC having as inputs $x_1, ..., x_n$.*

Proof. Using the well-known Weierstrass Approximation Theorem (see [RR93, pp. 65-66]), we know that there exists some polynomial p in n indeterminates such that, for all $(x_1, ..., x_n) \in S$,

$$|f(x_1, ..., x_n) - p(x_1, ..., x_n)| < \varepsilon.$$

Hence, on J,

$$|f(x_1(t), ..., x_n(t)) - p(x_1(t), ..., x_n(t))| < \varepsilon.$$

It is a straightforward task to build a FF-GPAC with inputs $x_1, ..., x_n$ that computes $\lambda t.p(x_1(t), ..., x_n(t))$. ■

2.5 Effectiveness

In all our previous models we have implicitly allowed the use of all reals (in computing units), without restrictions. This seems unfeasible because in the real world we cannot discriminate two arbitrary reals.[12] Hence, these models don't seem to be "effective" from the computability point of view. In this section we will give a framework to deal with this topic.

We shall consider the case of the GPAC (that includes the FF-GPAC). In the GPAC, and for every $k \in \mathbb{R}$, if we connect a constant function unit to a constant multiplier with factor k, we obtain a unit that generates the function $C_k = \lambda.k$ (see fig. 2.5.1).

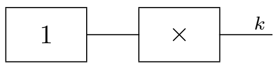


Figure 2.5.1: A constant unit made from a constant function unit and a multiplier unit.

We can then talk of new units, the *constant units*, where each constant unit is a zero-input, one-output unit that is associated to a real number k, outputting the constant function $C_k = \lambda.k$.

We also see that if we allow the constant units to be used by the GPAC, we don't get any increase in the computational power. Nevertheless, we can remove the constant function units and the constant multiplier units, because they can be obtained from constant units and multipliers (see fig. 2.5.2).

[12]However, this is not a consensual assumption. For example, in the BSS model [BCSS98] we can compare two arbitrary reals.

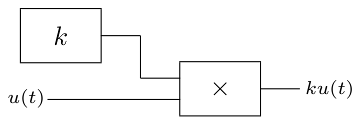


Figure 2.5.2: A constant multiplier built with a constant unit and a multiplier unit.

If we consider the kind of modifications indicated above we can also change the integrator unit. We can assume that the initial output of an integrator is 0 and not an arbitrary pre-fixed number $a \in \mathbb{R}$, without altering the computational power. In fact, one of the "new" integrators is certainly one of the "old" ones. But one of the "old" integrators can be made using an adder, a constant unit, and one of the "new" integrators in a straightforward way (see fig. 2.5.3).

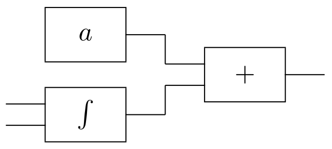


Figure 2.5.3: A circuit using an integrator with initial condition 0 that simulates an integrator with initial condition a.

With this procedure we may only use constant units associated to specific values (e.g. -1,0,1), restricting the GPAC and the FF-GPAC, and obtaining more effective models (in the sense indicated above). Hence, from now on, in the FF-GPAC, we will not consider constant multipliers, constant function units, multipliers, and integrators with initial condition $\neq 0$ (but we do consider constant units).

2.6 The Initialization Procedure

We now move our focus to what we will refer as *the initialization procedure* of a GPAC. This procedure is supposed to be performed before any computation. It can be compared with the differential analyzer (cf. [Bus31]), where we have to set up the device initially and we have, in the case of the mechanical version, to turn the shafts from one initial position to the positions where they should be at the beginning of the computation. Then we make the appropriate connections and we check that the initial displacements of the shafts match properly.

Although this procedure is not referred in existing literature, we believe that it is important and has a role to play in the GPAC (and in the FF-GPAC). In fact, we believe that when we are setting up a differential analyzer, we are already computing, because we need some procedure to make the adequate connections, specially in complicated circuits. However, the GPAC seen as a model of the differential analyzer misses that aspect. Hence, we will introduce a process by which we pick isolated units and then connect them, accordingly to some rule, until the circuit is set up and the main computation can be started.

This will be useful for working with an effective FF-GPAC, as we will see on the following chapter.

When we start an initialization procedure for a GPAC, we suppose that all units are disconnected. We will only use the following units: integrators, constant units, input units, and adders. This follows the results indicated in the previous section. We suppose that initially all the inputs and outputs of the units take the value 0. The only exceptions are the constant units and the input units. We assume that the initial value of the output of a constant unit associated to the value k is k and that the output of an input unit has initially some value a (that corresponds to the initial value that the external input associated to it takes when the main computation starts).

The initialization procedure is done during some interval of time $[t_i, t_f]$.[13] If we want to initialize a GPAC $\mathcal{U}$ with units $\mathcal{A}_1, ..., \mathcal{A}_n$, we start by connecting the units. We assume that we can only establish one connection at once. Then, if we want to interconnect one input of $\mathcal{A}_j$ with the output of $\mathcal{A}_k$, we proceed as follows:

1. If the input and output take the same value, simply connect them.

2. If the input take some value a and the output take some value b, with $a \neq b$, then introduce into the input, during the interval of time $[t'_0, t'_1]$, the unary function x given for each t by

$$x(t) = a + \frac{t - t'_0}{t'_1 - t'_0}(b - a).$$

   When $t = t'_1$, stop introducing the function and establish the connection. Notice that, although the input of an integrator unit may not be of class C^1 (the derivative has singularities in the points t'_0 and t'_1), we still can compute Riemann-Stieljes integrals.[14]

The condition that we can only establish one connection at once means that if two different connections are established in intervals of time $[t_a, t_b]$ and $[t'_a, t'_b]$, then $[t_a, t_b] \cap [t'_a, t'_b] = \varnothing$. So, associated to this procedure of initialization, we can have the type of program sketched in figure 2.6.1.
The instruction "dim" is used to define the units. The parameter a used in the definition of the input unit specifies its initial value. We initialize the circuit by connecting the inputs and outputs one by one. The notation "input($\mathcal{A}_k$,2)" means that we are referring to the input 2 of unit $\mathcal{A}_k$. For example, if $\mathcal{A}_k$ is an integrator, the input 2 may be the input associated to the variable of integration, and the input 1 the one associated to the integrand. In general, the numbering of the inputs may be arbitrary but needs to be specified. The instruction "apply

[13]Note that, because the initialization procedure is assumed to be performed before any computation, the "main computation" must be done in some interval $[\bar{t}_0, \bar{t}_f)$, where $\bar{t}_0$ is the initial value and $\bar{t}_0 \geq t_f$.

[14]However, we can take x to be a C^∞ function. For example, the function σ presented on lemma 3.3.8 could be used for this purpose.

external input($\mathcal{A}_1$)" means that we start the main computation by introducing the input through the input unit $\mathcal{A}_1$. If we have several input units, we can compute simultaneously with several inputs, with some instruction of the type "apply external input($\mathcal{A}_1, \mathcal{A}_2$)." Finally, "send result($\mathcal{A}_n$)" means that we are sending the results of our computation to the exterior through the output of unit $\mathcal{A}_n$. Other variations of these instructions are possible, but we will not go further on this topic.

```
dim ($\mathcal{A}_1, a$) as input unit
dim $\mathcal{A}_2$ as adder
        ⋮
dim $\mathcal{A}_n$ as integrator
begin
    connect input($\mathcal{A}_k$,2) with output($\mathcal{A}_j$)
                    ⋮
    connect input($\mathcal{A}_r$,1) with output($\mathcal{A}_s$)
    apply external input($\mathcal{A}_1$)
    send result($\mathcal{A}_n$)
end
```

Figure 2.6.1: A program for an initialization procedure.

Note that the order by which the inputs are initialized affects the final result. For example, in figure 2.6.2, we want to initialize the integrator indicated, where the inputs associated to the integrand and to the variable of integration are connected to constant units associated to the value 1.

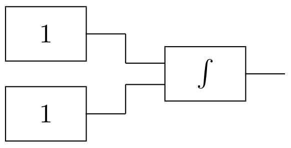


Figure 2.6.2: A circuit in which the order of initialization matters.

If we initialize first the integrand and then the variable of integration, the final output will be 1. If we switch that order, the final output will be 0. So, in order to "program" a device that simulates GPACs, we have to specify the order by which the connections are made. This can be handled by the kind of program already described.

This procedure cannot be applied to all GPACs. For example, the GPAC of figure 2.2.1 cannot be initialized. Nevertheless, we have the following result.

Theorem 2.6.1 *Every FF-GPAC can be initialized.*

Proof. We have to prove that if $\mathcal{U}$ is a FF-GPAC, then $\mathcal{U}$ can be initialized properly. We now give a procedure to do this. Suppose that an appropriate sequence of integrators is $\mathcal{U}_1, ..., \mathcal{U}_n$. We first initialize the integrator $\mathcal{U}_1$ in the following way: first connect the input associated to the variable of integration to the output of the respective linear circuit. Then connect the inputs of this linear circuit. This can obviously be done, because the output of the linear circuit only depends on the values of the inputs of the FF-GPAC. Next connect the input associated to the integrand to the output of the respective linear circuit. Then plug the inputs of this linear circuit to the respective units. This can also be done.

We now proceed by induction. Suppose that we had already connected the integrators $\mathcal{U}_1, ..., \mathcal{U}_{k-1}$, and the linear circuits corresponding to them. We want to do the same for $\mathcal{U}_k$. So, first plug the input of $\mathcal{U}_k$ associated to the variable of integration to its respective linear circuit. Then connect properly the inputs of this linear circuit. The output of this linear circuit depends only on the outputs of $\mathcal{U}_1, ..., \mathcal{U}_{k-1}$ (and in the inputs of the FF-GPAC) that do not change during the process of connection. Moreover, the input of the integrand is not connected, assuming always the value 0 and, hence, the output of $\mathcal{U}_k$ will remain 0.

Next connect the linear circuit associated to the integrand in the following way: first plug the output to the integrator. Next plug the inputs of this linear circuit. This could be problematic. In fact, since one of the inputs of the linear circuit is possibly the output of $\mathcal{U}_k$, when we rise the input of this linear circuit in order to "catch" the output of $\mathcal{U}_k$, this can change the output of $\mathcal{U}_k$, and the circuit may never initialize. A possible situation where this problem may occur is sketched in figure 2.6.3. But the variable of integration of $\mathcal{U}_k$ depends only on the outputs of the integrators $\mathcal{U}_1, ..., \mathcal{U}_{k-1}$ (and in the inputs of the FF-GPAC). Hence, even if the output of the linear circuit associated to the integrand input changes, the variable of integration does not change its value and the output of the integrator will remain constant.

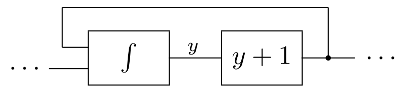


Figure 2.6.3: A circuit with a possibly problematic initialization procedure.

Hence, all integrators and linear circuits of the circuit can be properly initialized. ■

In consequence of this result we will always admit, from now on, that every FF-GPAC has an initialization procedure. Because the same circuit can have different initialization procedures, we now consider that a FF-GPAC consists of a circuit and an initialization procedure, although we will only indicate the initialization procedure when needed.

We think that it is useful at this moment to make some comments, trying to relate in some way our work with other abstract computers, namely with the Turing machine.

A Turing machine operates accordingly to some set of instructions, computing with the initial data present on its tape. All values and operations performed must be built-in or externally supplied. If we had the "hardware" of a Turing machine (including the tape and the head) and some device capable of handling with a set of instructions, we would only need to specify the program of the machine and its initial data to simulate the computation of any Turing machine. This is very similar to what actual computers do and is a major advance towards analog computers (and, in particular, to the differential analyzer) where, in general, every time we want to compute a new program, we have to re-configure the hardware.

If we had some device capable of handling with the type of program indicated above and if it had access to an unlimited number of units, it could simulate any FF-GPAC. Hence, this device would be comparable to an universal Turing machine. Moreover, this device operates in a simple way. It simply compares the values of inputs and outputs of some units and acts in each input in order to "catch" the respective output. This seems realistic, because it is relatively easy to build devices that compares values. For example, we may easily make devices that can compare two values of electric voltage.

The major drawback of this device is the ability of simulating the various units. Is it physically possible to simulate an arbitrary number of integrators, adders, etc.? Possibly not. But actual computers also don't simulate Turing machines, but finite versions of it. Perhaps we could do the same for this device.

The initialization procedure indicated in this section may also be used to initialize "effective" FF-GPACs using only constant units associated to values in $A \subsetneqq \mathbb{R}$. This will be important for the next chapter.

Chapter 3

Recursion Theory Over the Reals

3.1 Introduction to Recursive Functions

In standard computation, recursion theory is a fundamental approach. It allows the description of computable functions in an elegant way and introduces function and set stratification into hierarchies. In this chapter we introduce the reader to standard recursion function theory. Then we will continue with the recent topic of recursive functions over the reals, indicating the more significative work already done and proposing new classes and results. And as one of the goals of this thesis, we will also establish links with the FF-GPAC model.

In recursion theory one usually considers *recursive functions* over some set A. These functions are obtained in the following way. We begin by taking some set B of functions such that, if $f \in B$, then f is a function from A^n to A, for some $n \in \mathbb{N}$. The functions of B are known as *basic functions.* Let

$$A_* = \bigcup_{n\in\mathbb{N}} \{f : f \text{ is a function from } A^n \text{ to } A\}$$

We also take some set OP of *operators* (also called *operations*), where each element $O \in OP$ is a mapping from A_*^n to A_*, for some $n \in \mathbb{N}$.[1] Then we consider the smallest set of functions containing B that is closed under the operations of OP.[2]

Definition 3.1.1 *Let χ be a set of functions and OP a collection of operators. Then $[\chi; OP]$ denotes the smallest set of functions containing χ and closed under the operations of OP. The set $[\chi; OP]$ is called a* function algebra.

[1]Generally speaking, an operator is a mapping from functions to functions.

[2]I.e., the set consists of all functions that can be generated from B by applying a finite number of times the operations of OP.

The approach taken here follows [Clo99]. We also use the following notation. In the conditions described above, let $f :$ and op be a function and an operator, respectively. Then $[f, B; OP]$ and $[B; OP, op]$ denote $[\{f\} \cup B; OP]$ and $[B; OP \cup \{op\}]$, respectively. Also, if $\mathcal{J} = [B; OP]$, then $\mathcal{J} + f$ denotes the function algebra $[f, B; OP]$.

Definition 3.1.2 *Let $\mathcal{J}$ be a function algebra. Then g is a $\mathcal{J}$-recursive function iff $g \in \mathcal{J}$.*

Definition 3.1.3 *Let $\mathcal{J}$ be a function algebra. Then x is a $\mathcal{J}$-recursive number iff it is a 0-ary $\mathcal{J}$-recursive function.*

It is important to make the distinction between the semantic representation and the syntactic representation of recursive functions. In the first case, we consider the functions in the algebra, and in the second case we consider the descriptions of the functions. More specifically, suppose that we are considering a function algebra $[B; OP]$. If the symbols f_i (called constant symbols), where $i \in I$, denote the elements of B and the symbols O_j (called operation symbols), where $j \in J$, denote the elements of OP, we can put into correspondence each function to a syntactical term. Syntactical terms are obtained in the following inductive manner: constant symbols are terms; $O(t_1, ..., t_n)$ is a term if O is an n-ary operation symbol and $t_1, ..., t_n$ are terms. A description of a function f in the algebra is a term that denotes f. Note that f can have distinct descriptions.

We will not be worried with this problem. The context will usually be sufficient to decide which representation is being used.

In classical recursion theory we take A as $\mathbb{N}$. The usual basic functions are

1. The 0-ary function, $Z : \to \mathbb{N}$, defined by $Z = \lambda.0$;
2. The successor function, $S : \mathbb{N} \to \mathbb{N}$, defined by $S = \lambda x.x + 1$;
3. The projections. For each $n, i \in \mathbb{N}$, where $1 \leq i \leq n$, $U_i^n : \mathbb{N}^n \to \mathbb{N}$ is called projection and is defined by $U_i^n = \lambda x_1 ... x_n.x_i$.

Note. In the following we denote Z simply by 0. We will also take

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

Here $0, S$, and U_i^n denote both the descriptions and the functions associated to them.

The usual operations are[3]

1. (C) 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

[3]The strings indicated in parenthesis denote the respective operation symbol. $\mathbf{x}$ represents a vector of variables $\mathbf{x} = (x_1, ..., x_n)$.

to these functions by that order yields the n-ary function h given by $h(\mathbf{x}) = g(f_1(\mathbf{x}), ..., f_p(\mathbf{x}))$;[4]

2. (REC) Primitive recursion: Suppose that f and g are functions of arity n and $n+2$, respectively. Then the primitive recursion operator applied to f and g (in that order) yields the $n+1$-ary function h defined by $h(\mathbf{x}, 0) = f(\mathbf{x})$ and $h(\mathbf{x}, y+1) = g(\mathbf{x}, y, h(\mathbf{x}, y))$;

3. $(\hat{\mu})$ μ-recursion: Suppose that f is an $n+1$-ary function satisfying the following condition: $(\forall \mathbf{x})(\exists y)(f(\mathbf{x}, y) = 0)$. Then the μ-recursion operator applied to $f(\mathbf{x}, y)$ yields the n-ary function given by[5]

$$\hat{\mu} y f(\mathbf{x}, y) = \inf\{y : f(\mathbf{x}, y) = 0\}.$$

4. (μ) Minimalization or Zero-finding: Suppose that f is an $n+1$-ary function. Then the minimalization operator applied to f yields the n-ary function defined on the following way: for each $\mathbf{x} \in \mathbb{N}^n$, $\mu y f(\mathbf{x}, y)$ is the smallest y such that $f(\mathbf{x}, y) = 0$, provided that $f(\mathbf{x}, z)$ is defined for all $z \leq y$. If no such y exists, then $\mu y f(\mathbf{x}, y)$ is let undefined.

Note that $\mu y f(\mathbf{x}, y)$ can be undefined. In order to deal in a more suitable way with this kind of functions, we can proceed as follows (cf. [BM77]). Let f be a function from A to B that can be undefined for some arguments (we say that f is a *partial function*). Then we may convert f to a new function $\bar{f}$ from A to $B \cup \{\perp\}$, where $\perp \notin A, B$ is a new symbol, in the following way

$$\bar{f} = \lambda x. \begin{cases} f(x) & \text{if } f(x) \text{ is defined} \\ \perp & \text{if } f(x) \text{ is not defined.} \end{cases}$$

We will usually use f to denote $\bar{f}$. In order to avoid awkward formulations we will consider that if $\perp$ is one argument of f, then the respective value of f will be $\perp$. Moreover, $dom(f) = \{x \in A : f(x) \neq \perp\}$. We also say that a function $f : A \to B$ is *total* if $dom(f) = A$.

Provided with these basic functions and operations, we can define the following classes:

- The *primitive recursive* functions $\mathcal{PR} = [0, S, U; C, REC]$;
- The *recursive* functions $\mathcal{R}_0 = [0, S, U; C, REC, \hat{\mu}]$;

[4]More precisely, we have defined a m-ary operation of composition, for each $m = 2, 3, ...$. Hence, the composition operator consists, in reality, of a class $\{C_m : m = 2, ...\}$, where C_m is the m-ary composition operator. However, we simply refer to this class as "the composition operator C". In fact, the arity of each operator is implicitly given by the number of arguments and need not to be, for our purposes, explicitly stated. We will use this notation for other operators, whenever possible.

[5]Although we use the variable y as an affix of $\hat{\mu}$, the μ-recursion operator applies only on the last variable, and not to an arbitrary variable. However, this suggestive notation is widely used and we will keep it (the same applies for future versions of this operator). Note also that the operator $\hat{\mu}$ is not computable in the traditional sense.

- The *partial recursive* functions $\mathcal{R} = [0, S, U; C, REC, \mu]$.

Note that if the basic functions have some property and the operators preserve it, then every function in the function algebra will also have this property. For example, we can conclude in this manner that all primitive recursive functions are total. We can also conclude the same result for recursive functions. However, it can be shown [Cut80] that there are recursive functions that are not primitive recursive (e.g. the Ackermann function). It can also be shown [Odi89, p. 129] that the recursive functions are exactly the partial recursive functions which happen to be total.

3.2 Recursive Functions Over $\mathbb{R}$

In classical recursion theory only recursive functions over the set $\mathbb{N}$ are considered. Although we obtain in this manner a rich and elegant theory, with strong interconnections with other models, there are no obvious limitations by which recursive functions should only be considered over the set $\mathbb{N}$. Of course, if we have some set A, one can always define some functions and operators in order to get a function algebra. Nevertheless, we would like to obtain interesting models. For example, it is known that partial recursive functions correspond to the class of functions that can be computed by a Turing machine (cf. [Cut80]).

Next, we introduce the reader to the recursion theory over the reals presented in [Moo96]. The presentation is adapted from [Cam02].

Definition 3.2.1 *Let c be a real number and n be a natural number. Then c_n denotes the n-ary function given, for each $\mathbf{x} \in \mathbb{R}^n$, by $c_n(\mathbf{x}) = c$. We also use c to denote c_0.*[6]

We will use the following operations:

- ($\int$) Integration: Suppose that $f_1, ..., f_m$ are n-ary functions, and $g_1, ..., g_m$ are $n+m+1$-ary functions. The integration operator applied to $f_1, ..., f_m$, $g_1, ..., g_m$, by that order, yields the $n+1$-ary function h defined for each $(\mathbf{x}, y) \in \mathbb{R}^{n+1}$ as follows. Let I be the largest interval in which a unique unary continuous function $\mathbf{s}$ satisfying

$$\begin{aligned} \mathbf{s}(0) &= \mathbf{f}(\mathbf{x}) \\ \partial_z \mathbf{s}(z) &= \mathbf{g}(\mathbf{x}, z, \mathbf{s}(z)), \quad \forall z \in I - S, \end{aligned}$$

  exists, where $S \subseteq I$ is a countable set of isolated points. Then, if $y \in I$, $h(\mathbf{x}, y) = s_1(y)$.[7] Otherwise $h(\mathbf{x}, y)$ is let undefined.

[6] The context will usually be sufficient to decide whether c denotes a constant or an 0-ary function.

[7] s_1 is the first component of $\mathbf{s}$.

- ($\bar{\mu}$) Minimalization or Zero-finding (on the reals): Suppose that f is an $n+1$-ary function. Let $y^- = \lambda\mathbf{x}.\sup\{y \in \mathbb{R}_0^- : f(\mathbf{x}, y) = 0\}$, $y^+ = \lambda\mathbf{x}.\inf\{y \in \mathbb{R}_0^+ : f(\mathbf{x}, y) = 0\}$, and

$$k = \lambda\mathbf{x}.\begin{cases} y^-(\mathbf{x}) & \text{if } -y^-(\mathbf{x}) \leq y^+(\mathbf{x}) \text{ or } y^+(\mathbf{x}) \text{ is undefined} \\ y^+(\mathbf{x}) & \text{if } -y^-(\mathbf{x}) > y^+(\mathbf{x}) \text{ or } y^-(\mathbf{x}) \text{ is undefined} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The minimalization operator applied on f yields a n-ary function defined by $\bar{\mu}yf(\mathbf{x}, y) = k(\mathbf{x})$ if $k(\mathbf{x})$ is defined and $f(\mathbf{x}, y)$ is defined for $y \in [-|k(\mathbf{x})|, |k(\mathbf{x})|]$. Else, $\bar{\mu}yf(\mathbf{x}, y)$ is let undefined.

Note that in order to match the integration operator of [Moo96], we allow the derivatives of functions to have a countable number of isolated singularities (we denote the set of these singularities by S).[8]

We only consider scalar functions, in opposition to [Moo96], where vectorial functions are allowed. We may consider that some vectorial function is recursive iff all its scalar components are recursive.

The operation of composition and the projection functions can also be obtained similarly to the case of recursive functions over $\mathbb{N}$. We will also use the 0-ary functions $0, 1$.

Definition 3.2.2 *The* $\mathbb{R}$*-recursive functions are* $[0, 1, U; C, \int, \bar{\mu}]$.

It is clear that this definition pretends to match the partial recursive functions over $\mathbb{N}$, where primitive recursion corresponds to integration and minimalization (over $\mathbb{N}$) corresponds to minimalization (over $\mathbb{R}$). Next, we show that the most usual real functions are $\mathbb{R}$-recursive (this result was first presented in [Moo96]).

Theorem 3.2.3 *The functions* -1, 0_1, 1_1, -1_1, $f_+ = \lambda xy.x+y$, $f_\times = \lambda xy.xy$, $\lambda x.1/x$, $f_\div = \lambda xy.x/y$, $\lambda x.e^x$, ln, sin, cos, tan *and* arctan *are all* $\mathbb{R}$*-recursive.*

Proof. Let f be the function associated to the description $\int(0, U_2^2)$. Then $f(0) = 0$ and $\partial_x f(x) = f(x)$. We can then conclude that $f = 0_1$.

Similarly, 1_1 can be defined by integration on the following way: $1_1(0) = 1$, $\partial_x 1_1(x) = 0(U_2^2(x, 1_1(x)))$.

Addition is defined by

$$f_+(x, 0) = x,$$
$$\partial_y f_+(x, y) = 1(U_3^3(x, y, f_+(x, y))).$$

We have $-1 = \bar{\mu}x(x+1)$. -1_1 is defined similarly to 1_1. Multiplication is defined by

$$f_\times(x, 0) = 0,$$
$$\partial_y f_\times(x, y) = U_1^3(x, y, f_\times(x, y)).$$

[8]By other words, S has no accumulation points.

$\lambda x.1/x$ is defined, for $x \in (-\infty, 0)$ (we can adapt this procedure for the case where $x \in (0, \infty)$), by $1/x = g(x-1)$ where

$$g(0) = 1,$$
$$\partial_x g(x) = -g^2(x).$$

$f_\div$, $\lambda x.e^x$, tan, arctan, cos and sin are defined, respectively, by

$$\begin{aligned} f_\div(x,y) &= f_\times(1/y, x), \\ e^0 &= 1, \quad \partial_x e^x = e^x, \\ \tan 0 &= 0, \quad \partial_x \tan x = \tan^2 x + 1, \\ \arctan 0 &= 0, \quad \partial_x \arctan x = 1/(1+x^2), \end{aligned}$$
$$\begin{bmatrix} \sin 0 \\ \cos 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \partial_x \begin{bmatrix} \sin x \\ \cos x \end{bmatrix} = \begin{bmatrix} \cos x \\ -\sin x \end{bmatrix}.$$

Switching the rows of the last equation, we easily see that cos is $\mathbb{R}$-recursive. Finally, we can define ln as $\ln x = h(x-1)$, where $h(0) = 0$ and $\partial_x h(x) = 1/(1+x)$. ∎

A problem that still remains until now is the problem of projections. Are they necessary when defining recursive functions over the reals? It is not difficult to show that, with the operators introduced, if the basic functions have arities at most n, then all the functions in the corresponding function algebra will also have arity at most n.[9] However, we would be interested in having functions of arbitrary arity. To achieve this purpose, we must include a succession of basic functions $f_1, f_2, ...$, with unlimited arities. But we would like to choose a set A of basic functions such that the function algebra $[-1, 0, 1, A; C, \int]$ would be the "smallest" with this property. Note that we have removed the operator $\bar{\mu}$ in order to obtain an analog to primitive recursive functions. But we must therefore include -1 as a basic function, because it was obtained using the operator $\bar{\mu}$.

We now show that $A = U$ satisfies the condition indicated above.

Theorem 3.2.4 *Let A be a set of real functions such that $0_1 \in A$ and for each $n \in \mathbb{N}$, there is an m-ary function $f \in A$, with $m > n$, such that f is defined on all $\mathbb{R}^m$.*[10] *Then $U \subseteq [-1, 0, 1, A; C, \int]$.*

Proof. We will first show that, for each $n \in \mathbb{N}$, $0_n \in [-1, 0, 1, A; C, \int]$. We use induction on n. For $n = 0, 1$ the result is obviously true. So, let $n > 1$. Then there exists a function $f_n \in A$, of arity $u_n \geq n+1$, defined on all $\mathbb{R}^{u_n}$. Let $h_1, ..., h_{u_n - n}$ be defined by

$$\mathbf{h}(\mathbf{x}, 0) = \mathbf{0}_{n-1}(\mathbf{x}),$$
$$\partial_y \mathbf{h}(\mathbf{x}, y) = \mathbf{0}_1(f_{u_n}(\mathbf{x}, y, \mathbf{h}(\mathbf{x}, y))).$$

[9] This shows, for instance, that proposition 1 of [Moo96] is incorrect: we cannot obtain U using only the basic functions 0 and 1.

[10] In reality, we only need f to be defined on $\mathbb{R}^{m-1} \times \{0\}$.

Then $0_n = h_1$ can be obtained by integration. We can also obtain the function 1_1 by integration

$$\begin{aligned} 1_1(0) &= 1, \\ \partial_x 1_1(x) &= 0_2(x, 1_1(x)). \end{aligned}$$

We now show that $U_k^n \in [-1, 0, 1, A; C, \int]$ for all $k, n \in \mathbb{N}$ such that $1 \leq k \leq n$. We use again induction on n. U_1^1 can be obtained by integration: $U_1^1(0) = 0$, $\partial_x U_1^1(x) = 1_1(0_2(x, U_1^1(x)))$.

Now let $n > 1$. We want to show that $U_k^n \in [-1, 0, 1, A; C, \int]$, for $1 \leq k \leq n$. If $k < n$, then

$$\begin{aligned} U_k^n(\mathbf{x}, 0) &= U_k^{n-1}(\mathbf{x}), \\ \partial_y U_k^n(\mathbf{x}, y) &= 0_{n+1}(\mathbf{x}, y, U_k^n(\mathbf{x}, y)). \end{aligned}$$

Finally, let $k = n$. Then

$$\begin{aligned} U_n^n(\mathbf{x}, 0) &= 0_{n-1}(\mathbf{x}), \\ \partial_y U_n^n(\mathbf{x}, y) &= 1_1(0_{n+1}(\mathbf{x}, y, U_n^n(\mathbf{x}, y))). \end{aligned}$$

We conclude that $U \subseteq A$. ■

So, if we want to obtain the smaller function algebra $[-1, 0, 1, A; C, \int]$ that contains functions of unlimited arity in the sense indicated in the previous theorem, we have to pick $A = U$. Hence, it is natural to use the class $[-1, 0, 1, U; C, \int]$ as an analog to the primitive recursive functions. We can also obtain similar notions to standard recursion theory, like oracles, by including some other functions to the set U.

Some comments about recursive functions over the reals are in order. As indicated in [Moo96] (and also in [CM01]), the definition of $\mathbb{R}$-recursive functions presented in this paper relies implicitly on other basic function $\times$ with the property that $0 \times x = 0$, even if $x = \perp$ and, in particular, if $x = \infty$.[11] The justification given is that we can take

$$x \times y = \int_0^y x dy. \tag{3.1}$$

However, we believe that this is not a convincing argument. In fact, in order to have $0 \times x = 0$, we must have

$$\int_0^0 x dy = 0,$$

even if x is ∞. But in this last case the Lebesgue integral is not defined and does not make any sense, unless we take functions not over $\mathbb{R}$, but over the extended

[11]Note that $x = \infty$ can be interpreted in two different contexts. If we are only considering constants (and functions) over $\mathbb{R}$, then x is undefined. If we consider the extended real line $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$, then x is defined and has value ∞.

real line $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$ [Str99, pp. 41,42].[12] But this is not the case that we are considering and, therefore, we cannot use the argument presented in formula (3.1) to justify the condition that $0 \times x = 0$ even if $x = \infty$.

We must be more careful when dealing with the kind of situation referred above. As we have indicated, when we introduced the symbol $\perp$ (p. 43), we only introduced it by reasons of simplicity. But we had to impose the condition that $\perp$ does not belong to the domain of any of the functions under consideration. This is not what happens with $\times$. Hence, in the case of $\times$, $\perp$ must be some specific symbol and cannot be understood as "undefined", at least in the sense that we have been considering. We can, of course, extend a partial function $f : A \to B$ to a total one $\bar{f} : A \to B \cup \{\perp\}$ by taking, for each $x \in A$,

$$\bar{f}(x) = \begin{cases} f(x) & \text{if } f(x) \text{ is defined} \\ \perp & \text{if } f(x) \text{ is not defined.} \end{cases}$$

And if we allow some functions to have $\perp$ as an argument we get a richer class of functions comparatively to the previous case. This is precisely what happens in [Moo96], where we have a "hypercomputational power" that enables us to solve the Halting problem and to obtain the entire Arithmetical Hierarchy (that we will consider in section 3.3). But we have no reason for taking this approach. For example, in the classical case, if a Turing machine computes a partial function $f : \mathbb{N} \to \mathbb{N}$ that is undefined for some $x \in \mathbb{N}$, then the machine simply does not halt the computation when the input is x. We do not expect the machine to have the following behavior: If the computation is "looping forever" then write a symbol $\perp$ in the tape, else write the result of the computation.

From the facts presented above, we believe that we have no reasons to insert the condition $0 \times \perp = 0$. We will also assume that, if the argument of a function is $\perp$, then the function itself returns $\perp$. Of course, we will loose in this manner the "hypercomputational power" of the previous model. However, this new model seems much more realistic. Another point that seems unnatural is the use of a set S of singularities when defining the operator of integration. Hence, we will use the following operator (introduced in [Cam02]), where $S = \varnothing$.

- (I) Proper Integration: Suppose that $f_1, ..., f_m$ are n-ary functions, and $g_1, ..., g_m$ are $n + m + 1$-ary functions. The proper integration operator applied to $f_1, ..., f_m, g_1, ..., g_m$, by that order, yields the $n+1$-ary function h defined for each $(\mathbf{x}, y) \in \mathbb{R}^{n+1}$ as follows. Let J be the largest interval in which a unique unary function $\mathbf{s}$ satisfying

$$\begin{aligned} \mathbf{s}(0) &= \mathbf{f}(\mathbf{x}) \\ \partial_z \mathbf{s}(z) &= \mathbf{g}(\mathbf{x}, z, \mathbf{s}(z)), \quad \forall z \in J, \end{aligned}$$

[12] We could also seek some analogy with, e.g. the Dirac delta distribution δ that satisfies $\int_0^0 \delta(x)dx = 1$. However, we must stress that the integral on left-hand side is used only as a mnemotechnic rule that facilitate calculations involving the Dirac delta function. Even the name "delta function" is a misnomer. The delta function is not a function, but a distribution, and one should not talk lightly about its value at x [SW97, p. 9]. In rigor, we could say that δ is a functional [SW97, pp. 5-10].

exists. Then, if $y \in J$, $h(\mathbf{x}, y) = s_1(y)$. Otherwise $h(\mathbf{x}, y)$ is let undefined.

This operator preserves smoothness, i.e., if $f_1, ..., f_m, g_1, ..., g_m$ are of class C^k, for $1 \leq k \leq \infty$, then if we apply the proper integration operator to these functions, the resulting function (if it exists) belongs to the same class [Lan93, theorem 14.5.2].

Definition 3.2.5 $\mathcal{D} = [-1, 0, 1, U; C, I]$.

Note that all the theorems introduced in this section are still true for $\mathcal{D}$. Moreover, all integers are $\mathcal{D}$-recursive numbers (they can be obtained from $-1, 0$, and 1, by addition). The same happens for every rational and for π, e ($\pi = 4\arctan(1)$ and $e = e^1$). In fact, using proper integration, take $k \in \mathbb{N}\backslash\{0\}$ and let $\mathbf{h} = (h_1, ..., h_k)$ be defined by

$$\mathbf{h}(0) = \mathbf{0}, \qquad \partial_x h_k(x) = 1, \ \partial_x h_{k-i}(x) = h_{k-i+1}(x),$$

for $i = 1, ..., k-1$. Then $h = h_1 = \lambda x. x^k/k!$. Hence, h is $\mathcal{D}$-recursive and $1/k! = h(1)$. In this manner we can generate all rationals of the form $1/k!$, for $k \geq 1$. Multiplying these values by integers, we can obtain any rational.[13]

Notice that we have defined the initial condition at the point 0 in the definition of proper integration. However, we can change this point to be a $\mathcal{D}$-recursive number. In fact, if y_0 is a $\mathcal{D}$-recursive number and z is the first component of $\mathbf{z}$, where

$$\begin{aligned} \mathbf{z}(\mathbf{x}, y_0) &= \mathbf{f}(\mathbf{x}), \\ \partial_y \mathbf{z}(\mathbf{x}, y) &= \mathbf{g}(\mathbf{x}, y, \mathbf{z}(\mathbf{x}, y)), \end{aligned}$$

we only have to take $\mathbf{h}(\mathbf{x}, 0) = \mathbf{f}(\mathbf{x})$, $\partial_y \mathbf{h}(\mathbf{x}, y) = \mathbf{g}(\mathbf{x}, y + y_0, \mathbf{h}(\mathbf{x}, y))$ and set $z(\mathbf{x}, y) = h_1(\mathbf{x}, y - y_0)$. Hence, z is $\mathcal{D}$-recursive.

3.3 μ-Recursion and Hierarchies

In this section we establish further connections between classical recursion theory and recursion theory over the reals, along the lines of [Moo96] and [Cam02]. Our first task is to present a class of recursive functions over the reals that have some analogy with the primitive recursive functions.[14] We will use the following convention: a class A of real functions contains a function $f : \mathbb{N}^k \to \mathbb{N}$, for some $k \in \mathbb{N}$, if f admits an extension $\tilde{f} \in A$. In [Cam02], it was shown that, for $k \in \mathbb{N}$, there exists a class constituted only by C^k functions that contains all the primitive recursive functions (proposition 3.4.4). In particular, the following function was used

$$\theta_k = \lambda t. \begin{cases} 0 & \text{if } t \leq 0, \\ t^k & \text{if } t > 0, \end{cases}$$

[13]This construction was first suggested by Cristopher Moore.

[14]A result of this kind was first presented in [CC97].

where $k \in \mathbb{N}$.[15] As indicated in [CMC00], this function may be seen as a C^{k-1} function that tests whether $x \geq 0$. Campagnolo showed that $\mathcal{D} + \theta_k$ contained the primitive recursive functions in the sense indicated above, for each $k \in \mathbb{N}$. However, we would like to obtain a “smaller” function algebra that contains the primitive recursive functions. Motivated by the fact that $C^0 \underset{\neq}{\supseteq} C^1 \underset{\neq}{\supseteq} ... \underset{\neq}{\supseteq} C^\infty$, we now present a similar result, but for C^∞ functions. We also present several results for classes that take minimalization operators.

Consider the C^∞ function θ_∞ defined by

$$\theta_\infty = \lambda t. \begin{cases} 0 & \text{if } t \leq 0, \\ \exp(-1/t) & \text{if } t > 0. \end{cases}$$

Definition 3.3.1 $\mathcal{G} = [-1, 0, 1, \theta_\infty, U; C, I]$.[16]

In the following, we will show that $\mathcal{G}$ contains all primitive recursive functions. We will also pay more attention to the minimalization operator. We begin by presenting an alternative version of this operator.[17]

- (μ) unrestricted μ-recursion: Suppose that f is a $n+1$-ary function. Let

  $$k = \lambda \mathbf{x}. \inf\{y \in \mathbb{R}_0^+ : f(\mathbf{x}, y) = 0\}.$$

  Then the μ operator applied to f is defined, for each $\mathbf{x}$, by $\mu y f(\mathbf{x}, y) = k(\mathbf{x})$, if $k(\mathbf{x})$ is defined and if $f(\mathbf{x}, y)$ is defined for $y \in [0, k(\mathbf{x})]$. Else $\mu y f(\mathbf{x}, y)$ is let undefined for $\mathbf{x}$.

We take the following classes

Definition 3.3.2 $\mathcal{M} = [-1, 0, 1, \theta_\infty, U; C, I, \mu]$.

Definition 3.3.3 $\bar{\mathcal{M}} = [-1, 0, 1, \theta_\infty, U; C, I, \bar{\mu}]$.

We also present a similar version of μ-recursion (remember the definition for standard recursion theory) for the real case.

- ($\hat{\mu}$) μ-recursion: Suppose that f is a total $n+1$-ary function such that, for each $\mathbf{x} \in \mathbb{R}^n$, $(\exists t \in \mathbb{R}_0^+)(f(\mathbf{x}, t) = 0)$. Then the μ-recursion operator applied to f is defined, for each $\mathbf{x} \in \mathbb{R}^n$, by

  $$\mu y f(\mathbf{x}, y) = \lambda \mathbf{x}. \inf\{y \in \mathbb{R}_0^+ : f(\mathbf{x}, y) = 0\}.$$

Definition 3.3.4 $\mathcal{M}_0 = [-1, 0, 1, \theta_\infty, U; C, I, \hat{\mu}]$.

[15]Notice that for $k \geq 1$, θ_k is of class C^{k-1}, but not of class C^k.

[16]In [Cam02] $\mathcal{G}$ was used to denote the class $\mathcal{D} + \theta_k$, for some fixed $k \in \mathbb{N}$. We took the same notation to designate $\mathcal{D} + \theta_\infty$, because this last function algebra has similar properties to $\mathcal{D} + \theta_k$.

[17]Nevertheless, some alternatives to μ-recursion have been proposed for recursive functions over the reals. For instance, in [Myc], μ-recursion is substituted by limit operators.

The $\hat{\mu}$ operator differs from the previous one by the condition that it can only be applied to a total $n+1$-ary function satisfying the condition that, for each $\mathbf{x} \in \mathbb{R}^n$, $(\exists t \in \mathbb{R}_0^+)(f(\mathbf{x}, t) = 0)$. Hence $\mathcal{M}_0 \subseteq \mathcal{M}$.

In the following, we show that $\mathcal{M} = \bar{\mathcal{M}}$. In order to accomplish this task, we need the following functions (cf. [Moo96]): the Heaviside step function Θ and the Kronecker δ-function δ. They are defined, for each $x \in \mathbb{R}$, by

$$\Theta(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0, \end{cases} \qquad \delta(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x \neq 0. \end{cases}$$

It was already shown that $\Theta, \delta \in \bar{\mathcal{M}}$ (see [Moo96]. Although the result is shown for the operation $\int$ instead of I, the adaptation is straightforward).

Lemma 3.3.5 $\mathcal{M} \subseteq \bar{\mathcal{M}}$.

Proof. We show the result by structural induction on the terms of $\mathcal{M}$. Concretely, we show that for each description A in $\mathcal{M}$, there exists a description B in $\bar{\mathcal{M}}$ such that A and B are associated to the same function (and have the same arity). For reasons of convenience, we will only say that A and B are associated in the remaining of this proof.

The basis step is immediate. For the inductive step, first take the symbol operator C. Suppose that we have a term $C(g, f_1, ..., f_p)$ in $\mathcal{M}$. Then, by induction hypothesis, $g, f_1, ..., f_p$ are associated to descriptions $\bar{g}, \bar{f}_1, ..., \bar{f}_p$ in $\mathcal{M}$, respectively. It is not difficult to see that $C(g, f_1, ..., f_p)$ is associated to $C(\bar{g}, \bar{f}_1, ..., \bar{f}_p)$. A similar argument holds for the symbol operator I. Finally, take the μ symbol operator. Suppose that $\mu y f$ is a description in $\mathcal{M}$. Then, by induction hypothesis, the $n+1$-ary description f is associated to some description $\bar{f}$ in $\bar{\mathcal{M}}$. Let A be a description in $\bar{\mathcal{M}}$ associated to the function $\lambda y.\Theta(y)y \in \bar{\mathcal{M}}$. Then $\mu y f$ is associated to the description $\bar{\mu} y(C(\bar{f}, U_1^{n+1}, ..., U_n^{n+1}, C(A, U_{n+1}^{n+1})))$. ■

Lemma 3.3.6 $\delta, \lambda x.\,|x|\,, \Theta \in \mathcal{M}$.

Proof. For the case of functions δ and $\lambda x.\,|x|\,$, we use an adaptation of the proof of proposition 4 in [Moo96]. Take

$$\delta = \lambda x.1 - \mu y((x^2 + y^2)(y - 1)).$$

We also have $\lambda x.\,|x| = \lambda x.\mu y(x^2 - y^2)$ and $\Theta = \lambda x.\delta(|x| - x)$.[18] ■

Theorem 3.3.7 $\mathcal{M} = \bar{\mathcal{M}}$.

Proof. By lemma 3.3.5, we only have to prove that $\bar{\mathcal{M}} \subseteq \mathcal{M}$. This proof is similar to the proof of lemma 3.3.5. We only sketch the case for the symbol operator $\bar{\mu}$ and we will not be much worried about notation. Let $\bar{\mu} y \bar{f} \in \bar{\mathcal{M}}$. Then, by induction hypothesis, the $n+1$-ary description $\bar{f}$ is associated to

[18]The expression for Θ was obtained by the student Luís Russo at IST, Lisbon.

some description f in $\mathcal{M}$. Let $P = \mu y(f(\mathbf{x}, y) f(\mathbf{x}, -y))$. Then, we conclude that $\bar{\mu} y \bar{f}(\mathbf{x}, y)$ is associated to

$$-P\delta(f(\mathbf{x}, -P)) + P\delta(f(\mathbf{x}, P)) - \delta(f(\mathbf{x}, P))\delta(f(\mathbf{x}, -P))P.$$

Hence, our result is shown. ■

Noting the way how we defined the classes $\mathcal{G}$, $\mathcal{M}_0$, and $\mathcal{M}$, it is natural to correspond them with the classes $\mathcal{PR}$, $\mathcal{R}_0$ and $\mathcal{R}$, respectively. We will now show relations between these classes. We follow closely the approach taken in [Cam02] in order to prove theorem 3.3.10.

Lemma 3.3.8 *There are total unary functions $\sigma, s \in \mathcal{G}$ with the following properties:*

1. $\sigma(x) = 0$ *if* $x \leq 0$ *and* $\sigma(x) = 1$ *if* $x \geq 1$,
2. $s(x) = j$, *whenever* $x \in [j, j + 1/2]$, *for all* $j \in \mathbb{N}$.

Proof. σ can be defined by $\sigma(0) = 0$ and $\partial_x \sigma(x) = b\theta_\infty(x(1 - x))$, where b is a suitable constant. Because $\theta_\infty(x(1 - x)) = 0$ for $x \leq 0$ or $x \geq 1$, and $\theta_\infty(x(1 - x)) > 0$ for $x \in (0, 1)$, we conclude that σ is equal to 0 for $x \leq 0$, strictly increasing for $x \in (0, 1)$, and constant, with value a, for $x \geq 1$. We only have to pick the right value of b in order to have $a = 1$. Taking

$$c = \int_0^1 \theta_\infty(x(1 - x))dx$$

and $b = 1/c \in \mathcal{G}$, we have the pretended result.

s can be defined by $s(0) = 0$ and $s'(x) = k\theta_\infty(-\sin 2\pi x)$, where k is some suitable constant. For $x \in [0, 1/2]$, we have $s(x) = 0$. On $(1/2, 1)$, s increases strictly. Similarly to the last case, we can take

$$k = \left(\int_0^1 \theta_\infty(-\sin 2\pi x)dx\right)^{-1} \in \mathcal{G}.$$

Hence, $s(1) = 1$. Using the same argument for $x \in [j, j + 1]$, for all $j \in \mathbb{N}$, we conclude that $s(x) = j$, whenever $x \in [j, j + 1/2]$. It is not difficult to see that both functions are total (note also that $s(x) = j$, whenever $x \in [-j, -j + 1/2]$, for $j \in \mathbb{N}$). ■

Notice that, because $\sigma, s \in \mathcal{G}$, we conclude that σ, s are functions of class C^∞.

Given a unary function $f : \mathbb{N} \to \mathbb{N}$, its *iteration* is the binary function h defined by $h(x, y) = f^{[y]}(x)$, where $f^{[0]}(x) = x$ and $f^{[y+1]}(x) = f(f^{[y]}(x))$, for every $x \in dom(f)$. The following result is from Gladstone [Gla71, theorem 3].

Lemma 3.3.9 $\mathcal{PR} = [0, S, U, \lambda xy. |x - y|; C, PI]$, *where PI stands for pure iteration with no parameters, i.e., given f, define $F(x, t) = f^{[t]}(x)$ for all $x, t \in \mathbb{N}$.*

Theorem 3.3.10 *Let $f \in \mathcal{PR}$. Then f has an extension in $\mathcal{G}$.*

Proof. We use the previous lemma to show this result. $\mathcal{G}$ contains $0, S$ and U. Taking $\lambda xy.(x-y)\sigma(x-y)$, where σ is given in lemma 3.3.8, we get a total extension to the reals of the cut-off difference.[19] We can also take $\lambda xy.(x \dot{-} y) + (y \dot{-} x)$ as an extension of $\lambda xy.\,|x-y|$. $\mathcal{G}$ is trivially closed under composition. Hence, we only have to show that if $f : \mathbb{N} \to \mathbb{N}$ has an extension $\bar{f}$ in $\mathcal{G}$, then the binary function $g = \lambda xt.f^{[t]}(x)$ also has an extension in $\mathcal{G}$.

The proof is constructive. We use a pair of variables in order to simulate the iteration function. The first variable is iterated during half of a period unit, while the second remains constant. Then, in the following half unit period the situation switches.

Let $r = \lambda t.s(t+1/4)$, where s is the function defined on lemma 3.3.8. Then $r(t) = j$, as long $t \in [j-1/4, j+1/4]$, for $j \in \mathbb{N}$. Consider the binary functions y_1 and y_2, defined by

$$\begin{aligned} y_1(x,0) &= x \\ y_2(x,0) &= x \\ \partial_t y_1(x,t) &= c(f(r(y_2)) - y_1)^3 \theta_\infty(\sin 2\pi t) \\ \partial_t y_2(x,t) &= c(r(y_1) - y_2)^3 \theta_\infty(-\sin 2\pi t), \end{aligned}$$

where c is some suitable constant. We analyze these functions for $x \in \mathbb{N}$ and we start with $t \in [0, 1/2]$. In this interval, we have $\partial_t y_2 = 0$. Hence, $y_2 = x$ is kept fixed. If $f(x) = x$, we trivially have $y_1(x,t) = y_2(x,t) = x$ for all $t \in \mathbb{R}$ and, hence, we may take y_1 as the extension of g. So, suppose that $f(x) \neq x$. We have

$$\frac{1}{(f(x) - y_1(x,1/2))^2} - \frac{1}{(f(x)-x)^2} = 2c\int_0^{1/2} \theta_\infty(\sin 2\pi t)dt.$$

Because $\int_0^{1/2} \theta_\infty(\sin 2\pi t)dt \simeq 0.1045 > 0.1$ we may prove that, if $c = 80$, for example, then $|f(x) - y_1(x,1/2)| < 1/4$ and consequently, $r(y_1(x,1/2)) = f(x)$. The solution of the equation has a similar behavior for the parameter t for subsequent intervals of the form $[j, j+1/2]$ and $[j+1/2, j+1]$, where $j \in \mathbb{N}$. Hence, for all $t \in \mathbb{N}$ and all x, $f^{[t]}(x) = r(y_1(x,t))$.

Note also that, for $x \in \mathbb{N}$ fixed, $r(y_1(x,t)) = f^{[j]}(x)$, for $t \in [-j-1/2, -j]$ and $j \in \mathbb{N}$, and that $\lambda t.y_1(x,t)$, $\lambda t.y_2(x,t)$ are total. ■

The following result appears naturally.

Theorem 3.3.11 *Let φ be a continuous function with the property that it coincides with a function $g \in \mathcal{D}$, in an interval $(-\infty, a]$, but that $\varphi(x) > g(x)$ on an interval $(a, a+c)$, where $c > 0$ and $a \in \mathbb{R}$. Let $f \in \mathcal{PR}$. Then f has an extension in $\mathcal{D} + \varphi$.*

[19] The cut-off difference is a binary function $\dot{-} : \mathbb{N}^2 \to \mathbb{N}$ given for each $x, y \in \mathbb{N}$ by $x \dot{-} y = \max\{0, x-y\}$.

Proof. The proof follows along the lines of theorem 3.3.10. The only problem is that we do not have the function θ_∞. We can replace this function by a unary function h such that $h(x) = 0$ for $x \in (-\infty, 0]$ (because of function σ in lemma 3.3.8), $h(x) \geq 0$ for $x \in [0, 1]$ (because of the functions sin that appear as arguments of θ_∞), and $h(x) > 0$ for some $x \in (0, 1/4)$ (in order that the integrals appearing in the proof of lemma 3.3.8 and in the proof of the previous theorem have a positive value) Let $b_1 \in (a, a + c)$ be a rational. Take also a rational $b_0 \in (-\infty, a]$ such that $(a - b_0) < 1/3(b_1 - a)$. Hence, $b_0, b_1 \in \mathcal{D}$, and using the function

$$h = \lambda x.\varphi(b_0 + (b_1 - b_0)x) - g(b_0 + (b_1 - b_0)x)$$

instead of θ_∞, we get the desired result. ■

Hence, we conclude that we could use other functions than θ_∞ in order to present similar results. In particular, we could use the functions θ_k referred at the beginning of this section (for $k \geq 1$).

We will need the following function (*characteristic function of* $\mathbb{N}$)

$$\chi_{\mathbb{N}} = \lambda x. \begin{cases} 1 & \text{if } x \in \mathbb{N} \\ 0 & \text{if } x \notin \mathbb{N}. \end{cases}$$

It can be defined by $\chi_{\mathbb{N}} = \lambda x.\Theta(x)\delta(\sin \pi x)$. We also take

$$\chi_{\mathbb{N}^n} = \lambda x_1 ... x_1.\chi_{\mathbb{N}}(x_1)...\chi_{\mathbb{N}}(x_n),$$

for $n \in \mathbb{N}\backslash\{0\}$. Notice that $\Theta, \delta, \chi_{\mathbb{N}^n} \in \mathcal{M}_0$, for $n \geq 1$.

Theorem 3.3.12 *Every function in $\mathcal{R}_0$ has a total extension in $\mathcal{M}_0$.*

Proof. We first show that $\mathcal{PR} \subseteq \mathcal{M}_0$. We proceed as in the proof of theorem 3.3.10. The only problem appear in the iteration simulation, where y_1 or y_2 may not be total. But we can overcome this problem easily by taking

$$\partial_t y_1(x, t) = \chi_{\mathbb{N}}(x)c(f(r(y_2)) - y_1)^3\theta_\infty(\sin 2\pi t)$$
$$\partial_t y_2(x, t) = \chi_{\mathbb{N}}(x)c(r(y_1) - y_2)^3\theta_\infty(-\sin 2\pi t).$$

Hence, $\mathcal{PR} \subseteq \mathcal{M}_0$. Now suppose that a $n + 1$-ary function $f \in \mathcal{R}_0$ has some total extension $\bar{f}$. Then we can find an extension g of $\lambda\mathbf{x}.\hat{\mu}yf(\mathbf{x}, y)$ given by

$$g = \lambda\mathbf{x}.\hat{\mu}y(\chi_{\mathbb{N}^n}(\mathbf{x})((\sin \pi y)^2 + (\bar{f}(\mathbf{x}, y))^2)).$$

We can easily see that $g \in \mathcal{M}_0$ whenever the operator $\hat{\mu}$ can be applied to f. Notice that the term $\sin \pi y$ guarantees us that, for each $\mathbf{x} \in \mathbb{N}^n$, $g(\mathbf{x}) = f(\mathbf{x}) \in \mathbb{N}$. This is because $\sin \pi y = 0$ iff $y \in \mathbb{N}$. ■

Theorem 3.3.13 *Every function in $\mathcal{R}$ has an extension in $\mathcal{M}$.*

Proof. It can be shown (see [Odi89, p. 129]) that if an n-ary function f is in $\mathcal{R}$, then there are primitive recursive functions U, T, such that

$$f = \lambda \mathbf{x}.U(\mu y T(\mathbf{x}, y)).$$

Let $\bar{U}, \bar{T} \in \mathcal{M}$ be total extensions of U, T, respectively (they exist, by theorem 3.3.12). Hence, an extension of f can be given by

$$g = \lambda \mathbf{x}.\bar{U}(\mu y((\sin \pi y)^2 + (\bar{T}(\mathbf{x}, y))^2)).$$

Trivially, $g \in \mathcal{M}$ satisfy our needs. ∎

We now recall [Odi89] that we can associate to an n-ary predicate R a function (the *characteristic function*) defined by

$$c_R = \lambda \mathbf{x}. \begin{cases} 1 & \text{if } R(\mathbf{x}) \text{ holds} \\ 0 & \text{if } R(\mathbf{x}) \text{ does not hold.} \end{cases}$$

We say that R is $\mathcal{J}$-recursive iff c_R is $\mathcal{J}$-recursive, were $\mathcal{J}$ is a function algebra over $\mathbb{N}$ or $\mathbb{R}$. We will usually abuse notation and identify a relation with its characteristic function.

We also recall [Odi89] that the set of recursive relations, $\mathcal{R}_0$, is at the bottom of a countably infinite hierarchy of increasingly uncomputable sets, the *Arithmetical Hierarchy.* A relation in the jth level ($j \in \mathbb{N}$) is obtained by applying j alternating quantifiers, $\exists$ and $\forall$, to a recursive relation. If the outermost quantifier of a relation of the previous type is $\exists$, then the relation is in Σ^0_j; if the outermost quantifier is an $\forall$, then the relation is in Π^0_j. We also set $\Delta^0_j = \Sigma^0_j \cap \Pi^0_j$ and let $\Delta^0_\omega = \cup_{n \in \mathbb{N}} \Delta^0_n$. It can be shown that $\Sigma^0_n \cap \Pi^0_n \subseteq \Delta^0_{n+1}$, i.e., these sets define an hierarchy, and that $\Delta^0_n \subsetneqq \Sigma^0_n$, $\Delta^0_n \subsetneqq \Pi^0_n$, i.e., this hierarchy does not collapse. Notice that Σ^0_0, Π^0_0 and Δ^0_0 are exactly the set of recursive relations.

In [Moo96], it was shown that every relation in Δ^0_ω is $\mathbb{R}$-recursive. However, this strong and unnatural result was obtained using the function $\times$ that satisfies $0 \times \infty = 0$. This function allows the introduction of an operator η that "senses" if a function has a zero in $\mathbb{R}$, yielding a hypercomputational power. With this operator we can simulate recursively the quantifiers $\exists$ and $\forall$, obtaining the entire arithmetical hierarchy. More, because the set of function from $\mathbb{N}$ to $\mathbb{N}$, $\mathbb{N}^{\mathbb{N}}$, has the same cardinality of $\mathbb{R}$, we can even quantify over functions, obtaining a broader hierarchy, known as the *Analytical Hierarchy* (see [Odi89]).

These results are, however, based on the assumption of including the function $\times$, that can handle with ∞, in the definition of $\mathbb{R}$-recursive functions. But for the reasons that we have seen, we didn't introduce this operator in the classes $\mathcal{G}, \mathcal{M}_0$ and $\mathcal{M}$. In this manner, we obtain more realistic, but much less general models. We will, of course, loose much of the results presented in [Moo96]. However, we still can manage to keep some of them. We now pass to this task.

Theorem 3.3.14 *A recursive relation R has an extension $\bar{R}$ to $\mathcal{M}_0$-decidable relations.*[20] *Moreover, this extension is true for the same values of R.*

[20]Notice that a relation seen as a function must be total.

Proof. Let R be an n-ary recursive relation. By theorem 3.3.12, we know that there exists some total extension $\bar{c}_R \in \mathcal{M}_0$ of the characteristic c_R of R. The only problem is that $\bar{c}_R$ may not be a relation, i.e., for non-integers arguments, its value may be different from 0 and 1. But letting

$$c_{\bar{R}} = \lambda \mathbf{x}.\chi_{\mathbb{N}^n}(\mathbf{x})\bar{c}_R(\mathbf{x}),$$

we obtain a characteristic for a relation $\bar{R}$ that satisfies the conditions of the lemma. ■

For recursive relations over the reals, we can define similar classes to those of the arithmetical hierarchy. We take the initial classes, $\bar{\Sigma}_0^0, \bar{\Pi}_0^0$, and $\bar{\Delta}_0^0$, to be the set of $\mathcal{M}_0$-recursive relations. We can define $\bar{\Sigma}_n^0$, $\bar{\Pi}_n^0$, and $\bar{\Delta}_n^0$, for $n \geq 1$, similarly to the case of natural functions, obtaining the *Arithmetical Hierarchy over the reals.*

Theorem 3.3.15 $\bar{\Sigma}_n^0 \cup \bar{\Pi}_n^0 \subseteq \bar{\Delta}_{n+1}^0$, *for all* $n \in \mathbb{N}$.

Proof. We proceed by induction on n. For $n = 0$, observe that if R is a n-ary $\mathcal{M}_0$-recursive relation, then

$$\begin{aligned} R(\mathbf{x}) &= (\exists y)R(U_1^{n+1}(\mathbf{x}, y), ..., U_n^{n+1}(\mathbf{x}, y)) = \\ &= (\forall y)R(U_1^{n+1}(\mathbf{x}, y), ..., U_n^{n+1}(\mathbf{x}, y)). \end{aligned}$$

Hence $R \in \bar{\Sigma}_0^0 \cap \bar{\Pi}_0^0 = \bar{\Delta}_1^0$. Now let $R \in \bar{\Sigma}_{n+1}^0$. We want to show that $R \in \bar{\Delta}_{n+2}^0$. We have that $R(\mathbf{x}) = (\exists y)(S(\mathbf{x}, y))$, for some relation $S \in \bar{\Pi}_n^0$. But, by induction hypothesis, $S \in \bar{\Delta}_{n+1}^0 \subseteq \bar{\Pi}_{n+1}^0$. Hence, $R(\mathbf{x}) \in \bar{\Sigma}_{n+2}^0$. On the other side

$$R(\mathbf{x}) = (\forall y)R(U_1^{n+1}(\mathbf{x}, y), ..., U_n^{n+1}(\mathbf{x}, y)) \in \bar{\Pi}_{n+2}^0.$$

Then, $R(\mathbf{x}) \in \bar{\Delta}_{n+2}^0$. Dually, we also conclude that $\bar{\Pi}_{n+1}^0 \subseteq \bar{\Delta}_{n+2}^0$. Hence, $\bar{\Sigma}_{n+1}^0 \cup \bar{\Pi}_{n+1}^0 \subseteq \bar{\Delta}_{n+2}^0$. ■

We can, therefore, conclude that we have indeed an hierarchy, although we don't know if it collapses. We can also prove the following theorem.

Theorem 3.3.16 *Let* A_n^0 *be a set of the Arithmetical Hierarchy, with* A_n^0 *equal to* Σ_n^0, Π_n^0 *or* Δ_n^0, *for some* $n \in \mathbb{N}$. *Let* R *be a relation in* A_n^0. *Then* R *has an extension* $S \in \bar{A}_n^0$ *that is true exactly for the same values of* R.

Proof. We proceed by induction in the structure of the sequence of quantifiers. The case $n = 0$ is provided by theorem 3.3.14. For the $\exists$ quantifier, notice that if R is a $n+1$-relation in $\mathbb{N}$ having an extension $\bar{R}$ to the reals that is true exactly for the same values of R, then we have that $(\exists y)R(\mathbf{x}, y)$ has a real extension defined by

$$(\exists y)\bar{R}(\mathbf{x}, y).$$

This extension is true for the same values of $(\exists y)(R(\mathbf{x}, y))$. If the quantifier is $\forall$ instead of $\exists$, then take as extension

$$(\forall y)(\chi_{\mathbb{N}^n}(\mathbf{x})(\chi_{\mathbb{N}}(y)R(\mathbf{x}, y) + (1 - \chi_{\mathbb{N}}(y)))).$$

and we have done. ■

3.4 A Subclass of Recursive Functions Over the Reals

We would like to have a computational model for recursive functions over the reals. This model must be analog to deal with real numbers. In order to tackle this question, we have introduced the models presented in the previous chapter. However, some problems may appear. For example, let h be defined on all the real line by

$$\begin{aligned} h(x,0) &= f(x), \\ \partial_y h(x,y) &= g(x,y,h(x,y)), \end{aligned}$$

for some $\mathcal{D}$-recursive functions f and g. Suppose that we had a device that could compute all $\mathcal{D}$-recursive functions. Hence h could be computed. It is natural that this machine would work in the following way: first calculate $h(x,0)$; then calculate $h(x,y)$. Suppose that we would like to compute $r = \lambda x.h(x,x)$ and that the machine described above just gave us the value $r(y)$ for some $y \in \mathbb{R}$. It seems reasonable to admit that if we would like to calculate $r(y+dy)$ from $r(y)$ with the device referred above, we would only need an "infinitesimal time." However, in order to calculate $r(y+dy)$ from $r(y)$, we would need to calculate $h(y+dy,0)$ and then $h(y+dy,y+dy) = r(y+dy)$. This seems to need more than an "infinitesimal time." Hence, we believe that we need to have some extra care with the proper integration and composition operators. So, we introduce a new subclass of $\mathcal{D}$.

Consider the alphabet constituted by the following symbols:

$$\begin{array}{ll} \text{Basic Functions} & \left\{ \begin{array}{l} 0, 1, -1, +, \times \\ U_i^n, \text{ for } 1 \le i \le n \text{ and each } n \in \mathbb{N}. \end{array} \right. \\ \text{Operators} & CM, INT \\ \text{Punctuation} & (,),, \end{array}$$

We will first focus in the definition of descriptions with *active* and *locked variables* (that we will abbreviate by AV and LV, respectively). These concepts will be used to bypass the problems described above for proper integration.

In what follows, we will introduce the FF-GPAC as a model for functions associated to these descriptions and, informally, a variable x_i of a description is active if the value of the corresponding function depends on x_i and if this variable may be freely updated when the function is implemented in a FF-GPAC. A variable x_i is locked if the value of the corresponding function depends on x_i, but the value of x_i must be fixed a priori, when generating the function in a FF-GPAC.[21] Note that, because we have not introduced a set of variables in our alphabet, when we refer to the variable x_i of the function f, we are referring to the ith argument of f.

[21] This is the case for the variable of the function r introduced above.

Definition 3.4.1 *Descriptions with active and locked variables are expressions formed with the following rules:*

1. $-1, 0, 1$ *are* 0*-ary descriptions without AVs or LVs;*

2. U_i^n *is a description of arity* n*, for* $1 \leq i \leq n$ *and each* $n \in \mathbb{N}$*. It has one active variable,* x_i*, and no LVs;*

3. $\times$ *and* $+$ *are descriptions of arity* 2*. They have* x_1 *and* x_2 *as AVs, and no LVs;*

4. *If* $G, F_1, ..., F_m$ *are descriptions, where* $F_1, ..., F_m$ *have arity* n*, and* G *has arity* m*, then* $CM(G, F_1, ..., F_m)$ *is a description of arity* n*. Let* $x_{k_1}, ..., x_{k_s}$ *and* $x_{u_1}, ..., x_{u_r}$ *be the AVs and LVs of* G*, respectively. Moreover, let* A_i, L_i *be the sets containing the AVs and LVs of* F_i*, respectively, for* $i = 1, ..., m$*. Then the sets of LVs and AVs of* $CM(G, F_1, ..., F_m)$ *are given by*

$$L = \left(\bigcup_{i=1}^{s} L_{k_i}\right) \cup \bigcup_{i=1}^{r} (A_{u_i} \cup L_{u_i}) \quad , \quad A = \left(\bigcup_{i=1}^{s} A_{k_i}\right) \backslash L,$$

   respectively;

5. *Let* $F_1, ..., F_m$ *be descriptions of arity* n*, and* $G_1, ..., G_m$ *be descriptions of arity* $n + m + 1$*. Suppose that the set constituted simultaneously by all the AVs and LVs of* $F_1, ..., F_m, G_1, ..., G_m$ *is* S_1*. Take* $S = S_1 \cap \{x_1, ..., x_n\}$*. If* G_i *don't have any LV among the variables* $x_{n+1}, ..., x_{n+m+1}$ *for every* $i = 1, ..., n$ *then* $INT(F_1, ..., F_m, G_1, ..., G_m)$ *is a description of arity* $n+1$*. It has as sets of AVs and LVs,* $\{x_{n+1}\}$ *and* S*, respectively.*

Definition 3.4.2 *The degree of a description* F *(abbreviated by* $\deg F$*) is the total number of occurrences of the symbols* CM *and* INT *in* F*.*

To each n-ary description, we will associate an n-ary function in the following way:

1. To the descriptions $-1, 0, 1$ correspond the 0-ary functions $-1, 0, 1$, respectively;

2. To each U_i^n correspond the projection $f : \mathbb{R}^n \to \mathbb{R}$ defined by $f = \lambda x_1 ... x_n . x_i$;

3. To the description $\times$ corresponds the function $\times : \mathbb{R}^2 \to \mathbb{R}$ defined by $\times = \lambda x_1 x_2 . x_1 x_2$. To the description $+$ corresponds the function $+ : \mathbb{R}^2 \to \mathbb{R}$ defined by $+ = \lambda x_1 x_2 . x_1 + x_2$;

4. If a description is given by $CM(G, F_1, ..., F_m)$ and to $G, F_1, ..., F_m$ are associated the functions $g, f_1, ..., f_m$ (where n is the arity of the f_i's), then we associate to $CM(G, F_1, ..., F_m)$ the n-ary function h defined, for each $\mathbf{x}$, by $h(\mathbf{x}) = g(f_1(\mathbf{x}), ..., f_m(\mathbf{x}))$;

5. Suppose that a description is given by $INT(F_1, ..., F_m, G_1, ..., G_m)$ and that to $F_1, ..., F_m, G_1, ..., G_m$ are associated the functions $f_1, ..., f_m, g_1, ..., g_m$ of arities $n, ..., n, n+m+1, ..., n+m+1$, respectively. Then we associate to $INT(F_1, ..., F_m, G_1, ..., G_m)$ the $n+1$-ary function h defined for each $(\mathbf{x}, y) \in \mathbb{R}^{n+1}$ as follows. Let I be the largest interval in which a unique unary function $\mathbf{s}$ satisfying
$$\mathbf{s}(0) = \mathbf{f}(\mathbf{x})$$
$$\partial_z \mathbf{s}(z) = \mathbf{g}(\mathbf{x}, z, \mathbf{s}(z)), \quad \forall z \in I,$$
exists. Then, if $y \in I$, $h(\mathbf{x}, y) = s_1(y)$. Otherwise $h(\mathbf{x}, y)$ is let undefined.

It is easy to verify (by induction on the degree of the description) that if a description H has as sets of AVs and LVs, A and L, respectively, then $A \cap L = \varnothing$. Moreover, if h is the function associated to H and $x_k \notin A \cup L$, then h does not depend on x_k, i.e.,
$$h(a_1, ..., a_k, ..., a_n) = h(a_1, ..., a_{k-1}, 0, a_{k+1}, ..., a_n),$$
for every $a_k \in \mathbb{R}$ and every $(a_1, ..., a_{k-1}, 0, a_{k+1}, ..., a_n)$ belonging to the domain of h.

Definition 3.4.3 $\mathcal{I} = [-1, 0, 1, U, +, \times; CM, INT]$.

Note that, because we are only restricting the use of integration in $\mathcal{I}$, relatively to $\mathcal{D}$, we have $\mathcal{I} \subseteq \mathcal{D}$. We don't know if this inclusion is proper. Nevertheless, the functions presented in theorem 3.2.3 still are $\mathcal{I}$-recursive. Similarly, every rational, π, and e still are $\mathcal{I}$-recursive numbers.

3.5 Analog Circuits and Recursive Functions Over $\mathbb{R}$

In this section we relate the work done in this chapter with the work developed on the previous one, as a main contribution to recursive function theory over the reals of Moore, Campagnolo, and Costa.

Theorem 3.5.1 *Suppose that a unary function f can be generated by a FF-GPAC $\mathcal{U}$, in some interval I, where all the constant units of $\mathcal{U}$ are associated to $\mathcal{I}$-recursive numbers. Suppose also that the computation starts with $x_0 = a$, where a is a $\mathcal{I}$-recursive number. Then f is $\mathcal{I}$-recursive on I.*

Proof. Suppose that f is generated by a FF-GPAC $\mathcal{U}$, with integrators $\mathcal{U}_2, ..., \mathcal{U}_n$ in an appropriate order, with outputs $y_2, ..., y_n$, respectively. Then, if $y_0 = \lambda x.1$, $y_1 = \lambda x.x$, we have
$$y_k' = \sum_{i=0}^{n} \sum_{j=1}^{k-1} c_{ij}^k y_i y_j', \qquad k = 2, ..., n,$$

for suitable $\mathcal{I}$-recursive constants c_{ij}^k. Consider the functions $g_k \in \mathcal{I}$ of arity $n+k-2$, for $k = 2, ..., n$, defined as

$$g_k = \lambda x_1 ... x_n z_2 ... z_{k-1} . c_{01}^k + \sum_{i=1}^{n} \left(c_{i1}^k x_i \right) + \sum_{j=2}^{k-1} \left(c_{0j}^k z_j \right) + \sum_{i=1}^{n} \sum_{j=2}^{k-1} \left(c_{ij}^k x_i z_j \right).$$

Note that

$$y_k' = g_k \left(y_1, ..., y_n, y_2', ..., y_{k-1}' \right).$$

Next, we prove by induction on k that there are $\mathcal{I}$-recursive functions g_k^*, $k = 2, ..., n$, such that

$$y_k' = g_k^*(y_1, ..., y_n).$$

For $k = 2$ the result is immediate. Take $g_2^* = g_2$. For arbitrary $k > 2$ we know that

$$y_k' = g_k \left(y_1, ..., y_n, y_2', ..., y_{k-1}' \right), \tag{3.2}$$

and also, by induction hypothesis,

$$y_i' = g_k^*(y_1, ..., y_n), \qquad \text{for } j = 2, ..., k-1.$$

Substituting the last $k-2$ equations in (3.2), we get

$$y_k' = g_k \left(y_1, ..., y_n, g_2^*(y_1, ..., y_n), ..., g_{k-1}^*(y_1, ..., y_n) \right).$$

If we take g_k^* as the composition of g_k with $U_1^n, ..., U_n^n, g_2^*, ..., g_{k-1}^*$, then we get the desired function that is also $\mathcal{I}$-recursive.

Now, suppose that f is defined on an interval I, and that the initial conditions of the integrators are prescribed at $x_0 = a$. We can, without loss of generality, consider $x_0 = 0$ for our next purpose.[22]

Then, y_2 can be obtained by integration in the following way:

$$\begin{bmatrix} y_2(0) \\ \vdots \\ y_n(0) \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \quad , \qquad \partial_x \begin{bmatrix} y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} g_2^* \\ \vdots \\ g_n^* \end{bmatrix}. \tag{3.3}$$

Switching the rows in the last equations, we can show that $y_2, ..., y_n$ are all $\mathcal{I}$-recursive. If y is generated by the FF-GPAC $\mathcal{U}$, then there exists $\mathcal{I}$-recursive constants $c_0, ..., c_n$, such that

$$y = c_0 + c_1 x + c_2 y_2 + ... + c_n y_n.$$

Therefore y is $\mathcal{I}$-recursive. ■

The previous theorem can be easily extended to every function algebra $\mathcal{J}$ satisfying $\mathcal{I} \subseteq \mathcal{J}$. In particular, we have the following case.

Definition 3.5.2 $\mathcal{I}_{\mathbb{R}} = [\mathbb{R}, U, +, \times; CM, INT]$.

[22]See the considerations at the end of section 3.2.

Corollary 3.5.3 *Suppose that a unary function f of class C^∞ can be generated by a FF-GPAC $\mathcal{U}$, in some interval I. Then f is $\mathcal{I}_\mathbb{R}$-recursive on I.*

Corollary 3.5.4 *Suppose that f is differentially algebraic on a closed, bounded interval I with non-empty interior. Then there is a closed subinterval $I' \subseteq I$ with non-empty interior such that, on I', f is $\mathcal{I}_\mathbb{R}$-recursive.*

Proof. This follows immediately from theorem 2.4.6. ■

In proposition 9 of [Moo96] it is stated that the unary functions computable by the T-GPAC are precisely the unary $\mathbb{R}$-recursive functions that do not use the operator $\bar{\mu}$ (the class of $\mathbb{R}$-recursive functions that do not use the operator $\bar{\mu}$ is denoted by M_0). However, this assertion present some gaps.[23] In fact, all unary functions generated by a T-GPAC must be differentiable on their entire domain. But $\lambda x.\, |x| \in M_0$ (cf. [CMC00, p. 647]) is not differentiable on $\mathbb{R}$.

Similarly, we conclude that $\mathcal{I}_\mathbb{R} \subsetneqq M_0^1$, where M_0^1 is the set of unary functions in M_0, because $\mathcal{I}_\mathbb{R}$ is constituted only by C^∞ function (the operators of $\mathcal{I}_\mathbb{R}$ preserve smoothness). Therefore, by corollary 3.5.3, we conclude that there are unary functions in M_0 that are not generated by a FF-GPAC (at least, in their entire domain).

We now seek a converse relation to theorem 3.5.1. Note that all the inputs in a FF-GPAC depend only on one independent variable, but functions in $\mathcal{I}$ may depend on several independent variables. Therefore, in order to achieve our purposes, we have to suppose that all the arguments of a function in $\mathcal{I}$ depend only on one parameter t. This is done as follows.

Theorem 3.5.5 *Suppose that $f \in \mathcal{I}$ is an n-ary function in which, without loss of generality, $x_1, ..., x_m$ are the locked variables and $x_{m+v}, ..., x_n$ are the active variables. Suppose that*

$$\tilde{f} = \lambda t. f(a_1, ..., a_m, 0, ..., 0, \varphi_{m+v}(t), ..., \varphi_n(t)),$$

is defined on some interval $[t_0, t_f)$, where $t_f > t_0$ may possibly be ∞, and $\varphi_{m+v}, ..., \varphi_n$ are C^1-functions. Then there exists a FF-GPAC having only as constant units those associated to the values $-1, 0, 1, a_1, ..., a_m$, working with inputs $\varphi_{m+v}, ..., \varphi_n$, that can generate $\tilde{f}$ in $[t_0, t_f)$

Proof. The proof is done by induction in the structure of the description. The functions $-1, 0, 1$ can be obtained in a straightforward way using constant units associated to their respective values. Similarly U_i^n can be obtained using a circuit built only with input units, by taking the output of the input unit associated to x_i.

For the case of $+$, it is enough to use an adder and two input units connected to the adder to show the result. In the case of product, use the circuit of figure

[23]The proof of this proposition relies on the lemma preceding it that states that M_0 is closed under differentiation and inversion. However, the proof of this lemma presents some vicious circles when considering the operation of integration.

1.1.4 (with extra input units on the right). To initialize it, first initialize the circuit of figure 1.1.4 and then connect it to the input units (by any order).

Inductive step: consider now the operators. Suppose that we have $f(\mathbf{x}) = g(f_1(\mathbf{x}), ..., f_k(\mathbf{x}))$ for all $\mathbf{x}$. Suppose, without loss of generality, that g has $x_1, ..., x_s$ and $x_{s+d}, ..., x_k$ as LVs and AVs, respectively. Because the locked variables of f are $x_1, ..., x_m$, we conclude that $f_1, ..., f_s$ cannot depend on $x_{m+1}, ..., x_n$. Let $x_{i_1}, ..., x_{i_\alpha}$, and $x_{j_1}, ..., x_{j_\beta}$ be the LVs and AVs of f_r, for $r = 1, ..., k$, with

$$i_1 < ... < i_\alpha \text{ and } j_1 < ... < j_\beta.$$

We have $i_\alpha, j_\beta \leq m$ for $r = 1, ..., s$ (and $i_\alpha \leq m$ for $r = s+d, ..., k$). Using the induction hypothesis, there exists FF-GPAC $\mathcal{F}_r$ that generates

$$\lambda t.f_r(a_{i_1}, ..., a_{i_\alpha}, 0, ..., 0, \varphi_{j_1}(t), ..., \varphi_{j_\beta}(t), 0, ..., 0),$$

(the actual order of the variables could be different, but this is not important for us) using only constant units associated to the values $-1, 0, 1, a_{i_1}, ..., a_{i_\alpha}$, for $r = 1, ..., k$. Now substitute in $\mathcal{F}_r$, for $r = 1, ..., s, s+d, ..., k$, the input units associated to x_t by a_t, for $t = 1, ..., m$, respectively (hence, these FF-GPACs compute f_r, with the first m arguments fixed). We still denote these new FF-GPACs by $\mathcal{F}_r$ (the initialization procedure is the same of the "old" FF-GPAC). Hence $\mathcal{F}_r$ computes

$$f_r(a_{i_1}, ..., a_{i_\alpha}, 0, ..., 0, a_{j_1}, ..., a_{j_\beta}, 0, ..., 0) = b_r,$$

for $r = 1, ..., s$. Also by induction hypothesis, there exists a FF-GPAC $\widetilde{\mathcal{G}}$ that computes

$$\lambda t.g(b_1, ..., b_s, 0, ..., 0, \widetilde{\varphi}_{s+d}(t), ..., \widetilde{\varphi}_k(t)),$$

This FF-GPAC only uses constant units associated to the values $-1, 0, 1, b_1, ..., b_s$. Next, we substitute all the constant units of $\widetilde{\mathcal{G}}$ associated to $b_1, ..., b_s$ by the FF-GPACs $\mathcal{F}_1, ..., \mathcal{F}_s$.

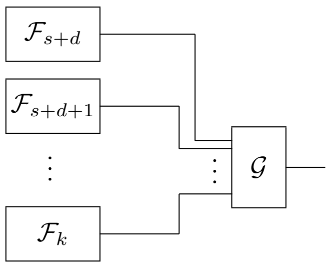


Figure 3.4.1: A circuit that computes the composition of functions.

We still maintain the initialization procedure (where to the former constant units correspond now the FF-GPACs $\mathcal{F}_1, ..., \mathcal{F}_s$). Hence, we obtain a new FF-GPAC $\mathcal{G}$, using only constant units associated to the values $-1, 0, 1, a_1, ..., a_m$, that computes

$$\lambda t.g(f_1, ..., f_s, 0, ..., 0, \widetilde{\varphi}_{s+d}(t), ..., \widetilde{\varphi}_k(t)),$$

With these FF-GPACs, we can build a FF-GPAC that computes the composition of functions. This is sketched in figure 3.4.1.

For initialization, first initialize properly $\mathcal{F}_{s+d}, ..., \mathcal{F}_k$. Then initialize properly $\mathcal{G}$, connecting the respective inputs associated to $\mathcal{F}_j$ when solicited, for $j = s+d, ..., k$.

Finally, suppose that $INT(F_1, ..., F_k, G_1, ..., G_k)$ is the description associated with f. Then, by similar arguments to the previous case, there are FF-GPACs $\mathcal{F}_p$, for $p = 1, ..., k$, using only constant units associated to the values $-1, 0, 1, a_1, ..., a_m$ that computes the values

$$f_p(a_1, ..., a_m, 0, ..., 0).$$

Note that if the AVs and LVs of $\bar{f}$ are $x_1, ..., x_j$, with $j \leq m$, then

$$\bar{f}(a_1, ..., a_j, 0, ..., 0) = \bar{f}(a_1, ..., a_m, 0, ..., 0).$$

We can also obtain FF-GPACs $\mathcal{G}_1, ..., \mathcal{G}_m$ using only constant units associated to the values $-1, 0, 1, a_1, ..., a_m$, such that $\mathcal{G}_j$ computes

$$\lambda t. g_j(a_1, ..., a_m, 0, ..., 0, \widetilde{\varphi}_{n+1}(t), ..., \widetilde{\varphi}_{n+k}(t)).$$

Finally

$$\tilde{f} = \lambda t. f(a_1, ..., a_m, 0, ..., 0, \varphi_n(t))$$

can be obtained by the FF-GPAC indicated in figure 3.4.2.

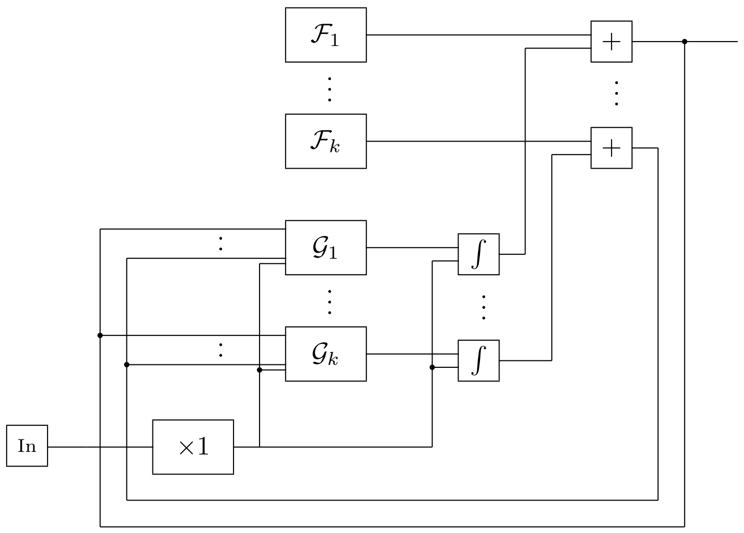


Figure 3.4.2: A circuit that solves integration.

In figure 3.4.2, the box marked with 'In' represents the input unit. To see that we can pick an appropriate enumeration of the integrators of the circuit represented in figure 3.4.2, pick appropriate enumerations for the various subFF-GPACs and then take the following general enumeration:

$$\times 1, \mathcal{F}_1, ..., \mathcal{F}_k, \int_1, ..., \int_k, \mathcal{G}_1, ..., \mathcal{G}_k,$$

where $\int_i$ represents the integrator connected to $\mathcal{G}_i$, for $i = 1, ..., k$, and $\times 1$ represents the constant multiplier (it is considered in the enumeration because we actually use integrators to build it). For the initialization procedure, initialize the FF-GPACs (and also connect the inputs) on the following order: $\mathcal{F}_1, ..., \mathcal{F}_k$, the adders, $\mathcal{G}_1, ..., \mathcal{G}_k$, $\int_1, ..., \int_k$, and finally $\times 1$.

We need the constant multiplier $\times 1$ because when we connect the input unit, we want to actualize *simultaneously* all the inputs connected to this unit (or else, the circuit could not work). ■

Note that the initialization procedure, along with the constant units, permit us to deal with effective FF-GPACs. Without this tool, we would be unable to present the previous theorem.

If, in the definition of $\mathcal{I}$, we change the constants $-1, 0, 1$ to some real constants b_t, $t \in J$, where $J \neq \varnothing$, then we can show, in a similar manner, that:

Theorem 3.5.6 *Let $\mathcal{J} = [B, U, +, \times; CM, INT]$, where $B = \{b_t \in \mathbb{R} : t \in J\} \neq \varnothing$. Suppose that $f \in \mathcal{J}$ is an n-ary function in which, without loss of generality, $x_1, ..., x_m$ are the locked variables and $x_{m+v}, ..., x_n$ are the active variables. Suppose that*

$$\tilde{f} = \lambda t. f(a_1, ..., a_m, 0, ..., 0, \varphi_{m+v}(t), ..., \varphi_n(t)),$$

is defined on some interval $[t_0, t_f)$, where $t_f > t_0$ may possibly be ∞, and $\varphi_{m+v}, ..., \varphi_n$ are C^1-functions. Then there exists a FF-GPAC having only as constant units those associated to the values $a_1, ..., a_m, b_t$, $t \in J$, with inputs $\varphi_{m+v}, ..., \varphi_n$, that can generate $\tilde{f}$ in $[t_0, t_f)$.

Proof. The proof is similar to the proof of the previous theorem. The only problem that may appear is in the case of INT, because we don't have, in principle, constant units associated to the value 1 in order to build the constant multiplier $\times 1$. But we can bypass this problem using instead an adder in which one entry is connected to a constant unit associated to the value 0. Apparently, we have the same problem with this constant unit, but it can be obtained from a FF-GPAC like the one of figure 2.6.2, using constant units associated to some $b \in B$ instead of constant units associated to the value 1. Then first initialize the input associated to the variable of integration and then the input associated to the integrand. The output of this FF-GPAC will be always 0. ■

From the proof of the previous theorem, we see that, because we are assuming that initially the output of every integrator is 0, we are implicitly inserting a constant unit associated to the value 0. If we allowed integrators to have initially as output any real number, then we would obtain all constant units, and this is not desirable.

Corollary 3.5.7 *Suppose that $f \in \mathcal{I}_{\mathbb{R}}$ is an n-ary function in which, without loss of generality, $x_1, ..., x_m$ are the locked variables and $x_{m+v}, ..., x_n$ are the active variables. Suppose that*

$$\tilde{f} = \lambda t.f(a_1, ..., a_m, 0, ..., 0, \varphi_{m+v}(t), ..., \varphi_n(t)),$$

is defined on some interval $[t_0, t_f)$, where $t_f > t_0$ may possibly be ∞, and $\varphi_{m+v}, ..., \varphi_n$ are C^1-functions. Then there exists a FF-GPAC having as inputs $\varphi_{m+v}, ..., \varphi_n$, that can generate $\tilde{f}$ in $[t_0, t_f)$.

Corollary 3.5.8 *Suppose that $f \in \mathcal{I}_{\mathbb{R}}$ is an n-ary function in which, without loss of generality, $x_1, ..., x_m$ are the locked variables and $x_{m+v}, ..., x_n$ are the active variables. Suppose that*

$$\tilde{f} = \lambda t.f(a_1, ..., a_m, 0, ..., 0, t, ..., t),$$

is defined on some interval $[t_0, t_f)$, where $t_f > t_0$ may possibly be ∞. Then $\tilde{f}$ is differentially algebraic on $[t_0, t_f)$.

Proof. This result follows from the previous corollary and corollary 2.4.4. ■

Can we do the reverse of what we have done in the previous theorems? In other words, if we have a FF-GPAC $\mathcal{U}$ with inputs $\varphi_1, ..., \varphi_n$, can we find a recursive function f over the reals such that an output of $\mathcal{U}$ may be represented as

$$u = \lambda t.f(\varphi_1(t), ..., \varphi_n(t)) \text{ ?} \tag{3.4}$$

We already have seen that, if $n = 1$, then the answer is yes (theorem 3.5.1). But for $n \geq 2$ we cannot do this, because of the path dependent behavior already described in the previous chapter.[24] And this problem will hold for all functions algebras. Hence, we believe that it is not possible to extend much further the results of this section.

3.6 Classical Recursion vs Real Recursion

In this section we make a standpoint on the relations between classical recursion and recursion over the reals.

In classical recursion we have the advantage of having a suitable computational model - the Turing machine. With this model we can naturally introduce complexity notions, such as time and space complexity. We can also code and decode a computation in terms of recursive functions.

With real functions, the situation changes. Although we introduced a model (the FF-GPAC) that enables us some connections with a subclass of recursive functions over the reals (the class $\mathcal{I}_{\mathbb{R}}$), these connections are not perfect. Take, for example, the case of functions depending on more than one variable, where

[24]Although we probably could do this for some particular cases, e.g. when $\varphi_1, ..., \varphi_n$ are $\mathcal{I}$-recursive functions.

the path dependent behavior appears. Note also that $\mathcal{I}_{\mathbb{R}}$ is almost at the bottom of the classes of recursive functions over the reals that we have introduced in this chapter (the bottom is $\mathcal{I}$). Hence, for example, in order to obtain a computational model for $\mathcal{M}$, we would need to introduce a stronger model than the FF-GPAC. However, we don't know if this goes beyond the physical limits. In fact, the FF-GPAC enables us to compute a large class of functions. Hence, stronger models than the FF-GPAC should be treated with care.

This is not the unique characteristic that apparently distinguishes classical recursion from recursion over the reals. For instance, in classical recursion, we have a bijective function $f : \mathbb{N}^2 \to \mathbb{N}$, where f and f^{-1} are computable (i.e., there are effective procedures to determinate their values) - cf. [BM77]. Hence, there are computable bijective functions between $\mathbb{N}^k$ and $\mathbb{N}^n$, for $k, n \in \mathbb{N}\backslash\{0\}$. Therefore, we can reduce the problem of working with functions on n variables to the problem of working with functions depending on only one variable.

Does this happen for the real case? Well, if we expect that f and f^{-1} depend continuously on their parameters, the answer is *no,* as the following theorem states [Con93, corollary 1.1.6].

Definition 3.6.1 *Two topological spaces X and Y are called homeomorphic if there exists a bijective function $f : X \to Y$ such that f and f^{-1} are continuous. The function f is called a homeomorphism.*

Theorem 3.6.2 *If $U \subseteq \mathbb{R}^n$ and $V \subseteq \mathbb{R}^m$ are open subsets such that U and V are homeomorphic, then $n = m$.*

Therefore, there are no homeomorphisms between $\mathbb{R}^n$ and $\mathbb{R}^m$, for $n \neq m$. Hence, several results that can be proved with these bijections in the standard recursion theory, are no longer valid for the real case. For example, we can no longer say that $\bar{\Sigma}_n^0$ is closed under existential quantification, as in the case for Σ_n^0 (cf. [Odi89]).

However, several connections exist between classical and real recursion. For example, refer to the previous section or to [CC97, CMC, CMC00, CM01, Cam02]. Therefore, we could see recursion theory over the reals as an extension of classical recursion theory, although with a (possibly) different structure.

Finally, we would like to make some last comments on computable functions over the reals. Although we might expect that computable functions over the reals are continuous, we don't know if we should take that condition. For instance, the classes $\mathcal{M}_0$ and $\mathcal{M}$ contain several discontinuous functions, such as Θ, that seem to be sufficiently simple to be considered "effective." And if we go to other fields such as electronics or physics, it is very common to work with discontinuous functions (e.g. square waves). Although these functions are actually continuous, in practice they change their values so quickly that they can be though and idealized (and they actually are) as discontinuous functions. Hence, the question of taking computable functions over the reals as continuous/discontinuous still deserves more attention.

Chapter 4

Conclusion

4.1 Final Remarks and Open Problems

In this dissertation we have introduced the reader to some models of analog computation and we have further developed them. Concretely, we have gone to the roots of analog computation theory, by introducing the pioneering work of Claude Shannon about the General Purpose Analog Computer. We then continued this work, indicating the main contributions done, namely by Pour-El, and also by Rubel and Lipshitz. We also supplied the reader with the main results known to the present.

We then continued our work on the GPAC, indicating several problems from which this model suffers. Then, we presented an alternative approach to the GPAC (FF-GPAC), and we showed that this new model is more robust than the GPAC. We also showed that this model preserves all the significative relations that were deduced for the case of the T-GPAC (in particular, it preserves the equivalence with differentially algebraic functions).

We continued by introducing the concept of effective GPAC and also the concept of initialization procedure, that we have found important in order to capture some properties of the differential analyzer.

The rest of the work is dedicated to recursion theory. We presented the classical theory and then pursued with a recent topic: recursion function theory over the reals. We introduced the basic definitions and results for this theory and also proposed some extensions to it. In particular, we have proposed similar notions to primitive recursive, recursive, and partial recursive functions for the real case. We also introduced an analog of the Arithmetical Hierarchy over the reals. Finally, we related recursive functions over the reals with the FF-GPAC and analyzed the relations between classical recursion and recursion theory over the reals.

Several open questions can be listed. Let us present some of them.

1. In a FF-GPAC (GPAC), what happens if we allow the use of other types of units, i.e., if we allow units that have a different behavior from the

ones considered here (for example, what happens if we allow a unit that outputs $\Gamma(x)$ when having as input the value x)? Can we still guarantee the existence and uniqueness of solutions in some interval? Can we still relate this new model with some class of recursive functions over the reals? In particular, what happens if we allow units with discontinuous outputs? (Note that the integrator units may not work in this case)

2. An important question that was already described for the GPAC still remains unsettled for the FF-GPAC: what is the influence of small perturbations? And if a FF-GPAC is subjected to some kind of perturbation, can we find a procedure by which we can compute with any preassigned precision? This is an important issue in order to determinate the feasibility of physical implementations for an arbitrary FF-GPAC. Although we didn't touch this topic in our research, we are aware of its importance. We believe that some work in this direction is possible by establishing some connections with the theory of ordinary differential equations.

3. Can we introduce natural complexity measures in defining the FF-GPAC? For example, we saw in theorem 2.4.3 that the number of integrators in a FF-GPAC can be related with the order of the differentially algebraic function that it generates. This result indicates that this task is apparently feasible, although possibly with some limitations.

4. Can we find some device that can simulate the initialization procedure and every FF-GPAC? Perhaps we might be able to find a device, maybe hybrid, which performs this task by introducing some bounds on the number of units used by a FF-GPAC.

5. Can we find some physically feasible models of computation that compute the functions in $\mathcal{G}$ and $\mathcal{M}$, similarly to the functions in $\mathcal{PR}$ and $\mathcal{R}$? Although we didn't answer this question, we believe that this is not possible. As we have seen, the subclass $\mathcal{I}_{\mathbb{R}}$ may be related to FF-GPAC computable functions, that includes a large class of functions. Hence, it seems that a model of computation for one of the classes indicated above would have too much computational power to be physically implemented.

6. Does the Real Arithmetical Hierarchy collapses? And are the classes $\bar{\Sigma}_n^0$ and $\bar{\Pi}_n^0$ closed under existential and universal quantification, respectively, as it happens in the classical case?

7. Should we expect computable functions over the reals to be continuous, or should we allow discontinuous functions to be computable? To answer this question, we probably have to consider not only mathematical results, but also physical arguments, in order to find which is the best approach.

Glossary of Symbols

$u(t)$	expression of function u depending on the parameter t (time)
$\int_{t_0}^{t} u(x)dv(x)$	expression of the Riemann-Stieljes integral
t	independent variable (time)
$P(x_1,...,x_n)$	expression of polynomial P on the variables $x_1,...,x_n$
$\mathbf{x}$	expression of vector $(x_1,...,x_n)$
$f(\mathbf{x})$	value of the scalar function f on $(x_1,...,x_n)$
$\mathbf{g}(\mathbf{x})$	value of the vectorial function $(g_1,...,g_n)$ on $(x_1,...,x_n)$
$\frac{d\mathbf{y}}{dx}, \mathbf{y}'$	expressions for $\left(\frac{dy_1}{dx},...,\frac{dy_n}{dx}\right)$
$\varphi_1,...,\varphi_n$	input functions applied to a circuit
$\mathcal{A},\mathcal{B},\mathcal{C},...$	letters denoting GPACs/FF-GPACs
$\mathcal{A}_1,\mathcal{A}_2,...$	units of a GPAC/FF-GPAC
$\mathcal{U}_1,...,\mathcal{U}_n$	integrators of the circuit $\mathcal{U}$
$\left\Vert\varphi\right\Vert_\infty$	sup-norm (on some interval I) $\left\Vert\varphi\right\Vert_\infty = \sup_{x\in I}\left\vert\varphi(x)\right\vert$
$\equiv 0$	$f\equiv 0$ on S iff $(\forall \mathbf{x}\in S)(f(\mathbf{x})=0)$
$\not\equiv 0$	$f\not\equiv 0$ on S iff $(\exists \mathbf{x}\in S)(f(\mathbf{x})\neq 0)$
$[B;OP]$	function algebra where B and OP are the sets of basic functions and operators, respectively
$\mathcal{J}$	letter denoting a function algebra
$\mathcal{J}+f$	the function algebra $[f,B;OP]$
Z	the 0-ary function defined by $Z=\lambda.0$ (in $\mathbb{N}$)
S	successor function: $S=\lambda x.x+1$ (in $\mathbb{N}$)
U_i^n	projection: $U_i^n=\lambda x_1...x_n.x_i$
U	$U=\{U_i^n : n,i\in\mathbb{N} \text{ and } 1\leq i\leq n\}$
C	composition operator
REC	primitive recursion operator
$\hat{\mu}$	μ-recursion operator
μ	minimalization operator; unrestricted μ-recursion operator
$\bar{\mu}$	minimalization operator (on the reals)
$\perp$	symbol for undefined
$\mathcal{PR}$	the set of primitive recursive functions
$\mathcal{R}_0$	the set of recursive functions

$\mathcal{R}$	the set of partial recursive functions
$\int$	integration operator
$\dot{-}$	cut-off difference
I	proper integration operator[1]
$\mathcal{D}$	$\mathcal{D} = [-1, 0, 1, U; C, I]$
θ_∞	the function that for each x is 0 for $x \leq 0$, and $\exp(-1/x)$ for $x > 0$
$\mathcal{G}$	$\mathcal{G} = [-1, 0, 1, \theta_\infty, U; C, I]$
$\mathcal{M}$	$\mathcal{M} = [-1, 0, 1, \theta_\infty, U; C, I, \mu]$
$\bar{\mathcal{M}}$	$\bar{\mathcal{M}} = [-1, 0, 1, \theta_\infty, U; C, I, \bar{\mu}]$
$\mathcal{M}_0$	$\mathcal{M}_0 = [-1, 0, 1, \theta_\infty, U; C, I, \hat{\mu}]$
Θ	Heaviside function, i.e. the function that for each x is 0 for $x < 0$ and 1 for $x \geq 0$
δ	Kronecker δ-function, i.e. the function that for each x is 0 for $x \neq 0$ and 1 for $x = 0$
$f^{[n]}$	function f iterated n times
$\chi_{\mathbb{N}}$	characteristic of $\mathbb{N}$
c_R	characteristic of the predicate R
$\exists$	existential quantifier
$\forall$	universal quantifier
$\Delta_j^0, \Sigma_j^0, \Pi_j^0$	sets in the j-level of the Arithmetical Hierarchy
Δ_ω^0	$\Delta_\omega^0 = \cup_{n \in \mathbb{N}} \Delta_n^0$
$\bar{\Delta}_j^0, \bar{\Sigma}_j^0, \bar{\Pi}_j^0$	sets in the j-level of the Arithmetical Hierarchy over the reals
$\times, +$	basic functions $\times, +$
CM	operator CM
INT	operator INT
$\deg F$	degree of the description F
$\mathcal{I}$	$\mathcal{I} = [-1, 0, 1, U, +, \times; CM, INT]$
$\mathcal{I}_{\mathbb{R}}$	$\mathcal{I}_{\mathbb{R}} = [\mathbb{R}, U, +, \times; CM, INT]$

[1]Sometimes I may be an interval. However the context will usually be sufficient to decide the role of I.

Bibliography

[Atk89] K. E. Atkinson. *An Introduction to Numerical Analysis,* John Wiley & Sons, 2nd Edition, 1989.

[BCSS98] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation,* Springer, 1998.

[BM77] J.L. Bell and M. Machover. *A course in Mathematical Logic,* North-Holland, 1977.

[Bow96] M. D. Bowles. U.S. technological enthusiasm and british technological skepticism in the age of the analog brain. *IEEE Annals of the History of Computing*, 18(4):5-15, 1996.

[BR89] G. Birkhoff and G. C. Rota. *Ordinary Differential Equations,* John Wiley & Sons, 4th Edition, 1989.

[Bra95] M. S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science,* 138(1):67-100, 1995.

[Bur71] Bureau of Naval Personnel. *Basic Machines and How They Work,* Dover Publications, Inc., 1971.

[Bus31] V. Bush. The differential analyzer. A new machine for solving differential equations. *J. Franklin Institute*, 212:447-488, 1931.

[Cam02] M. L. Campagnolo. *Computational Complexity of Real Valued Recursive Functions and Analog Circuits.* PhD thesis, Instituto Superior Técnico/UTL, 2002.

[Car77] B. C. Carlson. *Special Functions of Applied Mathematics*, Academic Press, 1977.

[CC97] M. L. Campagnolo and J. F. Costa. From recursive to $\mathbb{R}$-recursive functions. Research report, Section of Computer Science, Department of Mathematics, IST/UTL, Lisbon, 1997.

[CJ89] R. Courant and F. John. *Introduction to Calculus and Analysis,* volume I, Springer-Verlag, 1989.

[Clo99] P. Clote. Computational models and function algebras. In E. R. Griffor, editor, *Handbook of Computability Theory,* 589-681, Elsevier, 1999.

[CM01] M. L. Campagnolo and C. Moore. Upper and lower bounds on continuous-time computation. In I. Antoniou, C. S. Calude, and M. J. Dinneen, editors, *Unconventional Models of Computation, UMC'2K*, DMTCS, 135-153, Springer-Verlag, 2001.

[CMC] M. Campagnolo, C. Moore, and J. F. Costa. An analog characterization of the Grzegorczyk hierarchy. *Journal of Complexity*, to appear.

[CMC00] M. L. Campagnolo, C. Moore, and J. F. Costa. Iteration, inequalities, and differentiability in analog computers. *Journal of Complexity*, 16(4):642-660, 2000.

[Con93] L. Conlon. *Differentiable Manifolds - A first Course,* Birkhäuser, 1993.

[Cut80] N. J. Cutland. *Computability: An introduction to Recursive Function Theory,* Cambridge University Press, 1980.

[Gla71] M. D. Gladstone. Simplifications of the recursive scheme. *Journal of Symbolic Logic*, 36(4):653-665, 1971.

[Hun96] T. W. Hungerford. *Algebra,* Springer, 8th printing, 1996.

[KCG94] P. Koiran, M. Cosnard, and M. Garzon. Computability with low-dimensional dynamical systems. *Theoretical Computer Science,* 132:113-128, 1994.

[KM99] P. Koiran and C. Moore. Closed-form analytic maps in one or two dimensions can simulate Turing machines. *Theoretical Computer Science,* 210:217-223, 1999.

[Koi93] P. Koiran. *Puissance de Calcul des Réseaux de Neurones Artificiels.* PhD thesis, École Normale Supérieure de Lyon, 1993.

[Lan93] S. Lang. *Real and Functional Analysis,* Springer-Verlag, 3rd Edition, 1993.

[Lip65] S. Lipschutz. *Theory and Problems of General Topology,* Schaum's Outline Series, McGraw-Hill, 1965.

[LR87] L. Lipshitz and L. A. Rubel. A differentially algebraic replacement theorem, and analog computation. *Proceedings of the A.M.S.,* 99(2):367-372, 1987.

[McC94] J. L. McCauley. *Chaos, Dynamics and Fractals - an Algorithmic Approach to Deterministic Chaos,* Cambridge University Press, First Paperback Edition, 1994.

[Moo90] C. Moore. Unpredictability and undecidability in dynamical systems. *Physical Review Letters,* 64:2354-2357, 1990.

[Moo96] C. Moore. Recursion theory on the reals and continuous-time computation. *Theoretical Computer Science,* 162:23-44, 1996.

[MP43] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics,* 5:115-133, 1943.

[Mun00] J. R. Munkres. *Topology,* Prentice-Hall, 2nd edition, 2000.

[Myc] J. Mycka. μ-recursion and infinte limits. *Theoretical Computer Science*, to appear.

[Odi89] P. Odifreddi. *Classical Recursion Theory,* Elsevier, 1989.

[Orp95] P. Orponen. On the computational power of continuous time neural networks. NeuroCOLT Report NC-TR 95-051, Royal Holloway College, Univ. London, Dept. of Computer Science, 1995.

[Orp97] P. Orponen. A survey of continuous-time computation theory. In D.-Z. Du and K.-I. Ko, editors, *Advances in Algorithms, Languages and Complexity*, 209-224, Kluwer Academic Publishers, 1997.

[Pou74] M. B. Pour-El. Abstract computability and its relations to the general purpose analog computer. *Transactions Amer. Math. Soc.*, 199:1-28, 1974.

[RR93] M. Renardy and R. C. Rogers. *An Introduction to Partial Differential Equations,* Springer-Verlag, 1993.

[Rub88] L. A. Rubel. Some mathematical limitations of the general-purpose analog computer. *Advances in Applied Mathematics,* 9:22-34, 1988.

[Rub93] L. A. Rubel. The extended analog computer. *Advances in Applied Mathematics*, 14:39-50, 1993.

[Sha41] C. Shannon. Mathematical theory of the differential analyzer. *J. Math. Phys. MIT*, 20:337-354, 1941.

[Sie99] H. T. Siegelmann. *Neural Networks and Analog Computation: Beyond the Turing Limit,* Birkhäuser, 1999.

[SS94] H. T. Siegelmann and E. D. Sontag. Analog computation via neural networks. *Theoretical Computer Science,* 131:331-360, 1994.

[Str88] G. Strang. *Linear Algebra and its Applications*, Harcourt Brace Jovanovich, 1988.

[Str99] D. W. Stroock. *A Concise Introduction to the Theory of Integration,* Birkhäuser, 3rd edition, 1999.

[SW97] A. I. Saichev and W. A. Woyczyński. *Distributions in the Physical and Enginnering Sciences*, volume 1, Birkhäuser, 1997.

[Zwi96] D. Zwillinger, editor-in-chief, *CRC Standard Mathematical Tables and Formulae,* CRC Press, 30th Edition, 1996.

Index