Bruno Loff

Physics, Computation and Definability

November 6, 2007

Para o Pedro
que me manteve
com eficácia
afastado desta tese

Resumo

A ligação entre física e computabilidade está ainda por esclarecer. Nesta dissertação apresentamos a nossa investigação sobre este tema, incluindo algumas ideas básicas em história e filosofia da computação e novos resultados matemáticos de teorias relacionadas.

A dissertação está dividida em três partes. A primeira parte diz respeito à história da computabilidade e à ligação entre física e computabilidade; discutiremos a natureza do conhecimento físico e as suas possíveis ligações com a computabilidade.

A segunda parte da dissertação é um estudo em *computação experimental*, uma área de investigação esculpida por Edwin Beggs e John Tucker, com o objectivo de investigar a ligação entre física e computabilidade; iremos considerar *experiências* com massas pontuais num mundo Newtoniano abstracto.

A terceira parte desenvolve a teoria das *funções reais recursivas*, originalmente criada para estudar fenómenos e sistemas dinâmicos em tempo contínuo de um ponto de vista computacional; iremos esclarecer quase completamente os aspectos mais gerais desta teoria.

Abstract

The relationship between physics and computability is not well-understood. In this dissertation we will expound our investigation of this relationship, by exploring some basic ideas in history and philosophy of computation and by developing mathematical models that have arisen from related problems.

The dissertation is divided into three parts. The first part is concerned with the history of computability and with the relationship between computability and physics; we will discuss the nature of physics and explore its *possible* connections with computability.

The second part of this dissertation is a case study in *experimental computation*, the conceptual framework that Edwin Beggs and John Tucker proposed in order to investigate the relationship between physics and computability; our *experiments* are made on an idealised Newtonian world for scattering and collecting point particles.

The third part studies the theory of *real recursive functions*, originally devised to study continuous-time phenomena and dynamical systems from a computational perspective; we will develop the more general aspects of the theory nearly to their full extent.

Acknowledgements

My first and most heartfelt *thank you* goes to José Félix Costa, who has taught me most of what I know in computability and complexity. I thank him for being my *primus magnus magister*: my teacher, guide, mentor, adviser, and reluctant friend.

I thank my family and friends for being a good reason to be alive, and for calling me a *Mathematician*, sometimes in such a convincing tone that I almost believe them.

A big word of thanks for everyone in Seminário de Lógica Matemática, to whom I have presented all the work in this thesis. I thank in particular to Mário Edmundo, António Fernandes and Fernando Ferreira for all the patience, help and useful comments they have offered.

For more useful comments, help and encouragement, I thank Edwin Beggs, Olivier Bournez, Carlos Caleiro, Barry Cooper, Martin Davis, Miguel Dionísio, Luis Mendes Gomes, Andrew Hodges, Jerzy Mycka, Kerry Ojakian, Sérgio Pequito, António Rito Silva, Giuseppe Trautteur, and John Tucker.

Preface

The majority of the work below is the result of a rich, exciting, a year and a half long collaboration between myself and José Félix Costa, along with Edwin Beggs, Jerzy Mycka, and John Tucker. The work originated five papers [Loff et al., 2007, Loff, 2007a, Loff and Costa, 2007, Costa et al., 2007, Beggs et al., 2007], one technical report [Loff, 2007b], and several presentations and seminar lessons. Nevertheless, most of the text was written exclusively for the dissertation — I am solely to blame for every fault.

The common link to all this work is the attempt to look at physics with a computability mindset. We will see, however, that we are progressively lead away from physics and into the study of mathematical structure with little physical relevance. This is emphasized by the three-part organisation of the dissertation. In Part I we will deal with those issues which are more directly connected to the interplay between physics, philosophy of science, and computability. Part II is an attempt in studying a physical model from a computability point of view — but several idealising assumptions are made. And in Part III, we study a mathematical model of definability which has some connections, at the basis, with the differential analyser; however, the model will have an exact correspondence with the analytical hierarchy, a highly abstract mathematical structure.

A fair knowledge in computability and complexity is assumed, but this knowledge varies for the different parts. In Part I, we assume some familiarity with computability, most especially Turing machines. Part II, specifically Chapter 7, requires several specialised concepts in structural complexity, but a basic knowledge of complexity, complemented with the small appendix in non-uniform complexity, should provide the necessary concepts. Part III is completely self-contained, except for a few well-known theorems in relativised computability.

Each part may be read without the other two, but it is advised to have read Part I prior to Part II, lest the later loose much of its relevance.

To anyone trying I wish them a good read.

— *Bruno Loff*

Contents

Part I A tool-set for investigating physics and computability

1 The Church–Turing thesis 5
1.1 Historical overview 5
1.2 Church, Turing and Gödel 6
1.3 A final phrasing 9

2 The physical Church–Turing thesis 13
2.1 Phrasing and meaning 13
2.2 Methodology 14

3 The simulation thesis 17
3.1 Phrasing and meaning 17
3.2 Methodology 18
3.3 A brief case-study 18

4 Criticism and motivation 21
4.1 Hypercomputation and academic disdain 21
4.2 The status of computability 22
4.3 The Church–Turing criterion for physics 23

Part II The Scatter Machine

5 Experimental computation 29
5.1 The methodology of Beggs and Tucker 30
5.2 Attaching a controller 31
5.3 Existence vs. construction 34

6 The scatter machine 37
6.1 The physical sub-theories 37
6.2 The equipment 39
6.3 The bisection method 43

7 Computational complexity 47
7.1 The error-free case 47
7.2 The precise error-prone case 52
7.3 Increasing the error 54
7.4 Introducing a constraint 61

8 Examining the results 65
8.1 The plausibility and relevance of the Scatter Machine 65
8.2 A weaker variant of the physical Church–Turing thesis 67
8.3 An unexpected argument 68

Part III Real recursive function theory

9 Introduction to real recursive functions . . . 75
9.1 Function algebras . . . 75
9.2 Weak conditions for existence and uniqueness . . . 79
9.3 The class of real recursive functions . . . 84
9.4 More on differential recursion . . . 85
9.5 More on the supremum limit operator . . . 89

10 Basic theory . . . 91
10.1 A variety of real recursive functions and operators . . . 91
10.2 The η-hierarchy . . . 97
10.3 Non-trivial real recursive operators . . . 98
10.4 Real versus classical recursive functions . . . 103

11 Universality . . . 109
11.1 The Euler method and differential recursion . . . 109
11.2 Totalisation operators and universality . . . 114

12 Understanding REC($\mathbb{R}$) . . . 117
12.1 The analytical hierarchy . . . 117
12.2 Real recursive functions and the analytical hierarchy . . . 120

13 Towards solving the problem of collapse . . . 123
13.1 Unrestricted iteration and stacks . . . 123
13.2 Universal level-bounded functions . . . 125
13.3 Conclusions for the problem of collapse . . . 127

Conclusions, final remarks and criticism 129

Appendix

Brief introduction to non-uniform complexity 131

A.1 The oracle Turing machine 131

A.2 The probabilistic Turing machine 132

A.3 Non-uniform complexity classes 133

Notation and glossary 137

References 139

Part I

A tool-set for investigating physics and computability

Introduction to Part I

There is an obvious connection between computability and physics — computation is often used to simulate and study physical systems, and our knowledge of physics, or lack of thereof, imposes technological bounds on what we can feasibly compute. However, there is also a more profound relationship between physics and computability. Physics, as a discipline, aims to uncover the *rules* which govern the Universe. Computability studies the *rules* of calculation which may effectively be carried out, say, by human beings. However, when we say "rule" in either case, it is not obvious if we are talking about the same thing, i.e., we have no answer to the question:

Are the *rules* governing the Universe anything like the *rules* used in computation?

In Part I we will develop a set of *thinking tools* to study the relationship between computability and physics. These tools are called *the physical Church–Turing thesis* and *the simulation thesis*. They are both variants of the Church–Turing thesis which are appropriately modified to become relevant for physics and the models of the universe. Our approach to this matter is new, despite the questions involved not being so. We will begin by studying the Church–Turing thesis itself in the first chapter, and then dedicate one chapter to each variant. For each variant of the Church–Turing thesis we will provide a carefully chosen phrasing, explain the meaning of this phrasing, and outline a method of investigation. The fourth chapter of Part I provides an assortment of thoughts on computability, non-computability, and physics, as well as motivation for Part II of this dissertation.

Accreditment. Part I arises from the authors own work and ideas, inspired by a previous collaboration with José Félix Costa [Loff and Costa, 2007]. Our main references are Aspray et al. [1985], Gandy [1988], Hodges [2006] and Sieg [2006].

1

The Church–Turing thesis

We will begin our discussion of the Church–Turing thesis with a short historical account. In Section 1.2, we will provide some criteria to answer the question of *what should be called Church–Turing thesis*; we also show how the views of Church and Turing on the idea of effectiveness indicate that an adequate formulation of the Church–Turing thesis is lacking. We suggest such a formulation in Section 1.3.

1.1 A historical overview of the Church–Turing thesis

The years from 1931 to 1936 witnessed the proposal of several mathematical definitions of computability, all independently discovered. Robin Gandy [1988] describes the different reasons that motivated the creation of the λ-calculus, the recursive functions, and the Turing machine. As Gandy puts it, there was 'something in the air' in those years, which different people caught. However, the motivations of those involved where not entirely the same.

The λ-calculus was invented by Alonzo Church, in the 1930s at Princeton. Stephen Kleene and John Rosser worked under Church at that time [cf. Aspray et al., 1985]. Kleene arrived in the fall of 1930, Rosser just a year later, and they both attended Church's logic course in 1931. Church assigned each of them a PhD topic: Kleene was to devise a theory of arithmetic based on Church's *set of postulates for the foundation of logic*,[1] which included λ-calculus, and Rosser was to figure out the connection between λ-calculus and combinatorial logic. There was a rich research environment in Princeton at the time, and from 1931 to 1934 the logic group was visited by John Von Neumann, Haskell Curry, Bertrand Russel, Alfred Whitehead and, in 1934, Kurt Gödel.

Among other subjects, such as his incompleteness results, Gödel lectured on the theory of recursive functions which he had devised in correspondence with Jaques Herbrand, prior to Herbrand's death in 1931. Gödel knew that a precise definition of *effective calculability* would subsequently allow for a rigorous notion of formal system. This was his motivation. Church also realised the importance of such a definition for proof theory, and thought that this formal notion may shed new light on the *Entscheidungsproblem*. The people at Princeton worked, both independently and together, to show that the set of Gödel-Herbrand recursive functions

[1] In the process of their PhD work, Kleene and Rosser have shown that Church's *set of postulates* was inconsistent, i.e., that every theorem was provable.

was exactly the set of λ-definable functions. The results they obtained suggested that the set of λ-definable functions was far greater than previously expected, which encouraged Church to write a letter to Kleene, near February 1934, boldly identifying the class of effectively calculable functions with the class of λ-definable functions. This is the first written account of:

Church's thesis

Every effectively calculable function is λ-definable.

The study of Church's thesis should thus concentrate on the interpretation of *effectively calculable.* Antony Galton [1996, 2006] provides a good example of such an inquiry. In 1936 Church published his article *An unsolvable problem of elementary number theory*, providing the first solution to the *Entscheidungsproblem.* In the same year, Turing [1936] introduces his machines. Turing's main motivation was to find a negative solution for the *Entscheidungsproblem*[Gandy, 1988], and Turing understood that for this it was necessary to precisely map out the limits of computability.

While Turing was probably disappointed for not being the first to negatively solve the *Entscheidungsproblem*, his model of computability had, in the words of Church, *the advantage of making the identification with effectiveness in the ordinary sense evident immediately.* On the other hand, and for those who were not so easily convinced, Turing provided a rigorous analysis of the process of calculation as done by a human computor. His arguments provide a thorough and persuasive defense of:

The human computor thesis

Every function computed by an idealised human being following a systematic procedure can be computed by a Turing machine.

Turing's reasoning was so convincing and insightful that Gandy [1988] calls *Turing's theorem* to the human computor thesis. It stands today as a precious heritage to Computer Science, Mathematics, and Philosophy.

If today the study of computation is more often done from an engineering point of view, it is important to remember that at the heart of this phenomenon lie profound issues of logic, mathematics, cognitive sciences and philosophy. Learning the history of computability puts us in contact with these problems, and allows us to better appreciate the elegance and beauty of the solutions which were put forward.

1.2 Church, Turing and Gödel on the idea of effectiveness

The human computor thesis, as formulated in the previous section, is frequently called *Church–Turing thesis* [e.g., in Sicard and Vélez, 1999, Copeland, 2002, Shagrir and Pitowsky, 2003, Ben-Amram, 2005]. In this section we will attempt to answer the following questions:

(1) Why should a Church–Turing thesis exist? Why not only Church's thesis?
(2) What form should the Church–Turing thesis have?

(3) Is the human computor thesis a good phrasing for the Church–Turing thesis?

The term *Church–Turing thesis* was coined by Kleene, to name the identification of effectively calculable function with Turing machine computability. Since Turing machine computability and λ-definability are provably equivalent, Kleene's formulation of the Church–Turing thesis is essentially Church's thesis, and it is hard to justify having such a formulation, except for the (non discardable) fact that it is Stephen Kleene's formulation. On the other hand, the existence of a Church–Turing thesis serves as a recognition of the historical importance of both Church's thesis, an example of brave ingenuity, and Turing's work, which made Church's thesis extremely credible; so we do not wish to simply discard the Church–Turing thesis. Church's thesis also has the additional problem that the notion of *effectively calculable* is extremely vague, and has many possible different meanings. We will thus propose a formulation which will narrow down the breadth of possible meanings of *effectively calculable* into a notion which depends less on personal interpretation and whim, and more on the understanding that the relevant historical figures had of their own thesis. These are the reasons for having a Church–Turing thesis, and for phrasing it as we will now propose.

In most formulations, the Church–Turing and related theses follow the pattern of identifying a more-or-less vaguely defined class of functions with the functions given by some standard model of computability:

Every function in the vaguely defined class X
is computable according to the model Y.

In Church's thesis, for instance, *class X* is the class of *effectively calculable functions*, and *model Y* is the λ-calculus. We will call this the **standard form**, and will refer to the vaguely defined class X as the **proposed class**.

We believe that the Church–Turing thesis should be stated in the standard form using Turing machines as the model (Y) of computability:

Every function in the vaguely defined class X
can be computed by a Turing machine.

The Turing machine is chosen for historical and pedagogical motives. It was the first model of computability for which representation theorem could be made, i.e., a rigorous, convincing argument to support the appropriateness of the notion. In fact, an exposition of Turing's model can make it intuitively evident that *computability* mimics the process of systematic calculation, and Turing's rigorous argument can confirm this intuition to near certainty. This will lead to an easier and more meaningful introduction to computability and to the Church–Turing thesis.

In the author's conviction, the proposed class (class X) should be our best historical approximation of:

- Church's concept of *effectively calculable function*, and
- Turing's concept of *function which would naturally be regarded as computable.*

These are the terms that Church and Turing used to name their intuitive notion of *computable*. The challenge is to find out what Church and Turing actually thought when they used these terms, and use the result of this research in the formulation of the Church–Turing thesis. With this discussion we have answered question (2). So we come to the third question, which can now be reformulated as:

(3b) Is the concept of function computed by an idealised human being following a systematic procedure a good historical approximation of Church and Turing's intuitive notion of computable function?

As may be expected, this is a difficult question: neither Church nor Turing seem to have given an explicit answer. It is fortunate that Andrew Hodges, Turing's biographer, has considered this very question in his [2006] article. He arguments that there is historical evidence to believe that neither Church nor Turing made a sharp distinction between a function which may be systematically computed by a human, and a function which may be computed by an arbitrary finite mechanism. Hodges begins by noting Church's review of Turing's [1936], where Church writes:

> *[Turing] proposes as a criterion that an infinite sequence of digits 0 and 1 be "computable" that it shall be possible to devise a computing machine, occupying a finite space and with working parts of finite size, which will write down the sequence to any desired number of terms if allowed to run for a sufficiently long time. As a matter of convenience, certain further restrictions are imposed in the character of the machine, but these are of such a nature as obviously to cause no loss of generality — in particular, a human calculator, provided with pencil and paper and explicit instructions, can be regarded as a kind of Turing machine.*

At a first glance Church's review seems bizarre: Turing makes no mention of occupied *space*, or of the *size* of any *parts*. Turing's argument in defense of his computing machine model clearly refers to human computors, who *write certain symbols on paper*, and who have *states of mind*. Church attributes to Turing an analysis of computability by finite machines, and the fact that Turing refers only to human computors is described as a *restriction imposed in the character of the machine* which causes, to Church, *no loss of generality*. Hodges also makes note of his review of Emil Post's article, where Church writes:

> *To define effectiveness as computability by an arbitrary machine, subject to restrictions of finiteness, would seem an adequate representation of the ordinary notion [...].*

Again we see effectiveness as finite mechanism, rather than human calculation. This provides evidence that effective computability was something that Alonzo Church identified with computability by a mechanism which is subject to restrictions of finiteness. Unfortunately, we do not appear to have such strong confirmation in Turing's case. However, Hodges [2006] gives us the following clues:

> *If Turing had regarded it [Church's perspective on the work of Turing] as seriously misrepresenting his ideas, he would not have been deterred from saying so by Church's seniority. He was shy socially but very confident of his own judgment in all sorts of*

matters. [...] But he recorded no dissent. If Turing had wished politely and properly to distance himself from Church's version of his definition, and re-assert his own, he had the opportunity in his 1938 doctoral thesis [...]. Yet in [the later published version of his thesis], when giving his own statement of the Church–Turing thesis, he simply characterized a computable function as one whose values can be found 'by some purely mechanical process', saying that this may be interpreted as 'one which could be carried out by a machine.'

On this basis, and if we wish to formulate the Church–Turing thesis as we have proposed, then we must conclude that the human computor thesis is not an appropriate phrasing. The next section will be dedicated to finding such a phrasing, and to develop on its meaning. As a concluding remark, it is interesting to note that Gödel also described Turing's work as an analysis of finite mechanisms [Sieg, 2006]. Here is a quotation from an unpublished essay (Collected Works III, p. 168):

[Turing] has shown that the computable functions defined [via the equational calculus] are exactly those for which you can construct a machine with a finite number of parts which will do the following thing. If you write down any number $n_1, \ldots, n_r$ on a slip of paper and put the slip of paper into the machine and turn the crank, then after a finite number of turns the machine will stop and the value of the function for the argument $n_1, \ldots, n_r$ will be printed on the paper.

1.3 Proposal for a final phrasing

In this section we will propose a specific formulation for the Church–Turing thesis, based on the criteria which were developed in the previous section. The formulation will then be explained at length, with several examples, so that the meaning becomes clearer. The proposed phrasing is the following:

The Church–Turing thesis
Every function computed by a finite mechanism
can be computed by a Turing machine.

The thesis is in the standard form, and the chosen model of computability is the Turing machine, for the reasons presented in the previous section. The presented class is the class of functions which are computed by some finite mechanism. We must develop two ideas. First, what is a finite mechanism, and second, what it means for a function to be computed by a finite mechanism. Gandy [1980] studies this question in depth: he provides a sensibly intuitive formalisation of the notion of computing mechanism, and shows that any device which obeys certain formal axioms (which he calls *principles for mechanisms*) can be simulated by a Turing machine. With this effort, Gandy separates the Church–Turing thesis into (1) a theorem asserting that any device obeying certain principles can compute no more than a Turing machine and (2) a thesis stating that every mechanism obeys the mentioned principles. This effort in formalisation will not be repeated here, for reasons which will be made clear in Section 4.2. Instead of offering a possibly controversial precise formalisation of *finite mechanism*, we will attempt to give a consensual, broad idea of what a finite mechanism should and should not be.[2]

The notion of mechanism should be understood in two senses: either *a device composed of parts*, or *a process by which something takes place*. We demand that a finite mechanism follows two principles: a **principle of finiteness** and a **principle of self-containment**.

A mechanism adheres to the principle of finiteness if (A) it produces the desired result in a finite number of steps, and (B) its entire behaviour can be specified by a finite description. This is a very broad principle, but it does come across as both permissive enough to include the necessary cases, like Turing machines, and strict enough to exclude uninteresting counter-examples, such as infinite circuits, or infinite time Turing machines. The nature of the "description" which specifies the behaviour of the mechanism is left intentionally vague, as it is difficult to give a satisfactory notion of specification without making use of a good definition of computability, which would, for the Church–Turing thesis, result in a vicious circle (cf. Section 5.3 and Galton [1996]). It is nevertheless important to observe that it is the *behaviour* of the mechanism which must be described, and not its *functioning*. Specifying the behaviour of a mechanism implies that we can always infer how the mechanism will react, in a given specific situation, from observing the specification. The difference can be illustrated by noticing that while we can describe the functioning of a Turing machine with any oracle (the machine *queries* the oracle, and then goes to one of two special states), we can not necessarily describe its behaviour (to which state the machine will actually go).

A mechanism follows the principle of self-containment if it acts with no insight and accesses no external entity during its functioning. This principle spares the Church–Turing thesis of more uninteresting counter-examples, such as the Turing machine with an oracle. Without this principle, one could have devices which, in spite of having a finitely describable behaviour, would make use of an external entity which might not have such a restriction.

We may also describe two principles which should be adhered by any process intended to compute a function: the **input-output principle** and the **principle of repeatability**.

The input-output principle gives us the basic setting of what it means to compute a function f. This principle demands that (A) there must be a way of introducing an input n and starting the computation, (B) a way of extracting an output m after the computation is completed, and (C) one can be certain, with an arbitrarily low probability of error, that $m = f(n)$.[3] We may give a trivial process which respects the input-output principle: we write a number in a

[2] Let it be noted that Gandy [1980] calls *thesis M* to the thesis identifying mechanism computability with Turing computability, and would probably criticise our formulation of the Church–Turing thesis. His reasons for doing so are outlined in [Hodges, 2006]. Our formulation is the most appropriate *only if we accept the criteria established in the previous section.*

[3] Note that when f is undefined for n, the computation may never be completed. Note also that this principle makes no mention as to *how* we can be certain that $m = f(n)$, i.e., how we verify the correctness of the computation.

sheet of paper, and put it in the box; then we remove the sheet from the box, and read the number. This amusing process may be seen as the computation of the identity function.

The principle of repeatability is our way of ensuring that not every accidental spewing of numbers can be called computation. This principle imposes that (A) the mechanism which carries out the computation can, in practice or in principle, carry it out an unlimited number of times with the same input-output behaviour, and (B) if two different instances of this mechanism are given the same input, they will produce the same output. The principle of repeatability effectively rules out the idea, occasionally expressed, that a (e.g. quantum) random number generator is computing a function when it outputs a requested amount of random data. Recycling the previous example of the sheet of paper in the box, see that we can repeat the process on the same box as often as we like, and still be computing the identity function; we also expect the same results if we use a different box.

The four principles here presented should be consensual, including for Church and Turing, since (1) the failing of any of the principles of finiteness, self-containment, or repeatability would result in trivial counterexamples to the Church–Turing thesis, and (2) there seems to be no real or abstract finite self-contained mechanism, devised with the purpose of computing a function, which does not comply with the input-output principle. These four principles are not meant to be final, or definitive. They are only tentative guidelines which serve to find common ground for the discussion and teaching of the Church–Turing thesis. The author believes that the instruction of these guidelines and of the Church–Turing thesis would add notable significance to the study of computability.

2

The physical Church–Turing thesis

The first tool which we will present to investigate the relationship between computability and physics is a variation of the Church–Turing thesis called *physical Church–Turing thesis*. We will begin by proposing a formulation for this thesis, and explain its meaning. We will then describe a rigorous methodology for investigation. A complete and extensive application of this methodology will be shown in Part II.

2.1 Phrasing and meaning

The physical Church–Turing thesis pertains to the class of functions which may be computed, in some way, by a physical system. We propose the following accurate formulation, in the standard form:

The physical Church–Turing thesis
Every function computed by a finite physical system
can be computed by a Turing machine

The proposed class of functions is the class of functions computed by finite physical systems. These computations must obey the input-output and repeatability principles which were explained in page 10. Here, a physical system is an *abstract process or device* described by a specified physical theory. We do not mean *physical system* as those which actually exist in the world: instead, we consider it to be a process or device which *could happen or exist* according to a specified physical theory (we may attribute various levels of realism to the systems we wish to study, cf. Section 5.3). We demand that this system follows a principle of finiteness, which is not exactly the same as the principle of finiteness that we presented for mechanisms. While we previously required that the behaviour of the computer could be *entirely specified* by a finite description, now we are only interested that its behaviour follows from *postulated* laws of a physical theory. Succinctly, we are only interested in *what* the system does, and not in *how* the system accomplishes what it does. Finiteness will be demanded in a very broad sense, meaning that each computation of the system must be performed in finite time (for the interested observer, and only for the values at which the function to be computed is defined), using only finite energy, occupying finite volume, *etc*. Without these restrictions, it would be easy to find systems computing more than a Turing machine.

The physical Church–Turing thesis can not be studied without the assumption that there *is* a real world, and the statement of the physical Church–Turing thesis is pertinent to the real world. The physical Church–Turing thesis *does not* state that every physical system described by every physical theory can not compute more than the Turing machine (we will see that this is not true in Part II). Instead, the physical Church–Turing thesis asserts that if a physical theory correctly explains what happens in reality, then — and only then — every function calculated by the systems described in the theory is computable. In this sense the physical Church–Turing thesis can be used as an argument in favour or against certain phenomena in physical theories [see, e.g., Nielsen, 1997]. Such an argument will be attempted in Part II, Section 8.3.

2.2 Methodology

We wish to investigate the physical Church–Turing thesis in order to better understand the physical relevance of the concept of computability. This study may be done on several levels of detail, and in several ways. We will below describe a methodology to research the physical Church–Turing thesis in the negative sense, i.e., by investigating candidate counter-examples. One may proceed differently, e.g., by showing that every system described by a physical theory can be simulated by a Turing machine [e.g. Smith, 1999]. The more general methodology of Edwin Beggs and John Tucker, intended for the study of experimental computation, will be presented in Section 5.1.

The method is composed of three steps. The **first step** is used for specification. We begin by defining a physical theory and maybe a sub-theory, to study. Examples of such theories are Newtonian mechanics, or Maxwell's electromagnetism. We could restrict such theories to sub-theories, i.e., restricted classes of problems where the laws of the more general theory still apply. Examples of such sub-theories are, respectively, a classical theory of particle collision, or optics. The definition of the physical theory can be made in various levels of detail: from an *ad-hoc* enumeration of physical laws or desired physical properties up to, when possible, full and precise axiomatics of the physical model.

After establishing the physical theory, we specify the physical system. Examples of physical systems are idealised billiards, or optical processors. Again, one can vary the detail of specification of such systems, but after the specification one must clearly explain how the input should be inserted into the system, and how the output should be extracted from the system, according to the input-output principle. This will be harder in some cases, such as in the case of idealised billiards, where the input may be the initial position and momentum of a ball and the output could be the number of times that the ball hit the rails. In other cases this will be trivial, such as in optical processors, where the design of the system already specifies how the input and output are done.

In the **second step** of the investigation we explain how the system specified up to this point serves as a candidate counter-example to the physical Church–Turing thesis. We describe how and to what extend does the system specified in the first step compute more than the Turing machine. Such an analysis may be with respect only to computability, or may also involve notions of computational complexity.

We conclude with a **third step**, which is the most important part of the research. After determining the computing system and how its computing power surpasses that of a Turing machine, we must investigate the source of non-computability, to understand *why* the system computes more than a Turing machine, and *what does it mean*. If it was possible to point out specific physical laws which allow for non-computability, then one should inquire if these postulates are still believed in, or if they are outdated, and consider if these laws are an essential part of the theory, or if one can find equally appropriate alternatives which remove the super-Turing effect. Such an investigation of the physical Church–Turing thesis may thus be used as a guide for physics: we will explore this idea further in Section 4.3. It is also important to reflect on what this discovery means for computability, if the super-Turing system is similar to known formalisms, such as infinite automata or oracle Turing machines, or if, on the other hand, suggests new interesting constructions (e.g., the work of Balcázar et al. [1997] on neural networks, suggesting a new hierarchy between P and P/*poly*).

This methodology is not meant to guide the discovery process, but proposes instead a uniform way to present results on the subject of the physical Church–Turing thesis. This is similar to the axiomatic method. The work which is described using the axiomatic method is usually not discovered in the same order as it is presented: to discover mathematical theorems one does not begin by defining the axioms and then mindlessly proving the theorems; the process is more complicated [Rota, 2006, Thurston, 2006]. Most studies regarding the physical Church–Turing thesis follow these steps, although the manner of presentation is often unclear and incomplete. Above all, this methodology offers a way to organize how we think about the results related to the physical Church–Turing thesis.

3

The simulation thesis

The second important variant of the Church–Turing thesis which we will study is the *simulation thesis*. We will begin in the same way as the previous chapter, by proposing and explaining a particular formulation of the simulation thesis, and by suggesting a methodology for research. We add a third section with a brief application of this methodology.

3.1 Phrasing and meaning

The simulation thesis is very similar to the physical Church–Turing thesis, with a subtle difference. The following formulation is proposed:

The simulation thesis
Every function which simulates a finite physical system
can be computed by a Turing machine.

An explicit effort has been made to keep the standard form, which will force us into explaining what is meant by a function which simulates a physical system. We could have written *every finite physical system can be simulated by a Turing machine,* but this would leave us with the similar problem of explaining how such a simulation should proceed.

We say that a function f simulates a physical system if it answers a question regarding the evolution of the system, when the initial conditions are given as input. We demand that (A) the question pertains to a deterministic event which follows from the initial conditions of the system, according to the considered physical theory, and that (B) this question can be answered by an observer somewhere in the system after finite time (for that observer). An example is the function which, given the dimensions, masses, positions and velocities describing an idealised billiard with sixteen balls, is valued 1 if after ten seconds any of the balls entered into any of the holes and is valued 0 otherwise.[1] This idea of *simulation as question answering* is the basis for maintaining the standard form in the presented formulation of the simulation thesis.

In a similar way to the physical Church–Turing thesis, the simulation thesis does not assert that every function simulating an arbitrary physical system can be computed by a Turing machine. Instead, the simulation thesis makes the proviso that the physical theory describing the system to be simulated gives a correct description of reality.

[1] Here, idealised billiard is an abstract version of an actual billiards table, with four sides and six holes. It is not the more complex billiard ball model of Fredkin and Toffoli [1982].

So how is the simulation thesis any different than the Church–Turing thesis? First of all, the simulation thesis is *stronger*, since it implies the physical Church–Turing thesis. But it can be the case that the physical Church–Turing thesis is true, while the simulation thesis remains false. These two theses are different because it is not necessary that all computational power of nature can be harnessed in a useful way: even considering that every function computed by a finite physical process is computable, we may still believe that the working of the universe can not be entirely mimicked by a Turing machine, and that although we *can not exploit* this non-computability, it nevertheless *can be observed* in some way. Toby Ord [2002] gives an extremely clear account of the difference between these two theses.[2]

3.2 Methodology

We now describe a methodology to investigate the simulation thesis. We will be brief, for this methodology is similar to the one presented in Section 2.2. We should also begin by a **first step** of specification. We determine the physical theory and sub-theory under consideration, and establish the finite physical system to be studied. We choose a question pertaining to a deterministic event which can be answered by an observer in the system after finite time.

We then move to the **second step**, where it is shown that the question can not be answered by a Turing machine. It is helpful, in this phase, attempting to identify which particular property of the phenomenon leads to this undecidability. Finally, in the **third step**, we must scrutinise both the physical theory and the significance of the obtained result for computability and physics.

3.3 A brief case-study

We will show an application of the above method, in a review of Warren Smith's [2006a] work.[3] Smith studies the Newtonian theory of gravitation for point masses in an Euclidean plane. The particles possess a certain position, velocity and mass. In a system with N bodies, their positions $\mathbf{x}_1, \ldots, \mathbf{x}_N$ obey the law of motion:

$$\ddot{\mathbf{x}}_i = G \sum_{j \neq i}^{N} m_j \frac{\mathbf{x}_j - \mathbf{x}_i}{\|\mathbf{x}_j - \mathbf{x}_i\|^3}.$$

He considers the particular system with a fixed number N of bodies, each with specific masses given by certain rational numbers. The initial conditions of such a system can be specified by a tuple of $4N$ real numbers — two real numbers for the position vector and two for the velocity vector, times N particles.

[2] Ord calls *mathematically harnessable* to a physical process computing a function. He was unaware of the historical point of view of Hodges [2006].

[3] [Loff and Costa, 2007, Loff, 2007b] survey further of work on the violation of the simulation thesis.

Smith's results tell us that there is no general algorithm which, given the initial conditions,[4] is able to decide the question: *Does any particle in the system intersect the unit ball centered at the origin in the first second of the system's evolution?* Such a question clearly satisfies our proviso, since the system is deterministic, and by observing the middle of the system one would be able to answer this question after one second. Smith [2006a] writes:

> *The present paper demonstrates (I claim) that unsimulable physical systems exist in Newton's laws of gravity and motion for point masses. However, it does not appear to demonstrate, that, if we lived in a universe governed by those laws, we could actually build a device with super-Turing computational power.*

We can assume that if the point mass does intersect the unit ball in the system then it will also intersect $B(0, 1 - \epsilon)$ for some predetermined $0 < \epsilon < 1$, i.e., Smith's undecidability result does not stem from the fact that the particles may intersect the unit ball arbitrarily near the perimeter. The non-computability of this example arises from the existence of non-collision singularities in Newtonian dynamics [see Xia, 1992], and of the inability to decide whether the initial conditions result, or do not result, in a singularity. The Newtonian system is not strictly finite, since the occupied area goes to infinity in finite time. Smith's study is exceptionally complete: after his exploration of Newtonian dynamics, Smith considers laws of gravitation which include relativistic phenomena (such as limited velocity), showing that under these laws one can *efficiently* simulate the N-body problem. This establishes Smith's work as a great contribution of evidence in favour of the simulation thesis and the physical Church–Turing thesis, when concerning celestial dynamics [for a similar result on quantum mechanics, see Smith, 1999].

The simulation thesis is the accurate statement that we live in a clockwork universe, a wildly controversial idea which some take as an obvious falsity and others as a necessary truth. Since it is the strongest thesis of its kind, the simulation thesis has tremendous implications, such as the strong AI thesis. A definite positive settlement of the simulation thesis requires an equally definitive physical theory, and we are thus led to believe that such a resolution will not happen. It is a major, open question whether the paradigms of modern physics describe simulable phenomena: a positive answer would give one reasonable argument for believing the simulation thesis.

[4] According to the conventions of computable analysis. Such initial conditions can be given, e.g., by writing the binary expansion of the $4N$ real numbers in an infinite tape.

4 Criticism and motivation

We have thus far answered *how* the study of the physical Church–Turing thesis and the simulation thesis may proceed. We have yet not explained, and that is the purpose of this chapter, *why* such a study should be conducted.

4.1 Hypercomputation and academic disdain

There are several written accounts criticising the relevance of the so-called hypercomputation [e.g. Davis, 2006a,b, Hodges, 1999]. We feel these may have inadvertently discouraged the study of a field with a wealth of philosophical and historical consequence. Some researchers of hypercomputation occasionally display a publicitary attitude, brandishing a flag of *enormous impact for computer science*. This has sometimes, and perhaps rightly, caused severe irritation and dissent, but it is the author's opinion that it should not be a matter of judgement for or against the field of hypercomputation.

We propose a different perspective of this discipline, that one should regard the studies of hypercomputation and non-computability as an investigation of the two mentioned variants of the Church–Turing thesis. This is a different point of view which describes the field of hypercomputation as *the in-depth study of physical theories in order to provide evidence for or against the physical Church–Turing thesis and the simulation thesis.*

It was with this goal in mind that we have studied the Church–Turing thesis and its variants, provided methods for research, and overviewed work on hypercomputation, in [Loff and Costa, 2007, Loff, 2007b]. We hope that this work will clear the way for the edification of a new discipline, and that it may serve as a *thinking tool* for physics and computability. The questions remain, of what is the interest of having such a tool, and why should the physical Church–Turing and the simulation theses be given any credit as a thinking tool for physics.

One reason is that these two theses are technologically and physically relevant. Evidence in favour of the theses provides a philosophical foundation, e.g., for automatic theorem provers, or physics simulation software; the substantiation of the physical Church–Turing thesis and the simulation thesis confirms that these fields use the best possible tool — the general purpose computer — and that every effort to learn how to use it better is worthwhile. On the other hand, evidence against these two theses suggests that it may be fruitful to dedicate some effort in developing new tools.[1]

We will study in Section 4.3 how we may use these variants of the Church–Turing thesis to guide research in physics. The reason that these two theses should be given any credit at all is that the mathematical definition of computability has acquired an impressive status. We will discuss this in the next section.

4.2 The surprising status of the concept of *computability*

Gian-Carlo Rota, in his article entitled *The pernicious influence of mathematics upon philosophy* [Rota, 2006], harshly criticises mathematised philosophy. He defends that the use of mathematical methods to study philosophical problems is a misguided endeavour, which impoverishes philosophy. Rota writes that human history supports the idea that there is no definite solution to philosophical problems, and that philosophy is thus destined to *failure* in a certain sense. This failure is pointed out by Rota as an essential part of philosophy, and that imprecision of definition is an important characteristic of philosophical undertaking.

The author agrees with Rota's view, and feels that it provides good reason against taking the Church–Turing thesis in the stipulative sense, i.e., we should not consider the Church–Turing thesis as a *definition* of function computed by a finite mechanism, and should maintain a certain degree of imprecision in the notion of finite mechanism. On the other hand, the Church–Turing thesis states exactly that itself would be a correct precise definition of the concept of function computed by a finite mechanism, and this statement is very plausible, *per* Turing's [1936] reasoning and other arguments since Turing [e.g., Gandy, 1980, 1988, Sieg, 2007]. So computability appears to be an exception to Rota's objections on the formalisation of philosophy. Gödel expressed this idea beautifully, before Princeton's 1946 conference on mathematics (Collected Works II, p.150):

> *[...] with [the concept of computability] one has for the first time succeeded in giving an absolute definition of an interesting epistemological notion, i.e., one not depending on the formalism chosen.*

This is surprising! It is not at all obvious that such a vague concept (effectively calculable, or mechanically computable) can be precisely captured. First, in an age of developing technology and massive knowledge, it may be expected that new mechanisms can be made to calculate new, different things, or that new logical principles of calculation are discovered. Second, it is usually easy to find examples of insufficiency in any attempted formalisation of a complex philosophical notion, e.g., the counter-examples studied by the philosophy of language against formal linguistics, or the blatant difference between the *emotions* of software agents and human emotional processes.[2] Notice that Rota did not mean to say that informal rigour — the pursuit of precise characterisation to common notions — is impossible: there are plenty of intuitive notions, such as circle, or area, which have been precisely formulated [cf. Kreisel,

[1] To this effect, Toby Ord's [2002, 2006] defined the notion of *physically harnessable* process: a process which, although it can not be said to compute a function, is used to simulate another, different, physical system.

[2] There are, as we have seen, attempts at formalising the Church–Turing thesis [see Gandy, 1980, Sieg, 2007, Boker and Dershowitz, 2006].

1987]. Rota's arguments are against the formalisation of *philosophical concepts*, because it is natural and expected (and good) that these concepts are complex, constantly evolving, and will always remain somewhat diffuse. It is thus unforeseen that the concept of mechanical computability can be so precisely represented, and the Church–Turing thesis is the embodiment of the hypothesis that it can.

Thus the Church–Turing thesis sets computability in a special, unique relationship with the intricate and confusing world of philosophical ideas. How far can we extend this status? Is the formal notion of computability just a singular aberration, or coincidence, pertaining to epistemology, or does it actually describe a true and profound limitation of the *real world*? These questions are the main motivating force behind parts I and II of this text. In the previous chapters we have developed some tools which will give us a few, non definitive, answers.

4.3 The Church–Turing criterion for physics

The experimental method is the most important, and perhaps the only scientific tool which can confirm the correctness of a physical theory, or tell us which of several modern theories is the most correct. Unfortunately, it is often difficult to *find* the experimental setting which will settle that some theory of physics is more appropriate over another.

Even when such experiments can be devised, they may not always be carried out, due to lack of funding, facilities, *etc.* It thus becomes important to *guide* research in physics, and consider only the most promising theory. The problem is that there is no equivalent to the experimental method which allows us to choose *the most promising theory.* So the direction of scientific research is usually guided by heuristics, beliefs and opinions which the scientist leaders, e.g., the lab head, or the Nobel laureate, have of which research topic is the most relevant. These heuristics, despite being extremely valuable, are occasionally somewhat arbitrary.[3]

Thus it would be good to have at our disposal certain guiding criteria, for the research in physics, which offer explicit and more precise motives to choose investigating one physical theory over the other, or consider a certain type of experiments instead of another. We present one such criterion.

The Church–Turing criterion for physics

If a physical theory violates the physical Church–Turing thesis or the simulation thesis, then some part or the whole of this theory is incorrect.

This is not to be understood as a thesis or as a principle, i.e., there is to be no explicit attempt to show that the Church–Turing criterion for physics is true. It is to be seen as a standard, an additional tool which may be used to guide the research in physics.

To the question of *how* the criterion may be used, we have answered by giving two methods for such an investigation. As to *why* should such criteria be used, we have explained in this section. Some motives for accepting the Church–Turing criterion as a valid criterion for

[3] The reader may disagree. In any case, decision makers usually do not know or tell which part of their heuristics is subjective and which part is not.

guiding the research in physics have been given in the previous section: the importance of the formal definition of computability, the sheer surprise that computability can be formalised, and the strong belief that computability is, indeed, completely characterised by the current formal definition, all these lead us to believe that the Church–Turing criterion for physics is an appropriate tool.

There is, however, another subtle reason to accept the Church–Turing criterion, a reason which is specifically adequate for physics. Physics, as a discipline, concerns the fundamental *laws* of the Universe, it aims to uncover the *rules* which govern the constituents of reality. The physicist may achieve, for this endeavour, two distinct levels of success: he may succeed at *describing the laws*, and he may additionally *understand the rules.*[4] The physicist will have *described the laws* of the Universe if he has given a full account of, say, the equations which govern the elementary components of reality. He will have *understood the rules* of the Universe when he can, at least in principle, solve these equations and predict the behaviour of these components. This second, higher level of success can only be accomplished, or we currently so believe, if these rules are computable by a Turing machine. A teacher and mentor of the author wrote [Costa, 2007]:

> *A non-computable science would be more like a painting in the National Gallery — to look at with respect, admiration, and fascination, but although interpreted by many, not really being interpreted by none besides digressions and elucubrations of the critics.*

And so, the physicist who believes that such a second level of understanding can be achieved should guide his research with the help of the Church–Turing criterion.

[4] At this level of discussion we speak *in principle*, that is, we do not mean to say that physics only succeeds when it has entirely explained reality, but that this is physics Utopian goal. Speaking in these simplistic terms ignores several problems in the philosophy of science, but consciously so.

Part II

The Scatter Machine:
A complete case-study of experimental computation

Introduction to Part II

Experimental computation is the study of how we can make computations through physical systems and experiments. This issue has recently become a target for controversy, because of the discovery that several physical systems seem to behave in a non-computable way [e.g. Pour-El and Richards, 1981, Hogarth, 1994, Siegelmann, 1999, Kieu, 2002]. These discoveries have been criticised for bearing little relevance to what one can compute in reality, or for being riddled with faults and imprecision [e.g., respectively, Weihrauch and Zhong, 2002, Shagrir and Pitowsky, 2003, Davis, 2006a, Smith, 2006b]. It can be said, at least, that some of these studies are not clear about the incurred goals, methods and implications.

A notable exception, which stands out for the clarity of purpose, technique and consequence, is the work of Edwin Beggs and John Tucker [2004, 2006, 2007a, 2007b]. They offer what may be the first clear exposition of what is *experimental computation*, what is the purpose of studying experimental computation, how it should be investigated and what is the expected value of such an investigation. Their methodology provides a foundation for improving our understanding of the link between physics and computation.

We have presented, in Section 2.2, a methodology to investigate counter-examples to the physical Church–Turing thesis. This methodology was devised independently from Beggs and Tucker, and can be seen as a more specific and simple version of their methods. We will give a complete account of their methodology in Chapter 5, along with other considerations on the nature of experimental procedures and experimental equipment.

The rest of Part II is dedicated to applying the methodology of Section 2.2. Chapters 6, 7 and 8 concern each of the three steps of our methodology. While we structure our work as specified in Section 2.2, we will also be careful to satisfy the additional requirements of Beggs and Tucker's methodology. The majority of results in these chapters come from a collaboration between the author, Edwin Beggs, José Félix Costa and John Tucker [Beggs et al., 2007].

Accreditment. The structural idea of Part II is due to José Félix Costa. Chapter 5 is our own presentation of the work of Edwin Beggs and John Tucker, and a pursuit of some ideas deriving from their work. Chapter 6 is a formalisation of the scatter machine of Beggs and Tucker, and the variations on the bisection method are derived from ideas and proofs by John von Neumann [1956], Hava Siegelmann [1999] and Eduardo Sontag, brought to the attention of the author by José Félix Costa. Chapter 7 also benefits from the same ideas and proofs, but adds a few of our own (most notably the proofs of Propositions 7.6 and 7.11, concocted after a long a discussion with José Félix Costa); the style of presentation attempts to follow that of Balcázar, Díaz and Gabarró [1988]; Chapter 8 is solely due to the author.

5

Experimental computation

We call **experimental computation** to the idea of computation as a result of **experimental procedures** carried out on a physical **equipment**. We could say that experimental computation begins with the advent of calculators, such as the Chinese abacus or Leibniz's multiplying machine [see Williams, 1997], since their invention surely entails a certain amount of *experimentation.* However, this should not be called experimental computation, because while experimentation was required to invent the devices, the computations themselves involve no actual experiment. We can instead consider that experimental computation started with the differential analyser. In this machine, the input is represented by physical quantities, which are then processed by a number of contraptions — the equipment —, and the output is obtained by measurement of physical quantities [cf. Nyce, 1994].

Beggs and Tucker [2007a] present a number of very satisfying methods and considerations for the study experimental computation. Their methods can be seen as a more general and detailed version of those presented in Chapter 2, with the following differences:

1. Their methodology is more general since it may be used to study any instance of experimental computation, and it is not exclusively concerned with finding counter-examples to the physical Church–Turing thesis.
2. They propose that one should avoid studying a full-fledged physical theory, but should instead attempt at finding the *minimal sub-theory* which allows for experimental computation.
3. When studying a theory T, Beggs and Tucker add that we should also consider the experimental domain of the theory, $E(T)$, comprising of postulates, laws, phenomena, and situations where T has been experimentally confirmed. We should discern whether and which experimental computations are still possible if we consider ONLY $E(T)$.

Besides their methodology, Beggs and Tucker contributed to the well-founding of the study of experimental computation by their clear discussion of related problems and issues [Beggs and Tucker, 2004, 2006, 2007b]. We will begin by fully describing their methodology in Section 5.1. In Section 5.2 we discuss their guidelines on developing a language to specify experimental procedures, and give a mathematically precise version of these guidelines by showing how to couple the experimental equipment to a Turing machine. Finally, in Section 5.3 we make some considerations on the construction of equipment for experimental computation.

5.1 The methodology of Beggs and Tucker

We now describe Beggs and Tucker's methodology for the investigation of experimental computations. In this section we occasionally paraphrase the article [Beggs and Tucker, 2007a]. The purpose of this methodology is to obtain a coherent unambiguous discussion of the computational power of physical systems described by a physical sub-theory.[1] This purpose is fulfilled by satisfying the following goals:

(1) Precisely define the physical sub-theory in question;
(2) Establish, by an explicit, exact construction, which physical systems that are models of the sub-theory are to be regarded as equipment;
(3) Explain and prove, with utmost clarity and rigour, which are the properties of this equipment;
(4) Specify experimental procedures to make use of these properties;
(5) Show that the procedures allow for some form computation;[2]
(6) Understand the source of the computational power;
(7) Clarify the relationship between computability and physics; and
(8) Make clear, precise and detailed statements about the whole process.

If a hypercomputer was found, then one also wishes to:

(a) Evaluate the physical credibility of the system, or, possibly,
(b) Reveal weaknesses in the theory.

These goals are achieved in four stages of investigation. The first two stages roughly correspond, respectively, to the first two steps described in Section 2.2, and the last two stages are analogous to step three.

The first stage of investigation is intended to satisfy goal (1), and it is called **defining a physical sub-theory**. This stage is carried out by starting with a general physical theory — which may be quite large, such as Newtonian mechanics — and pinpointing those properties of the theory that can be used in experimental computation. We then form a sub-theory, which will be denoted by T, containing only these properties. We may, for instance, study in detail a few examples with computational properties, and then attempt to find the smallest fragment of the physical theory which is needed to specify and study these examples.

By limiting our investigation to a physical sub-theory, it should, in the latter stages, be easier to understand which postulates of the more general theory are responsible for the computational power of the physical systems. Another benefit is that conclusions which we derive for a restricted sub-theory are expected to generalise to the theory as a whole.

It should be emphasized that *it does not matter whether we think of the theory as true, roughly applicable, or know it to be false.* A rigorous study of the computational properties of the theory should give a better understanding of the relationship between (non-)computability

[1] The [2005] article by Udi Boker and Nachum Dershowitz offer a valuable discussion of how we can compare computational power among different models of computation.
[2] E.g., by showing that these procedures obey the input-output and repeatability principles in order to compute a function.

and physics, and of the scope of applicability of the theory itself. Such a study may also help us investigating other theories, in which we deposit more belief.

Goals (2) to (5) are pursued in the second stage of the methodology, which is called **classifying computers in the physical sub-theory**. In this stage we search for ways of embedding functions, sets, logical formulae, algorithms, programs, computers and hypercomputers into the physical systems described by T. Several candidates for equipment and experimental procedure may arise; we choose those which we find more relevant and appropriate. Above all, the chosen systems must be explained, and their properties must be demonstrated, with the most possible clarity, rigour and care. If certain points can not be made completely clear, perhaps by the difficult nature of the theory or of the system, the faults of clarity must be fully exposed. This should ensure that criticism to the constructed model of experimental computation will be of a profound nature, rather than being the result of unfortunate misinterpretation, as sometimes happens in the hypercomputation literature.

In the third stage we must analyse the computations of the equipments and procedures studied in the second stage, in order to understand which properties or laws of the sub-theory lead to different computational powers. We should also seek necessary and sufficient conditions for the systems of T to implement exactly the Turing-computable functions. This stage is called **mapping the border between different computational powers in the sub-theory**, and it is concerned with accomplishing goal (6), and part of goal (7).

The fourth and final stage, concerning goals (7), (a) and (b), is called **reviewing and refining the physical theory**. In this stage we determine the plausibility of our physical systems. It is not possible to mathematically prove that the sub-theory T is correct, but we can obtain experimental validation of a restricted part of T. This part of T is expected to be more faithful to experimental results, and we call it the **experimental domain** of T, denoted $E(T)$. If our physical systems are also models of $E(T)$, then we expect that their existence is plausible. Notice that this means that the systems *could exist* in the real world, not that they could actually be built. The difference will be emphasized in Section 5.3.

While studying the plausibility of our physical systems, one usually finds more credible or interesting variants of the equipment and experimental procedures, and new ways of refining the sub-theory into something more realistic. These discoveries should be exposed as openly and clearly as possible. An important part of the whole methodology, and of this stage of investigation in particular, is to form a portfolio of systems and sub-theories with different computational properties. This portfolio will serve as a guide to future work, and should allow for a better understanding of which physical properties result in computational power.

5.2 Attaching a controller

In their [2007a], Beggs and Tucker give some guidelines on how to develop a language for specifying experimental procedures. The idea is that to each equipment which is a model of a sub-theory T corresponds an enumerable set of experimental **actions** that can be *performed* on the equipment, and an enumerable set of **responses** that can be *measured* from the equipment.

One may then use a procedural language to specify the experimental procedures governing the computation. The experimental procedures would be described by programs, as in Figure 5.1. These programs can perform experimental actions, measure the equipment's responses and make any necessary calculations.

```
Perform action $A_1$.
If measured response $R_1$ then do something,
else if measured response $R_2$ then do something else
```

Fig. 5.1. A program specifying an experimental procedure

We will now show how these guidelines can be made completely precise, by attaching a Turing machine to control the equipment. We assume that the equipment can perform experimental actions from an enumerable set A, and react with measurable responses from an enumerable set R. The action/response behaviour of the equipment may not be deterministic, and may depend on the actions which where previously performed and on the responses which where previously obtained.

Thus, the behaviour of the equipment is represented by a certain stochastic process M. M assigns, to each sequence $e_1 \dots e_n$ of events that happened in the equipment, the probability that a certain response r is measured when performing some action a. This probability we denote by $\mathbb{P}(M(e_1 \dots e_n a) = r)$. An *event* can be either a commanded action in A, or the absence of an action, which we have represented with the symbol $\bigcirc$. Thus, M is a stochastic process indexed in the set

$$H = \{e_1 \dots e_n a : (e_i \in A \vee e_i = \bigcirc) \wedge a \in A\}$$

of sequences of events ending with an action. For each $e_1 \dots e_n a \in H$, $M(e_1 \dots e_n a)$ is a discrete random variable with values in R, such that

$$\sum_{r \in R} \mathbb{P}(M(e_1 \dots e_n a) = r) = 1.$$

The equipment is called deterministic if for each $e_1 \dots e_n a \in H$ there is only one r with $\mathbb{P}(M(e_1 \dots e_n a) = r) > 0$, and is called probabilistic otherwise.

The link between the Turing machine and the equipment is made in a way similar to the oracle Turing machine, which is described in the appendix. We assume that the set of actions and the set of responses are indexed with the strings of the working alphabet Σ^*, i.e., $A = \{a_w : w \in \Sigma^*\}$ and $R = \{r_w : w \in \Sigma^*\}$. The coupled Turing machine $\mathcal{M}$ has two special states, called the **action state** and the **response state**, and two special tapes, called the **action tape** and the **response tape**.

The Turing machine $\mathcal{M}$ is said to **command an action** a_w on the equipment, when it enters the action state with the string w written on the action tape. In order to perform action a_w, $\mathcal{M}$ begins by commanding the action. The Turing machine computation is then interrupted, and the equipment performs the action a_w. If the response of the system is r_v, then the response tape is erased, v is written on it, and $\mathcal{M}$ resumes its computation in the response state. $\mathcal{M}$ is then said to have **measured response** r_v.

The sequence of events $e_1 \dots e_t$ mentioned above is obtained by the history of commanded actions. If $\mathcal{M}$ commands action a at time step $t+1$ of the computation, then each event e_i in $e_1 \dots e_t$ is given by:

1. $e_i = a_w$ if $\mathcal{M}$ commanded the action a_w at time step i, and
2. $e_i = \bigcirc$ if $\mathcal{M}$ commanded no action at time step i.

The probability of $\mathcal{M}$ measuring the response r at time $t+2$ is $\mathbb{P}(M(e_1 \dots e_t a) = r)$.

Example 5.1. We take the equipment M to be a balanced coin. The set of actions will then be $A = \{\mathit{toss}\}$ and the set of responses is $R = \{\mathit{heads}, \mathit{tails}\}$. Then $H = \{e_1 \dots e_n a : (e_i = \mathit{toss} \vee e_i = \bigcirc) \wedge a = \mathit{toss}\}$, and we set

$$\mathbb{P}(M(e_1 \dots e_i a) = \mathit{heads}) = \mathbb{P}(M(e_1 \dots e_i a) = \mathit{tails}) = \frac{1}{2}.$$

A Turing machine coupled to such an equipment is computationally equivalent to a probabilistic Turing machine, which is described in the Appendix. □

Three things should be noted on our way of formalising the link between the Turing machine and the equipment.

First of all, the characterisation of the equipment is very general, but it is assumed that the behaviour of the equipment depends only on the actions and the number of computational steps between actions. This is a reasonable assumption to make of any equipment to be used in experimentation.

Secondly, the behaviour of the equipment does, indeed, depend on the number of steps between actions. This is intended to model the use of clocks in controlling experiments — a chemical experiment, for instance, must be carried out with precise timings. The way that the number of Turing machine steps relates to the number of time units for the equipment must be clearly specified, and may vary for each considered equipment. We could say, for instance, that each step of the Turing machine takes exactly 10^{-6} seconds. It may even be the case that the Turing machine becomes an inadequate model to control certain types of equipment, because in order to command the action a_w, the Turing machine requires at least $|w|+1$ steps of computation.

Thirdly, in the case where the number of actions (or responses) is finite, it is appropriate to replace the action tape (resp. response tape) with a finite number of additional special states, one for each action (resp. response).

A full computation of a Turing machine with equipment on a finite string x begins in the same way as for a normal Turing machine, and the input x is said to be accepted or rejected, as in the Turing machine, if the computation halts, respectively, in the accepting or rejecting state.

We may then establish a decision criterion for the Turing machine coupled with deterministic equipment. A set B of finite strings is decided by such a machine $\mathcal{M}$ if, for every input $w \in \Sigma^*$,

w is accepted if $w \in B$ and rejected when $w \notin B$. We say that $\mathcal{M}$ decides B in polynomial time, if $\mathcal{M}$ decides B, and for every $w \in \Sigma^*$ the number of steps in the computation is polynomial in the size of w.

However, the decision criterion for a Turing machine coupled with probabilistic equipment must be different, because the machine will not, in general, deterministically accept or reject the input. Several criteria could be put forward, and we will here describe one of the most common probabilistic decision criterion. For a set $B \subseteq \Sigma^*$, a Turing machine $\mathcal{M}$ attached to a probabilistic equipment, and an input $w \in \Sigma^*$, the **error probability** of $\mathcal{M}$ for input w is the probability of $\mathcal{M}$ rejecting w if $w \in B$, or the probability of $\mathcal{M}$ accepting w if $w \notin B$. We say that $\mathcal{M}$ decides B with **bounded error probability** if there is a number $\gamma < \frac{1}{2}$, such that the error probability of $\mathcal{M}$ for any input w is smaller than γ. B is decided in polynomial time if, for every input x, the number of steps in the possible computations is always polynomial in the length of x.

And so ends our treatment of this matter. We will make use of this type of computational model in the next chapter, when we study a specific equipment called the scatter machine.

5.3 Existence vs. construction

Up to this point in our discussion, the physical systems that form the equipment are only required to *possibly exist*, either according to an abstract sub-theory or according to its experimental domain. Seen in this way, an equipment is a black box obeying the postulates of the physical sub-theory, and exists only in an abstract, idealised world.

The fact that such a system may be conceived in this way could already be relevant for the philosophy of science, theoretical physics, or computation, but when studying experimental computation one may aim at technological relevance. It is important, in this case, to have a method for building the equipment.

Beggs and Tucker [2007a] study this problem and offer some ideas on how a *construction language* can be devised in order to specify the construction of equipment. Such a construction language would make use of a number of primitive specifications, such as the placement of parts, or the generation of certain physical fields. It is important that these primitive specifications are known to be satisfiable by current technology, if one does not wish to conceal the problem of building the device into the problem of satisfying the specification. The construction language then allows for a finite combination of these primitive specifications in a way similar to a programming language, and this combination of primitive specifications forms the specification of the equipment.

It is expected that from a complete specification of the experimental equipment one can, for every number n, obtain a finite number of primitive specifications which must be satisfied in order to build an equipment that can perform experimental computation, say, on every input of size n. This is the case with the computing machines of today (if we where to regard them as equipment).

When we consider construction languages, similar to programming languages, that specify methods for constructing computing equipment, we must be careful in noticing the circularity which we have introduced. Suppose we are studying a hypercomputer that we have shown to be in the experimental domain of some physical sub-theory. Unless an error was committed, this means that a hypercomputer can exist in the universe. But can it be built? It may even be the case that a sequence of primitive specifications will allow us to build such a hypercomputer, but that this sequence is not computable by a Turing machine. Thus, if our construction language is similar to current programming languages, it would be inadequate to specify the construction of our device.

The problem is, in essence, that it is not possible to give a satisfactory notion of *specification* without having an adequate notion of *computability*, and this means trouble if we wish to specify devices which may have a higher computational power than the Turing machine. Three solutions for this problem come to mind:

1. *The problem does not exist.* If there is actually no sequence of primitive specifications which will allow for the construction of the equipment using current technology, then the problem becomes irrelevant.
2. *The equipment already exists*, and it can be used to control the construction of other similar equipments.[3]
3. *Such a hypercomputer is useless.* It is natural that any new, more powerful machine has to be constructed using the currently available computational power, and that if it can not be so constructed, then the machine is really a ghost hypercomputer, which *could exist*, but never will.

We will thus define four levels of realism which can be attributed to an equipment for experimental computation. For the weakest level of realism, we say that an equipment **exists in imagination**, if it is a model of a theory of physics. We further say that it can **exist in theory**, if it is a model of a physical theory in which we believe. An equipment will be said to **exist in practice**, if it belongs to the experimental domain of a physical theory. For the stronger level of realism, we will say of an equipment that it can **exist in actuality**, if it may be built using current technology.

This classification may be used to describe the plausibility or technological relevance of the equipment under study. In the following chapters, we will study an equipment that only *exists in imagination*, and provide a complete analysis of its computational power. It may intrigue the reader why we should be bothered to study an equipment which is not physically plausible. We will provide a few reasons in Section 8.1.

[3] This is not as absurd as it seems, if we adopt the view, believed by some, that the human brain is hypercomputational [Penrose, 1989, 1994, Bringsjord and Zenzen, 2003].

6

The scatter machine

We now introduce the scatter machine, a physical device to shoot and collect particles in a Newtonian world. We dedicate each of the three sections in this chapter to the presentation of the physical sub-theory, the equipment and the experimental method.

6.1 The physical sub-theories

We will consider a very simple and idealised sub-theory T of Newtonian mechanics, concerning the elastic collision of point particles against rigid barriers that are laid out in the two-dimensional Euclidean plane. Each system of T is a finite collection of

1. barriers, where the particles bounce,
2. particle detectors, in the form of rectangular holes where the particles fall, and
3. cannons, which shoot particles, and can be moved along an axis.

Plenty of idealising assumptions are made, regarding the systems of T:

A. Barriers can be placed between any two points in the Euclidean plane.
B. The barriers do not move, i.e., their extremities remain fixed.
C. The barriers are rigid and perfectly straight.
D. Particles are dimensionless.
E. There is no friction affecting the movement of the particles.
F. Particles can be flawlessly detected.
G. Every particle has the same, infinitely precise mass.
H. Collisions are perfectly elastic, but particles are transparent to cannons and other particles.
I. Cannons can shoot a particle with an infinitely precise velocity, at any infinitely precise point in time.
J. Cannons can be placed at any point along their axes.

We will see that the systems of T can be used as idealised equipment in order to perform hypercomputation, but more importantly we will show in the next section that the assumptions C to J can be removed or made more realistic, and that the *only* assumptions which must remain unaltered to allow for the existence of a hypercomputer are A and B.

The sub-theory T can be easily and conveniently formalised, in the following way. T is the class of systems of the form $\langle B, D, C\rangle$, where B is the set of barriers, D is the set of particle detectors and C is the set of cannons.

Each **barrier** $b \in B$ is formalised as a pair $b = \langle \mathbf{x}, \mathbf{y}\rangle$ of points in the Euclidean plane. The barrier b will **occupy** the points between $\mathbf{x}$ and $\mathbf{y}$, *excluding* y. We denote the set of points occupied by b as $occ(b)$, given by $occ(b) = \{(1-z)\mathbf{x} + z\mathbf{y} : z \in [0,1)\}$. The particles of the system reflect perpendicularly to the line from $\mathbf{x}$ to $\mathbf{y}$. We formalise this by saying that b **reflects** towards $ref(b)$, which is a unit vector perpendicular to $occ(b)$. If R represents the linear operator of rotation in $\frac{\pi}{2}$ radians, then $ref(b)$ is expressed by $ref(b) = R(\frac{\mathbf{y}-\mathbf{x}}{\|\mathbf{y}-\mathbf{x}\|})$.

Each **particle detector** $d \in D$ is represented by a closed, bounded interval $d = [x_1, x_2] \times [y_1, y_2]$ of the Euclidean plane. A particle is said to have been **detected** by d if its position lies inside this interval.

A **cannon** $c \in C$ is also formalised as a pair $c = \langle \mathbf{x}, \mathbf{y}\rangle$ of points in the euclidean plane. The cannon c can be placed at any point between $\mathbf{x}$ and $\mathbf{y}$, this time *including* y, and we call **axis** to the set of these points. More precisely, we denote the axis of c by $axis(c)$, given by $axis(c) = \{(1-z)\mathbf{x} + z\mathbf{y} : z \in [0,1]\}$. The cannon c is said to be **aiming** along a direction perpendicular to its axis. This direction is represented by a vector, $aim(c)$, with the expression $aim(c) = R(\frac{\mathbf{y}-\mathbf{x}}{\|\mathbf{y}-\mathbf{x}\|})$. We will sometimes refer to the points in the axis of c using numbers between 0 and 1. For each such number $z \in [0,1]$, we refer to point $(1-z)\mathbf{x} + z\mathbf{y}$ as **point** z **along the axis of** c.

We demand that no barrier, particle detector or cannon axis intersect at any point, i.e., if A is $occ(b_1), d_1$ or $axis(c_1)$ and B is $occ(b_2), d_2$ or $axis(c_2)$ for any barriers b_1, b_2, particle detectors d_1, d_2 or cannons c_1, c_2, then $A \cap B = \varnothing$.

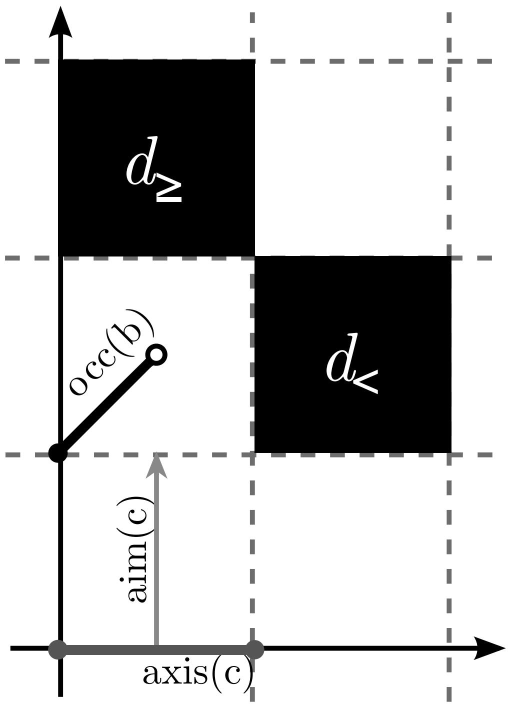


Fig. 6.1. A schematic drawing of a system for particle collisions.

Example 6.1. Figure 6.1 is a drawing of the system $S_1 = \langle \{b\}, \{d_<, d_\geqslant\}, \{c\} \rangle$, with the barrier $b = \langle (0,1), (\frac{1}{2}, \frac{3}{2}) \rangle$, the detectors $d_< = [1,2] \times [1,2]$ and $d_\geqslant = [0,1] \times [2,3]$, and the cannon $c = \langle (0,0), (1,0) \rangle$. The vector $aim(c)$ is positioned at point $\frac{1}{2}$ along the axis of c. □

One may interact with the systems of T by commanding the cannons to shoot particles, and by observing when a particle is detected. Consider the system $S = \langle B, D, C \rangle$. At each point in time there will be a number of particles moving in the system. The particles are introduced by ordering a cannon $c \in C$ to shoot a particle, from a point $\mathbf{a} \in axis(c)$, with a certain scalar velocity V, and at a certain instant $t_0 \in \mathbb{R}_0^+$. The cannon then shoots a particle which will remain in the system until it is detected by a particle detector $d \in D$. Take δ to be Dirac's delta function, and $\cdot$ to represent the scalar product. Then the position $\mathbf{x}$ and velocity $\mathbf{v}$ of such a particle are governed, component-wise, by the equations of movement:

$$\mathbf{x}(t_0) = \mathbf{a} \qquad \partial_t \mathbf{x}(t) = \mathbf{v}(t),$$

$$\mathbf{v}(t_0) = V \times aim(c) \qquad \partial_t \mathbf{v}(t) = -\sum_{b \in B} \sum_{o \in occ(b)} 2\left(\delta(\|x(t) - o\|) \times \mathbf{v}(t) \cdot \mathit{ref}(b)\right) \mathit{ref}(b),$$

According to these equations, a particle hitting a barrier b will preserve the velocity component parallel to $occ(b)$, and invert the component parallel to $\mathit{ref}(b)$.

Example 6.2. Suppose that at time $t = 0$ one commands the cannon c of the system S_1 in Example 6.1 to fire from point z along its axis, with speed 1. We can see that the particle will fall on one of the detectors at time $t = 2$, and that it will fall on $d_\geqslant$ if $z \geqslant \frac{1}{2}$, and will fall on $d_<$ if $z < \frac{1}{2}$. □

The sub-theory has a few imperfections, which we will not amend. These are:

1. Should a particle collide in parallel with a certain barrier, the particle will go through it.
2. Particles do not bounce on cannons, nor on other particles.
3. We impose a specific behaviour for a particle hitting the edge of a barrier, which is an atypical situation in the study of collisions.

We will ignore these fallbacks because they are irrelevant for the systems that we will consider. In these systems, particles will never be in parallel with barriers, nor will the collision with other particles be of any interest, and the case when a particle hits the edge of a barrier will also be made irrelevant.[1] In order to understand why this is the case, and why we may drop assumptions C to J above, we will need to introduce the specific systems of T to be used as equipment.

6.2 The equipment

We will use as equipment a specific family of systems indexed in the interval $[0,1]$, $\{S_x\}_{x \in [0,1]}$. The systems in this family are called **scatter machines**. A scatter machine is a system with

[1] However, if we considered particle collisions, the sub-theory T would become adequate to describe the computing billiards of Edward Fredkin and Tommaso Toffoli [1982].

two barriers, one cannon and two particle detectors. The cannon is facing the barriers, and these form a right angle, called the *wedge*. The vertex of the wedge is aligned with a certain position x along the cannon's axis. One of the particle detectors is to the right of the barriers and the other is to the left. Formally, $S_x = \langle\{b_l, b_r\}, \{d_l, d_r\}, \{c\}\rangle$, with $b_l = \langle(3+x, 1), (1+x, 3)\rangle$, $b_r = \langle(5+x, 3), (3+x, 1)\rangle$, $d_l = [0,1] \times [0,3]$, $d_r = [6,7] \times [0,3]$, and $c = \langle(3,0), (4,0)\rangle$. This is schematised in Fig. 6.2.

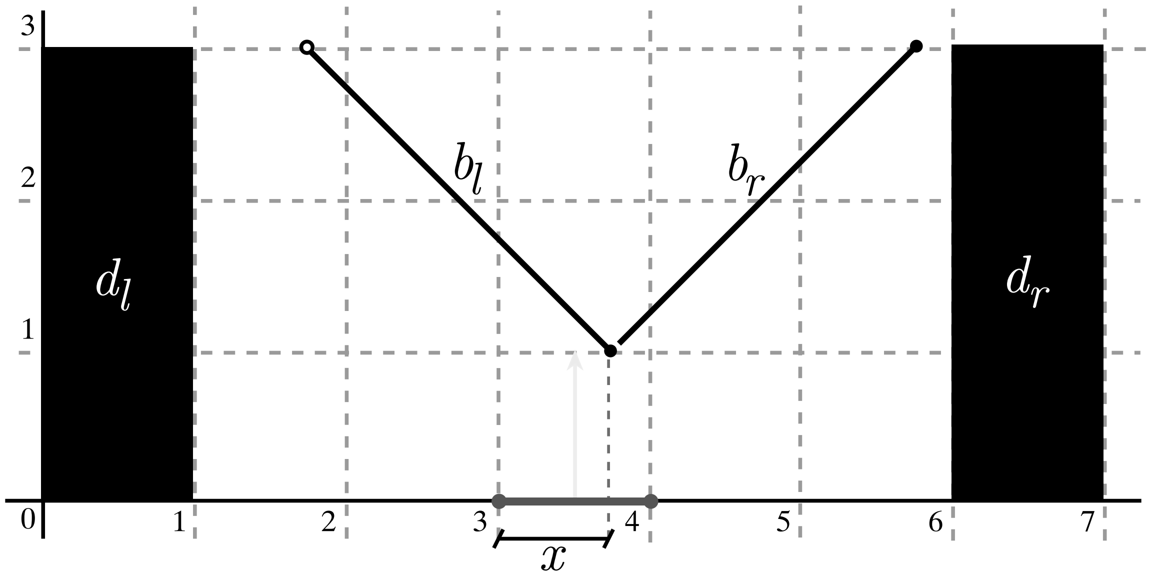


Fig. 6.2. A schematic drawing of a scatter machine.

We can now formalise the equipment as a stochastic process, in the way we have explained in Section 5.2. Each scatter machine S_x will allow for a number of actions from a countable set A. Each action in A commands the cannon of S_x to fire from a specific dyadic rational point along the cannon's axis.[2] We thus enumerate the actions in A using the word 1 and the binary words beginning with 0: $A = \{a_1\} \cup \{a_{0s} : s \in \{0,1\}^*\}$. The action $a_{s_1 \ldots s_n}$ in A commands the cannon to fire from the point

$$\sum_{k=1}^{n} 2^{-k+1} s_k \in [0,1]$$

along the cannon's axis, with some predefined fixed speed.[3]

After the command, the cannon *attempts* to position itself at this point,[4] and shoots a particle with some predetermined scalar velocity. It is clear that the particle will always hit the wedge, either in the left or the right barrier. Our formalisation is such that the point $(3+x, 1)$ is occupied by the barrier b_l, and thus the particle will go left when it hits the vertex.

[2] The set of dyadic rationals is the subset of rational numbers with a finite binary expansion. Notice that we can correspond each dyadic rational with a finite binary sequence. The sequence "0" corresponds to the number 0, "1" corresponds to the number 1, and a sequence "0s", where "s" is a binary sequence, corresponds to the number 0.s, to be interpreted in the binary expansion.

[3] Note that we allow for $s_1 \ldots s_n$ to end with a 0, and so each action a_z, where z is a dyadic rational, is indexed by an infinite number of sequences $z, z0, z00, \ldots$.

[4] Remember that shooting at point $z \in [0,1]$ along the cannon's axis will, for a scatter machine, actually mean shooting from the point $(3+z, 0) \in axis(c)$.

We will see, however that the particle will never hit the vertex, or that it will do so with probability 0, and we can safely disregard this case.

The equipment can thus respond in one of two ways. After some time, either a particle was detected by the right detector, or it was detected by the left detector. So, the set R of responses is given by $R = \{\textit{right}, \textit{left}\}$, with the obvious meaning. We can now understand why the assumptions C to I in the previous section matter little for the behaviour of the system. It is clear that the only distinguishing mechanism which must be respected is whether the particle goes left, or right, after hitting the wedge. This property remains more or less the same independently of assumptions C to I, as long as the collisions are *sufficiently* elastic, the barriers are *reasonably* straight, and provided that there is *not too much* friction, *etc.*[5] The final assumption (J), however, seems to make a crucial difference. The behaviour of the stochastic process which we will use to model the scatter machine will depend on whether or not the cannon can be placed exactly where the action commanded, but we will show, in the next chapter, that each case will result in a hypercomputer.

Example 6.3. If we make the assumption J of the previous section, then the machine S_x can be used to decide whether a dyadic rational $y \in [0, 1]$ is smaller or equal to x. In order to decide this, we perform action a_y, where y is represented as a word in $\{1\} \cup \{0s : s \in \{0, 1\}^*\}$. If we measure the response *left*, we then see that $y \leqslant x$, and we know that $y > x$ when we measure the response *right*. □

We will consider three weaker versions of assumption J, which will give us three variations of the sub-theory T. For the first variant of T, T_1, we will suppose that the cannon can be placed with infinite precision at any *dyadic rational* point along its axis. For the remaining two variants, T_2 and T_3, we assume that we can command the cannon to be placed along any dyadic rational point along its axis, but that there is a certain dyadic rational error in the placement, denoted by ε. Specifically, we will consider that if we command the cannon to be placed at the dyadic rational $y \in [0, 1]$, then the cannon will in fact be placed somewhere in the interval $[y - \varepsilon, y + \varepsilon]$, with a uniform probability distribution. For the theory T_2, the error of the placement is controllable: to the command a_y corresponds the error $\varepsilon = 2^{-|y|-1}$. Here, $|y|$ is taken to be the size of the sequence used to index a_y. This implies that in the sub-theory T_2 the action a_y is different for different binary representations of the dyadic rational y. Finally, in the theory T_3, we will assume that the error of placement ε is a fixed value, taken to be $\varepsilon = \frac{1}{2}$ in this dissertation.

We are now ready to complete our formalisation. We call **error-free** to the scatter machines modeled by T_1. An error-free scatter machine S_x is a deterministic equipment, governed by the probability distribution:

$$\mathbb{P}(S_x(e_1 \dots e_n a_y) = \textit{left}) = \begin{cases} 1 & \text{if } y \leqslant x, \\ 0 & \text{if } y > x. \end{cases}$$

[5] See Beggs and Tucker [2007b] for other considerations, such as particles with different shapes.

The probability of measuring a *right* response can be inferred, since the sum of the probabilities over the set of responses must be 1. We will call **error-prone** to the scatter machines modeled by T_2 and T_3. The models of T_2 will be called **precise** and the models of T_3 will be called **imprecise**. An error-prone scatter machine S_x is a probabilistic equipment, governed by:

$$\mathbb{P}(S_x(e_1 \dots e_n a_y) = \mathit{left}) = \begin{cases} 1 & \text{if } y < x - \varepsilon, \\ \frac{1}{2} + \frac{x-y}{2\varepsilon} & \text{if } x - \varepsilon \leqslant y \leqslant x + \varepsilon, \\ 0 & \text{if } y > x + \varepsilon \end{cases}$$

The error ε will be $2^{-|y|-1}$ for precise error-prone scatter machines and will be the fixed value $\varepsilon = \frac{1}{2}$ for imprecise error-prone scatter machines. The stochastic process governing the behaviour of the scatter machines will always be a Markov chain, since the response of new experiments will never depend on previous outcomes. But, further than that, the response to a new action also does not depend on previous actions. So we always have, for any $r_1, \dots, r_n \in \{\mathit{left}, \mathit{right}\}$:

$$\mathbb{P}(S_x(\dots a_1 \dots a_{n-1} \dots a_n) = r_n | S_x(\dots a_1 \dots a_{n-1}) = r_{n-1}, \dots, S_x(\dots a_1) = r_1) = \mathbb{P}(S_x(a_n) = r_n).$$

The dots in the above expression stand for the possible events of type $\bigcirc$.

In Section 5.2, we have shown how to connect a Turing machine to an arbitrary equipment. By connecting it to a scatter machine, we obtain a theoretical computation device, which we call the **analog-digital scatter machine**. We will use $\mathcal{S}_x$ to designate an analog-digital scatter machine obtained by coupling S_x with a Turing machine. Depending on which assumptions are made on the placement of the cannon, the analog-digital scatter machines will also be called error-free or error-prone, precise or imprecise. The device works as described in Section 5.2. Actions are preformed by writing a sequence $y \in \{1\} \cup \{0s : s \in \{0,1\}^*\}$ on the action tape, and entering the action state. We will refer to the action state of an analog-digital scatter machine as **shooting state**. The coupled Turing machine is then interrupted, and the scatter machine performs action a_y. The response is signaled to the coupled Turing machine by resuming the computation either in the **left state** or in the **right state**, with the appropriate meaning.

We repeat the decision criteria.

Definition 6.4. *Let $A \subseteq \Sigma^*$ be a set of words over Σ. We say that an error-free analog-digital scatter machine $\mathcal{S}_x$ **decides** A if, for every input $w \in \Sigma^*$, w is accepted if $w \in A$ and rejected when $w \notin A$. We say that $\mathcal{S}_x$ **decides** A **in polynomial time**, if $\mathcal{S}_x$ decides A, and there is a polynomial p such that, for every $w \in \Sigma^*$, the number of steps of the computation is bounded by $p(|w|)$.*

For a set $A \subseteq \Sigma^*$, an error-prone analog-digital scatter machine $\mathcal{S}_x$, and an input $w \in \Sigma^*$, let the **error probability** of $\mathcal{S}_x$ for input w be either the probability of $\mathcal{S}_x$ rejecting w, if $w \in A$, or the probability of $\mathcal{S}_x$ accepting w, if $w \notin A$.

Definition 6.5. *Let $A \subseteq \Sigma^*$ be a set of words over Σ. We say that an error-prone analog-digital scatter machine $\mathcal{S}_x$* ***decides*** *A with bounded error probability if there is a number $\gamma < \frac{1}{2}$, such that the error probability of $\mathcal{S}_x$ for any input w is smaller than γ. We call* ***correct*** *to those computations which correctly accept or reject the input. We say that $\mathcal{S}_x$* ***decides*** *A* ***in polynomial time****, if $\mathcal{S}_x$ decides A, and there is a polynomial p such that, for every input $w \in \Sigma^*$, the number of steps in every correct computation is bounded by $p(|w|)$.*

Standard proof techniques of structural complexity can be used to show that if there is an error-prone analog-digital scatter machine $\mathcal{S}_x$ which decides A in polynomial time with an error probability bounded by $\gamma < \frac{1}{2}$, then, for any polynomial p, there is another error-prone machine which decides A in polynomial time with an error probability, for inputs of size n, bounded by $\frac{1}{2^{p(n)}}$. We may also assume, without loss of generality, that if an error-prone analog-digital scatter machine $\mathcal{S}_x$ decides a set A in polynomial time, then all of its computations halt after the same number of steps [cf. Balcázar et al., 1988, Chapter 6].

6.3 The bisection method

We have laid out the relevant physical sub-theories and the equipment. We now present three experimental procedures, one for each sub-theory, which allow us to extract information from a scatter machine. The first of these experimental procedures was introduced by Beggs and Tucker [2007b]. The procedure consists of a sequence of bisections, and it is thus called the **bisection method**. It allows us to obtain, using an error-free analog-digital scatter machine $\mathcal{S}_x$, any number of digits of the binary expansion of x.

We can safely assume that x is not a dyadic rational, because if this is the case then we can trivially construct an analog-digital scatter machine that outputs every digit in the binary expansion of x, by embedding this (finite) expansion into the finite control of the machine.

The method can be carried out by an error-free analog-digital scatter machine using three work tapes (plus the action tape). The first tape is used for general rough work and the remaining two work tapes store two values, a and b, which we will simultaneously regard as dyadic rationals and as words in $\{1\} \cup \{0s : s \in \{0,1\}^*\}$. The bisection method begins by setting $a = 0$ and $b = 1$. Then, at each stage of the method, the cannon is fired at point $c = \frac{a+b}{2}$. If the *left* response was measured, then on the next stage we set $a = c$. If, on the other hand, we measured the *right* response, then we set $b = c$ on the next stage of the method.

Throughout the procedure, we always have $a \leqslant x \leqslant b$, and $|b - a|$ is halved at each stage. Thus, in order to obtain n digits of the binary expansion of x, we carry out $n + 1$ stages of the bisection method. After $n + 1$ stages, we have that $0 \leqslant |x - a| \leqslant |b - a| = 2^{-n-1}$. Since x is assumed not to be a dyadic rational, its binary expansion is unique, and so we see that $a \upharpoonright_n = x \upharpoonright_n$, i.e., the first n digits of the binary expansion of a are equal to the first n digits of the binary expansion of x.

At each stage of the bisection method one must calculate c, write it in the action tape, and copy it to one of the two work tapes keeping a and b. These three operations are done in linear time on the size of a and b. At stage n, the size of a and b is at most n, and so we conclude that it takes $\mathrm{O}(n^2)$ steps to obtain n digits of x.

The bisection method will not work on error-prone machines, but it turns out that for precise error-prone analog-digital scatter machines we can use a simple variant of the bisection method, which we will call the **modified bisection method**. This second method can also be used by an error-free machine.

We use an analog-digital scatter machine $\mathcal{S}_x$ with three tapes, as before, two of which store a and b. Again, we begin by setting $a = 0$ and $b = 1$, at stage 1. Then at each stage we calculate $c = \frac{a+b}{2}$, and command the cannon to shoot at point c. Now we must make sure that at the k-th stage the dyadic rational c is represented with a word of size at least $k + 1$. If, after the cannon being shot, we got a left hit, then a will be set to $\frac{a+c}{2}$, and if the we got a right hit, then b will be set to $\frac{b+c}{2}$. This means that at each stage we still update a or b to become closer to c, but now we do it more slowly.

Notice that by commanding the cannon to shoot at point c in its axis, and by specifying c with at least $k + 1$ bits, we ensure that the error in placing the cannon is at most $\varepsilon_k = 2^{-k-2}$. This means that the particle will be shot from somewhere in the interval $[c - \varepsilon_k, c + \varepsilon_k]$. Thus, if the particle bounces to the left, we can be sure that x is not anywhere in the interval $[a, c - \varepsilon_k)$, and if it bounces to the right we are certain that x is not in $(c + \varepsilon_k, b]$. Now, at the end of each stage we set a to $\frac{a+c}{2} = a + \frac{b-a}{4}$ or b to $\frac{b+c}{2} = a + \frac{3(b-a)}{4}$. This means that after stage k we have $|b - a| = (\frac{3}{4})^k$. More importantly, at the beginning of stage k we have $\frac{a+c}{2} = c - \frac{b-a}{4} = c - 2^{-k(2-\log 3)-2} \leqslant c - \varepsilon_k$, and in a similar way we get $\frac{b+c}{2} \geqslant c + \varepsilon_k$. We can thus conclude that if the particle goes left, then x is nowhere in the interval $[a, \frac{a+c}{2})$, and if the particle goes right, then x will not be in the interval $(\frac{b+c}{2}, b]$. This is why at each stage we change a or b accordingly, thus ensuring that $a \leqslant x \leqslant b$ at every stage.

In order to obtain n digits of the binary expansion of x, we carry out $3(n + 1)$ stages of the modified bisection method. After $3(n+1) > \frac{n+1}{|\log_2 \frac{3}{4}|}$ stages, we have that $0 \leqslant |x - a| \leqslant |b - a| = (\frac{3}{4})^{3(n+1)} < 2^{-n-1}$, and so $x \restriction_n = a \restriction_n$. For each of these $3(n + 1)$ stages, $\mathrm{O}(n)$ digits must be written in the query tape, and so the complete method can be carried out in $\mathrm{O}(n^2)$ steps of computation.

The modified bisection method may still not be used by an imprecise error-prone analog-digital scatter machine. In fact, it is not at all obvious that such a machine can extract any useful information from the vertex. We now present a third method, called the **probabilistic bisection method**, to be used by an imprecise error prone machine. This method is inherently probabilistic, and is concocted from the ideas of Siegelmann [1999] and von Neumann [1956].

We will show that we can use an imprecise error-prone analog-digital scatter machine $\mathcal{S}_x$ to *guess* the digits of the binary expansion of x with bounded error probability.

The probabilistic bisection method consists in performing a certain number z of trials. At each trial, we command the cannon to fire at the point $\frac{1}{2}$ of its axis, and count the number of times that we measured a *left* response. We thus perform z Bernoulli trials, where the probability, at each trial, of measuring a *left* response is given by:

$$\mathbb{P}\left(S_x\left(a_{01}\right) = \mathit{left}\right) = \frac{1}{2} + \frac{x - \frac{1}{2}}{2\varepsilon} = x.$$

Now let $\#\mathit{left}$ denote the total number of times that a left response was measured. Then $\#\mathit{left}$ can be regarded as a random variable with expected value $\mu = zx$ and variance $\nu = zx(1-x)$. After running z Bernoulli trials, we make the guess that x equals $\tilde{x} = \frac{\#\mathit{left}}{z}$. We can determine what is the probability of x and $\tilde{x}$ having less than n digits in common, i.e., the probability that $|\tilde{x} - x| > 2^{-n-1}$. By Chebyshev's inequality, we see that for every Δ,

$$\mathbb{P}(|\#\mathit{left} - \mu| > \Delta) = \mathbb{P}(|z\tilde{x} - zx| > \Delta) = \mathbb{P}\left(|\tilde{x} - x| > \frac{\Delta}{z}\right) \leqslant \frac{\nu}{\Delta^2}.$$

Choosing $\Delta = z2^{-n-1}$, we get

$$\mathbb{P}(|\tilde{x} - x| > 2^{-n-1}) \leqslant \frac{x(1-x)2^{2n+2}}{z}$$

And so, the probability of making a mistake in guessing n digits of x can be bounded to δ by running $z \geqslant \delta^{-1}x(1-x)2^{2n+2} \in \mathrm{O}(2^{2n})$ trials. In other words, we can guess n digits of x with bounded error probability by making a number of experiments exponential in n. We conclude that even the imprecise error-prone analog-digital scatter machines can make use of the position of the vertex, only it takes an exponential rather than polynomial number of steps in doing so.

It becomes immediately evident that analog-digital scatter machines can decide *any* subset of $\mathbb{N}$. If we take the real-valued characteristic of a set $A \subseteq \mathbb{N}$:

$$x = \sum_{k=0}^{\infty} a_k 2^{-k-1},$$

where $a_k = 1$ if $k \in A$ and $a_k = 0$ if $k \notin A$, then we can use an analog-digital scatter machine $\mathcal{S}_x$ to decide A: in order to decide whether some value w is in A, we obtain w digits of x using the appropriate bisection method, and accept w if and only if the w-th digit of x is 1. It would thus appear that the analog-digital scatter machine is a very uninteresting model of computation, but we can overcome this by imposing *feasibility* or *efficiency* constraints, such as bounded time, on the computations of the analog-digital scatter machine. This will result in a rich complexity-theoretical characterisation, and will be the purpose of the next chapter.

7

The computational complexity of the analog-digital scatter machine

This chapter is dedicated to studying the polynomial time computations of the analog-digital scatter machine. It turns out that non-uniform complexity provides the most adequate characterisations of this computational power, and it is thus vital to have some notions of this subject in order to understand this chapter. The reader will find the required material on oracle Turing machines, probabilistic Turing machines and non-uniform complexity in part A of the appendix.

We will use the proof methods that Hava Siegelmann and Eduardo Sontag applied to the study of Analog Recurrent Neural Networks [cf. Siegelmann, 1999]. We study the error-free analog-digital scatter machine in Section 7.1, the precise error-prone machine in Section 7.2, and the imprecise error-prone analog-digital scatter machine in Section 7.3. In Section 7.4, we will make some considerations on the amount of resources spent by the scatter machine in each action.

7.1 The error-free case

We begin by studying the error-free machine. We will show that the class of sets decidable by error-free analog-digital scatter machines in polynomial time is exactly P/*poly*. We will begin by sketching the proof for advice Turing machines, and then give a more detailed proof for polynomial size circuits.

Linear precision suffices

We show that the configuration of an error-free analog-digital scatter machine after t steps only depends on the first t digits of the binary expansion of the vertex position.

Lemma 7.1. *Let $\mathcal{S}_x$ be an error-free analog-digital scatter machine, and let $\tilde{\mathcal{S}}_y$ be an error-free machine with the same finite control, but with the vertex placed at $y = x\restriction_t$. Then, for any input w, the configurations of $\mathcal{S}_x$ and $\tilde{\mathcal{S}}_y$ after t steps of computation are the same.*

Proof. This comes trivially from the behaviour of the error-free analog-digital scatter machine. Upon leaving a shooting state in the t-th step of computation, at most $t-1$ symbols may have been written on the action tape specifying the cannon position. Thus the position of the cannon is of the form $z = \frac{n}{2^{t-1}}$, for some $n \in \{0, 1, \dots, 2^{t-1}\}$. Since for any z with this form we have that $z \leqslant x$ if and only if $z \leqslant x \upharpoonright t$, the answer state must be the left state in $\mathcal{S}_x$ if and only if this is the case in $\tilde{\mathcal{S}}_y$. □

We may now sketch the proof that the error-free analog-digital scatter machine can decide P/*poly* in polynomial time, by the way of polynomial advice. Let A be a set in P/*poly*, and, by definition, let $B \in$ P, $f \in$ *poly* be such that

$$w \in A \iff \langle w, f(|w|)\rangle \in B.$$

Let $\tilde{f} : \mathbb{N} \to \Sigma^*$ be a function, also in *poly*, such that, if the symbols of $f(n)$ are $\xi_1\xi_2\dots\xi_{p(n)}$, then

$$\tilde{f}(n) = 0\xi_1 0\xi_2 \dots 0\xi_{p(n)}$$

We can create a error-free analog-digital scatter machine which also decides A, setting the vertex at the position

$$x = 0.\tilde{f}(1)11\tilde{f}(2)11\tilde{f}(3)11\dots$$

Given any input w of size n, the error-free analog-digital scatter machine $\mathcal{S}_x$ can use the bisection method to obtain $f(n)$ in polynomial time. Then the machine uses the polynomial-time algorithm which decides B, and accepts if and only if $\langle w, f(n)\rangle$ is in B. Thus we have shown that an error-free analog-digital scatter machine can decide any set in P/*poly* in polynomial time.

As for the converse, let C be any set decided in polynomial time by an error-free analog-digital scatter machine with the vertex at the position x. Lemma 7.1 ensures that to decide on any input w, the machine only makes use of $p(|w|)$ digits of x, where p is a polynomial bounding the decision time. Thus we can see that the set must be in P/*poly*, using the advice function $g \in$ *poly*, given by

$$g(n) = x \upharpoonright_{p(n)} .$$

Circuits of polynomial size

We will show a more rigorous proof of the same fact, by way of polynomial size circuits. We construct such a proof not only because circuits are representations of real-life technology, but also because the implementation through a circuit improves our understanding of the analog-digital scatter machine. It is known that P/*poly* is the class of sets decided by families of circuits with a polynomial number of gates. We will explicitly construct the polynomial size circuits which simulate an error-free analog-digital scatter machine, and then show that polynomial-size circuit families can be encoded into the vertex position, in order to be simulated by an error-free analog-digital scatter machine.

Proposition 7.2. *Every set decided in polynomial time by an error-free analog-digital scatter machine can be decided by a polynomial size family of circuits.*

The proof is similar to that of John E. Savage, as presented in [Balcázar et al., 1988], adapted for the error-free analog-digital scatter machine.

Proof. Let A be a set decided by an error-free analog-digital scatter machine $\mathcal{S}_x$ in polynomial time p. Let Q be the set of states, Σ be the working alphabet, $N = \lceil \log_2 |Q| \rceil + 1$ and $M = \lceil \log_2 |\Sigma| \rceil + 1$. We assume, without loss of generality, that $\mathcal{S}_x$ has only two tapes (one regular tape for input and rough work and one action tape), and that the heads of both tapes are always aligned with each other during the computation. We also assume that there is only one final state, and that after reaching this state the machine keeps looping forever in the same state. We will construct a circuit which will simulate $\mathcal{S}_x$ for inputs of an arbitrary size n.

We begin by encoding the set Q of states into a binary representation with N bits which excludes any sequence beginning with 0. The initial bit in the sequence may then be used to signal if the state is valid or not. We assume that the shooting, left and right states are encoded by three binary sequences, respectively q_s, q_l and q_r, such that the bit-wise OR of q_s with q_l or q_r gives, respectively, q_l and q_r.[1] We encode each symbol in Σ as a binary sequence of M bits, excluding the sequence 0^M. At each time step of the computation we associate with the k-th cells of the two tapes a representation of their local parameters: the contents of the cells, whether the tape heads are over the cells, and, should the heads be scanning the cells, the state of $\mathcal{S}_x$ at that moment. So this representation will be given by three binary sequences $s^k(t)$, $\sigma_1^k(t)$ and $\sigma_2^k(t)$, such that

$$s^k(t) = \begin{cases} 0^N & \text{if } \mathcal{S}_x \text{ is not reading the } k\text{-th cell at time } t, \\ q_i & \text{if } \mathcal{S}_x \text{ is reading the } k\text{-th cell at time } t \text{ and in the state } q_i; \end{cases}$$

$\sigma_1^k(t)$ and $\sigma_2^k(t)$ represent the contents of the k-th cells at time t. It is by these three sequences that we will represent a cell in the circuit simulation of an error-free analog-digital scatter machine. The three sequences are modified every time-step of the simulated computation, by use of two functions we will call *state* and *symbol*. The function *state* : $\{0,1\}^{N+2M} \to \{0,1\}^{3N}$ is given by the following expression:

$$state(q, \sigma_1, \sigma_2) = \begin{cases} (0^N, 0^N, 0^N) & \text{if } q \text{ begins with a } 0, \\ (q', 0^N, 0^N) & \text{if } \mathcal{S}_x \text{ goes to } q', \text{ moving the heads left}, \\ (0^N, q', 0^N) & \text{if } \mathcal{S}_x \text{ goes to } q', \text{ not moving the heads}, \\ (0^N, 0^N, q') & \text{if } \mathcal{S}_x \text{ goes to } q', \text{ moving the heads right}. \end{cases}$$

We will call $l-state$, $m-state$ and $r-state$, to the left, middle and right parts of the *state* function.

The function *symbol* : $\{0,1\}^{N+2M} \to \{0,1\}^{2M}$ gives the symbol to be printed on the cells, prior to moving the tape heads. We set $symbol(q, \sigma_1, \sigma_2) = (\sigma_1, \sigma_2)$ to any sequence q which

[1] The encoding could set, e.g., q_s to 001, q_l to 011 and q_r to 101: this way we have $q_s \oplus q_l = q_l$ and $q_s \oplus q_r = q_r$, where $\oplus$ represents the bit-wise OR.

begins with a 0, since the tape head must be on the cells for the written symbol to change. If q represents a valid state, then $symbol(q, \sigma_1, \sigma_2)$ is given by the transition function.

We are now ready to specify the circuit, which is represented in Fig. 7.1. Since $\mathcal{S}_x$ works in polynomial time, we only need to consider a polynomial number of cells. For each of these cells, and for each time step, we create a *tape cell circuit* which takes as input $N + 2M$ bits and computes the *state* and *symbol* functions. The first N bits of input are given by the ORing of the $l{-}state$, $m{-}state$ and $r{-}state$ of the neighbouring cells in the previous step. The OR gates that go into inputs 2 to N will also receive the output of the *scatter machine circuit*, which we will describe bellow. At each time step, there is only one tape cell circuit which receives a 1 on its first input: this 1 signals the position of the heads.

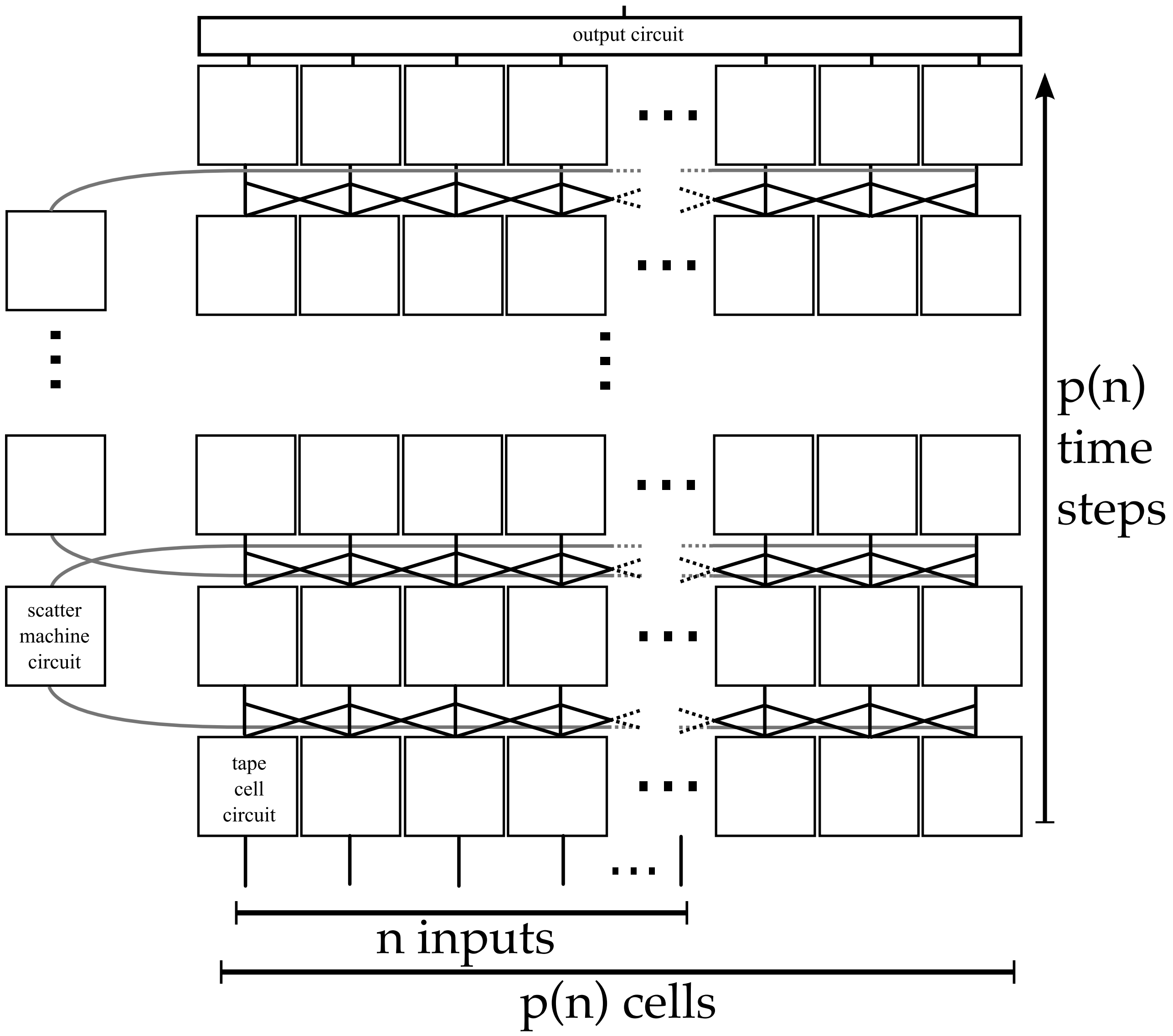


Fig. 7.1. A circuit to simulate the scatter machine.

We set $\tilde{x} = x \restriction_{p(n)}$ to be the truncation of the position of the vertex, and $a = \tilde{x} \times 2^{p(n)}$. The scatter machine circuit computes the function $sm : \{0,1\}^{N+Mp(n)} \to \{0,1\}^N$, given by

$$sm(q,c) = \begin{cases} 0^N & \text{if } q \neq q_s, \\ q_l & \text{if } c \leqslant a, \\ q_r & \text{if } c > a. \end{cases}$$

The first N bits of input of the circuit represent the state of the machine and are given by an ORing of every $m-state$ function in the previous level. The last $Np(n)$ bits represent the contents of the action tape and are obtained by ORing the second part of the *symbol* function in the previous level. We connect the last $N-1$ outputs of the scatter machine circuit to the OR gates which feed into the inputs number 2 to N of every tape cell circuit in the following level.

We add an *output circuit* which takes the ORing of every $m-state$ function in the last level and checks if the obtained bits represent the final state. The size of the tape cell and the output circuits is O(1), while each scatter machine and connections between levels cost O($p(n)$). Thus the whole circuit has a cost of $O(p(n)^2)$, which is polynomial in the size of the input. □

In order to have an error-free analog-digital scatter machine simulate families of circuits, we will need to code the circuits in some fashion. We thus make use of the following:

Lemma 7.3 (circuit encoding). *There is an injective, total encoding function of feed-forward circuits into binary sequences, say $\tilde{c}$, such that*

1. *every circuit of size n is encoded into a binary sequence of size $n^2 + 2n$, and*
2. *the sequence* 11 *never appears in the encoding of any circuit.*

Proof. We encode a circuit of n gates as a graph. In our coding, the first $n^2 - n$ bits represent the edge matrix. Since the graph is feed-forward, we encode $\frac{n^2-n}{2}$ edges, times two bits 01 to signal that there is an edge, and two bits 00 to mean that there is no edge. The last $3n$ bits represent the type of gate: 000 codes for an AND gate, 001 codes for an OR gate, and 010 codes for a NOT gate. Notice that 11 will never appear in the encoding. □

Proposition 7.4. *Every set decided by a polynomial size family of circuits can be decided in polynomial time by an error-free analog-digital scatter machine.*

Proof. Let A be a set which is decided by a family $\mathcal{C} = \mathcal{C}_1, \mathcal{C}_2, \dots$ of circuits. Let p be a polynomial bound on the size of the family of circuits, i.e., p is a polynomial such that $\mathcal{C}_n$, which decides the membership in A for inputs of size n, has no more than $p(n)$ gates. We construct an error-free analog-digital scatter machine, $\mathcal{S}_x$, with the vertex positioned at $x = 0.\tilde{c}(\mathcal{C}_1)11\tilde{c}(\mathcal{C}_2)11\dots$ We can find the digits which encode the circuit $\mathcal{C}_n$ among the first

$$k = \sum_{i=1}^{n} \left(p(i)^2 + 2p(i) + 2\right)$$

digits of x. Thus, for $\mathcal{S}_x$ to decide if w of size n is in A, $\mathcal{S}_x$ extracts the first k digits of x using the bisection method, obtains the digits which encode $\mathcal{C}_n$, and uses any polynomial-time algorithm for simulating the circuit. This is done in a polynomial number of steps, since k is polynomial in n. □

As already mentioned, the class of sets decided by polynomial size circuits is exactly $\mathrm{P}/poly$ [cf., e.g., Balcázar et al., 1988], and so we conclude the following.

Theorem 7.5. *The class of sets decided by error-free analog-digital scatter machines in polynomial time is exactly* $\mathrm{P}/poly$.

7.2 The precise error-prone case

In this chapter we investigate the class of sets decidable in polynomial time by a precise error-prone analog-digital scatter machine. We will conclude that the computational power of these machines is not altered by considering a controllable error, since they decide exactly $\mathrm{BPP}//poly = \mathrm{P}/poly$ in polynomial time.

Linear precision suffices

As in section 7.1, we show that only a linear number of digits of the vertex position influences the outcome of the computation. Since the behaviour of the error-prone analog-digital scatter machines is probabilistic, we can not ensure that the state of the machine is the same after $t(n)$ steps. Instead we show that if a machine halts in $t(n)$ steps with small enough error-probability, then this machine will decide the same set if the vertex position is truncated to $\mathrm{O}(t(n))$ digits.

Proposition 7.6. *Let $\mathcal{S}_x$ be a precise error-prone analog-digital scatter machine, deciding some set in time $t(n)$ with error probability bounded by $\gamma < \frac{1}{4}$. Let $\tilde{\mathcal{S}}_y$ be a precise error-prone analog-digital scatter machine, with the same finite control as $\mathcal{S}_x$ and with the vertex placed at $y = x\restriction_{5t(n)}$. Then $\mathcal{S}_x$ and $\tilde{\mathcal{S}}_y$ make the same decision on every input of size smaller or equal to n.*

Proof. For each input w with $|w| = n$, the machines $\mathcal{S}_x$ and $\tilde{\mathcal{S}}_y$ can perform a limited number of distinct computations. Each computation corresponds to different outcomes in the possible scatter machine experiments carried out by $\mathcal{S}_x$. There are only two possible outcomes, left or right, and so the number of distinct computations is at most $2^{t(n)}$. Among these computations, some will give a correct decision for w, and some will give an incorrect decision.

If the machines $\mathcal{S}_x$ and $\tilde{\mathcal{S}}_y$ command the cannon to shoot from the same dyadic position z, then we can express the probability that the outcome for $\mathcal{S}_x$ is different than the outcome for $\tilde{\mathcal{S}}_y$, which we denote by δ_z. Then δ_z is bounded by a value δ, given by

$$\delta_z \leqslant \delta = \frac{|x-y|}{2\varepsilon}.$$

After carrying out $t(n)$ steps or less, the size of the word representing z, which was written in the action tape, is bounded by $t(n)$, and thus the error in aiming the cannon is no smaller than $\tilde{\varepsilon} = 2^{-t(n)-1}$. We can then conclude that $\delta \leqslant 2^{t(n)}|x - y|$. Since the chosen truncation gives us $|x - y| \leqslant 2^{-5t(n)}$, we then bound $\delta \leqslant 2^{-4t(n)}$.

We thus see that if the $\mathcal{S}_x$ and $\tilde{\mathcal{S}}_y$ have the same configuration, at some point in their computations, then the probability that their configuration becomes different in the following step of computation is at most $\delta \leqslant 2^{-4t(n)}$. Now, let C denote the set of the computations that make an incorrect decision. Each computation $c \in C$ is represented by a sequence $c_0, c_1, \ldots, c_{t(n)}$ of configurations, and to each configuration we can assign a probability value a_{i+1} which represents the probability that the machine $\mathcal{S}_x$ goes from configuration c_i to c_{i+1}. The value a_{i+1} can be understood as the probability of $\mathcal{S}_x$ making a bad transition after configuration c_i. The probability of $\mathcal{S}_x$ making an incorrect decision, i.e., the error probability of $\mathcal{S}_x$, is given by

$$\sum_{c \in C} \prod_{i=1}^{t(n)} a_i \leqslant \gamma < \frac{1}{4}. \tag{7.1}$$

Now, in the worst-case scenario, the probability of $\tilde{\mathcal{S}}_y$ making a bad transition will be increased by δ for every possible bad transition. We can see that, in this case, the error probability of $\tilde{\mathcal{S}}_y$ is less or equal to

$$\sum_{c \in C} \prod_{i=1}^{t(n)} (a_i + \delta) \leqslant \sum_{c \in C} \left(\prod_{i=1}^{t(n)} \max(a_i, 2^{-3t(n)}) \prod_{i=1}^{t(n)} \left(1 + 2^{-t(n)}\right) \right).$$

A simple calculation will show us that $\prod_{i=1}^{t(n)} \left(1 + 2^{-t(n)}\right) < 2$. From (7.1) and $|C| \leqslant 2^{t(n)}$ we deduce

$$\sum_{c \in C} \prod_{i=1}^{t(n)} \max(a_i, 2^{-3t(n)}) \leqslant \frac{1}{4}.$$

And so $\sum_{c \in C} \prod_{i=1}^{t(n)} (a_i + \delta) < \frac{1}{4} \times 2 = \frac{1}{2}$. Thus, $\tilde{\mathcal{S}}_y$ decides in the same way as $\mathcal{S}_x$, with bounded error probability. □

This allows us to show the following.

Proposition 7.7. *Every set decided by a precise error-prone analog-digital scatter machine in polynomial time is in* BPP//*poly.*

Proof. Let A be a set decided by a precise error-free analog-digital scatter machine $\mathcal{S}_x$ in polynomial time p, and with an error probability bounded by $\frac{1}{4}$. We use the advice function $f \in poly$, given by $f(n) = x \upharpoonright_{5p(n)}$, to construct a probabilistic Turing machine $\mathcal{M}$ which decides A in polynomial time and with bounded error probability.

Given any dyadic rational $\tilde{x} \in [0, 1]$, the machine $\mathcal{M}$ can carry out a Bernoulli trial X with an associated probability $\mathbb{P}(X = 1) = \tilde{x}$. If $\tilde{x}$ has the binary expansion $\xi_1 \ldots \xi_k$, the machine $\mathcal{M}$ tosses its balanced coin k times, and constructs a word $\tau_1 \ldots \tau_k$, where τ_i is 1 if the coin turns up heads and 0 otherwise. The Bernoulli trial will have the outcome 1 if $\xi_1 \dots \xi_k < \tau_1 \dots \tau_k$, and 0 otherwise, and this will give the desired probability.

The probabilistic machine $\mathcal{M}$ will decide if $w \in A$ by simulating $\mathcal{S}_x$ on the input w as if the vertex was placed at the position $\tilde{x} = x \upharpoonright_{5p(n)}$. In order to mimic the shooting of the cannon from the position z, which should have an error $\varepsilon = 2^{-|z|-1}$, the machine will carry out a Bernoulli trial X with an associated dyadic probability

$$\mathbb{P}(X = 1) = \begin{cases} 1 & \text{if } z < \tilde{x} - \varepsilon, \\ \frac{1}{2} + \frac{\tilde{x} - z}{2\varepsilon} & \text{if } \tilde{x} - \varepsilon \leqslant z \leqslant \tilde{x} + \varepsilon, \\ 0 & \text{if } z > \tilde{x} + \varepsilon. \end{cases}$$

Then $\mathcal{M}$ will simulate a left hit when $X = 1$ and a right hit when $X = 0$. As we have seen in the previous proposition, $\mathcal{S}_x$ will, when simulated in this way, decide the same set in polynomial time and with bounded error probability. □

Using the modified bisection method

The following inclusion can be demonstrated by using the modified bisection method.

Proposition 7.8. *Every set in* BPP//*poly can be decided in polynomial time by a precise error-prone analog-digital scatter machine.*

Proof. Let A be an arbitrary set in BPP//*poly*. Since BPP//*poly* = P/*poly*, let $B \in$ P, $f \in$ *poly* be witnesses that $A \in$ P/*poly*. Let $\tilde{f} : \mathbb{N} \to \Sigma^*$ be a function, also in *poly*, such that

$$\tilde{f}(n) = 0\xi_1 0\xi_2 \dots 0\xi_{p(n)}, \text{ where } f(n) = \xi_1 \xi_2 \dots \xi_{p(n)}$$

We can now create a precise error-prone analog-digital scatter machine, $\mathcal{S}_x$, which decides A, setting the vertex at the position

$$x = 0.\tilde{f}(1)11\tilde{f}(2)11\tilde{f}(3)11\dots$$

If we assume that $|f(n)| \leqslant p(n)$ for some polynomial p, then $|\tilde{f}(n)| \leqslant 2p(n)$, and $\tilde{f}(n)$ may be obtained from the first $2np(n) + 2n - 2$ digits of x. Given an input w of size n, the machine uses the modified bisection method to obtain these digits, which should require $\mathrm{O}(n^2 p(n)^2)$ steps, and then uses the obtained advice to decide in polynomial time whether $\langle x, f(n) \rangle$ is in B. $\mathcal{S}_x$ accepts if this is the case, and rejects otherwise. □

7.3 Increasing the error

We will now study the class of sets decidable in polynomial time by an imprecise error-prone analog-digital scatter machine. We begin by showing that the computations of an imprecise error-prone analog-digital scatter machine only depend on a number of digits of the position of the vertex logarithmic in the number of computation steps. We will then show that such machines may, in polynomial time, make probabilistic guesses of up to a logarithmic number of digits of the position of the vertex. We will then conclude that these machines decide exactly BPP/*log*∗.

Logarithmic precision suffices

The first result is similar to that of the previous two sections. We will show that the decision of an imprecise error-prone analog-digital scatter machine remains the same if we only keep a logarithmic number of digits of the position of the vertex. However, this result is harder to prove, because we may not make δ (as in the proof of Proposition 7.6) as small as before. We have considered, in the proof of Proposition 7.6, the set of all computations making a wrong decision on a certain input. In order to prove something similar to Proposition 7.6 for imprecise error-prone machines, we will need to study the structure of this set of computations in more detail.

Our first observation is that the set of possible computations can be arranged in a tree-like structure, where each path of the tree from the root to a leaf corresponds to a possible computation. In this tree of probabilistic computations, we may assign a probability value to each edge. We will obtain the following structure.

Definition 7.9. *A* ***probabilistic tree*** *A is a pair $A = \langle \mathcal{T}_A, p_A\rangle$, where*

1. *$\mathcal{T}_A = \langle N_0, \mathcal{N}_A, \mathcal{E}_A\rangle$ is a finite rooted tree with a set $\mathcal{N}_A$ of nodes, a set $\mathcal{E}_A$ of edges, and a root $N_0 \in \mathcal{N}_A$,*
2. *p_A is a function from $\mathcal{E}_A$ to $[0, 1]$, which assigns a* probability value *to each edge of $\mathcal{T}_A$, and*
3. *For every $N \in \mathcal{N}_A$, the sum of the probability values for the outgoing edges of N is no more than 1, i.e.,*
$$\sum_{N\to M\in\mathcal{E}_A} p_A(N \to M) \leqslant 1.$$

The ***outdegree*** *of A is simply the outdegree of $\mathcal{T}_A$, i.e., the maximum number of outgoing edges from any node in $\mathcal{T}_A$. The* ***height*** *of A is the length of the largest path from the root of $\mathcal{T}_A$ to any of its leaves. For each path $N_0 \to N_1 \to \ldots \to N_t$, where N_0 is the root and N_t is a leaf, we define the corresponding* ***probability path*** *in A by the sequence $a_1 \ldots a_t$ given by $a_i = p_A(N_{i-1} \to N_i)$, $1 \leqslant i \leqslant t$. We denote the set of all probability paths in A by $Paths(A)$. The* ***probability value*** *of A is given by:*

$$Prb(A) = \sum_{a_1\ldots a_k\in Paths(A)} \prod_{i=1}^{k} a_i.$$

We will be interested in considering the case when every probability value in every probability path is increased by a positive value δ. We will then denote the ***added probability value*** *of A by $\delta > 0$, by*

$$Prb(A+\delta) = \sum_{a_1 \ldots a_k \in Paths(A)} \prod_{i=1}^{k} (a_i + \delta).$$

*Finally, A will be called **partial** whenever $Prb(A) < 1$, and it will be called **grown** when all paths from the root to the leaves have the same length.*

Proposition 7.10. *Let $t, u \in \mathbb{N}$. Let A be a probabilistic tree with height t and outdegree of at most 2. Then there is a probabilistic tree C with height t and outdegree at most $u+2$ such that*

$$(\forall N \to M \in \mathcal{E}_C)\ p_C(N \to M) \leqslant \frac{1}{u}, \tag{7.2}$$

$$Prb(A) = Prb(C) \tag{7.3}$$

and for any $\delta > 0$,

$$Prb(A+\delta) \leqslant Prb(C+\delta) \tag{7.4}$$

Furthermore, if A is grown, then so is C.

Proof. The proof is by induction in the height of A. If $t = 0$, then A must have a single node (the root), and setting $C = A$ will trivially verify (7.2), (7.3) and (7.4). Now suppose that $t = n+1$. Then, because A has an out-degree of at most 2, then A must have a root node and one or two rooted subtrees. We will ignore the case when there is only one subtree, because it is proved in the exact same way as the case for two subtrees.

So let A_1 and A_2, of height n, be the two subtrees of A below N. Let N_1 and N_2 be the root nodes of A_1 and A_2, $p_1 = p_A(N \to N_1)$ and $p_2 = p_A(N \to N_2)$. Then choose $\alpha_1, \alpha_2 \in (0, \frac{1}{u}]$, $\beta_1, \beta_2 \in \{0, 1, \ldots, u\}$ to be the unique numbers such that

$$p_1 = \beta_1 \frac{1}{u} + \alpha_1 \qquad p_2 = \beta_2 \frac{1}{u} + \alpha_2.$$

Then $\beta_1 + \beta_2 \leqslant u$.

By induction hypothesis there must be two trees C_1 and C_2 which verify (7.2), (7.3) and (7.4), with respect to A_1 and A_2. Furthermore, C_1 and C_2 will be grown whenever A_1, resp. A_2, are grown.

Now construct the tree C in the following way. The root of C will be a node N with $\beta_1 + \beta_2 + 2 \leqslant u + 2$ outgoing edges; $\beta_1 + 1$ of these edges will connect to the root of C_1, the first β_1 of which will have a probability value $\frac{1}{u}$, and the last one will have a probability value α_1; the remaining $\beta_2 + 1$ edges connect to the root of C_2, and the first β_2 of these will have a probability value $\frac{1}{u}$, and the last edge will have a probability value of α_2. This is illustrated in Fig. 7.2.

Clearly, because C_1 and C_2 satisfy (7.2), then so will C. The probability value of C is given by

$$Prb(C) = \sum_{c_1 \ldots c_k \in Paths(C)} \prod_{i=1}^{k} c_i = (\beta_1 \frac{1}{u} + \alpha_1) Prb(C_1) + (\beta_2 \frac{1}{u} + \alpha_2) Prb(C_2).$$

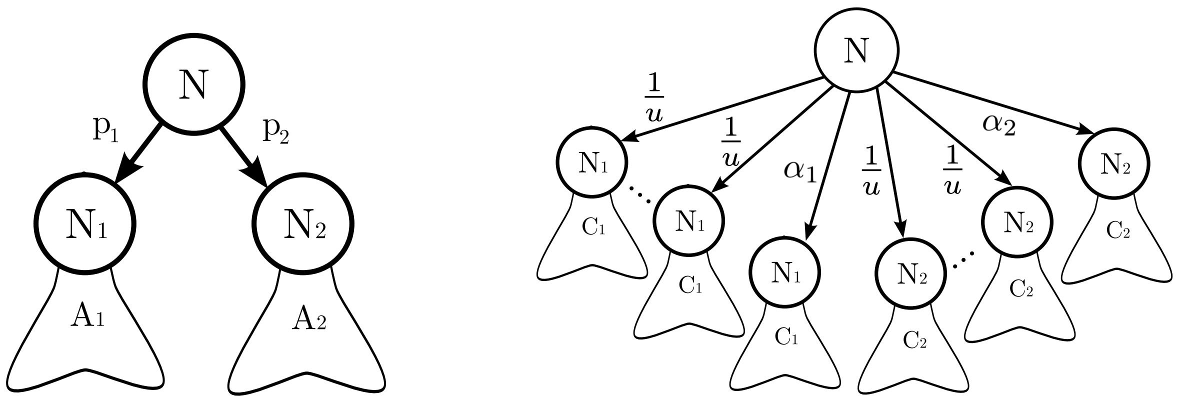


Fig. 7.2. Inductive step: changing from A (to the left) to C (to the right).

This value simplifies to $p_1 Prb(A_1) + p_2 Prb(A_2) = Prb(A)$, which then gives (7.3). Finally, we may calculate

$$Prb(C+\delta) = \left(\beta_1\left(\frac{1}{u}+\delta\right) + (\alpha_1+\delta)\right) Prb(C_1+\delta) + \left(\beta_2\left(\frac{1}{u}+\delta\right) + (\alpha_2+\delta)\right) Prb(C_2+\delta).$$

And this last expression is greater or equal to $(p_1+\delta)Prb(A_1+\delta) + (p_2+\delta)Prb(A_2+\delta) = Prb(A+\delta)$, resulting in (7.4). The outdegree of C is no more than $u+2$, and C will be grown whenever A is grown. $\square$

Proposition 7.11. *Let $\mathcal{S}_x$ be an imprecise error-prone analog-digital scatter machine. Suppose $\mathcal{S}_x$ decides some set in constructible time t, with error probability bounded by $\gamma < \frac{1}{4\mathrm{e}}$.*[2] *Then if we set $y = x \restriction_{2\log(t(n))+7}$, and $\tilde{\mathcal{S}}_y$ is an imprecise error-prone machine with the same finite control as $\mathcal{S}_x$, then $\mathcal{S}_x$ and $\tilde{\mathcal{S}}_y$ make the same probabilistic decision on every input of size less or equal to n.*

Proof. The proof goes along the same lines as the proof of Proposition 7.6. Let w be an input of size $m \leqslant n$. We construct the probabilistic tree E of all possible computation of $\mathcal{S}_x$ on w. Each node in this tree is a configuration. At the root of the tree is the initial configuration. If a configuration M follows deterministically from a configuration $N \in \mathcal{N}_E$, then we add M to $\mathcal{N}_E$ and $N \to M$ to $\mathcal{E}_E$, setting $p_E(N \to M) = 1$. However, if a configuration $N \in \mathcal{N}_E$ is in the shooting state, then there may be a positive probability of obtaining a configuration N_1 in the left state or a configuration N_2 in the right state, after the scatter machine equipment carries out the experiment. In this case, we add N_1 and N_2 to $\mathcal{N}_E$, as well as $N \to N_1, N \to N_2$ to $\mathcal{E}_E$, and set $p_E(N \to N_1)$ to the probability of obtaining a left hit after shooting the cannon with the value z written on the query tape in configuration c. Obviously, we set $p_E(N \to N_2) = 1 - p_E(N \to N_1)$. We may assume without loss of generality that every possible computation in $\mathcal{S}_x$ halts after the same number of steps, because if this is not the case then we may, by the time-constructibility of t, choose another error-prone analog-digital scatter machine in which a clock forces this condition. For this reason, the tree E will be a

[2] Here, "e" stands for the Napier's number.

grown tree with height $t(m)+1$, and every path in $\mathit{Paths}(E)$ will have a length of exactly $t(m)$.

Preparing the tree E in this way, we may select from the paths of E those which lead to a bad decision, and obtain a grown partial tree A, of height $t(m)+1$ and outdegree at most 2. Then the error probability of $\mathcal{S}_x$ will be $\mathit{Prb}(A)$. As in the proof of Proposition 7.6, we can see that the probability of $\mathcal{S}_x$ and $\tilde{\mathcal{S}}_y$ obtaining a different response after making the same experiment with the cannon set at position z, δ_z, is bounded by a value δ:

$$\delta_z \leqslant \delta = \frac{|x-y|}{2\varepsilon}.$$

Where $\varepsilon = \frac{1}{2}$ is the fixed error when aiming the cannon. So, because $2^5 > \mathrm{e}^3$, we get

$$|x-y| \leqslant 2^{-2\log t(m)-7} < \frac{1}{4\mathrm{e}^3 t(m)^2}.$$

This gives us $\delta < \frac{1}{4\mathrm{e}^3 t(m)^2}$. Now, in the worst case scenario, the probability of choosing a bad transition, i.e., the probability of staying in a configuration in the tree A of bad computation paths, will be increased by δ at each transition. And so the error probability of $\tilde{\mathcal{S}}_y$ will be bounded above by

$$\sum_{a_1 \dots a_{t(m)} \in \mathit{Paths}(A)} \prod_{i=1}^{t(m)} (a_i + \delta) = \mathit{Prb}(A+\delta).$$

By Proposition 7.10, there must be a grown probabilistic tree C with height $t(m)+1$ and outdegree at most $t(m)+2$ such that

$$(\forall N \to M \in \mathcal{E}_C)\; p_C(N \to M) \leqslant \frac{1}{t(m)}, \tag{7.5}$$

$$\mathit{Prb}(A) = \mathit{Prb}(C), \text{ and} \tag{7.6}$$

$$\mathit{Prb}(A+\delta) \leqslant \mathit{Prb}(C+\delta). \tag{7.7}$$

We begin by showing that

$$\sum_{c_1 \dots c_{t(m)} \in \mathit{Paths}(C)} \prod_{i=1}^{t(m)} \max\left(c_i, \frac{1}{4t(m)\mathrm{e}^3}\right) < \frac{1}{2\mathrm{e}}. \tag{7.8}$$

In fact, we may separate $\mathit{Paths}(C)$ into two disjoint sets P_1 and P_2. In P_1 we will find the paths $c_1 \dots c_{t(m)}$ such that every $c_i \geqslant \frac{1}{4t(m)\mathrm{e}^3}$. In P_2 we will find the remaining paths, i.e., those for which some $c_k < \frac{1}{4t(m)\mathrm{e}^3}$. Then,

$$\sum_{c_1 \dots c_{t(m)} \in \mathit{Paths}(C)} \prod_{i=1}^{t(m)} \max\left(c_i, \frac{1}{4t(m)\mathrm{e}^3}\right) = \sum_{c_1 \dots c_{t(m)} \in P_1} \prod_{i=1}^{t(m)} c_i + \sum_{c_1 \dots c_{t(m)} \in P_2} \prod_{i=1}^{t(m)} \max\left(c_i, \frac{1}{4t(m)\mathrm{e}^3}\right).$$

Now, because $c_i < \frac{1}{t(m)}$, by (7.5), because there are at most $(t(m)+2)^{t(m)}$ paths in C, and because in every $c_1 \dots c_{t(m)} \in P_2$ there is some $c_k < \frac{1}{4t(m)\mathrm{e}^3}$, we find

$$\sum_{c_1 \dots c_{t(m)} \in P_1} \prod_{i=1}^{t(m)} c_i + \sum_{c_1 \dots c_{t(m)} \in P_2} \prod_{i=1}^{t(m)} \max\left(c_i, \frac{1}{4t(m)\mathrm{e}^3}\right) < Prb(C) + \frac{1}{4\mathrm{e}^3}\left(\frac{t(m)+2}{t(m)}\right)^{t(m)} .$$

From (7.6) we see that $Prb(C) = Prb(A) < \frac{1}{4\mathrm{e}}$, and so

$$\sum_{c_1 \dots c_{t(m)} \in Paths(C)} \prod_{i=1}^{t(m)} \max\left(c_i, \frac{1}{4t(m)\mathrm{e}^3}\right) < \frac{1}{2\mathrm{e}} + \frac{\mathrm{e}^2}{4\mathrm{e}^3}$$

which is exactly (7.8). We are now ready to end the proof. Because of (7.7), we have

$$Prb(A+\delta) \leqslant Prb(C+\delta) \leqslant \sum_{c_1 \dots c_{t(m)} \in Paths(C)} \prod_{i=1}^{t(m)} \left(\max\left(c_i, \frac{1}{4\mathrm{e}^3 t(m)}\right) + \delta\right) .$$

We therefore find

$$Prb(A+\delta) \leqslant \sum_{c_1 \dots c_{t(m)} \in Paths(C)} \prod_{i=1}^{t(m)} \max\left(c_i, \frac{1}{4\mathrm{e}^3 t(m)}\right) \prod_{i=1}^{t(m)} \left(1 + 4\mathrm{e}^3 t(m)\delta\right) . \qquad (7.9)$$

Now, because of our bound $\delta < \frac{1}{4\mathrm{e}^3 t(m)^2}$, we see that

$$\prod_{i=1}^{t(m)} \left(1 + 4\mathrm{e}^3 t(m)\delta\right) < \prod_{i=1}^{t(m)} \left(1 + \frac{1}{t(m)}\right) < \mathrm{e}.$$

Considering this bound along with (7.8), and substituting in (7.9), we get $Prb(A+\delta) < \frac{1}{2}$, as intended. □

Proposition 7.12. *Every set decided by an imprecise error-prone analog-digital scatter machine in polynomial time is in* BPP//*log*∗.

Proof. The proof is in every way similar to the proof of Proposition 7.7, using the advice function $f(n) = x\upharpoonright_{2\log p(n)+7}$. □

Guessing the advice

We will now use the probabilistic bisection method to show that it is possible for an imprecise error-prone analog-digital scatter machine to guess any prefix advice function in *log* in polynomial time.

Lemma 7.13. *For any $\delta < \frac{1}{2}$, and prefix function $f \in log$, there is an imprecise error-prone analog-digital scatter machine which obtains $f(n)$ in polynomial time with an error of at most δ.*

Proof. Let f be a prefix function in *log*, and $a, b \in \mathbb{N}$ be such that $|f(n)| = \lfloor a \log n + b \rfloor$. We show that there is a real value $0 \leqslant x \leqslant 1$ such that it is possible to obtain the value $f(n)$ from a logarithmic number of digits of x. Since f is a prefix function, we can consider the sequence φ which is the limit of $f(n)$ as $n \to \infty$. Let φ_n be the n-th symbol (1 or 0) in this sequence, and set $x = \sum_{n=1}^{\infty} \varphi_n 2^{-n}$. Then, since f is not ultimately constant, the digits in the binary expansion of x are exactly the symbols of φ. Thus, the value $f(n)$ can be obtained from the first $\lfloor a \log n + b \rfloor$ digits of the binary expansion of x.

We may then use an imprecise error-prone analog-digital scatter machine $\mathcal{S}_x$, together with the probabilistic bisection method, to obtain the first $\lfloor a \log n + b \rfloor$ digits of x with bounded error probability δ in $\mathrm{O}(2^{2\lfloor a \log n + 2b \rfloor}) \in \mathrm{O}(n^{2a})$ steps of computation. □

Proposition 7.14. *An imprecise error-prone analog-digital scatter machine can decide any set in* BPP//*log*∗ *in polynomial time.*

Proof. Let A be an arbitrary set in BPP//*log*∗. Let $\mathcal{M}$ be a probabilistic Turing machine which decides A with prefix advice $f \in log$ in a polynomial number of steps, and with an error $\gamma < \frac{1}{7}$. Let $a, b \in \mathbb{N}$ be such that $|f(n)| = \lfloor a \log n + b \rfloor$. We construct an imprecise error-prone analog-digital scatter machine, $\mathcal{S}_x$, which estimates $f(n)$ with an error $\delta < \frac{1}{4}$, as in the Lemma 7.13. The vertex of $\mathcal{S}_x$ is in the position $x = \frac{1}{2} - \varepsilon + 2r\varepsilon$, and there is a probability of at least $\frac{3}{4}$ that after obtaining $\tilde{r}$ the error is $|\tilde{r} - r| \leqslant 2^{-a \log n - b}$. After estimating $\tilde{r}$, $\mathcal{S}_x$ prepares the cannon to be placed in the position $\tilde{x} = \frac{1}{2} - \varepsilon + 2\tilde{r}\varepsilon$, which must then obey $|\tilde{x} - x| = 2\varepsilon|\tilde{r} - r| \leqslant 2\varepsilon 2^{-a \log n - b}$. This way, whenever the coupled Turing machine begins an experiment, the cannon will be placed somewhere in the interval $[\tilde{x} - \varepsilon, \tilde{x} + \varepsilon]$, with a uniform probability. So the probability of obtaining a left hit is $\tilde{p} = \frac{1}{2} + x - \tilde{x}$, while the probability of a coin-toss of $\mathcal{M}$ getting a heads is $p = \frac{1}{2}$. The machine $\mathcal{S}_x$ will then use the cannon as a coin, in order to simulate the probabilistic machine $\mathcal{M}$: a left hit will represent a heads and a right hit will represent a tails. During the computation of $\mathcal{M}$, as well as during the simulation of $\mathcal{M}$ performed by $\mathcal{S}_x$, a sequence of random bits will be generated. The probability of $\mathcal{M}$ obtaining a *bad* sequence, i.e., which leads $\mathcal{M}$ to a wrong decision, is bounded above by γ. For a specific sequence $r_1 \dots r_t$ with R heads, the probability of it being generated by $\mathcal{M}$ is

$$\mathbb{P}(\mathcal{M} \leftarrow r_1 \dots r_t) = p^R (1-p)^{t-R},$$

while the probability of the same sequence being generated by $\mathcal{S}_x$ is

$$\mathbb{P}(\mathcal{S}_x \leftarrow r_1 \dots r_t) = \tilde{p}^R (1-\tilde{p})^{t-R}.$$

Take $q = (1-p) = \frac{1}{2}$ and $\tilde{q} = (1-\tilde{p})$. We will consider the two cases (I) $\tilde{p} \leqslant p$ and (II) $\tilde{p} > p$. For case (I) we get $\tilde{q} \geqslant q$ and arrive at

$$\frac{\mathbb{P}(\mathcal{S}_x \leftarrow r_1 \dots r_t)}{\mathbb{P}(\mathcal{M} \leftarrow r_1 \dots r_t)} \leqslant \left(\frac{\tilde{q}}{q}\right)^t = \left(1 + \frac{\tilde{q} - q}{q}\right)^t \leqslant \left(1 + \frac{|\tilde{q} - q|}{q}\right)^t.$$

In the case (II), where $\tilde{q} < q$, we obtain

$$\frac{\mathbb{P}(\mathcal{M} \leftarrow r_1 \ldots r_t)}{\mathbb{P}(\mathcal{S}_x \leftarrow r_1 \ldots r_t)} \leqslant \left(1 + \frac{|q - \tilde{q}|}{\tilde{q}}\right)^t .$$

Since q and $\tilde{q}$ are bounded below by $\frac{1}{4}$, then both of the above fractions are, respectively in the case (I) and (II), bounded above by

$$(1 + 4|\tilde{q} - q|)^t \leqslant \exp(4|\tilde{x} - x|t) \leqslant \exp\left(8\varepsilon 2^{-a \log n - b} t\right) .$$

Since $\mathcal{M}$ decides in polynomial time, we may assume that the size of the needed random sequence is bounded by $t \leqslant n^k$ for some k, and since we can always pad the advice f, we will also assume that $a \geqslant k$, $b \geqslant \log(\frac{8\varepsilon}{\ln \tilde{\gamma}})$,[3] for some $1 < \tilde{\gamma} < \min(\frac{1}{7\gamma}, \frac{6}{5})$. For these values of a, b and $\tilde{\gamma}$, the fractions $\dfrac{\mathbb{P}(\mathcal{S}_x \leftarrow r_1 \ldots r_t)}{\mathbb{P}(\mathcal{M} \leftarrow r_1 \ldots r_t)}$ and $\dfrac{\mathbb{P}(\mathcal{M} \leftarrow r_1 \ldots r_t)}{\mathbb{P}(\mathcal{S}_x \leftarrow r_1 \ldots r_t)}$ are, respectively for case (I) and (II), less or equal to

$$\exp\left(8\varepsilon 2^{-a \log n - b} t\right) \leqslant \tilde{\gamma}$$

We now make use of the obvious identity $\mathbb{P}(A) = \dfrac{\mathbb{P}(A)}{\mathbb{P}(B)}\mathbb{P}(B)$. For case (I), when $\tilde{p} \leqslant p$, we conclude that the probability of $\mathcal{S}_x$ obtaining a bad sequence is bounded by

$$\mathbb{P}(\mathcal{S}_x \leftarrow \text{bad seq.}) \leqslant \tilde{\gamma}\mathbb{P}(\mathcal{M} \leftarrow \text{bad seq.}) \leqslant \tilde{\gamma}\gamma < \frac{1}{7} < \frac{1}{3}$$

For case (II), when $\tilde{p} > p$, let us suppose that the probability of $\mathcal{S}_x$ obtaining a good sequence is less or equal to $\frac{5}{7}$. Then the probability of $\mathcal{M}$ obtaining such a good sequence is also bounded, by

$$\mathbb{P}(\mathcal{M} \leftarrow \text{good seq.}) \leqslant \tilde{\gamma}\mathbb{P}(\mathcal{S}_x \leftarrow \text{good seq.}) \leqslant \tilde{\gamma}\frac{5}{7} < \frac{6}{7}.$$

We know this is not true, since $\mathbb{P}(\mathcal{M} \leftarrow \text{good seq.}) \geqslant (1 - \gamma) = \frac{6}{7}$. By contradiction we infer that $P(\mathcal{S}_x \leftarrow \text{good seq.}) > \frac{5}{7}$, and thus $P(\mathcal{S}_x \leftarrow \text{bad seq.}) \leqslant \frac{2}{7} < \frac{1}{3}$.

We can see that the probability of obtaining a bad approximation of r is less than $\frac{1}{4}$, and the probability of using a good approximation to generate a bad sequence of simulated coin tosses is less than $\frac{1}{3}$, so the probability of $\mathcal{S}_x$ making an error must be less than $\frac{1}{4} + \frac{3}{4} \times \frac{1}{3} = \frac{1}{2}$. So $\mathcal{M}$ correctly decides A with an error probability bounded by $\frac{1}{2}$. □

7.4 Introducing a constraint

Nowhere in our study of the scatter machine, or of experimental computation in general, did we account for the amount of resources which would, presumably, be necessary to carry out experiments in the equipment. In the case of the scatter machine, resources such as energy, time, mass, volume, *etc*, are constant for each action, but if the scatter machine is to be

[3] Here, ln stands for the natural logarithm.

considered as a versatile metaphor, it is important to make a though experiment where this is not true.

We will thus introduce an *ad-hoc* mechanism to study this case. In this section we will investigate a variation of the error-free analog-digital scatter machine, called **resource-spending error-free analog-digital scatter machine**, obtained in the following way. We define that, for this machine, each action **uses** a certain number of resource units. The interesting case will only occur when the amount of used resources is high, and so we will say that action a_y uses $2^{|y|}$ resource units, where y is seen as a word in $\{1\} \cup \{0s : s \in \{0,1\}^*\}$.

This definition can be intuitively understood as saying that *the extraction of more information requires an exponential increase in the number of resources used.* We now impose that each computation is performed in a polynomial number of steps *and* with a polynomial number of resource units. We show that in this case the power of the analog-digital scatter machine is reduced.

Logarithmic precision suffices

In a similar way as before, we will show that when using a polynomial number of resource units, the resource-spending error-free analog-digital scatter machine only makes use of a bounded number of digits of the vertex position.

Lemma 7.15. *Let $\mathcal{S}_x$ be a resource-spending error-free analog-digital scatter machine. Let $\tilde{\mathcal{S}}_y$ be a resource-spending error-free machine with the same finite control, but with the vertex placed at the position $y = x \upharpoonright_{\lceil \log e \rceil}$. Then, for any input w, the state of $\mathcal{S}_x$ and $\tilde{\mathcal{S}}_y$ after using e or less resource units is the same.*

Proof. The proof is very similar to that of Lemma 7.1. Upon leaving the shooting state and having spent at most e resource units, no more than $\log e$ symbols may have been written on the action tape, and thus the position of the cannon is of the form $z = \frac{n}{2^{\log e}}$, for some $n \in \{0, 1, \ldots, 2^{\log e}\}$. Since $z \leqslant x$ if and only if $z \leqslant x \upharpoonright_{\lceil \log e \rceil}$, the answer state must be the left state in $\mathcal{S}_x$ if and only if this is the case in $\tilde{\mathcal{S}}_y$. □

This lemma allows us to obtain the following result.

Proposition 7.16. *Every set decided by a resource-spending error-free analog-digital scatter machine in polynomial time and using polynomial resources is in* P/*log*∗.

Proof. Take an arbitrary set A decided in polynomial time, and using a polynomial number n^k of resource units, by a resource-spending error-free analog-digital scatter machine $\mathcal{S}_x$. The advice function f, given by

$$f(n) = x \upharpoonright_{\lceil k \log n \rceil},$$

is a prefix function in *log*. We construct a Turing machine $\mathcal{M}$ which simulates $\mathcal{S}_x$. To simulate $\mathcal{S}_x$ on inputs of size n, $\mathcal{M}$ simulates $\mathcal{S}_x$ using the vertex position $\tilde{x} = f(\tilde{n})$, for any $\tilde{n} \geqslant n$.

Every experiment performed by $\mathcal{S}_x$ is replaced by a simple less-or-equal predicate: if $s_1 \dots s_m$ is written on the action tape, then $\mathcal{M}$ simulates a left hit if $\sum_{i=0}^{m} 2^{-i} s_i \leqslant \tilde{x}$, and a right hit otherwise. By the Lemma 7.15, and since $\mathcal{S}_x$ is assumed to use at most n^k resource units, the set decided in this way is exactly A. Because $\mathcal{S}_x$ can be simulated in polynomial time, and $\tilde{n}$ can be arbitrarily larger than n, we see that $A \in \mathrm{P}/\mathit{log}{*}$. □

Equivalence to P/*log*∗

As the reader may have intuited, resource-spending error-free analog-digital scatter machines may use the vertex as logarithmic prefix advice. This will lead to the following theorem.

Theorem 7.17. *The class of sets decided by resource-spending error-free analog-digital scatter machines in polynomial time and using polynomial resources is exactly* $\mathrm{P}/\mathit{log}{*}$.

We must still demonstrate the following inclusion:

Proposition 7.18. *Any set in* $\mathrm{P}/\mathit{log}{*}$ *can be decided by a resource-spending error-free analog-digital scatter machine in polynomial time and using polynomial resources.*

Proof. Let A be an arbitrary set in $\mathrm{P}/\mathit{log}{*}$, with the witnesses $B \in \mathrm{P}$ and prefix function $f \in \mathit{log}$ such that $|f(n)| = \lfloor a \log n + b \rfloor$ for some $a, b \in \mathbb{N}$. We show that there is a real value $0 \leqslant x \leqslant 1$ such that it is possible to obtain the value $f(n)$ from a logarithmic number of digits of x. If $f(n)$ is ultimately constant, then the result is trivial, and so we assume it is not so. Since f is a prefix function, we can consider the infinite sequence φ which is the limit sequence obtained from $f(n)$ as $n \to \infty$. Let φ_k be the k-th symbol (1 or 0) in this sequence, and set $x = \sum_{k=1}^{\infty} \varphi_k 2^{-k}$. Since f is not ultimately constant the digits in the binary expansion of x are exactly the symbols of φ.[4] Then the value $f(n)$ can be obtained from the first $\lfloor a \log n + b \rfloor$ digits of x. To decide A we construct an resource-spending error-free analog-digital scatter machine $\mathcal{S}_x$. Using the bisection method, $\mathcal{S}_x$ may obtain the required digits by making $\mathrm{O}(|f(n)|)$ experiments, each writing $\mathrm{O}(|f(n)|)$ symbols in the action tape. This amounts to polynomial time $\mathrm{O}((a \log n + b)^2) \subset \mathrm{O}(n)$ and polynomial resources $\mathrm{O}((a \log n + b) 2^{a \log n + b}) \subset \mathrm{O}(n^{a+1})$. Having obtained the value of $f(n)$ in this way, $\mathcal{S}_x$ uses the polynomial time algorithm for B. □

[4] This is the reason why we demand that f is not ultimately constant. If this were the case, then the corresponding real number r would have two binary expansions, one ending with an infinite trail of 0s, and the other with an infinite sequence of 1s.

8

Examining the results

In the previous chapter we have characterised the polynomial time computations of the analog-digital scatter machine, under different assumptions regarding the placement of the cannon. In every case, we arrive at a hypercomputational power. The class P/*poly*, for instance, contains the undecidable *tally halting set* $\{0^n : \text{the Turing machine coded by } n \text{ halts on input } 0\}$.

We will devote this chapter to analysing the implications of these results. In Section 8.1, we will describe why the scatter machine is not physically plausible, but why it is nevertheless physically and philosophically relevant. Section 8.2 will concern a weaker variant of the physical Church–Turing thesis, suggested by the scatter machine and other similar models. In Section 8.3, we show how the scatter machine can be used to form a possible argument on the structure of the Universe.

8.1 The plausibility and relevance of the Scatter Machine

As we have studied in Section 5.3, an equipment for experimental computation can achieve one of four different levels of plausibility. The scatter machine, as the reader already realised, is a highly implausible physical system, which can only be said to exist in imagination. The sub-theory T, as well as its variants T_1, T_2 and T_3, are highly abstract, and most of their idealised assumptions do not apply in reality. In order to use the scatter machine equipment as an oracle, we need to assume that we can have an infinitely sharp wedge, or at least a wedge which can be made arbitrarily sharp throughout the computation, and that the vertex is placed on a precise point x. Without these assumptions, the scatter machine becomes useless, since its computational properties arise exclusively from the value of x. The existence of an arbitrarily sharp wedge contradicts atomic theory, and for this reason the scatter machine, as described in Chapter 6, is not a valid counterexample to the physical Church–Turing thesis. We could also find other reasons.

If this is the case, then what is the relevance of the analog-digital scatter machine as a model of computation? We have two answers to this question.

Our first answer is that the scatter machine is relevant when it is seen as a metaphor, and as a *Gedankenexperiment*. Notice that the essential properties of the scatter machine do not stem from the use of barriers for deflection, or from the use of cannons for shooting the particles: the fundamental mechanism arises from *the possibility of measuring an answer to the predicate*

$y \leqslant x$, for a FIXED *real value x and a* GIVEN *dyadic rational y.* In our discussion, we could have replaced the barriers, particles, cannons and particle detectors with any other physical system with this behaviour. So the scatter machine becomes a tool to answer the more general question:

If we have a physical system to measure an answer to the predicate $y \leqslant x$, to what extent can we use this system in feasible computations?

The error or difficulty that such a measurement entails can be represented, in the scatter machine, by making assumptions on the placement of the cannon. We have given an answer to this question for the case when there is no error and for two simplistic models of error. We have also studied the case when more precise measurements require a high amount of some resource.

Using the scatter machine in this metaphorical sense, it is very interesting to notice that a system to measure the answer to the predicate $y \leqslant x$ is very limited in its computational power, unless the precision of measurement can be made arbitrarily high — such as in Sections 7.1 and 7.2. Without this proviso, the access to the information of x becomes difficult, or extremely slow. Even with controllable error, it does not appear that such a system can be feasibly used, e.g., to answer the halting problem or to solve NP-complete problems. In the first case, it would take an unfeasible number of $\mathrm{O}(2^n)$ experiments to obtain an answer for the halting problem on inputs of size n, and for the second case, it is unlikely that NP is contained in P/*poly*, because this would imply the collapse of the polynomial hierarchy [cf. Karp and Lipton, 1980].

The second answer was already hinted in Section 5.1. Our study of the scatter machine shows that certain idealisations in Newtonian mechanics imply the existence of systems in which we do not believe in.[1] This allows us to revise the applicability of the relevant sub-theories, and discover that non-computable phenomena can arise even when we bound the time, mass, volume and energy of the considered systems. We are then led to find the necessary conditions for the systems of T to implement only Turing computable functions. At this point, it should be plain to the reader that if $x \in [0, 1]$ is a computable real number, in the sense of Turing [1936], then the analog-digital scatter machine $\mathcal{S}_x$ can be simulated by an ordinary Turing machine, and the hypercomputational power is lost. This has led Beggs and Tucker [2007a] to consider whether it makes sense to introduce concepts of classical computability into physical theories. Such a bold idea is the result of studying experimental computation, and it is supported by the same arguments presented in Section 4.3 for the Church–Turing criterion.

The author believes that parts I and II of this dissertation provide a good guide for one way to explore the relationship between physics and computability. Thus far, we have refrained from stating the author's personal opinion on the possibility of hypercomputation. This was done consciously and purposefully, for three reasons. First, the presented work is the outcome of collaborations and discussions with different people, who do not always share the author's

[1] The idea of a *real* scatter machine is patently absurd, even comical. Still, it does provide bad science fiction with a promising number of dialogues (*"Is the giant triangle aligned for hypercomputation?" — "Aye, Captain!" — "Then command the cannon to shoot at the prearranged dyadic rational!"*).

opinion on the plausibility of hypercomputation. Second, the author's personal opinion is precisely and only that; being an opinion on an open problem, it is necessarily based on faith, vague intuitions, partial results, and poetic inclination. The third, and most important reason, is that the purpose of this text is not to persuade the reader that hypercomputers may or may not exist, but to convince that *the study of hypercomputational physical phenomena is interesting and important,* since the existence or non-existence of these phenomena carries important implications for physics, computability and the philosophy of science.

Having said that, we will reveal our opinion, because it is pertinent for this section to do so. The author personally believes that the simulation thesis, and by implication every thesis presented so far, is true. A certain faith that the rules of the universe can be fully understood, as mentioned in Section 4.3, forms the basis of our belief. The partial results of Turing [1936], Gandy [1980] and Sieg [2007] for the human computor and the Church–Turing theses offer some, preliminary evidence for the much stronger simulation thesis. Our main reason, however, for believing in the simulation thesis, is a highly unscientific feeling of awe towards the mathematical notion of computability, a recurring surprise that such a characterisation was found, and a compulsive admiration for those who found it. These motives lead the author to believe that the notion of computability conceals a profound truth about the Universe, and the simulation thesis is the most polished expression of what this truth might be.

We could be drastically wrong, however. The point to be emphasised is that the relationship between computability and physics should be investigated further, and that the three methods which we have presented can be used to guide such an investigation. The scatter machine, for its simplicity and clarity, serves as an ideal introduction to such research, and thus becomes relevant, irrespectively of its physical plausibility.

8.2 A weaker variant of the physical Church–Turing thesis

Consider the hypothesis, which we believe to be false, that the scatter machine, as an equipment, could exist in practice. Should we then procure an adequate construction language, we would likely find that it could only specify systems where the vertex of the wedge is placed at computable points. This is because current technology is clearly *not* hypercomputational, and it would thus presumably be unable to overview the placement of the wedge at a non-computable position.

This hypothetical situation illustrates the case of a hypercomputer that can exist in practice, but not in actuality. There are other examples of abstract physical systems in the same circumstances, such as Hava Siegelmann's [1999] suggestion of a system of mirrors to simulate Analog Recurrent Neural Networks, and Pour-el and Richard's non-computable wave function [cf. Beggs and Tucker, 2004, Section 6.1]. The type of systems that, in the words of Martin Davis [2006a], have no more non-computability than the non-computability which is *built into them*, suggests the following weaker variant of the physical Church–Turing thesis.

The weak physical Church–Turing thesis
Every function computed by an effectively buildable finite physical system can be computed by a Turing machine.

With this thesis, we recover the useful and vague notion of *effectiveness*, but now say that it is the construction of the system which should be effective, rather than the behaviour itself.

The weak physical Church–Turing thesis concerns most especially the relativistic computer of Mark Hogarth [Hogarth, 1994, Etesi and Németi, 2002, Welch, 2006], and the quantum adiabatic computer of Kieu [Kieu, 2002, 2003]. These are the candidate possible violations of the weak physical Church–Turing thesis. It seems that the described systems could exist in theory, but this may be due to overly generous theoretical assumptions, such those of Section 6.1 above. It is a matter of great debate whether these systems can exist in practice [see Tsirelson, 2001, Hodges, 2005, Smith, 2006b, Hagar and Korolev, 2006, Kieu, 2001, 2006a,b, Németi and Dávid, 2006]. This debate would surely benefit from a thorough study in the style of Beggs and Tucker.

If it does become settled that the systems are in the experimental domain of some theory, it is still unclear, and this is the crux of the weak physical Church–Turing thesis, whether we can actually build these systems. In the case of the relativistic and quantum adiabatic computers, however, there is no part of the system embedding a non-computable set or number, in the way that the vertex position x does for the scatter machine. We then expect that if construction is revealed impossible, it should be for other, more interesting reasons [e.g., regarding the relativistic computer, see Shagrir and Pitowsky, 2003, p. 88].

8.3 An unexpected argument on the structure of the Universe

Real numbers and real-valued quantities are used in physics to describe our Universe, in spite of most modern physical theories imposing an upper bound (such as the Planck length) on the precision of measurements. These bounds lead some physicists to believe that continuity is only an idealised abstraction of the actual structure of the Universe, defending the idea that the universe has a discrete structure, somewhat like a grid. Other physicists believe that the assumption of continuity is essential to explain certain phenomena.

While the debate has lasted for several years, we are nowhere near a definite answer, and the consequences of each assumption are not well understood. In particular, it is difficult to comprehend the significance of this issue for computation, i.e., whether the assumption of continuity rules out simulability, and if a grid-like universe must necessarily be "computable". In this section we show how the physical Church–Turing thesis, along with our results on the scatter machine, may be used as an argument against the existence of *fixed* real-valued quantities in the universe.

We have made clear, in Section 8.1, that the computational properties of the scatter machine emerge from the possibility of measuring whether a fixed non-computable real value x is smaller or equal to a given dyadic rational y. We may then conclude, if we assume that the physical

Church–Turing thesis is true, that no system with this possibility will be a model of a correct physical theory. In other words, *if the physical Church–Turing thesis is true, then there exists no real-valued physical quantity which is simultaneously fixed, non-computable and measurable.*

This is, modulo the physical Church–Turing thesis, a statement on the structure of the universe, but it may be slightly improved. It is not unreasonable to assume that if such a fixed real-valued physical quantity exists, then it is not computable. Such an assumption, not being necessarily true, is sensible by a simple argument of cardinality — it would, indeed, be a tremendous coincidence if any fixed real-valued quantity belonged to the countable set of computable real numbers. We could also argue that if a physical quantity is fixed, then we can probably measure it in some way, and using the scatter machine as a metaphor, we see that we do not need to be able to measure this quantity with absolute precision. A measurement with increasing precision, or even a probabilistic measurement, should suffice to result in non-computable behaviour.

These two assumptions carry some amount of uncertainty and speculation, but they are tenable, and even natural. The interesting outcome is that they imply the following unusual proposition.

If the physical Church–Turing thesis is true,
then there are no fixed real-valued quantities in the universe.

This proposition is supported by the idea, defended by many physicists, that space-time has a discrete structure [see, e.g., Gibbs, 1995], but it can also be taken as evidence in favour of this idea. The compelling, novelty aspect is that a thesis on the limits of computation seems to have implications on the nature of reality.

This is another important perspective which arises in the study of experimental computation. If until now the relationship between physics and computation was mostly dominated by the limits which the former imposes on the later, this interaction can now be reversed. There are already some examples of this turnaround, e.g., Nielsen [1997] writes:

> *Acceptance of the Church-Turing thesis therefore forces us to conclude that [...] only a limited class of observables correspond to measurements which may be performed, in principle, on quantum mechanical systems.*

We expect that physically relevant variants of the Church–Turing thesis and the solid framework of experimental computation can now be used in order to make similar statements, and to gauge the plausibility and applicability of physical theories. This offers, to our knowledge, a novel way of looking at computability, physics, and the interplay between the two.

Part III

Real recursive function theory

Introduction to Part III

In 1996 Cris Moore published a seminal paper, *Recursive theory on the reals and continuous-time computation* [Moore, 1996], where he defines an inductive class of vector valued functions over ℝ, aiming to provide a framework to study continuous-time phenomena from a computational perspective. This class was defined as the closure of some basic functions for the operators of composition, solving of first-order differential equations and a kind of minimalisation.

Some work was done since then, using Moore's definition [Campagnolo et al., 2000, 2002, Mycka, 2003a,b], but unfortunately some of Moore's assumptions were not very consensual among people interested in the field. Most of these controversial assumptions were consequences of Moore's attempt to bring the minimalisation operator — used in the classical recursive functions — into a continuous context. So in their Mycka and Costa [2004], Jerzy Mycka and José Félix Costa gave a similar definition of Moore's inductive class of functions, replacing minimalisation with the taking of infinite limits. We will cite both papers (Moore [1996], Mycka and Costa [2004]) when appropriate.

Restrictions of this inductive scheme will give rise to several interesting characterisations of computability [Graça and Costa, 2003, Bournez and Hainry, 2004, 2006, Campagnolo and Ojakian, 2006] and complexity [Campagnolo et al., 2002, Campagnolo, 2002, Bournez and Hainry, 2005, Mycka and Costa, 2006a,b] of real functions; this is an analogue of the studies in *sub-recursion* of classical recursion theory. Connections with other areas have appeared, e.g., the study of periodic real recursive functions and the connections between infinite time Turing machines and real recursive functions [Gomes, 2006, Gomes and Costa, 2007].

We will, however, be concerned with the more general aspects of the theory. In their papers, Mycka and Costa [Mycka and Costa, 2004, 2006c, 2007] have laid out the basics for our study. The most important or distinguishing feature of the more *general* theory, opposed to what we have called sub-recursion, is the access to unrestricted infinite limits, which can be made to work as a search operator over ℝ.

The purpose of this text is to lay out a solid foundation for the theory of real recursive functions, and survey the known results in the more general aspects of the theory. We will begin in Chapter 9 by studying function algebras and differential equations; after this preparation, we introduce the inductive definition for the class of real recursive functions, and finish the chapter with some further considerations on differential equations and infinite limits. In Chapter 10, we prove the most fundamental results in the theory, and show how we may stratify the class of real recursive functions into a hierarchy. Chapter 11 will solve the problem of universality, with a proof that there is no universal real recursive function. In Chapter 12 we show that there is an exact correspondence between the class of real recursive functions and the analytical hierarchy of predicates. Finally, in Chapter 13, we attempt some advance in the problem of collapse; this problem was believed to have been solved in [Loff et al., 2007], but an error turned up after a solid foundation was put in place — we expound the current status of this open problem.

Accreditment. The structure and form of Part III are mostly due to the author alone, but have benefited from a careful and thorough review by José Félix Costa and Jerzy Mycka. Most of the results in chapters 9 and 10 are not new. Chapters 11 and 13 are due to the author, José Félix Costa and Jerzy Mycka [Loff et al., 2007], and Chapter 12 is due to the author [Loff, 2007a]. Throughout Part III we have used ideas from Olivier Bournez, Manuel Campagnolo, José Félix Costa, Daniel Graça, Akitoshi Kawamura, Emmanuel Hainry, Cris Moore and Jerzy Mycka, despite the proofs shown here often being very different from theirs. We will always attribute each result to the authors who have originally proven it, by citing the appropriate sources.

Introduction to real recursive functions

This chapter will establish the theoretical basis for study of real recursive functions. We begin by studying function algebras and their related problems in Section 9.1. Some preliminaries on the Cauchy problem will be given in Section 9.2. The *primordial* function algebra which characterises the class of real recursive functions will be given in Section 9.3. Finally, we will make some general considerations on two specific operators in chapters 9.4 and 9.5.

9.1 Function algebras

A function algebra is a characterisation of a set of functions by the inductive closure, for some operators, of another set of functions. This concept is frequently used in recursion theory, and more recently to obtain characterisations of complexity classes [Clote, 1999].

Definition 9.1. *Let $\mathscr{F}$ be a class of functions, $\mathcal{F} \subseteq \mathscr{F}$ be a set of such functions, and $\mathcal{O} \subseteq \cup_{k\in\mathbb{N}}\{O : \mathscr{F}^k \to \mathscr{F}\}$ be a set of operators. The* ***Inductive Closure*** *of $\mathcal{F}$ for $\mathcal{O}$, written $\mathcal{A} = [\mathcal{F}; \mathcal{O}]$, is the smallest set containing $\mathcal{F}$, such that if $f_1, \ldots, f_k \in \mathcal{A}$ are in the domain of the k-ary $O \in \mathcal{O}$, then $O(f_1, \ldots, f_k) \in \mathcal{A}$. When taken together with $\mathcal{O}$, the inductive closure $[\mathcal{F}; \mathcal{O}]$ is called a* ***function algebra****.*

A function algebra is said to be ***enumerable*** *if both $\mathcal{F}$ and $\mathcal{O}$ are enumerable.*

We will simplify the notation by writing $\mathcal{A} = [\mathcal{F}; \mathcal{O}]$ to let $\mathcal{A}$ designate both the inductive closure $[\mathcal{F}; \mathcal{O}]$ and the function algebra $([\mathcal{F}; \mathcal{O}], \mathcal{O})$, when appropriate.

Remark 9.2. We show that the definition is well-founded, i.e., that for any such $\mathcal{F}$ and $\mathcal{O}$ there is a unique smallest $\mathcal{A}$ that satisfies the conditions of closure. This follows easily from Kleene's fixed point theorem. Take the complete lattice of sets of functions in $\mathscr{F}$, $L = \wp(\mathscr{F})$, under the partial order of inclusion. It is easy to see that the function $\mathfrak{f} : L \to L$, given by $\mathfrak{f}(F) = F \cup \mathcal{F} \cup \{O(f_1, \ldots, f_k) : f_1, \ldots, f_k \in F$ are in the domain of $O \in \mathcal{O}\}$, is continuous. We may conclude, by Knaster and Tarski, that there is a least fixed point of f. By Kleene's fixed point theorem,

$$\mathcal{A} = \cup_{k\in\mathbb{N}}\, \mathfrak{f}^k(\varnothing)$$

is this least fixed point. Such an $\mathcal{A}$ satisfies the closure condition since it is a fixed point of $\mathfrak{f}$, and the uniqueness condition since it is the least fixed point. $\square$

Notation 9.3. We make a liberal use of the square bracket notation, e.g., if f, g are functions, $\mathcal{F}$ is a class of functions, and $O_1, \ldots, O_k$ are operators, then we set

$$[f, g, \mathcal{F}; O_1, \ldots, O_n] = [\{f, g\} \cup \mathcal{F}; \{O_1, \ldots, O_n\}]. \qquad \square$$

Example 9.4. Consider the class $\mathscr{F}$ of partial, scalar, multiple-argument functions over $\mathbb{N} = \{0, 1, 2, \ldots\}$. Examples of functions in $\mathscr{F}$ are the zero function z, such that $z(x) = 0$; the successor function s, given by $s(x) = x + 1$; and the projection functions, where each u_i^n obeys $u_i^n(x_1, \ldots, x_n) = x_i$. Take $\mathbf{x}$ to designate an arbitrary sequence $\mathbf{x} = x_1, \ldots, x_n$. We may consider the composition operators $\mathcal{c}^m$, such that for every $g : \mathbb{N}^m \to \mathbb{N}, h_1, \ldots, h_m : \mathbb{N}^n \to \mathbb{N}$, the function $\mathcal{c}^m(g, h_1, \ldots, h_m) : \mathbb{N}^n \to \mathbb{N}$ is given by

$$\mathcal{c}^m(g, h_1, \ldots, h_m)(\mathbf{x}) = g(h_1(\mathbf{x}), \ldots, h_m(\mathbf{x})).$$

We will also consider the primitive recursion operator $\mathcal{p}$, which, for every given $f : \mathbb{N}^n \to \mathbb{N}$ and $g : \mathbb{N}^{n+2} \to \mathbb{N}$, sets

$$\mathcal{p}(g, h)(\mathbf{x}, 0) = g(\mathbf{x}) \text{ and } \mathcal{p}(g, h)(\mathbf{x}, y + 1) = h(\mathbf{x}, y, \mathcal{p}(g, h)(\mathbf{x}, y))$$

The class PRIM of **primitive recursive functions** may then be given by the function algebra PRIM $= [z, s, u_i^n; \mathcal{c}^m, \mathcal{p}]$. $\square$

It is important, in the study of a function algebra, to consider self-referential properties of the inductive closure. Function algebras provide a natural way of doing so, in particular for the case when the set of initial functions and the set of operators are countable.

Definition 9.5. *Let $\mathcal{A} = [\mathcal{F}; \mathcal{O}]$ be an enumerable function algebra, for a set $\mathcal{F} = \{f_1, f_2, \ldots\}$ of functions and a set $\mathcal{O} = \{O_1, O_2, \ldots\}$ of operators. The set $D_\mathcal{A}$ of* ***descriptions*** *of $\mathcal{A}$, is the smallest set of words in $\{\texttt{fun}, \texttt{Op}, \langle, \rangle, ,, 0, 1, \ldots, 9\}^*$, such that*

1. *$\langle \texttt{fun}, n \rangle \in D_\mathcal{A}$ for all n, and*
2. *if $d_1, \ldots, d_k \in D_\mathcal{A}$, then $\langle \texttt{Op}, n, d_1, \ldots, d_k \rangle \in D_\mathcal{A}$.*

We write $\mathcal{D}_\mathcal{A}$ to stand for the set of ***good descriptions*** *in $D_\mathcal{A}$; by good description we mean any description d such that*

1. *d is $\langle \texttt{fun}, n \rangle$ and $f_n \in \mathcal{F}$ (i.e., $\mathcal{F}$ has at least n functions), and in this case d is said to* ***describe*** *f_n, or*
2. *d is $\langle \texttt{Op}, n, d_1, \ldots, d_k \rangle$, for some good descriptions $d_1, \ldots, d_k$ which describe $g_1, \ldots, g_k$ in the domain of $O_n \in \mathcal{O}$; then d is said to describe $O(g_1, \ldots, g_k)$.*

Notation 9.6. With $\mathcal{A} = [\mathcal{F}; \mathcal{O}]$ given as above, let f denote a function in $f_n \in \mathcal{F}$ and O denote an operator in $O_n \in \mathcal{O}$. Then we write $\texttt{fun}_f$ to denote the pair $\texttt{fun}, n$, and $\texttt{Op}_O$ to denote the pair $\texttt{Op}, n$. In this sense, $\langle \texttt{fun}_f \rangle$ is the description $\langle \texttt{fun}, n \rangle$ and $\langle \texttt{Op}_O, d_1, \ldots, d_k \rangle$ is the description $\langle \texttt{Op}, n, d_1, \ldots, d_k \rangle$. $\square$

When studying a function algebra one wishes to understand the extent of functions which it contains. One then often considers a particular operator $O : \mathscr{F}^k \to \mathscr{F}$, and attempts to determine whether the function algebra is closed for O or not (meaning that the respective inductive closure is closed for O or not). Many important problems in computer science may be equated to the proof or disproof that certain function algebras are closed for certain operators, e.g., P = NP if and only if P is closed for a bounded minimization operator $\bar{\mu}$ [cf. Bellantoni, 1992, Clote, 1999], i.e., if $\mathrm{P} = [\mathrm{P}; \bar{\mu}]$.[1]

The proof that a function algebra $\mathcal{A}$ is closed for an operator may or may not be constructive. In the former case, one considers that the closure is effective.

Definition 9.7. *An enumerable function algebra $\mathcal{A}$ is said to be **effectively closed** under an operator $O : \mathscr{F}^k \to \mathscr{F}$ if there is an effective procedure which, given good descriptions of functions $f_1, \ldots, f_k \in \mathcal{A}$ in the domain of O, obtains a good description of the function $O(f_1, \ldots, f_k)$.*

Function algebras usually offer natural measures of complexity, simply by looking at the descriptions which describe a certain function. One such syntactic measure, which is frequently considered, is the number of nested applications of a certain operator, or set of operators. This is called the rank.

Definition 9.8. *Let $\mathcal{A} = [\mathcal{F}; \mathcal{O}]$ be an enumerable function algebra, with $\mathcal{O} = \{O_1, O_2, \ldots\}$, and let $\tilde{\mathcal{O}}$ be a subset of $\mathcal{O}$. The **rank of a good description** $d \in \mathcal{D}_\mathcal{A}$ for the set of operators $\tilde{\mathcal{O}}$ under the function algebra $\mathcal{A}$, $\mathrm{rk}(d)$, is inductively defined as:*

1. $\mathrm{rk}(\langle \mathtt{fun}, n\rangle) = 0$,
2. *if* $O_n \notin \tilde{\mathcal{O}}$, *then* $\mathrm{rk}(\langle \mathtt{Op}, n, d_1, \ldots, d_k\rangle) = \max(\mathrm{rk}(d_1), \ldots, \mathrm{rk}(d_k))$, *and*
3. *if* $O_n \in \tilde{\mathcal{O}}$, *then* $\mathrm{rk}(\langle \mathtt{Op}, n, d_1, \ldots, d_k\rangle) = \max(\mathrm{rk}(d_1), \ldots, \mathrm{rk}(d_k)) + 1$.

*The **rank of a function** $f \in \mathcal{A}$ for $\tilde{\mathcal{O}}$ under $\mathcal{A}$, $\mathrm{rk}(f)$, is given by:*

$$\mathrm{rk}(f) = \min\{\mathrm{rk}(d) : d \textit{ is a good description which describes } f\}$$

Notation 9.9. The manner of denoting the rank of $\tilde{\mathcal{O}}$ under $\mathcal{A}$, using the word rk, does not make the dependency in $\tilde{\mathcal{O}}$ and $\mathcal{A}$ explicit. We chose to do this so as not to over-encumber the notation. When it becomes necessary to disambiguate between different possible function algebras or operator sets, we will the denote the rank by $\mathrm{rk}^{\mathcal{A}}_{\tilde{\mathcal{O}}}$. □

Example 9.10. One may consider, in the function algebra $\mathrm{PRIM} = [z, s, u^n_i; c^m, p]$ for primitive recursive functions, the rank of the primitive recursion operator. Intuitively, the rank $\mathrm{rk}^{\mathrm{PRIM}}_p(f) \equiv \mathrm{rk}(f)$ of a function $f : \mathbb{N} \to \mathbb{N}$ is the smallest number of nested *for loops* necessary for any program to compute f without *while loops* (from the basic functions z, s, and u^n_i). □

[1] Note that P and NP are often taken as a class of subsets of $\mathbb{N}$, and not a class of functions. The problem P = NP for sets is equivalent to the problem "P = NP" for functions.

The rank of a set of operators, as a measure of complexity, is rich enough to stratify function algebras into interesting hierarchies.

Definition 9.11. *Let $\mathcal{A} = [\mathcal{F}; \mathcal{O}]$ be an enumerable function algebra, with $\mathcal{O} = \{O_1, O_2, \ldots\}$, and let $\tilde{\mathcal{O}}$ be a subset of $\mathcal{O}$. The* ***rank hierarchy*** *for $\tilde{\mathcal{O}}$ under $\mathcal{A}$ is the $\mathbb{N}$-indexed family of functions:*

$$H_n = \{f \in \mathcal{A} : \mathrm{rk}(f) \leqslant n\}.$$

Notation 9.12. We take the same precautions of Notation 9.9 so as not to burden the notation for the rank hierarchy. When we wish to disambiguate between hierarchies for different operator sets or under different function algebras, we will either index our denotation as in $H_n^{\mathcal{A},\tilde{\mathcal{O}}}$, or assign different letters to the different hierarchies. □

An important problem for such a hierarchy is whether it collapses, or degenerates, i.e., whether there is a number k such that $\mathcal{A} \subseteq H_k$. Again, many problems in computer science can be seen as a problem of rank-hierarchy degeneration. Recycling the previous example on P *vs.* NP, it can be shown that the polynomial time hierarchy collapses if and only if the rank hierarchy for a bounded minimization operator $\bar{\mu}$, under the function algebra $[\mathrm{P}; \bar{\mu}]$, collapses.

Example 9.13. The rank hierarchy for the primitive recursion operator under the function algebra $\mathrm{PRIM} = [z, s, u_i^n; c^m, p]$, $H_n \equiv H_n^{\mathrm{PRIM},p}$, can be intuitively understood as the stratification of the primitive recursive functions by the number of nested *for loops* needed to compute each function.This hierarchy is known not to collapse [cf. Meyer and Ritchie, 1967]. □

Example 9.14. The function algebra $\mathrm{PRIM}^+ = [z, s, u_i^n, +; c^m, p]$, where $+$ represents the 2-ary sum, is an alternative function algebra for the class of primitive recursive functions. This algebra gives Kalmár's class of elementary functions, $\mathcal{E}$, at the second level of the rank hierarchy for primitive recursion. I.e., $\mathcal{E}$ is equal to $H_2 \equiv H_2^{\mathrm{PRIM}^+,p}$. □

Proposition 9.15. *Let $\mathcal{A} = [\mathcal{F}; \mathcal{O}]$ be an enumerable function algebra, let $\tilde{\mathcal{O}}$ be a subset of $\mathcal{O}$, and set $\mathcal{V} = \mathcal{O} - \tilde{\mathcal{O}}$. The rank-hierarchy for $\tilde{\mathcal{O}}$ can be inductively defined by:*

1. *$H_0 = [\mathcal{F}; \mathcal{V}]$,*
2. *$I_n = H_n \cup \{\tilde{O}(f_1, \ldots, f_k) : \tilde{O} \in \tilde{\mathcal{O}}$ and $f_1, \ldots, f_k \in H_n \cap Dom(\tilde{O})\}$, and*
3. *$H_{n+1} = [I_n; \mathcal{V}]$.*

The previous proposition better illustrates the idea of the rank hierarchy: the next level of the hierarchy is obtained by allowing one further application of the operators in the operator set. We will skip the proof, which is obtained by a simple induction.

There are many typical problems of interest to a function algebra. We have already mentioned the problem of closure and the problem of collapse. Another important problem is the problem of universality.

Definition 9.16. *A binary function $f : D \times D \to R$ in an enumerable function algebra $\mathcal{A}$ is said to be* ***universal*** *if there is an effective procedure which, given a good description d of a function $g : D \to R$ in $\mathcal{A}$, will construct an element $\tilde{d} \in D$ such that, for every $x \in D$, $f(\tilde{d}, x) \simeq g(x)$.*[2]

It is thus important, in the study of a function algebra, to know if it has a universal function. The last problem which we will refer that generally concerns function algebras, is the problem of alternative characterisation. In order to understand the function algebra more clearly, it is important to find alternative ways to obtain the same class of functions, or discover that previously known classes of functions are exactly given by a function algebra.

Example 9.17. As we have seen, Meyer and Ritchie have solved the problem of collapse. It is interesting to note that the class of primitive recursive functions does not have a universal function: a simple argument based on the second recursion theorem would show that if this were the case, then PRIM would be closed under the minimisation operator, which we know is false. An alternative characterisation can be obtain by substituting the recursion operator by an iteration operator, and adding a few basic functions [Gladstone, 1971]. $\square$

Example 9.18. To realise that the collapse of a rank hierarchy may depend on the specific function algebra, and not only on the relevant operator, consider the class of partial recursive functions. If we take μ to stand for the unbounded minimization operator, the class of partial recursive functions, PREC, can be given by PREC $= [z, s, u_i^n; \mathfrak{c}, \mathfrak{p}, \mu]$. Kleene's normal form theorem implies that any partial recursive function can be given using a fixed number of primitive recursions, and so, despite the fact that the rank hierarchy for $\mathfrak{p}$ does not collapse under PRIM, the same rank hierarchy *does* collapse under the function algebra PREC. $\square$

We will below study a specific class of real-valued vector functions, which we call real recursive functions. This class is an analogue of Kleene's partial recursive functions, and it was first conceived, in a primitive form, by Cris Moore Moore [1996]. We will give a complete definition of this class in Section 9.3, using a function algebra. In the following chapter, however, we will review some basic properties of ordinary differential equations.

9.2 Weak conditions for existence and uniqueness

In the next section we will discuss a partial operator, called the differential recursion operator, which will give us solutions to very simple differential equations. It is thus in our interest to study conditions for existence and uniqueness of such solutions. Let f denote a *total* function from $\mathbb{R}^{n+1} \to \mathbb{R}^n$, and J represent a (possibly unbounded) open interval (A, B) of the real line, with $t_0 \in J$. Consider the Cauchy problem of the form

$$g(t_0) = \mathbf{g_0} \quad \partial_t g(t) = f(t, g(t)). \tag{9.1}$$

[2] Notice that this definition is only suitable given certain assumptions on the nature of the functions in $\mathcal{A}$.

A **solution** of (9.1) on J is a continuous function $g : \mathbb{R} \to \mathbb{R}^n$ which takes the value $\mathbf{g_0}$ at $t = t_0$ and satisfies the differential equation for every t in J. We will show that a few straightforward properties of f will ensure that the solution of (9.1) exists and is unique. The proofs will require some basic knowledge of Banach spaces; the unfamiliarised reader will find a good reference in [Kreyszig, 1989].

Definition 9.19 (see, e.g, Teschl [2007], Weinholtz [2000]). *A total function $f : \mathbb{R}^m \to \mathbb{R}^n$ is called **locally Lipschitz** if for every compact set $C \subset \mathbb{R}^m$ there is a constant K such that all $\mathbf{x}, \mathbf{y} \in C$ verify the Lipschitz condition*

$$\|f(\mathbf{x}) - f(\mathbf{y})\| \leqslant K\|\mathbf{x} - \mathbf{y}\|. \tag{9.2}$$

*The smallest such K is called the **Lipschitz constant** of f for C.*

Notice that the concept of Lipschitz constant is well-defined, by the compactness of C and the continuity of f and of the Euclidean norm. The local Lipschitz property implies other weaker properties, such as continuity. In fact, letting $B(\mathbf{x}, r)$ (or $\overline{B}(\mathbf{x}, r)$) denote the open (resp., closed) ball of radius r around $\mathbf{x}$, should we take an arbitrary $\varepsilon > 0$ and point $\mathbf{x} \in \mathbb{R}^m$, and let K be the Lipschitz constant of f for $C = \bar{B}(\mathbf{x}, 1)$; then $\|\mathbf{x} - \mathbf{y}\| < \frac{\varepsilon}{K}$ implies that $\|f(\mathbf{x}) - f(\mathbf{y})\| < K\|\mathbf{x} - \mathbf{y}\| < \varepsilon$.

The name *locally Lipschitz* is motivated because a total function $f : \mathbb{R}^m \to \mathbb{R}^n$ is locally Lipschitz if and only if around every point $\mathbf{z} \in \mathbb{R}^m$ there is a neighbourhood V of $\mathbf{z}$ and a constant K such that all $\mathbf{x}, \mathbf{y} \in V$ satisfy (9.2). Clearly, if F is locally Lipschitz, then we may take $V = \overline{B}(\mathbf{z}, 1)$ as this neighborhood. Now suppose that around every point $\mathbf{z} \in \mathbb{R}^m$ there is a neighbourhood $V_\mathbf{z}$ of $\mathbf{z}$ where (9.2) is satisfied — this implies that f is continuous, by a similar argument to the above paragraph. Now suppose that f is not locally Lipschitz, and take a compact set $C \subset \mathbb{R}^m$, and two sequences $\mathbf{x}_i$, $\mathbf{y}_i$ in C such that

$$\|f(\mathbf{x}_i) - f(\mathbf{y}_i)\| > 2^i\|\mathbf{x}_i - \mathbf{y}_i\|. \tag{9.3}$$

By the compactness of C, further suppose that $\mathbf{x}_i$ and $\mathbf{y}_i$ converge as $i \to \infty$, respectively to $\mathbf{x}$ and $\mathbf{y}$. Then the continuity of f implies that $f(\mathbf{x}_i) \to f(\mathbf{x})$ and $f(\mathbf{y}_i) \to f(\mathbf{y})$, and so

$$\|\mathbf{x}_i - \mathbf{y}_i\| < 2^{-i}\|f(\mathbf{x}_i) - f(\mathbf{y}_i)\| \to 0;$$

which means $\mathbf{x} = \mathbf{y}$. But then for every large enough i both $\mathbf{x}_i$ and $\mathbf{y}_i$ will be in the corresponding neighbourhood $V_\mathbf{x}$ of $\mathbf{x}$, and because $\mathbf{x}_i$ and $\mathbf{y}_i$ satisfy (9.3), then (9.2) can not be satisfied in $V_\mathbf{x}$, a contradiction.

The following simpler conditions imply the local Lipschitz property.

Theorem 9.20. *Let $f : \mathbb{R}^m \to \mathbb{R}^n$ be a total function.*

a. The function f is locally Lipschitz if and only if every closed ball verifies (9.2).
b. The function f is locally Lipschitz if and only if every closed m-cube verifies (9.2).

c. If f is everywhere differentiable and $\mathrm{d}f$ is bounded in every compact set, then f is locally Lipschitz.

d. If f is continuously differentiable, then it is locally Lipschitz.

Proof. **a.** The sufficient condition is obvious, since a closed ball is compact. Now take any compact set C. Since C is bounded, some closed ball will contain C, and the constant K which ensures that (9.2) is satisfied in this closed ball will trivially suffice to ensure that (9.2) is also satisfied for C. The equivalence **b.** is proved similarly.

c. Suppose that f is differentiable, and $\mathrm{d}f$ is bounded in every compact set. Let C be an arbitrary compact set. Recall that, for any $\mathbf{x} \in \mathbb{R}^m$, $\mathrm{d}f(\mathbf{x}) \in \mathcal{L}(\mathbb{R}^m \to \mathbb{R}^n)$ is a bounded linear operator, and its norm is given by

$$\|\mathrm{d}f(\mathbf{x})\| = \sup_{\mathbf{y}\in\mathbb{R}^m} \frac{\|\mathrm{d}f(\mathbf{x})(\mathbf{y})\|}{\|y\|}.$$

Then let M_1 be a bound for $\|\mathrm{d}f\|$ in C, which exists by hypothesis.

Set $s : \mathbb{R}^m \times \mathbb{R}^m \to [0, +\infty)$ so that

$$s(\mathbf{x}, \mathbf{y}) = \begin{cases} \frac{\|f(\mathbf{y})-f(\mathbf{x})-\mathrm{d}f(\mathbf{x})(\mathbf{y}-\mathbf{x})\|}{\|\mathbf{y}-\mathbf{x}\|} & \text{if } \mathbf{x} \neq \mathbf{y} \\ 0 & \text{otherwise.} \end{cases}$$

Then s is bounded in $C \times C$, because f is continuous and $\mathrm{d}f$ is bounded in C. So let M_2 be an upper bound for s in $C \times C$. Thus,

$$\|f(\mathbf{y}) - f(\mathbf{x})\| \leqslant \|f(\mathbf{y}) - f(\mathbf{x}) - \mathrm{d}f(\mathbf{x})(\mathbf{y} - \mathbf{x})\| + \|\mathrm{d}f(\mathbf{x})(\mathbf{y} - \mathbf{x})\| \leqslant (M_2 + M_1)\|\mathbf{y} - \mathbf{x}\|,$$

and f obeys the Lipschitz condition in C. By the arbitrary nature of C, f must be locally Lipschitz.

d. If f is continuously differentiable, then $\mathrm{d}f$ is continuous and therefore bounded on any compact set: **c.** gives us the result. □

Henceforth, in this chapter, f will always designate a total function.

Theorem 9.21 (Picard). *If f is locally Lipschitz, then there exists a solution g of (9.1) in a neighbourhood of t_0.*

Proof. Let $I = [t_0 - a, t_0 + a]$ for some $a > 0$, and let $R \subset \mathbb{R}^n$ be the closed n-cube centered on $\mathbf{g}_0$ with side of length $b > 0$. Since f is continuous and $I \times R$ is compact, let M be the maximum of f over $I \times R$, and let K be its Lipschitz constant for $I \times R$. We choose some $0 < \varepsilon < \min\{a, \frac{b}{M}, \frac{1}{K}\}$ and show that we may obtain a solution g of (9.1) in the interval $J = [t - \varepsilon, t + \varepsilon]$. Use the letter $\mathcal{C}$ to denote the space of continuous functions from J to R, under the supremum norm:

$$\|f\| = \sup_{t\in I} \|f(t)\|.$$

$\mathcal{C}$ is a Banach space, and so by the theorem of Banach, any strict contraction operator has a fixed point. We show that the operator T, given by

$$(Tg)(t) = g_0 + \int_{t_0}^{t} f(s, g(s))ds,$$

is a strict contraction on $\mathcal{C}$, for the constant $k = \varepsilon K < 1$. See that if $g \in \mathcal{C}$, then $Tg \in \mathcal{C}$, since for any $t \in J$ we have $(Tg)(t) \in R$:

$$\|(Tg)(t) - \mathbf{g}_0\| \leqslant \|g(t_0) - g_0\| + \left| \int_{t_0}^{t} \|f(s, g(s))\| ds \right| \leqslant \varepsilon M < b.$$

Additionally, T is a strict contraction, because

$$\|Tg - Th\| \leqslant \sup_{t \in I} \int_{t_0}^{t} \|f(s, g(s)) - f(s, h(s))\| ds \leqslant \varepsilon K \|g - h\| = k\|g - h\|.$$

We conclude that if we let g denote the fixed point of T, then g is a solution of (9.1) on the interval J. □

We can see that the cube R does not need to be centered on $\mathbf{g}_0$ for the argument of the proof above to work, all that is necessary is that $\mathbf{g}_0$ is an interior point of R.

Corollary 9.22. *If f, g, b, J, and R are as in the proof above, then for any $\tilde{\mathbf{g}_0}$ with $\|\tilde{\mathbf{g}_0} - \mathbf{g}_0\| < \frac{b}{2}$, there will also be a solution $\tilde{g}$ to the problem*

$$\tilde{g}(t_0) = \tilde{\mathbf{g_0}} \quad \partial_y \tilde{g}(t) = f(t, \tilde{g}(t)),$$

and $\tilde{g}$ will be defined at least on $[t_0 - \frac{\varepsilon}{2}, t_0 + \frac{\varepsilon}{2}]$.

We will make use of the following uniqueness theorem in the remaining chapters.

Theorem 9.23. *If f is locally Lipschitz, and g is a solution of (9.1) on the interval J, then g is the unique solution of (9.1) on J.*

Proof. Suppose that there where two solutions g and $\tilde{g}$ to (9.1) on the interval J. Set $h(t) = \tilde{g}(t) - g(t)$. Let $t_0 \in [a, b] \subset J$ and let $C \subset R^n$ be an arbitrary compact set such that $g([a, b]) \subset C$ and $\tilde{g}([a, b]) \subset C$. Let K be the Lipschitz constant for f in $[a, b] \times C$, according to (9.2).

Clearly we have $h(t_0) = 0$, because both $g(t_0) = \tilde{g}(t_0) = \mathbf{g_0}$. If we denote the scalar product with $\cdot$, then, for every $t \in [a, b]$,

$$\partial_t \|h(t)\|^2 = 2h(t) \cdot \partial_t h(t) = 2(\tilde{g}(t) - g(t)) \cdot (f(t, \tilde{g}(t)) - f(t, g(t))).$$

From (9.2) we obtain

$$(\tilde{g}(t) - g(t)) \cdot (f(t, \tilde{g}(t)) - f(t, g(t))) \leqslant K(\tilde{g}(t) - g(t)) \cdot (\tilde{g}(t) - g(t)) = K\|\tilde{g}(t) - g(t))\|^2.$$

This gives us $\partial_t \|h(t)\|^2 \leqslant 2K\|h(t)\|^2$, and so we must conclude that

$$\partial_t \left(\|h(t)\|^2 \mathrm{e}^{-2Kt}\right) = (\partial_t \|h(t)\|^2)\mathrm{e}^{-2Kt} - 2K\|h(t)\|^2 \mathrm{e}^{-2Kt} \leqslant 0.$$

So we see that $\|h(t)\|^2 \mathrm{e}^{-2Kt}$ does not increase on $[a, b]$, and since $\|h(0)\| = 0$, then it follows that $\|h(t)\| = 0$ for every $t \in [a, b]$, i.e., $g = \tilde{g}$ on $[a, b]$. The proof is done for an arbitrary compact set C such that $g([a, b]) \subset C$ and $\tilde{g}([a, b]) \subset C$. Now, should we take

$$C = \{\mathbf{y} : \|\mathbf{y}\| \leqslant \max_{t \in [a,b]} \|g(t)\| + \|\tilde{g}(t)\|\},$$

then C is well defined by the continuity of g and $\tilde{g}$, and will be compact. Also, C contains $g([a, b])$ and $\tilde{g}([a, b])$. We then conclude by the previous argument that g and $\tilde{g}$ are equal on $[a, b]$, and everywhere on J, since a and b are arbitrary. □

The uniqueness theorem ensures immediately the following.

Corollary 9.24. *If f is locally Lipschitz, and $g, \tilde{g}$ are two solutions of (9.1) respectively in the intervals $J, \tilde{J}$, then $g = \tilde{g}$ on $J \cap \tilde{J}$.*

For some locally Lipschitz function f, let S denote the set of solutions of (9.1). We set A to be the infimum of $Dom(g)$ for every $g \in S$, and B will denote the supremum of $Dom(g)$ over every $g \in S$. The previous corollary will provide that the following concept is well defined.

Definition 9.25. *Let f be locally Lipschitz. Then the **maximal solution** of (9.1) is the function g defined in (A, B) such that if $\tilde{g}$ is a solution of (9.1) over some interval (a, b), then $g(t) = \tilde{g}(t)$ for all $t \in (a, b)$. The interval (A, B) is called the **maximal interval** of (9.1).*

It will be in our interest to show that the solutions of (9.1) are well-behaved in A and B.

Theorem 9.26. *Let f be locally Lipschitz, and let g denote the maximal solution of (9.1), defined on $J = (A, B)$. Then $B < +\infty$ (or $A > -\infty$) if and only if $\lim_{t \to B^-} \|g(t)\| = +\infty$ (resp. $\lim_{t \to A^+} \|g(t)\| = +\infty$).*

Proof. If $\lim_{t \to \tilde{t}^-} \|g(t)\| = +\infty$ for some finite $\tilde{t}$, then g can not respect the equality (9.1) at point $\tilde{t}$, and thus $\tilde{t} = B < +\infty$. For the converse, suppose that $B < +\infty$, but that $\lim_{t \to B^-} g(t) = \mathbf{g}_1 \in \mathbb{R}^n$. Then the Cauchy problem

$$\tilde{g}(B) = \mathbf{g_1} \quad \partial_y \tilde{g}(t) = f(t, \tilde{g}(t)) \tag{9.4}$$

would have a solution $\tilde{g}$ on some neighbourhood $[B - \varepsilon, B + \varepsilon]$ of B, by Theorem 9.21. But then, the function $\hat{g}$ given by $\hat{g}(t) = g(t)$ for $t < B$, $\hat{g}(t) = \tilde{g}(t)$ for $B \leqslant t < B + \varepsilon$ testifies that g is not the maximal solution. By contradiction, we conclude that either $B = +\infty$, or that $\lim_{t \to B^-} g(t)$ is undefined.

But it can not be the case that $\lim_{t \to B^-} g(t)$ is undefined, while $\|g(t)\|$ is bounded for t sufficiently close to B, as we will now show, again by *reductio ad absurdum.* Suppose that this where the case, i.e., that for some sequence $t_i < B$ we would have $t_i \to B$ as $i \to \infty$, but $g(t_i)$ would diverge, while being bounded in the norm. By the boundedness of $\|g(t_i)\|$ and the convergence (thus boundedness) of t_i, take some compact set C containing $(t_i, g(t_i))$, and let $M = \max_{(t,\mathbf{z})\in C} \|f(t, \mathbf{z})\|$. M is well defined by continuity of f and compactness of C. Choose some sub-sesequence $\tilde{t}_i$ of t_i such that

$$\sum_{i=1}^{\infty} |\tilde{t}_{i+1} - \tilde{t}_i| < \infty, \tag{9.5}$$

which is ensured by convergence of t_i, and also such that, for some fixed $\delta > 0$,

$$\|g(\tilde{t}_i) - g(\tilde{t}_j)\| > \delta;$$

this is ensured by the divergence of $g(t_i)$. The mean value theorem brings us to

$$\delta < \|g(\tilde{t}_i) - g(\tilde{t}_j)\| \leqslant M|\tilde{t}_i - \tilde{t}_j|.$$

But taking the sum with the proper indexes, this contradicts (9.5). So we conclude that divergence of $g(t_i)$ implies unboundedness of $\|g(t_i)\|$, and so $\lim_{t\to B^-} \|g(t)\| = +\infty$. We proceed in a similar way for A. □

9.3 The class of real recursive functions

We are now prepared to describe our function algebra. We take $\mathscr{F}$ to be the class of partial, vector-valued, multiple argument functions over $\mathbb{R}$, i.e., the class of partial functions $f : \mathbb{R}^m \to \mathbb{R}^n$ for some $m, n \in \mathbb{N}$. We accept functions of arity 0, and call them **constants**, or **values**.

There will be two kinds of basic functions: the constant functions, and the projections. The constant functions are denoted -1^n, 0^n, and 1^n, for every $n = 0, 1, 2, \ldots$, and are given by $-1^n(x_1, \ldots, x_n) = -1$, $0^n(x_1, \ldots, x_n) = 0$ and $1^n(x_1, \ldots, x_n) = 1$. The projections are denoted with U_i^n, for each $n = 1, 2, \ldots$ and $1 \leqslant i \leqslant n$; they are given by $\mathrm{U}_i^n(x_1, \ldots, x_n) = x_i$.

The class will be closed under a countable number of partial operators over $\mathscr{F}$. The first operator is the **composition operator**, denoted by $\mathbf{C}$. Given two functions $f : \mathbb{R}^k \to \mathbb{R}^n$ and $g : \mathbb{R}^m \to \mathbb{R}^k$, the function $\mathbf{C}(f, g)$ goes from $\mathbb{R}^m$ to $\mathbb{R}^n$, and is given by

$$\mathbf{C}(f, g)(\mathbf{x}) = f(g(\mathbf{x})), \text{ for every } \mathbf{x} \in \mathbb{R}^m.$$

The domain of $\mathbf{C}(f, g)$ is $Dom(\mathbf{C}(f, g)) = \{\mathbf{x} \in Dom(g) : g(\mathbf{x}) \in Dom(f)\}$

Our second operator is the **differential recursion operator**, denoted with $\mathbf{R}$. Let $f : \mathbb{R}^{n+1} \to \mathbb{R}^n$ be a total locally Lipschitz function. Consider, for a fixed $\mathbf{x} \in \mathbb{R}^m$, the Cauchy problem

$$g(\mathbf{x}, 0) = \mathbf{x} \quad \partial_t g(\mathbf{x}, t) = f(t, g(\mathbf{x}, t)). \tag{9.6}$$

Then $\mathbf{R}(f)$ is a function from $\mathbb{R}^{n+1}$ to $\mathbb{R}^n$. For every fixed $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{R}(f)(\mathbf{x}, t) = g(\mathbf{x}, t)$, where $g(\mathbf{x}, \cdot)$ is the maximal solution of (9.6). The domain of $\mathbf{R}(f)$ is $Dom(\mathbf{R}(f)) = \{(\mathbf{x}, t) : \mathbf{x} \in \mathbb{R}^m, A(\mathbf{x}) < t < B(\mathbf{x})\}$, where A, B give the extrema of the maximal interval. In the next chapter we will show that the concept of differential recursion is well founded, and provide a few examples.

Following this we have the **infinite supremum limit operator**, denoted by $\mathbf{Ls}$. This operator takes any function $f : \mathbb{R}^{m+1} \to \mathbb{R}^n$, and maps it into the component-wise infinite supremum limit, i.e., for every $i = 1, \ldots, n$,

$$(\mathbf{Ls}(f)(\mathbf{x}))_i = \limsup_{y \to \infty} (f(\mathbf{x}, y))_i \, .$$

For the sake of abbreviation, we write simply

$$\mathbf{Ls}(f)(\mathbf{x}) = \limsup_{y \to \infty} f(\mathbf{x}, y).$$

Then $Dom(\mathbf{Ls}(f)) = \{\mathbf{x} \in \mathbb{R}^m : \limsup_{y \to \infty} f(\mathbf{x}, y) \text{ exists}\}$. We will further discuss the $\mathbf{Ls}$ operator in Section 9.5.

The final operator is called the **aggregation operator**, denoted by the symbol $\mathbf{V}$. The aggregation operator takes two functions $f : \mathbb{R}^m \to \mathbb{R}^k$ and $g : \mathbb{R}^m \to \mathbb{R}^n$ and joins them into a single vector function $\mathbf{V}(f, g) : \mathbb{R}^m \to \mathbb{R}^{k+n}$. As expected, this is given by

$$\mathbf{V}(f, g)(\mathbf{x}) = (f(\mathbf{x}), g(\mathbf{x})),$$

and $Dom(\mathbf{V}(f, g)) = Dom(f) \cap Dom(g)$.

We end this section with the central definition of this text.

Definition 9.27. *The class of real recursive functions, denoted by* $\mathrm{REC}(\mathbb{R})$*, is given by the function algebra*

$$\mathrm{REC}(\mathbb{R}) = [-1^n, 0^n, 1^n, \mathrm{U}_i^n; \mathbf{C}, \mathbf{R}, \mathbf{Ls}, \mathbf{V}].$$

We will use the letter Η (capital eta — not to be confused with the Latin capital h: $H \neq \mathrm{H}$) to denote $\mathrm{REC}(\mathbb{R})$ given by this specific structure. We will, bellow, give alternative characterisations of $\mathrm{REC}(\mathbb{R})$, using other algebraic structures; these will also be denoted by a capital Greek letter.

9.4 More on differential recursion

The operator of differential recursion is an attempt to mimic the functioning of an idealised disk-and-wheel integrator. Such integrators have been invented in the nineteenth century by Lord Kelvin [cf. Williams, 1997], and have been used to implement the famous differential analyser of Vannevar Bush Bush and Hazen [1931]. To those who have studied real recursive functions, this has always given a certain sensation of security: the well-foundedness and good-behaviour of the differential recursion operator was freely assumed.

Recently, however, the well-foundedness of differential recursion was no longer a matter of consensual agreement. Authors, such as Akitoshi Kawamura Kawamura [2007], and referees of ours, have remarked that differential recursion needs a more thorough and precise treatment. The paper [Kawamura, 2007] provides a successful attempt, with a much stronger differential recursion operator than that shown above. In this dissertation, we have opted to limit the scope of the differential recursion operator. As stated in the previous chapter, $\mathbf{R}(f)$ will only be defined when f is a total locally Lipschitz function. This will ensure that whenever $f \in Dom(\mathbf{R})$, the solution $\mathbf{R}(f)$ exists (Theorem 9.21), is unique (Theorem 9.23), and has a good behaviour in the extremities of its domain of definition (Theorem 9.26). We then see that the operator $\mathbf{R}$ is well-defined and well-behaved.

We *could* have imposed weaker conditions and obtain similar properties, but in our every attempt these would become too technical and complex. Local Lipschitz conditions, on the other hand, are considered in any standard text on ordinary differential equations [e.g. Driver, 1978, Walter, 1998, Weinholtz, 2000]. Furthermore, we have a few, simple ways to assess whether f will be in the domain of $\mathbf{R}$ (Theorem 9.20).

The definition of solution ensures us that $\mathbf{R}(f)$ will always be continuously differentiable in the last variable, i.e., $\mathbf{R}(f)(\mathbf{x}, \cdot)$ is continuously differentiable for every fixed $\mathbf{x}$. We show the following stronger result.

Theorem 9.28. *For any $f : \mathbb{R}^{n+1} \to \mathbb{R}^n$ in the domain of $\mathbf{R}$, $g = \mathbf{R}(f)$ is locally Lipschitz.*

The proof will require the use of the following lemma.

Lemma 9.29 (Gronwall-Reid). *Let C be a given constant and $k : J \to \mathbb{R}$ a given non-negative continuous function on an interval $J \subseteq \mathbb{R}$ with $t_0 \in J$. Then, if $v : J \to [0, +\infty)$ is continuous and, for all $t \in J$,*

$$v(t) \leqslant C + \left| \int_{t_0}^{t} k(s)v(s)ds \right| ; \tag{9.7}$$

it follows that, again for all $t \in J$,

$$v(t) \leqslant C \exp\left(\int_{t_0}^{t} k(s)ds \right).$$

Proof. With $t \geqslant t_0$, the inequality (9.7) may be rewritten as

$$k(t)v(t) - k(t)\left(C + \left| \int_{t_0}^{t} k(s)v(s)ds \right| \right) \leqslant 0.$$

Taking $Q(t) = C + \left| \int_{t_0}^{t} k(s)v(s)ds \right|$, we again rewrite (9.7) as $\partial_t Q(t) - k(t)Q(t) \leqslant 0$. Multiplying $Q(t)$ by $\exp(-\int_{t_0}^{t} k(s)ds)$, this implies

$$\partial_t \left(Q(t) \exp \left(- \int_{t_0}^t k(s) ds \right) \right) \leqslant 0.$$

Seeing that $Q(t_0) = C$, and integrating this last inequality from t_0 to t, we get

$$Q(t) \exp \left(- \int_{t_0}^t k(s) ds \right) - C \leqslant 0 \iff Q(t) \leqslant C \exp \left(\int_{t_0}^t k(s) ds \right).$$

From (9.7) and the definition of $Q(t)$, we obtain the intended result. For $t < t_0$, the result is proved similarly. □

Proof (of Theorem 9.28). Let $J \subset \mathbb{R}$ be a closed interval with diameter d_J. Choose an arbitrary $\mathbf{x} \in \mathbb{R}^n$. Use $S_\mathbf{x}$ to denote the unit neighborhood of the graph $(t, g(\mathbf{x}, t))$ when t ranges over J:

$$S_\mathbf{x} = \{(t, \mathbf{z}) : t \in J, \|\mathbf{z} - g(\mathbf{x}, t)\| < 1\}.$$

By the continuity of g in its last variable, then $S_\mathbf{x}$ is compact, and so f has a Lipschitz constant in $S_\mathbf{x}$, say $K_\mathbf{x}$.

Suppose that for some $\tilde{\mathbf{x}} \in \mathbb{R}^n$, $0 < \delta < 1$, we have $\|\tilde{\mathbf{x}} - \mathbf{x}\| < \delta$. Then

$$\|g(\tilde{\mathbf{x}}, t) - g(\mathbf{x}, t)\| \leqslant \|\tilde{\mathbf{x}} - \mathbf{x}\| + \left| \int_0^t \|f(s, g(\tilde{\mathbf{x}}, s)) - f(s, g(\mathbf{x}, s))\| ds \right|.$$

As long as $g(\tilde{\mathbf{x}}, t)$ remains on $S_\mathbf{x}$, we have, by the previous Lemma,

$$\|g(\tilde{\mathbf{x}}, t) - g(\mathbf{x}, t)\| \leqslant \delta + \left| \int_0^t K_\mathbf{x} \|g(\tilde{\mathbf{x}}, s) - g(\mathbf{x}, s)\| ds \right| \leqslant \delta \exp(K_\mathbf{x} |t|).$$

But if we choose $\delta < \exp(-K_\mathbf{x} d_J)$, then we conclude that $g(\tilde{\mathbf{x}}, t)$ remains in $S_\mathbf{x}$ for all $t \in J$. Furthermore, if for any given $\varepsilon > 0$ we set $\delta < \min(\varepsilon, 1) \exp(-K_\mathbf{x} d_J)$, then by the previous equation we get

$$\|g(\tilde{\mathbf{x}}, t) - g(\mathbf{x}, t)\| < \varepsilon \text{ for all } t \in J.$$

With this, we have proved that g is continuous in $\mathbb{R}^n \times J$, and thus continuous in $\mathbb{R}^{n+1}$ because J is arbitrarily chosen. So let $C \subset \mathbb{R}^n$ be a compact set. Because g is continuous, $D = J \times g(C \times J)$ is also compact, and so let K_f be the Lipschitz constant of f in D, and M_f be the maximum of $\|f\|$ in D. Now, choosing arbitrary $\mathbf{x}, \tilde{\mathbf{x}} \in C$ we may repeat the previous argument using the Lipschitz constant K_f which does not depend on $\mathbf{x}$. For any $t \in J$, we find

$$\|g(\tilde{\mathbf{x}}, t) - g(\mathbf{x}, t)\| \leqslant \|\tilde{\mathbf{x}} - \mathbf{x}\| + \left| \int_0^t K_f \|g(\tilde{\mathbf{x}}, s) - g(\mathbf{x}, s)\| ds \right| \leqslant \|\tilde{\mathbf{x}} - \mathbf{x}\| \exp(K_f d_J).$$

Also, if instead we fix $\mathbf{x}$ and choose arbitrary $t, \tilde{t} \in J$, we get

$$\|g(\mathbf{x}, \tilde{t}) - g(\mathbf{x}, t)\| \leqslant \left| \int_{\tilde{t}}^t \|f(s, g(\mathbf{x}, s))\| ds \right| \leqslant |\tilde{t} - t| M_f$$

And so $K_g = 2(\exp(K_f d_J) + M_f)$ bounds the Lipschitz constant of g for $C \times J$, because for any $(\mathbf{x}, t), (\tilde{\mathbf{x}}, \tilde{t}) \in C \times J$,

$$\|g(\tilde{\mathbf{x}}, \tilde{t}) - g(\mathbf{x}, t)\| \leqslant \|g(\tilde{\mathbf{x}}, \tilde{t}) - g(\mathbf{x}, \tilde{t})\| + \|g(\mathbf{x}, \tilde{t}) - g(\mathbf{x}, t)\| \leqslant \|(\tilde{\mathbf{x}}, \tilde{t}) - (\mathbf{x}, t)\| K_g. \qquad \square$$

We conclude that the values of the solutions have a continuous dependence on the initial condition. We will show that the same happens for the domain of definition.

Theorem 9.30. *Let $f : \mathbb{R}^{n+1} \to \mathbb{R}^n$ be in the domain of $\mathbf{R}$, and set $g = \mathbf{R}(f)$. Then the functions $A : \mathbb{R}^n \to \mathbb{R}^- \cup \{-\infty\}$, $B : \mathbb{R}^n \to \mathbb{R}^+ \cup \{+\infty\}$ such that $Dom(g(\mathbf{x}, \cdot)) = (A(\mathbf{x}), B(\mathbf{x}))$ are continuous.*

Proof. Begin by noticing that as a result of Theorems 9.21 and 9.23, the functions A and B are well-defined. So let $\mathbf{x}_i \to \mathbf{x}$, and set $B = B(\mathbf{x})$. By Theorem 9.26, we have two cases: either (I) $B = +\infty$ or (II) $\lim_{t \to B^-} \|g(\mathbf{x}, t)\| = +\infty$.

Consider case (I), and set $t_i = B(\mathbf{x}_i)$. Suppose, by contradiction, that $|t_i| \not\to +\infty$ — either because t_i is bounded, or because it oscillates. We may then assume, without loss of generality, that $t_i \to T$, because if this is not the case, then we may choose some sub-sequence of $\mathbf{x}_i$ which makes t_i converge. Again without loss of generality (by choice of a sub-sequence), assume that, for some constant $\varepsilon > 0$, $t_i \in J = [T - \varepsilon, T + \varepsilon]$. J can be seen as an interval where every $\mathbf{x}_i$ gives $\lim_{t \to t_i} \|g(\mathbf{x}_i, t)\| = +\infty$. Then we have, for every $\mathbf{x}_i$, that some $\tilde{t}_i \in J$ ensures that for all $t \in J$

$$\|g(\mathbf{x}_i, \tilde{t}_i) - g(\mathbf{x}, t)\| > 1. \tag{9.8}$$

We may find such a $\tilde{t}_i$ because $g(\mathbf{x}, t)$ is continuous, defined, and thus bounded in J, while every $\|g(\mathbf{x}_i, t)\|$ diverges to infinity in J. Now choose, by compactness of J, some convergent sub-sequence $\hat{t}_i$ of $\tilde{t}_i$ with $\hat{t}_i \to \hat{T} \in J$. Find the corresponding sequence $\hat{\mathbf{x}}_i = \mathbf{x}_j$ where $\hat{t}_i = \tilde{t}_j$. We conclude that $\hat{\mathbf{x}}_i \to \mathbf{x}$, $\hat{t}_i \to \hat{T}$, and yet by (9.8):

$$\|g(\hat{\mathbf{x}}_i, \hat{t}_i) - g(\mathbf{x}, \hat{T})\| > 1.$$

So g is discontinuous at point $(\mathbf{x}, \hat{t})$, which contradicts Theorem 9.28.

We may now deal with case (II). Suppose that the sequence $t_i = B(\mathbf{x}_i)$ goes to $+\infty$. Take some compact set C containing every $\mathbf{x}_i$, which exists by convergence of $\mathbf{x}_i$. Then choose some arbitrary value $B < +\infty$, and let $\delta > 0$, by continuity of g, be a value such that $\|\mathbf{y} - \tilde{\mathbf{y}}\| < \delta$ implies $\|g(\mathbf{y}, t) - g(\tilde{\mathbf{y}}, t)\| < 1$, for all $(\mathbf{y}, t), (\tilde{\mathbf{y}}, t) \in C \times [0, B]$. Choose a large enough i such that $\|\mathbf{x} - \mathbf{x}_i\| < \delta$ and $t_i > B$. Then for any $t \in [0, B]$ we have $\|g(\mathbf{x}, t) - g(\mathbf{x}_i, t)\| < 1$. Because $t_i > B$, $g(\mathbf{x}_i, \cdot)$ can not diverge on $[0, B]$, and so neither can $g(\mathbf{x}, \cdot)$. Being B arbitrarily large, we must conclude that $g(\mathbf{x}, \cdot)$ never diverges for positive values, and so $B = +\infty$. $\square$

At first sight, it might appear that we would prefer a more general differential recursion, where f would be allowed to depend on $\mathbf{x}$, or where the initial condition can be given at any point t_0. However, we will show in the next chapter that the Cauchy problem

$$g(\mathbf{x}, t_0) = g_0(\mathbf{x}) \quad \partial_t g(\mathbf{x}, t) = f(t, g(\mathbf{x}, t), \mathbf{x}) \tag{9.9}$$

can be reduced to the form (9.6).

A trivial example is the exponential function.

Example 9.31. The exponential function, exp, is trivially given by the differential recursion

$$\exp(0) = 1 \quad \partial_t \exp(t) = \exp(t).$$

In the notation of (9.9), we have $g_0 = 1$ and $f(t, x) = x$. □

We provide another simple example to give sin and cos, which is nevertheless appropriate to show the expressive power of differential recursion. In the next chapter, we will see that every function needed for the example is real recursive, and conclude that sin and cos are real recursive.

Example 9.32. Consider the differential recursion schema

$$g(0) = (0, 1) \qquad \partial_t g(t) = (g_2(t), -g_1(t)).$$

With the notation of (9.9), we have $g_0 = (0, 1)$ and $f(t, \mathbf{z}) = ((\mathbf{z})_2, -(\mathbf{z})_1)$. Easily, $f \in Dom(\mathbf{R})$. The solution can be recognised as $g = (\sin, \cos)$. □

9.5 More on the supremum limit operator

The class REC($\mathbb{R}$) is a subset of $\mathscr{F}$, i.e., it is composed of partial, multiple-argument vector functions over $\mathbb{R}$. This may cause confusion, because it is not immediately obvious how the concept of infinite supremum limit applies to such functions. We will give a rigorous, yet simple characterisation of the concept.

Any partial function $f : \mathbb{R}^m \to \mathbb{R}^n$ can be uniquely identified with its **graph**. The graph of f, G_f, is a predicate over $\mathbb{R}^{m+n}$, given by:

$$G_f(\mathbf{x}, \mathbf{z}) \iff \mathbf{x} \in Dom(f) \text{ and } \mathbf{z} = f(\mathbf{x}).$$

Given any function $f : \mathbb{R}^{m+1} \to \mathbb{R}^n$, and setting $g(\mathbf{x}, w) = \sup_{y>w} f(\mathbf{x}, y)$, the graph of g will be given by:

$$G_g(\mathbf{x}, w, \mathbf{z}) \iff (\forall y > w)\ \mathbf{z}_i \geqslant (f(\mathbf{x}, y))_i \wedge (\forall u_i < z_i)(\exists y > w)\ u_i < (f(\mathbf{x}, y))_i, \text{ for } i = 1, \dots, m.$$

In words, $\mathbf{z}$ is the supremum of $f(\mathbf{x}, \cdot)$ for $y > w$ if it is, component-wise, the least upper bound of $f(\mathbf{x}, y)$ for every $y > w$. We could further extend this symbolic expression to explicitly show its dependence on G_f:

$$G_g(\mathbf{x}, w, \mathbf{z})$$
$$\Longleftrightarrow \tag{9.10}$$
$$\underline{(\forall y > w)\exists \mathbf{v}}\ \left[\underline{G_f(\mathbf{x}, y, \mathbf{v})} \wedge \mathbf{z}_i \geqslant \mathbf{v}_i\right] \wedge (\forall u_i < \mathbf{z}_i)(\exists y > w)\exists \mathbf{v}\ \left[G_f(\mathbf{x}, y, \mathbf{v}) \wedge u_i < \mathbf{v}_i\right], \text{ for } i = 1, \ldots, m.$$

See that G_g is still the graph of a function, for if some $(\mathbf{x}, w, \mathbf{z}), G_g(\mathbf{x}, w, \mathbf{v})$, then $\mathbf{z} = \mathbf{v}$. Now, we have that

$$(\mathbf{x}, w) \in Dom(g) \iff \exists \mathbf{z}\, G_g(\mathbf{x}, w, \mathbf{z}). \tag{9.11}$$

From (9.11) and (9.10), considering most especially the underlined part of (9.10), we may see that if for some particular $(\mathbf{x}, w)$, we have $(\mathbf{x}, y) \notin Dom(f)$ for some $y > w$, then $(\mathbf{x}, w) \notin Dom(g)$. Thus, in order for $g(\mathbf{x}, w)$ to be defined, $f(\mathbf{x}, y)$ will have to be defined for every $y > w$. If $f(\mathbf{x}, y)$ is defined for all $y > w$, for some $(\mathbf{x}, w)$, then the predicate (9.10) gives the component-wise supremum.

The same will apply to the supremum limit. Given $f : \mathbb{R}^{m+1} \to \mathbb{R}^n$, setting $h = \mathbf{Ls}f$ (i.e., $h(\mathbf{x}) = \limsup_{y\to\infty} f(\mathbf{x}, y)$), we get

$$G_h(\mathbf{x}, \mathbf{z}) \iff \mathbf{z} = \lim_{w\to+\infty} \sup_{y>w} f(\mathbf{x}, y) = \lim_{w\to+\infty} g(\mathbf{x}, w).$$

Explicitating the dependence on G_g, we find

$$G_h(\mathbf{x}, \mathbf{z}) \iff (\forall \varepsilon > 0)(\exists \tilde{w} > 0)(\forall w > \tilde{w})\underline{\exists \mathbf{v}\, G_g(\mathbf{x}, w, \mathbf{v})} \wedge \|\mathbf{v} - \mathbf{z}\| < \varepsilon.$$

As we have seen, the underlined sub-predicate will not be valid unless $f(\mathbf{x}, y)$ is total for all $y > w$. Since w is universally quantified, we conclude the following.

Remark 9.33. Take any fixed $\mathbf{x} \in \mathbb{R}^m$. If $f(\mathbf{x}, y)$ is undefined for arbitrarily large y, then $\mathbf{Ls}(f)(\mathbf{x})$ will be undefined, i.e., $\mathbf{x} \notin Dom(\mathbf{Ls}(f))$.

Furthermore, in order for the supremum limit to be defined, it is required that every one of its components is defined. The following two remarks are in order:

Remark 9.34. Take any fixed $\mathbf{x} \in \mathbb{R}^m$. If *at least one component* of $f(\mathbf{x}, y)$ is undefined for arbitrarily large y, then $\mathbf{Ls}(f)(\mathbf{x})$ will be undefined, i.e., $\mathbf{x} \notin Dom(\mathbf{Ls}(f))$.

Remark 9.35. Take any fixed $\mathbf{x} \in \mathbb{R}^m$. If *at least one of* $\limsup_{y\to\infty}(f(\mathbf{x}, y))_i$ is undefined, then $\mathbf{Ls}(f)(\mathbf{x})$ will also be undefined, i.e., $\mathbf{x} \notin Dom(\mathbf{Ls}(f))$.

The study we have made here also applies to the remaining operators. Regarding undefinedness and partiality, we use the same principle as classical recursion theory:
Strict Undefinedness. *If a function is given an undefined parameter, or results in an undefined component, then the function will be undefined.*

E.g., $0 \times \perp = \perp$.

10

Basic theory

This chapter is devoted to the basic results of real recursive function theory. We will study the most elementary real recursive functions and operators in Section 10.1; in Section 10.2 we show a naturally-arising hierarchy of real recursive functions. More complex real recursive operators will be given in Section 10.3. Finally, in Section 10.4, we will consider the relationship between real recursive functions and partial recursive functionals.

10.1 A variety of real recursive functions and operators

We begin with the most basic operations over $\mathbb{R}$.

Proposition 10.1 (Moore [1996], Mycka and Costa [2004]). *The binary addition, subtraction and multiplication are real recursive.*

Proof. For addition, consider the following differential recursion scheme:

$$+(x,0)=x \quad \partial_y+(x,y)=1^2(y,+(x,y))=1.$$

Subtraction is obtained by replacing 1 with -1. For multiplication, set

$$g(x_1,x_2,0)=(x_1,x_2) \quad \partial_y g(x_1,x_2,y)=\mathbf{V}(U_3^3,0^3)(t,g(x_1,x_2,t))=((g(x_1,x_2,y))_2,0).$$

Then $g(x_1,x_2,y)=(x_1+x_2y,x_2)$ is the solution, and so

$$\times(x,y)=\mathbf{C}(U_1^2,\mathbf{C}(g,\mathbf{V}(0^2,\mathbf{V}(U_1^2,U_2^2))))(x,y)=(g(0,x,y))_1. \qquad \square$$

The expression for multiplication is not very simple, because we have not allowed ourselves to specify initial conditions that depend on the parameter $\mathbf{x}$. However, we said in the end of Section 9.4 that REC($\mathbb{R}$) is closed for a more general form of differential recursion. We will soon prove this rigorously, but just now we will continue with a few more very basic facts.

Proposition 10.2. *Take k scalar functions $f_1,\ldots,f_k:\mathbb{R}^m\to\mathbb{R}$. Then the function $F=(f_1,\ldots,f_k)$ is real recursive.*

Proof. Set $F = \mathbf{V}(f_1, \mathbf{V}(f_2, \ldots, \mathbf{V}(f_{k-1}, f_k)\ldots))$. □

This will allow us to simplify our notation. As a corollary we get:

Corollary 10.3. *For any $m, n \in \mathbb{N}$, the m-ary, n-component constants -1^m_n, 0^m_n, 1^m_n are real recursive.*

We could invent more similar assertions, such as the following two.

Proposition 10.4. *If $f : \mathbb{R}^2 \to \mathbb{R}$ is real recursive, then so is $\tilde{f} : \mathbb{R}^2 \to \mathbb{R}$, given by $\tilde{f}(x, y) = f(y, x)$*

Proof. Take $\tilde{f} = \mathbf{C}(f, \mathbf{V}(U^2_2, U^2_1))$. □

Proposition 10.5. *If $f : \mathbb{R} \to \mathbb{R}^2$ is real recursive, then so is $\tilde{f} : \mathbb{R} \to \mathbb{R}^2$, given by $\tilde{f}(x) = ((f(x))_2, (f(x))_1)$.*

Proof. Take $\tilde{f} = \mathbf{C}(\mathbf{V}(U^2_2, U^2_1), f)$. □

The main point, which we will not rigorously prove to avoid the tedious details, is that any fixed switching of components, or of the order of the arguments, or any selection of components, or a mixture of all of these things can be obtained in a straightforward way by using projections, composition and aggregation. We will take this for granted from this point forward. We may now prove the following, without excessive detail.

Proposition 10.6. *Let $t_0 \in \mathbb{R}$ be a real recursive constant, let $g_0 : \mathbb{R}^m \to \mathbb{R}^n$ be an arbitrary real recursive function, and let $f : \mathbb{R}^{m+n+1} \to \mathbb{R}^n$ be a total locally Lipschitz real recursive function. Then the maximal solution $g : \mathbb{R}^{m+1} \to \mathbb{R}^n$ of the differential equation*

$$g(\mathbf{x}, t_0) = g_0(\mathbf{x}) \quad \partial_t g(\mathbf{x}, t) = f(t, g(\mathbf{x}, t), \mathbf{x}), \tag{10.1}$$

is real recursive.

Proof. Begin by considering the following differential recursion, where $\mathbf{z}$ ranges over $\mathbb{R}^n$, and $\mathbf{x}$ over $\mathbb{R}^m$:

$$\tilde{g}(\mathbf{z}, \mathbf{x}, 0) = (\mathbf{z}, \mathbf{x}) \quad \partial_t \tilde{g}(\mathbf{z}, \mathbf{x}, t) = (f(t + t_0, \tilde{g}(\mathbf{z}, \mathbf{x}, t)), \underbrace{0, \ldots, 0}_{m}).$$

The solution exists and is unique, because the function given by $(f(t+t_0, \mathbf{v}), 0, \ldots, 0)$ is locally Lipschitz. Then the solution $\tilde{g}$ verifies $\tilde{g}(g_0(\mathbf{x}), \mathbf{x}, t - t_0) = (g(\mathbf{x}, t), \mathbf{x})$ (this is derived by a simple calculation), and so g is real recursive using composition and projections. □

By using only the addition and subtraction functions, we have obtained the more general form of differential recursion (10.1). We could now show that multiplication is real recursive, simply by displaying the differential recursion scheme $\times(x, 0) = 0$, $\partial_y \times (x, y) = x$. By showing that REC($\mathbb{R}$) is closed under a differential recursion operator of the form (10.1), we have simplified the proofs ahead.

Proposition 10.7 (Moore [1996], Mycka and Costa [2004]). *The restrictions to the domain $(0, +\infty)$ of the inverse, division and square root functions are real recursive. The exponential, logarithm, power, sine, cosine and arc-tangent functions are real recursive. The real numbers π and e are real recursive constants.*

Proof. The restricted division and logarithm functions are obtained simultaneously, through the differential recursion scheme:

$$\begin{cases} \frac{1}{1} = 1, \\ \log(1) = 1, \end{cases} \qquad \text{and} \qquad \begin{cases} \partial_x \frac{1}{x} = -1 \times \left(\frac{1}{x}\right) \times \left(\frac{1}{x}\right) = -\frac{1}{x^2}, \\ \partial_x \log(x) = \frac{1}{x}. \end{cases}$$

In the differential recursion scheme of Proposition 10.6, we have $t_0 = 1$, $g_0 = (1, 1)$ and the total, locally Lipschitz function $f(t, z_1, z_2) = (-(z_1)^2, z_1)$. The solution is, therefore, unique, and we obtain the restricted inverse function and the logarithm function, as intended. The following expressions, using the differential recursion scheme (10.1), give us the remaining functions

1. $\frac{x}{y} = x \times \frac{1}{y}$ gives us the restricted division;
2. $\mathrm{e}^0 = 1$, $\partial_x \mathrm{e}^x = \mathrm{e}^x$ solves to the exponential function;
3. $x^y = \mathrm{e}^{\log(x)y}$, where $x > 0$, is the power function;
4. $\sqrt{x} = x^{\frac{1}{2}}$ gives us the square root from the power function, restricted to positive x;
5. $(\sin, \cos)(0) = (0, 1)$, $\partial_x(\sin, \cos)(x) = (\cos, -\sin)(x)$;
6. $\arctan(0) = 0$, $\partial_x \arctan(x) = \frac{1}{x^2+1}$;
7. $\mathrm{e} = \mathrm{e}^1$;
8. $\pi = 4 \times \arctan(1)$. □

Proposition 10.8 (Moore [1996], Mycka and Costa [2004]). *Kronecker's δ and Heaviside's Θ, given by*

$$\delta(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad \Theta(x) = \begin{cases} 1 & \text{if } x \geqslant 0 \\ 0 & \text{otherwise;} \end{cases}$$

are real recursive.

Proof. Set $\delta(x) = \limsup_{y \to \infty} \left(\frac{1}{x^2 + 1}\right)^y$ and $\Theta(x) = \left(\limsup_{y \to \infty} \frac{1}{1 + 2^{-xy}}\right) + \frac{1}{2}\delta(x)$. □

Proposition 10.9 (Moore [1996], Mycka and Costa [2004]). *The sawtooth wave function, denoted by r, and the square wave function, denoted by s, are real recursive.*

Proof. The square function is given by $s(x) = \Theta(\sin(\pi x))$. We can build the sawtooth using the recursion scheme $\tilde{r}(0) = 0$ and $\partial_x \tilde{r}(x) = 2\sin(\pi x)^2 s(x) - \frac{1}{2}$. We get $r(x) = s(x)\tilde{r}(x + 1) + (1 - s(x))\tilde{r}(x)$ (cf. Fig. 10.2). □

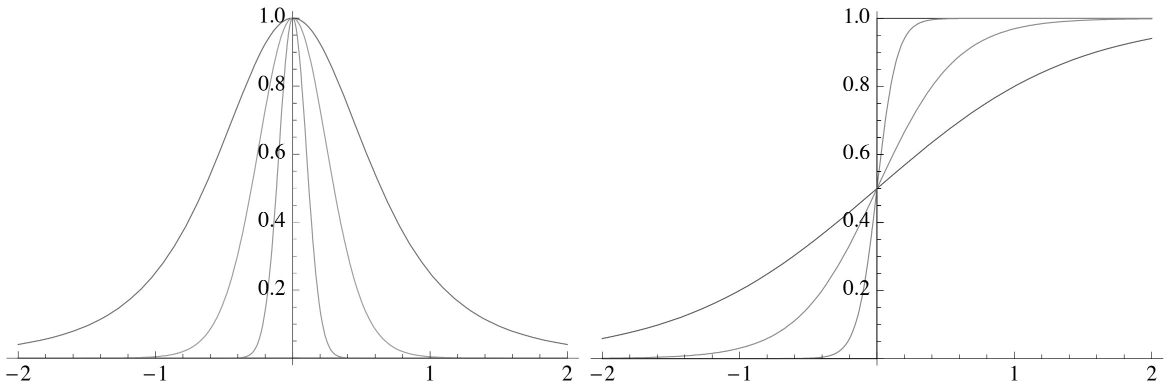
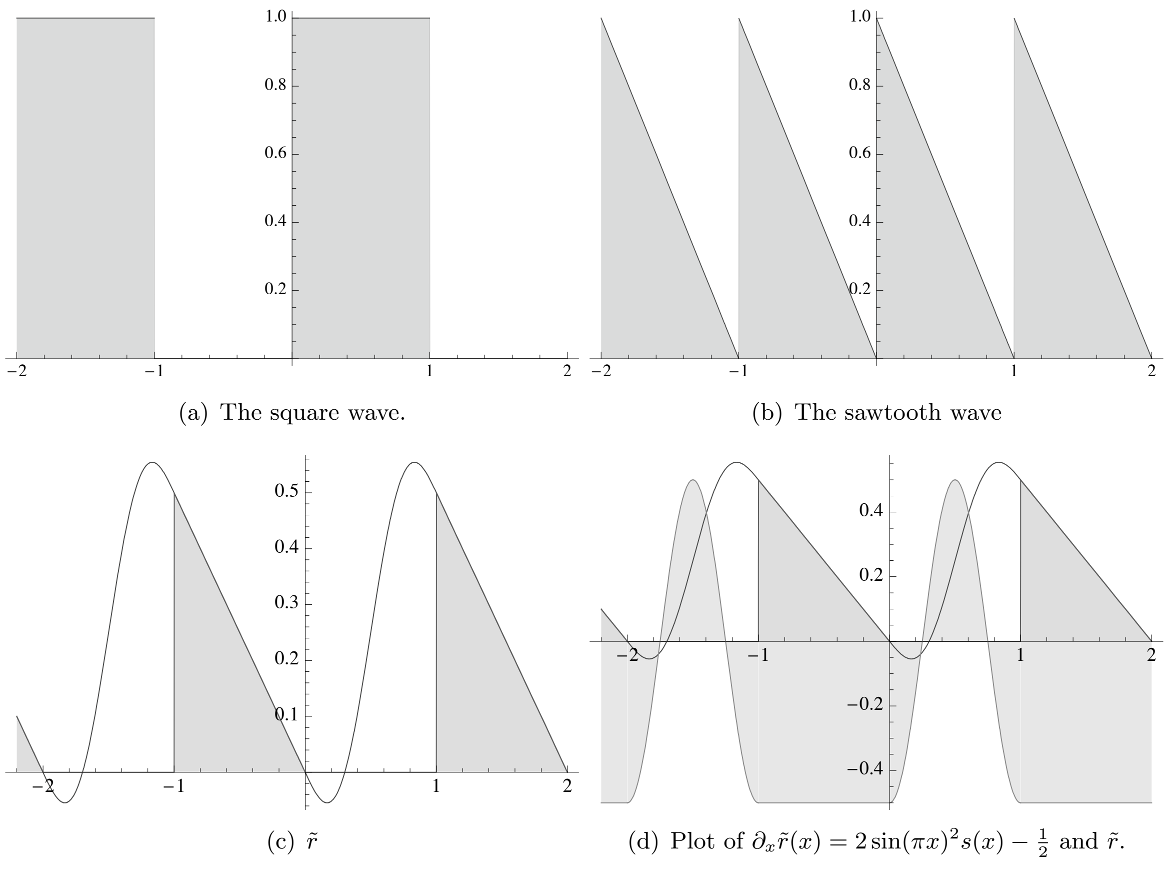


Fig. 10.1. Convergence of the shown expressions to δ and Θ.



(a) The square wave. (b) The sawtooth wave

(c) $\tilde{r}$ (d) Plot of $\partial_x\tilde{r}(x) = 2\sin(\pi x)^2 s(x) - \frac{1}{2}$ and $\tilde{r}$.

Fig. 10.2. Wave functions.

Proposition 10.10. *The characteristics* $\chi_=$ *of equality,* $\chi_\leqslant$ *of inequality and* $\chi_<$ *of proper inequality are real recursive.*

Proof. Take $\chi_=(x,y) = \delta(y-x)$, $\chi_\leqslant(x,y) = \Theta(y-x)$, and $\chi_<(x,y) = \chi_\leqslant(x,y) - \chi_=(x,y)$. □

Proposition 10.11. *The unrestricted inverse,unrestricted division and unrestricted square root functions are real recursive.*

Proof. Just take the restriction of $\frac{1}{x}$ to positive x, and set

$$\frac{1}{x} = (\chi_<(0,x) - \chi_<(x,0))\frac{1}{(\chi_<(0,x) - \chi_<(x,0))x} = \begin{cases} \frac{1}{x} & \text{if } x > 0, \\ -\frac{1}{-x} & \text{if } x < 0. \end{cases}$$

where in the left we mean the unrestricted inverse, and in the right we use the inverse restricted to positive values, which was already defined in Proposition 10.7. Unrestricted division is obtained in the same way as for the restricted case. In an analogous way, we take the restricted square root (to the right), and define an unrestricted square root (to the left):

$$\sqrt{x} = \chi_=(x,0) \times \sqrt{x + \chi_=(x,0)} = \begin{cases} \sqrt{x} & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \qquad \square$$

We will often use the characteristics of equality and inequality to define a function by cases, as in the previous proof. We will make the syntactical abbreviation $\chi_{\neq}(x,y) = 1 - \chi_=(x,y)$, $\chi_>(x,y) \equiv \chi_<(y,x)$ and $\chi_\geqslant(x,y) \equiv \chi_\leqslant(y,x)$.

Proposition 10.12. *The floor function, the ceil function, the absolute value function, the Euclidean and supremum norms over* $\mathbb{R}^n$ *are real recursive.*

Proof. We use the following expressions:

1. $\lfloor x \rfloor = x - r(-x)$;
2. $\lceil x \rceil = x + r(x)$;
3. $|x| = (2\Theta(x) - 1)x$;
4. $\|\mathbf{x}\|_2 = \sqrt{x_1^2 + \ldots + x_n^2}$;
5. $\|(x_1, x_2)\|_\infty = \chi_>(|x_1|, |x_2|)x_1 + \chi_\leqslant(|x_1|, |x_2|)x_2$;
6. $\|\mathbf{x}\|_\infty = \|(x_1, \|(x_2, \| \ldots \|(x_{n-1}, x_n)\|_\infty) \ldots \|_\infty)\|_\infty$.

Definition 10.13. *The* ***sigmoidal function****,* σ*, is given by*

$$\sigma(x) = \frac{\mathrm{e}^x}{1 + \mathrm{e}^x}.$$

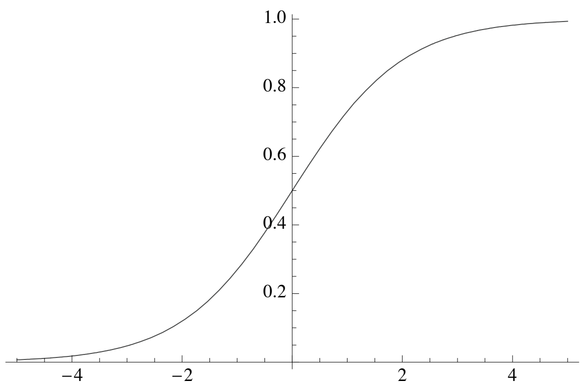


Fig. 10.3. Plot of $\sigma(x)$.

Proposition 10.14. *The sigmoidal function is real recursive. Furthermore, for any function $f : \mathbb{R}^{n+1} \to \mathbb{R}$, we have*

$$\limsup_{y\to\infty} f(\mathbf{x}, y) = z \in \mathbb{R} \iff \limsup_{y\to\infty} \sigma(f(\mathbf{x}, y)) = \sigma(z) \in (0,1),$$

$$\limsup_{y\to\infty} f(\mathbf{x}, y) = +\infty \iff \limsup_{y\to\infty} \sigma(f(\mathbf{x}, y)) = 1, \text{ and}$$

$$\limsup_{y\to\infty} f(\mathbf{x}, y) = -\infty \iff \limsup_{y\to\infty} \sigma(f(\mathbf{x}, y)) = 0.$$

Proof. The expression given for the sigmoidal function is a composition of functions that we have already shown to be real recursive, and thus σ itself must be real recursive. The first property with respect to the infinite supremum limit is a consequence of σ being a strictly increasing surjection to $(0,1)$; we find that for any $\mathbf{x} \in \mathbb{R}^n$, $y \in \mathbb{R}$,

$$\sigma(\sup_{z\geqslant y} f(\mathbf{x}, z)) = \sup_{z\geqslant y} \sigma(f(\mathbf{x}, z)).$$

We then make use of the definition of supremum limit; we know that $\limsup_{y\to\infty} f(\mathbf{x}, y) \in \mathbb{R}$ if and only if there is a real value r such that $\| \sup_{z\geqslant y} f(\mathbf{x}, z)\| < r$ for all sufficiently large y. This is equivalent, as we have seen, to $\sigma(-r) < \sup_{z\geqslant y} \sigma(f(\mathbf{x}, z)) < \sigma(r)$, for some r, which is a sufficient and necessary condition for $\limsup_{y\to\infty} \sigma(f(\mathbf{x}, y)) \in (0,1)$ to hold. So the existence of both supremum limits of f and $\sigma \circ f$ is equivalent. The additional fact that, when these limits are defined,

$$\limsup_{y\to\infty} \sigma(f(\mathbf{x}, y)) = \sigma\left(\limsup_{y\to\infty} f(\mathbf{x}, y)\right)$$

can be derived from the continuity of σ and its inverse. The remaining equivalences are proven in the same fashion. $\square$

Definition 10.15. *The infinite infimum limit operator,* $\mathbf{Li}$*, and the infinite limit operator,* $\mathbf{L}$*, are given by*

$$\mathbf{Li}(f)(\mathbf{x}) = \liminf_{y\to\infty} f(\mathbf{x}, y) \qquad \mathbf{L}(f)(\mathbf{x}) = \lim_{y\to\infty} f(\mathbf{x}, y);$$

where $f : \mathbb{R}^{m+1} \to \mathbb{R}^n$ is in $\mathscr{F}$.

Proposition 10.16 (Mycka [2003a]). REC($\mathbb{R}$) *is effectively closed under* $\mathbf{Li}$ *and* $\mathbf{L}$*.*

Proof. We set $\mathbf{Li}(f)(\mathbf{x}) = -\limsup_{y\to\infty} -f(\mathbf{x}, y)$. It is known that $\lim_{y\to\infty} f(\mathbf{x}, y)$ is defined if and only if $\limsup_{y\to\infty} f(\mathbf{x}, y)$ and $\liminf_{y\to\infty} f(\mathbf{x}, y)$ are both defined and equal, and in this case $\lim_{y\to\infty} f(\mathbf{x}) = \limsup_{y\to\infty} f(\mathbf{x}, y)$. So we set

$$\mathbf{L}(f)(\mathbf{x}) = \frac{1}{\chi_{=}(\limsup_{y\to\infty} \sigma(f(\mathbf{x}, y)), \liminf_{y\to\infty} \sigma(f(\mathbf{x}, y)))} \limsup_{y\to\infty} f(\mathbf{x}, y). \qquad \square$$

This ends the most elementary part of real recursive function theory. We have seen that a number of functions are real recursive. In Section 10.3, we will see that this class extends even further. Hopefully, the reader will nurture a growing astonishment at the expressive power of such a simple inductive definition. A full and insightful characterisation will have to wait until Chapter 12. In the next chapter, we will stratify REC($\mathbb{R}$) into a hierarchy.

10.2 The η-hierarchy

Remember from page 85 that we use the capital letter H (eta) to designate the function algebra for REC($\mathbb{R}$), given in Definition 9.27.

Definition 10.17. *The η-**hierarchy** is the rank hierarchy for the limit operator under the algebra* H *for* REC($\mathbb{R}$). *We use* H_n *to denote the n-th level of this hierarchy. In Symbols*

$$\mathrm{H}_n = H_n^{\mathrm{H},\mathbf{Ls}} = \{f \in \mathrm{REC}(\mathbb{R}) : \mathrm{rk}_{\mathbf{Ls}}^{\mathrm{H}}(f) \leqslant n\}.$$

A clearer picture for this hierarchy may be obtained from the following corollary of Proposition 9.15.

Corollary 10.18. *The η-hierarchy is inductively given by:*

1. $\mathrm{H}_0 = [-1^n, 0^n, 1^n, U_i^n; \mathbf{C}, \mathbf{R}, \mathbf{V}]$,
2. $\tilde{\mathrm{H}}_n = \mathrm{H}_n \cup \{\mathbf{Ls}(f) : f : \mathbb{R}^{m+1} \to \mathbb{R}^k$ *is in* $\mathrm{H}_n\}$, *and*
3. $\mathrm{H}_{n+1} = [\tilde{\mathrm{H}}_n; \mathbf{C}, \mathbf{R}, \mathbf{V}]$.

The following corollary comes from the proofs of the previous chapter.

Corollary 10.19. *The following functions and constants are in* H_0*:*

1. *The addition, subtraction and multiplication functions.*
2. *The inverse, division and square root functions, restricted to a positive argument.*
3. *The exponential, logarithm, power, sine, cosine and arc-tangent functions.*
4. *The numbers* π *and* e.
5. *The sigmoidal function.*

The following functions are in H_1*:*

1. *Kronecker's* δ *and Heaviside's* Θ.
2. *The sawtooth wave function* r *and the square wave function* s.
3. *The characteristics* $\chi_=$, $\chi_\leqslant$ *and* $\chi_<$.
4. *The unrestricted inverse, unrestricted division and unrestricted square root functions.*
5. *The floor, ceil, absolute value, supremum norm and euclidean norm.*

If $t_0 \in \mathbb{R}, g_0 : \mathbb{R}^m \to \mathbb{R}^n, f : \mathbb{R}^{m+n+1} \to \mathbb{R}^n$ *are in* H_i, *then the solution* g *of (10.1) is also in* H_i. *If* $f : \mathbb{R}^{m+1} \to \mathbb{R}^m$ *is in* H_i, *then* $\mathbf{Li}(f), \mathbf{L}(f)$ *are in* H_{i+1}.

Most proofs will now include the position of functions in the η-hierarchy.

10.3 Non-trivial real recursive operators

The operator **Ls** serves as an analogue to the minimalisation operator of classical recursion theory. However, this operator has a distinct feature: there is a real recursive way of telling whether or not the infinite supremum limit exists.

Definition 10.20. *For an $(m+1)$-ary function $f \in \mathscr{F}$, the η^s* ***operator*** *gives an m-ary function, $\eta^s(f)$, such that:*

$$\eta^s(f)(\mathbf{x}) = \begin{cases} 1 & \textit{if } \limsup_{y\to\infty} f(\mathbf{x},y) \textit{ exists,} \\ 0 & \textit{otherwise.} \end{cases}$$

The η^i and η operators are defined in the same way, but with respect to lim inf *and* lim.

Here we will show that we may obtain $\eta(f)$, $\eta^s(f)$ and $\eta^i(f)$ when f is a total function. In fact, REC($\mathbb{R}$) is effectively closed for these operators, i.e., our restriction on f is not required, but we will withhold the (more complicated) proof for now.

Theorem 10.21 (Mycka and Costa [2004],Loff et al. [2007]). *If f is a total function in H_i, then $\eta(f), \eta^s(f)$ and $\eta^i(f)$ are in H_{i+1}.*

Proof. We make the proof for a scalar function $f : \mathbb{R}^m \to \mathbb{R}$; the proof generalises to the vector case, by Proposition 10.2. The function $\eta^s(f)$ is given by the following real recursive expression:

$$\eta^s(f)(\mathbf{x}) = 1 - \chi_=\left(\limsup_{y\to\infty} \sigma(f(\mathbf{x},y)), 1\right) - \chi_=\left(\limsup_{y\to\infty} \sigma(f(\mathbf{x},y)), 0\right).$$

Then $\limsup_{y\to\infty} \sigma(f(\mathbf{x},y))$ exists, because $\sigma \circ f$ is a bounded total function, and $\eta^s(f)(\mathbf{x}) = 0$ if and only if $\limsup_{y\to\infty} \sigma(f(\mathbf{x},y)) \in \{0,1\}$, which provides the intended behaviour according to Lemma 10.14. A similar expression gives us $\eta^i(f)$.

We have seen that $\lim_{y\to\infty} f(\mathbf{x},y)$ exists if and only if the supremum and infimum limits of f exist and are equal. If they exist, the supremum and infimum limits of f are equal if and only if the supremum and infimum limits of $\sigma \circ f$ are equal. For this reason $\eta(f)$ can be set as

$$\eta(f)(\mathbf{x}) = \eta^s(f)(\mathbf{x}) \times \eta^i(f)(\mathbf{x}) \times \chi_=\left(\liminf_{y\to\infty} \sigma(f(\mathbf{x},y)), \limsup_{y\to\infty} \sigma(f(\mathbf{x},y))\right). \qquad \square$$

Definition 10.22. *The* ***restricted iteration operator***, $\bar{\mathbf{I}}$, *transforms an n-ary, total, locally Lipschitz function $g \in \mathscr{F}$ with n components, into a total $(n+1)$-ary function $\bar{\mathbf{I}}(g)$ with n components, given by*

$$\bar{\mathbf{I}}(g)(\mathbf{x},t) = g^{\lfloor |t| \rfloor}(\mathbf{x}) = \underbrace{g \circ g \circ \ldots \circ g}_{\lfloor |t| \rfloor \textit{ times}}(\mathbf{x}).$$

The following theorem is one of the most fundamental results in the field.

Theorem 10.23 (Mycka and Costa [2004], Loff et al. [2007]). REC($\mathbb{R}$) *is effectively closed under* $\bar{\mathbf{I}}$. *Furthermore, if* $g \in Dom(\bar{\mathbf{I}}) \cap \mathrm{H}_i$, *then* $\bar{\mathbf{I}}(g) \in \mathrm{H}_{\max(i,1)}$.

Proof. Let $g \in \mathrm{H}_i$ be an n-ary, total, locally Lipschitz function with n components. Let f be the $(1+2n)$-ary function with $2n$ components given by:

$$f(t, \mathbf{y}, \mathbf{z}) = \begin{pmatrix} (g(\mathbf{z}) - \mathbf{z})\dfrac{\pi}{2}\sin(\pi t) s(t) \\ \\ \dfrac{(\mathbf{y}-\mathbf{z})\pi\sin(\pi t)}{\cos(\pi t) - 1 + \delta(\cos(\pi t) - 1)}(1 - s(t)) \end{pmatrix}.$$

The components of f are shown using a column vector; the first line gives the first n components, and we will call these the *first part* of f; the remaining components will be called the *second part* of f. Several observations can be made:

1. The first part of f will be zero whenever t is in an interval of the form $[2k+1, 2k+2]$, and
2. The second part of f wil be zero for t in $[2k, 2k+1]$.
3. For a fixed $\mathbf{z}$, and any $\mathbf{y} : [2k, 2k+1] \to \mathbb{R}^n$, $t \in (2k, 2k+1)$,

$$\int_{2k}^{t} f(s, \mathbf{y}(s), \mathbf{z})ds = (g(\mathbf{z}) - \mathbf{z})\frac{\pi}{2}\int_{2k}^{t}\sin(\pi s)ds = (g(\mathbf{z}) - \mathbf{z})\frac{1-\cos(\pi t)}{2}.$$

4. If $\mathbf{y}$ is fixed, $\mathbf{z}(2k+1) = \mathbf{z}_0$, then $\mathbf{z}(t) = \mathbf{z}_0 + (\mathbf{y} - \mathbf{z}_0)\frac{1+\cos(\pi t)}{2}$ is the unique solution to

$$\partial_t \mathbf{z}(t) = \frac{(\mathbf{y} - \mathbf{z}(t))\pi\sin(\pi t)}{\cos(\pi t) - 1 + \delta(\cos(\pi t) - 1)}$$

   in the interval $(2k+1, 2k+2)$.
5. The function f is total locally Lipschitz, because it is the composition of a total locally Lipschitz function with g.

To understand why (3) and (4) are important, we show the plot of the solutions $\frac{1-\cos(\pi t)}{2}$ and $\frac{1+\cos(\pi t)}{2}$, and of its derivatives. Notice how these functions go from 0 to 1, and how the derivatives are locally Lipschitz.

So, should we take the differential recursion scheme

$$(\mathbf{y}, \mathbf{z})(\mathbf{x}, 0) = \mathbf{x} \qquad \partial_t(\mathbf{y}, \mathbf{z})(\mathbf{x}, t) = f(t, (\mathbf{y}, \mathbf{z})(\mathbf{x}, t))$$

where $(\mathbf{y}, \mathbf{z})$ is regarded as a function from $\mathbb{R}^{n+1}$ to $\mathbb{R}^{2n}$, we may set[1]

$$\bar{\mathbf{I}}(g)(\mathbf{x}, t) = \mathbf{y}(\mathbf{x}, 2\lfloor |t| \rfloor).$$

These functions can be explained the following way: as t changes from 0 to 1, $\mathbf{z}$ is constant and $\mathbf{y}$ goes through the distance from $\mathbf{x}$ to $g(\mathbf{x})$. For $t \in [1, 2]$, $\mathbf{y}$ is constant and $\mathbf{z}$ catches

[1] The differential recursion scheme could be changed so that the scaling $2 \times \lfloor |t| \rfloor$ would become unnecessary.

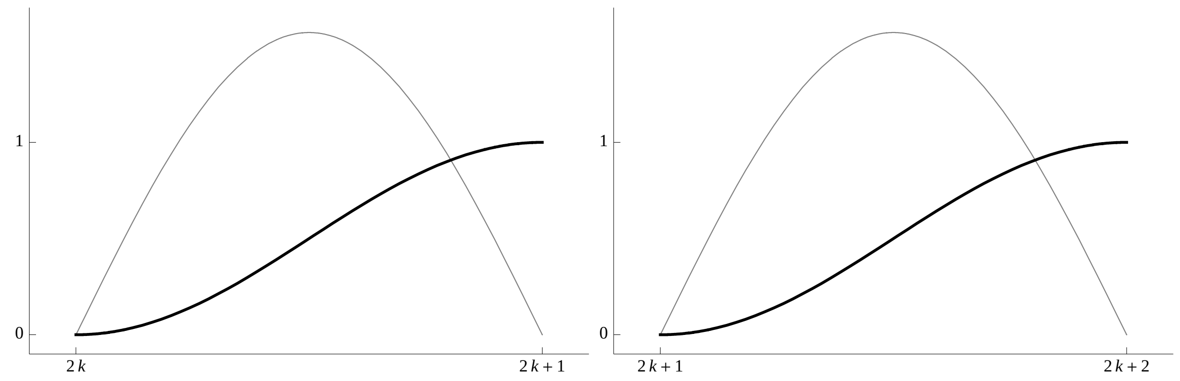


Fig. 10.4. Plot of $\frac{1-\cos(\pi t)}{2}$ between $2k$ and $2k+1$, and of $\frac{1+\cos(\pi t)}{2}$ between $2k+1$ and $2k+2$, with their respective derivatives drawn in a lighter line.

up, hence $\mathbf{z}(\mathbf{x}, 2) = \mathbf{y}(\mathbf{x}, 2) = g(\mathbf{x})$. If $t > 2$, then the same cycle begins again, and, for every $n \in \mathbb{N}$, $\bar{\mathbf{I}}(g)(\mathbf{x}, n) = \mathbf{y}(2n) = \mathbf{z}(2n)$. □

Example 10.24. You may see a plot of (y, z) for various functions in Fig. 10.5, page 101. The function y is shown in a thick line, and z is shown in a thin, dashing line. □

Definition 10.25. *The* ***restricted smooth iteration operator****,* $\tilde{\mathbf{I}}$*, takes an* n*-ary, total, locally Lipschitz function* $g \in \mathscr{F}$ *with* n *components, and gives a total* $(n+1)$*-ary locally Lipschitz function* $\tilde{\mathbf{I}}(g)$ *with* n *components, such that*

$$\bar{\mathbf{I}}(g)(\mathbf{x}, t) = \mathbf{y}(\mathbf{x}, 2t\chi_{\geqslant}(t, 0)),$$

where $\mathbf{y}$ *is given by the previous proposition.*

The restricted iteration operator is not smooth; in fact, it gives a discontinuous function.The smooth iteration operator $\tilde{\mathbf{I}}$ still verifies $\tilde{\mathbf{I}}(g)(\mathbf{x}, n) = \bar{\mathbf{I}}(g)(\mathbf{x}, n)$ for all natural n, but has the following advantage, which may be concluded from Theorem 9.28.

Theorem 10.26. *If* g *is total and locally Lipschitz, then so is* $\tilde{\mathbf{I}}(g)$*.*

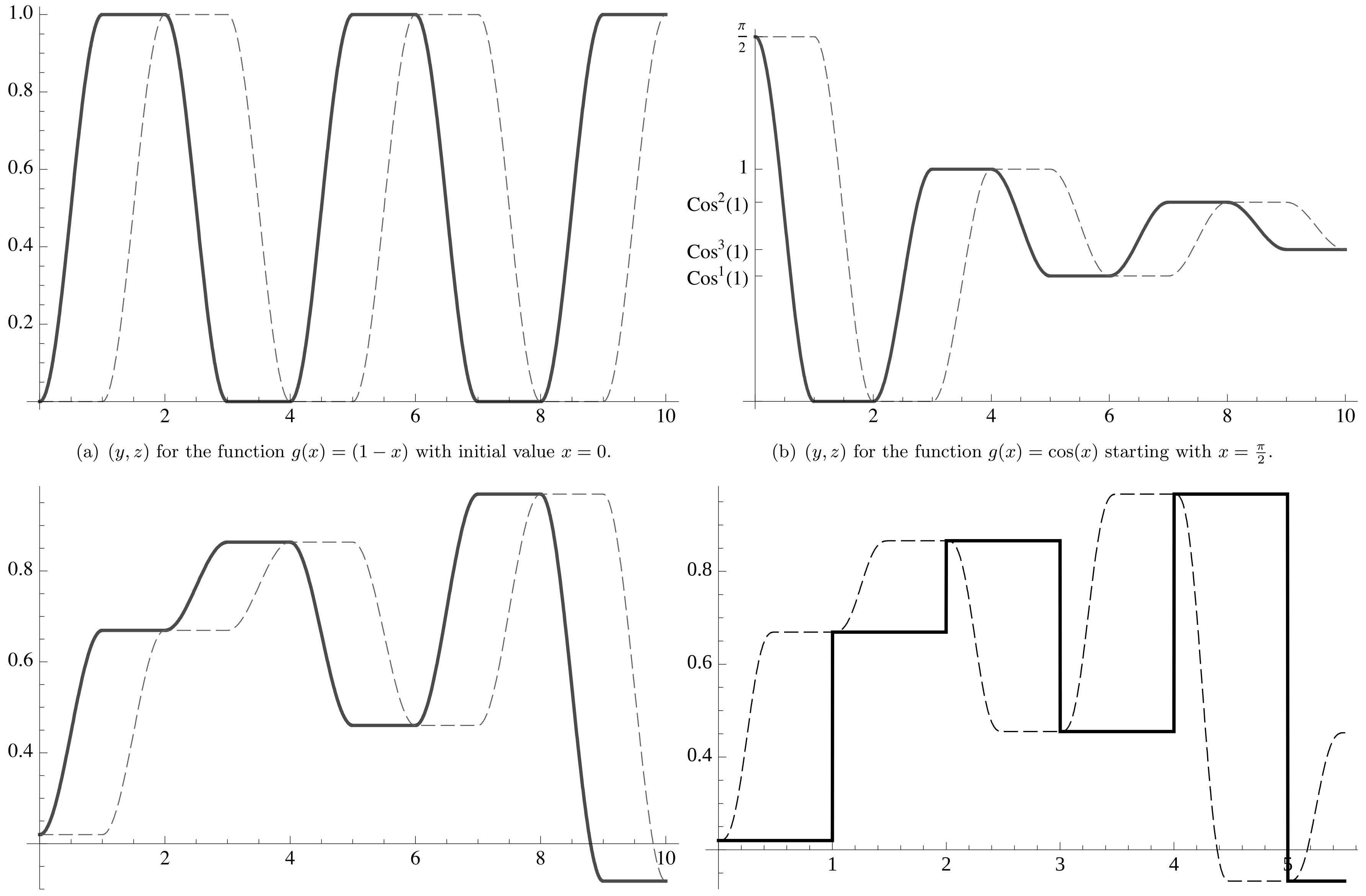
Remark 10.27. $\tilde{\mathbf{I}}(g)$ is also given by:

$$\tilde{\mathbf{I}}(g)(\mathbf{x}, t) = \begin{cases} \mathbf{x} & \text{if } t \leqslant 0, \\ g^n(\mathbf{x}) & \text{if } t \in (n + \frac{1}{2}, n+1] \text{ for some } n \in \mathbb{N}, \\ g^n(\mathbf{x})\xi + g^{n+1}(\mathbf{x})(1-\xi) & \text{if } t \in (n, n + \frac{1}{2}] \text{ for some } n \in \mathbb{N}. \end{cases}$$

Above, ξ is an abbreviation for $\frac{1-\cos(2\pi(t-n))}{2}$. □

Iteration is very useful, and very powerful.

Proposition 10.28 (Mycka and Costa [2007]). *If* g *is an* $(m+1)$*-ary total locally Lipschitz function with* n *components in* H_i*, then there are two functions* S *and* P *in* $\mathrm{H}_{\max(i,1)}$*, such that*



(a) (y, z) for the function $g(x) = (1 - x)$ with initial value $x = 0$.

(b) (y, z) for the function $g(x) = \cos(x)$ starting with $x = \frac{\pi}{2}$.

(c) (y, z) for the logistic function $g(x) = 3.9x(1 - x)$ with initial value $x = 0.22$.

(d) Plot of $\bar{\mathbf{I}}(g)$ (thick line) and $\tilde{\mathbf{I}}(g)$ (dashed line) for the logistic function $g(x) = 3.9x(1 - x)$ with initial value $x = 0.22$.

Fig. 10.5. Plot of $\mathbf{y}(t)$ and $\mathbf{z}(t)$ for various functions.

$$S(\mathbf{x}, n) = \sum_{i=1}^{\lfloor |n| \rfloor} g(\mathbf{x}, i) \qquad P(\mathbf{x}, n) = \prod_{i=1}^{\lfloor |n| \rfloor} g(\mathbf{x}, i).$$

Proof. We show the proof only for S, since the proof for P is very similar. Begin by setting

$$\tilde{S}(\mathbf{x}, \mathbf{y}, i) = (\mathbf{x}, \mathbf{y} + g(\mathbf{x}, i), i + 1).$$

Then $\tilde{S}$ is total and locally Lipschitz, and $(\mathbf{x}, S(\mathbf{x}, n), n) = \bar{\mathbf{I}}(\tilde{S})(\mathbf{x}, 0, 1, n)$. Thus S may be obtained by composition, projections and aggregation. □

Very pathological functions of analysis are real recursive too. Take, for instance, the everywhere continuous and nowhere differentiable Weierstraß function w, given by

$$w(x) = \sum_{n=0}^{\infty} a^n cos(b^n \pi x), \qquad 0 < a < 1, \quad ab > 1 + \frac{1}{3}\pi.$$

As a corollary of the previous proposition, taking the sums of the continuously differentiable $a^n cos(b^n \pi x)$ to the limit, we get:

Corollary 10.29 (Loff et al. [2007]). *The Weierstraß function is real recursive for any real recursive numbers a and b.*

There are a number of operators which are real recursive, but we may generally say that they come in two flavours. Some of these operators make some calculations with some function, and other operators could be called search operators, because their expressive power arises from searching for some value with certain properties. Differential recursion and iteration are examples of the former. While not entirely obvious, infinite limits and the η operators are good examples of search operators; solving an infinite limit consists in finding the value which is approximated by a function as one argument grows.

Definition 10.30. *Let $f : \mathbb{R}^{m+1} \to \mathbb{R}$ be in $\mathscr{F}$. The* **Sup** *and* **Inf** *operators are given, component-wise, by*

$$\mathbf{Sup}(f)(\mathbf{x}) = \sup_{y \in \mathbb{R}} f(\mathbf{x}, y) \text{ and } \mathbf{Inf}(f)(\mathbf{x}) = \inf_{y \in \mathbb{R}} f(\mathbf{x}, y).$$

In a similar way to the infinite limits, $\mathbf{Sup}(f)(\mathbf{x})$ is undefined if $f(\mathbf{x}, y)$ is undefined for any $y \in \mathbb{R}$; **Sup** and **Inf** are typical examples of search operators. In order to show that $\mathrm{REC}(\mathbb{R})$ is effectively closed for **Sup** and **Inf**, we create a periodic function, and take the supremum or infimum limit of that function.

Theorem 10.31 (Loff [2007a]). $\mathrm{REC}(\mathbb{R})$ *is effectively closed for* **Sup** *and* **Inf**. *Furthermore, if $f \in \mathrm{H}_i$, then $\mathbf{Sup}(f), \mathbf{Inf}(f) \in \mathrm{H}_{i+2}$*

Proof. Consider the function $\tilde{f}$, given by $\tilde{f}(\mathbf{x}, z, w) = f(\mathbf{x}, z\sin(w))$. Because $\sin(y)$ surjectively maps $[w, +\infty)$ into $[-1, 1]$, for any $w \in \mathbb{R}$, we find

$$\limsup_{w\to+\infty} \tilde{f}(\mathbf{x}, z, w) = \sup_{y\in[-z,z]} f(\mathbf{x}, y).$$

Then we set $\mathbf{Sup}(f)(\mathbf{x}) = \lim_{z\to+\infty} \limsup_{w\to\infty} \tilde{f}(\mathbf{x}, z, w)$. We proceed in the same way for $\mathbf{Inf}$. □

Another search operator is minimalisation over the reals, denoted with a (boldface) $\boldsymbol{\mu}$.

Definition 10.32. *Let $f : \mathbb{R}^{m+1} \to \mathbb{R}$ be in $\mathscr{F}$. The $\boldsymbol{\mu}$ operator is given by*

$$\boldsymbol{\mu}(f)(\mathbf{x}) = \inf\{y \in [0, +\infty) : f(\mathbf{x}, y) = 0\}.$$

Theorem 10.33. REC($\mathbb{R}$) *is effectively closed for minimalisation. Furthermore, if $f \in \mathrm{H}_i$, then $\boldsymbol{\mu}(f) \in \mathrm{H}_{\max(3,i+2)}$.*

Proof. Recall that the sigmoidal function, σ, surjectively maps $[0, +\infty)$ to $[\frac{1}{2}, 1)$, preserving the order (cf. Fig. 10.3). Its inverse $\sigma^{-1}(y) = \log(y) - \log(1 - y)$ is undefined for $y = 1$. Take

$$\tilde{f}(\mathbf{x}, y) = \chi_{\neq}(f(\mathbf{x}, |y|), 0) + \chi_{=}(f(\mathbf{x}, |y|), 0)\sigma(|y|) = \begin{cases} 1 & \text{if } f(\mathbf{x}, |y|) \neq 0, \\ \sigma(|y|) & \text{if } f(\mathbf{x}, |y|) = 0. \end{cases}$$

and set $\boldsymbol{\mu}(f)(\mathbf{x}) = \sigma^{-1}(\mathbf{Inf}(\tilde{f})(\mathbf{x}))$. □

10.4 Real versus classical recursive functions

We will study the relationship between real recursive functions and classical recursive functionals. These are a class of partial, multiple argument functions from $\mathbb{R}^k \times \mathbb{N}^m$ to $\mathbb{N}^n$. We will denote such a functional by using a semicolon to separate the real-valued arguments, which we will write on the left, from the natural-valued arguments, shown in the right. For instance $F(x; n)$.

Notation 10.34. We use w, x, y, z to denote variables ranging over $\mathbb{R}$, and a, b, c, i, j to denote variables ranging over $\mathbb{N}$. The corresponding vector forms $\mathbf{w}, \mathbf{x}, \ldots$ and $\mathbf{a}, \mathbf{b}, \ldots$ will denote vector-valued variables over tuples of $\mathbb{R}$ and $\mathbb{N}$. □

Take the following basic functionals as examples.

1. The zero functionals, where each $\mathcal{Z}^k$ is such that $\mathcal{Z}^k(x_1, \ldots, x_k; a) = 0$;
2. The successor functionals, where each $\mathcal{S}^k$, given by $\mathcal{S}^k(x_1, \ldots, x_k; a) = a + 1$;

3. The projection functionals, where each $\mathcal{U}_j^{k,m}$ obeys

$$\mathcal{U}_j^{k,m}(x_1, \dots, x_k; a_1, \dots, a_m) = a_j;$$

4. The oracle functionals, $\mathcal{O}_i^k$, such that[2]

$$\mathcal{O}_i^k(x_1, \dots, x_k; b) = x_i(b) \text{ (the } b\text{-th digit of the binary expansion of } x_i).$$

We write $\mathcal{C}$, $\mathcal{R}$ and μ to stand for the composition, primitive recursion and minimalisation operators. Given $F : \mathbb{R}^k \times \mathbb{N}^{m'} \to \mathbb{N}^n$, $G : \mathbb{R}^k \times \mathbb{N}^m \to \mathbb{N}^{m'}$, the functional $\mathcal{C}(F, G) : \mathbb{R}^k \times \mathbb{N}^m \to \mathbb{N}^n$ is given by

$$\mathcal{C}(F, G)(\mathbf{x}; \mathbf{a}) = F(\mathbf{x}; G(\mathbf{x}; \mathbf{a})).$$

Given two functionals $F : \mathbb{R}^k \times \mathbb{N}^m \to \mathbb{N}$ and $G : \mathbb{R}^k \times \mathbb{N}^{m+2} \to \mathbb{N}$, $\mathcal{R}(F, G) : \mathbb{R}^k \times \mathbb{N}^{m+1} \to \mathbb{N}$ is given by

$$\mathcal{R}(F, G)(\mathbf{x}; \mathbf{a}, 0) = F(\mathbf{x}; \mathbf{a}),$$

$$\mathcal{R}(F, G)(\mathbf{x}; \mathbf{a}, b + 1) = G(\mathbf{x}; b, \mathcal{R}(F, G)(\mathbf{x}; \mathbf{a}, b), \mathbf{a}).$$

The minimalisation operator μ takes a functional $F : \mathbb{R}^k \times \mathbb{N}^{m+1} \to \mathbb{N}$ and gives $\mu(F) : \mathbb{R}^k \times \mathbb{N}^m \to \mathbb{N}$ such that

$$\mu(F)(\mathbf{x}; \mathbf{a}) = \min\{b \in \mathbb{N} : F(\mathbf{x}; \mathbf{a}, b) = 0\}.$$

Finally, $\mathcal{V}$ is the aggregation operator: If $F : \mathbb{R}^k \times \mathbb{N}^m \to \mathbb{N}^n$, $F : \mathbb{R}^k \times \mathbb{N}^m \to \mathbb{N}^k$ are two functionals, then $\mathcal{V}(F, G) : \mathbb{R}^k \times \mathbb{N}^m \to \mathbb{N}^{n+k}$ comes from

$$\mathcal{V}(F, G)(\mathbf{x}; \mathbf{a}) = (F(\mathbf{x}; \mathbf{a}), G(\mathbf{x}; \mathbf{a})).$$

Now take care in the following definition. We begin by defining two classes of *functions*, and only then the relevant class of functionals.

Definition 10.35. *The class of* ***primitive recursive functions****,* PRIM*, is given by the function algebra*

$$\text{PRIM} = [\mathcal{Z}^0, \mathcal{S}^0, \mathcal{U}_j^{0,m}; \mathcal{C}, \mathcal{R}, \mathcal{V}].$$

The class of ***partial recursive functions****,* PREC*, is given by the function algebra*

$$\text{PREC} = [\mathcal{Z}^0, \mathcal{S}^0, \mathcal{U}_j^{0,m}; \mathcal{C}, \mathcal{R}, \mathcal{V}, \mu].$$

The class of ***partial recursive functionals****,* PRECF*, is given by the function algebra (k is* ***not*** *fixed):*

$$\text{PRECF} = [\mathcal{Z}^k, \mathcal{S}^k, \mathcal{U}_j^{k,m}, \mathcal{O}_i^{k,m}; \mathcal{C}, \mathcal{R}, \mathcal{V}, \mu].$$

[2] The definition is ambiguous because a dyadic rational number x has two different binary expansions. In this case, $x(b)$ is the b-th digit in the binary expansion ending in an infinite string of 0s.

Note 10.36. We have described functionals as functions from $\mathbb{R}^k \times \mathbb{N}^m$ to $\mathbb{N}^n$, but in the literature functionals are usually defined as functions from $(\mathbb{N} \to \mathbb{N})^k \times \mathbb{N}^m$ to $\mathbb{N}$. It is important to understand that our approach is only superficially different. We may use any simple bijection from $\mathbb{N} \to \mathbb{N}$ to $\mathbb{R}$, and from $\mathbb{N}$ to $\mathbb{N}^n$, to obtain this result. □

The reason we call PRIM and PREC classes of functions, is because the following may be trivially obtained by induction:

Proposition 10.37. *Every functional in* PRIM *and* PREC *has zero real-valued arguments, i.e., has a signature* $\mathbb{R}^0 \times \mathbb{N}^m \to \mathbb{N}^n$.

We will then omit the $\mathbb{R}^0$ part from the signature of functions in PRIM and PREC. In fact, we could easily take the algebras for PRIM and PREC given, resp., in pages 76 and 79, and show that a function(al) $F : \mathbb{R}^0 \times \mathbb{N}^m \to \mathbb{N}^n$ is in the algebras for primitive recursive (or partial recursive) functions of Definition 10.35 if and only if there are n functions $f_1, \dots, f_n$ in the algebras of Example 9.4 (resp. 9.18) such that $F(;\mathbf{a}) = (f_1(\mathbf{a}), \dots, f_n(\mathbf{a}))$.

To get a clearer picture of what a partial recursive functional is, we give a computational characterisation.

Theorem 10.38. *A function* $f : \mathbb{R}^k \times \mathbb{N}^m \to \mathbb{N}^n$ *is in* PRECF *if and only if there is a Turing machine with* $k + m + n$ *tapes with the following behaviour. If we take the binary expansion of* $x_1, \dots, x_k$*, and write it in the first* k *tapes (this expansion might be infinite), write the numbers* $a_1, \dots, a_m$ *in each of the following* m *tapes, and begin the computation; then, if* $f(\mathbf{x};\mathbf{a})$ *is defined, the Turing machine will halt after a finite number of steps, and print* $(f(\mathbf{x};\mathbf{a}))_1, \dots, (f(\mathbf{x};\mathbf{a}))_n$ *in the last* n *tapes; if* $f(\mathbf{x};\mathbf{a})$ *is undefined, then the machine will not halt.*

The rest of this section will be dedicated to:

(I) Showing that every primitive recursive function is *real recursive*, in some sense;
(II) Proving that every partial recursive function is *real recursive*;
(III) Concluding that every partial recursive functional is *real recursive*;

To prove (I), we see that every primitive recursive function has a real extension which is real recursive.

Proposition 10.39. *If* $F : \mathbb{N}^m \to \mathbb{N}^n$ *is in* PRIM*, then there is a real recursive total locally Lipschitz function* $f : \mathbb{R}^m \to \mathbb{R}^n$ *in* H_1 *such that*

$$F(;\mathbf{a}) = f(\mathbf{a}) \text{ for all } \mathbf{a} \in \mathbb{N}^m.$$

Proof. Our proof is by structural induction on the function algebra for primitive recursive functions. This is clearly true for $\mathcal{Z}$ and $\mathcal{S}$, by taking the real recursive zero and add-one functions, and the inductive step for composition and aggregation is trivial, using the corresponding real recursive operators. Now suppose that $H : \mathbb{N}^{m+1} \to \mathbb{N}$ in PRIM is given by $H = \mathcal{R}(F, G)$, for some $F : \mathbb{N}^m \to \mathbb{N}$, $G : \mathbb{N}^{m+2} \to \mathbb{N}$, i.e.,

$$H(;\mathbf{a}, 0) = F(;\mathbf{a}) \qquad H(;\mathbf{a}, n+1) = G(;n, H(;\mathbf{a}, n), \mathbf{a}).$$

Then by the induction hypothesis, let f and g be two total locally Lipschitz real recursive functions in H_1 such that $F(;\mathbf{a}) = f(\mathbf{a})$ and $G(;n, b, \mathbf{a}) = g(n, b, \mathbf{a})$ for all $\mathbf{a} \in \mathbb{N}^m$ and $n, b \in \mathbb{N}$. Form

$$\tilde{h}(n, \mathbf{b}, \mathbf{a}) = (n+1, g(n, \mathbf{b}, \mathbf{a}), \mathbf{a})$$

by aggregation. Then let h be given by $(n, h(\mathbf{a}, n), \mathbf{a}) = \tilde{\mathbf{I}}(\tilde{h})(0, f(\mathbf{a}), \mathbf{a}, n)$, from which we find $h \in \mathrm{H}_1$. This function h will be a real recursive extension of H, and will also be total and locally Lipschitz by Theorem 10.26. □

Now recall the normal form theorem of Kleene.

Theorem 10.40 (Normal form theorem). *For every natural $m, n > 0$ are two primitive recursive functions $U : \mathbb{N} \to \mathbb{N}^n$ and $T : \times\mathbb{N}^{m+2} \to \mathbb{N}$ with the following property. Take any partial recursive function $F : \mathbb{N}^m \to \mathbb{N}^n$, and there will be a number e, called a **code** of F, such that*

1. *$F(\mathbf{a})$ is defined if and only if $(\exists b \in \mathbb{N})\ T(e, \mathbf{a}, b) = 0$, and*
2. *$F(\mathbf{a}) = U(\mu(T)(e, \mathbf{a}))$.*

By Proposition 10.39, there will be a real recursive extension of T and U, but we may furthermore ensure that:

Proposition 10.41. *If $F : \mathbb{N}^{m+1} \to \mathbb{N}$ and $f : \mathbb{R}^{m+1} \to \mathbb{R}$ in H_1 are such that*

$$F(;\mathbf{a}, b) = f(\mathbf{a}, b) \text{ for all } \mathbf{a} \in \mathbb{N}^m, b \in \mathbb{N}.$$

then there is another real recursive function $\tilde{f} : \mathbb{R}^{m+1} \to \mathbb{R}$ in H_1 such that

$$\mu(F)(;\mathbf{a}) = \boldsymbol{\mu}(\tilde{f})(\mathbf{a}) \text{ for all } \mathbf{a} \in \mathbb{N}^m.$$

Proof. Just set $\tilde{f}(\mathbf{a}, y) = f(\mathbf{a}, \lfloor y \rfloor) r(y) + f(\mathbf{a}, \lfloor y+1 \rfloor)(1 - r(y))$. □

The function $\tilde{f}$ is just a linear interpolation of f on the last argument. If $F(b) = 1$ when b is not a prime, and $F(b) = 0$ if b is a prime, then the $\tilde{f}$ we would obtain is shown in Fig. 10.6.

We now get the following corollary.

Theorem 10.42 (Mycka and Costa [2004] and see also Graça et al. [2005]). *If $F : \mathbb{N}^m \to \mathbb{N}^n$ is in PREC, then there is a real recursive function $f : \mathbb{R}^m \to \mathbb{R}^n$ in H_3 such that*

$$F(;\mathbf{a}) \simeq f(\mathbf{a}) \text{ for all } \mathbf{a} \in \mathbb{N}^m.$$

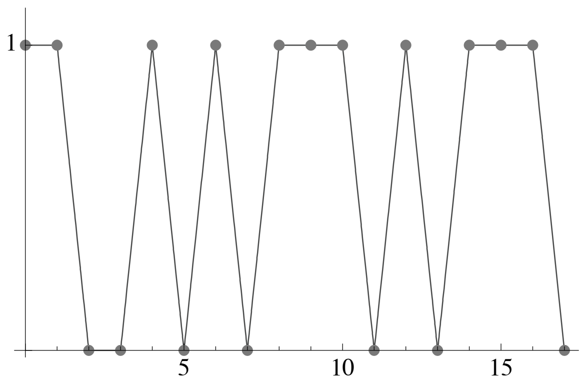


Fig. 10.6. A plot of $\tilde{f}(y)$, where $F(b)$ is given by the dots.

Proof. Let U, T be the primitive recursive functions given by the normal form theorem, u, t be their real recursive extensions given by Proposition 10.39, and let $\tilde{t}$ be given from t by the previous proposition. For any partial recursive function F, let e be one of its codes. Then set $f(\mathbf{x}) = u(\boldsymbol{\mu}(\tilde{t})(e, \mathbf{x}))$, and conclude that f is a real extension of F. The function f will be in H_3, because $u, \tilde{t}$ are in H_1, and by Proposition 10.33. □

Thus concluding (II); in order to prove (III), we use the fact that any converging computation will always make use of only a finite part of the oracle.

Definition 10.43. *If* $\mathbf{x} \in \mathbb{R}^k$, *then let* $\mathbf{x}\restriction_n^{\mathbb{N}}$ *denote, for some* $n \in \mathbb{N}$, *the vector in* $\mathbb{N}^k$ *given by*

$$\mathbf{x}\restriction_n^{\mathbb{N}} = (x_1\restriction_n \times 2^n, \dots, x_k\restriction_n \times 2^n).$$

Proposition 10.44. *The function given by* $f(\mathbf{x}, n) = \mathbf{x}\restriction_n^{\mathbb{N}}$ *is real recursive.*

Proof. For $x \in \mathbb{R}$, $x\restriction_n \times 2^n = \lfloor x 2^n \rfloor$. The rest comes from aggregation, *etc.* □

We can now use Theorem II.3.11 from Odifreddi [1989], which for our purposes can be formulated as follows.

Proposition 10.45. *A functional* $F : \mathbb{R}^k \times \mathbb{N}^m \to \mathbb{N}^n$ *is in* PRECF *if and only if there is a partial recursive function* $\tilde{F} : \mathbb{N}^{k+m} \to \mathbb{N}^n$ *with the property that* $F(\mathbf{x}; \mathbf{a}) \simeq \mathbf{b}$ *if and only if there is an* $n \in \mathbb{N}$ *such that* $\tilde{F}(; \mathbf{x}\restriction_n^{\mathbb{N}}, \mathbf{a}) \simeq \mathbf{b}$.

The following corollary can be understood as saying that for any given input the behaviour of a recursive functional only depends on a finite part of the oracle.

Corollary 10.46. *If a functional* $F : \mathbb{R}^k \times \mathbb{N}^m \to \mathbb{N}^n$ *is in* PRECF *then there is a partial recursive function* $\tilde{F} : \mathbb{N}^{k+m} \to \mathbb{N}^n$ *with the property that* $F(\mathbf{x}; \mathbf{a}) \simeq \mathbf{b}$ *if and only if there is an* $m \in \mathbb{N}$ *such that* $\tilde{F}(; \mathbf{x}\restriction_n^{\mathbb{N}}, \mathbf{a}) \simeq \mathbf{b}$ *for all natural* $n \geqslant m$.

And so we conclude this chapter, with the following.

Theorem 10.47. *If* $F : \mathbb{R}^k \times \mathbb{N}^m \to \mathbb{N}^n$ *is in* PRECF*, then there is a real recursive function* $f : \mathbb{R}^{k+m} \to \mathbb{R}^n$ *in* H_4 *such that*

$$F(\mathbf{x}; \mathbf{a}) \simeq f(\mathbf{x}, \mathbf{a}) \text{ for all } \mathbf{x} \in \mathbb{R}^k, \mathbf{a} \in \mathbb{N}^m.$$

Proof. Let $\tilde{F}$ be given from the previous corollary, and let $\tilde{f}$ be a real recursive function in H_3 extending $\tilde{F}$. Then take

$$f(\mathbf{x}, \mathbf{a}) = \lim_{n \to +\infty} \tilde{f}(\mathbf{x} \restriction^{\mathbb{N}}_{\lfloor n \rfloor}, \mathbf{a})$$

and $f \in \mathrm{H}_4$ will extend F. □

We have shown that every partial recursive function or functional has a real recursive extension. However, the class of real recursive functions stretches much further. We will see in Chapter 12 that any predicate in the analytical hierarchy is real recursive, and as a corollary many non-computable functions, such as the Busy Beaver function, have real recursive extensions.

11

Universality

This chapter is devoted to solving the problem of universality. We will begin in Section 11.1 with a series of considerations on the Euler method to approximate solutions of our simple differential recursion scheme (9.6). Our first conclusion will be that we may replace differential recursion with the restricted iteration operator of Definition 10.22, by adding a few basic functions to the algebra H. With this, we will form a new algebra I that also gives REC($\mathbb{R}$). In Section 11.2 we show that we may *totalise* any real recursive function, and this will solve the problem of universality, in the negative sense: we will conclude that there is no universal real recursive function.

11.1 The Euler method and differential recursion

Let $f : \mathbb{R}^{n+1} \to \mathbb{R}^n$ be a total locally Lipschitz function; consider again the Cauchy problem

$$g(\mathbf{x}, 0) = \mathbf{x} \quad \partial_t g(\mathbf{x}, t) = f(t, g(\mathbf{x}, t)), \tag{11.1}$$

We study some interesting properties of the Euler method to approximate solutions to (11.1).

Definition 11.1. *The* ***Euler broken line*** *for the Cauchy problem (11.1) is a total function $\hat{g} : \mathbb{R}^{n+2} \to \mathbb{R}^n$. For any fixed $z > 0$, let $\delta = \frac{1}{z}$, $t_0 = 0$, $t_{i+1} = t_i + \delta$. Abbreviate $\hat{g}(\mathbf{x}, t, z) \equiv \hat{g}_z(\mathbf{x}, t)$. Then $\hat{g}$ is given by:*

$$\hat{g}_z(\mathbf{x}, t_0) = \mathbf{x};$$

if $t_i \leqslant t < t_{i+1}$, then

$$\hat{g}_z(\mathbf{x}, t) = \hat{g}_z(\mathbf{x}, t_i) + (t - t_i) f(t_i, \hat{g}_z(\mathbf{x}, t_i));$$

and if $-t_{i+1} < t \leqslant -t_i$, then

$$\hat{g}_z(\mathbf{x}, t) = \hat{g}_z(\mathbf{x}, -t_i) - (t + t_i) f(-t_i, \hat{g}_z(\mathbf{x}, -t_i)).$$

The ***bounded Euler broken line*** *for the Cauchy problem (11.1) is a total function $\tilde{g} : \mathbb{R}^{n+3} \to \mathbb{R}^n$. We set δ, t_i as before, and abbreviate $\tilde{g}(\mathbf{x}, t, M, z) \equiv \tilde{g}_{M,z}(\mathbf{x}, t)$. Let j, k denote the smallest natural numbers such that $\|\hat{g}_z(\mathbf{x}, -t_j)\| \geqslant M$ and $\|\hat{g}_z(\mathbf{x}, t_k)\| \geqslant M$. If there is no such k (or j), then t_{k-1} (resp. $-t_{j-1}$) will denote $+\infty$ (resp. $-\infty$). Then $\tilde{g}$ is given by*

$$\tilde{g}_{M,z}(\mathbf{x},t) = \begin{cases} \hat{g}_z(\mathbf{x},-t_{j-1}) & \textit{if } t \leqslant -t_{j-1} \\ \hat{g}_z(\mathbf{x},t) & \textit{if } -t_{j-1} < t < t_{k-1} \\ \hat{g}_z(\mathbf{x},t_{k-1}) & \textit{if } t_{k-1} \leqslant t \end{cases}$$

The bounded Euler broken line is a piece-wise linear approximation of the solution g of (11.1), using Euler's method, which is bounded in the norm by a value M, and where the number of approximation steps for a segment of length 1 is given by z. Fig. 11.1 illustrates how the bounded Euler broken line looks for different values of M and z.

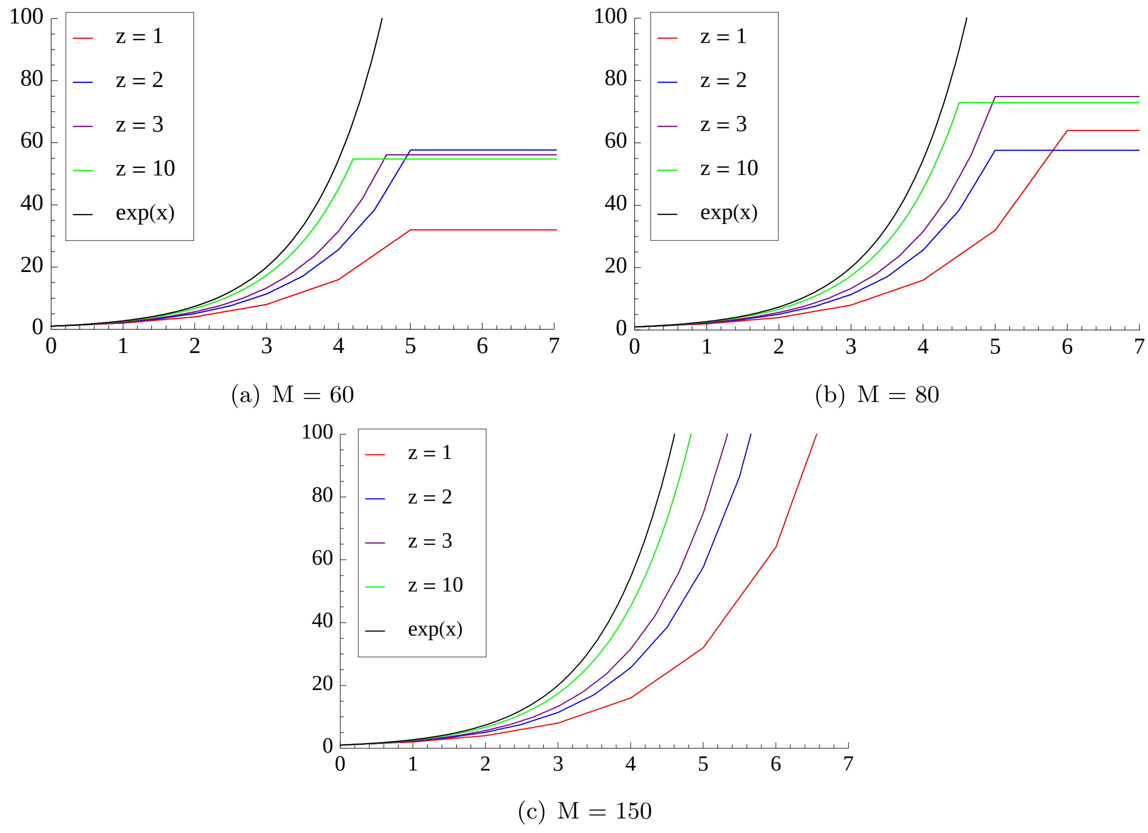


Fig. 11.1. The bounded Euler broken line, for $x = 1$ and $f(t, y) = y$, with various values of M and z.

Theorem 11.2. *Let g be a solution of (11.1), and let $\tilde{g}$ be its Euler broken line. Fix an arbitrary $\mathbf{x} \in \mathbb{R}^n$, and compact interval $J \subset Dom(g)$. Then, for every large enough $M \in \mathbb{R}$, the function $\tilde{g}_{M,z}(\mathbf{x},\cdot)$ converges to $g(\mathbf{x},\cdot)$ uniformly in J as $z \to +\infty$. In symbols,*

$$(\forall\varepsilon > 0)\exists\tilde{M}(\forall M > \tilde{M})\exists\tilde{z}(\forall z > \tilde{z})(\forall t \in J)\ \|\tilde{g}_{M,z}(\mathbf{x},t) - g(\mathbf{x},t)\| < \varepsilon.$$

Proof. Take any $\varepsilon > 0$. By the compactness of J and continuity of g (Theorem 9.28), set

$$M \geqslant \max_{t\in J} \|g(\mathbf{x},t)\| + |t| + \varepsilon. \tag{11.2}$$

Let B denote the closed (compact) cylinder $B = J \times \overline{B}(0, M) \subset \mathbb{R}^{n+1}$. Because f and g are locally Lipschitz (Theorem 9.28), then choose two constants K_f and K_g such that

$$\|g(\mathbf{x}, \tilde{t}) - g(\mathbf{x}, t)\| \leqslant K_g|\tilde{t} - t| \text{ for all } t, \tilde{t} \in J; \tag{11.3}$$

$$\|f(\tilde{t}, \tilde{\mathbf{y}}) - f(t, \mathbf{y})\| \leqslant K_f|\tilde{t} - t| + K_f\|\tilde{\mathbf{y}} - \mathbf{y}\| \text{ for all } (t, \mathbf{y}), (\tilde{t}, \tilde{\mathbf{y}}) \in B. \tag{11.4}$$

Choose $K \in \mathbb{R}$ to be greater than both K_f and $K_g + 1$, and choose z and δ, so that letting d_J denote the diameter of J,

$$z > \frac{1}{\varepsilon} K(\mathrm{e}^{K d_J} - 1), \qquad \delta = \frac{1}{z}, \qquad t_0 = 0, \quad t_{i+1} = t_i + \delta$$

We will show, by induction on i, that setting $\Delta_t = \|\tilde{g}_{M,z}(\mathbf{x}, t) - g(\mathbf{x}, t)\|$, then

$$\Delta_t < \frac{1}{2} K\delta(\mathrm{e}^{Kt} - 1)$$

for any positive $t \in J$. Because of our choice of z, this ensures that $\Delta_t < \varepsilon$, which is what we intend to prove. The hypothesis is true for $t_i = t_0 = 0$, because $\Delta_0 = 0$. Now suppose it is true for some t_i. Then, choosing any $t \in [t_i, t_{i+1}] \cap J$,

$$\Delta_t = \|\tilde{g}_{M,z}(\mathbf{x}, t) - g(\mathbf{x}, t)\| = \left\|\tilde{g}_{M,z}(\mathbf{x}, 0) - g(\mathbf{x}, 0) + \int_0^t f(t_i, \tilde{g}_{M,z}(\mathbf{x}, t_i)) - f(s, g(\mathbf{x}, s)) ds\right\|$$

$$\leqslant \Delta_{t_i} + \int_{t_i}^t \|f(t_i, \tilde{g}_{M,z}(\mathbf{x}, t_i)) - f(s, g(\mathbf{x}, s))\| ds$$

$$\leqslant \Delta_{t_i} + \int_{t_i}^t \|f(t_i, \tilde{g}_{M,z}(\mathbf{x}, t_i)) - f(t_i, g(\mathbf{x}, t_i))\| ds + \int_{t_i}^t \|f(t_i, g(\mathbf{x}, t_i)) - f(s, g(\mathbf{x}, s))\| ds$$

By induction hypothesis, $\Delta_{t_i} = \|\tilde{g}_{M,z}(\mathbf{x}, t_i) - g(\mathbf{x}, t_i)\| < \varepsilon$, and so $(t_i, \tilde{g}_{M,z}(\mathbf{x}, t_i)) \in B$. Then we may apply the Lipschitz properties (11.3) and (11.4) to the integrals, and find

$$\Delta_t \leqslant \Delta_{t_i} + K_f(t - t_i)\Delta_{t_i} + K_f(K_g + 1)\left|\int_{t_i}^t |s - t_i| ds\right|$$

Which implies $\Delta_t \leqslant (1 + K(t - t_i))\Delta_{t_i} + \frac{1}{2}K^2(t - t_i)^2$. Using again the induction hypothesis, and $(t - t_i) \leqslant \delta$,

$$\Delta_t \leqslant \frac{1}{2} K\delta \mathrm{e}^{Kt_i}(1 + K(t - t_i)) - \frac{1}{2} K\delta(1 + K(t - t_i)) + \frac{1}{2} K^2\delta(t - t_i).$$

Now, because $1 + x \leqslant \mathrm{e}^x$ and $K^2 > K$, we arrive at

$$\Delta_t \leqslant \frac{1}{2} K\delta(\mathrm{e}^{Kt} - 1) < \varepsilon,$$

as intended. The proof is symmetrical for a negative t. □

Corollary 11.3. *If g is a solution of (11.1), and $\tilde{g}$ is its Euler broken line, then*

$$\limsup_{M\to+\infty}\limsup_{z\to+\infty}\tilde{g}_{M,z}(\mathbf{x},t)=g(\mathbf{x},t)\ \textit{for}\ (\mathbf{x},t)\in \mathit{Dom}(g).$$

We may, in fact, obtain the following.

Proposition 11.4. *If g is a solution of (11.1), and $\tilde{g}$ is its Euler broken line, then*

$$\limsup_{M\to+\infty}\limsup_{z\to+\infty}\tilde{g}_{M,z}(\mathbf{x},t)\simeq g(\mathbf{x},t).$$

Proof. We already know that g will equal $\mathbf{Ls}(\mathbf{Ls}(\tilde{g}))$ where it is defined, and so we only need to show that $\mathbf{Ls}(\mathbf{Ls}(\tilde{g}))$ will be undefined where g is undefined. This is an easy conclusion deriving from Theorem 9.26, and from the definition of the bounded Euler broken line. For any fixed $\mathbf{x}$, and any compact interval J, we have two cases. Either that $g(\mathbf{x},\cdot)$ is bounded in J, and so $\limsup_{z\to+\infty}\tilde{g}_{M,z}(\mathbf{x},t)=g(\mathbf{x},t)$ for every $t\in J$ and every large enough M; or otherwise $\limsup_{t\to B^+}\|g(\mathbf{x},t)\|=+\infty$ for some positive B in J (the case is symmetrical for a negative A). In this case, g will be continuous and defined for every $0\leqslant t<B$. So let $t_M\in J$ be the smallest value for which $\|g(\mathbf{x},t_M)\|\geqslant M$, i.e,

$$t_M=\min\{t\in J:\|g(\mathbf{x},t_M)\|\geqslant M\}$$

(which is well defined because g is continuous). Then by definition of $\tilde{g}$

$$\limsup_{z\to+\infty}\tilde{g}_{M,z}(\mathbf{x},t)=\begin{cases}g(\mathbf{x},t) & \text{if } t<t_M,\\ g(\mathbf{x},t_M) & \text{if } t\geqslant t_M.\end{cases}$$

By the continuity of g, $\|g(\mathbf{x},t)\|$ is always bounded for any compact interval $[0,T]\subset[0,B)$, and so $t_M>T$ if $M\geqslant\max_{t\in[0,T]}\|g(\mathbf{x},t)\|$. So $t_M\to B$ as $M\to+\infty$. We may then conclude that

$$\limsup_{M\to+\infty}\limsup_{z\to+\infty}\|\tilde{g}_{M,z}(\mathbf{x},t)\|=+\infty$$

for any $t\in[B,+\infty)$, and so $[B,+\infty)$ is disjoint from $\mathit{Dom}(\mathbf{Ls}(\mathbf{Ls}(\tilde{g})))$. □

The Euler broken line can be obtained in a real recursive way.

Proposition 11.5 (Campagnolo et al. [2002], Loff et al. [2007]). *For any real recursive f in $\mathit{Dom}(\mathbf{R})$, the Euler broken line $\tilde{g}$ of $g=\mathbf{R}(f)$ is real recursive.*

Proof. Let b be given by

$$b(t,\mathbf{y},\delta,M)=\chi_<(\|\mathbf{y}+\delta\times f(t,\mathbf{y})\|,M)=\begin{cases}1 & \text{if } \|\mathbf{y}+\delta\times f(t,\mathbf{y})\|<M;\\ 0 & \text{otherwise.}\end{cases}$$

Now take the auxiliary function $\hat{g}$, given by

$$\hat{g}(t, \mathbf{y}, \delta, M) = (t + \delta \times b(t, \mathbf{y}, \delta, M), \mathbf{y} + \delta \times f(t, \mathbf{y}) \times b(t, \mathbf{y}, \delta, M), \delta, M)\,.$$

It is straightforward to see that if f is real recursive, then so is $\hat{g}$. The function $\hat{g}$ calculates the point $(t_{i+1}, \tilde{g}_{M,z}(\mathbf{x}, t_{i+1}), \delta, M)$ in the bounded Euler broken line, when given the current $t = t_i$, $\mathbf{y} = \tilde{g}_z(\mathbf{x}, t_i)$, $\delta = \frac{1}{z}$ and M. If $\delta = -\frac{1}{z}$, then $\hat{g}$ will calculate $(-t_{i+1}, \tilde{g}_z(\mathbf{x}, -t_{i+1}), \delta, M)$ instead. We may then use another function, $\bar{g}$, with its i-th component given by:

$$(\bar{g}(\mathbf{x}, t, M, z))_i = \left(\hat{g}^{\lfloor tz \rfloor}\left(0, \mathbf{x}, \frac{t}{|t|z}, M\right)\right)_{i+1}\,.$$

The function $\bar{g}$, given t, z, M, calculates $\tilde{g}(\mathbf{x}, t_i)$ for the largest $t_i \leqslant t$. Then $\tilde{g}$ may be given by a linear interpolation:

$$\tilde{g}_{M,z}(\mathbf{x}, t) = r(zt)\bar{g}(\mathbf{x}, t, M, z) + (1 - r(zt))\bar{g}\left(\mathbf{x}, t + \frac{1}{z}, M, z\right)\,. \qquad \square$$

Supported by this proposition, we define the following new operator.

Definition 11.6. *The* ***Euler operator*** $\mathbf{E} : \mathscr{F} \to \mathscr{F}$, *takes a total locally Lipschitz function* $f : \mathbb{R}^{m+1} \to \mathbb{R}^n$, *and maps* $f \mapsto \tilde{g}$, *where* $\tilde{g}$ *is the Euler broken line of Definition 11.1.*

Then Proposition 11.5 is equivalent to saying that $\mathbf{E}$ is a real recursive operator. We may, however, take the following stronger result.

Theorem 11.7. *The class of real recursive functions is also given by the function algebra*

$$[-1^n, 0^n, 1^n, \mathrm{U}_i^n; \mathbf{C}, \mathbf{E}, \mathbf{Ls}, \mathbf{V}].$$

Proof. Clearly this function algebra only gives real recursive functions, because REC($\mathbb{R}$) contains every basic function and is closed for every operator by Proposition 11.5. However, because $\mathbf{R} = \mathbf{Ls} \circ \mathbf{Ls} \circ \mathbf{E}$ (Proposition 11.4), this algebra will also give us every real recursive function. $\square$

We will, however, use the following.

Theorem 11.8 (Loff et al. [2007]). *The class of real recursive functions is given by the function algebra*

$$\mathrm{REC}(\mathbb{R}) = \mathrm{I} = [-1^n, 0^n, 1^n, \mathrm{U}_i^n, +, \times, x^y; \mathbf{C}, \bar{\mathbf{I}}, \mathbf{Ls}, \mathbf{V}].$$

Above, $+$, $\times$ and x^y denote the binary addition, product and exponentiation (the later being undefined for negative bases).

Proof. We only need to show that we may obtain $\mathbf{E}$ using only restricted iteration, $+, \times, x^y$ and the remaining operators. But looking at the proof of Proposition 11.5, we see that this is the case, but we additionally use restricted inverse $1/\cdot$, the characteristic of strict inequality $\chi_<$, the absolute value function $|\cdot|$, the euclidean norm $\|\cdot\|$, and the floor function $\lfloor\cdot\rfloor$. However, all of these may be obtained from the basic functions: $\frac{1}{x} = x^{-1}$; the expressions given in Propositions 10.8 may still be used to give Kronecker's δ and Heaviside's Θ — and thus $\chi_<$ and $|\cdot|$; the Euclidean norm comes from $\sqrt{\cdot}$ and $\times$, and $\sqrt{x} = x^{\frac{1}{2}}$; finally, iterating the successor function $s(x) = x + 1$ we get $\lfloor x \rfloor = \bar{\mathbf{I}}(s)(0, x)$. In this way, we may obtain $\mathbf{E}$ using the shown function algebra. □

This specific function algebra was named, using the capital Greek letter I (iota), because it will be fundamental in most of the following results.

11.2 Totalisation operators and universality

We will solve the problem of universality in two steps. First, we show that any real recursive function can be extended to a total function. Then, we show that if there was a universal real recursive function, this would be impossible.

Definition 11.9. *Given a function $f : \mathbb{R}^m \to \mathbb{R}^n$ in $\mathscr{F}$, its **totalisation** is a function $(\chi_f, \tau_f) : \mathbb{R}^m \to \mathbb{R}^{n+1}$, such that*

$$\chi_f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in Dom(f), \\ 0 & \text{otherwise}; \end{cases} \qquad \tau_f(\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{if } \mathbf{x} \in Dom(f), \\ 0 & \text{otherwise.} \end{cases}$$

Theorem 11.10 (Loff et al. [2007]). *If f real recursive function, then so is its totalisation.*

Proof. The proof is by structural induction on the algebra I. Because the basic functions are all total, the result is trivial for the basic case. If f is $\mathbf{C}(g, h)$, then the totalisation of g and h is real recursive, by the induction hypothesis, and then $\chi_f(\mathbf{x}) = \chi_g(\tau_h(\mathbf{x})) \times \chi_h(\mathbf{x})$, $\tau_f(\mathbf{x}) = \tau_g(\tau_h(\mathbf{x})) \times \chi_h(\mathbf{x})$ gives the result for composition. The result is also trivial for restricted iteration, because it maps total functions to total functions. Now, if f is $\mathbf{Ls}(g)$, then the totalisation of g must be real recursive, by our induction hypothesis. Then, for any $\mathbf{x}$,

$$f(\mathbf{x}) = \limsup_{y\to\infty} g(\mathbf{x}, y),$$

and we must have one of two cases:

1. $g(\mathbf{x}, y)$ is defined for every large enough y; in symbols,

$$\exists \tilde{y} (\forall y > \tilde{y})\ (\mathbf{x}, y) \in Dom(g);$$

2. $g(\mathbf{x}, y)$ is undefined for arbitrarily large y; in symbols,

$$\forall \tilde{y} (\exists y > \tilde{y})\ (\mathbf{x}, y) \notin \mathit{Dom}(g);$$

In the first case, then clearly $f(\mathbf{x}) = \limsup_{y \to \infty} \tau_g(\mathbf{x}, y)$, but in the second case, this might not be so. However, we have that

$$\limsup_{y \to \infty} \left[\tau_g(\mathbf{x}, y) + (1 - \chi_g(\mathbf{x}, y))y\right]$$

will be defined if and only if $f(\mathbf{x})$ is defined,[1] and will be equal to $f(\mathbf{x})$ if it is indeed defined. So we may set

$$\chi_f(\mathbf{x}) = \eta^s(\tau_g + (1 - \chi_g) \times U^{n+1}_{n+1})(\mathbf{x})$$

$$\tau_f(\mathbf{x}) = \chi_f(\mathbf{x}) \times \limsup_{y \to \infty} \left(\chi_f(\mathbf{x}) \times \tau_g(\mathbf{x}, y)\right).$$

The induction step for aggregation is very simple: if f is $\mathbf{V}(g, h)$, we may use the induction hypothesis and set $\chi_f(\mathbf{x}) = \chi_g(\mathbf{x}) \times \chi_h(\mathbf{x})$ and $\tau_f = (\tau_g, \tau_h)$. □

Now we may solve the problem of universality. We will simplify our result by making the following assumption.

Assumption on Gödelisation. There is an effective enumeration d_e of all the descriptions in D_{I}. □

This assumption can be made into a theorem by any standard method of encoding. Whenever d_e is a good description (Definition 9.5), ϕ_e will denote the described function, and e is said to be a **code** of ϕ_e. We may then specialise Definition 9.16 into the following.

Definition 11.11. *A real recursive function $\Psi : \mathbb{R}^{m+1} \to \mathbb{R}^n$ is called **universal** if for every $e \in \mathbb{N}$, $\mathbf{x} \in \mathbb{R}^m$, we have*

$$\Psi(e, \mathbf{x}) \simeq \phi_e(\mathbf{x})$$

whenever d_e is a good description of an m-ary function with n components.

Theorem 11.12 (Loff et al. [2007], see also Mycka and Costa [2006c]). *There is no universal real recursive function.*

Proof. A diagonal argument will be give us *reductio ad absurdum.* For clarity we present only the case when $m = n = 1$, but the argument may easily be extended. Suppose that there was a universal real recursive function $\Psi : \mathbb{R}^2 \to \mathbb{R}$. Then by the previous theorem we could find its real recursive totalisation χ_Ψ and τ_Ψ, and the function given by

$$g(x) = \log(1 - \chi_\Psi(x, x)) = \log(1 - \chi_{\phi_x}(x)) = \begin{cases} 1 & \text{if } x \notin \mathit{Dom}(\phi_x), \\ \perp & \text{otherwise;} \end{cases}$$

[1] Recall that we treat $+\infty, -\infty$ and $\perp$ in the same way. If this is somehow confusing, remember that we may use the sigmoidal function, as we did in order to prove Propositions 10.16 and 10.21 .

would be a real recursive function of arity 1. So let e be a code of g. We have that $e \in Dom(g)$ if and only if $e \notin Dom(\phi_e)$, which is the contradiction we sought. $\square$

Understanding REC($\mathbb{R}$)

We have shown in Section 10.3 that REC($\mathbb{R}$) is effectively closed for the **Sup** and **Inf** search operators. We may strengthen this result in the following way.

Theorem 12.1 (Loff [2007a]). *The class of real recursive functions is given by the function algebra*

$$\mathrm{REC}(\mathbb{R}) = [-1^n, 0^n, 1^n, \mathrm{U}_i^n, +, \times, x^y; \mathbf{C}, \bar{\mathbf{I}}, \mathbf{Sup}, \mathbf{V}].$$

Proof. Let $\mathcal{A}$ denote the given algebra. All we have done was replace the infinite supremum limit by the supremum in the algebra I. We know that I is closed for the supremum operator, and so all we need to show is that we may obtain the infinite supremum limit in the algebra $\mathcal{A}$. This algebra is also closed for the infimum, because $\mathbf{Inf}(f) = -\mathbf{Sup}(-f)$. But by definition,

$$\limsup_{y\to\infty} f(\mathbf{x}, y) = \inf_{y\in\mathbb{R}} \sup_{z>y} f(\mathbf{x}, z) = \inf_{y\in\mathbb{R}} \sup_{z\in\mathbb{R}} f(\mathbf{x}, z^2 + y),$$

and so $\mathcal{A}$ is also closed for **Ls**. □

We therefore reduce our original function algebra to a fairly trimmed down inductive definition. However, in this chapter we will show that the expressive power of this function algebra is much greater than what was anticipated. In Section 12.1 we will introduce the analytical hierarchy of predicates, and show that the graph of any real recursive function is in this hierarchy. In 12.2 we will show the converse, that any function with a graph in the analytical hierarchy must be real recursive. The expressive power of the analytical hierarchy is evidently great, and so this result explains why it seems to be so hard to find a function which is not real recursive.

12.1 The analytical hierarchy

The analytical hierarchy is a hierarchy of predicates of second-order arithmetic, and is studied in a variety of contexts. It was originally devised by Lusin (1925) for the then-incipient field of descriptive set theory and discovered independently by Kleene (1955) in the study of recursion on higher types. The name ‘analytical’ is used because second-order arithmetic allows for the formalisation of elementary analysis.

We present the analytical hierarchy of predicates, and relate it with the η-hierarchy.

Definition 12.2. *A predicate P over real and natural numbers is called* ***recursive*** *if there is a partial recursive functional F such that*

$$F(\mathbf{x};\mathbf{a}) = \begin{cases} 1 & \text{if } P(\mathbf{x},\mathbf{a}) \text{ holds,} \\ 0 & \text{otherwise.} \end{cases}$$

A predicate Q over real and natural numbers is called ***arithmetical*** *if it is given using natural number quantifiers over a recursive predicate, i.e., if for some recursive predicate P,*

$$Q(\mathbf{x},\mathbf{a}) \iff (\forall b_1)(\exists b_2)\dots(\forall b_{n-1})(\exists b_n)P(\mathbf{x},\mathbf{a},\mathbf{b}).$$

Definition 12.3. *The* ***analytical hierarchy of predicates*** *consists of three $\mathbb{N}$-indexed families of predicates over real and natural numbers:*

1. *Σ^1_0 is the class of arithmetical predicates, and $\Pi^1_0 = \Sigma^1_0$.*
2. *Σ^1_{n+1} is the class of predicates given by $\exists y\ P(\mathbf{x},y,\mathbf{a})$, with P in Π^1_n.*
3. *Π^1_{n+1} is the class of predicates given by $\forall y\ P(\mathbf{x},y,\mathbf{a})$, with P in Σ^1_n.*
4. *$\Delta^1_n = \Sigma^1_n \cap \Pi^1_n$.*

We will call **analytical** to the predicates in the analytical hierarchy. We write Δ^1_ω to stand for $\cup_{n\in\mathbb{N}}\Delta^1_n$, which is exactly the set of all analytical predicates. We will make abundant use of the following result [cf. Odifreddi, 1989, p.377].

Proposition 12.4. *(a) Σ^1_{n+1} is closed for existential quantification over $\mathbb{R}$.*
(b) Π^1_{n+1} is closed for universal quantification over $\mathbb{R}$.
(c) Π^1_{n+1} and Σ^1_{n+1} are closed for existential and universal quantification over $\mathbb{N}$.
(d) We may exchange quantifiers over $\mathbb{N}$ with quantifiers over $\mathbb{R}$, i.e.,
(di) If $P \in \Sigma^1_n$ then some $\tilde{P}$ also in Σ^1_n is such that $\forall a\ P \iff \forall x\ \tilde{P}$.
(dii) If $P \in \Pi^1_n$ then some $\tilde{P}$ also in Π^1_n is such that $\exists a\ P \iff \exists x\ \tilde{P}$.

Recall the following definition from page 89.

Definition 12.5. *The* ***graph*** *of a function $f : \mathbb{R}^m \to \mathbb{R}^n$, denoted G_f, is the $(n+m)$-ary predicate given by*

$$G_f(\mathbf{z},\mathbf{x}) \iff \mathbf{x} \in Dom(f) \wedge \mathbf{z} = f(\mathbf{x}).$$

Definition 12.6. *We say that a function $f : \mathbb{R}^m \to \mathbb{R}^n$ is in Σ^1_k if its graph is in Σ^1_k. Similarly for Π^1_k and Δ^1_k.*

We know that quantifiers may be used to express a rich variety of mathematical ideas, and so we expect that there are many functions in the analytical hierarchy.

Proposition 12.7. *The functions $1^n, \bar{1}^n, 0^n, \mathrm{U}^n_i, +, \times, x^y, |\cdot|$ and $\lfloor\cdot\rfloor$, as well as the predicates of equality and inequality over the reals, are in Δ^1_0.*

Proof. We begin by showing that there is a recursive way to decide the predicate over the reals given by the expression 'x and y are not different up to the nth digit', written $x =_n y$. An algorithm to decide this predicate needs to solve the ambiguity of the representation of a real number by binary expansion, and we can make it work the following way: given two real numbers x, y and a natural number n, we obtain the first n digits of the two reals and verify if they are the same. If they are, then we decide that $x =_n y$. If the digits are not equal we consider the first different digit — one is 0 and the other 1 — and check if the digits after the 0 are all 1s and the digits after the 1 are all 0s.[1] If so, then we decide that $x =_n y$, and we decide that $x \neq_n y$ otherwise. The predicate of real number equality is then given by: $\forall n \;\; x =_n y$, which is in Δ^1_0. For the function $+$, we define a predicate, of expression $z =_n x + y$, that decides if $z = x + y$ for the first n digits of z, x and y. This function computes the sum of the truncations of x and y to the nth fractionary digit and checks if resulting rational number coincides with z to the nth digit using the method shown above. If so, the function is valued 1, and 0 otherwise. Now we have that $z = x + y$ if and only if $\forall n \; z =_n x + y$, which is Δ^1_0. The proof is similar for the remaining operations. □

A single real number can code any finite tuple of real numbers by alternating the digits of the real numbers in the tuple (we will make use of this fact in the next chapter). In this sense, we write $y_{n,i}$ to stand for the i-th real number in the n-ary tuple coded by y. For an m-ary tuple $\mathbf{y}$, we write $\mathbf{y}_{n,i}$ to stand for the tuple $((y_1)_{n,i}, \ldots, (y_m)_{n,i})$. Then it is not hard to see that if some n-ary predicate P is in Δ^1_n (or Σ^1_n, or Π^1_n), then the $(n+1)$-ary predicate $\tilde{P}$ given by

$$\tilde{P}(\mathbf{y}, n) \iff (\forall i \leqslant n) P(\mathbf{y}_{n,i})$$

is also in Δ^1_n (resp. Σ^1_n, Π^1_n).

Proposition 12.8 (Loff [2007a]). *All real recursive functions belong to the analytical hierarchy, in the sense of Definition 12.6.*

Proof. The result is proved by induction on the structure of REC($\mathbb{R}$) presented in Proposition 12.1. Proposition 12.7 gives us the result for the atomic functions. Proposition 12.4 will suffice to show closure under the operators. If f and g are in Σ^1_n, then $\mathbf{C}(f, g)$ is in Σ^1_n, since:

$$\mathbf{z} = \mathbf{C}(f, g)(\mathbf{x}) \iff \exists \mathbf{y} \; \mathbf{z} = f(\mathbf{y}) \wedge \mathbf{y} = g(\mathbf{x}).$$

Let f be an m-ary total locally Lipschitz function with m components in Σ^1_n. Then $\bar{\mathbf{I}}(f)$ is in Σ^1_n, since $\mathbf{z} = \bar{\mathbf{I}}(f)(\mathbf{x}, y)$ if and only if

$$\exists \mathbf{w} \exists k \, [k = \lfloor |y| \rfloor \wedge \mathbf{w}_{k,1} = f(\mathbf{x}) \wedge ((\forall i \leqslant k))[\mathbf{w}_{k,i+1} = f(\mathbf{w}_{k,i})] \wedge \mathbf{z} = \mathbf{w}_{k,k}]$$

If f is an $(m+1)$-ary function in Σ^1_n, then $\mathbf{Sup}(f) \in \Pi^1_{n+1} \subseteq \Sigma^1_{n+2}$, since

[1] e.g. $x = 101.1\underline{1}0000$ and $y = 101.1\underline{0}1111$, where the first different digit is underlined.

$$\mathbf{z} = \mathbf{Sup}(f)(\mathbf{x}) \iff (\forall i \leqslant m)\forall y\ z_i \geqslant (f(\mathbf{x}, y))_i \wedge (\forall \tilde{z} < z_i)\exists \tilde{y}\ \tilde{z} < (f(\mathbf{x}, \tilde{y}))_i.$$

Furthermore, if f, g are in Σ^1_n then $\mathbf{V}(f, g)$ is trivially also in Σ^1_n. □

12.2 Real recursive functions and the analytical hierarchy

In this chapter, we show one of the most important results of this text:

Theorem 12.9 (Loff [2007a]). REC(ℝ) *is the class of functions with a graph in the analytical hierarchy, i.e.,*

$$\mathrm{REC}(\mathbb{R}) = \{f : \textit{the predicate given by } \mathbf{z} = f(\mathbf{x}) \textit{ is in } \Delta^1_\omega\}.$$

This will be carried out in a few steps. We have already shown in the previous section that every real recursive function has a graph in the analytical hierarchy. We will now prove (I) that every predicate in the analytical hierarchy has a real recursive characteristic, and (II) that if the graph of a function has a real recursive characteristic then the function itself is real recursive.

Proposition 12.10 (Mycka and Costa [2004]). *The characteristic of every predicate $P \in \Pi^1_1$ has a real recursive extension.*

Proof. We use the normal form theorem [Odifreddi, 1989, p.380] for Π^1_1 predicates, which states that $P \in \Pi^1_1$ if and only if some recursive predicate R verifies

$$P(\mathbf{x}, \mathbf{a}) \iff \forall y \exists b\ R(\mathbf{x}, y; \mathbf{a}, b).$$

But then, setting $Q(\mathbf{x}, y; \mathbf{a}) \iff \exists b\ R(\mathbf{x}, y; \mathbf{a}, b)$, we get a predicate $Q \in \Sigma^0_1 \subset \Delta^0_2$.[2] From Shoenfield's limit lemma [Odifreddi, 1989, p. 373], there must then be a recursive functional G such that

$$\lim_{b \to +\infty} G(\mathbf{x}, y; \mathbf{a}, b) = \chi_Q(\mathbf{x}, y; \mathbf{a}) = \begin{cases} 1 & \text{if } \exists b\ R(\mathbf{x}, y; \mathbf{a}, b) \text{ holds} \\ 0 & \text{otherwise.} \end{cases}$$

Above, the variable b ranges over the natural numbers. Therefore, by Proposition 10.47, G must have a real recursive extension g, and so the characteristic of Q:

$$\lim_{z \to +\infty} g(\mathbf{x}, y, \mathbf{a}, \lfloor z \rfloor) = \chi_Q(\mathbf{x}, y; a)$$

must also have a real recursive extension. We then set

[2] These are levels in the arithmetical hierarchy, which is defined in a similar way to the analytical hierarchy; cf. [Odifreddi, 1989, p.367].

$$\chi_P(\mathbf{x}, \mathbf{a}) = \inf_{y \in \mathbb{R}} \chi_Q(\mathbf{x}, y, \mathbf{a}) = \begin{cases} 1 & \text{if } P(\mathbf{x}, \mathbf{a}) \text{ holds} \\ 0 & \text{otherwise} \end{cases}$$ □

Proposition 12.11 (Loff [2007a]). *The characteristic of every predicate P in the analytical hierarchy is real recursive.*

Proof. All predicates in $\Delta^1_0 \subset \Pi^1_1$ have real recursive characteristics, by the previous proposition. We now show that if P is an $(n+1)$-ary predicate with a real recursive characteristic χ_P, then there are real recursive characteristics of the predicates given by $\forall y P(\mathbf{x}, y)$ and $\exists y P(\mathbf{x}, y)$. We have shown in Proposition 10.31 that if a function is real recursive, then so is its supremum and infimum over $\mathbb{R}$. So we have that $\forall y P(\mathbf{x}, y)$ if and only if $\mathbf{Inf}(\chi_P)(\mathbf{x}) = 1$ and that $\exists y P(\mathbf{x}, y)$ if and only if $\mathbf{Sup}(\chi_P)(\mathbf{x}) = 1$. This way we conclude that all analytical predicates have real recursive characteristics. □

The proof of (II) is easy for scalar functions.

Proposition 12.12 (Loff [2007a]). *Let χ_f denote the characteristic function of the graph of $f : \mathbb{R}^m \to \mathbb{R}$, i.e.,*

$$\chi_f(z, \mathbf{x}) = \begin{cases} 1 & \text{if } z = f(\mathbf{x}) \\ 0 & \text{otherwise.} \end{cases}$$

If χ_f is real recursive, then so is f.

Proof. We construct a search operator, somewhat like minimisation, but with the whole $\mathbb{R}$ as search domain. Consider again the function $\sigma(x) = \frac{e^x}{1+e^x}$ and its inverse $\sigma^{-1}(y) = \log(y) - \log(1-y)$. The function σ surjectively maps $\mathbb{R}$ into $(0, 1)$. So let

$$F(\mathbf{x}, z) = (1 - \chi_f(z, \mathbf{x})) + \chi_f(z, \mathbf{x})\sigma(z) = \begin{cases} \sigma(z) & \text{if } z = f(\mathbf{x}), \\ 1 & \text{otherwise.} \end{cases}$$

We may then set

$$f(\mathbf{x}) = \sigma^{-1}(\mathbf{Inf}(F)(\mathbf{x})).$$ □

Because every graph of every function in the analytical hierarchy must be real recursive (Proposition 12.11), and if the graph of such a scalar function is real recursive, then so is the function itself (Proposition 12.12), we get the following.

Corollary 12.13. *Every scalar function in the analytical hierarchy is real recursive.*

Notice that the class of functions with a graph in the analytical hierarchy is closed for component selection, and this immediately gives us Theorem 12.9. But we may make a more explicit proof.

Definition 12.14. *The function $\tilde{\gamma}$ is the injection from $(0,1)^2$ to $(0,1)$ given by*

$$\tilde{\gamma}(x,y) = \sum_{i=1}^{\infty} D(x,i)2^{-2i+1} + D(y,i)2^{-2i},$$

where $D(x,i)$ denotes the i-th digit of the binary expansion of x; $\tilde{\gamma}_1$ and $\tilde{\gamma}_2$ denote each component of the inverse of $\tilde{\gamma}$, given by

$$\tilde{\gamma}_1(z) = \sum_{i=1}^{\infty} D(z,2i-1)2^{-i}, \qquad \tilde{\gamma}_2(z) = \sum_{i=1}^{\infty} D(z,2i)2^{-i}.$$

Then we set $\gamma, \gamma_1, \gamma_2$ to be given by

$$\gamma(x,y) = \tilde{\gamma}(\sigma(x),\sigma(y)), \qquad \gamma_1(z) = \sigma^{-1}(\tilde{\gamma}_1(z)), \qquad \gamma_2(z) = \sigma^{-1}(\tilde{\gamma}_2(z))$$

We may easily see that γ, γ_1 and γ_2 are scalar functions in the analytical hierarchy, forming an injection from $\mathbb{R}^2$ to $(0,1)$. So by Corollary 12.13, with the use of composition and aggregation, we conclude the following.

Proposition 12.15 ([Mycka, 2003a]). *The functions γ, γ_1 and γ_2 are real recursive. Furthermore, for every n there are two real recursive functions $\boldsymbol{\gamma}_n : \mathbb{R}^n \to (0,1)$ and its inverse $\boldsymbol{\gamma}_n^{-1} : (0,1) \to \mathbb{R}^n$ forming an injection from $\mathbb{R}^n$ to $(0,1)$.*

We could construct such functions H_3 by using sums and infinite limits. The following final corollary implies Theorem 12.9.

Corollary 12.16. *Let χ_f denote the characteristic function of the graph of $f : \mathbb{R}^m \to \mathbb{R}^n$. If χ_f is real recursive, then so is f.*

Proof. Consider the function $\tilde{f}$, given by

$$\tilde{f}(\mathbf{x}) = \boldsymbol{\gamma}_n(f(x)).$$

Then $\tilde{f}$ is scalar, and its characteristic function is given by

$$\chi_{\tilde{f}}(z,\mathbf{x}) = \begin{cases} 1 & \text{if } z \in (0,1) \text{ and } \chi_f(\boldsymbol{\gamma}_n^{-1}(z),\mathbf{x}) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

This expression gives a real recursive function, and so $\tilde{f}$ is real recursive by Proposition 12.12. But then $f(\mathbf{x}) = \boldsymbol{\gamma}_n^{-1}(\tilde{f}(\mathbf{x}))$, and therefore f is also real recursive. □

Towards solving the problem of collapse

Consider the rank hierarchy for the infinite limit operator under the algebra I:

Definition 13.1. *The **ι-hierarchy** is the rank hierarchy for the limit operator under the algebra* I *for* REC($\mathbb{R}$)*. We use* I_k *to denote the n-th level of this hierarchy. In symbols,*

$$\mathrm{I}_k = H_k^{\mathrm{I},\mathbf{Ls}} = \{f \in \mathrm{REC}(\mathbb{R}) : \mathrm{rk}_{\mathbf{Ls}}^{\mathrm{I}}(f) \leqslant k\}.$$

A clearer picture for this hierarchy may be obtained from the following corollary of Proposition 9.15.

Corollary 13.2. *The ι-hierarchy is inductively given by:*

1. $\mathrm{I}_0 = [-1^n, 0^n, 1^n, U_i^n; \mathbf{C}, \bar{\mathbf{I}}, \mathbf{V}]$,
2. $\tilde{\mathrm{I}}_k = \mathrm{I}_k \cup \{\mathbf{Ls}(f) : f : \mathbb{R}^{m+1} \to \mathbb{R}^n$ *is in* $\mathrm{I}_k\}$*, and*
3. $\mathrm{I}_{k+1} = [\tilde{\mathrm{I}}_k; \mathbf{C}, \bar{\mathbf{I}}, \mathbf{V}]$.

The purpose of this chapter is to show that the ι-hierarchy does not collapse. In Section 13.1 we show that REC($\mathbb{R}$) is closed for unrestricted iteration, and that there is a real recursive way to manipulate stacks of functions. In Section 13.2 we construct restrictions of universal real recursive functions for bounded levels in the ι-hierarchy. In Section 13.3 we will conclude that the ι-hierarchy does not collapse, and explain why this result does not immediately imply a similar result for the η-hierarchy.

13.1 Unrestricted iteration and stacks

From the previous chapter we immediately see that REC($\mathbb{R}$) is closed for unrestricted iteration.

Definition 13.3. *The **iteration operator** $\mathbf{I}$ maps any n-ary function f with n components into an $(n+1)$-ary function with n components $\mathbf{I}(f)$, given by*

$$\mathbf{I}(f)(\mathbf{x}, t) = f^{\lfloor |t| \rfloor}(\mathbf{x}) = \underbrace{f \circ f \circ \ldots \circ f}_{\lfloor |t| \rfloor \ \textit{times}}(\mathbf{x}).$$

Notation 13.4. When the given number of iterations is expected to be a natural number, we omit the flooring and absolute value symbols in the number of iterations. Then $f^n(\mathbf{x})$ is an abbreviation to $f^{\lfloor |n| \rfloor}(\mathbf{x})$.

Proposition 13.5. REC($\mathbb{R}$) *is effectively closed under* **I**.

Proof. The analytical hierarchy is easily closed under **I**, by using the same expression we have used in the proof of Proposition 12.8 to obtain unrestricted iteration. □

Using this unrestricted iteration operator, we may find a way to manipulate stacks of real values by using the pairing function γ of the previous chapter. We denote a stack of real numbers by $\#x_n \ldots x_1$, where # marks the top of the stack. We represent an empty stack by the number 0. The stack $\#x_n \ldots x_1$ is represented with the number $\gamma(x_n, \ldots \gamma(x_1, 0) \ldots)$. Recall that the range of γ is $(0, 1)$, and so there is no risk of confusing an empty stack with a non-empty stack.

We can then define four *basic stack manipulation functions.* The psh function, which pushes a value on top of the stack, is given by $\mathrm{psh}(S, x) = \gamma(x, S)$. The pop function removes the top of the stack: $\mathrm{pop}(S) = \gamma_2(S)$. The top function gives the value on the top of the stack, and 0 if the stack is empty: $\mathrm{top}(S) = (1 - \delta(S))\gamma_1(S + \frac{1}{2}\delta(S))$. The emp function gives 1 if the stack is empty and 0 otherwise: $\mathrm{emp}(S) = \delta(S)$. We abbreviate $\mathrm{top}(\mathrm{pop}^{n-1}(S)) \equiv \mathrm{tp}(S, n)$. More complex stack manipulation functions can be defined using the four basic functions. The function swt, for instance, pushes the top of the stack into the (n + 1)th position: $\mathrm{swt}(S, n) = U_1^4(\mathbf{I}(f)(S, 0, n, 0, 2n + 2))$, where

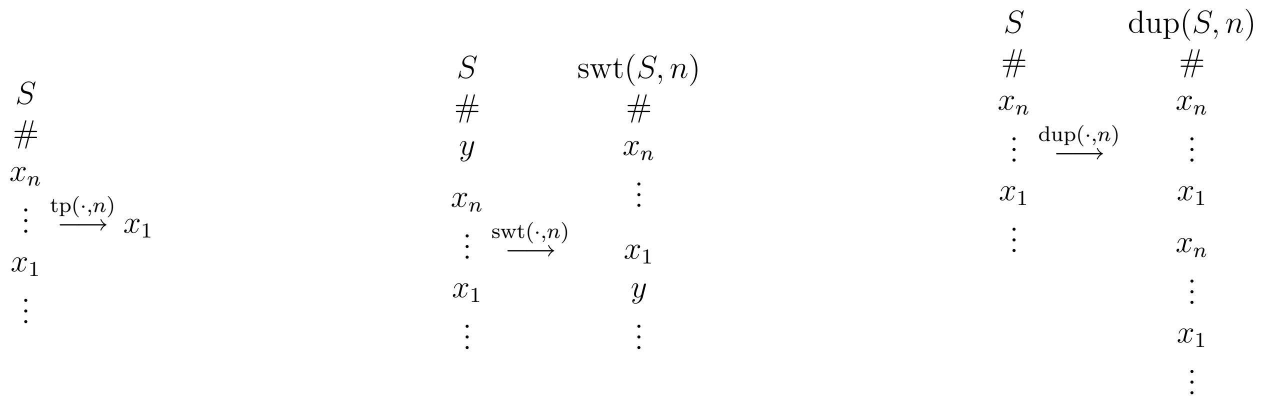
$$f(S_1, S_2, n, r) = \begin{cases} (\mathrm{pop}(S_1), 0, n, \mathrm{top}(S_1)) & \text{if } r = 0 \text{ and } n > 0, \\ (\mathrm{pop}(S_1), \mathrm{psh}(S_2, \mathrm{top}(S_1)), n - 1, r) & \text{if } r \neq 0, \text{ and } n > 0, \\ (\mathrm{psh}(S_1, r), S_2, 0, 0) & \text{if } r \neq 0 \text{ and } n = 0, \\ (\mathrm{psh}(S_1, \mathrm{top}(S_2)), \mathrm{pop}(S_2), 0, 0) & \text{if } r = 0 \text{ and } n = 0. \end{cases}$$

Notice that the above definition by cases can be *implemented* using the characteristics of equality and inequality, along with products and sums. The function dup duplicates the top n elements of the stack: $\mathrm{dup}(S, n) = U_1^4(\mathbf{I}(g)(S, 0, 0, n, 3n + 1))$, where

$$g(S_1, S_2, S_3, n) = \begin{cases} (\mathrm{pop}(S_1), \mathrm{psh}(S_2, \mathrm{top}(S_1)), 0, n - 1) & \text{if } n > 0, \\ (S_1, S_2, S_2, 0) & \text{if } n = 0, \text{ and emp}(S_3), \\ (\mathrm{psh}(S_1, \mathrm{top}(S_2)), \mathrm{pop}(S_2), S_3, 0) & \text{if } n = 0 \text{ and not emp}(S_2), \\ (\mathrm{psh}(S_1, \mathrm{top}(S_3)), S_2, \mathrm{pop}(S_3), 0) & \text{if } n = 0 \text{ and not emp}(S_3). \end{cases}$$

The effect of $\mathrm{tp}(S, n)$, $\mathrm{swt}(S, n)$ and $\mathrm{dup}(S, n)$ on the stack S are illustrated on Fig. 13.1.

In the next chapter, we will construct the universal real recursive functions $\Psi_k^{m,n}$ for m-ary, n-component functions in I_k, by manipulating an analogue of an *execution stack.*



S
$\#$
x_n
$\vdots \xrightarrow{\mathrm{tp}(\cdot,n)} x_1$
x_1
$\vdots$
S | $\mathrm{swt}(S,n)$
$\#$ | $\#$
y | x_n
x_n | $\vdots$
$\vdots \xrightarrow{\mathrm{swt}(\cdot,n)}$ | x_1
x_1 | y
$\vdots$ | $\vdots$
S | $\mathrm{dup}(S,n)$
$\#$ | $\#$
x_n | x_n
$\vdots \xrightarrow{\mathrm{dup}(\cdot,n)}$ | $\vdots$
x_1 | x_1
$\vdots$ | x_n
$\vdots$
x_1
$\vdots$

Fig. 13.1. Applying tp, swt and dup to a stack S

13.2 Universal level-bounded functions

We show that the ι-hierarchy does not collapse by constructing functions $\Psi_k^{m,n}$ which are universal for I_k.

Definition 13.6. *A function* $\Psi_k^{m,n} : \mathbb{R}^{m+1} \to \mathbb{R}^n$ *is called a* ***universal function for*** I_k *if for every good description* $d_e^{m,n}$ *with* $\mathrm{rk}(d_e^{m,n}) \leqslant k$*, and every* $\mathbf{x} \in \mathbb{R}^m$*,*

$$\Psi_k^{m,n}(e,\mathbf{x}) \simeq \phi_e^{m,n}(\mathbf{x}).$$

Theorem 13.7 (Loff et al. [2007]). *For any* m, n, k*, there is a universal function* $\Psi_k^{m,n}$ *for* I_k*.*

Proof. We construct functions that simulate any real recursive function step-by-step, given its code, by maintaining two stacks. On the first stack we keep real values and on the second stack we maintain codes of descriptions of real recursive functions or of stack manipulation instructions. A switch instruction $\mathrm{swt}(\cdot, n)$ is represented by the number $3n+1$, and a duplicate instruction $\mathrm{dup}(\cdot, n)$ is represented by $3n + 2$. The description d_e is represented by $3e$. If the description in the top of the second stack describes an n-ary function, it is expected that n real values are in the first stack, each corresponding to one argument, with the last argument on top. To implement the aggregation operator, it will be necessary to duplicate and switch the contents of the stack, and that is why we encode the swt and dup instructions. Let $\tilde{\Psi}_0$ be given by:

$$\tilde{\Psi}_0(S_1,S_2)=\begin{cases}
(\mathrm{psh}(\mathrm{pop}^n(S_1),1),\mathrm{pop}(S_2)) & \text{if } \frac{1}{3}\mathrm{top}(S_2) \text{ is } \langle \texttt{fun}_{1^n}\rangle,\\
(\mathrm{psh}(\mathrm{pop}^n(S_1),-1),\mathrm{pop}(S_2)) & \text{if } \frac{1}{3}\mathrm{top}(S_2) \text{ is } \langle \texttt{fun}_{-1^n}\rangle,\\
(\mathrm{psh}(\mathrm{pop}^n(S_1),0),\mathrm{pop}(S_2)) & \text{if } \frac{1}{3}\mathrm{top}(S_2) \text{ is } \langle \texttt{fun}_{0^n}\rangle,\\
(\mathrm{psh}(\mathrm{pop}^n(S_1),\mathrm{tp}(S_1,n-i+1)),\mathrm{pop}(S_2)) & \text{if } \frac{1}{3}\mathrm{top}(S_2) \text{ is } \langle \texttt{fun}_{U_i^n}\rangle,\\
(\mathrm{psh}(\mathrm{pop}^2(S_1),\mathrm{tp}(S_1,2)+\mathrm{top}(S_1)),\mathrm{pop}(S_2)) & \text{if } \frac{1}{3}\mathrm{top}(S_2) \text{ is } \langle \texttt{fun}_{+}\rangle,\\
(\mathrm{psh}(\mathrm{pop}^2(S_1),\mathrm{tp}(S_1,2)\times\mathrm{top}(S_1)),\mathrm{pop}(S_2)) & \text{if } \frac{1}{3}\mathrm{top}(S_2) \text{ is } \langle \texttt{fun}_{\times}\rangle,\\
(\mathrm{psh}(\mathrm{pop}^2(S_1),\mathrm{tp}(S_1,2)^{\mathrm{top}(S_1)}),\mathrm{pop}(S_2)) & \text{if } \frac{1}{3}\mathrm{top}(S_2) \text{ is } \langle \texttt{fun}_{x^y}\rangle,\\
(S_1,\mathrm{psh}(\mathrm{psh}(\mathrm{pop}(S_2),3e_1),3e_2)) & \text{if } \frac{1}{3}\mathrm{top}(S_2) \text{ is } \langle \texttt{Op}_{\mathbf{C}},d_{e_1},d_{e_2}\rangle,\\
(\mathrm{pop}(S_1),\mathrm{psh}^{\lfloor|\mathrm{top}(S_1)|\rfloor}(\mathrm{pop}(S_2),3e)) & \text{if } \frac{1}{3}\mathrm{top}(S_2) \text{ is } \langle \texttt{Op}_{\bar{\mathbf{I}}},d_e\rangle,\\
(S_1,\mathrm{aggr}(\mathrm{pop}(S_2),3e_1,3e_2)) & \text{if } \frac{1}{3}\mathrm{top}(S_2) \text{ is } \langle \texttt{Op}_{\mathbf{V}},d_{e_1},d_{e_2}\rangle,\\
(\mathrm{swt}(S_1,n),\mathrm{pop}(S_2)) & \text{if } \mathrm{top}(S_2) \text{ is } 3n+1,\\
(\mathrm{dup}(S_1,n),\mathrm{pop}(S_2)) & \text{if } \mathrm{top}(S_2) \text{ is } 3n+2,\\
(S_1,S_2) & \text{if } \mathrm{emp}(S_2).
\end{cases}$$

where for every m-ary descriptions d_{e_1} with n components and d_{e_2} with k components, $\mathrm{aggr}(S,3e_1,3e_2)$ carries out the following pushes to the stack S (the corresponding instructions are shown in parenthesis):

1. push $3m+2$ (duplicate the top m elements);
2. push $3e_1$ (apply the function described by e_1);
3. push $3(m+n-1)+1$ a total of n times (move the result of applying this function below the previously duplicated values);
4. push $3e_2$ (apply the function described by e_2).

By induction on the structure of I we conclude that if S_1 encodes a stack with the real numbers $x_n,\ldots,x_1,y_j,\ldots,y_1$, and S_2 encodes a stack with the numbers $3e,e_1,\ldots e_k$, where d_e describes an n-ary function with m components and $\mathrm{rk}(d_e)=0$, then, by iterating $\tilde{\Psi}_0$, S_2 will eventually contain only $e_1,\ldots,e_k$ and then we will have m real numbers in the top of S_1, given by each component of $\phi_e(x_1,\ldots,x_n)$, followed by $y_j,\ldots,y_1$. This is illustrated by Figure 13.2.

$$\begin{array}{cc ccc cc}
S_1 & S_2 & & & & S_1 & S_2\\
\# & \# & & & & \# & \#\\
x_n & 3e & & & & (\phi_e(x_1,\ldots,x_n))_m & e_1\\
\vdots & e_1 & & & & \vdots & \vdots\\
x_1 & \vdots & \xrightarrow{\tilde{\Psi}_0} & \ldots & \xrightarrow{\tilde{\Psi}_0} & (\phi_e(x_1,\ldots,x_n))_1 & e_k\\
y_j & e_k & & & & y_j & \\
\vdots & & & & & \vdots & \\
y_1 & & & & & y_1 &
\end{array}$$

Fig. 13.2. Stack manipulation by $\tilde{\Psi}_0$

We may set

$$\Psi_0(S,e) = U_1^2(\lim_{z\to+\infty} \mathbf{I}(\tilde{\Psi}_0)(S, \mathrm{psh}(0,e)),$$

and then Ψ_0 is a universal function when the input and output is given using a stack. Now we set

$$(\Psi_0^{m,n}(e,\mathbf{x}))_i = \mathrm{tp}(\Psi_0(\mathrm{psh}(0,\mathbf{x}), 3e), i);$$

obtaining a universal function for I_0; above, $\mathrm{psh}(0,\mathbf{x})$ abbreviates $\mathrm{psh}(\ldots\mathrm{psh}(0,x_1),\ldots x_m)$.

Now, for every $k \geqslant 1$, we define $\tilde{\Psi}_k$ as

$$\tilde{\Psi}_k(S_1,S_2) = \begin{cases} (\limsup_{y\to\infty} \Psi_{k-1}(\mathrm{psh}(S_1,y),3e), \mathrm{pop}(S_2)) & \text{if } \frac{1}{3}\mathrm{top}(S_2) \text{ is } \langle \mathtt{Op}_{\mathbf{Ls}}, d_e\rangle, \\ \tilde{\Psi}_0(S_1,S_2) & \text{otherwise;} \end{cases}$$

$$\Psi_k(S,e) = U_1^2(\lim_{z\to+\infty} \mathbf{I}(\tilde{\Psi}_k)(S, \mathrm{psh}(0,e));$$

and

$$(\Psi_k^{m,n}(e,\mathbf{x}))_i = \mathrm{tp}(\Psi_k(\mathrm{psh}(0,\mathbf{x}), 3e), i);$$

In this way, $\Psi_k^{m,n}$ is in some bounded level of the I hierarchy (not necessarily I_k), and simulates any real recursive function in I_k when given one of its codes with rank bounded by k. □

13.3 Conclusions for the problem of collapse

Notice that the existence of our universal functions for I_k will not give us the result of non-collapse. We will need the following concept.

Definition 13.8. *A number $e \in \mathbb{N}$ is called a* ***low-rank code*** *of ϕ_e if $\mathrm{rk}(d_e) = \mathrm{rk}(\phi_e)$.*

And now we show:

Proposition 13.9 (Loff et al. [2007]). *There is no real recursive function which restricts a universal function to low-rank codes, i.e., there is no real recursive function $\psi^{m,n}$ such that if e is a low-rank code then*

$$\psi^{m,n}(e,\mathbf{x}) \simeq \phi_e(\mathbf{x}) \textit{ for all } \mathbf{x} \in \mathbb{R}^m,$$

whenever d_e is a good description of an m-ary function ϕ_e with n components.

Proof. We take the same expression for g of the proof of Proposition 11.12, now using a totalisation of $\psi^{m,n}$. We then choose a low-rank code e of g, rather than any code of g. □

Now we may conclude:

Theorem 13.10 (Loff et al. [2007]). *The ι-hierarchy does not collapse.*

Proof. Suppose, by contradiction, that it would collapse. Then for some number k we would have $\mathrm{REC}(\mathbb{R}) \subseteq \mathrm{I}_k$. But then every low-rank code would have a rank of at most k, and we could set $\psi^{m,n} = \Psi_k^{m,n}$ and obtain a universal function restricted to low-rank codes. This contradicts the previous proposition, and so we are forced to conclude that the ι-hierarchy does not collapse. □

But how about the η-hierarchy? The non-collapsing character of the ι-hierarchy does not imply the non-collapse of the η-hierarchy, because we have not shown that every bounded level in the η-hierarchy is fully contained on some bounded level in the ι-hierarchy. In fact, we use two infinite limits for each differential recursion in our proof that $\mathrm{REC}(\mathbb{R}) \subseteq \mathrm{I}$, and so by nesting differential recursions in a description $d \in \mathcal{D}_{\mathrm{H}}$, our proof would give a description $\tilde{d} \in \mathcal{D}_{\mathrm{I}}$ with twice the rank as the number of nested differential recursions. We believe that this does not have to be the case, i.e., that the η-hierarchy is collapse-equivalent on the ι-hierarchy, in the following sense:

Definition 13.11. *Two hierarchies $\mathcal{A}_\omega$ and $\mathcal{B}_\omega$ are called **collapse-equivalent** if for all n there is a number m such that $\mathcal{A}_n \subseteq \mathcal{B}_m$ and $\mathcal{B}_n \subseteq \mathcal{A}_m$.*

We could easily prove the following proposition.

Proposition 13.12. *The ι-hierarchy and the analytical hierarchy are collapse equivalent.*

We conclude this chapter with an open conjecture, once believed to have been proven true [Loff et al., 2007]. We now know that the proof was flawed. The conjecture implies the non-collapse of the η-hierarchy, and this is the justification for this chapter's name.

Conjecture 13.13. The η-hierarchy and the ι-hierarchy are collapse equivalent.

Conclusions, final remarks and criticism

This dissertation has dealt with many issues, all directly or indirectly related with the link between physics and computation. The main purpose of Part I was to emphasise the importance of investigating this relationship; we have tried to provide historically and physically relevant thinking tools for such an investigation. It is our hope that Part I establishes adequate naming and formulation for the Church–Turing thesis, the physical Church–Turing thesis and the simulation thesis.

Much more could have been said, and there where several issues left undiscussed — such as biological computing, or the strong A.I. thesis. We did not present the counterparts to the physical Church–Turing and simulation theses; these claim, respectively, that there *are* physical systems computing more than the Turing machine, or physical systems which may not be simulated by a Turing machine. These anti-theses appear in our [Loff and Costa, 2007].

Overall, this relationship is a hard subject to study. It is not the same thing to say that *Newtonian mechanics describes non-computational systems*, than it is to say that there are an infinite number of primes. While the former sentence seems to be true, by ours or Smith's [2006a] work, the results are still open to re-interpretation. It is difficult to take everything in consideration, and to produce relevant statements. This is why we value Beggs and Tucker's methodology so highly, because it requires an *open and explicit* self-criticism and an analysis of the relevance of the obtained results.

In Part II we offer a case study of an investigation of the physical Church–Turing thesis. Throughout our discussion, we have always made clear that the systems under consideration arise from a very idealised theory of Newtonian mechanics. It is therefore not always possible to grasp how the study is relevant, and this is why we dedicated Section 8.1 to this problem.

One of the most interesting results in Part II is that the hypercomputational effect is not removed when we consider a simplistic error model, i.e., that a physical oracle may still be read by sampling, up to any desired precision, when making certain assumptions on the sampling process. Future work should concentrate on more realistic error models, to discover whether the computational power drops to a more reasonable level using these models. We conjecture that very simple changes in the error model should suffice to remove the super-Turing power of the scatter machine.

Part III was dedicated to the development of real recursive function theory. Our intention was to write a foundational text on the more general aspects of real recursion theory. This was accomplished with some, partial success, and we intend to further develop part III into a longer, more detailed text. The introduction of the Lipschitz condition requires a more in-depth treatment: there is a nice relationship between the Lipschitz constant of a function $f \in Dom(\mathbf{R})$ and the size of the disk in a disk-and-wheel integrator, which must be further explored.

We would like to solve the problem of the collapse of the η-hierarchy, and believe that Conjecture 13.13 will play an important part in this endeavour. Further exploration of the more general aspects of the theory may concern the relationship between real recursive functions and generalised computability, or see how the concept of differential recursion could be adapted to higher types.

The relationship between physics and computability still needs plenty of exploration, and we predict that this will be an important field in the coming years. The study of physics and dynamical systems often results in elegant and beautiful mathematical models and theories; hopefully, these will provide surprising connections and make for a positive contribution to our mathematical heritage.

A

Brief introduction to non-uniform complexity

The Turing machine is the result of Turing's conceptual analysis of the process of systematic calculation carried out by humans. This conceptual analysis did not, however, intend to model the possibility that the human being who carries out the calculation may use an auxiliary device which he does not entirely understand. To cover this situation, Turing introduced his o-machines, now called oracle Turing machines. When we combine experiments and algorithms, in the following sections, we will obtain a model of computation similar to the oracle Turing machine. For this reason, we begin with a thorough discussion of deterministic oracle Turing machines and of probabilistic Turing machines. We then briefly recall some notions of non-uniform complexity, which will characterise our model of computation.

A.1 The oracle Turing machine

The deterministic oracle Turing machine is a device comprising a **control unit**, a finite number of **tapes** and an **oracle**. The tapes are a memory medium, formed by a linear arrangement of an infinite number of individual **cells** (say, of paper). In each cell may be printed one symbol from a finite alphabet Σ. The oracle is a question-answering black box. Each word in Σ^* is interpreted as a question, and to each question the oracle deterministically gives an answer, yes or no. An oracle Turing machine has at least two tapes, where calculations and general rough work can be carried out. One of the tapes, called the **query tape**, is selected for the purpose of asking questions to the oracle. Another, previously designated tape, called **input tape**, is used for input.

The control unit is the mechanism which governs the computation, by reading and writing symbols in the tapes and by consulting the oracle. The control unit is said to be finite, since it can only be in one of a finite number of internal **states**. The control unit reads and writes symbols in the tapes by means of certain **tape heads**, one for each tape. Each tape head is placed in a specific cell of its tape, and can read or write a symbol in the cell it is placed on, or move to one of the adjacent cells, to left or to the right. It is based on the symbols read by the tape heads and on the current state that the control unit decides what to do next. Among the states of the finite control unit are six special states. Three of these states are used by the control unit to begin and halt the computation: these are called the **initial state**, the

accepting state, and the **rejecting state**. The remaining three states serve to interact with the oracle. These are called the **query state**, the **"yes" state**, and the **"no" state**.

The control unit divides the computation in distinct computational steps. At each step, the control unit reads the symbols underneath the tape heads, and, based on the symbols which where read and on the current state, decides (1) which symbols should be written in the tapes, (2) whether each tape head moves to the left or to the right and (3) what should be the internal state of the finite control in the next step of computation. In order to ask the oracle a question, the control unit writes the question in the query tape and moves to the query state. The finite control will then be interrupted, and the oracle will answer the question by resuming the computation in either the "yes" state or the "no" state, with the obvious meaning.

We call **configuration** to the *global condition* of the Turing machine at a certain step, i.e., the state of the finite control, plus the contents of the tapes, plus the position of the tape heads. We can see that the configuration of an oracle Turing machine at the next step deterministically depends on the configuration of the oracle Turing machine at the current step.

A computation of the oracle Turing machine on a word w begins with w written in the input tape of the machine, with the input tape head placed on the left-most symbol of w, and with the control unit in the initial state. The computation will proceed as long as the control unit does not enter either the accepting or rejecting states. The input w is said to be **accepted** if the computation halts in the accepting state, and **rejected** if it halts in the rejecting state.

We may then establish a decision criterion for the deterministic oracle Turing machine. Let $A \subseteq \Sigma^*$ be a set of words over Σ. We say that A is decided by an oracle Turing machine $\mathcal{M}$ if, for every input $w \in \Sigma^*$, w is accepted by $\mathcal{M}$ when $w \in A$ and rejected by $\mathcal{M}$ when $w \notin A$. We say that $\mathcal{M}$ decides A in polynomial time, if $\mathcal{M}$ decides A, and, for every $w \in \Sigma^*$, the number of steps of the computation is polynomial in the size of w.

Notice that to each oracle corresponds a unique set O containing exactly the words to which the oracle answers *yes*. We may thus denote an oracle by its corresponding set O. In this sense, we write $\mathrm{P}(O)$ to stand for the class of sets decidable in polynomial time by machines with the oracle O, and abbreviate $\mathrm{P} = \mathrm{P}(\varnothing)$.

A.2 The probabilistic Turing machine

A probabilistic Turing machine is very similar to the oracle Turing machine, but instead of a question answering black box, the finite control makes use of a **coin**. The probability of the coin turning up **heads** is $p \in [0, 1]$, and the probability of it giving **tails** is $q = 1 - p$. It is common to assume that $p = \frac{1}{2}$, i.e., that the coin is balanced. This assumption is vital for the usual understanding of probabilistic machines, since a probabilistic machine with a rational p can be simulated by a deterministic machine, but this is not necessarily true for an arbitrary real p.

The coin can be seen as a non-deterministic oracle which will always, when queried, answers *yes* with probability p, and *no* with probability q. We can see that in this situation the query tape is useless, and may be removed, but that otherwise the behaviour of the probabilistic machine is in every way like that of the oracle Turing machine. We thus choose to maintain the previous description, but renaming the query, "yes" and "no" states to be called the **toss state**, the **heads state** and the **tails state**.

The decision criterion, however, must be different, because unless the control unit makes no use of the coin, the machine will not deterministically accept or reject the input, i.e., the configuration of a probabilistic Turing machine at the next step does not follow deterministically from the configuration at the current step. Several criteria could be put forward, and we will here describe one of the most common probabilistic decision criterion. For a set $A \subseteq \Sigma^*$, a probabilistic Turing machine $\mathcal{M}$, and an input $w \in \Sigma^*$, the **error probability** of $\mathcal{M}$ for input w is the probability of $\mathcal{M}$ rejecting w if $w \in A$, or the probability of $\mathcal{M}$ accepting w if $w \notin A$. We say that $\mathcal{M}$ decides A with **bounded error probability** if there is a number $\gamma < \frac{1}{2}$, such that the error probability of $\mathcal{M}$ for any input w is smaller than γ.[1] A is decided in polynomial time with bounded error probability if, for every input w, the number of steps in the possible computations is always polynomial in the length of w. We write BPP to stand for the class of sets decidable in polynomial time with bounded error probability using a balanced coin.

Below we will study the analog-digital scatter machine, a device which is obtained by allowing the finite control of a Turing machine to interact with a scatter machine, much in the same way as the oracle Turing machine interacts with an oracle and the probabilistic Turing machine interacts with a coin. We will see that, under varying restrictions, the scatter machine will play the role of either an oracle or a coin.

A.3 Non-uniform complexity classes

We will see that non-uniform complexity gives the most adequate characterisations of the computational power of the analog-digital scatter machine.[2] Non-uniform complexity classifies problems by studying families of finite machines (e.g., circuits) $\{\mathcal{C}_n\}_{n\in\mathbb{N}}$, where each $\mathcal{C}_n$ decides the restriction of some problem to inputs of size n. It is called *non-uniform*, because for every $n \neq m$ the finite machines C_n and C_m can be entirely unrelated, while in uniform complexity the algorithm is the same for inputs of every size. A way to connect the two approaches is by means of *advice classes*: one assumes that there is a unique algorithm for inputs of every size, which is aided by certain information, called *advice*, which may vary for inputs of different sizes. The advice is given, for each input w, by means of a pairing function $\langle \cdot, \cdot \rangle : \Sigma^* \times \Sigma^* \to \Sigma^*$. This function and its inverses $(\cdot)_1$ and $(\cdot)_2$ are all computable in linear time and such that $(\langle w, v \rangle)_1 = w$, $(\langle w, v \rangle)_2 = v$, and $\langle (w)_1, (w)_2 \rangle = w$.

[1] Under the assumption that $p = \frac{1}{2}$, one can represent every possible computation of a probabilistic Turing machine on an input w as a complete binary tree, where each node of the tree corresponds to the global state, or configuration, of the machine after each possible sequence of coin tosses. Then the probability of error is simply the fraction of leaves of this tree which incorrectly accept or reject w.

[2] For an interesting definition of how we can compare computational power among different models of computability, see the work of Udi Boker and Nachum Dershowitz [2005].

Definition A.1. *Let $\mathcal{B}$ be a class of sets and $\mathcal{F}$ a class of functions. The advice class $\mathcal{B}/\mathcal{F}$ is the class of sets A for which some $B \in \mathcal{B}$ and some $f \in \mathcal{F}$ are such that, for every w, $w \in A$ if and only if $\langle w, f(|w|)\rangle \in B$.*

$\mathcal{F}$ is called the *advice class* and f is called the *advice function*. Examples for $\mathcal{B}$ are P, or BPP. We will be considering two instances for the class $\mathcal{F}$: *poly* is the class of functions with polynomial size values, i.e., *poly* is the class of functions $f : \mathbb{N} \to \Sigma^*$ such that, for some polynomial p, $|f(n)| \in \mathrm{O}(p(n))$; *log* is the class of functions $g : \mathbb{N} \to \Sigma^*$ such that $|g(n)| \in \mathrm{O}(\log(n))$. We will also need to treat prefix non-uniform complexity classes. For these classes we may only use prefix functions, i.e., functions f such that $f(n)$ is always a prefix of $f(n+1)$. The idea behind prefix non-uniform complexity classes is that the advice given for inputs of size n must also be useful to decide smaller inputs.

Definition A.2. *Let $\mathcal{B}$ be a class of sets and $\mathcal{F}$ a class of functions. The prefix advice class $\mathcal{B}/\mathcal{F}*$ is the class of sets A for which some $B \in \mathcal{B}$ and some prefix function $f \in \mathcal{F}$ are such that, for every length n and input w with $|w| \leqslant n$, $w \in A$ if and only if $\langle w, f(n)\rangle \in B$.*

As examples we have P/*log*∗, or BPP/*log*∗. It is a matter of some controversy whether this is the appropriate definition of BPP/$\mathcal{F}$. Notice that by demanding that there is a set $B \in$ BPP, and a function $f \in \mathcal{F}$ such that $w \in A$ if and only if $\langle w, f(|w|)\rangle \in B$, we are demanding that B is decided with bounded error probability *for any possible advice*, instead of the more intuitive idea that the error only has to be bounded for the correct advice. This leads to the following definitions.

Definition A.3. BPP//*poly is the class of sets A for which a probabilistic Turing machine $\mathcal{M}$, a function $f \in$ poly, and a constant $\gamma < \frac{1}{2}$ exist such that $\mathcal{M}$ rejects $\langle w, f(|w|)\rangle$ with probability at most γ if $w \in A$ and accepts $\langle w, f(|w|)\rangle$ with probability at most γ if $w \notin A$.*

Definition A.4. BPP//*log∗ is the class of sets A for which a probabilistic Turing machine $\mathcal{M}$, a prefix function $f \in$ log, and a constant $\gamma < \frac{1}{2}$ exist such that, for every length n and input w with $|w| \leqslant n$, $\mathcal{M}$ rejects $\langle w, f(n)\rangle$ with probability at most γ if $w \in A$ and accepts $\langle w, f(n)\rangle$ with probability at most γ if $w \notin A$.*

It can be shown that BPP//*poly* = BPP/*poly*, but it is unknown whether BPP//*log*∗ ⊆ BPP/*log*∗. We can safely assume without loss of generality that, for P/*log*∗, BPP/*log*∗ and BPP//*log*∗, the length of any advice $f(n)$ is exactly $\lfloor a \log n + b \rfloor$, for some $a, b \in \mathbb{N}$ which depend on f.

It is important to notice that the usual non-uniform complexity classes contain undecidable sets, e.g., P/*poly* contains the undecidable set

$$\{0^n : \text{the Turing machine coded by } n \text{ halts on input } 0\}.$$

Classes with larger advice usually become uninteresting: exponential advice, for instance, allows a Turing machine to decide any set in linear time. The non-computability of the non-uniform complexity classes results exclusively from the non-computability of the advice functions. In fact, for any class $\mathcal{B}$ of recursive sets, and any class $\mathcal{F}$ of advice functions, $\mathcal{B}/(\mathcal{F} \cap \text{PREC}) \subseteq \text{REC}$, where PREC is the class of partial recursive functions and REC is the whole class of recursive sets. This means that computable advice results in computable behaviour.

B

Notation and glossary

The following notation was used.

1. For any $x \in real$, its **truncation** to the n-th digits is denoted by $x\upharpoonright_n$.
2. $\|\cdot\|$ or $\|\cdot\|_2$ will denote the **Euclidean norm**; $\|\cdot\|_\infty$ denotes the **supremum norm**; in $\mathbb{R}^n$, these are given by

$$\|\mathbf{x}\| = \sqrt{x_1^2 + \ldots + x_n^2} \qquad \|\mathbf{x}\|_\infty = \max(|x_1|, \ldots, |x_n|).$$

3. For some $\mathbf{x} \in \mathbb{R}^n$, $B(\mathbf{x}, r)$ denotes the **open ball** of radius r around $\mathbf{x}$, i.e.,

$$B(\mathbf{x}, r) = \{\mathbf{y} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{y}\| < r\}.$$

   We add a bar, as in $\bar{B}(\mathbf{x}, r)$, to denote the **closed ball** of radius r around $\mathbf{x}$, given by

$$\bar{B}(\mathbf{x}, r) = \{\mathbf{y} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{y}\| \leqslant r\}.$$

4. Given a bounded set $A \subset \mathbb{R}^n$, we use d_A to denote its **diameter**, given by

$$d_A = \sup\{\|x - y\| : x, y \in A\}.$$

5. The symbol $\equiv$ is occasionally used to denote syntactical equivalence or abbreviation.
6. The symbol $\simeq$ is used to express equality with undefinition, i.e., $X \simeq Y$ if X and Y are both defined and equal, or both undefined.
7. **Vectors** are denoted with boldface letters.

The following less-familiar terms and concepts where used.

1. A closed n**-cube** with center $\mathbf{x}$ and side r is the subset of $\mathbb{R}^n$ given by

$$\{\mathbf{y} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{y}\|_\infty \leqslant \frac{r}{2}\}.$$

2. For a (possibly partial) function $F : \mathbb{N}^m \to \mathbb{N}^n$, a **real extension** of F is any function $f : \mathbb{R}^m \to \mathbb{R}^n$ such that

$$F(\mathbf{a}) = f(\mathbf{a}) \text{ for all } \mathbf{a} \in \mathbb{N}^m.$$

A similar convention is applied to mixed signatures (e.g., $F : \mathbb{R}^k \times \mathbb{N}^m \to \mathbb{N}^n$).

3. The **derivative** of a function $f : \mathbb{R}^m \to \mathbb{R}^n$ at a point $\mathbf{x}_0 \in \mathbb{R}^m$, if it exists, is the (bounded) linear map from $\mathbb{R}^m$ to $\mathbb{R}^n$, denoted $\mathrm{d}f(\mathbf{x}_0)$, such that

$$\lim_{\mathbf{x}\to\mathbf{x}_0} \frac{f(\mathbf{x}) - f(\mathbf{x}_0) - (\mathrm{d}f(\mathbf{x}_0))(\mathbf{x}-\mathbf{x}_0)}{\|\mathbf{x}-\mathbf{x}_0\|} = 0.$$

   Then, the **derivative** of f is the function $\mathrm{d}f : \mathbb{R}^m \to \mathcal{L}(\mathbb{R}^m, \mathbb{R}^n)$ (where $\mathcal{L}(\mathbb{R}^m, \mathbb{R}^n)$ denotes the set of (necessarily bounded) linear maps from $\mathbb{R}^m$ to $\mathbb{R}^n$) which gives $\mathrm{d}f(\mathbf{x})$ at each point $\mathbf{x}$. Then f is called **differentiable** in $A \subseteq \mathbb{R}^m$ if $\mathrm{d}f$ is defined in A; f is called **continuously differentiable** in A if $\mathrm{d}f$ is defined and continuous in A.

References

William Aspray, Charles Gillispie, Frederik Nebeker, and Albert Tucker. The Princeton mathematics community in the 1930s. Published online at `http://www.princeton.edu/mudd/math/`, 1985.

José Luis Balcázar and Montserrat Hermo. The structure of logarithmic advice complexity classes. *Theoretical Computer Science*, 207(1):217–244, 1998.

José Luis Balcázar, Josep Días, and Joaquim Gabarró. *Structural Complexity I.* Springer-Verlag, 1988.

José Luis Balcázar, Ricard Gavaldà, and Hava Siegelmann. Computational power of neural networks: a Kolmogorov complexity characterization. *IEEE Transactions on Information Theory*, 43(3):1175–1183, 1997.

Edwin Beggs and John Tucker. Computations via experiments with kinematic systems. Research report 4.04, University of Wales Swansea, 2004.

Edwin Beggs and John Tucker. Embedding infinitely parallel computation in newtonian kinematics. *Applied Mathematics and Computation*, 178(1):25–43, 2006.

Edwin Beggs and John Tucker. Can Newtonian systems, bounded in space, time, mass and energy compute all functions? *Theoretical Computer Science*, 371(1):4–19, 2007a.

Edwin Beggs and John Tucker. Experimental computation of real numbers by Newtonian machines. *Proceedings of the Royal Society*, 463(2082):1541–1561, 2007b.

Edwin Beggs, José Félix Costa, Bruno Loff, and John Tucker. The computational complexity of the analog-digital scatter machine, 2007. To be completed.

Stephen Bellantoni. *Predicative Recursion and Computational Complexity.* PhD thesis, University of Toronto, 1992.

Amir Ben-Amram. The Church-Turing thesis and its look-alikes. *ACM SIGACT News*, 36(3): 113–114, 2005.

Udi Boker and Nachum Dershowitz. How to compare the power of computational models. In Barry Cooper, Benedikt Löwe, and Leen Torenvliet, editors, *New computational paradigms (CiE 2005)*, volume 3526 of *Lecture Notes in Computer Science*, pages 54–64. Springer–Verlag, 2005.

Udi Boker and Nachum Dershowitz. Abstract effective models. *Electronic Notes in Theoretical Computer Science*, 135(3):15–23, 2006.

Olivier Bournez and Emmanuel Hainry. Real recursive functions and real extensions of recursive functions. In Maurice Margenstern, editor, *Machines, Computations and Universality (MCU 2004)*, volume 3354 of *Lecture Notes in Computer Science*, pages 116–127. Springer–Verlag, 2004.

Olivier Bournez and Emmanuel Hainry. Elementarily computable functions over the real numbers and $\mathbb{R}$-sub-recursive functions. *Theoretical Computer Science*, 348(2–3):130–147, 2005.

Olivier Bournez and Emmanuel Hainry. Recursive analysis characterized as a class of real recursive functions. *Fundamenta Informaticae*, 74(4):409–433, 2006.

Selman Bringsjord and Michael Zenzen. *Superminds.* Springer, 2003.

Vannevar Bush and Harold Hazen. The differential analyser. *Journal of the Franklin Institute*, 212(4):447–488, 1931.

Manuel Campagnolo. The complexity of real recursive functions. In Cristian Calude, Michael Dinneen, and Ferdinand Peper, editors, *Unconventional Models of Computation (UMC 2002)*, volume 2509 of *Lecture Notes in Computer Science*, pages 1–14. Springer–Verlag, 2002.

Manuel Campagnolo and Kerry Ojakian. The elementary computable functions over the real numbers: Applying two new techniques. *Archive for Mathematical Logic.*, 2006. To appear.

Manuel Campagnolo, Cris Moore, and José Félix Costa. Iteration, inequalities, and differentiability in analog computers. *Journal of Complexity*, 16(4):642–660, 2000.

Manuel Campagnolo, Cris Moore, and José Félix Costa. An analog characterization of the Grzegorczyk hierarchy. *Journal of Complexity*, 18(4):977–1000, 2002.

Peter Clote. *Handbook of Computability Theory*, volume 140 of *Studies in Logic and the Foundations of Mathematics*, chapter 17 – Computation Models and Function Algebras, pages 589–681. Elsevier, 1999.

Jack Copeland. The Church–Turing thesis. In Edward Zalta, editor, *The Stanford Enciclopedia of Phylosophy.* Published online at `http://plato.stanford.edu/archives/fall2002/entries/church-turing/`, 2002.

José Félix Costa. Physics and computation. 2007. Accepted for publication.

José Félix Costa, Bruno Loff, and Jerzy Mycka. The new promise of analog computation. In *Computation and Logic in the Real World (CiE 2007)*, volume 4497 of *Lecture Notes in Computer Science*, pages 189–195. Springer–Verlag, 2007.

Martin Davis. The myth of hypercomputation. In Christof Teuscher, editor, *Alan Turing: the life and legacy of a great thinker*, pages 195–212. Springer, 2006a.

Martin Davis. Why there is no such discipline as hypercomputation. *Applied Mathematics and Computation*, 178(1):4–7, 2006b.

Rodney Driver. *Introduction to Ordinary Differential Equations.* Harper and Row, 1978.

Gábor Etesi and István Németi. Non-Turing computations via Malament–Hogarth space-times. *International Journal of Theoretical Physics*, 41(2):341–370, 2002.

Edward Fredkin and Tommaso Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21:219–253, 1982.

Antony Galton. The Church–Turing thesis: still valid after all these years? *Applied Mathematics and Computation*, 178(1):93–102, 2006.

Antony Galton. The Church–Turing thesis: its nature and status. In Peter Millican and Andy Clark, editors, *Machines and Thought: The Legacy of Alan Turing*, volume 1, pages 137–164. Oxford University Press, 1996.

Robin Gandy. Church's thesis and principles for mechanisms. In Jon Barwise, H. Jerome Keisler, and Kenneth Kunen, editors, *The Kleene Symposium*, pages 55–111. North-Holland, 1980.

Robin Gandy. The confluence of ideas in 1936. In Rolf Herken, editor, *The universal Turing machine: a half-century survey*, pages 55–111. Oxford University Press, 1988.

Phil Gibbs. The small scale structure of space-time. Archive preprint `http://lanl.arxiv.org/abs/hep-th/9506171v2`, 1995.

M. D. Gladstone. Simplifications of the recursion scheme. *Journal of Symbolic Logic*, 36(4): 653–665, 1971.

Luís Mendes Gomes. *Applications of Real Recursive Infinite Limits.* PhD thesis, University of Azores, 2006.

Luís Mendes Gomes and José Félix Costa. Hybrid finite computation. In Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *Computation and Logic in the Real World (CiE 2007)*, pages 178–185, 2007.

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

Daniel Graça, Manuel Campagnolo, and Jorge Buescu. Robust simulations of Turing machines with analytic maps and flows. In *New Computational Paradigms, CiE 2005*, volume 3526 of *Lecture Notes in Computer Science*, pages 169–179. Springer–Verlag, 2005.

Amit Hagar and Alexandre Korolev. Quantum hypercomputability? *Minds and Machines*, 16 (1):87–93, 2006.

Andrew Hodges. Can quantum computing solve classically unsolvable problems? Archive preprint `http://arxiv.org/quant-ph/0512248`, 2005.

Andrew Hodges. Did Church and Turing have a thesis about machines? In Adam Olszewski, Jan Wolenski, and Robert Janusz, editors, *Church's thesis after 70 years*, pages 242–252. Ontos Verlag, 2006.

Andrew Hodges. The professors and the brainstorms. Published online at `http://www.turing.org.uk/philosophy/sciam.html`, 1999.

Mark Hogarth. Non-Turing computers and non-Turing computability. *Proceedings of the Biennial Meeting of the Philosophy of Science Association*, 1:126–138, 1994.

Richard Karp and Richard Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the twelfth annual ACM symposium on Theory of computing (STOC 1980)*, pages 302–309. ACM Press, 1980.

Akitoshi Kawamura. Differential recursion and differentially algebraic functions. Archive preprint `http://arxiv.org/abs/0704.0301`, 2007. Revised version of *Real primitive recursive functions and differential algebraicity*, presented at *Computability in Europe 2006, Logical approaches to computational barriers.*

Tien Kieu. Reply to "the quantum algorithm of Kieu does not solve the Hilbert's tenth problem". Archive preprint `http://arxiv.org/abs/quant-ph/0111020`, 2001.

Tien Kieu. Quantum hypercomputation. *Minds and Machines*, 12(4):541–561, 2002.

Tien Kieu. Quantum algorithms for the Hilbert's tenth problem. *International Journal of Theoretical Physics*, 42:1461–1478, 2003.

Tien Kieu. Reply to Andrew Hodges. Archive preprint `http://arxiv.org/abs/quant-ph/0602214`, 2006a.

Tien Kieu. On the identification of the ground state based on occupation probabilities: an investigation of Smith's apparent counterexamples. Archive preprint `http://arxiv.org/abs/quant-ph/0602145`, 2006b.

Georg Kreisel. Churh's thesis and the ideal of informal rigour. *Notre Dame Journal of Formal Logic*, 28(4):499–519, 1987.

Erwin Kreyszig. *Introductory functional analysis with applications.* John Wiley & Sons, 1989.

Bruno Loff. A functional characterisation of the analytical hierarchy. In Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *Computability in Europe 2007: Computation and Logic in the Real World*, pages 247–256, 2007a.

Bruno Loff. On two variants of the Church–Turing thesis. Technical report, Instituto Superior Técnico, Technical University of Lisbon, 2007b.

Bruno Loff and José Félix Costa. Five views of hypercomputation. *International Journal of Unconventional Computing*, 2007. Accepted for publication.

Bruno Loff, José Félix Costa, and Jerzy Mycka. Computability on reals, infinite limits and differential equations. *Applied Mathematics and Computation*, 191:353–371, 2007.

Albert Meyer and Dennis Ritchie. The complexity of loop programs. In *Proceedings of the 22nd National Conference*, pages 465–470. Thompson Book Company, 1967.

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

Jerzy Mycka. μ-recursion and infinite limits. *Theoretical Computer Science*, 302:123–133, 2003a.

Jerzy Mycka. Infinite limits and R-recursive functions. *Acta Cybernetica*, 16(1):83–91, 2003b.

Jerzy Mycka and José Félix Costa. Real recursive functions and their hierarchy. *Journal of Complexity*, 20(6):835–857, 2004.

Jerzy Mycka and José Félix Costa. The $P \neq NP$ conjecture in the context of real and complex analysis. *Journal of Complexity*, 22(2):287–303, 2006a.

Jerzy Mycka and José Félix Costa. The conjecture $P \neq NP$ presented by means of some classes of real functions. In Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John Tucker, editors, *Logical Approaches to Computational Barriers (CiE 2006)*, volume 3988 of *Lecture Notes in Computer Science*, pages 47–57. Springer–Verlag, 2006b.

Jerzy Mycka and José Félix Costa. A new conceptual framework for analog computation. *Theoretical Computer Science*, 374(1–3):277–290, 2007.

Jerzy Mycka and José Félix Costa. Undecidability over continuous-time. *Logic Journal of the IGPL, Oxford University Press*, 14:649 – 658, 2006c.

István Németi and Gyula Dávid. Relativistic computers and the Turing barrier. *Applied Mathematics and Computation*, 178(1):118–142, 2006.

Michael Nielsen. Computable functions, quantum measurements and quantum dynamics. *Physical Review Letters*, 79:2915–2918, 1997.

James Nyce. Nature's machine: mimesis, the analog computer and the rhetoric of technology. In Ray Paton, editor, *Computing with biological metaphors*. Chapman & Hall, 1994.

Piergiorgio Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and the Foundations of Mathematics*. North Holland, 1989.

Toby Ord. Hypercomputation: computing more than the Turing machine. Honor's thesis, University of Melbourne, 2002.

Toby Ord. The many forms of hypercomputation. *Applied mathematics and computation*, 178 (1):143–153, 2006.

Roger Penrose. *The Emperor's New Mind*. Oxford University Press, 1989.

Roger Penrose. *Shadows of the Mind*. Oxford University Press, 1994.

Marian Pour-El and Ian Richards. The wave equation with computable initial data such that its unique solution is not computable. *Advances in Mathematics*, 39(4):215–239, 1981.

Gian-Carlo Rota. The pernicious influence of Mathematics upon Philosophy. In Reuben Hersh, editor, *18 unconventional essays on the nature of Mathematics*, pages 220–230. Springer, 2006.

Oron Shagrir and Itamar Pitowsky. Physical hypercomputation and the ChurchTuring thesis. *Minds and Machines*, 13(1):87–101, 2003.

Andrés Sicard and Mario Vélez. The Church–Turing thesis. Published online at `http://citeseer.ist.psu.edu/sicard99churchturing.html`, 1999.

Wilfried Sieg. Gödel on computability. *Philosophia Mathematica*, 14(2):189–207, 2006.

Wilfried Sieg. Church without dogma – axioms for computability. Submitted, 2007.

Hava Siegelmann. *Neural Networks and Analog Computation: Beyond the Turing Limit*. Birkhäuser, 1999.

Warren Smith. Church's thesis meets the N-body problem. *Applied Mathematics and Computation*, 178(1):154–183, 2006a.

Warren Smith. Three counterexamples refuting Kieu's plan for "quantum adiabatic hypercomputation"; and some uncomputable quantum mechanical tasks. *Applied Mathematics and Computation*, 178(1):184–193, 2006b.

Warren Smith. Chuch's thesis meets quantum mechanics. Published online at `http://www.math.temple.edu/~wds/homepage/churchq.ps`, 1999.

Gerald Teschl. Ordinary differential equations and dynamical systems. Available at `http://www.mat.univie.ac.at/~gerald/ftp/book-ode/ode.pdf`, 2007.

William Thurston. On proof and progress in Mathematics. In Reuben Hersh, editor, *18 unconventional essays on the nature of Mathematics*, pages 37–55. Springer, 2006.

Boris Tsirelson. The quantum algorithm of Kieu does not solve the Hilbert's tenth problem. Archive preprint `http://arxiv.org/abs/quant-ph/0111009`, 2001.

Alan Turing. On computable numbers. In *Proceedings of the London Mathematical Society*, volume 42 of *the second series*, pages 230–265, 1936.

John von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In *Automata Studies*, pages 43–98. Princeton University Press, 1956.

Wolfgang Walter. *Ordinary Differential Equations*. Springer-Verlag, 1998.

Klaus Weihrauch and Ning Zhong. Is wave propagation computable or can wave computers beat the Turing machine? *Proceedings of the London Mathematical Society*, 85(2):312–332, 2002.

António Weinholtz. *Equações Diferenciais — Uma Introdução*. Universidade de Lisboa, Faculdade de Ciências, Departamento de Matemática, 2000.

Philip Welch. The extent of computation in Malament-Hogarth spacetimes. Archive preprint `http://arxiv.org/abs/gr-qc/0609035`, 2006.

Michael Williams. *A History of Computing Technology*. IEEE Computer Society Press, 2nd edition, 1997.

Zhihong Xia. The existence of noncollision singularities in Newtonian systems. *The Annals of Mathematics, Second Series*, 135(3):411–468, 1992.