Applications
of
Real Recursive Infinite Limits

Luís Miguel Pacheco Mendes Gomes

Dissertation submitted to the University of Azores to obtain the degree of Doutor in Informatics (speciality of Theory of Computation).

July 2006

Luís Miguel Pacheco Mendes Gomes

Applications of Real Recursive Infinite Limits

Dissertation submitted to the Universidade dos Açores to obtain the degree of Doutor in Informatics (speciality of Theory of Computation), supervised by José Félix Costa, Associate Professor with Aggregation of the Department of Mathematics, Instituto Superior Técnico, Universidade Técnica de Lisboa.

July 2006

In memory of my grandmother Margarida (1911-2004)
and
my mother Regina (1944-2006)

Resumo

Usando a teoria das funções reais recursivas, que deriva da proposta original em [Moo96], mostramos como cada função periódica definida por partes, que admite um desenvolvimento em série de Fourier, pode ser definida como uma destas funções reais recursivas. Demonstramos, também, que o poder computacional de um certo tipo de autómatos finitios em tempo contínuo está limitado à computação de sinais que são descritos por funções lineares parcialmente periódicas definidas por partes, as quais constituem um subconjunto muito restrito de sinais que podem ser gerados por funções reais recursivas.

Uma função real recursiva com limites infinitos é apresentada para simular máquinas de Turing em tempo infinito, restrito a ω^2, bem como o seu poder computacional, nomeadamente para decidir as respectivas aproximações ω^2 aos problemas da paragem e, ainda, a hierarquia da aritmética recorrendo a um número finito de limites. Para isso, é introduzido um novo esquema de iteração nos ordinais até ω^2, que simula as máquinas de Turing em tempo infinito com a codificação para inputs binários finitos, introduzida por Christopher Moore, e o sistema de equações diferenciais da simulação da máquina de Turing, introduzido, recentemente, por Jerzy Mycka e José Félix Costa.

Abstract

Taking the most simple kind of finite state automata, typically used in the digital stage, whose states are continuous instead of discrete, we show that such automata can only recognize periodic infinite patterns. In our case such patterns are generated by real recursive functions, a new trend in analog computation, which are an extension to reals of Kleene' s recursive functions. And, thus, we show that automata can only recognize periodic real recursive functions, which we also show that are naturally approximated by Fourier series. With these results in hand, we are bringing together not only the concept of periodicity into the real recursive function theory, and consequently the Fourier series, but also the automata, with continuous states, and their computational limits, in a new mathematical characterization of hybrid finite computation.

Another application of real recursive functions with infinite limits is introduced for the simulation of infinite time Turing machines and their computational power, namely their ability to decide their halting problems, restricted to ω^2, and the arithmetic sets with a finite number of limits. To do this, we introduce a new kind of iteration schema over ordinals and we recover the codification over the reals for finite inputs, introduced by Christopher Moore, as well as the system of differential equations involved in the simulation of Turing machines presented, recently, by Jerzy Mycka e José Félix Costa.

Preface

The theory of Analog Computation, where the internal states of a computer are continuous rather than discrete, has enjoyed a recent resurgence of interest. This stems partly from a wider program of exploring alternative approaches to computation, such as neural and quantum computation; partly as an abstraction of numerical algorithms where real numbers can be thought of as entities in themselves, rather than as strings of digits, and partly from a desire to use the tools of computation theory to better classify the variety of continuous dynamical systems used to model our world (or at least its classical limit).

Cristopher Moore, in 1996, define a class of recursive functions over the reals, including many functions which are uncomputable, analogous to the classical recursive functions on the natural numbers, corresponding to a conceptual analog computer that operates in continuous time. He also stratifies such class into a μ-hierarchy, according to the number of uses of the zero-minimalisation operator μ—the classical minimalisation operator over reals. At the lowest level, he obtains continuous functions that are differentially algebraic, and computable by Shannon's General Purpose Analog Computer (the so called GPAC), and, at higher levels, he obtains increasingly discontinuous and complex functions. In the last years, recursive functions over the reals have been considered, first as a model of analog computation, and second to obtain analog characterizations of classical computational complexity classes. However, the minimalisation operator has not been considered, partially, because it does not fit well the analytic realm of analog computation.

Jerzy Mycka and José Félix Costa, in 2003, introduce a most natural operator borrowed from Analysis: the operator of taking a limit, which can be used properly to enhance the recursion theory over the reals, providing good so-

lutions to puzzling problems raised by the mentioned Moore's model. Moreover, apart from the analytical characterisation of classical complexity classes and the introduction of a bounded quantification to treat nondeterminism, the counting of nested limits required to define a real recursive function generates leads us to consider that the class of real recursive functions can be stratified by a potentially infinite hierarchy—a hierarchy of infinite limits called η-hierarchy. In the first meaningful level of such hierarchy we have the extensions of classical primitive recursive functions; in the second level, we have partial recursive functions; and, in the following level, we have the solution to the Halting Problem.

The core of this dissertation, which main objective is to study two applications of real recursive functions emphasizing the role of infinite limits, is separated into 2 parts: the first part, which corresponds to the Chapter 3, we deal with a subclass of real recursive functions called the periodic real recursive functions, and show that classical finite state automata, usually taken by the Hybrid Systems community, only recognize signals generated by such real recursive functions; the second part, which corresponds to the Chapter 5, we make an incursion into Hypercomputation, and show that there are real recursive functions which are able to simulate infinite time Turing machines wtih their computational power in ω^2, using a new iteration schema over ordinals, which is also simulate by a particular system of differential equations. And, chapters 1 and 2 provide, respectively, a short historical perspective of analog computation since its roots, and a short presentation equipped with concepts and results needed from classical recursion theory and real recursion theory with infinite limits, and, moreover, Chapter 4 presents a short overview about ordinals, infinite time Turing machines and their halting problems.

Finally, I would like to emphasize that all the mentioned work was done in the last one and half year, which also produced one paper to be submitted, titled "Hybrid finite computation", and another on going paper that will be titled "Simulating infinite time Turing machines with real recursive functions".

Acknowlegments

Firstly, I wish to thank specially to José Félix Costa (*Professor Félix*) who is, undoubtably, an inexhaustible source of wisdom, motivation and enthusiasm in a truly scientific spirit, that, fortunately, I have been the privilege of his rigorously and patient guidance since the mid 2004.

I wish to thank Bruno Loff for his fruitful discussions and suggestions in the last stage of my research.

I wish to thank Maria Isabel Marques Ribeiro, the Head of Department of Mathematics of University of Azores, for her support and encouragement, and also to thank my colleagues of Section of Informatics of Department of Mathematics of University of Azores for their availability to minimize my teaching duties during the academic year of 2005/2006.

Finally, I wish to thank Laura and (the little) Francisca to their existence, support, and efforts to make me happy in the bad days.

This work was partially supported by the European Union program PRODEP III, Medida 5/Acção 5.3-Formação Avançada de Docentes do Ensino Superior and Concurso 2/5.3/PRODEP/2001.

Notation

α, β, λ (possibly indexed) range over Ord

a, b, c, d (possibly indexed or primed) range over $\mathbb{Z}$

f, g, h, F, G (possibly indexed or primed) denote functions

i, j, k, n, m, p (possibly indexed) range over $\mathbb{N}$

s (possibly indexed or primed) range over piecewise linear signals

t (possibly indexed or primed) range over $\mathbb{R}_0^+$

u, v, w (possibly indexed) range over $\mathbb{Z}^\omega$

x, y, z (possibly indexed) range over $\mathbb{R}$

Contents

Chapter 1

Analog computation: historical perspective

Since the 1930's, the digital computational paradigm has been the most important computational model, mainly due to the unifying work of Turing which clarified the notion of algorithm. Rapidly, several other equivalent approaches to digital computation appeared (e.g., recursive functions in the sense of Kleene, originating a consistent theoretical ground to classical computation theory. Nevertheless, the computation need not to be digital. In fact, the first computers were analog, where internal states are continuous rather than discrete. The analog computers were well suited to solve ordinary differential equations but, infortunately, because of the inexistence of a coherent theoretical basis to analog computation and because 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 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 this situation.

In analog computers, each real number is handled exactly and it is considered an intrinsic quantity, whereas in digital computers such number is represented (and approximated) by (a finite) sequence of bits. It seems that analog computation is more appropriated for studying notions of computability

and complexity for continuous dynamical system (generally speaking, every computational model can be seen as a dynamical system). The main property that distinguishes analog model from digital ones is the use of a continuous state space instead of a discrete state space. Besides this feature there is no agreement upon the properties that characterize an analog model of computation. Recent research shows that Turing machines, when converted into discrete dynamical systems, can be embedded in analog systems. So, we can 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. Current research suggests some lines of work. Some analog models can be seen as high dimensional dynamical systems (highly parallel models), e.g., neural networks [Sie98], and others may be seen as low dimensional dynamical systems. One the other way, we may classify analog models as discrete time models or as continuous time models. However, this is not a rigid characterization and it is possible to find hybrid models (e.g., [Bra95]).

We go back to the roots of analog computation theory by starting with Claude Shannon's so-called General Purpose Analog Computer (GPAC).[1] This was defined as a mathematical model of an analog device, the Differential Analyzer, whose fundamental principles were described by Lord Kelvin in 1876 [Kel76]. The Differential Analyzer was developed at MIT under the supervision of Vannevar Bush, and it was indeed built in 1931 and rebuilt with important improvements in 1941. The input of Differential Analyzer was the rotation of one or more drive shafts and its output was the rotation of one or more output shafts. The main units were gear boxes and mechanical friction wheel integrators, the latter invented by the Italian scientist Tito Gonella in 1825. From the early 1940s, the differential analyzers at Manchester, Philadelphia, Boston, Oslo and Gothenburg, among others, were used to solve problems in engineering, atomic theory, astrophysics, and ballistics, until they were dismantled in the 1950s and 1960s following the advent of electronic analog computers and digital computers (see [Bow96, Hol96] for further details).

The first main paradigm of analog computation was Shannon's GPAC.

[1] In spite of being called "general'", which distinguish it from special purpose analog computing devices, the GPAC is not a uniform model in the sense of von Neumann.

In [Sha41], Shannon showed that the GPAC generates the *differentially algebraic functions*, which are unique solutions of polynomial differential equations with arbitrary real coefficients. This set of functions includes simple functions like the exponential and trigonometric functions as well as sums, products, and compositions of these, and solutions of differential equations formed from them. Marion Pour-El in [PE88] made this proof rigorous by introducing the crucial notion of *domain of generation*. However, it is known that, even if the boundary condition is computable by the GPAC, the Dirichlet problem on the disk can not, in general, be solved by the GPAC [Rub93]. Moreover, the Euler's function Γ is not computable by the GPAC, since it is not differentially algebraic [Rub88]. Lee Rubel, in [Rub93], proposed the Extended Analog Computer (EAC). This model has not only more computational power than GPAC and also produces the solutions of a broad class of Dirichlet boundary-value problems for partial differential equations. However, Lee Rubel stresses that the EAC is a conceptual computer and that it is not known whether it can be realized by actual physical, chemical or biological devices.

At the present, only a few researches are delving into the world of general purpose analog computing. Actually, one of the most notable is Jonathan Mills at Indiana University, which patented a new analog computer that he calls Kirchhoff-Lukasiewicz Machine,[2] designed with arrays of Lukasiewicz logic gates based on the continuous valued Lukasiewicz logic. This machine uses simplified electronic components and "continuous value logic" that makes it able to work incredibly fast and process more sensory inputs than a digital computer can handle. Curiously, he began studying butterfly wing patterns, which is also described by differential equations, and trying to model them with an array of Lukasiewicz logic gates. The speed and simplicity of fabrication of Kirchoff-Lukasiewicz Machines suggests that analog machines do have a future. It is important to notice that this technology is capturing attention outside academia, including calls from NASA and Nortel, the Canadian telecommunications company, to discuss possible applications.

The problems of scientific computing often arise from the study of con-

[2] After physicist G. R. Kirchhoff and mathematician J. Lukasiewicz.

tinuous processes, and questions about computability and complexity over the reals are of central importance in laying the foundations for the subject. The first step is defining a suitable computational model for functions over the reals. The notion of a function changed its meaning through centuries. Before Cantor's work it was usually interpreted as some method of computation. Later a function was seen as any relation satisfying some conditions, however not necessarily given in a constructive way. Analog Computation can be viewed as a modern way of an implementation of pre-Cantorian point of view into current mathematics.

Computability and complexity over discrete spaces have been very well studied since the 1930s. Different approaches have been proved to yield equivalent definitions of computability and nearly equivalent definitions of complexity. From the tradition of logic we have the notions of recursiveness and Turing machine, and from computational complexity we have variations of Turing machines and abstract Random Access Machines, which closely model actual computers. All of these converge to define the same well-accepted notion of computability. The Church-Turing thesis asserts that this formal notion of computability is broad enough, at least in the discrete setting, to include all functions that could reasonably be constructed to be computable.

In the continuous setting, where the objects are numbers in $\mathbb{R}$, computability and complexity have received less attention and there is no accepted computation model. Turing defined the notion of a single computable real number in his landmark 1936 paper [Tur39]: a real number is computable if its decimal expansion can be computed in the discrete sense (i.e., output by some Turing machine). But he did not go on to define the notion of computable real function. One of the big successes of discrete computability theory is the insolvability results: especially the solution of Hilbert's 10th problem (see [Mat93]). The theorem states that there is no procedure (e.g., no Turing machine) which always correctly determines whether a given Diophantine equation has a solution. The result is convincing because of general acceptance of the Church-Thesis.

Büchi and others initiated the study of ω-automata and Büchi machines, involving automata and Turing machine computations of length ω which

accept or reject infinite input. Gerald Sacks and many others (see [Sac90]) founded the field of higher recursion theory, including α-recursion and E-recursion, a huge body of work analyzing computation on infinite objects. Lenore Blum, Michael Shub and Steve Smale have presented a model of computation on the real numbers, the so called BSS machines, a kind of flowchart machine where the basic units of computation consist of real numbers, in a full glorious precision [BSS89]. Apart from all this mathematical work, Joel David Hamkins and Andy Lewis have been proposed a new model of infinitary computation called infinite time Turing machines [HL00]. There is no suggestion at all of how such devices might be engineered or even conceived in a physical theory. And, thus, these devices are considered in a logic context only. However, this model offers the strong computational power of higher recursion theory while remaining very close in spirit to the computability concept of Turing machines. Notice that the BSS machines were the original inspiration for infinite time Turing machines.[3] In another direction, the theory of higher recursion provides a model of infinitary computation by setting a very general theoretical context for recursion on infinite objects, and one should expect many parallels between it and the theory of infinite time Turing machines.

José Félix Costa and Jerzy Mycka and others have been working towards the recursive definition of computational classes of functions over $\mathbb{R}$. The first presentation of such theory, analogous to Kleene's classical theory of recursive functions over $\mathbb{N}$, was attempted by Christopher Moore in [Moo96]. Roughly speaking, these real recursive functions are generated by constants, projections, composition, minimization, and also by a fundamental operator called differential recursion instead of the classical recursive operator. In [MC04a], an infinite limit operator is added instead of minimalisation in [Moo96].[4] In [CMC02], it is shown that a linear form of the differential recursion scheme gives rise to an analog characterization of (Kálmar's) elementary functions [Kal43] and to an analog characterization of Grzegorczyk hierarchy

[3] Jeffrey. Kidder and Joel David Hamkins heard Lenore Blum's lectures for the Berkeley Logic Colloquium in 1989, and had the idea to generalize the Turing machine concept in a different direction: to infinite time rather then infinite precision [Ham01].

[4] See [MC04a] for an exhaustive discussion about the role of minimization and infinite limits in real recursion theory.

[Grz55]. In [CMC00] it is shown that the GPAC is not closed under iteration and that a subclass of real recursive functions coincides with the class of GPAC-computable functions. And, finally, it was shown how to capture higher computational classes through the limit operator. Recently, Olivier Bournez and Emmanuel Hainry at INRIA showed, in [BH05], that a specific kind of limit together with differential recursion make the class of real recursive functions an exact extension to the real numbers (we can say in the sense of Computable Analysis [Wei00]) of the classical recursive functions.

Chapter 2

Computation over the reals: the general framework

In this chapter, we provide the essential background about real recursion theory following the proposal that have been developed by José Félix Costa, Manuel Campagnolo, Daniel Graça and Jerzy Mycka, and, more recently, by Olivier Bournez and Emmanuel Hainry, which has its roots on the seminal work of Christopher Moore. More concretely, we review recursion theory over natural numbers, following the complete survey given by [Odi89], and over real numbers, following the recent work introduced in [MC04a], emphasing the η-hierarchy of real recursive functions.

The first presentation of recursive functions over $\mathbb{R}$, analogous to Kleene's classical theory of recursive functions over $\mathbb{N}$, was attempted by Christopher Moore in the mid 1990's [Moo96]. Such functions over $\mathbb{R}$ are generated by a differential recursion operator instead of the classical recursion operator over $\mathbb{N}$. More recently, José Félix Costa and Jerzy Mycka introduced another fundamental operator called infinite limit [MC04a]. Later on, Olivier Bournez and Emmanuel Hainry showed in [BH05] that a specific kind of limit operator together with differential recursion makes the class of real recursive functions an exact extension to real numbers, in the sense of Computable Analysis [Wei00], of classical recursive functions.

2.1 Recursion theory over $\mathbb{N}$

The classical recursion theory, initiated by Kleene in the mid 1930s as an alternative approach to discrete computation over discrete time, is the study of (partial) recursive functions over $\mathbb{N}$. In such theory, an inductive approach is taken by starting from initial functions, corresponding to $\mathcal{O}$ (the zero function), $\mathcal{S}$ (the successor function) and $\mathcal{I}^n$ ($n > 0$) (the projection functions), and by successively building up new functions using composition, recurrence and minimalization operators. Formally,

Definition 2.1.1. The *class of partial recursive functions* $REC(\mathbb{N})$ is generated from the initial recursive functions $\mathcal{O}(x) = 0$ of arity 1, $\mathcal{S}(x) = x + 1$ of arity 1 and, for every $n > 0$, $\mathcal{I}_i^n(x_1, \ldots, x_n) = x_i$ of arity n, for every $1 \leq i \leq n$, and by the following operators:

- Composition: If $f_1, \ldots, f_m$ are partial recursive functions of arity n, and g is a partial recursive function of arity m, then

$$h(x_1, \ldots, x_n) = g(f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$$

  is a partial recursive function of arity n.

- Recurrence: If f is a partial recursive function of arity n, and g is a partial recursive function of arity $n + 2$, then h is defined as follows:

$$\begin{aligned} h(x_1, \ldots, x_n, 0) &= f(x_1, \ldots, x_n) \\ h(x_1, \ldots, x_n, y + 1) &= g(x_1, \ldots, x_n, y, h(x_1, \ldots, x_n, y)) \end{aligned}$$

  is a partial recursive function of arity $n + 1$.

- Mimimalization: If f is a partial recursive function of arity $n + 1$, then

$$\mu y(f(x_1, \ldots, x_n, y) = 0) = \begin{cases} least\ y\ such\ that & \\ f(x_1, \ldots, x_n, z) \downarrow, & \\ for\ every\ z \leq y,\ and & \\ f(x_1, \ldots, x_n, y) = 0, & if\ y\ exists \\ \perp & otherwise \end{cases}$$

  is a partial recursive function of arity n. □

Since the minimalization operator searches arbitrary large values of y to find at least one value such that $f(x_1, \ldots, x_n, y) = 0$, it corresponds to the 'while' statement; there may be no such y and, thus, the 'while' might never halt. Then h is undefined on that value $(x_1, \ldots, x_n)$, denoted by $\perp$, and, then, it is a partial function. The functions that can be generated with composition, primitive recursion and minimalization are called partial recursive; if a function f is total, i.e., defined for every $(x_1, \ldots, x_n)$, then f is primitive recursive. The class $REC(\mathbb{N})$ turn out to correspond exactly to many other definitions of computability, including Turing machines, which is considered a deep and universal definition of computability.

The iteration operation is also fundamental in the classical theory of computation. So,

Definition 2.1.2. A function f of arity 2 is defined by *iteration* from a function g of arity 1 if $f(x, n) = g^n(x)$ where $g^n(x) = g(g^{n-1}(x))$ (by convention $g^0(x) = x$). □

And, the following result makes it clear:

Proposition 2.1.1. *The class of primitive recursive functions is the smallest class of functions such that:*

1. *contains $\mathcal{O}$, $\mathcal{S}$, and $\mathcal{I}^n$ functions, together with coding and decoding functions*
2. *is closed under composition;*
3. *is closed under iteration.*

Proof. See Proposition I.5.10 (pp. 72-73) in [Odi89]. □

The result just presented can be previously improved. First of all, M. D. Gladstone in [Gla67, Gla71] shows that the introduction of new initial functions can be avoided: the class of primitive recursive functions is the smallest class containing the initial functions, and it is closed under composition and iteration. Second the iteration schema can be further weakened in the following schema of *pure iteration*: $f(n) = g^{(n)}(0)$. Some initial functions are needed here, since by composition and pure iteration we never get, from the initial functions, any function of arity 2. Possible choices of initial functions that generate the primitive recursive functions by pure iteration are in [Gla71]. R.

M. Robinson, in [Rob47], also show that the pure iteration schema is enough to generate the unary primitive recursive functions by composition, starting from two (but not from only one) appropriate unary functions. Thus we do not need to use nonunary functions to get any unary primitive recursive function.

2.2 Recursion theory over $\mathbb{R}$

In [Moo96], it was defined a set of (vector-valued) functions over $\mathbb{R}^n$, called $\mathbb{R}$-recursive functions, analogously to the approach taken by Kleene in $\mathbb{N}$ [Kle55]: the discrete recursion operator is replaced by a continuous integration operator and, thus, the set of $\mathbb{R}$-recursive functions generates a continuous time computation model. More recently, Jerzy Mycka and José Félix Costa present, in [MC04a], several changes and exhaustive comments about the original proposal in [Moo96]. The main change provided by them is the replacing of Moore's μ-operator, a counterpart of the classical minimalization operator, by an infinite limit operator. This is a powerful idea to implement the levels of the arithmetical hierarchy into subclasses of the real recursive vector functions. It is also important to notice that the minimalisaion operator can be derived from infinite limits as we can see in [Myc03], although the contrary might not be strictly true.

Recall that, in [BH05], it is also introduced a kind of infinite limit operator in order to capture, together with composition and differential recursion, exactly the whole class of classical recursive functions. But, here, we prefer the concept of primitive limit as being operative and allowing to reason about computability and complexity in Analysis in a natural way.

So, we are going to present the class of real recursive vector functions following the approach found in [MC04a]. But, first, it is important to explain why vectorial notation is used to generate the class of real recursive functions. The main reason is related with the functions sin and cos, that plays a major role as clock functions, which are solutions of a second-order differential equation of the form: $\partial_y^2\, h(y) + h(y) = 0$, i.e.

$$\begin{pmatrix} \partial_y\, h(y) \\ \partial_y\, g(y) \end{pmatrix} = \begin{pmatrix} g(y) \\ -h(y) \end{pmatrix}.$$

Definition 2.2.1. The *class of real recursive vector functions* $REC(\mathbb{R})$ [1] is generated from the real recursive scalars $0,\ 1,\ -1$ and the real recursive projections $I_n^i(x_1,\ldots,x_n) = x_i,\ 1 \leq i \leq n,\ n > 0$ and by the following operators:

- Composition: if f is a real recursive vector function with n k-ary components and g is a real recursive vector function with k m-ary components, then the vector function with n m-ary components, $1 \leq i \leq n$,

$$\lambda x_1 \ldots \lambda x_m.\ f_i(g_1(x_1,\ldots,x_m),\ldots,g_k(x_1,\ldots,x_m))$$

  is real recursive.

- Differential recursion: if f is a real recursive vector function with n k-ary components and g is a real recursive vector function with n $(k+n+1)$-ary components, then the vector function h of n $(k+1)$-ary components which is the solution of the Cauchy problem, $1 \leq i \leq n$,

$$h_i(x_1,\ldots,x_k,0) = f_i(x_1,\ldots,x_k),$$

$$\partial_y h_i(x_1,\ldots,x_k,y) = g_i(x_1,\ldots,x_k,y,h_1(x_1,\ldots,x_k,y),\ldots,h_n(x_1,\ldots,x_k,y))$$

  is real recursive whenever h is of the class C^1 on the largest interval containing 0 in which a unique solution exists.[2]

- Infinite limits: if f is a real recursive vector function with n $(k+1)$-ary components, then the vector functions h, h^i, h^s with n k-ary components, $1 \leq i \leq n$,

$$h_i(x_1,\ldots,x_k) = \lim_{y\to\infty} f_i(x_1,\ldots,x_k,y),$$

[1] Hereafter, we abbreviate real recursive vector functions by real recursive functions.

[2] Suppose $g(x,y,z)$ is a continuous function in a rectangle of the form $\{\langle y,z\rangle : a < y < b \text{ and } c < z < d\}$; if $\langle 0, f(x)\rangle$ is a point of this rectangle, then there exists an $\epsilon > 0$ and a function $h(x,y)$ defined for $-\epsilon < y < \epsilon$ that solves the Cauchy problem

$$\partial_y h(x,y) = g(x,y,h(x,y)),$$

with $h(x,0) = f(x)$, for all x such that both f and g are defined (EXISTENCE THEOREM). Suppose $g(x,y,z)$ and $\partial_z g(x,y,z)$ are continuous functions in a rectangle of the form $\{\langle y,z\rangle : a < y < b \text{ and } c < z < d\}$. If $\langle 0, f(x)\rangle$ is a point of this rectangle and if $h_1(x,y)$ and $h_2(x,y)$ are two functions that solve the Cauchy problem for all $-\epsilon < y < \epsilon$, than $h_1(x,y) = h_2(x,y)$ (UNIQUENESS THEOREM).

$$h_i^i(x_1,\ldots,x_k) = \liminf_{y\to\infty} f_i(x_1,\ldots,x_k,y),$$
$$h_i^s(x_1,\ldots,x_k) = \limsup_{y\to\infty} f_i(x_1,\ldots,x_k,y)$$

are all real recursive.

- Assembling and designating components: (a) arbitrary real recursive vector functions can be defined by assembling scalar real recursive function components into a vector function; (b) if f is a real recursive vector function, then each of its components is a real recursive scalar function.
- Real recursive numbers: arbitrary real recursive scalar functions of arity 0 are called real recursive numbers. □

In order to understand the above definition and to see how simple it is the construction of a real recursive function without infinite limits, we provide some interesting examples, that were already introduced in [MC04a].

Example 2.2.1. Constant functions 0_n, 1_n, -1_n which are n-ary can be derived from unary constant functions by means of projections. For example $1_n(x_1,\ldots,x_n) = 1$ can be defined as $1_1(I_n^1(x_1,\ldots,x_n)) = 1$. Constant functions of arity one can be derived by differential recursion: $\mathbf{0}(0) = 0$, $\partial_y\mathbf{0}(y) = I_2^2(y,\mathbf{0}(y))$; $u(0) = c, \partial_y u(y) = \mathbf{0}(I_2^1(y,u(y)))$, where $c = 1, -1$.

The functions $+$, $\times$, $-$, exp, $\sin$, $\cos$, $\lambda x.\ \frac{1}{x}$, $/$, $\log$ and $\lambda xy.\ x^y$ are real recursive vector functions. Let us define $+(x,0) = I_1^1(x) = x$, $\partial_y + (x,y) = 1_3(x,y,+(x,y))$. Analogously, $\times(x,0) = \mathbf{0}_1(x)$, $\partial_y \times (x,y) = I_3^1(x,y,\times(x,y))$, hence we have by a composition $-(x,y) = +(x,\times(-1,y))$. The exponentiation can be defined as $exp(0) = 1$, $\partial_y exp(y) = I_2^2(y,exp(y))$. Furthermore, the vector $(\sin(x),\cos(x))$ and its components can be defined by such differential recursion:

$$\begin{pmatrix}\sin\\ \cos\end{pmatrix}(0) = \begin{pmatrix}0\\ 1\end{pmatrix},\ \partial_y\begin{pmatrix}\sin\\ \cos\end{pmatrix}(y) = \begin{pmatrix}I_3^3\\ -I_3^2\end{pmatrix}(y,\sin y,\cos y).$$

Now for $\lambda x.\ \frac{1}{x}$, we define $h(x) = \frac{1}{x+1}$ in the following way: $h(0) = 1$, $\partial_x h(x) = \times(-1,\times(h(x),h(x)))$ (h is defined in the interval $(-1,\infty)$), and then we compose h with $\lambda x.\ x-1$. The division is simply a composition of $\times$ and $\lambda x.\ \frac{1}{x}$ (with the domain equal to $(0,\infty)$, but we can extend the division to the negative numbers via a definition by cases). In the case of $\log$ we start with

the definition of $\lambda x.\ \log(x+1)$ by $\log(1)=0$, $\partial_x \log(x+1) = \frac{1}{I_2^1(x,\log(x+1))+1}$, to finish with a shift of the argument. Next, $x^0 = 1_1(x), \partial_y x^y = g(x,y,x^y) = \log(x)\cdot x^y$. □

For differential recursion, the domain is restricted to an interval of continuity and, thus, preserving the analiticity of functions that can be generated by such differential schema. For example, using differential recursion we can not define functions such as $\lambda x.|x|$. It is excluded the possibility of operations on undefined functions: functions are strict in the meaning that for undefined arguments they are also undefined.

To obtain more interesting functions (e.g. the several forms of η-function in Definition 2.3.5) the addition of the operators of infinite limits are required. Let us point out that introducing infinite limits gets discontinuous functions.

Example 2.2.2. The Kronecker δ function, the signum function, and absolute value are real recursive functions. The Heaviside Θ function, the binary maximum $\max$, the square-wave function s, the function p such that $p(x)=1$ for $x\in[2n,2n+1)$ and $p(x)=0$ for $x\in[2n+1,2n+2)$, and the floor function are all real recursive too. □

It is sufficient to take the following definitions: if $\delta(0)=1$ and for all $x\neq 0$ we have $\delta(x)=0$, then let us define $\delta(x) = \liminf_{y\to\infty}(\frac{1}{1+x^2})^y$. From the function $\lambda xy.\ \frac{2}{1+e^{-xy}}-1$, we obtain

$$sgn(x) = \liminf_{y\to\infty}\ \frac{2}{1+e^{-xy}} - 1 = \begin{cases} 1 & if \quad x>0 \\ 0 & if \quad x=0 \\ -1 & if \quad x<0 \end{cases}, \text{ and } |x| = sgn(x)x.$$

Let $\Theta(x) = (sgn(x)+\delta(x)+1)/2$, then we have $\max(x,y) = y+(x-y)\Theta(x-y)$ and $s(x)=\Theta(\sin(\pi x))$.
The function p can be given by $\lambda x.\ s(x)(1-\delta(\sin(\frac{(x-1)\pi}{2})))$. Finally the floor function has the definition below

$$\lfloor x\rfloor = w(x)p(2x) + w(x-\frac{1}{2})(1-p(2x)),$$

where $w(x)=j$ if $x\in[j,j+\frac{1}{2})$. Such function w can be defined by the differential recursion: $w(0)=0,\ \partial_x w(x) = 4\sin^2(2\pi x)\Theta(-\sin(2\pi x))$. □

In some examples above we can use in constructions the predicate of equality $eq = \lambda xy.\ \delta(x-y)$. Sometimes we will use Θ to control whether

points are in a given interval. Then for $x \in [a, \infty)$ we have the characteristic function $\Theta(x-a)$ and for $x \in [a, b]$ we can define $\Theta_{[a,b]}(x) = \Theta(x-a)\Theta(b-x)$.

Let us add that we can find real recursive numbers (computable reals) as values of real recursive functions of arity one for, let us say, an argument equal to 0. Of course the argument can be changed to a real recursive number t by a composition of a given real recursive function with $\lambda x.\ x + t$. In this sense e and π are computable reals: $e = exp(1)$, $\pi = 4\arctan(1)$, where $\arctan(0) = 0,\ \partial y \arctan(x) = \frac{1}{1+x^2}$. Also Euler's constant

$$\gamma = \lim_{n\to\infty}((\sum_{k=1}^{n} \tfrac{1}{k}) - \log(n))$$

is a computable real number because it can be established by real recursive expression

$$\gamma = -\lim_{y\to\infty} \int_0^y e^{-x}\ \log(x)\ dx.$$

A (unique) solution for the differential recursion, in the Definition 2.2.1, is guaranteed including in the definition the existence and uniqueness theorem on the largest interval containing 0. More recently, it has been discussing a definition for a solution for such differential recursion schema in order to say, precisely, what is a solution for it. For example, in differential recursion of Definition 2.2.1, it is not imposed that functions f_i and g_i, for $1 \leq i \leq n$, are of class C^1. Several examples have been taking to motivate the adequacy of the definition of a solution to a system of differential equations.

Example 2.2.3. Consider the following differential schema

$$h(0) = 2 - \sqrt{3},\ \partial_y h(y) = \frac{y}{h(y) - 2},$$

where the functions f and g involved are the constant $2 - \sqrt{3}$ and $\lambda yz.\ \frac{y}{z-2}$, respectively. Although g is not C^1 in all components, the largest solution is $h(y) = 2 - \sqrt{y^2 + 3}$ that is defined in $\mathbb{R}$. □

The literature about differential equations (e.g. [Arn92]) say that *a solution... is a function of the independent variable that, when substituted into the equation as the dependent variable, satisfies the equation for all values of the independent variable. That is, a function $h(y)$ is a solution if it satisfies the following standard form of differential recursion*

$$\partial_y h(y) = g(y, h(y))$$

for every y in $\mathbb{R}$. But in many cases, there is a unique function h in C^1 that satisfies the above equation for all y where g is defined, although g has a countable number of discontinuities in $\mathbb{R}$. In this case, we can adopt h as the desired (generalized) solution of the above differential recursion schema.

The above discussion take us to say, formally, what we intend by a solution of a system of differential equations defined by the differential recursive schema of Definition 2.2.1. So,

Definition 2.2.2. A *solution* for a system of differential equations

$$\partial_y h_1(x_1, \ldots, x_k, y) = g_1(x_1, \ldots, x_k, y, h_1(x_1, \ldots, x_k, y), \ldots, h_n(x_1, \ldots, x_k, y)),$$

$$\vdots$$

$$\partial_y h_n(x_1, \ldots, x_k, y) = g_n(x_1, \ldots, x_k, y, h_1(x_1, \ldots, x_k, y), \ldots, h_n(x_1, \ldots, x_k, y)),$$

giving the initial conditions

$$h_1(x_1, \ldots, x_k, 0) = f_1(x_1, \ldots, x_k)$$

$$\vdots$$

$$h_n(x_1, \ldots, x_k, 0) = f_n(x_1, \ldots, x_k),$$

is a vector function $\hat{h} : \mathbb{R}^{k+1} \to \mathbb{R}^n$ such that:

- a unique solution h to the system of differential equations exists in some open interval I containing 0;
- the vector function $\hat{h}$ satisfies the equations in $J \supseteq I$ such that J is an open interval up to a countable number of non-Zeno discontinuities [3] in the sense that, for every $y \in J$, $\hat{h}_i(x_1, \ldots, x_k, y)$ and

  $$g_i(x_1, \ldots, x_k, y, \hat{h}_1(x_1, \ldots, x_k, y), \ldots, \hat{h}_n(x_1, \ldots, x_k, y))$$

  are both defined, for all $1 \leq i \leq n$,

  $$\partial_y \hat{h}_i(x_1, \ldots, x_k, y)$$

  is defined and it holds that

[3] It means that for each finite open interval there exist only a finite number discontinuities.

$\partial_y \hat{h}_i(x_1, \ldots, x_k, y) = g_i(x_1, \ldots, x_k, y, \hat{h}_1(x_1, \ldots, x_k, y), \ldots, \hat{h}_n(x_1, \ldots, x_k, y))$;

- the vector functions h and $\hat{h}$ coincide on I;
- the vector function $\hat{h}$ is the unique continuous extension of h. □

The vector function $\hat{h}$ is called *generalized solution* of the system of differential equations if it is the maximal solution according with the previous items. When dealing with our inductive definitions, we will work with this definition of a solution to the differential recursion schema.

Example 2.2.4. Consider the scheme $h(0) = 0$, $\partial_y h(y) = \frac{1}{\sec(y)}$. The solution is $h(y) = sin(y)$, e.g, for $y \in (-\frac{\pi}{2}, \frac{\pi}{2})$. But, if we ask for the largest *generalized* solution in C^1 or even in C^0, the answer is $h(y) = sin(y)$, despite the fact that our differential relation will only be satisfied outside a countable number of points. □

A particularly interesting real recursive vector function in $REC(\mathbb{R})$ is the iteration function (that we take the original result from [Moo96]). There are, since the work of Branicky (see [Bra95]), many ways to simulate in continuous time the iteration of a discrete time function, particularly the simulation of a Turing machine given in [Moo96] or, more recently, the one given in [MC04a].

Proposition 2.2.1. *If f is a real recursive scalar total function of arity n, then the iteration of f, F, is a real recursive scalar function of arity $(n+1)$, such that, for all $y \geq 0$, $F(x_1, \ldots, x_n, y) = f^{\lfloor |y| \rfloor}(x_1, \ldots, x_n)$*

Proof. See Proposition 11 (pp. 13-14) in [Moo96]. □

In [CMC00], it is shown that, for every $k > 1$, the class $\mathcal{G} + \theta_k$ is closed under iteration but $\mathcal{G}$ is not, where $\mathcal{G}$ is the class of primitive $\mathbb{R}$-recursive functions whose derivatives are bounded on the interval on which they are defined and it is also equivalent to GPAC computable functions.

2.3 The η-hierarchy

In [Moo96], it is established a μ-hierarchy to stratify the class of $\mathbb{R}$-recursive functions according to the number of nested minimalizations, called the μ-number, which is defined inductively with respect to a given variable. For

any $\mathbb{R}$-recursive function f, let $M(f) = max_i\ M_{x_i}(s)$, minimized over all expressions s that define f and, thus, the μ-hierarchy is defined as sets

$$M_j = \{f : M(f) \leq j\}.$$

So, the set of $\mathbb{R}$-recursive functions is defined by $\cup_j\ M_j$.

Here, we consider limits instead of minimalizations and use the notion of syntactic n-ary descriptions for real recursive functions, following the approach found in [MC04a]. As we will see, the collection of descriptors of a given real recursive function is defined inductively giving some atomic and operator descriptors for the existent sorts of real recursive functions.

The η-hierarchy describe the level of nesting limits in the definition of a given real recursive function which is a measure of the difficulty of such function given in terms of its degree of (dis)continuity. More important, it is the η-operator which give us, not only the tool to control the domain and singularities of functions, but also the appropriated tool to simulate Turing machines with real recursive functions.

Because each real function has an infinite but a countable number of expressions that can define it, hence there is also an infinite but a countable number of descriptors for a given real recursive function.

Next, we say how we can obtain, inductively, the set of descriptors of a given real recursive function f which is denoted by $< f >$.

Definition 2.3.1. The *collection of descriptors* of a real recursive function is inductively defined as follows:

- i_n^j is a n-ary description of I_n^j, $1 \leq j \leq n \in \mathbb{N}$;
- for every $n \in \mathbb{N}$,
  - 1_n is a n-ary description of $\lambda x_1 \ldots x_n.1$, for all $(x_1, \ldots, x_n) \in \mathbb{R}^n$;
  - $\overline{1}_n$ is a n-ary description of $\lambda x_1 \ldots x_n. -1$, for all $(x_1, \ldots, x_n) \in \mathbb{R}^n$;
  - 0_n is a n-ary description of $\lambda x_1 \ldots x_n.0$, for all $(x_1, \ldots, x_n) \in \mathbb{R}^n$;
- if $< h >=< h_1, \ldots, h_m >$ is a k-ary description of the real recursive function h and $< g >=< g_1, \ldots, g_k >$ is a n-ary description of the real recursive function g, then $c(< h >, < g >)$ is a n-ary description of the composition of h and g;

- if $< h >=< h_1, \ldots, h_m >$ is a k-ary description of the real recursive function h and $< g >=< g_1, \ldots, g_k >$ is a $(k+n+1)$-ary description of the real recursive function g, then $dr(< h >, < g >)$ is a $(k+1)$-ary description of the function defined as differential recursion as in Definition 2.2.1;

- if $< h >=< h_1, \ldots, h_m >$ is a $(n+1)$-ary description of the real recursive function h, then $l(< h >)$, $li(< h >)$, $ls(< h >)$ is a n-ary description of an appropriate infinite limit (respectively $\lim$, $\liminf$ and $\limsup$) of h defined as infinite limits as in Definition 2.2.1;

- if $< f_1 >, \ldots, < f_m >$ are n-ary descriptions of real recursive k-ary scalars $f_1, \ldots, f_m$, then $v(< f_1 >, \ldots, < f_m >)$ is a k-ary description of the real recursive function $f = (f_1, \ldots, f_m)$. □

Example 2.3.1. We construct the description of real recursive function $\lambda x.\frac{1}{x}$. From Example 2.2.1, recall that we define $h(x) = \frac{1}{x+1}$ in the following way: $h(0) = 1$, $\partial_x h(x) = \times(-1, \times(h(x), h(x)))$ (h is defined in the interval $(-1, \infty)$), and then we compose h with $\lambda x.\, x-1$, in order to obtain $\lambda x.\frac{1}{x}$. So, we need to have the description for $\lambda xy.x+y$, which is $dr(i_1^1, 1_3)$, and for $\lambda xy.x-1$, which is $c(dr(i_1^1, 1_3), v(i_1^1, \overline{1}_1))$, and, finally, for $\lambda xy.xy$, which is $dr(0_1, i_3^1)$. Then, $\lambda x.x^2$ has the description $c(dr(0_1, i_3^1), v(i_1^1, i_1^1))$, and $\lambda xz. - z^2$ has the description $c(dr(0_1, i_3^1), v(\overline{1}_2, c(dr(0_1, i_3^1), v(i_2^2, i_2^2))))$. And, finally, the description of $\lambda x.\frac{1}{x}$ is $c(dr(1_0, < \lambda xz. - z^2 >), < \lambda x.x - 1 >)$. □

The η-number for a description of some real recursive function is a way to count nested limits in inductive descriptions given just above. Formally,

Definition 2.3.2. For a given n-ary *description* s of a real recursive vector function f, let $E_i^k(s)$ (with respect to the i-th variable of the k-component) be defined inductively as follows:

1. $E_i^1(0_n) = E_i^1(\overline{1}_n) = E_i^1(1_n) = 0$;

2. $E_i^m(c(< h >, < g >)) = max_{1 \le j \le k}\,(E_j^m(< h >) + E_i^j(< g_j >))$, where h is a n components of k-ary vector and g is a k-components m-ary vector;

3. for the differential recursion, we have

(a) for $i \leq k$,

$$E_i^j(dr(< f >, < g >)) =$$
$$max\ (E_i^1(< f_1 >), \ldots, E_i^1(< f_n >)), E_i^1(< g_1 >), \ldots,$$
$$E_i^1(< g_n >), E_{k+1}^1(< g_1 >), \ldots, E_{k+1}^1(< g_n >)))$$

(b) for $i = k + 1$

$$E_i^j(dr(< f >, < g >)) =$$
$$max_{1 \leq m \leq n}(max(E_{k+m+1}^1(< g_1 >), \ldots, E_{k+m+1}^1(< g_n >)))$$

where f is a n components k-ary vector and g is a n components $(k + n + 1)$-ary vector;

4. for infinite limits, we have

$$E_i^k(l(< h >)) = E_i^k(li(< h >)) = E_i^k(ls(< h >)) =$$
$$max(E_i^k(< h >), E_{n+1}^k(< h >)) + 1$$

where h is a k components $(n + 1)$-ary vector. □

For n-ary description $< h >$ of m components, the η-number for $< h >$,

$$E(< h >) = max_k max_i E_i^k(< h >),$$

for every $1 \leq i \leq n$ and $1 \leq k \leq m$. Thus, the η-number for a real recursive function can be defined as follows:

Definition 2.3.3. Let f be a real recursive function. Then $\eta(f)$ is the *minimum of* $E(< f >)$ for every description of f. □

We know that the set of real valued functions is uncountable, and that the subset of vector functions mapping integers to integers is uncountable too. However the set of possible descriptions of real recursive functions is countable. Then we conclude immediately the two following facts: (a) there are uncountably many non real recursive functions and (b) there are uncountably many non real recursive functions that map integers to integers.[4]

Having an η-number for each real recursive function, we can also define an η-hierarchy as the measure of the difficulty of real recursive functions.

[4] Also, we can state that there are uncountably many non real recursive vector functions that map real recursive numbers to real recursive numbers.

Definition 2.3.4. The *η-hierarchy* is an $\mathbb{N}$-indexed family of sets

$$\mathrm{H}_j = \{f : \eta(f) \leq j\}.$$

□

If $f \in \mathrm{H}_j$, then j nested limits is used to define f. Here it is the way of other equivalent definition: if f is a real recursive function, then $E(f) = j$ if at most j nested η operations are necessary to create f_{total} such that f_{total} is defined everywhere and if $f(x_1, \ldots, x_n)$ is defined, then

$$f_{total}(x_1, \ldots, x_n) = f(x_1, \ldots, x_n).$$

As an example, we can classify some of the real recursive functions already introduced and, additionally, classify some important functions of mathematics, such as Bessel functions, Euler function, Riemann zeta function (see [MC04a]), that can be also expressed in terms of real recursiveness according to Definition 2.2.1.

For a proper analysis of real recursive functions it is important to control the domain and singularities of these functions.

Definition 2.3.5. For every function $f : \mathbb{R}^{n+1} \to \mathbb{R}$, let

$$\eta_y f(x_1, \ldots, x_n, y) = \begin{cases} 1 & \text{if } \lim_{y\to\infty} f(x_1, \ldots, x_n, y) \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$

$$\eta^i_y f(x_1, \ldots, x_n, y) = \begin{cases} 1 & \text{if } \liminf_{y\to\infty} f(x_1, \ldots, x_n, y) \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$

$$\eta^s_y f(x_1, \ldots, x_n, y) = \begin{cases} 1 & \text{if } \limsup_{y\to\infty} f(x_1, \ldots, x_n, y) \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$

□

The $\eta_y f(x_1, \ldots, x_n, y)$ defined just above is a characteristic function for the set of $(x_1, \ldots, x_n)$ such that $lim_{y\to\infty} f(x_1, \ldots, x_n, y)$ is well defined (without singularities). Analogously, $\eta^i_y f(x_1, \ldots, x_n, y)$ and $\eta^s_y f(x_1, \ldots, x_n, y)$ play the same role for $\liminf_{y\to\infty} f(x_1, \ldots, x_n, y)$ and $\limsup_{y\to\infty} f(x_1, \ldots, x_n, y)$, respectively.

The problem arises whether such operators are real recursive. If the answer is to the question, whether we can define them by standard operators,

is yes, we may patch any partial function to total one. For example, let f be a total real recursive function and

$$F_{total}(x_1, \ldots, x_n) = \lim_{y \to \infty}(\eta_y f(x_1, \ldots, x_n, y)) f(x_1, \ldots, x_n, y)$$

and $F(x_1, \ldots, x_n, y) = \lim_{y \to \infty} f(x_1, \ldots, x_n, y)$. So, F_{total} is total because if $F(x_1, \ldots, x_n)$ is defined, then $F_{total}(x_1, \ldots, x_n) = F(x_1, \ldots, x_n)$; otherwise

$$F_{total}(x_1, \ldots, x_n) = 0.$$

The class of real recursive functions is closed under η, η^i and η^s if the functions obtained by these operators from real recursive functions can be constructed as real recursive functions. Thus, $REC(\mathbb{R})$ is closed under η, η^i and η^s.

Proposition 2.3.1. *If f is a total real recursive vector function, then ηf, $\eta^i f$ and $\eta^s f$ are total real recursive vector functions .*

Proof. See Proposition 17 (pp. 16-17) in [MC04a]. □

Recently, in [Myc05], it was presented another application of η-hierarchy that established a connection among such hierarchy, Baire classes and effective Baire classes. It has been served to identify some problems of analog computation with descriptive set theory [Mos80].

Chapter 3

The power of finite state machines in the general framework

In a broad sense, hybrid computation includes all computing techniques combining some of the features of digital computations with some of the features of analog computations. Recall that digital computation has been dominated by the unified work of Turing since mid 1930s, while analog computation has not yet experienced that unification. Consequently, there is also a lack of consensus about the most appropriated formal characterization for hybrid computation. But it is well known that hybrid computation occurs at the crossroad of several scientific directions: it is based on several ideas coming from computer science and mathematics and gathered in hybrid systems.

More concretely, very restricted classes of piecewise signals, namely the piecewise constant signals (see [Rab03, AMP95]) and the piecewise linear signals (e.g. [ACHH93]), have been extensively studied not only by hybrid systems community but also by dynamical systems community (e.g. [KCG94]). But not much attention is placed in this issue by continuous computation community (e.g. [Orp97]) when such signals are considered piecewise functions.

Here we intend to establish a relationship between recursive functions over the reals (see [MC04a, CMC00, Moo96]), the side of analog (continuous) computation, and a certain kind of finite state automata over continuous time

(see [Rab03, Hen96]), whose computational power is the recognition of periodic piecewise linear signals, the side of discrete computation. So, we are looking for a computational model, based on finite state automata, which are able to recognize piecewise infinite signals.

Recall that, periodic functions, in particular the periodic piecewise smooth functions, are the most suitable functions in the Fourier Theory which has a vast range of applications (see [Bee03, Vre03]). So, the intended relationship must be constructed having in mind the fundamental of Fourier series in order to bring these periodic functions to the range of real recursive functions.

We will see that the square wave and sawtooth wave functions, that were taken originally in [Moo96] to show that iteration is indeed a real recursive function, have very simple Fourier series. Moreover, it will be introduced what we need to know about Fourier series in order to show that Fourier series are indeed real recursive scalar functions, using real recursive infinite limits in [MC04a]. And, finally, we will show that the computational power of a special kind of finite state automata over continuous time found in [Rab03] only recognize periodic real recursive functions.

3.1 Periodic functions and Fourier series

The classical theory of Fourier series and integrals, as well as Laplace transforms, is of great importance for physical and technical applications, since it enable us to reason about many periodic phenomena in nature, and, then, periodic functions play the main role to model them. In what follows, we will carry only Fourier series, and a particular type of periodic functions, to real recursive function theory. All background about Fourier analysis can be found in [Bee03], which will be our source of notation and terminology.

Definition 3.1.1. Let f be a real function.[1] We say that f is *periodic* with period $z > 0$ if, for every $x \in R$, $f(x + z) = f(x)$. □

Several examples of well known periodic functions exist (e.g. sinusoidal functions) but in order to illustrate the construction of Fourier series for some of them, we prefer to consider the periodic functions occurring in [Moo96]

[1] A real function is a function whose domain and co-domain is $\mathbb{R}$. Hereafter, we assume only real functions

(see Proposition 11), namely the square wave function s and the sawtooth wave function r.

Example 3.1.1. For some $m \in \mathbb{N}$.

- **Square wave function**.

$$s(x) = \begin{cases} 1 & if \quad x \in [2m, 2m+1] \\ 0 & if \quad x \in (2m+1, 2m+2) \end{cases}$$

- **Sawtooth wave function**.

$$r(x) = \begin{cases} x - 2m & if \quad x \in [2m, 2m+1) \\ -x + 2m + 2 & if \quad x \in [2m+1, 2m+2) \end{cases}$$

□

In order to establish the Fourier series for an arbitrary periodic function we need first to calculate the so called Fourier coefficients, say in $[0, z]$, as follows:

Definition 3.1.2. If f is a periodic function with period z, then the *Fourier coefficients* a_n and b_n of $f(x)$, if they exist, are given, for every $n \in \mathbb{N}$, by

$$a_n = \frac{2}{T} \int_0^z f(x) cos(\frac{2\pi nx}{z}) dx$$

and

$$b_n = \frac{2}{T} \int_0^z f(x) sin(\frac{2\pi nx}{z}) dx.$$

□

Using the Fourier coefficients, defined just above, we can define the Fourier series associated with a given periodic function.

Definition 3.1.3. If a_n and b_n are the Fourier coefficients of the periodic function f with period z, then the *Fourier series* of $f(x)$ is defined by

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n cos(\frac{2\pi nx}{z}) + b_n sin(\frac{2\pi nx}{z})).$$

□

In what follows, we give the Fourier coefficients for the square wave and sawtooth wave signals.

Example 3.1.2. We assume $m = 0$.

- For the square wave function s,

$$a_0 = 1 \text{ and } b_0 = 0$$

and, for every $n > 0$,

$$a_n = \frac{1}{\pi n} sin(\pi n),\ b_n = -\frac{1}{\pi n}(cos(\pi n) - 1)$$

- For the sawtooth wave function r,

$$a_0 = 1 \text{ and, for every } n > 0,\ a_n = \tfrac{1}{(\pi n)^2} cos(2\pi n) + \tfrac{1}{(2\pi n)^2}.$$
$$b_0 = -\tfrac{1}{2} \text{ and, for every } n > 0,$$
$$b_n =$$
$$-\tfrac{1}{\pi n} cos(\pi n) - \tfrac{1}{(\pi n)^2} sin(\pi n) + \tfrac{1}{\pi n} sin(\pi n) + \tfrac{1}{(\pi n)^2} cos(2\pi n) - \tfrac{1}{(\pi n)^2} cos(\pi n)$$

□

We do emphasize that for an arbitrary periodic function f the Fourier series will not necessarily converge for every $x \in \mathbb{R}$, and in case of convergence will not always equal $f(x)$.

Definition 3.1.4. A function f is called *piecewise continuous* on $[a, b]$ if f is continuous for every $x \in (a, b)$, except possibly in a finite number of points $x_1, \ldots, x_n$. Moreover, $f(a^+)$ and $f(b^-)$, $f(x_i^+)$ and $f(x_i^-)$ should exist for every $i = 1, \ldots, n$. A piecewise continuous function f on the interval $[a, b]$ is called *piecewise smooth* if the first derivative of f is piecewise continuous. [2] □

[2] We do emphasize that the function occurring between partition points is not necessarily linear as it is a tradition in real recursive functions theory.

We say that a function f is piecewise continuous on $\mathbb{R}$ if f is piecewise continuous on each subinterval $[a, b]$ of $\mathbb{R}$. Analogously, a function f is called *piecewise smooth* on $\mathbb{R}$ if f is piecewise smooth on each subinterval $[a, b]$ of $\mathbb{R}$.

Next, we present the fundamental theorem which shows that, for piecewise smooth functions, the Fourier series does equal f at the points of continuity.

Theorem 3.1.1. *Let f be a periodic piecewise smooth function on $\mathbb{R}$ with Fourier coefficients a_n and b_n. Then*

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n cos(nx) + b_n sin(nx)) = \frac{1}{2}(f(x^+) + f(x^-)).$$

Proof. See **Fundamental Theorem of Fourier Series** (pp. 90-92) in [Bee03]. □

And finally, we have a result, that will be needed in next section, which establishes that the Fourier series of a periodic piecewise smooth function converges to the derivative of the function itself.

Theorem 3.1.2. *Let f be a periodic piecewise smooth function on $\mathbb{R}$ with Fourier coefficients a_n and b_n, and f' be a piecewise smooth function on $\mathbb{R}$. Then*

$$f'(x) = \sum_{n=1}^{\infty} (n b_n cos(nx) - n a_n sin(nx)) = \frac{1}{2}(f'(x^+) + f'(x^-)).$$

Proof. See **Theorem of Differentiation of Fourier Series** (pp. 101-102) in [Bee03]. □

The above concepts and results about Fourier theory are the essentials to study the relationship between Fourier series and real recursive functions as we will see in the next section.

3.2 Periodic real recursive functions

As we said before, Fourier series enable us to reason about many periodic phenomena in nature, and, then, periodic functions play the main role to model them. In what follows, we will carry on Fourier series, and a particular

type of periodic functions, to real recursive function theory. All background about Fourier analysis can be found in [Bee03], which has been our source of notation and terminology.

Definition 3.2.1. We say that a real recursive function f is *periodic* with a real recursive period z if, for every $x \in \mathbb{R}$, $f(x+z) = f(x)$. □

Proposition 3.2.1. *If f is a periodic recursive function with a real recursive period z (> 0), then the Fourier coefficients a_n and b_n are real recursive numbers.*

Proof. For every $x \in \mathbb{R}$ and every $n \in \mathbb{N}$, $\frac{2}{z} f(x) cos(\frac{2\pi nx}{z})$ and $\frac{2}{z} f(x) sin(\frac{2\pi nx}{z})$ are real recursive expressions. For some $n \in \mathbb{N}$, let g_n be a recursive function defined as follows: $g_n(0) = 0,\ \partial_x\ g_n(x) = \frac{2}{z} f(x) cos(\frac{2\pi nx}{z})$. So, we obtain the following real recursive expression:

$$g_n(y) = \frac{2}{z} \int_{0 \leq x \leq y} f(x) cos(\frac{2\pi nx}{z}) dx$$

Therefore, $g_n(z) = a_n$ is a real recursive number. Analogously, for b_n. □

The necessary and sufficient condition for the application of fundamental theorem for Fourier series say us that the function must be periodic piecewise smooth on $\mathbb{R}$. Until now, we did not say nothing about this kind of functions in real recursive functions framework.

Proposition 3.2.2. *If $f_1, \ldots, f_k$ are real recursive scalar total functions and $a_1, \ldots, a_k$ are real recursive numbers such that $a_1 < \ldots < a_k$, then*

$$\lambda x.\ \Theta_{[a_1,a_2)}(x) \times f_1(x) + \ldots + \Theta_{[a_k,+\infty)}(x) \times f_k(x)$$

is a real recursive function.

Proof. Notice that $\Theta_{[0,a_1]}, \ldots, \Theta_{[a_k,+\infty)}$, $\times$ and $+$ are real recursive functions. □

We call the real recursive function given just above a *piecewise real recursive function* if it is according with the mentioned conditions. Thus, we are ready to generate a bridge between Fourier series and the real recursive function theory and, in broad sense, to periodic functions via their Fourier expansions.

Proposition 3.2.3. *If f has a Fourier expansion with real recursive numbers as coefficients, then f is a real recursive expression.*

Proof. The expressions $sin(\frac{2\pi nx}{z})$ and $cos(\frac{2\pi nx}{z})$ are both real recursive, since $z \neq 0$. Then,

$$a_n sin(\frac{2\pi nx}{z}) + b_n cos(\frac{2\pi nx}{z})$$

is also a real recursive expression, because a_n and b_n, the Fourier coefficients, are real recursive numbers (see Proposition 1.), as well as, its finite sum

$$\sum_{n=1}^{\lfloor y \rfloor} a_n sin(\frac{2\pi nx}{z}) + b_n cos(\frac{2\pi nx}{z}).$$

Then

$$\lim_{y \to 0} \sum_{n=1}^{\lfloor y \rfloor} a_n sin(\frac{2\pi nx}{z}) + b_n cos(\frac{2\pi nx}{z})$$

is also a real recursive expression. And, finally,

$$\frac{a_0}{2} + \lim_{y \to 0} \sum_{n=1}^{\lfloor y \rfloor} a_n sin(\frac{2\pi nx}{z}) + b_n cos(\frac{2\pi nx}{z})$$

is a real recursive expression, which is the expression for the Fourier expansion. □ □

Definition 3.2.2. A function f is said to be *partially periodic* with period z if there exists z' such that, for every $x \geq z'$, $f(x + z) = f(x)$. □

Each partially periodic function f is indeed periodic after a point z' and, between 0 and z', we will require that there exists a finite number of discontinuities. It is obvious that if $z' = 0$, f is periodic in the sense of Definition 3.2.1.

Corollary 3.2.1. *Every partially periodic function is a real recursive function in the sense of Proposition 3.2.2.* □

In particular, piecewise functions defined on non-negative reals which are divided in two distinct parts: the first part it is defined between 0 and some non-negative real x, where there is a finite number of points of discontinuity, and the second part it is defined on non-negative reals greater than x but it is periodic. We will see that the finiteness of points of discontinuity in the left

of x and periodicity of the function in the right of x is the essential propriety to tackle the restriction of finite state machines in the recognition of infinite (piecewise) signals.

3.3 Automata can only recognize periodic real recursive functions

Automata theory is, usually, faced as the study of sets of strings or ω-strings over a finite alphabet accepted by finite state machines. Recently, some work has been done to lift concepts of automata theory from discrete to continuous time [Rab03]. Instead of signals defined over a discrete sequences of time instants, it is considered signals defined over non-negative reals. An interesting subclass of such signals is the set of piecewise continuous functions, because the well-known relationship with Fourier analysis (see e.g. [Vre03]).

In this section, we will study the computational power of continuous automata which are able to process piecewise continuous signals. For this, as we will see during the exposition and the proof of the result itself that is irrelevant the form of the function taken in each interval of time of each piecewise continuous signal. So, we restrict our attention to piecewise linear signals which are, particularly, apropriated for a representation based on ω-words, which holds the definition of the required automata in its simplest form. Previous work had been done with the simplest class of piecewise signals: the class of piecewise constant signals (see e.g. [Rab03]); and also with piecewise constant derivatives signals [AMP95].

We construct a certain kind of finite state automaton, whose states are continuous instead of discrete, and show that they only recognize partially periodic piecewise real recursive functions, i.e. partially-periodic piecewise real recursive functions with period $z\ (>0)$ which have a finite number of discontinuities between 0 and z. We assume that piecewise real recursive functions are defined over $\mathbb{R}^0_+$. In particular, without loss of generality, we study partially periodic piecewise linear functions because, between two points of discontinuity, the derivative of such functions is constant and, then, it is representable by a computable number in the classical sense (e.g. integer). In this case, we take the approach based on signals and automata over continuous-

time found in [Rab03] that inspired us.

In general, a *signal* is a function from $\mathbb{R}_0^+$ to $\mathbb{R}$. The well known square-wave and sawtooth-wave functions are examples of such signals. In what follows, we describe piecewise linear real recursive functions as signals which, in turn, are described by ω-words over $\mathbb{Z}$.

Definition 3.3.1. A *piecewise linear signal* s over $\mathbb{Z}$ is a four-tuple $< \alpha, \beta, \theta, \tau >$, where α, β and θ are ω-words over $\mathbb{Z}$ such that, for every $i \in \mathbb{N}$, if $\alpha_i > 0$ then $\theta_i > \beta_i$; otherwise, if $\alpha_i < 0$ then $\theta_i < \beta_i$; otherwise, $\theta_i = \beta_i$, and τ is an unbounded increasing ω-word over $\mathbb{Z}$ such that $\tau_0 = 0$ and, for every $i \in \mathbb{N}$ and every $t \in [\tau_i, \tau_{i+1})$, s_i of s defined on $[\tau_i, \tau_{i+1})$ by

$$s_i(t) = \beta_i + \int_{\tau_i}^{t} \alpha_i dt'$$

is a real recursive function. □

In each interval $[\tau_i, \tau_{i+1})$, the derivative of the (linear) real recursive function s_i is denoted by α_i, the value of $s_i(\tau_i)$ is denoted by β_i (i.e. the initial value) and, finally, θ_i denotes the maximum or minimum value taken by s_i in $[\tau_i, \tau_{i+1})$ according to the value of α_i. And, thus, we are not providing the piecewise linear signal itself but its first derivative which it is not necessarily continuous. We denote the set of piecewise linear signals over $\mathbb{Z}$ by $\mathbf{PLIN}(\mathbb{Z})$. Alternatively, each piecewise linear signal s above can also be represented by

$$s(t) = \lim_{y \to \infty} \sum_{1 \leq i \leq \lfloor y \rfloor} \Theta_{[\tau_i, \tau_{i+1})} \times s_i(t).$$

In general, $s(t)$ is not a real recursive function.

In operational sense, for $\tau_i \leq t < \tau_{i+1}$, if $\alpha_i > 0$, $s_i(t)$ increases from β_i until θ_i, since $\theta_i > \beta_i$; otherwise, if $\alpha_i < 0$ then $s_i(t)$ decreases from β_i until θ_i, since $\theta_i < \beta_i$; otherwise, $s_i(t)$ remains constant. As this construction suggests, each of such signals is a discontinuous piecewise linear function which has an infinite number of discontinuities. But, in particular, for every $i \in \mathbb{N}$, if $\beta_{i+1} = s_i(\theta_i)$ then we obtain a continuous piecewise linear signal.

With the above representation for signals it is easy to say what is a partially-periodic piecewise linear signal based on it. A particular case of such signals are the periodic piecewise linear signals.

Definition 3.3.2. We say that a piecewise linear signal $< u, v, w, \tau >$ over $\mathbb{Z}$ is *partially periodic* with period p if there exists $i \in \mathbb{N}$ such that, for every $j > i$, $(u_j, v_j, w_j) = (u_{j+p}, v_{j+p}, w_{j+p})$. □

As we can see above, the sufficient condition for a piecewise linear signal to be partially periodic is obtained by considering a time instant for which the triple formed by the first derivative, the initial value, and the maximum (or minimum) value of the signal, after a given period greater than 0, are equal. Notice that in the beginning of time such signals exhibits a non-periodic pattern with a finite number of discontinuities which is followed by a periodic pattern that, in our case, is a periodic piecewise linear signal. We denote by $\mathbf{PPPLIN}(\mathbb{Z})$ the set of partially periodic piecewise linear signals, whose representation are generated by computable infinite sequences α, β, θ and τ. And, thus, every signal in $\mathbf{PPPLIN}(\mathbb{Z})$ is a real recursive function.

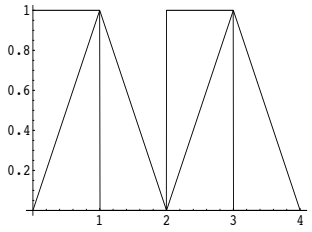

Figure 3.1: Square and triangle waves

Consider the simplest form of automaton where the set of states is finite. Usually, such states are seen as abstract entities suitable to describe discrete behaviors of systems in a given level of abstraction [Min72]. But, here, we take an approach that has been taken by hybrid systems community (e.g. [Tra98]) where, in a simplest case, we have a state variable which is described by a continuous behavior over time. In this case, the automaton represents the evolution of the system through time where each transition represents the change of the regime of operation (i.e. the state of the automaton) which is described by a differential equation. So, to be coherent with signals considered above, each automaton has its state equipped with a first-order differential equation of the form $\frac{ds}{dt} = k$. Formally,

Definition 3.3.3. A *continuous automaton* $\mathcal{A}$ over $\mathbb{Z}$ is a triple (Q, δ, q_0) where

- $Q = \{(c, b, d) \in \mathbb{Z}^3 : \ b \leq d \ and \ c \geq 0\} \cup \{(c, b, d) \in \mathbb{Z}^3 : \ b > d \ and \ c < 0\}$ is a finite set (of *states*);

- $\delta : Q \times \mathbb{N}^2 \to Q$ is a function (the *transition function*) such that, for every $(c, b, d), (c', b', d') \in Q$ and $(a, a') \in \mathbb{N}^2$ such that $a' > a$,

$$\delta((c, b, d), (a, a')) = (c', b', d')$$

  iff $c' \leq 0$ if $c > 0$, or $c' \geq 0$ if $c < 0$, or (($c' \in \mathbb{Z} - \{0\}$) or ($c' = 0$ and $b' \neq b$)) if $c = 0$.

- $q_0 \in Q$ (the *initial state*). □

By the condition imposed, in the definition just above, to define the transition function, we can see that a transition take place only when the value of c changes, except in the case of $c = 0$, which can remain as 0, and in this case we must take $b' \neq b$.

We say that a piecewise linear signal $\sigma =< u, v, w, \tau >$ over $\mathbb{Z}$ is *accepted* by (or is a solution of) $\mathcal{A}$ if there exists an infinite sequence $(b_0, c_0, d_0) \ldots$ over Q such that (b_0, c_0, d_0) is the initial state, and, for every $i \in \mathbb{N}$, $b_i = v_i$, $c_i = u_i$, $d_i = w_i$ and $(b_{i+1}, c_{i+1}, d_{i+1}) = \delta((b_i, c_i, d_i), (t_i, t_{i+1}))$.

As an example, consider the square wave and triangle wave signals in figure above. It is easy to see that continuous automata have, respectively, $Q_{sq} = \{(1, 0, 1), (0, 0, 0)\}$, $q_0^{sq} = (1, 0, 1)$ and

$$\delta_{sq} = \{((1, 0, 1), (i, i + 1), (0, 0, 0)), ((0, 0, 0), (i + 1, i + 2), (1, 0, 1)) : \ i \geq 0\},$$

and $Q_{tri} = \{(0, 1, 1), (1, -1, 0)\}$, $q_0^{tri} = (0, 1, 1)$

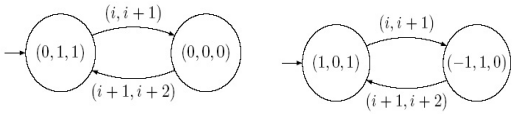

Figure 3.2: Continuous automata for square and triangle waves

and

$$\delta_{tri} = \{((0, 1, 1), (i, i+1), (1, -1, 0)), ((1, -1, 0), (i+1, i+2), (0, 1, 1)) : \ i \geq 0\}.$$

Proposition 3.3.1. *A piecewise linear signal s is accepted by a continuous automaton $\mathcal{A}$ if and only if $s \in \mathbf{PPPLIN}(\mathbb{Z})$.*

Proof. If s is a piecewise linear signal accepted by $\mathcal{A}$, then there exists an infinite sequence $(b_0, c_0, d_0) \ldots$ over Q such that (b_0, c_0, d_0) is the initial state and, for every $i \in \mathbb{N}$, $\delta((b_i, c_i, d_i), (t_i, t_{i+1})) = (b_{i+1}, c_{i+1}, d_{i+1})$. Since Q is finite, there exists $i < j$ such that $(b_i, c_i, d_i) = (b_j, c_j, d_j)$. Let $p = j - i$, Therefore, for every $n \geq i$, $(b_n, c_n, d_n) = (b_{n+p}, c_{n+p}, d_{n+p})$. Conversely, if $s =< u, v, w, \tau >$ is a partially periodic piecewise linear signal, then there exist n_0 and $p > 0$ such that, for every $n \geq n_0$, $u_n = u_{n+p}$, $v_n = v_{n+p}$, and $w_n = w_{n+p}$. Consider the continuous automaton $\mathcal{A} = (Q, \delta, q_0)$ where $Q = \{(v_i, u_i, w_i) \in \mathbb{Z}^3 : i \in \mathbb{N}\}$ and, for every $i \geq 0$, $\delta((v_i, u_i, w_i), (t_i, t_{i+1}) = (v_{i+1}, v_{i+1}, w_{i+1})$, and (v_0, u_0, w_0) is the initial state. So, $\mathcal{A}$ accepts s. Since s is partially periodic, then there exists $i \in \mathbb{N}$ such that, for every $j \geq i$, $(v_j, u_j, w_j) = (v_{j+(j-i)}, u_{j+(j-i)}, w_{j+(j-i)})$. Therefore, Q is finite. □

The above result show us that each continuous finite state automaton can accepted only periodic piecewise signals, no matters what function we take between consecutive discontinuities. And, thus, periodicity impose the computational power for the continuous finite state automata.

Real recursion theory, introduced in [Moo96], has been considered as a model of analog computation. As it was enhanced in [MC04a], the operator of taking a limit captured from Analysis, can be also used properly to provide the opportunity to bring together classical computation and real and complex Analysis. In this chapter, we bring together Fourier Analysis and continuous finite state automata. Then, we show that the ingredients needed to deal with Fourier series, namely the piecewise smooth periodic functions, can be embodied in the framework of real recursive vector functions, originally introduced in [Moo96] and revised and expanded in [MC04a]. It seems obvious that not all piecewise smooth periodic functions can be accepted by continuous finite state automata (e.g. [Rab03]). Then, we also show that a special kind of automaton over continuous time are only be able to accept periodic piecewise linear signals, for which it is possible a characterization by ω-words is provided.

Chapter 4

Infinite time Turing machines

In this chapter, we present the essential background about ordinal numbers and their structural properties over which it is considered two distinct class of ordinals, namely the clockable and writable ordinals, to describe the running time of infinite time Turing machines and their halting problems. As we will see, in the next chapter, these are the essential ingredients to simulate infinite time Turing machines with real recursive functions with infinite limits, operating in ordinal time.

4.1 Hypercomputation machines

Hypercomputation (or super-Turing computation) is the study of machines that can compute more than the Turing machines. The analysis of hypercomputation began in Turing's 1939 paper 'Systems of Logic Based on Ordinals' [Tur39]. There, Turing introduced the O-machine, a Turing machine equipped with an additional resource (such as an oracle), which could compute functions that were beyond the power of Turing machines. It was, however, much less practical in its design and not intended to represent a method that could be followed by human beings. Nevertheless, its importance as an abstract model for analysing and extending the concept of computation was recognized and it has made a great impact in classical recursion theory.

Once having specified the operation of the classical Turing machines, one knows what it means for a function on the natural numbers to be computable, and for a set of natural numbers to be recursive or recursively enumerable.

The halting set, the set of programs halting on a given input, appears as a canonical example of a set that is recursively enumerable but not recursive. Suppose you had some way to answer the halting problem. With this set in hand, what could you compute from it? To answer this question, it had been developed a theory of oracle computation (or relative computability). The notion of relative computability reveals a vast mathematical structure: the Turing degrees.

In early 20th century, Bertrand Russell, Ralph Black, and Hermann Weyl independently proposed the idea of a process that performs its first step in one unit of time and each subsequent step in half the time of the step before. So, such a process could complete an infinity number of steps in two time units. The application of this temporal patterning to Turing machines has been discussed briefly by Ian Stewart and in much more depth by David Copeland under the name of accelerated Turing machines [Cop98]. Since Turing's account of his machines has no mention of how long it takes to perform an individual step, this acceleration is not in conflict with the mathematical conception of a Turing machine.

More recently, Joel Hamkins and Andy Lewis presented a model of a Turing machine that operates for transfinite numbers of steps, the so called infinite time Turing machine, which is a natural extension of the Turing machine to transfinite ordinal times [HL00]. The existence of accelerated Turing machines has implications concerning the theoretical limits of computation because, for example, they solve the halting problem. Every new model of computation naturally also provides a corresponding new halting problem. Thus, in the supertask context of infinite time Turing machines, we have the supertask halting problem. Once we have introduced the halting problems for infinite time Turing machines, it is again natural to ask what we could compute if we have some means to solve these problems, and we are going to the notion of oracle computation. There are, however, two natural types of oracles to use in the infinite time Turing machine context.

Several other approaches have been presented to treat infinite computation as extensions of finite computation. We can refer the most important ones: Büchi machines [PP04] and higher-order recursion [Sac90].

4.2 Some basic facts about ordinals

In this section, we present some elements of ordinals, namely the definition of ordinal number and some of its structural properties, induction and recursion over ordinals. But, before we get in into the formal presentation of the essential concepts that we will need to understand the structure of time involved in the behavior of infinite time Turing machines, we describe briefly the construction of ordinal numbers in the classical constructive way. For further details, we should consult [End77].[1]

The first ordinal 0 is the empty set $\emptyset$ and, then, $1 = \{\emptyset\} = \{0\}$, $2 = \{\emptyset, \{\emptyset\}\} = \{0, 1\}$, and so on. These are the finite ordinals, corresponding to the natural numbers. The first infinite ordinal is $\omega = \{0, 1, 2, \ldots\}$, which is the set of natural numbers $\mathbb{N}$, and it is also the first limit ordinal, indeed the smallest limit ordinal apart from 0. After ω comes $\omega + 1 = \omega \cup \{\omega\}$, $\omega + 2 = (\omega + 1) + 1$, $\omega + 3$, and so on. The next limit ordinal is $\omega 2$, which is the set consisting of all n, where $n \in \omega$, and all $\omega + n$, where $n \in \omega$. Then come $\omega 2 + 1$, $\omega 2 + 2$, ..., $\omega 3$, $\omega 3 + 1$, $\omega 4$, ..., ωn, ..., ω^2, ...

We can specify an ordering $<$ on the collection of all ordinals by defining $\alpha < \beta$ if $\alpha \in \beta$. If α is an ordinal, then the successor of α is the ordinal $\alpha + 1 = \alpha \cup \{\alpha\}$, which is the least ordinal greater than α. Then $\alpha + 1$ is said to be a successor ordinal. If α is a successor ordinal, say $\alpha = \beta + 1$, we denote β by $\alpha - 1$. An ordinal α is a limit ordinal if α is not the successor of any ordinal.

Definition 4.2.1. We say that a set A is *transitive* iff, for all sets u, v, if $u \in v$ and $v \in A$, then $u \in A$. □

Let n be a number defined as a set like the construction given above. Then n is transitive and, every $x \in n$ is transitive. This leads to:

Definition 4.2.2. A set α is an *ordinal number* if α is transitive and well ordered by $\in$. □

We denote by Ord the set of ordinal numbers. If $\alpha, \beta \in Ord$, then we write $\alpha < \beta$ instead of $\alpha \in \beta$ and $\alpha \leq \beta$ instead of $\alpha < \beta$ or $\alpha = \beta$.

[1] A suggestion made by J. D. Hamkins himself.

Some important structural properties concerning the set Ord are easy to proof using the *Axiom of Foundation*, namely Ord is transitive, linearly ordered and well ordered by relation $<$.

Definition 4.2.3. An ordinal $\alpha \in Ord$ is a *successor ordinal* if there exists $\beta < \alpha \in Ord$ such that $\alpha = \beta + 1$. Moreover, an ordinal $\alpha \in Ord$ is a *limit ordinal* if α is not a successor ordinal and $\alpha \neq 0$. □

The most remarkable fact about ordinals is that the principles of induction and recursion can be extended from the set $\mathbb{N}$ to the set Ord. According to the various types of ordinals, we can distinguish the initial case 0, the successor case and the limit case. The following theorem, which looks more like the familiar principle of complete induction, establishes an induction principle over Ord.

Theorem 4.2.1. *Let $\varphi(x_1, \ldots, x_n, v)$ be an $\in$-formula and assume that*

- $\varphi(x_1, \ldots, x_n, 0)$*;*
- *for every $\alpha \in Ord$, if $\varphi(x_1, \ldots, x_n, \alpha)$, then $\varphi(x_1, \ldots, x_n, \alpha + 1)$;*
- *for every $\alpha \in Ord$, if α is a limit ordinal, then, for every $\beta < \alpha$,*

$$\text{if } \varphi(\beta, x_1, \ldots, x_n), \text{ then } \varphi(\alpha, x_1, \ldots, x_n).$$

Then, for every $\alpha \in Ord$, $\varphi(x_1, \ldots, x_n, \alpha)$.

Proof. See Chapter 8 (pp. 209-240) in [End77]. □

The most transfinite construction principle is the construction by recursion along the ordinals.

Theorem 4.2.2. *Let $G : V \to V$ be a definable function. Then, there is a unique definable function $F : Ord \to V$ such that, for every $\alpha \in Ord$ satisfies the recursion equation*

$$F(\alpha) = G(F(\alpha) \upharpoonright).$$

Proof. See Chapter 8 (pp. 209-240) in [End77]. □

The recursion rule G will usually be described separately for the initial case, the successor case and for limit ordinal case.

Theorem 4.2.3. *Let $G_0 \in V$, $G_{succ} : V \to V$ and $G_{lim} : V \to V$ be definable functions. Then, there exists a unique definable function $F : Ord \to V$ such that*

- $F(0) = G_0$;
- *for every $\alpha \in Ord$, $F(\alpha + 1) = G_{succ}(F(\alpha))$;*
- *for every $\alpha \in Ord$, if α is a limit ordinal, then $F(\alpha) = G_{lim}(F(\alpha) \uparrow)$.*

Proof. See Chapter 8 (pp. 209-240) in [End77]. □

An example of a recursive construction is the von Neumann hierarchy $(V_\alpha : \alpha \in Ord)$ with $V_\alpha = \emptyset$, $V_{\alpha+1} = P(V_\alpha)$ and $V_\lambda = \cup_{\alpha<\lambda} V_\alpha$. The standard arithmetic operations (i.e. addition and multiplication) have well-known recursive definitions which can be extended to all the ordinals by transfinite recursion. Moreover, such operations are continuous at limit ordinals with respect to ordinals limits.

Definition 4.2.4. Let $(\alpha_i : i < \lambda)$ be a non-decreasing sequence of ordinals of limit length λ. Then

- $\lim_{i<\lambda}\ \alpha_i = \cup_{i<\lambda}\ \alpha_i$ is the *limit* of $(\alpha_i : i < \lambda)$;
- $\liminf_{i<\lambda}\ \alpha_i = \lim_{i<\lambda}\ min\{\alpha_i : i \leq j < \lambda\}$ is the *inferior limit* of

$$(\alpha_i : i < \lambda).$$

□

4.3 Infinite time Turing machines

In [HS01], for convenience, an infinite time Turing machine has three tapes: one for input, one for scratch and one for output. Because there seems to be no need to limit ourselves to finite input and output—the machines have plenty of time to consult the entire input tape and to write on the entire output tape before halting—the natural context for these machines is Cantor Space 2^ω, the space of infinite binary sequences. For our purposes here, we refer the members of 2^ω as real numbers, intending by this terminology to mean infinite binary sequences. We regard the set of natural numbers i.e.,

the set of binary sequences with initial finite block of 1s and then all 0s, as a subset of 2^ω by identifying the number **0** with the sequence $< 0, 0, 0, \ldots >$, the number **1** with $< 1, 0, 0, \ldots >$, the number **2** with $< 1, 1, 0, \ldots >$, and so on.

Here, to simplify, we assume an infinite time Turing machine with a single tape that, as it is usual for classical Turing machines (*vide* [Min72] for more details), consists of a finite set of states, sometimes also called the finite control of the machine, and a semi-infinite tape, divided into cells, equipped with a tape head which can move right or left, scanning the cells of the tape, one at a time. A semi-infinite tape means that it has no rightmost cell, but it does have a leftmost cell and, thus, when the head is on the leftmost cell, it is not allowed to move left. At each moment, the machine is in one of the states, and, then it can read the content of the scanned cell, change its content, move head right or left, and change its state. All these operations form a computation step, and are uniquely defined by a transition function, which is a function of the current state and the symbol read from the tape.

In [HS01], it is presented an infinite time Turing machine with only one tape. One the one hand, it is shown that the two kind of machines give rise to exactly the same class of decidable sets, the same degree structure and, at least for functions whose range is contained in $\{0, 1\}$, the set of natural numbers given just above, 2^ω and $\{1\} \times 2^\omega$. One the other hand, they shown that there are computable functions which are not computable by any one-tape machine; indeed, the class of one-tape computable functions is not even closed under composition. Nevertheless, every computable function is in a precise sense nearly computable by a one-tape machine, and the closure of the class of one-tape computable functions under composition yields the full class of all infinite time computable functions.

Definition 4.3.1. An *infinite time Turing machine with a single tape* $\mathcal{M}$ over the binary alphabet is a five-tuple

$$(Q, \delta, q_0, q_{limit}, F)$$

where:

- Q is a finite set (of states),

- $\delta : Q \times \{0,1\} \to Q \times \{0,1\} \times \{L,R\}$ is a partial function (the transition function),
- $q_i \in Q$ (the initial state),
- $q_{limit} \in Q$ (the limit state),
- $F \subseteq Q$ (the set of halting states). □

Notice that, in the above definition, R means moving one cell to the right and L means moving one cell to the left, if there are cells to the left; otherwise, do not move.

The only difference from classical Turing machines is the infinite time Turing machine's limit state, and the fact that if the computation doesn't reach the halt state at any given $\beta < \lambda$ for a limit ordinal λ, then it will go into the limit state and set all cells according to the limit configuration described below.

A configuration of an infinite time Turing machine with a single tape $\mathcal{M}$, hereafter abbreviated infinite time Turing machine, is a complete recording of all relevant data in a given moment.

Definition 4.3.2. A *configuration* of $\mathcal{M}$ is an infinite (binary) sequence

$$w_0 \dots w_i q w_{i+1} \dots$$

where $q \in Q$ and $i \in \mathbb{N}$. We assume that the head of $\mathcal{M}$ is scanning w_{i+1}. The *initial configuration* of $\mathcal{M}$ is the configuration $q_i w_0 w_1 \dots$, the *limit configuration* is the configuration $q_{limit} w_0 w_1 \dots$ and an *halting configuration* is the configuration $w_0 \dots w_i q w_{i+1} \dots$, where $q \in F$. □

To determine the configuration of the machine at any successor ordinal time, the new configuration is defined from the previous one according to the classical Turing machine rules. But, at a limit ordinal time, the configuration of the machine is defined based on all preceding configurations, since the last ordinal.

Definition 4.3.3. Given an infinite sequence of configurations

$$C_{\omega_{j-1}} C_{\omega_{j-1}+1} C_{\omega_{j-1}+2} \dots C_{\omega_j}$$

The *limit configuration* C_{ω_j} in the limit ordinal ω_j is the configuration $q_{limit}w$ where

$$w_i^{\omega_j} = \limsup_{k \to \infty} \; w_i^{\omega_{j-1}+k}$$

where $w_i^{\omega_{j-1}+k}$ denotes the value of w_i in the ordinal $\omega_j + k$ (for each $k \in \mathbb{N}$), and $w_i^{\omega_j}$ denotes the value of w_i in the limit ordinal ω_j. □

Recalling what occurs in each limit ordinal stage: the heads resets to the left-most cell; the machine is placed in the special limit state; and the values in the cells of the tapes are updated by computing a kind of limit of the previous values that cell has displayed. If the values in a cell have stabilized before a limit stage, then the limit value displayed by that cell at the limit stage will be this stabilized value; otherwise, when the cell's value has alternated from 0 to 1 and back again unboundedly often before a limit stage, then the limit value is set to 1. This limit value is equivalent, by definition, to computing for each cell the limit sup of the previous values displayed in the cell. With the limit stage configuration thus completely specified, the machine simply continues computing. If after some amount of time the halting state is reached, the machine gives as output whatever is written on the output tape.

A computation can now be defined as a sequence of configurations:

Definition 4.3.4. Given an infinite time Turing machine $\mathcal{M}$ and an input w, a *computation* of $\mathcal{M}$ on w is a (finite or infinite) sequence of configurations of $\mathcal{M}$ on w, each step from a configuration to the next obeys the transition function, and ends with a halting configuration or a limit configuration. □

Configurations and computations can be described as well by words over some alphabet, and we will identify configurations and computations with the words describing them.

We aim now to establish a limit on the complexity of decidable sets. We introduced the idea of a configuration, which is a real number coding the complete description of an infinite time Turing machine while it is computing. Thus, in some canonical manner, a configuration codes the program the machine is running, the position and state of the head and the complete content of the tape. Now let us say that a transfinite sequence of configurations accords with the program P when each successive configuration is obtained by running P on the configuration described by the previous configuration

and the limit configurations are obtained from the earlier ones according to the computation rules. We will say that a sequence of configurations according to a program is settled when the last configuration has either obtained a halting state or else repeats an earlier configuration, in the strong sense of a computation repeating itself used after Theorem 4.2.1 (above). The sequence of configurations represents, in the first case, a halting computation and, in the second a computation which will endlessly repeat. Thus, a settled sequence of configurations informs us of the outcome of an infinite time computation. With these ideas we can establish the complexity of such computations ([Wel00]).

An interesting result about computation, introduced in [HL00], that leads our attention only to countable ordinals.

Theorem 4.3.1. *Every halting infinite time computation is countable.* □

Proof. See Theorem 1.1 (pp. 6) in [HL00]. □

Corollary 4.3.1. *Every infinite time computation either halts or repeats itself in a countably many steps.* □

Proof. See Corollary 1.2 (pp. 7) in [HL00]. □

In the classical sense, the function $f : 2^{<\omega} \to 2^{<\omega}$ is said to be Turing computable if there is a Turing machine that computes it. A Turing machine that computes f may fail to halt for an input u and, in this case, f is undefined for u. Thus, Turing machines can compute both total and partial functions. In the case of infinite time Turing machines, we have

Definition 4.3.5. A partial function $f : 2^{\omega} \to 2^{\omega}$ is *infinite time computable* if there exists a (finite) program P_f such that, for every $x \in Dom(f)$, $f(x) = P_f(x)$. □

We assume that only functions with one argument are computed by our infinite time Turing machines. Infinite time Turing machines that compute infinite time functions can also be used to accept languages.

Definition 4.3.6. Let A be a subset of 2^{ω}. We say that A is *recursive* if the characteristic function C_A is computable by an infinite time Turing machine. Moreover, we say that A is *recursively enumerable* if the partial characteristic function $\hat{C}_A$ is computable by an infinite time Turing machine. □

We can also stratify the computable sets according to how long the computations take, i.e., infinite time decidibility is conditioned by a number of computation steps.

Definition 4.3.7. Let A be a subset of 2^ω and $\mathcal{M}$ be an infinite time Turing machine. We say that A is *α-recursive* if the characteristic function C_A is computable by $\mathcal{M}^{\leq\alpha}$. Moreover, we say that A is *α-recursively enumerable* if the partial characteristic function $\hat{C}_A$ is computable by $\mathcal{M}^{\leq\alpha}$. □

Thus, restricting to the case of finite input and finite time, the function f over $2^{<\omega}$ (i.e., the space of finite binary sequences) is ω-computable exactly when f is computable in the Turing machine sense.

4.4 Time ordinals and infinite time halting problems

Infinite time Turing machines are connected to classes of ordinals, namely the clockable and writable ordinals.

Definition 4.4.1. We say that an ordinal α is *clockable* if there is an infinite time Turing machine which on input **0** stops in exactly α many steps of computation (i.e., the α^{th} step of computation is the act of changing to the halting state). □

Any natural number is clockable, ω is clockable, and if α is clockable, then $\alpha + 1$ and $\alpha + \omega$ are clockable. Moreover, every ordinal up to ω^2 (i.e. the first ordinal which is a limit of limit ordinals), and if α is clockable, then $\alpha + \beta$ is clockable, for every $\beta < \omega^2$. In [HL00], we can see informal descriptions of infinite time Turing machines which can recognize all such examples of clockable ordinals.

Theorem 4.4.1. *Every recursive ordinal is clockable.* □

Proof. See **Recursive Clocks Theorem** (pp. 15) in [HL00]. □

The recursive ordinals extends at least up to ω_1^{CK}, the supremum of recursive ordinals. But, there are gaps in clockable ordinals, i.e. ordinals to which infinite time Turing machines cannot count, though they count higher. The following theorem reveals the structure of such gaps.

Theorem 4.4.2. *(Existence) There are gaps in clockable ordinals. In fact, the first gap above any clockable ordinal has size ω. (Big Gaps) For every clockable ordinal α, there are gaps of size at least α in the clockable ordinals.* □

Proof. See **Gap Existence Theorem** (pp. 18) and the **Big Gaps Theorem** (pp. 19) on [HL00]. □

Definition 4.4.2. We say that $w \in 2^\omega$ is *writable* if there is an infinite time Turing machine which can write w as the final output on input **0**. Moreover, we say that an ordinal ω is *writable* if there is a writable infinite binary word that codes ω. □

Theorem 4.4.3. *(Many Gaps) If α is a writable ordinal, then there are at least α many gaps of size at least α in the clockable ordinals. Moreover, if α is either clockable or writable, then the exact ordinal number of gaps of size at least α is neither clockable nor writable.* □

Proof. See **Many Gaps Theorem** (pp. 20) in [HL00]. □

These ordinals extend beyond the recursive ordinals and their supremum is inaccessible. The above theorem shows that there are no gaps in the writable ordinals as well as, it also shows that there are long stretches of clockable ordinals without any gaps.

Theorem 4.4.4. *(No Gaps) There are no gaps in the writable ordinals. (Gapless Blocks) There are large gapless blocks of clockable ordinals. Indeed, if α is writable in λ many steps, then $\lambda + \beta$ is clockable for every $\beta \leq \alpha$.* □

Proof. See **No Gaps Theorem** (pp. 20) and **Gapless Blocks Theorem** (pp. 23) in [HL00]. □

Much of the classical computability theory generalizes to the supertask context of infinite time Turing machines. For example, the classical *s-m-n* **theorem** and the **Recursion Theorem**. But some other classical results do not generalize: there is a non-computable function whose graph is recursively enumerable (for more details see **Lost Melody Theorem** in [HL00], pp. 28).

Perhaps, the most important question about classical Turing machines is the halting problem for which there is not a natural recursive characteristic function for it (or it is undecidable). The halting problem is to find an effective

procedure (i.e. an algorithm) that, given any Turing machine $\mathcal{M}$, say represented by its code m, and given any number n, will enable us to determine whether or not $\mathcal{M}$, given that number as input, ever halts.

Any function that is computable by a classical Turing machine is computable by an infinite time Turing machine. But the infinite time Turing machines are strictly more powerful than their classical counterparts. For example, the halting problem for classical Turing machines is decidable in ω many steps by infinite time Turing machines [HL00]. This argument is generalized to show that the infinite time Turing machines can decide membership in any given recursively enumerable set in ω many steps.

The classical halting problem has an infinite time counterpart.

Definition 4.4.3. The *bold-face halting set* $HALT$ for infinite time Turing machines is defined as follows:

$$\{(\mathcal{M}, x) : \mathcal{M}(x) \downarrow\}$$

In particular, the *light-face halting set* $halt$ for infinite time Turing machines is defined as follows:

$$\{\mathcal{M} : \mathcal{M}(\mathbf{0}) \downarrow\}.$$

□

Notice that $\mathcal{M}(x) \downarrow$ (resp. $\mathcal{M}(\mathbf{0}) \downarrow$) means that the infinite time Turing machine $\mathcal{M}$ stops for a given input x (resp. $\mathbf{0}$).

The classical arguments directly generalize to show that the bold-face and light-face halting sets for infinite time Turing machines are recursively enumerable but not recursive (using the classical diagonalization argument).

Theorem 4.4.5. *$HALT$ and $halt$ are recursively enumerable but not recursive.* □

Proof. See **Halting Problem Theorem** (pp. 24) in [HL00]. □

Approximations of both types of halting problems can exist and they produce several results when we consider such approximations as recursive and recursively enumerable sets. The α-approximations for $HALT$ and $halt$ can be given as follows.

Definition 4.4.4. The *α-approximation bold-face halting set* $HALT_\alpha$ for infinite time Turing machines is defined as follows:

$$\{(\mathcal{M}, x) : \mathcal{M}(x) \downarrow^{\leq\alpha}\}.$$

In particular, the *α-approximation light-face halting set* $halt_\alpha$ for infinite time Turing machines is defined as follows:

$$\{\mathcal{M} : \mathcal{M}(\mathbf{0}) \downarrow^{\leq\alpha}\}.$$

□

As it is assumed above, $\mathcal{M}(x) \downarrow^{\leq\alpha}$ (resp. $\mathcal{M}(\mathbf{0}) \downarrow^{\leq\alpha}$) means that the infinite time Turing machine $\mathcal{M}$ stops for a given input x (resp. $\mathbf{0}$) in less or equal α steps. The halting sets $HALT_\alpha$ and $halt_\alpha$ can be used, for example, in the proof of $P \neq NP \cap co-NP$ for infinite time Turing machines [Ham05].

No matters the type of ordinal, namely clockable or writable, the approximations of $HALT$ and $halt$ by one of these ordinals are decidable.

Proposition 4.4.1. *The following results are true:*

1. *If $\alpha < \beta$ then $HALT_\alpha$ is a proper subset of $HALT_\beta$.*
2. *If α is writable or clockable, then $HALT_\alpha$ and $halt_\alpha$ are recursive.* □

Proof. In [HL00], for 1., see Corollary 4.3 (pp.26) and, for 2., see Theorem 4.5. □

Notice that the first result is only for the α-approximation bold-face halting set $HALT_\alpha$.

The above approximations of $HALT$ and $halt$ can also provide relationships with approximations of decidibility and semi-decidibility in limit ordinals and in the supremum of clockable ordinals.

Theorem 4.4.6. *The following results are true:*

1. *For every limit ordinal α, neither $HALT_\alpha$ nor $halt_\alpha$ is α-recursive. But, if α is clockable, then $HALT_\alpha$ and $halt_\alpha$ are α-recursively enumerable and $(\alpha+1)$-recursive.*
2. *The set $halt_\alpha$ is recursive for every α below the supremum of the clockable ordinals.*

3. *If γ is the supremum of clockable ordinals, then $HALT_\gamma$ is recursively enumerable but not recursive.*

Proof. In [HL00], for 1., see Theorem 4.4 (pp. 26), for 2., see Theorem 4.6 (pp. 27), and, for 3., see Theorem 4.7 (pp.27). □

The study of halting sets and their approximations led us to the notion of relative computation and then to the infinite time Turing degrees which reveals, using the Post Problem to infinite time computations, the power and limitations of computation of infinite time Turing machines. An exhaustive presentation of such results can be found in [HL00]. These developments motivates the rise of infinitary complexity theory, which includes a solution of the infinite time Turing machine analogue of the $P \neq NP$ conjecture [Ham05].

Chapter 5

Embedding infinite time Turing machines in the general framework

The problem of infinity, which can appear in the sequel of non finishing computation, is a source of problems in theory and practice – the most important is, certainly, the halting problem. Usually, the first step to improve this situation is to change the behavior of a Turing machine, as we can see in the study of more powerful models of computation such as infinite time Turing machines, revised in the previous chapter, belonging to the so called hypercomputation (see [Gal06, Ord06] for a rigorous and complete discussion about this subject).

Recall that the class of real recursive functions is defined inductively, in a similar manner to Kleene's recursive functions. The basic functions are 1, -1, 0 and the projections, and the class is the closure of these basic functions for composition, solving of first order differential equations and the taking of infinite limits. As we already saw, the η-hierarchy is based on the minimum number of nested infinite limits required to describe some real recursive function. From a computational perspective, the η-hierarchy can also measure how many transfinite computational steps are required to compute some real recursive function. If we have a machine that performs ω computational steps, then we can solve the halting problem for Turing machines. Moreover, if such machine performs ω computational steps ω times, or ω^2

steps, then we can compute more than all Turing-computable functions. E.g., we can solve the halting problem using six nested infinite limits [MC04a] and simulate all Turing-computable functions using eight nested infinite limits. In this sense, the number of infinite limits required to define some function supplies us with a very natural way of ascertaining the non-computability of functions by Turing machines.

In this chapter, we extend the iteration to ordinals and, then, we simulate the infinite time Turing machines until ω^2, getting back all the work done in [Moo96, MC04a] for Turing machines, and their ω^2-approximations of halting problems with real recursive functions with infinite limits, again emphasizing the role of infinite limits in the theory of real recursive functions of Jerzy Mycka and José Félix Costa [MC04a], that extends the seminal work of Cristopher Moore [Moo96].

Notice that, several approaches have been taken to simulate Turing machines with finite dimensional analytical maps and flows [GCB05, KM99, Bra95, KCG94] that, using a very simple construction (see below), can be thought of as neural networks [SS95] or hybrid systems [AMP95], or even as optical ray tracing in three dimensions [RTY90]. However, piecewise linear functions, such as in [Moo90], are not very realistic from a physical point of view, although such maps can be smoothed into infinitely differentiable maps [Moo90], because most physical dynamical systems are analytic, at least in a perfectly classical world.

5.1 Simulating Turing machines in ω

Several authors have been shown that finite dimensional maps and flows can simulate Turing machines. The general approach is to associate each configuration of a given Turing machine to a point of the space $\mathbb{R}^n$ and, then, show that there is a dynamical system, with state space in $\mathbb{R}^n$, that embeds its evolution. It is also known that Turing machines can be simulated on compact spaces even of those of low dimension (e.g., [KCG94]).

Another approach has been taken to simulate the evolution of Turing machines with continuous flows in $\mathbb{R}^n$ (e.g., [Bra95] or, more recently, [MC04a]). Even knowning that those flows can be infinitely differentiable, no analytic

form of iterating the map that simulates the transition function of a Turing machine had been proposed until the recent work of Daniel Graça, Manuel Campagnolo and Jorge Buescu in [GCB05]. In such work, it is shown that Turing machines can be simulated by finite dimensional maps and flows which are both analytic and robust, considering simulations, in the presence of noise, on unbounded spaces. Such work is, in some sense, related to [KM99], where a constructive simulation of Turing machines using closed-form analytic maps is presented.

A Turing machine consists of an infinite tape for storing the input, output and scratch, and a finite set of internal states. The contents of the tape are finite strings over a given alphabet of symbols (hereafter, we assume a binary alphabet). This machine works in discrete steps. In each step it scans the symbol from the current position of the tape (under the head of the machine), change this symbol according to its current internal state, and moves the position of the tape to left or right with a change of its internal state. Some internal states are distinguished as final, when the machine reaches one of them, then it stops. Here, we assume a Turing model that obeys the classical constraints: (a) the set of internal states is finite; (b) the input is finite; (c) if the machine stops, then the output is finite,

Following [KM99], we associate the digits of the x and y coordinates of a point with left and right halves of a Turing machine's tape. Then we can shift the tape head by a basis constant x and y, and write on the tape by adding constants to them. Thus two dimensions suffice for a map or three for a continuous-time flow. Let $\mathcal{M}$ be a Turing machine with n states and m tape symbols, we will construct the real recursive function $f_{\mathcal{M}} : \mathbb{R}^2 \to \mathbb{R}^2$ that simulates $\mathcal{M}$. For each $(x, y) \in dom(f_{\mathcal{M}})$, let

$$x = q + n \sum_{i=0}^{\infty} (m+1)^i a_i$$

where q denotes the current state, a_0 denotes the code of a symbol under the head of $\mathcal{M}$, and, for every $i \in \mathbb{N}$, $0 \leq a_i < m$ (the blank symbol has code 0), that encodes the right half of the tape and the current state q $(0 \leq q < n)$, and let

$$y = \sum_{i=1}^{\infty} (m+1)^{i-1} a_{-i}$$

that encodes the left half of the tape.

We also need a real recursive characteristic of divisibility, h, such that $h(p,n) = 1$, if $p|n$; otherwise, $h(p,n) = 0$, which can also be given by the following expression:

$$h(p,n) = (\frac{sin(\pi n)}{psin(\frac{\pi n}{p})})^2$$

Although the function h is a division by 0 whenever $p|n$, it is only use to condense the following more complex expression:

$$\frac{sin(n\pi)}{psin(\frac{n\pi}{p})} = \frac{1}{p}\sum_{i=1}^{\lfloor \frac{p}{2} \rfloor}(-1)^{i+1}\left(\begin{array}{c} p-i-1 \\ i \end{array} \right)(2cos(\frac{n\pi}{2}))^{p-2i}$$

which is also real recursive and has no such singularities.

Let S_{q,a_0} be the new state, A_{q,a_0} be the printed symbol, and Δ_{q,a_0} be -1, 1 for moves in the left or the right direction, respectively, or 0 for final states (for which, by convention, there are not any movement). Then,

$$f_{\mathcal{M}}(x,y) = \sum_{q=0}^{n-1}\sum_{a_0=0}^{m-1}\Delta_{q,a_0}^2 h((m+1)n, x-q-na_0)\times$$

$$(\frac{1+\Delta_{q,a_0}}{2}(x_r,y_r) + \frac{1-\Delta_{q,a_0}}{2}\sum_{a_{-1}=0}^{m-1}h(m+1, y-a_{-1})(x_l,y_l)) \tag{5.1}$$

where

$$(x_r,y_r) = (S_{q,a_0} + \frac{x-q-na_0}{m+1}, (m+1)y + A_{q,a_0})$$

$$(x_l,y_l) = (S_{q,a_0} + (m+1)(x-q+n(A_{q,a_0}-a_0)) + na_{-1}, \frac{y-a_{-1}}{m+1})$$

is the required real recursive function to simulate $\mathcal{M}$. Notice that the above real recursive function $f_{\mathcal{M}}$ is analytical which has as a consequence in the lowest level of complexity of the simulation.

Proposition 5.1.1. *There are real recursive functions from the class H_1, which can simulate any Turing machine.* □

Proof. The n-th iteration of $f_{\mathcal{M}}$ given by (5.1), denoted by $f_{\mathcal{M}}^n$, is obtained by the real recursive functions $f, g : \mathbb{R} \to \mathbb{R}^2$ defined by the following system of equations:

$$f(0) = g(0) = (x,y)$$

$$\partial_z\, g(z) = (f_{\mathcal{M}}(f(z)) - f(z))\frac{\pi}{2} sin(\pi z)s(z)$$

$$\partial_z\, f(z) = \frac{(g(z) - f(z))\pi sin(\pi z)}{cos(\pi z) + \delta(cos(\pi z) - 1) - 1} s(-z)$$

where $s(z) = \Theta(sin(\pi z))$. Then, we can make

$$f_{\mathcal{M}}^{||\lfloor z \rfloor||}(x, y) = f(x, y, 2z)s(2z) + g(x, y, 2z - 1)(1 - s(2z)),$$

since $s(2z)$ is 1, if $z \leq ||\lfloor z \rfloor|| + \frac{1}{2}$; otherwise, 0.

The $||\lfloor z \rfloor||$-th iteration of $f_{\mathcal{M}}$ satisfies the following equation:

$$f_{\mathcal{M}}^{||\lfloor z \rfloor||} = f(2||\lfloor z \rfloor||) = g(2||\lfloor z \rfloor||).$$

It can be explained in the following way: as z changes from 0 to 1, then f is constant and g goes through the distance from (x, y) to $f_{\mathcal{M}}(x, y)$. For $z \in [1, 2]$, g is constant and f catches up, hence $f(2) = g(2) = f_{\mathcal{M}}(x, y)$. If $z > 2$ then the same cycle begins again (see Figure 5.1 below for $||\lfloor z \rfloor|| = 3$). Because $s \in H_1$ hence $f_{\mathcal{M}}^{||\lfloor z \rfloor||} \in H_1$. □

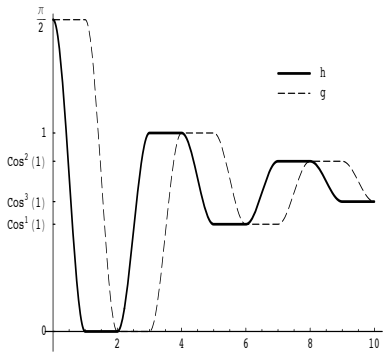

Figure 5.1: Iteration of $f_{\mathcal{M}}$. Courtesy of Bruno Loff

Let $H(x, y)$ be defined by

$$H(x, y) = \sum_{q \in F} \sum_{a=0}^{m-1} h((m + 1)n, x - q - na).$$

So, $H(x, y) = 1$, if the state written in x is final; otherwise, $H(x, y) = 0$. Such function constitutes the auxiliary function to construct the real recursive function to solve the halting problem for Turing machines as in [MC04a].

Proposition 5.1.2. *For any Turing machine $\mathcal{M}$, there exists a real recursive function in H_6 which is the characteristic function of the halting problem for $\mathcal{M}$.* □

Proof. Let

$$H_{\mathcal{M}}(x,y) = (\eta_z f_{\mathcal{M}}^{||\lfloor z \rfloor||}(x,y))H(\lim_{z\to\infty}(\eta_z f_{\mathcal{M}}^{||\lfloor z \rfloor||}(x,y))f_{\mathcal{M}}^{||\lfloor z \rfloor||}(x,y)).$$

Then $H_{\mathcal{M}}$ is the real recursive characteristic function of the halting problem for $\mathcal{M}$ which belongs to H_6. □

To obtain the function computed by $\mathcal{M}$, it is enough to iterate the steps up to the reaching of the final state by $\mathcal{M}$. If $\mathcal{M}$ ends in the final state for some (initial) tape (x,y), then there exists $n_0 \in \mathbb{N}$ such that the sequence $f_{\mathcal{M}}^n(x,y)$ is constant for some $n \geq n_0$.

Definition 5.1.1. Let $\mathcal{M}$ be a Turing machine. We say that the function $F(x,y)$ is ω- *computable* by $\mathcal{M}$, denoted by $F_{\mathcal{M}}(x,y)$, if

$$F_{\mathcal{M}}(x,y) = \lim_{z\to\infty}[f_{\mathcal{M}}^{||\lfloor z \rfloor||}(x,y)g(H(\lim_{z\to\infty}(\eta_z f_{\mathcal{M}}^{||\lfloor z \rfloor||}(x,y))f_{\mathcal{M}}^{||\lfloor z \rfloor||}(x,y)))]$$

where g is a function not defined at 0; otherwise, g is 1. [1] □

Notice that $F_{\mathcal{M}}$ is defined whenever the limit exists and the value of H is 1 (i.e., the machine $\mathcal{M}$ reaches for the initial tape (x,y) a final state); otherwise, $F_{\mathcal{M}}$ is undefined.

Proposition 5.1.3. *Given an ω-computable function $F_{\mathcal{M}}$ by a Turing machine $\mathcal{M}$, there exists a real recursive function in H_7 which computes $F_{\mathcal{M}}$.*

Proof. See Definition 5.1.1. □

5.2 Simulating infinite time Turing machines in ω^2

In this section, we will see that an elementary iteration scheme over ordinals restricts the time of simulation of infinite time Turing machines and both forms of halting problems to ω^2.

To show that arithmetical hierarchy can be computed by a finite number of limits using an infinite time Turing machines, as the result obtained in

[1] E.g., $g(x,y) = \lim_{y\to\infty} \frac{1}{1-exp(-|x|y)}$.

[HL00], we should be concerned, firstly, about an iteration schema to real recursive functions that works properly over ordinals.

So, we will say how to iteration, which is a fundamental operator in the classical theory of computation [Odi89] as well as in real recursion theory [Moo96], can be extended to countable finite ordinal numbers using the limit operator [MC04a].

Definition 5.2.1. Let $f \in REC(\mathbb{R})$. Then the *transfinite iteration* over Ord is defined as follows:

1. $f^0(x_1, \ldots, x_n) = (x_1, \ldots, x_n)$;
2. $f^\alpha(x_1, \ldots, x_n) = f(f^{\alpha-1}(x_1, \ldots, x_n))$, if α is not a limit ordinal;
3. $f^{\alpha'}(x_1, \ldots, x_n) = \lim_{i\to\infty} f^{\alpha+i}(x_1, \ldots, x_n)$, if α is a limit ordinal and α' is the next limit ordinal.
4. $f^{\alpha'}(x_1, \ldots, x_n) = \limsup_{i\to\infty} f^{\alpha+i}(x_1, \ldots, x_n)$, if α is a limit ordinal and α' is the next limit ordinal.
5. $f^{\alpha'}(x_1, \ldots, x_n) = \liminf_{i\to\infty} f^{\alpha+i}(x_1, \ldots, x_n)$, if α is a limit ordinal and α' is the next limit ordinal.

□

In particular, if we restrict the iteration to an upper bound given by the limit of limit ordinals, i.e., ω^2, then

$$\lim_{k\to\infty} f^{\omega k}(x_1, \ldots, x_n) = f^{\omega^2}(x_1, \ldots, x_n).$$

i.e., we need to use an infinite number of limits to reach ω^2. We should emphasize that, in (3) (the same for (4) and (5)) above, the limit of iteration has to exist for every limit ordinal ωk ($k \in \omega$) until ω^2; otherwise, the iteration until then become undefined.

In the proof of Proposition 5.1.1, we saw a coding scheme for Turing machines that is suitable to define the required two-phase of their transition functions (see [Bra95] for the motivation and results of this approach). However, the coding scheme for Turing machines presented in the previous section does not fit exactly the intended coding scheme for infinite time Turing

machines. But neverthless, in this section, we consider only finite binary input over $2^{<\omega}$ and, thus, we can regain and study the limits of the approach taken in the last section. But, we have to emphasize that natural inputs for infinite time Turing machines are (infinite) sequences in 2^{ω} instead. In this case, we have to use another codification for such sequences in 2^{ω} based on representations in a given natural base (e.g., based on base 2) as the one discussed in [SS95] and, then, construct a different real recursive function to simulate the required transition function for infinite time Turing machines. Having in mind that simplification, we must say what we intend for a function over $\{0,1\}^{<\omega}$ which is computable by an infinite time Turing machine operating in time until ω^2.

Definition 5.2.2. A function $f : 2^{<\omega} \to 2^{<\omega}$ is ω^2-*computable* if there is an infinite time Turing machine that, when starting in the initial configuration and, when following its (finite) set of instructions, reaches an halting configuration in ω^2. □

Additionally to x and y, which represent, respectively, the coding of the right, together with the code of the state, and the left half of the tape, we need an extra variable z to store the content of the limit configuration that is updated in each computational step between consecutive limit ordinals. With these four integer variables, we can simulate directly an infinite time Turing machine (over a binary alphabet) $\mathcal{M}$. Assume that $0 \leq q < n$, where n is the number of states of $\mathcal{M}$ and, for every $i \geq 0$, $a_i \in \{0,1\}$.

Let $\mathcal{M}$ be an infinite time Turing machine equipped with a single tape over a binary alphabet (hereafter we assume $m = 2$) and n states, then we will construct a real recursive function $f_{\mathcal{M}} : \mathbb{R}^4 \to \mathbb{R}^4$ that simulates $\mathcal{M}$ in ω^2. For each $(x_1, x_2, x_3, x_4) \in dom(f_{\mathcal{M}})$, let

$$x_1 = q + n \sum_{i=0}^{\infty} (m+1)^i a_i$$

where q denotes the current state, a_0 denotes the code of a symbol under the head of $\mathcal{M}$, and, for every $i \in \mathbb{N}$, $0 \leq a_i < m$ (the blank symbol has code 0), that encodes the right half of the tape and the current state q $(0 \leq q < n)$, and let

$$x_2 = \sum_{i=1}^{\infty} (m+1)^{i-1} a_{-i}$$

that encodes the left half of the tape. And, for the other required (auxiliary) tape, let

$$x_3 = n\sum_{i=0}^{\infty}(m+1)^i a_i$$

where, as we assumed for the simulation of Turing machines above, a_0 denotes the code of a symbol under the head of $\mathcal{M}$, and, for every $i \in \mathbb{N}$, $0 \leq a_i < m$ (the blank symbol has code 0), that encodes the right half of the tape, and let x_4 encodes the left half of the (auxiliary) tape as x_2 above.

Again, let S_{q,a_0} be the new state, A_{q,a_0} be the printed symbol, and Δ_{q,a_0} be -1, 1 for moves in the left or the right direction, respectively, or 0 for final states (for which, by convention, there are not any movement). Then,

$$f_{\mathcal{M}}(x_1,x_2,x_3,x_4) = \sum_{q=0}^{n-1}\sum_{a_0=0}^{m-1}\Delta_{q,a_0}^2 h((m+1)n, x_1 - q - na_0)$$

$$(\frac{1+\Delta_{q,a_0}}{2}(x_1^r,x_2^r,x_3^r,x_4^r)+$$

$$\frac{1-\Delta_{q,a_0}}{2}\sum_{a_{-1}=0}^{m-1} h(m+1, x_2 - a_{-1})(x_1^l,x_2^l,x_3^l,x_4^l)) \tag{5.2}$$

where

$$(x_1^r,x_2^r,x_3^r,x_4^r) =$$

$$(S_{q,a_0}+\frac{x_1-q-na_0}{m+1},(m+1)x_2+A_{q,a_0}),\frac{x_3-na_0}{m+1},(m+1)x_4+max\{A_{q,a_0},a_0\})$$

$$(x_1^l,x_2^l,x_3^l,x_4^l) =$$

$$(S_{q,a_0}+(m+1)(x_2-q+n(A_{q,a_0}-a_0))+na_{-1},\frac{x_2-a_{-1}}{m+1},$$

$$(m+1)(x_3+nmax\{A_{q,a_0},a_0\}-na_0)+na_{-1},\frac{x_4-a_{-1}}{m+1})$$

is the required real recursive function to simulate $\mathcal{M}$. Notice that the above real recursive function $f_{\mathcal{M}}$ is analytical which has as a consequence the lowest level of complexity of the simulation. Moreover, in each computational step between consecutive limit ordinals until ω^2, we calculate the maximum $max\{A_{q,a_0},a_0\}$, between the current symbol in the tape a_0 and the symbol that will printed in the tape A_{q,a_0}, that we keep track in the auxiliary pair (x_3,x_4). So, in each limit ordinal, the pair (x_3,x_4) gives us the codification of

the corresponding limit configuration which is the one obtained by the original infinite time Turing machine applying the $\limsup$ operator to each cell of the tape (see [HL00] for further details).

Let $f_{\mathcal{M}}$ be a real recursive function that simulates the transition function of an infinite time Turing machine $\mathcal{M}$. Then, for every ωk and $k \in \omega$,

$$\lim_{n\to\infty} f_{\mathcal{M}}^{\omega k+n}(x_1^{\omega k}, x_2^{\omega k}, x_3^{\omega k}, x_2^{\omega k}) = (x_1^{\omega(k+1)}, x_2^{\omega(k+1)}, x_3^{\omega(k+1)}, x_4^{\omega(k+1)})$$

where $(x_1^{\omega(k+1)}, x_2^{\omega(k+1)}, x_3^{\omega(k+1)}, x_4^{\omega(k+1)})$ is the code of a limit configuration of $\mathcal{M}$ in $\omega(k+1)$.

In the previous section, we saw how to simulate the transition function of a Turing machine with a particular real recursive function and, moreover, how the iteration of such function, in ω, can be simulated with a suitable system of differential equations. Next, we will see how the same system of differential equations can be reused to define the transition function of an infinite time Turing machine $\mathcal{M}$, iterating the corresponding real recursive function simulation $f_{\mathcal{M}}$ (in 5.2), restricting ourselves to ω^2 and to the iteration schema given in Definition 5.2.1. Considering $H_\omega = \cup_{n\geq 0} H_n$, we have

Proposition 5.2.1. *There are real recursive functions from H_ω which can simulate any infinite time Turing machine in ω^2.* □

Proof. Let $f_{\mathcal{M}}$ be the real recursive function that simulates an infinite time Turing machine $\mathcal{M}$. We are looking for real recursive functions $f, g : \mathbb{R} \to \mathbb{R}^4$ defined by a system of differential equations, analogous to the one given in (5.2), to simulate the iteration between consecutive limit ordinals until ω^2. Then,

- If $(x_1, x_2, x_1 - q_0, x_2)$ is the initial configuration, then

$$\lim_{z\to\infty} f_{\mathcal{M}}^{|\lfloor z \rfloor|}(x_1, x_2, x_1 - q_0, x_2) =$$

$$\lim_{z\to\infty} f(2|\lfloor z \rfloor|) = \lim_{z\to\infty} g(2|\lfloor z \rfloor|) = (x_3^\omega + q_{limit}, x_4^\omega, x_3^\omega, x_4^\omega).$$

- For every $k > 0$, if $(x_1^{\omega k}, x_2^{\omega k}, x_1^{\omega k} - q_{limit}, x_2^{\omega k})$ is the limit configuration in ωk, then

$$\lim_{z\to\infty} f_{\mathcal{M}}^{\omega k+|\lfloor z \rfloor|}(x_1^{\omega k}, x_2^{\omega k}, x_1^{\omega k} - q_{limit}, x_2^{\omega k}) = \lim_{z\to\infty} f^{\omega k}(2|\lfloor z \rfloor|) =$$

$$\lim_{z\to\infty} g^{\omega k}(2|\lfloor z \rfloor|) = (x_3^{\omega(k+1)} + q_{limit}, x_4^{\omega(k+1)}, x_3^{\omega(k+1)}, x_4^{\omega(k+1)})$$

where $f^{\omega k}(2|\lfloor z \rfloor|)$ and $g^{\omega k}(2|\lfloor z \rfloor|)$ is the solution of the following system of differential equations

$$f^{\omega k}(0) = g^{\omega k}(0) = (x_1^{\omega k}, x_2^{\omega k}, x_3^{\omega k}, x_4^{\omega k})$$

$$\partial_z\, g^{\omega k}(z) = (f_{\mathcal{M}}(f^{\omega k}(z)) - f^{\omega k}(z))\frac{\pi}{2} sin(\pi z) s(z)$$

$$\partial_z\, f^{\omega k}(z) = \frac{(g^{\omega k}(z) - f^{\omega k}(z))\pi sin(\pi z)}{cos(\pi z) + \delta(cos(\pi z) - 1) - 1} s(-z)$$

where $s(z) = \Theta(sin(\pi z))$.

$$f_{\mathcal{M}}^{\omega k + |\lfloor z \rfloor|}(x_1^{\omega k}, x_2^{\omega k}, x_3^{\omega k}, x_4^{\omega k}) =$$

$$f^{\omega k}(x_1^{\omega k}, x_2^{\omega k}, x_3^{\omega k}, x_4^{\omega k}, 2z)s(2z) + g^{\omega k}(x_1^{\omega k}, x_2^{\omega k}, x_3^{\omega k}, x_4^{\omega k}, 2z-1)(1 - s(2z)),$$

since $s(2z)$ is 1, if $z \leq |\lfloor z \rfloor| + \frac{1}{2}$; otherwise, 0.

In this case, for every $k > 0$, $f_{\mathcal{M}}^{\omega k} \in H_{max(1,k)}$ □

It is important to notice that we are using nothing more than a simple modification of the system of differential equations, introduced in [MC04a], to simulate our transfinite iteration. In each limit ordinal, we take a different initial configuration that is obtained by our iteration schema, applied since the last limit ordinal. Moreover, we can easily see that the above simulation of infinite time Turing machines is restricted to ω^2 because we only have available a finite number of limits to use, according to the intended classification in the η-hierarchy.

As we have seen in the previous chapter, the α-approximation for the bold-face halting problem $HALT_\alpha$ (the light-face halting problem $halt_\alpha$) for an infinite time Turing machine $\mathcal{M}$ asks us if we can reach the halting state in fewer or equal than α steps of computation, giving an input (x, y) (an input **0**). As we already know from [HL00], these two approximations are decidable since α is writable or clockable. More generally, the bold-face halting problem for an infinite time Turing machine $\mathcal{M}$ asks: does $\mathcal{M}$, given an input (x, y), reachs the halting state? Or, similarly, for the light-face halting problem: does $\mathcal{M}$, given the input **0**, reachs the halting state? According to [HL00], both versions of such halting problem are semi-decidable.

Adopting the above iteration schema, both ω^2-approximations of bold-face and light-face halting sets can be solved by real recursive functions in the following way:

Proposition 5.2.2. *For every infinite time Turing machine $\mathcal{M}$,*

1. *there exists a real recursive function in H_6 which is the characteristic function of the halting problem $HALT_{\omega^2}$ for $\mathcal{M}$.*

2. *there exists a real recursive function in H_6 which is the characteristic function of the halting problem $halt_{\omega^2}$ for $\mathcal{M}$.* □

Proof. Let $H(x) = 1$ if the state written in x is final; otherwise 0, which can be defined by

$$H(x) = \sum_{q \in F} \sum_{a=0}^{m-1} h((m+1)n, x - q - na).$$

Let

$$H_{\mathcal{M}}(x_1, x_2, x_1, x_2) =$$

$$(\eta_y f_{\mathcal{M}}^{\omega \lfloor y \rfloor \rfloor}(x_1, x_2, x_1, x_2)) H(\lim_{y \to \infty} (\eta_y f_{\mathcal{M}}^{\omega \lfloor y \rfloor \rfloor}(x_1, x_2, x_1, x_2)) f_{\mathcal{M}}^{\omega \lfloor y \rfloor \rfloor}(x_1, x_2, x_1, x_2))$$

where

$$\eta_y f^{\omega \lfloor y \rfloor \rfloor}(x_1, x_2, x_1, x_2) = \begin{cases} 1 & \text{if } \lim_{y \to \infty} f^{\omega \lfloor y \rfloor \rfloor}(x_1, x_2, x_1, x_2) \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$

The function $H_{\mathcal{M}}$ is a real recursive characteristic function of the halting problem for $\mathcal{M}$. For $halt_\alpha$ is a particular case of $HALT_\alpha$ for input $\mathbf{0}$. □

Recall that an infinite time Turing machine can reach an halting state in the following circunstances: in between limit configurations and, then, it remains to the ω^2 limit configuration, or in some limit configuration where the limit state is also an halting state. So, we only need to observe the limit configurations until the halting appears and, then, to do this until ω^2 which justifies the real recursive function in H_6 constructed in Proposition 5.2.2.

To obtain the function computed by an infinite time Turing machine $\mathcal{M}$, it is enough to iterate the steps up to the reaching of limit configuration provided by $\mathcal{M}$ in ω^2. If $\mathcal{M}$ ends in the final state for some (initial) tape configuration (x_1, x_2), then there exist $k_0, n_0 \in \mathbb{N}$ such that $f_{\mathcal{M}}^{\omega k+n}(x_1, x_2, x_1, x_2)$ is constant for some $n \geq n_0$ until the next limit ordinal, i.e., $\omega(k+1)$, and, ever since, for every $k > k_0$.

Definition 5.2.3. Let $\mathcal{M}$ be an infinite time Turing machine. We say that the function $F_{\mathcal{M}}(x_1, x_2)$ is ω^2-*computable* by $\mathcal{M}$ if

$$F_{\mathcal{M}}(x_1, x_2) =$$

$$\lim_{z\to\infty}[f_{\mathcal{M}}^{\omega\lfloor z\rfloor\rfloor}(x_1, x_2, x_1, x_2)g(H(\lim_{z\to\infty}(\eta_z f_{\mathcal{M}}^{\omega\lfloor z\rfloor\rfloor}(x_1, x_2, x_1, x_2))$$

$$f_{\mathcal{M}}^{\omega\lfloor z\rfloor\rfloor}(x_1, x_2, x_1, x_2)))].$$

where g is defined as in Definition 5.1.1. □

Then $F_{\mathcal{M}}$ is defined whenever the limit exists in each limit ordinal ωk (for $k \in \mathbb{N}$), and the value of H is 1 (i.e., the machine $\mathcal{M}$ reaches for the initial tape (x_1, x_2) a final state); otherwise is undefined.

So, we can establish a result that, in the next section, will correlate the arithmetical hierarchy with the infinite time Turing machines through their simulation with real recursive funtions with infinite limits.

Proposition 5.2.3. *Every ω^2-computable function is obtained by a real recursive function in H_7.*

Proof. Since the result of a given function, computed by an infinite time Turing machine over ω^2, is obtained in a limit configuration in ω^2, the real recursive function for $F_{\mathcal{M}}$, given in Definition 5.2.3, is in H_7. □

5.3 The arithmetical hierarchy in ω^2

In this section, we correlate the arithmetical hierarchy with the η-hierarchy through the simulation of infinite time Turing machines in ω^2 by real recursive functions with infinite limits in [MC04a].

We will introduce just below the relations of natural numbers taken from the arithmetical and analitical hierarchies. For a complete study of such hierarchies and their characterizations in classical recursion theory you should consider the survey in [Odi89].

The class $\Sigma_0^0 = \Pi_0^0$ contains only relations of natural numbers which have recursive characteristic functions. The upper stages of Σ_n^0 can be constructed, inductively, from the lower stages in the following way, for every $n \in \mathbb{N}$:

$$\Sigma^0_{n+1} = \{P : (\exists P' \in \Pi^0_n) P(m_1, \ldots, m_k) \equiv \exists m P'(m_1, \ldots, m_k, m)\}$$

and

$$\Pi^0_{n+1} = \{P : (\exists P' \in \Sigma^0_n) P(m_1, \ldots, m_k) \equiv \forall m P'(m_1, \ldots, m_k, m)\}.$$

and, for every $n \in \mathbb{N}$, we also have

$$\Delta^0_n = \Sigma^0_n \cap \Pi^0_n.$$

We call $\Delta^0_\omega = \cup_{n \geq 0} \Delta^0_n$ the class of arithmetical relations.

In particular, the class Σ^0_1 is sometimes called the class of recursively enumerable sets. Due to the indecidibility of the halting problem for Turing machines, membership of some $n \in \mathbb{N}$ for some $R \in \Sigma_1$ is only semi-decidable, i.e., some any given n, there is an algorithm which is guaranteed to terminate if $n \in R$; but not otherwise. This fact establish the relationship between arithmetical hierarchy and the computational power of Turing machines.

The next well-known result—the Post's Theorem—gives a different, purely recursion-theoretical, definition for arithmetical hierarchy, and explains the similarities of the various levels.

Theorem 5.3.1. *A relation R is:*

1. *Δ^0_{n+1} if and only if R is recursive relative to a set in Σ^0_n or a Π^0_n relation.*
2. *Σ^0_{n+1} if and only if R is recursively enumerable in a Σ^0_n or a Π^0_n relation.*

Proof. See **Post's Theorem** (pp. 372) in [Odi89]. □

Another interesting characterization of the arithmetical hierarchy is the so called the limit lemma of Schönfield, which says that Δ^0_2 sets are those which can be recursively approximated. But, in general,

Proposition 5.3.1. *For every $n \geq 1$, A is Δ^0_{n+1} if and only if there is an $n+1$-ary (total) recursive function g such that*

$$c_A(x) = \lim_{m_1 \to \infty} \ldots \lim_{m_n \to \infty} g(x, m_1, \ldots, m_n).$$

Proof. See **The Limit Lemma** (pp. 374) in [Odi89]. □

Next, we must look for the results obtained by Jerzy Mycka and José Félix Costa about the relationship between the arithmetical hierarchy and the η-hierarchy given in [MC04a]. And, then, from Proposition 5.1.1 and knowing that all natural recursive sets and relations have Turing computable total characteristics (given by real recursive functions), it is shown that:

Proposition 5.3.2. $\Sigma^0_0 = \Pi^0_0 \subset H_2$, *and, for every* $i \in \mathbb{N}$, Σ^0_i *and* Π^0_i *are in* H_{i+2}, *i.e.,* Δ^0_ω *is real recursive.*

Proof. See Propositions 5.1 and 5.2 in [MC04b]. □

The class of analytical relations is denoted by Δ^1_ω is obtained analogously to arithmetical hierarchy Δ^0_ω. In particular, $\Delta^1_0 = \Delta_\omega$.

Proposition 5.3.3. Π^1_1 *is in* H_6.

Proof. See Proposition 26 (pp. 24-25) in [MC04a]. □

Recently, Bruno Loff, Jerzy Mycka and José Félix Costa show by easy analytical maps that hyper-arithmetical hierarchies — such as the analytical hierarchy — are non-degenerate, i.e., that they do not collapse. These proofs are not done in the classical manner (see [Odi89] for a classical proof), but as corollaries of the non-collapsing character of the η-hierarchy, and will provide a clue on how analytical methods can be used to prove classical results of computability and complexity theories.

As it shown in [HL00], the computational power of infinite time Turing machines is beyond the resolution of halting problem, namely they decide sets of arithmetical and analytical hierarchies which are indecidable by Turing machines. To decide a predicate of the form $\exists n R(x, n)$, where $n \in \mathbb{N}$, we can simply try out all the possible values of n in turn. One either finds a witness n or else knows at the limit that there is no such witness, and in this way decides whether $\exists n R(x, n)$. Iterating this idea, we conclude by induction on the complexity of the statement that any first order number theoretic question is decidable with only a finite number of limits, that is, before stage ω^2. In fact, the class of sets that are decidable in time uniformly before ω^2 is exactly the class of arithmetic sets, the sets of reals that are definable by a statement using quantifiers over $\mathbb{N}$ (see [HL00, Theorem 2.6]).

The next theorem, introduced in [HL00], shows another example of the computational power of infinite time Turing machines that transcends the computational power of Turing machines.

Theorem 5.3.2. *Every arithmetic set in* Σ^0_ω *is recursive.*

Proof. See Theorem 2.1 (pp. 8) in [HL00]. □

The inductive argument in the proof of the above theorem shows that a Σ_n^0 set is recursive whose characteristic function is computed by an infinite time Turing machine using at most n limits. Next, we identify the classes of recursive (arithmetic) sets which take relatively few limits to be computed by a given infinite time Turing machine.

Theorem 5.3.3. *The arithmetic sets are exactly the recursive sets whose characteristic functions are computed using a bounded number of limits.*

Proof. See Theorem 2.6 (pp. 12) in [HL00]. □

In fact, the class of sets that are decidable in time uniformly before ω^2 is exactly the arithmetic sets, the sets of reals that are definable by a statement using quantifiers over the natural numbers.

Corollary 5.3.1. *Every arithmetic set can be computed by an ω^2-computable function in H_7.*

Proof. See Proposition 5.2.3. □

In fact, we show that the simulation of the transition function of a Turing machine, in terms of real recursive functions with infinite limits [MC04a], can be regain to simulate a particular kind of infinite time Turing machines, namely whose are restricted to finite binary inputs and operating only until ω^2. With a particular form of iteration over ordinals, we justify the restriction to ω^2 because the finiteness of η-hierarchy. Moreover, we show that the ω^2-approximation of bold-face and light-face halting problems for infinite time Turing machines can be achieved by a real recursive function in H_6. And, then, to compute a ω^2-computable function, the function computable by our restricted form of infinite time Turing machines, we need a real recursive function in H_7, and, thus, their computational power is sufficient to compute the arithmetic hierarchy, as the result obtained by Joel Hamkins in [HL00].

Chapter 6

Conclusions and further work

We feel that the work on the computability theory and computational complexity of analog computation, as well as its applications, considering the approach based on real recursive functions with infinite limits, introduced in [MC04a], after the seminal work of [Moo96], is very far from its end. In this dissertation, we provide a modest and incipient contribution to open such approach to hybrid computation, and to contribute to launch the seeds to embedding several hypercomputation models [Ord06], including other than infinite time Turing machines of Joel Hamkins and Andy Lewis.

We summarize our work, succinctly, emphasizing the main contributions to the development of real recursive theory, particularly the role of real recursive functions with infinite limits [MC04a] in the following applications:

1. In the first application, we show that the ingredients needed to deal with Fourier series, namely the piecewise smooth periodic functions, can be embodied in the framework of real recursive vector functions, originally introduced in [Moo96] and revised and expanded, with infinite limits, in [MC04a]. It seems obvious that not all piecewise smooth periodic functions can be accepted by finite automata over continuous time (e.g. [Rab03]). Then, we also show that a special kind of automaton over continuous time are only be able to accept partially periodic piecewise linear signals, for which it is possible a characterization by ω-words.

2. In the second application, we introduce a new kind of iteration schema

over ordinals and we regain the codification over the reals for finite inputs, introduced in [Moo96], as well as the system of differential equations involved in the simulation of Turing machines presented, recently, in [MC04a]. Then, we show that there are recursive functions with infinite limits to simulate infinite time Turing machines and their computational power, namely their ability to decide their halting problems in H_6, restricted to ω^2, and the arithmetic sets with a finite number of limits in H_7.

Below, we list some questions raised by our work, providing also new directions for further research .

- Although probably not very relevant for a computability-oriented study, but very interesting when we want to study another application of infinite limits, a recursive class of distributions, also known as generalized functions, could also be constructed using supremum and infimum operators over an interval (see [Myc03] for further details).

- It would be interesting to study how our model of hybrid computation can be extended to take into account interaction in an analog network, following the definitions of interaction suggested by Boris Trakhtenbrot in [Tra99].

- Investigating a more robust iteration schema, we must be able to extend the given simulation for infinite time Turing machines until ω^ω.

- Ralf Schindler initiated, in [Sch03], the study of infinite time complexity theory by showing the $P \neq NP$ conjecture for infinite time Turing machines, using methods from descriptive set theory to analyze the complexity of classes P and NP. Recall that all reals have length ω and the polynomial functions of ω are bounded by those of the form ω^n. Then, a $A \subset 2^\omega$ is in P if there is a program p and a natural number n such that p decides A and halts on all inputs in time less than ω^n. Moreover, $A \subset 2^\omega$ is in NP if there is a program p and a natural number n such that $x \in A$ if and only if there is y such that p accepts (x, y), and p halts on all inputs in time less than ω^n.

It would be interesting to study such conjecture using methods of real and complex analysis to define P and NP classes in the light of the above (new) definition of these classes for infinite time Turing machines, considering the development already done for complexity theory of real functions in [MC06].

- Peter Koepke has been working on a generalization of standard Turing computability on tapes of length ω, to computability on tapes of arbitrary ordinal length [Koe05]. Such work was inspired by both infinite time Turing machines of Joel D. Hamkins, Jeff Kidder and Andy Lewis [HL00] and Silver machines introduced by Jack H. Silver (see [Ric79]).

  It would be very interesting to understand deeply how and what Turing machines compute in transfinite ordinal time and space, having in mind this very recent work of Peter Koepke ([Koe05]). And, then, to see how we can transform our simulation of infinite time Turing machines by real recursive functons with infinite limits to provide a simultaneous simulation of Turing machines in both ordinal time and ordinal space.

- New directions for application of Scott topology of continuous domains have been emerged in computation on classical spaces, because continuous domains are the natural setting for continuous mathematics, since the representations are more direct and straightforward than those provided by algebraic domains (see [Eda97] for further details). In [ES98], it is shown that the notion of computability in Scott continuous domains is equivalent to the approach taken in [Wei00], which is equivalent to that of [PE89].

  It would be interesting to study how the approach to analog computation based on real recursive functions with infinite limits in [MC04a] can be also correlate with the above definition of computability over the reals in Scott continuous domains in [ES98].

- Computer simulations of the current iteration schema of a given real recursive function light up several deficiencies when we try to solve computationally, and then iterate, the system of linear differential equations [CMC00, CMC02], in part, because of the presence of θ-clock functions. So, we need other forms of systems of differential equations and, also,

to study a new schema for iteration of elementary real recursive functions, which are solutions of linear differential equations of higher order, and show that such functions do not increase more rapidly than Busy Beaver function.

Bibliography

[ACHH93] R. Alur, C. Courcubetis, T. Henzinger, and P. Ho. Hybrid Automata: an algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer-Verlag, 1993.

[AMP95] E. Asarin, O. Maler, and A. Pnueli. Reachability Analysis of Dynamical Systems having Piecewise-Constant Derivatives. *Theoretical Computer Science*, 138:35–65, 1995.

[Arn92] V. I. Arnold. *Ordinary Differential Equations*. Springer Textbook Series. Springer-Verlag, 1992.

[Bee03] R. J. Beerends. *Fourier and Laplace Transforms*. Cambridge University Press, 2003.

[BH05] O. Bournez and E. Hainry. Elementarily Computable Functions Over the Real Numbers and $\mathbb{R}$-Sub-Recursive Functions. *Theoretical Computer Science*, 2005. to appear.

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

[Bra95] M. S. Branicky. Universal Computation and Other Capabilities of Hybrid and Continuous Dynamical Systems. *Theoretical Computer Science*, 138:67–100, 1995.

[BSS89] L. Blum, M. Shub, and S. Smale. On the theory of computation and complexity over the real numbers: NP-completeness, recur-

sive functions and universal machines. *Bulletin of the American Mathematical Society*, 21:1–46, 1989.

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

[CMC02] M. Campagnolo, C. Moore, and J. F. Costa. An analog characterization of the Grzegorczyk hierarchy. *Journal of Complexity*, 18 (4):977–1000, 2002.

[Cop98] B. J. Copeland. Even Turing machines can compute uncomputable functions. In *Unconventional Models of Computation*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer-Verlag, 1998.

[Eda97] A. Edalat. Domains for Computation in Mathematics, Physics and Exact Real Arithmetic. *Bulletin of Symbolic Logic*, 3 (4):401–452, 1997.

[End77] H. B. Enderton. *Elements of Set Theory*. Academic Press, 1977.

[ES98] A. Edalat and P. Sunderhaut. Domains for Computation in Mathematics, Physics and Exact Real Arithmetic. *Theoretical Computer Science*, 210:73–98, 1998.

[Gal06] A. Galton. The Church-Turing thesis: Still valid after all these days? *Applied Mathematics and Computation (Special Issue of Hypercomputation*, 178(1):93–102, 2006.

[GCB05] D. Graça, M. L. Campagnolo, and J. Buescu. Robust simulations of Turing machines with analytic maps and flows. In *Proceedings of CiE'05, New Computational Paradigms*, volume 3526, pages 169–179. Springer-Verlag, 2005.

[Gla67] M. D. Gladstone. A reduction of the recursion scheme. *Journal of Symbolic Logic*, 32:505–508, 1967.

[Gla71] M. D. Gladstone. Simplification of the recursion scheme. *Journal of Symbolic Logic*, 36:653–665, 1971.

[Grz55] A. Grzegorczyk. Computable functionals. *Fund. Math.*, 42:168–202, 1955.

[Ham01] J. D. Hamkins. Supertask Computation. In *Proceedings of the Foundation of the Formal Sciences III, Complexity in Mathematics and Computer Science*, Trends in Logic, pages 1–18. Springer-Verlag, 2001.

[Ham05] J. D. Hamkins. Infinitary Computability with Infinite Time Turing Machines. In *Proceedings of the CiE 2005*, volume 3526 of *Lecture Notes in Computer Science*, pages 180–187. Springer-Verlag, 2005.

[Hen96] T. Henzinger. The Theory of Hybrid Automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science (LICS)*, pages 278–292. IEEE Computer Society Press, 1996.

[HL00] J. D. Hamkins and A. Lewis. Infinite time Turing machines. *The Journal of Symbolic Logic*, 65(2):567–604, 2000.

[Hol96] P. A. Holst. Svein Rosseland and the Oslo Analyser. In *IEEE Annals of the History of Computing*, volume 18(4), pages 16–26, 1996.

[HS01] J. D. Hamkins and D. Seabold. Infinite time Turing machines with only one tape. *Mathematical Logic Quartely*, 47(2):271–287, 2001.

[Kal43] L. Kalmár. Egyzzerü példa eldönthetetlen aritmetikai problémára. *Mate és Fizikai Lapok*, 50:1–23, 1943.

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

[Kel76] W. Thomson (Lord Kelvin). On an instrument for calculating the integral of the product of two given functions. In *Proceedings of Royal Society of London*, volume 24, pages 266–268, 1876.

[Kle55] S. Kleene. Arithmetical predicates and quantifiers. *Transactions American Mathematical Society*, 79:312–340, 1955.

[KM99] P. Koiran and C. Moore. Closed-form analytic maps in one and two dimensions can simulate universal Turing machines. *Theoretical Computer Science*, 210 (1):217–223, 1999.

[Koe05] P. Koepke. Turing computations on ordinals. *The Bulletin Symbolic Logic*, 11(3), 2005.

[Mat93] Y. Matiyasevich. *Hilbert's 10th Problem*. Foundations of Computing Series. The MIT Press, 1993.

[MC04a] J. Mycka and J. F. Costa. Real recursive functions and their hierarchy. *Journal of Complexity*, 20:835–857, 2004.

[MC04b] J. Mycka and J. F. Costa. The computational power of continuous dynamic systems. In *Machines, Computations, and Universality (MCU 2004)*, volume 3354 of *Springer-Verlag*, pages 163–174, 2004.

[MC06] J. Mycka and J. F. Costa. The $P \neq NP$ conjecture in the context of real and complex analysis. *Journal of Complexity*, 22(2):287–303, 2006.

[Min72] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1972.

[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.

[Mos80] Y. Moschovakis. *Descriptive Set Theory*. North-Holland, 1980.

[Myc03] J. Mycka. μ-Recursion and infinite limits. *Theoretical Computer Science*, 302:123–133, 2003.

[Myc05] J. Mycka. Real recursive functions and Baire classes. *Fundamenta Informaticae*, 65 (3):263–278, 2005.

[Odi89] P. Odifreddi. *Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers*, volume 125 of *Studies in Logic & the Foundations of Mathematics*. North-Holland, 1989.

[Ord06] T. Ord. The many forms of hypercomputation. *Applied Mathematics and Computation (Special Issue of Hypercomputation*, 178(1):143–115, 2006.

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

[PE88] M. B. Pour-El. Abstract computability and its relations to the general purpose analog computer. *Advances in Applied Mathematics*, 9:22–34, 1988.

[PE89] M. B. Pour-El. *Computability in Analysis and Physics*. Perspectives in Mathematical Logic. Springer-Verlag, 1989.

[PP04] D. Perrin and J. E. Pin. *Infinite Words*. Pure and Applied Mathematics. Elsevier, 2004.

[Rab03] A. Rabinovich. Automata over Continuous Time. *Theoretical Computer Science*, 300:331–363, 2003.

[Ric79] T. R. Richardson. *Silver machine approach to the constructible universe*. PhD thesis, University of California, Berkeley, 1979.

[Rob47] R. M. Robinson. Primitive recursive functions. *Bulletin of the American Mathematical Society*, 53:925–942, 1947.

[RTY90] J. H. Reif, J. D. Tygar, and A. Yoshida. The computability and complexity of optical beam tracing. In *Proceedings of 31st Annual Symposium on Foundations of Computer Science*, pages 106–114, 1990.

[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.

[Sac90] G. E. Sacks. *Higher Recursion Theory*. Perspectives in Mathematical Logic. Springer-Verlag, 1990.

[Sch03] R. D. Schindler. $P \neq NP$ for infinite time Turing machines. *Monatshefte für Mathematik*, 139:335–340, 2003.

[Sha41] C. Shannon. Mathematical theory of the differential analyser. *Journal Mathematical Physics*, 20:337–354, 1941.

[Sie98] H. Siegelmann. *Neural Networks and Computational Complexity*. Progress in Theoretical Computer Science. Birkhauser Verlag, 1998.

[SS95] H. Siegelmann and E. Sontag. On the computational power of neural nets. *Journal of Computer and Systems Sciences*, 50:132–150, 1995.

[Tra98] B. Trakhtenbrot. Automata and Hybrid Systems. Technical report, Uppsala University, 1998.

[Tra99] B. Trakhtenbrot. Automata and Their Interaction: Definitional Suggestions. *Fundamentals of Computation Theory*, 1684:54–89, 1999.

[Tur39] A. M. Turing. Systems of Logic Based on Ordinals. In *Proceedings London Mathematical Society*, volume 45, pages 161–228, 1939.

[Vre03] A. Vretblad. *Fourier Analysis and Its Applications*, volume 223 of *Graduate Texts in Mathematics*. Springer-Verlag, 2003.

[Wei00] K. Weihrauch. *Computable Analysis: An Introduction*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, 2000.

[Wel00] P. D. Welch. The Length of Infinite Time Turing Machine Computations. *Bulletin of the London Mathematical Society*, 32(2):129–136, 2000.

Index