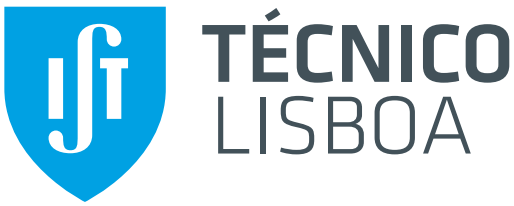

$BPP//\log\star$

Random Walks as Oracles

Pedro Cortez Rodrigues

Thesis to obtain the Master of Science Degree in

Mathematics and Applications

Supervisor: Prof. José Félix Gomes da Costa

Examination Committee

Chairperson: Prof. Maria Cristina Sales Viana Serôdio Sernadas
Supervisor: Prof. José Félix Gomes da Costa
Members of the Committee: Prof. André Nuno Carvalho Souto

March 2015

Acknowledgments

I would like to thank my advisor, José Félix Costa, for all the guidance, encouragement and patience, during the elaboration of this thesis. I also want to thank him for proposing me this subject and for making every effort to help me relating it with all the areas of Mathematics that I like, introducing me to scientific research.

I would also like to thank my family and friends for their support.

Resumo

O emparelhamento de máquinas de Turing com oráculos é uma das técnicas utilizadas para aumentar o poder computacional das máquinas ou para melhorar a sua eficiência.

Nesta dissertação é estudado o poder computacional de máquinas de Turing analógico-digitais, ou seja, máquinas de Turing acopladas com experiências físicas, que são tratadas como oráculos. Consideramos um oráculo físico genérico e abstracto, obedecendo a certos axiomas, analisamos o seu poder computacional e utilizamo-lo para abstrair algum do trabalho já feito relativamente a máquinas de Turing analógico-digitais. Mostramos também que estes oráculos permitem generalizar a máquina de Turing probabilística, comparando a máquina analógico-digital obtida com a máquina probabilística que tem números reais como probabilidade de transição.

Depois, concretizamos o nosso oráculo genérico com a experiência do passeio aleatório, apresentando um interessante estudo sobre a experiência. De seguida, limitamos o número de chamadas ao oráculo, com o objectivo de estudar a estrutura interna de $BPP//\log\star$. Analisamos o poder computacional das máquinas de passeio aleatório com certos limites no número de chamadas ao oráculo. Provamos um importante resultado sobre classes de complexidade não-uniforme para o caso probabilístico. Por fim, apresentamos uma hierarquia de classes em $BPP//\log\star$, correspondente às máquinas de Turing analógico-digitais com o número de chamadas ao oráculo limitado por certas funções do tamanho do input.

Palavras-chave: Máquinas de Turing. Oráculos físicos. Complexidade não-uniforme. Passeios aleatórios. $BPP//\log\star$.

Abstract

Coupling Turing machines with oracles is one of the techniques used to boost the computational power, or only the efficiency, of Turing machines.

In this thesis, we study the computational power of analogue-digital Turing machines, i.e., Turing machines coupled with physical experiments, which are treated as oracles. We consider a generic and abstract physical oracle, satisfying certain axioms, we analyse its computational power and we use it to abstract some of the work already done for analogue-digital Turing machines. We also prove that these oracles allow us to generalize the probabilistic Turing machine, comparing the obtained analogue-digital machine with the probabilistic machine that has real numbers as probability of transition.

Then, we concretize our generic oracle with the random walk experiment, presenting an interesting review of the experiment. After that, we bound the number of calls to the oracle, in order to study the inner structure of $BPP//\log\star$. We analyse the computational power of random walk machines with certain bounds on the number of oracle calls. We also prove an important result about non-uniform complexity classes for the probabilistic case. Finally, we present a hierarchy of complexity classes in $BPP//\log\star$, corresponding to the analogue-digital Turing machines with the number of oracle calls bounded by certain functions on the size of the input.

Keywords: Turing machines. Physical oracles. Non-uniform complexity. Random walks. $BPP//\log\star$.

Contents

List of Figures

List of Tables

List of Abbreviations

AD machine	Analogue-digital Turing machine
BB machine	Broken balance Turing machine
BBE	Broken balance experiment
BS machine	Balance scale Turing machine
BSE	Balance scale experiment
$\mathcal{C}_3$	A set of Cantor numbers
C_i	The i-th Catalan number
χ_A	The characteristic function of A
$d(D, D')$	The distance between probabilistic trees (T, D) and (T, D'), with the same query tree T and assignments D and D', respectively
$\mathcal{F}$	A class of advice functions
$\mathcal{F}\star$	A class of prefix advice functions
$\mathcal{F} \prec \mathcal{G}$	There exists $g \in \mathcal{G}$ such that, for every $f \in \mathcal{F}$, $\|f\| \prec \|g\|$
$f_t(m, \beta)$	The largest possible difference in the probability of acceptance for two probabilistic trees with the same t-ary query tree of height m, such that their distance is bounded by β
λ	The empty word
$P_A(T, D)$	The acceptance probability of the probabilistic tree (T, D)
$P_R(T, D)$	The rejection probability of the probabilistic tree (T, D)
$\rho(T)$	The set of all possible assignments of probabilities to the edges of T
$f \prec g$	$f \in o(g)$
RWE	Random walk experiment
RW machine	Random walk Turing machine
$\mathcal{T}_m^t$	The set of all t-ary probabilistic trees of height m
Σ	Alphabet
Σ^*	The set of all (finite) words over the alphabet Σ

Σ^ω	The set of all infinite sequences over the alphabet Σ
$z\downarrow_n$	The pruning or the padding of the word z until getting n bits
$\|w\|$	Size of the word w

Glossary

Advice function	A total function $f : \mathbb{N} \to \Sigma^*$.
Bisection method	A root-finding method that repeatedly bisects an interval and then selects the subinterval in which a root must lie for further processing. In other contexts, this method is called binary search method.
C^n	Given a function $f : D \subseteq \mathbb{R} \to \mathbb{R}$ and a number $n \in \mathbb{N}$, $f \in C^n$ if f and all of its derivatives $f', ..., f^{(n)}$ are continuous.
Dyadic rational	A number of the form $n/2^k$ where n is an integer and k is a natural number. Every dyadic rational has finite binary expansion.
Generalized probabilistic Turing machine	Probabilistic Turing machine such that the probability of transition is a real number.
Mean value theorem	If a function f is continuous on the closed interval $[a, b]$ and differentiable on the open interval (a, b), then there exists at least one point $c \in (a, b)$ such that $f(b) - f(a) = (b - a)f'(c)$.
Negative binomial distribution	The distribution of the number of trials until we have k successes, where each trial has a probability of success p.
$o(f)$	The class of functions g such that, for all $c \in \mathbb{R}_{>0}$ there exists $p \in \mathbb{N}$ such that, for all $n \geq p$, $g(n) < cf(n)$.
$O(f)$	The class of functions g such that there exists $c \in \mathbb{R}_{>0}$, $p \in \mathbb{N}$ such that, for all $n \geq p$, $g(n) < cf(n)$.
Oracle	A set used by the Turing machine, that can be consulted in one time step, querying about set membership of a word.
Prefix function	A function f such that, for every n, $f(n)$ is a prefix of $f(n + 1)$.

Preimage	Given $f : X \to Y$, the preimage of $y \in Y$ is $f^{-1}(y) = \{x \mid f(x) = y\}$.
Rolle's theorem	If a function f is continuous on the closed interval $[a, b]$ and differentiable on the open interval (a, b), then, if $f(a) = f(b)$, then there exists at least one point $c \in (a, b)$ such that $f'(c) = 0$.
Tally set	A language over an unary alphabet, namely $\{0\}$, i.e., a set of words of the form 0^k.
Time constructible	A function $f : \mathbb{N} \to \mathbb{N}$ is said to be time constructible if there exists a deterministic Turing machine M and a number $p \in \mathbb{N}$ such that, for any input word of size $n \geq p$, M halts after exactly $f(n)$ transitions.
Time schedule	A time constructible function in the size of the query that establishes the time that the machine must wait for the answer of the oracle.
Turing machine	Abstract computing device, introduced by Alan Turing, that manipulates symbols on tapes according to a set of rules, the transition function.
Weierstrass extreme value theorem	If a function f is continuous on the compact set $D \subset \mathbb{R}$, then f has a maximum and a minimum on D.

Chapter 1

Introduction

Coupling oracles to Turing machines is one of the methods used to boost the computational power or only to speed up the computations of standard Turing machines. Conventionally, an oracle is a set. However, in [7] and [9], Beggs et al. proposed that the classical oracle could be replaced by an analogue device. The main idea is to couple a Turing machine with a physical experiment. The Turing machine interacts with the oracle by means of the query tape, providing the device with the initial conditions needed to initialize the experiment (if any). This communication between the machine and the oracle, i.e., the whole process of transferring data to, and from, the analogue device is ruled by some protocol, that consumes time and may introduce some error in the system. There are three types of protocols that have been considered: infinite precision, unbounded precision and fixed precision.

As one can expect, the main difference between these analogue-digital Turing machines, or AD machines, and classical oracle machines is that, unlike the classical oracles, it takes time to consult a physical oracle, because a physical experiment is executed, so, the oracle consultation is not any more a single step computation, but it lasts a number of time steps that depend on the size of the query. Usually, a time schedule is defined, i.e., a time constructible function in the size of the query that establishes the time that the machine shall wait for the oracle computation to produce a result. Then, the finite control of the Turing machine changes to other state, according to the outcome of the oracle, and the computation proceeds.

In this work, we analyse the random walk experiment and construct a machine that uses it as an oracle, launching particles and receiving as result if they were, or not, absorbed during the scheduled time. We chose this experience, not to be just another analogue-digital machine such as the ones already analysed, but as an attempt to abstract all previously studied experiences, for the case of fixed precision. Leaving aside the Physics, there is no need to consider concrete results of physical science and we get an experiment from the field of Probability.

1.1 State of the art

John von Neumann, in [41], introduced stochastic computation, while exploring automata, stating that "mechanical devices as well as electrical ones are statistically subject to failure" and establishing the goal of finding a bound on the probability of error, "such that useful results can still be achieved". Working with analogue-digital machines, we have to consider this possible error of analogue devices, using stochastic and probabilistic methods in order to achieve the desired performance.

Hava Siegelmann has already studied the non-uniform complexity class $BPP//\log\star$, presenting computation models with that computational power. In [39], she studies rational stochastic networks, i.e., neural networks with rational weights and real probabilities, which have the same computational power than the machines that we present in this dissertation and she uses probabilistic techniques to approximate the real probabilities, as we do in this work.

When talking about the class $BPP//\log\star$, or even about analogue-digital computation, it is usual to think of hypercomputation, i.e., computing beyond the Turing limit. As the machines we work with, hypercomputation models are often presented in theoretical contexts, without regard to the conditions for their physical realizability. Martin Davis defends that the hypercomputation subject does not exist. In [29], he states that the only way a machine can go beyond the Turing limit is being provided with non-computable information and in [30] he says that, even if a machine could compute beyond the Turing computable, we would not be able to certify that fact. Although, Younger et al. believe that is possible to physically realize Super-Turing machines and their project to do so is described in [42]. We consider that, in order to implement these models, it is needed that the nature follows some distribution, so that we can work with probabilistic results such as Chebyshev's inequality. Otherwise, we have no guarantee that there is convergence of the sequence of relative frequencies (see [31] for an interesting discussion on the existence of a limit for relative frequencies) and we may not be able to prove any bounds for the computational power of analogue-digital machines. However, in this dissertation, we are not interested in the physical realizability of the machines, but only in studying the limits of analogue-digital systems.

A particular kind of analogue-digital machine that has been studied is the one that uses as oracle the scatter experiment, first in its sharp version (see [7]) and later in its smooth form (see [2]).

The sharp scatter experiment, represented in Figure 1.1, aims to measure the position of a vertex, by shooting a particle, from a cannon, towards a sharp wedge. The particle is fired with a fixed velocity v, from a cannon in position z, and will collide with the wedge that has its vertex at position y. After colliding, the particle will be reflected, either to left or right, and captured by a detection box. Depending on the position of the vertex, y, and the position of the cannon, z, we consider the following cases: if $z < y$, the particle hits the left side of the wedge and goes to the left collecting box; if $z > y$, the particle hits the right side of the wedge and goes to the right collecting box; else, the particle is not detected in any of the two boxes. The wedge has triangular shape and its sides are at 45° to the line of the cannon, so that the particle fired by the cannon is reflected perpendicularly to the line of the cannon. Therefore, in the case that $z \neq y$ the particle takes no more than $\frac{2d}{v}$ seconds to be detected in each box and, so the experimental time is bounded by a constant. In fact, since the collecting boxes are placed at equal

distance from the vertex, no matter the position of the cannon, the experiment takes always the same time. So, we can manipulate d and v in order to ensure that the experiment takes exactly the time of one transition of the Turing machine.

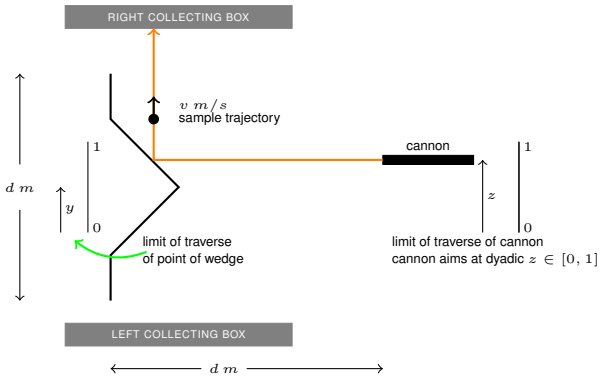

Figure 1.1: The sharp scatter experiment.

The smooth scatter experiment is a variation of the sharp scatter experiment, with a different experimental apparatus. While in the sharp case we consider a vertex in a sharp wedge, in the smooth case we are interested in measuring a vertex in a smooth wedge, being the shape of the wedge given by a function that satisfies some suitable conditions. The main difference between the sharp and the smooth case is the experimental time. In the sharp case, we have seen that the time is constant, however, in the smooth case the experimental time is unbounded, since when the particle hits the wedge close to the vertex, the experimental time goes to infinity. The experimental apparatus of the smooth scatter experiment is represented in Figure 1.2.

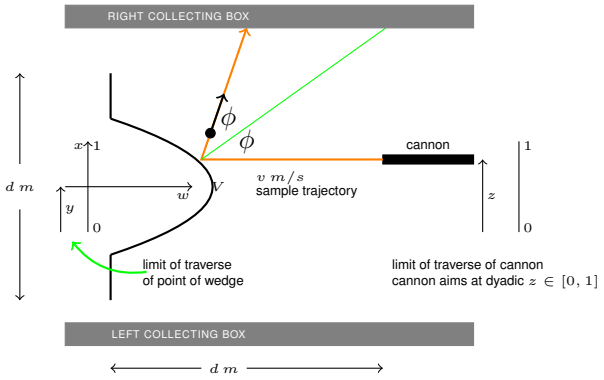

Figure 1.2: The smooth scatter experiment.

A Turing machine coupled with a sharp (or smooth) scatter experiment is called a sharp (or smooth) scatter machine. The scatter machine interacts with the scatter experiment using the query tape. To initialize the experiment, the Turing machine writes in the query tape the parameters for the experiment, namely the position of the cannon. After some time, the machine will make a transition to other state according to the outcome of the experiment ('left' or 'right'). As discussed before, in the smooth case, the experiment can take a lot of time, thus, to ensure that the machine does not wait unbounded time,

we will equip it with a time schedule, i.e., a time constructible function on the size of the query that determines the time that the machine has to wait for an answer from the oracle. If the Turing machine does not have an answer after the waiting time, we consider the outcome of the oracle to be 'timeout'.

For both cases, the communication between the Turing machine and the experiment is ruled by one of the following protocols:

- infinite precision: when z is read in the query tape and the cannon is set in position z;
- unbounded precision: when z is read in the query tape and the cannon is set in a position randomly and uniformly chosen in $(z - 2^{-|z|}, z + 2^{-|z|})$;
- fixed precision ε ($\varepsilon > 0$): when z is read in the query tape and the cannon is set in a position randomly and uniformly chosen in $(z - \varepsilon, z + \varepsilon)$.

This communication between the digital and the analogue devices also takes time, thus, in the sharp case, although the experimental time is constant, we can consider polynomial or exponential protocols, as we do in the smooth case. All things considered, the obtained results in terms of lower and upper bounds for the complexity classes computed by these machines, for all precision protocols, are described in the tables below: Table 1.1 for sharp scatter machine results and Table 1.2 for smooth scatter machine results. These results were presented in [9, 2].

	Infinite		Unbounded	Fixed
Lower Bound	$P/poly$ Polytime protocol	$P/\log\star$ Exptime protocol	$P/poly$ Polytime protocol	$BPP//\log\star$
Upper Bound	$P/poly$ Polytime protocol	$P/\log\star$ Exptime protocol	$P/poly$ Polytime protocol	$BPP//\log\star$

Table 1.1: Complexity classes for the sharp scatter machine.

We have seen that the sharp scatter experiment has constant time, so, it might seem strange to the reader that the class $P/\log\star$ is considered in Table 1.1. Actually, although the experimental time is constant, we can force the communication protocol between the Turing machine and the analogue device to be exponential. Therefore, this class appears artificially.

	Infinite	Unbounded	Fixed
Lower Bound	$P/\log\star$	$BPP//\log\star$	$BPP//\log\star$
Upper Bound	$P/\log\star$	$BPP//\log^2\star$ Exponential schedule	$BPP//\log^2\star$ Exponential schedule
Upper Bound Explicit Time	—	$BPP//\log\star$ Exponential schedule	$BPP//\log\star$ Exponential schedule

Table 1.2: Complexity classes for the smooth scatter machine.

In [13, 11], Beggs et al. formulated a conjecture, the BCT conjecture, that states that for all physical theories $\mathcal{T}$, for all "realistic" physical measurements based upon $\mathcal{T}$, the time for the physical experiment is at least exponential in the size of precision. This means that, in order to estimate the n-th bit of the value that we are measuring, the time needed is at least exponential in n.

The BCT conjecture leads to another conjecture, the analogue-digital Church-Turing Thesis, discussed in [15], that states that no possible abstract analogue-digital device can have more computational capabilities in polynomial time than $BPP//\log\star$.

The various measurement experiments can be divided into three non empty sets: two-sided, threshold and vanishing. Moreover, by BCT conjecture, all the experiments have exponential time[1]. Thus, the reason why so many different experiments have been considered is to verify or try to refute BCT conjecture.

1.2 Our work

In Chapter 2, we consider an abstract generic oracle, with two possible outcomes, e.g. 'yes' or 'no', and such that no parameters are needed to be given in order to initialize the experiment. We prove lower and upper bounds on its computational power. We denote the probability of getting the outcome 'yes' by $F(\sigma)$, where σ is the unknown parameter that characterizes the experiment.

We proved the following lower bound theorem:

Theorem 1.1. *Every set in $BPP//\log\star$ is decidable by an AD machine with 'yes' probability $F(\sigma)$ in polynomial time.*

As we are trying to establish as lower bound the non-uniform complexity class $BPP//\log\star$, we must find a way to encode the advice information into the parameter σ. We use an encoding based on the Cantor set $\mathcal{C}_3$, already used in [7] and [10]:

$$\mathcal{C}_3 = \left\{ x \in \mathbb{R} : x = \sum_{k=1}^{\infty} x_k 2^{-3k}, \text{ where } x_k \in \{1, 2, 4\} \right\}.$$

Let $A \in BPP//\log\star$ and let $\mathcal{M}$ be a probabilistic Turing machine with advice $f \in \log\star$ that decides A in polynomial time with error probability bounded by γ. Let $y(f)$ be the encoding of the advice function f. Then, we need to show that the AD machine with parameter $\sigma = y(f)$ can simulate, in polynomial time, the behaviour of $\mathcal{M}$. So, our AD machine should be able to read the advice and to simulate a fair coin (for the probabilistic behaviour).

To read the advice, we make multiple calls to the oracle, doing a frequency analysis of the results, so that we can estimate the probability of getting the outcome 'yes'. Then, combining that result with the expression for that probability, $F(\sigma)$, we use some numerical methods to approximate σ. We obtain that the machine can read a number of bits of σ logarithmic on the size of the input, in polynomial time.

[1] Although the sharp scatter experiment has constant time, it is not "realistic", since its shape is not a continuously differentiable function.

To prove that our machine can simulate a fair coin, we show that it can be seen as a biased coin and that, given a biased coin, we can produce a sequence of fair coin tosses in linear time on the size of the sequence.

As an upper bound for the classes computed by AD machines with 'yes' probability $F(\sigma)$, we also want a non-uniform complexity class. More, we want to make lower and upper bounds coincide. So, we tried to set $BPP//\log\star$ as an upper bound.

Theorem 1.2. *Every set decided by an AD machine with unknown parameter σ and with 'yes' probability $F(\sigma)$ in polynomial time is in $BPP//\log\star$.*

To prove this result, we must use and explore some results of probabilistic trees. Our AD machines are composed by a deterministic Turing machine and a stochastic oracle. Thus, the computations are deterministic, except for oracle consultations. Therefore, the computations of an AD machine with 'yes' probability $F(\sigma)$ can be represented by a binary tree, where each node stands for one run of the experiment and its left and right sub-trees correspond to the two possible outcomes of the experiment. Since the oracle is stochastic, each branch has a probability associated with it (in this case, since both σ and the time schedule are fixed for the whole computation, all the left branches have equal probability, F(σ), while all the right branches have probability $1 - F(\sigma)$). We use probabilistic trees to show that a small variation in the probability of 'yes' does not affect the decision of the machine. Then, we conclude that a number of bits of the probability of 'yes' logarithmic in the size of the input suffices to preserve the decision of the machine (with a small variation of the error). So, we consider a probabilistic machine $\mathcal{M}$ and we set that number of bits as the advice function. A probabilistic machine can behave has a deterministic machine, thus, we only need to show that, using the advice, it can simulate the calls to the oracle. We can do this using the fair coin of the probabilistic machine, using fair coin tosses to simulate a biased coin with heads probability equal to the probability of 'yes'.

With these proposed lower and upper bounds, we get the following result:

Corollary 1.3. *The class of sets which are decidable in polynomial time by AD machines with unknown parameter σ and with 'yes' probability $F(\sigma)$ is exactly $BPP//\log\star$.*

Also in Chapter 2, we use these results to prove the lower bound theorems for the broken balance experiment (see [10, 37]) and for the balance scale experiment (see [11, 37]).

Then, we generalize the probabilistic machine with the generalized probabilistic machine which is a probabilistic machine such that the transition probability p take a real value in $(0, 1)$. Showing that generalized probabilistic machines have the same computational power than our AD machines, we prove the following theorem:

Theorem 1.4. *The class of sets which are decidable in polynomial time by generalized probabilistic Turing machines with bounded error probability is exactly $BPP//\log\star$.*

In Chapter 3, we concretize our AD machine, presenting an experiment in the desired conditions, the random walk experiment (RWE).

Consider the experiment of having a particle moving along an axis. The particle is sent from position $x = 0$ to position $x = 1$. Then, at each positive integer coordinate, the particle moves right, with probability σ, or left, with probability $1 - \sigma$. If the particle ever returns to its initial position, $x = 0$, it is absorbed (it remains there). In this process, the particle takes steps of one unit, at time intervals also of one unit, thus, our experiment is a discrete unidimensional random walk with one absorbing barrier (see [40]).

We are interested in the probability that the particle is absorbed. We do as in [34]. Let p_i be the probability of absorption (at $x = 0$) when the particle starts at $x = i$. In our model, the particle is launched from $x = 0$ but it only starts its random walk at $x = 1$, so, we want to know the value of p_1. It is easy to see that

$$p_1 = (1 - \sigma) + \sigma p_2.$$

From $x = 2$, to be absorbed, the particle must initially move from $x = 2$ to $x = 1$ (not necessarily in one step), and then from $x = 1$ to $x = 0$ (again, not necessarily in one step). Both movements are made, independently, with probability p_1, thus, p_2 is just p_1^2. More generally, we have $p_k = p_1^k$. Therefore, the equation for the unidimensional random walk with absorption at $x = 0$ is given by

$$p_1 = (1 - \sigma) + \sigma p_1^2,$$

and we have a quadratic equation in p_1, with solutions $p_1 = 1$ and $p_1 = \frac{1-\sigma}{\sigma}$. For $\sigma = \frac{1}{2}$, the solutions coincide and $p_1 = 1$. For $\sigma < \frac{1}{2}$, the second solution is impossible, because $\frac{1-\sigma}{\sigma} > 1$, so, we must have $p_1 = 1$. For $\sigma = 1$, the particle always moves to the right, so $p_1 = 0$. Thus, for the sake of continuity of p_1, for $\sigma > \frac{1}{2}$, we must choose $p_1 = \frac{1-\sigma}{\sigma}$. Consequently, we get

$$p_1 = \begin{cases} 1 & \text{if } \sigma \leq \frac{1}{2} \\ \frac{1-\sigma}{\sigma} & \text{if } \sigma > \frac{1}{2} \end{cases}.$$

So, if $\sigma \leq \frac{1}{2}$, with probability 1 the particle always returns, but the number of steps is unbounded.

Now, let us consider a Turing machine coupled with a random walk experiment. This machine was introduced in [26]. To use the RWE as an oracle, we admit that the probability that the particle moves forward, σ, encodes some advice.

Unlike scatter machines, the RWE does not need any parameters to be initialized, so, the Turing machine does not provide the oracle with any dyadic rational, it just "pulls the trigger" to start the experiment, i.e., the query corresponds to the impulse given to the particle at position $x = 0$ so it moves to $x = 1$. As we saw before, for every $\sigma \in (0, 1)$ the time that the particle takes to return is unbounded, thus, we need to provide a time schedule. If the particle is absorbed during the time schedule, the finite control of the Turing machine changes to the 'yes' state, otherwise, the finite control changes to the 'no' state. Defining the time schedule is an important and complex matter. For a fixed $\sigma \in (0, \frac{1}{2})$ (in this interval we know that the particle always returns), depending on the time schedule being too small or too long, we can have always the same outcome ('yes' or 'no'). For now, let us consider that the time

schedule is constant.

Making an analogy between paths of the random walk and the well formed sequences of parentheses, and using some results about a generalization of Catalan numbers, presented in [19], we obtain that the probability that the particle is absorbed during the scheduled time T is given by

$$\sum_{\substack{t=1 \\ t\ odd}}^{T-1} \frac{1}{t}\binom{t}{\frac{t+1}{2}}(1-\sigma)^{\frac{t+1}{2}}\sigma^{\frac{t+1}{2}-1} .$$

This is the probability of getting the outcome 'yes' from the oracle.

Showing that the random walk machine respects the conditions of the AD machines from Chapter 2, we obtain the following result:

Corollary 1.5. *The class of sets which are decidable in polynomial time by deterministic random walk machines is exactly* $BPP//\log\star$.

Also in Chapter 3, we bound the number of calls to the oracle, trying to study the inner structure of $BPP//\log\star$ and find a hierarchy of complexity classes in $BPP//\log\star$. With that goal in mind, we define the iterated logarithmic functions $\log^{(k)}(n)$:

- $\log^{(0)}(n) = n$;
- $\log^{(k+1)}(n) = \log(\log^{(k)}(n))$.

Denoting by $\log^{(k)}$ the class of advice functions f such that $|f(n)| \in O(\log^{(k)}(n))$, we use techniques similar to the ones used in Chapter 2 to prove the following:

Theorem 1.6. *The class of sets which are decidable in polynomial time by deterministic random walk machines that can make up to* $O(\log^{(k)}(n))$ *calls to the oracle, where* n *is the input size, is exactly* $BPP//\log^{(k+1)}\star$.

Then, we present various definitions of concepts related to non-uniform complexity classes, in order to prove the important result:

Theorem 1.7. *If* $\mathcal{F}$ *and* $\mathcal{G}$ *are two classes of reasonable sublinear advice functions such that* $\mathcal{F} \prec \mathcal{G}$, *then* $BPP//\mathcal{F} \subsetneq BPP//\mathcal{G}$.

Finally, we present the hierarchy that we found in $BPP//\log\star$.

Chapter 2

Stochastic oracles: an abstract model

As explained before, a lot of work has been done in coupling oracles with Turing machines, in order to exceed the computational power of the Turing machine.

In this chapter, we will consider a Turing machine coupled with a generic stochastic oracle[1], with two possible outcomes, 'yes' or 'no', such that: the query to the oracle consists in a simple trigger to start the experiment, i.e., no parameters are given to the analogue device; if the query time is less than the scheduled time, then the finite control of the Turing machine changes to the 'yes' state; otherwise the finite control changes to the 'no' state.

Let $\mathcal{M}$ be an AD Turing machine coupled with a physical oracle in the above conditions. For every time schedule T and unknown parameter σ (associated with the physical experiment), the probability of 'yes' in one oracle call is $F(\sigma, T)$.

During the following sections, we will consider a fixed time schedule and we will only be concerned about F as a function of σ, thus, we may write $F(\sigma)$, instead of $F(\sigma, T)$. For convenience, we assume that $F \in C^1$ (referring to F as a function of σ) and $F'(\sigma) \neq 0$, for every $\sigma \in [0, 1]$. Let $\nu > 0$ be a lower bound for $|F'|$ in $[0, 1]$. We know that this value exists, due to Weierstrass extreme value theorem, since $|F'|$ is a continuous function on a closed interval. Suppose also that we can compute n bits of F in time $O(2^n)$.[2]

In this chapter, we study the computational power of these machines, obtaining the non-uniform class $BPP//\log\star$. We also illustrate the application of these results to the AD machines that use as oracle the balance scale experiment and the broken balance experiment (see [10]). In Section 2.8, we study these machines as a generalization of probabilistic Turing machines, what might seem intuitive, since, if $F(\sigma) = \sigma$, these machines behave as Turing machines with the ability of tossing biased coins with probability of heads σ.

[1] An abstract model of a physical oracle with a distribution that assigns probabilities to its outcomes, but without specifying neither that distribution, nor any concrete physical properties of the experiment.

[2] This asymptotic time is not the time that the machine takes to consult the oracle, but only the time that it takes to compute the function F that gives the probability of outcome 'yes'.

2.1 Encoding the advice

Since the goal is to prove that these machines compute some non-uniform complexity class, we must provide the machine with a method to access the advice function value for the size of the input. To do so, we encode the advice function into a real number that will be the unknown parameter of the oracle.

The set $\mathcal{C}_3$ of Cantor numbers is the set of every real number x such that

$$x = \sum_{k=1}^{\infty} x_k 2^{-3k}, \text{ for } x_k \in \{1, 2, 4\},$$

i.e., every real number whose binary expansion is composed by triplets of the form 001, 010, or 100.

This set has been used many times to encode real numbers, because it has the advantage that there is no need to distinguish between two very close numbers in order to get the most significant bit. Take as example the two binary expansions 10^ω and 01^ω. They represent the same number and, in the second case, to know the most significant bit, we must read the whole sequence. In $\mathcal{C}_3$, as every number has infinitely many 0's in its binary expansion, that technical problem does not exist. For example, in [39, 20, 32], different Cantor sets are used to encode real numbers, but in [7] and [10], Beggs et al. used the same codification that we present here. Let us study some properties of these numbers.

Proposition 2.1 (Beggs et al. [10])**.** *For every $x \in \mathcal{C}_3$ and for every dyadic rational z, with size $|z| = m$,*

(a) if $|x - z| \leq \frac{1}{2^{i+5}}$, then the binary expansions of x and z coincide on the first i bits;

(b) $|x - z| > \frac{1}{2^{m+10}}$.

Proof. (a) Let us suppose that x and z coincide on the first $i - 1$ bits and differ on the i-th bit. There are two possible cases: $z < x$ or $z > x$.

- $z < x$: Then, $z_i = 0$ and $x_i = 1$. In the worst case (the case in which x and z are closer to one another), the binary expansion of z after the i-th bit starts with a sequence of 1's and the binary expansion of x starts with a sequence of 0's. As $x \in \mathcal{C}_3$, we have the following possibilities:

	i	$\lvert x - z \rvert$
z	...011111...	
x (case $i \equiv_3 0$)	...100100...	$> 2^{-(i+3)}$
x (case $i \equiv_3 1$)	...100001...	$> 2^{-(i+5)}$
x (case $i \equiv_3 2$)	...100010...	$> 2^{-(i+4)}$

- $z > x$: Then, $z_i = 1$ and $x_i = 0$. In the worst case, the binary expansion of z after the i-th bit starts with a sequence of 0's and the binary expansion of x starts with a sequence of 1's. As in the previous case, we have three different possibilities:

	i	$\lvert x-z\rvert$
z	...100000...	
x (case $i \equiv_3 0$)	...010010...	$> 2^{-(i+2)}$
x (case $i \equiv_3 1$)	...010100...	$> 2^{-(i+2)}$
x (case $i \equiv_3 2$)	...011001...	$> 2^{-(i+3)}$

In any case, $|x-z| > 2^{-(i+5)}$, which contradicts the initial hypothesis. Therefore, if $|x-z| \leq 2^{-(i+5)}$, then the binary expansions of x and z coincide on the first i bits;

(b) Since the binary expansion of z after the m-th bit is only composed of 0's and every number $x \in \mathcal{C}_3$ has at most four consecutive zeros, in the best fit, z and x cannot coincide in the $(m+5)$-th bit. So, by (a), $|x-z| > 2^{-(m+10)}$.

□

Taking advantage of these properties, we present a technique to encode advice functions of signature $f : \mathbb{N} \to \{0,1\}^*$ into elements of $\mathcal{C}_3$.

As in [7], the encoding of a word $w \in \Sigma^*$, where $\Sigma = \{0,1\}$, is denoted by $c(w)$ and is obtained replacing every 0 in w by 100 and every 1 by 010. Given a function $f \in \log\star$ (see Appendix A), the encoding of f, $y(f) = \lim y(f)(n)$, is recursively defined by:

- $y(f)(0) = 0.c(f(0))$
- if $f(n+1) = f(n)s$, then

$$y(f)(n+1) = \begin{cases} y(f)(n)c(s) & \text{if } n+1 \text{ is not a power of } 2 \\ y(f)(n)c(s)001 & \text{if } n+1 \text{ is a power of } 2 \end{cases}$$

By construction, $y(f) \in \mathcal{C}_3$, since it corresponds to replacing the bits of f by 100 and 010 and adding separators 001 at the end of the codes for every $f(2^k)$, $k \in \mathbb{N}$. To extract the value of $f(n)$, we just have to find the number $m \in \mathbb{N}$ such that $2^{m-1} < n \leq 2^m$ and then read $y(f)$ in triplets, until we find the $(m+1)$-th separator. Then, it is only needed to ignore the separators and replace each 100 triplet by 0 and each 010 triplet by 1. Since f is logarithmic, we know that $|f(2^m)| = O(\log 2^m) = O(m)$, so, we need $3O(m) + 3(m+1) = O(m)$ bits to get the value of $f(2^m)$ and, consequently, $O(\log n)$ bits to get the value of $f(n)$.

2.2 Reading the advice

Now that we have encoded the advice function into a real number in $(0,1)$ and we can use it as the unknown parameter of our AD machine, we will prove that, using the oracle, we can recover the advice function value for the input size. The following lemma generalizes [39], [7], [10], [16] and [12].

Lemma 2.2. *An AD machine with 'yes' probability $F(\sigma)$, given input x of size $|x| = n$, can read $O(\log(n))$ bits of the unknown parameter σ in polynomial time in n, with error probability bounded by γ.*

Proof. We know that each oracle call has $P(\text{`}yes\text{'}) = F(\sigma)$. Thus, if we make ξ oracle calls, the number of times that the experiment returns 'yes', α, is a random variable with binomial distribution. Let us consider $X = \alpha/\xi$, the random variable that represents the relative frequency of 'yes'. We have the expected value $\mathbb{E}[X] = \mathbb{E}[\alpha]/\xi = \xi F(\sigma)/\xi = F(\sigma)$ and the variance $\mathbb{V}[X] = \mathbb{V}[\alpha]/\xi^2 = \xi F(\sigma)(1 - F(\sigma))/\xi^2 = F(\sigma)(1 - F(\sigma))/\xi$. Chebyshev's inequality states that, for every $\delta > 0$,

$$P(|X - \mathbb{E}[X]| > \delta) \leq \frac{\mathbb{V}[X]}{\delta^2} \leq \frac{F(\sigma)(1 - F(\sigma))}{\xi\delta^2} \leq \frac{1}{\xi\delta^2}.$$

By setting $\delta = \nu 2^{-k-6}$, in order to get the desired proximity between X and $F(\sigma)$, we get

$$P(|X - F(\sigma)| > \nu 2^{-k-6}) \leq \frac{2^{2k+12}}{\xi\nu^2}$$

and if we want an error probability of at most γ, we must make a number of oracle calls given by

$$\xi \geq \frac{2^{2k+12}}{\gamma\nu^2}.$$

With error probability $\leq \gamma$ we have $|X - F(\sigma)| \leq \nu 2^{-k-6}$. So, we have used X to estimate $F(\sigma)$. To prevent us from making any mistakes, we must ensure that X is an image of F for some $\sigma \in [0, 1]$, so, if $X \notin F([0, 1])$,[3] we set X to the minimum or maximum value of $F([0, 1])$, depending on X being below or above $F([0, 1])$. Now, we use the bisection method to get $F^{-1}(X)$, i.e., to find the root of $X - F(\sigma)$. Since F' is continuous and $F'(\sigma) \neq 0$ for every $\sigma \in [0, 1]$, F is one-to-one, by Rolle's theorem. Let σ_X be the preimage[4] of X by F and let σ' be the value returned by the bisection method.[5] Since F is a continuous one-to-one function in $[0, 1]$ and X is an image of F, we know that $X - F(\sigma)$ has exactly one zero in $[0, 1]$. If $X - F(0) = 0$ or $X - F(1) = 0$, then $\sigma' = 0$ or $\sigma' = 1$, otherwise, the zero is in $(0, 1)$ and $X - F(0)$ and $X - F(1)$ have opposite signs. In this case, the bisection method converges in $\log(1/\varepsilon)$ steps, where ε is the error of the method. If we want to read the first k bits of σ, we must have $|\sigma - \sigma'| \leq 2^{-k-5}$. We know that

$$|\sigma - \sigma'| \leq |\sigma - \sigma_X| + |\sigma_X - \sigma'|$$

and if we set $|\sigma - \sigma_X| \leq 2^{-k-6}$ and $|\sigma_X - \sigma'| \leq 2^{-k-6}$, we get the intended result. Due to the value chosen for δ, and applying mean value theorem, we already have, with error probability bounded by γ,

$$|\sigma - \sigma_X| \leq \frac{1}{\nu}|F(\sigma) - X| \leq 2^{-k-6}.$$

Thus, we only need an error of 2^{-k-6} in the bisection method. So, the method converges in $\log(2^{k+6}) = k + 6$ steps and it takes $O(kG(k))$ time, where $G(k)$ is the cost of calculating $F(\sigma)$ with k-bits precision.

In conclusion, we can read the first k bits of σ in total time $O(2^{2k} + kG(k))$. Remember that the cost of computing k bits of F is $O(2^k)$. So, in order to get polynomial time in n, the number of bits of σ that

[3] The continuous image of a compact set is a compact set.

[4] See Glossary. In this case, as we made sure that X is an image of F and F is one-to-one, the preimage of X is the unique $\sigma_X \in [0, 1]$ such that $F(\sigma_X) = X$.

[5] In other contexts, this method is known as binary search method.

we can read, k, has to be logarithmic on the size of the input. □

This result will be used in Section 2.4, to prove the lower bound on the computational power of our AD machines.

2.3 Tossing coins

To prove lower bound, we also need to prove that the AD machine can simulate the behaviour of a probabilistic Turing machine, i.e., that it can simulate tosses of an unbiased coin. The proof of the following lemma is a complete proof of a construct seen in [7] and in [1].

Lemma 2.3 (Beggs et al. [7])**.** *Take a biased coin with probability of heads $\delta \in (0,1)$ and $\gamma \in (0,1)$. Then, with probability at least γ, we can use a sequence of independent biased coin tosses of length*

$$\frac{n}{\delta(1-\delta)}\left(1+\frac{1}{\sqrt{1-\gamma}}\right)$$

to produce a sequence of independent fair coin tosses of length n.

Proof. The method to simulate one fair coin toss is to toss the biased coin twice and

- if the result is HT, output H;
- if the result is TH, output T;
- if the result is HH or TT, toss the coin twice again.

Using this method, the probability of getting H is the same as the probability of getting T, so the simulated coin is fair.

The probability of succeeding, i.e., getting result HT or TH in one step is $r = 2\delta(1-\delta)$ and the probability of having to repeat the tosses is $s = 1 - r$. We want to produce a sequence of n fair coin tosses, so we need n successes. The random variable T_n that counts the number of trials until we get n successes has negative binomial distribution. Thus, according to [36] and [33], T_n has probability mass function given by

$$P(T_n = k) = \binom{k-1}{n-1} r^n (1-r)^{k-n},$$

with expected value and variance

$$\mu = \frac{ns}{r} + n = \frac{n}{r}, \qquad v = \frac{ns}{r^2}.$$

By Chebyshev's inequality,

$$P(|T_n - \mu| \geq t) \leq \frac{v}{t^2}.$$

Setting $t = \beta n$, we get

$$P(T_n \geq \mu + \beta n) \leq \frac{ns}{r^2(\beta n)^2} \leq \frac{1}{\beta^2 r^2}.$$

To get the probability of failure less than $1-\gamma$, we need

$$\beta \geq \frac{1}{r\sqrt{1-\gamma}}.$$

Thus, the total number of trials is

$$\frac{n}{r} + \frac{n}{r\sqrt{1-\gamma}} = \frac{n}{r}\left(1 + \frac{1}{\sqrt{1-\gamma}}\right) = \frac{n}{2\delta(1-\delta)}\left(1 + \frac{1}{\sqrt{1-\gamma}}\right).$$

Since for each trial we need to toss a coin twice, the total number of coin tosses is

$$\frac{n}{\delta(1-\delta)}\left(1 + \frac{1}{\sqrt{1-\gamma}}\right).$$

□

Observe that the error probability in generating a sequence of fair coin tosses is associated only with the possibility of not getting a sequence of size n. If we have such a sequence, we know it was well generated.

2.4 Lower bound

We have already seen that there is an error associated with the estimation of the unknown parameter and with the coin tosses. Due to the stochastic nature of the oracle, our AD machines can make errors. For a set $A \subseteq \Sigma^*$, an AD machine $\mathcal{M}$ with unknown parameter σ and with 'yes' probability $F(\sigma)$, and an input $w \in \Sigma^*$, the error probability of $\mathcal{M}$ for input w is either the probability of $\mathcal{M}$ rejecting w, if $w \in A$ or the probability of $\mathcal{M}$ accepting w, if $w \notin A$. We define the decision criterion of these AD machines.

Definition 2.4. Let $A \subseteq \Sigma^*$. We say that an AD machine $\mathcal{M}$ with unknown parameter σ and with 'yes' probability $F(\sigma)$ decides A in polynomial time if $\mathcal{M}$ runs in polynomial time on all inputs and there exists $\gamma < 1/2$ such that, for every input w,

- if $w \in A$, $\mathcal{M}$ accepts w with error probability bounded by γ;
- if $w \notin A$, $\mathcal{M}$ rejects w with error probability bounded by γ.

Now, we can state and prove the lower bound theorem for our AD machines.

Theorem 2.5. *Every set in $BPP//\log\star$ is decidable by an AD machine with 'yes' probability $F(\sigma)$ in polynomial time.*

Proof. Let A be an arbitrary set in $BPP//\log\star$ and $\mathcal{M}$ a probabilistic Turing machine with advice $f \in \log\star$, which decides A in polynomial time with error probability bounded by $\gamma_1 \in (0, 1/2)$.

Let $\mathcal{M}'$ be an AD machine with unknown parameter $y(f)$, where $y(f)$ represents the encoding of f into Cantor set C_3 and let γ_2 be a positive real number such that $\gamma_1 + \gamma_2 < 1/2$. Let w be a word such

that $|w| = n$. We know, by Lemma 2.2, that $\mathcal{M}'$ can estimate a logarithmic number of bits of $y(f)$, and, thus, read $f(n)$, in polynomial time, with an error probability bounded by γ_2.

Now we need to simulate independent unbiased coin tosses. Let γ_3 be such that $\gamma_1 + \gamma_2 + \gamma_3 < 1/2$. We have $P(\text{`}yes\text{'}) = F(\sigma) = \delta$ and $P(\text{`}no\text{'}) = 1 - \delta$, so by Lemma 2.3 $\mathcal{M}'$ can produce a sequence of fair coin tosses in time linear on the size of the sequence, to within a probability of γ_3, thus it can behave probabilistically. Therefore, $\mathcal{M}'$ can decide A in polynomial time, with error probability bounded by $\gamma_1 + \gamma_2 + \gamma_3 < 1/2$. □

2.5 Probabilistic Trees

In this section, we define probabilistic tree and present several results about it. In the context of our work, probabilistic trees were already studied in [9], [10] and [1], however, in this dissertation, we complete and clarify some unclear aspects. We define not only acceptance probability but also rejection probability and we analyse the case of probabilistic trees with one single node, that was left behind in previous works.

During the computation of an AD machine $\mathcal{M}$, for a particular input w, the finite control of the machine may eventually turn to the query state, which triggers the physical experiment, and, after the scheduled time, it will be in one of two states, the 'yes' state or the 'no' state. Then, depending on that result, the machine will perform some deterministic computations before a new call to the oracle and the subsequent experiment. Therefore, the oracle calls of the AD machine can be represented in a binary tree. Furthermore, as we know that the probability of 'yes' in a single oracle call is $F(\sigma)$, we can assign a probability to each edge of the tree.

More generally, we can define probabilistic trees.

Definition 2.6. A query tree T is an ordered tree (V, E), associated with an oracle Turing machine, where each node in V is a configuration of the machine in the query state or in a halting state and each edge in E is a deterministic computation of the Turing machine between consecutive oracle calls or between oracle calls and halting configurations. Moreover, every node with zero children is called a leaf and is labeled with 'A' or 'R', depending on whether it abstracts an accepting or rejecting configuration.

In a query tree, every internal node represents a call to the oracle. As our oracle is stochastic, there is a probability associated to each outcome of the oracle. Thus, we can assign probabilities to the edges of the query tree.

A single computation of a Turing machine, over the input w, corresponds to a path in the tree, beginning in the root and ending in a leaf. If the path ends in a leaf labeled with 'A', the machine halts in one accepting state, otherwise, it halts in one rejecting state.

Definition 2.7. A probabilistic tree is a pair (T, D) where T is a query tree, being V its set of nodes and E its set of edges, and $D : E \to [0, 1]$ is an assignment of probabilities to the edges of T, such that the probabilities of the outgoing edges of every internal node sum 1.

Let $\rho(T)$ be the set of all possible assignments of probabilities to the edges of T.

Definition 2.8. Given a probabilistic tree (T, D), its probability of acceptance, $P_A(T, D)$, is the sum of the products of probabilities along the edges in every path that leads to an accepting node, i.e.,

$$P_A(T, D) = \sum_{c \in C_A} \left(\prod_{i=1}^{m_c} D(c[i]) \right),$$

where C_A is the set of all paths that end in an accepting leaf, m_c is the length of the path c and $c[i]$ is the i-th edge of that path.

Similarly, we define probability of rejection, $P_R(T, D)$, as the sum of the products of probabilities along the edges in every path that leads to a rejecting node.

$$P_R(T, D) = \sum_{c \in C_R} \left(\prod_{i=1}^{m_c} D(c[i]) \right),$$

where C_R is the set of all paths that end in a rejecting leaf, m_c is the length of the path c and $c[i]$ is the i-th edge of that path.

Obviously, we have that $P_A(T, D) + P_R(T, D) = 1$.

If the query tree T is composed by only one node, i.e., a leaf, labeled with 'A', the probability of acceptance is a sum of only one empty product, thus $P_A(T, D) = 1$, and the probability of rejection is an empty sum, thus $P_R(T, D) = 0$. The case in which the leaf is labeled with 'R' is analogous. Therefore, $P_A(T, D)$ and $P_R(T, D)$ are well-defined, even for the case of one single node.

Note that we can assume that all leaves are at the same depth, i.e., all the paths from the root to the leaves have the same length, because, even if some computation ends without making m calls to the oracle, we can continue the computation, doing nothing besides the oracle calls, and assigning to all the resulting leaves the same label, depending on the state in which the computation had ended.

Let us define distance between two probabilistic trees that have the same query tree.

Definition 2.9. Given two probabilistic trees (T, D) and (T, D') with the same query tree, T, the distance between those probabilistic trees, denoted by $d(D, D')$ is given by

$$d(D, D') = \max_{e \in E} \{D(e) - D'(e)\}.$$

We can restrict the domain of trees that we want to work with, focusing on t-ary probabilistic trees, i.e., probabilistic trees such that each internal node of the query tree has exactly t children.

Let us denote by $\mathcal{T}_m^t$ the set of all t-ary probabilistic trees of height m.

If $m = 0$, the probabilistic tree is composed of one single node, and it corresponds to the case where the machine does not consult the oracle. If T is a query tree of height 0, we have that, for every $D \in \rho(T)$,[6] $P_A(T, D) = 1$ or $P_A(T, D) = 0$, according to the label of the single leaf of T being 'A' or 'R', respectively.

Definition 2.10. For every $m \in \mathbb{N}, \beta \in [0, 1]$ and $t \in \mathbb{N}$, we define a function, $f_t : \mathbb{N} \times [0, 1] \to [0, 1]$, that gives the largest possible difference in the probability of acceptance for two probabilistic trees with the

[6] There is only one assignment of probabilities to the edges of T: the empty one.

same t-ary query tree of height m, such that their distance is bounded by β. That function is given by

$$f_t(m,\beta) = \sup\{|P_A(T,D) - P_A(T,D')| : T \in \mathcal{T}_m^t, D, D' \in \rho(T), d(D,D') \leq \beta\}.$$

Now, let us establish a bound for $f_t(m,\beta)$.

The proof of the following proposition is a complete proof of the construct seen in [9] and in [10].

Proposition 2.11. *For any $m \in \mathbb{N}, \beta \in [0,1]$ and $t \in \mathbb{N}$, $f_t(m,\beta) \leq (t-1)m\beta$.*

Proof. The proof is done by induction on m.

For $m = 0$, the result is true, since the query tree T has only one node and there is only one assignment of probabilities $D \in \rho(T)$. Thus, we have

$$\begin{aligned} f_t(0,\beta) &= \sup\{|P_A(T,D') - P_A(T,D'')| : T \in \mathcal{T}_0^t, D', D'' \in \rho(T), d(D',D'') \leq \beta\} \\ &= \sup\{|P_A(T,D) - P_A(T,D)| : T \in \mathcal{T}_0^t\} = 0. \end{aligned}$$

Now, assume that the result is true for m, i.e. $f_t(m,\beta) \leq (t-1)m\beta$. We shall prove that $f_t(m+1,\beta) \leq (t-1)(m+1)\beta$.

Let $T \in \mathcal{T}_{m+1}^t$. Then, as T is a t-ary tree of height $m+1$, T has t edges $e_1, e_2, ..., e_t$ leaving the root and t probabilistic trees $T_1, T_2, ..., T_t \in \mathcal{T}_m^t$ that correspond to the subtrees of the root.

Let $D, D' \in \rho(T)$ such that $d(D,D') \leq \beta$ and let $D_1, D_2, ..., D_t$ and $D_1', D_2', ..., D_t'$ be the restrictions of D and D' to $T_1, T_2, ..., T_t$ respectively. We have

$$P_A(T,D) = D(e_1)P_A(T_1,D_1) + D(e_2)P_A(T_2,D_2) + ... + D(e_t)P_A(T_t,D_t);$$

and

$$P_A(T,D') = D'(e_1)P_A(T_1,D_1') + D'(e_2)P_A(T_2,D_2') + ... + D'(e_t)P_A(T_t,D_t').$$

Since $D(e_t) = 1 - D(e_1) - D(e_2) - ... - D(e_{t-1})$ and $D'(e_t) = 1 - D'(e_1) - D'(e_2) - ... - D'(e_{t-1})$, we have

$$\begin{aligned} &|P_A(T,D) - P_A(T,D')| = \\ &\quad |(D(e_1) - D'(e_1))(P_A(T_1,D_1) - P_A(T_t,D_t)) + (D(e_2) - D'(e_2))(P_A(T_2,D_2) - P_A(T_t,D_t)) \\ &\quad + ... + (D(e_{t-1}) - D'(e_{t-1}))(P_A(T_{t-1},D_{t-1}) - P_A(T_t,D_t)) \\ &\quad + D'(e_1)(P_A(T_1,D_1) - P_A(T_1,D_1')) + D'(e_2)(P_A(T_2,D_2) - P_A(T_2,D_2')) \\ &\quad + ... + D'(e_t)(P_A(T_t,D_t) - P_A(T_t,D_t'))|. \end{aligned}$$

Using the definition of $f_t(m,\beta)$ and the fact that the difference of two real numbers in [0,1] is less

than or equal to 1,

$$\begin{aligned}|P_A(T,D) - P_A(T,D')| &\leq |D(e_1) - D'(e_1)| + |D(e_2) - D'(e_2)| + ... + |D(e_{t-1}) - D'(e_{t-1})| \\ &\quad + D'(e_1)f_t(m,\beta) + D'(e_2)f_t(m,\beta) + ... + D'(e_t)f_t(m,\beta)| \\ &\leq (t-1)\beta + f_t(m,\beta).\end{aligned}$$

Thus, by induction hypothesis,

$$|P_A(T,D) - P_A(T,D')| \leq (t-1)\beta + (t-1)m\beta \leq (t-1)(m+1)\beta.$$

Therefore,

$$\begin{aligned}f_t(m+1,\beta) &= \sup\{|P_A(T,D) - P_A(T,D')| : T \in \mathcal{T}^t_{m+1}, D, D' \in \rho(T), d(D,D') \leq \beta\} \\ &\leq (t-1)(m+1)\beta.\end{aligned}$$

□

Analogously, we can define a function $g_t : \mathbb{N} \times [0,1] \to [0,1]$, that gives the largest possible difference in the probability of rejection for two probabilistic trees with the same t-ary query tree of height m, such that their distance is bounded by β,

$$g_t(m,\beta) = \sup\{|P_R(T,D) - P_R(T,D')| : T \in \mathcal{T}^t_m, D, D' \in \rho(T), d(D,D') \leq \beta\}$$

and we can prove for g_t the same bound as for f_t, i.e., for any $m \in \mathbb{N}, \beta \in [0,1]$ and $t \in \mathbb{N}$, $g_t(m,\beta) \leq (t-1)m\beta$.

Now that we have introduced some notions about probabilistic trees, we can focus in our problem.

As our oracle has only two outcomes, we can represent the calls to the oracle of an AD machine by a binary probabilistic tree, such as in Figure 2.1.

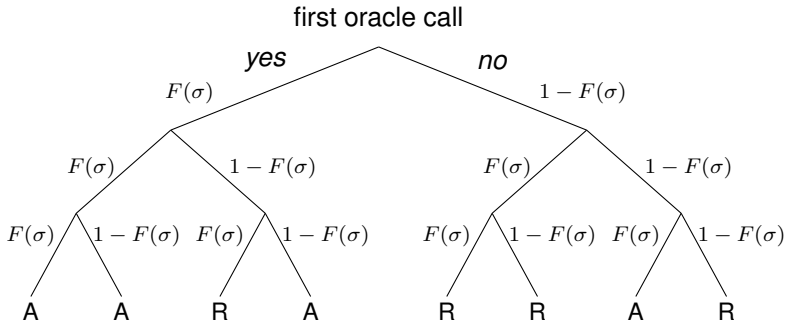

Figure 2.1: The oracle consultations of an AD machine with unknown parameter σ and with 'yes' probability $F(\sigma)$ as a binary probabilistic tree in the case where there are three oracle calls.

For binary trees, the important property of Proposition 2.11 gives the following:

Corollary 2.12. *For any $m \in \mathbb{N}$ and $\beta \in [0,1]$, $f_2(m,\beta) \leq m\beta$.*

Our main goal for this section is to show that the sets decided by our AD machine can also be decided by a probabilistic machine, with some advice. We know that a probabilistic Turing machine is associated with a computation tree which is a binary tree where all the leaves are labeled with 'A' or 'R' for *accept* or *reject*, respectively. In the proof of Theorem 2.15, we build a probabilistic tree, from the computation tree of a probabilistic Turing machine, and we verify that the probabilities of acceptance are consistent.

The following lemma generalizes results in [9] and [16].

Lemma 2.13. *Let $\mathcal{M}$ be an AD machine with unknown parameter σ and with 'yes' probability $F(\sigma)$, deciding some set A in time $t(n)$ with error probability bounded by $\gamma < 1/4$. Let $\mathcal{M}'$ be an identical AD machine, with the exception that the unknown parameter is $\tilde{\sigma}$ and the probability of 'yes' is given by $\tilde{F}$. If*

$$|F(\sigma) - \tilde{F}(\tilde{\sigma})| < \frac{1}{8t(n)},$$

then for any word of size $\leq n$, the probability of $\mathcal{M}'$ making an error when deciding A is less or equal than $3/8$.

Proof. In time $t(n)$, at most $t(n)$ calls to the oracle are made, so the query tree T associated to $\mathcal{M}$ and $\mathcal{M}'$ has maximum depth $t(n)$. Note that the query tree of $\mathcal{M}$ and $\mathcal{M}'$ is the same, since the machines only differ in the probabilities associated with the calls to the oracle. Let w be a word such that $|w| \leq n$. Let $D \in \rho(T)$ be the assignment of probabilities to the edges of T corresponding to the unknown parameter σ and 'yes' probability $F(\sigma)$ and $D' \in \rho(T)$ the assignment of probabilities given by unknown parameter $\tilde{\sigma}$ and 'yes' probability $\tilde{F}(\tilde{\sigma})$.

As $|F(\sigma) - \tilde{F}(\tilde{\sigma})| < 1/8t(n)$, the difference between any particular probability is at most

$$k = \frac{1}{8t(n)}.$$

To calculate the probability of an incorrect result for the machine with unknown parameter $\tilde{\sigma}$ and 'yes' probability given by $\tilde{F}$, we have to consider two different cases.

- $w \notin A$: In this case, an incorrect result corresponds to $\mathcal{M}'$ accepting w. Applying Corollary 2.12, we can bound the probability of acceptance for $\mathcal{M}'$ in the following way:

$$\begin{aligned} P_A(T,D') &\leq P_A(T,D) + |P_A(T,D') - P_A(T,D)| \\ &\leq \gamma + t(n)k \\ &\leq \gamma + t(n)\frac{1}{8t(n)} \\ &= \frac{1}{4} + \frac{1}{8} = \frac{3}{8}. \end{aligned}$$

- $w \in A$: In this case, an incorrect result corresponds to $\mathcal{M}'$ rejecting w. Applying Corollary 2.12,

we can bound the probability of rejection for $\mathcal{M}'$ in the following way:

$$\begin{aligned} P_R(T, D') &\leq P_R(T, D) + |P_R(T, D') - P_R(T, D)| \\ &\leq \gamma + t(n)k \\ &\leq \gamma + t(n)\frac{1}{8t(n)} \\ &= \frac{1}{4} + \frac{1}{8} = \frac{3}{8}. \end{aligned}$$

In both cases, the error probability is bounded by $3/8$. □

Lemma 2.13 has the following corollary:

Corollary 2.14. *Let $\mathcal{M}$ be an AD machine with unknown parameter σ and with 'yes' probability $F(\sigma)$, deciding some set in time $t(n)$ with error probability bounded by $\gamma < 1/4$. Let $\mathcal{M}_n$ be an identical AD machine, also with unknown parameter σ, but with the exception that the probability of 'yes' is given by $F(\sigma)\downarrow_{\log t(n)+3}$. Then, $\mathcal{M}_n$ decides the same set as $\mathcal{M}$, also in time $t(n)$, but with error probability bounded by $3/8$.*

2.6 Upper bound

Theorem 2.15. *Every set decided by an AD machine with unknown parameter σ and with 'yes' probability $F(\sigma)$ in polynomial time is in $BPP//\log\star$.*

Proof. Let A be a set decided by an AD machine $\mathcal{M}$ with unknown parameter σ and 'yes' probability $F(\sigma)$ in polynomial time $t(n)$ and with error probability bounded by 1/4.[7]

We want to construct a probabilistic Turing machine with advice, $\mathcal{M}'$, which decides A.

Let us use the advice function $f(n) = F(\sigma)\downarrow_{\log t(n)+3}$. We have $f \in \log\star$.

By Corollary 2.14, we know that an AD machine with 'yes' probability $f(n)$ decides the same for words of size $\leq n$ than $\mathcal{M}$, but with error probability $\leq 3/8$.

Thus, $\mathcal{M}'$ must simulate the behaviour of an AD machine with unknown parameter σ and with 'yes' probability $f(n)$. We do that with fair coin tosses.

$f(n) = F(\sigma)\downarrow_{\log t(n)+3}$ is a dyadic rational with denominator $2^{\log t(n)+3}$. Thus, $m = 2^{\log t(n)+3} f(n) \in [0, 2^{\log t(n)+3})]$ is an integer. Now, make $k = \log t(n) + 3$ fair coin tosses, interpreting the results τ_i (taken in order) as either 0 or 1. Then, test if $\tau_1\tau_2...\tau_k < m$, where $\tau_1\tau_2...\tau_k$ is viewed as the binary representation of an integer. If the test is true, return 'yes', otherwise return 'no'. The probability of returning 'yes' is $m/2^k = f(n)$, as required. The time taken is polynomial in n.

Let us observe that the probabilistic machine $\mathcal{M}'$ has in fact an error probability bounded by $3/8$, showing that its probability of acceptance, obtained by counting accepting configurations in its computation tree coincide with the probability of acceptance from the probabilistic tree of the simulated AD machine.

[7] Using the same technique as in Proposition B.2, it can be proved that we may consider any bound of the form $2^{-q(n)}$, for any polynomial q, for the error probability of our AD machines.

The computation tree of $\mathcal{M}'$ has maximum depth $t(n)(\log t(n)+3)$, since there are at most $t(n)$ calls to the oracle and, for each call, $\mathcal{M}'$ tosses $k = \log t(n) + 3$ coins. Moreover, those are the only steps of the computation in which $\mathcal{M}'$ behaves probabilistically. To obtain a probabilistic tree that corresponds to this computation tree, we proceed in the following way: for every ik level, $i \in \mathbb{N}$, we create a new level in the probabilistic tree, assigning probability $f(n)$ to every left edge and $1 - f(n)$ to every right edge. The obtained tree has height $t(n)$.

Now, let us see that the probabilities are consistent. In the k level of the computation tree, we have 2^k nodes, and we know that m of those correspond to the result 'yes', so, their subtrees are equal. For each of those m nodes, if we descend more k levels in their subtrees, we have m nodes that correspond to 'yes' result. So, at this $2k$ level, we know that there are m^2 nodes corresponding to having 'yes' in the two first experiments, and so on. When we have traversed the whole tree, we know that in the last level, we have $2^{kt(n)}$ leaves, and, for example, $m^{t(n)}$ of those correspond to having 'yes' in all the experiments. So, the probability of having 'yes' in all the experiments is

$$\frac{m^{t(n)}}{2^{kt(n)}} = \left(\frac{m}{2^k}\right)^{t(n)} = f(n)^{t(n)}.$$

In the probabilistic tree, this probability would be the product of $t(n)$ 'yes' probabilities, so $f(n)^{t(n)}$, as expected. The other probabilities coincide as well. □

Using Theorems 2.5 and 2.15, we get the following corollary:

Corollary 2.16. *The class of sets which are decidable in polynomial time by AD machines with unknown parameter σ and with 'yes' probability $F(\sigma)$ is exactly $BPP//\log\star$.*

2.7 Some concrete cases

The results that we have presented in this chapter abstract some of the previous work (in [7, 14, 10]) relative to Turing machines with physical experiments as oracles, both two-sided experiments and threshold experiments, in the case of fixed precision.

As mentioned in Chapter 1, the measurement experiments that we have been considering can be divided into three types: threshold experiments, two-sided experiments and vanishing experiments.

A threshold experiment is an experiment that approximates the unknown value just from one side, i.e., it approximates the unknown value y either with values just from above or with values just from below, z, checking either if $y < z$ or if $z < y$, but not both. This type of measurement is schematized in Figure 2.2.

A two-sided experiment is an experiment that approximates the unknown value from two sides, i.e., it approximates the unknown value y with values from above or with values from below, z, checking if $y < z$ and $z < y$. This type of measurement is schematized in Figure 2.3.

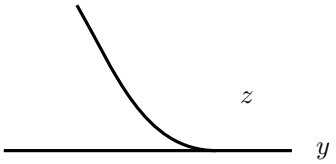

Figure 2.2: Threshold measurement.

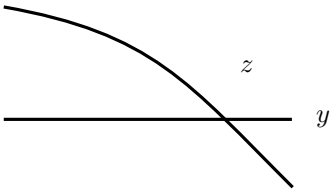

Figure 2.3: Two-sided measurement.

A vanishing experiment is an experiment in which we are only able to test the condition "$z \neq y$". It approximates the unknown value y using the physical time taken by the experiment. In order to do that, we will need to perform two instances of the experiment (two queries to the oracle) and determine which of these two queries is answered first. This type of measurement is schematized in Figure 2.4.

Figure 2.4: Vanishing measurement.

It was also mentioned in Chapter 1 that there are many experiments that have been studied, of all types. In [18], it is presented an overview of the three types of experiments and several examples are given for each type. In this section, we pay special attention to threshold and two-sided experiments. In the case of threshold experiments, the experiments that have been studied include the broken balance experiment (see [10, 37]), the Rutherford scattering experiment (see [10, 37]) and the photoelectric effect experiment (see [10, 37]). In the case of two-sided experiments, the experiments already studied include the scatter experiment (in its sharp and smooth versions) [8], the balance scale experiment (see [11, 37]), the collider experiment (see [12]) and the Wheatstone bridge (see [13]). We will focus on the balance experiment, instead of the scatter experiment (that we have studied in Chapter 1), because it is also an intuitive experiment and it exists in both versions.

2.7.1 The broken balance experiment

The broken balance experiment (BBE) consists of a balance scale with two plates. In the right plate of the balance is placed a body with unknown mass, y, that we intend to measure. To do so, we place

[8] In this section, we only care about the smooth scatter experiment, since is the one that is physically realizable and that has exponential time.

test masses z on the left plate of the balance. If $z < y$, then the plates will not move since the rigid block prevents the right plate from moving down; if $z > y$, then the left plate will move down; if $z = y$, the plates will not move. The scheme of this experiment is represented in Figure 2.5.

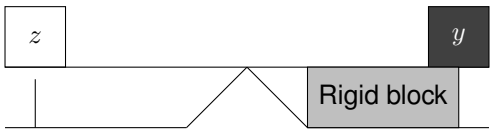

Figure 2.5: The broken balance experiment.

In the physical apparatus, a pressure-sensitive stick is placed below the left plate of the balance, such that, if the left plate moves down, it touches the pressure-sensitive stick and it reacts producing a signal. Several physical properties of the experiment are discussed in [10] and [37]. Here, we just need to specify the experimental time of the BBE, which, for every unknown mass y and test mass z, is given by the function:

$$T_{exp}(z,y) = \begin{cases} c \times \sqrt{\frac{z+y}{z-y}} & \text{if } z > y \\ \infty & \text{otherwise} \end{cases},$$

where c is a constant.

A broken balance machine, or BB machine, is a Turing machine coupled with a BBE. The Turing machine interacts with the BBE using the query tape. To initialize the experiment, the Turing machine writes in the query tape the parameters for the experiment, namely the test mass. After the test mass is placed in the left plate (according to some precision protocol), the machine will wait $T(|z|)$ units of time, where T is the time schedule. If the pressure-sensitive stick produces a signal during the scheduled time, the oracle returns 'yes', otherwise, it returns 'timeout'.

Unlike the AD machines that we have studied in this chapter, in the cases that we consider now, the Turing machine makes queries to the oracle, providing it with a dyadic rational, needed to initialize the experiment. This communication between the Turing machine and the physical oracle is ruled by one of the following protocols:

- infinite precision: when z is read in the query tape and a test mass $z' = z$ is placed in the left plate;
- unbounded precision: when z is read in the query tape and a test mass z' is placed in the left plate, where z' is randomly and uniformly chosen in $(z - 2^{-|z|}, z + 2^{-|z|})$;
- fixed precision ε ($\varepsilon > 0$): when z is read in the query tape and a test mass z' is placed in the left plate, where z' is randomly and uniformly chosen in $(z - \varepsilon, z + \varepsilon)$.

In this section, we are only concerned about the fixed precision protocol.

We present the lower bound theorem, for fixed precision case, proved in [10, 37], but we reconstruct the proof, using the results obtained in this chapter.

Theorem 2.17. *If $A \in BPP//\log\star$ and $\varepsilon \in (0, 1/2)$, then A is decidable by a BB machine with fixed precision ε in polynomial time.*

Proof. Let A be an arbitrary set in $BPP//\log\star$ and $\mathcal{M}$ a probabilistic Turing machine with advice $f \in \log\star$, which decides A in polynomial time with error probability bounded by $\gamma_1 \in (0, 1/2)$. Let $y = y(f)$, where $y(f)$ represents the encoding of f into Cantor set C_3, and let $\mu = 1/2 + \varepsilon - 2y\varepsilon$. We have that $1/2 - \varepsilon \leq \mu \leq 1/2 + \varepsilon$.

Let $\mathcal{M}'$ be a BB machine such that the unknown mass of the BBE oracle is μ. Fixing a time schedule T, we consider that the machine $\mathcal{M}'$ can only call the BBE oracle with the query $1\rfloor_\ell$, where ℓ is a fixed natural number. This means that we always want to place a test mass $1/2$ on the left plate and wait $T(\ell)$ units of time, since ℓ is the length of the queries. The experiment will be performed with initial condition $z' \in (1/2 - \varepsilon, 1/2 + \varepsilon)$. We can split this interval in two:

- an interval $(1/2 - \varepsilon, \mu + \eta)$ in which the result of the experiment is always 'timeout';
- an interval $(\mu + \eta, 1/2 + \varepsilon)$ in which the result of the experiment is always 'yes';

where η is such that $T_{exp}(\mu + \eta, \mu) = T(\ell)$.

Thus, $\mathcal{M}'$ can be seen as an AD machine with an oracle with two possible outcomes and a constant time schedule ($T(\ell)$) and such that

$$P(\text{`}yes\text{'}) = \frac{\frac{1}{2} + \varepsilon - \mu - \eta}{\frac{1}{2} + \varepsilon - \frac{1}{2} + \varepsilon} = y - \frac{\eta}{2\varepsilon}.$$

From $T_{exp}(\mu + \eta, \mu) = T(\ell)$, we get that

$$\eta = \frac{2\mu}{(\frac{T(\ell)}{c})^2 - 1},$$

and, consequently, $P(\text{`}yes\text{'})$ is linear on y, so it fulfils the conditions settled at the beginning of the chapter.

Therefore, we can apply Lemma 2.2 and we get that, for every input w such that $|w| = n$, $\mathcal{M}'$ can estimate a logarithmic number of bits of $y = y(f)$, and, thus, read $f(n)$, in polynomial time, with an error probability bounded by γ_2, where γ_2 is a positive real number such that $\gamma_1 + \gamma_2 < 1/2$.

Let γ_3 be such that $\gamma_1 + \gamma_2 + \gamma_3 < 1/2$. The probability of getting the outcome 'yes' is always the same, thus, we have $P(\text{`}yes\text{'}) = \delta$ and $P(\text{`}no\text{'}) = 1 - \delta$, so by Lemma 2.3 $\mathcal{M}'$ can produce a sequence of fair coin tosses in time linear on the size of the sequence, to within a probability of γ_3, thus it can behave probabilistically. Therefore, $\mathcal{M}'$ can decide A in polynomial time, with error probability bounded by $\gamma_1 + \gamma_2 + \gamma_3 < 1/2$. □

2.7.2 The balance scale experiment

The balance scale experiment (BSE) consists in trying to determine the mass y of a body placed in one plate of a balance, by means of approximations by another body, with dyadic rational mass z, placed on the other plate. The test mass, z, can be bigger or smaller than y: if $z < y$, the plate that contains the unknown mass y will move down; if $z > y$, the plate that contains the test mass z will move down; if $z = y$, the plates will not move. The main difference from the broken balance experiment is that,

instead of only observing a reaction if the test mass is bigger than the unknown mass, now we observe a reaction for every $z \neq y$. The scheme of the BSE is represented in Figure 2.6.

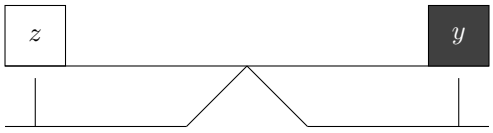

Figure 2.6: The balance scale experiment.

In the physical apparatus, a pressure-sensitive stick is placed below each plate of the balance, such that, if any plate moves down, it touches the corresponding pressure-sensitive stick and it reacts producing a signal. Several physical properties of the experiment are discussed in [37]. The experimental time of the BSE, for every unknown mass y and test mass $z \neq y$, is given by the function:

$$T_{exp}(z, y) = c \times \sqrt{\frac{z+y}{|z-y|}},$$

where c is a constant.

A balance scale machine, or BS machine, is a Turing machine coupled with a BSE. The Turing machine interacts with the BSE using the query tape. To initialize the experiment, the Turing machine writes in the query tape the parameters for the experiment, namely the test mass. After the test mass is placed in the left plate (according to some precision protocol), the machine will wait $T(|z|)$ units of time, where T is the time schedule. If any pressure-sensitive stick produces a signal during the scheduled time, the oracle returns 'left' or 'right', according to the pressure-sensitive stick that had produced the signal, otherwise, it returns 'timeout'.

We present the lower bound theorem, for fixed precision case, and we prove it, using the results obtained in this chapter.

Theorem 2.18. *If $A \in BPP//\log\star$ and $\varepsilon \in (0, 1/2)$, then A is decidable by a BS machine with fixed precision ε in polynomial time.*

Proof. Let A be an arbitrary set in $BPP//\log\star$ and $\mathcal{M}$ a probabilistic Turing machine with advice $f \in \log\star$, which decides A in polynomial time with error probability bounded by $\gamma_1 \in (0, 1/2)$. Let $y = y(f)$, where $y(f)$ represents the encoding of f into Cantor set C_3, and let $\mu = 1/2 + \varepsilon - 2y\varepsilon$. We have that $1/2 - \varepsilon \leq \mu \leq 1/2 + \varepsilon$.

Let $\mathcal{M}'$ be a BS machine such that the unknown mass of the BSE oracle is μ. Fixing a time schedule T, we consider that the machine $\mathcal{M}'$ can only call the BSE oracle with the query $1\restriction_\ell$, where ℓ is a fixed natural number. This means that we always want to place a test mass $1/2$ on the left plate and wait $T(\ell)$ units of time, where ℓ is the length of the queries. The experiment will be performed with initial condition $z' \in (1/2 - \varepsilon, 1/2 + \varepsilon)$. We can split this interval in three:

- an interval $(1/2 - \varepsilon, \mu - \eta)$ in which the result of the experiment is always 'right';
- an interval $(\mu - \eta, \mu + \eta)$ in which the result of the experiment is always 'timeout';

- an interval $(\mu + \eta, 1/2 + \varepsilon)$ in which the result of the experiment is always 'left';

where $\eta > 0$ is such that $T_{exp}(\mu + \eta, \mu) = T(\ell)$.

If we focus only on getting the outcome 'left', $\mathcal{M}'$ can be seen as an AD machine with an oracle with two possible outcomes (considering the outcomes 'right' and 'timeout' as a single outcome) and a constant time schedule ($T(\ell)$) and such that

$$P(`left') = \frac{\frac{1}{2} + \varepsilon - \mu - \eta}{\frac{1}{2} + \varepsilon - \frac{1}{2} + \varepsilon} = y - \frac{\eta}{2\varepsilon}.$$

From $T_{exp}(\mu + \eta, \mu) = T(\ell)$, we get that

$$\eta = \frac{2\mu}{(\frac{T(\ell)}{c})^2 - 1},$$

and, consequently, $P(`left')$ is linear on y, so it fulfils the conditions settled at the beginning of the chapter.

Therefore, we can apply Lemma 2.2 and we get that, for every input w such that $|w| = n$, $\mathcal{M}'$ can estimate a logarithmic number of bits of $y = y(f)$, and, thus, read $f(n)$, in polynomial time, with an error probability bounded by γ_2, where γ_2 is a positive real number such that $\gamma_1 + \gamma_2 < 1/2$.

Let γ_3 be such that $\gamma_1 + \gamma_2 + \gamma_3 < 1/2$. The probability of getting the outcome 'left' is always the same, thus, we have $P(`left') = \delta$, so by Lemma 2.3 $\mathcal{M}'$ can produce a sequence of fair coin tosses in time linear on the size of the sequence, to within a probability of γ_3, thus it can behave probabilistically. Therefore, $\mathcal{M}'$ can decide A in polynomial time, with error probability bounded by $\gamma_1 + \gamma_2 + \gamma_3 < 1/2$. □

2.8 Real numbers as probabilities of transition

After all the discussion about these AD machines, there is one question that arises: is it possible to remove the oracle and reduce everything to a probabilistic Turing machine with a probability of transition that is a real number? In this section , we will answer affirmatively to that question.

In [39], Hava Siegelmann presents the following definition of probabilistic Turing machines:

Definition 2.19 (Siegelmann [39])**.** A probabilistic Turing machine is a machine that computes as follows:

(a) Every step of the computation can have two outcomes, one chosen with probability p and the other with probability $1 - p$;

(b) All computations on the same input require the same number of steps;

(c) Every computation ends with *reject* or *accept*.

Observe that, although every step of the computation can be made in exactly two possible ways, the machine can still take deterministic steps, if there is no difference in the corresponding actions of the two possible successor configurations.

In the classical definition of probabilistic Turing machine, p takes the value $1/2$.

To avoid confusion, we call generalized probabilistic Turing machine to the probabilistic Turing machine of Definition 2.19, i.e., the probabilistic machine such that p can take any real value in $(0, 1)$.

Our goal in this section is to prove that generalized probabilistic Turing machines are equivalent to the AD machines presented in this chapter.

First, we discuss the meaning of accepting or rejecting some input, in terms of generalized probabilistic Turing machines.

The tree of computations of a generalized probabilistic Turing machine is a full binary tree, i.e., all leaves are at the same depth, and a computation of the machine consists in a path from the root to a leaf, being an accepting or rejecting computation, depending on whether it ends in *reject* or *accept*, respectively. However, unlike classical probabilistic Turing machines, in which all computations have the same probability, for generalized probabilistic machines, we associate with each computation a probability which is the product of the probabilities at each computation step. The probability of accepting an input is the sum of the probabilities associated with the accepting computations of the machine on the given input. The error probability of the machine, over some input, is the sum of the probabilities associated with the computations that give the wrong answer.

Definition 2.20. Let $A \subseteq \Sigma^*$. We say that a generalized probabilistic Turing machine $\mathcal{M}$ decides A with bounded error probability in polynomial time if $\mathcal{M}$ runs in polynomial time on all inputs and there exists $\gamma < 1/2$ such that, for every input w,

- if $w \in A$, $\mathcal{M}$ accepts w with error probability bounded γ;
- if $w \notin A$, $\mathcal{M}$ rejects w with error probability bounded by γ.

Now, we can analyse the relation between generalized probabilistic Turing machines and the AD machines that we have studied.

Theorem 2.21. *Every set decided by an AD machine with unknown parameter σ and with 'yes' probability $F(\sigma)$ in polynomial time is decided by a generalized probabilistic Turing machine with bounded error probability in polynomial time.*

Proof. Let A be a set decided by an AD machine $\mathcal{M}$ with unknown parameter σ and 'yes' probability $F(\sigma)$ in polynomial time $t(n)$ and with error probability bounded by $\gamma < 1/2$.

We construct a generalized probabilistic Turing machine, $\mathcal{M}'$, which decides A.

First, we need to define the probability of transition. Let it be $F(\sigma)$ or $1 - F(\sigma)$. Now, we just need to prove that $\mathcal{M}'$ can simulate $\mathcal{M}$. For that, it needs to simulate the deterministic steps and the calls to the oracle.

For the deterministic steps, as we have noted above, a generalized probabilistic machine can perform deterministic computations, being sufficient that it makes the same operations on both outcomes of each computation step.

For the oracle calls, we know that, in each call, the machine $\mathcal{M}$ obtains the outcome 'yes' with probability $F(\sigma)$ and 'no' with probability $1 - F(\sigma)$ and, then, it continues the computation. Thus, $\mathcal{M}'$ just has to make one transition, in which, with probability $F(\sigma)$ it will make the computations of $\mathcal{M}$ relative

to the outcome 'yes', and with probability $1 - F(\sigma)$ it will make the computations of $\mathcal{M}$ relative to the outcome 'no'.

The machine $\mathcal{M}'$ makes the same number of transitions than machine $\mathcal{M}$, so, it runs in polynomial time. Also, as it makes the exactly same operations than $\mathcal{M}$, with the same probability, it decides A with error probability bounded by $\gamma < 1/2$. □

To prove the reverse statement, we have to restrict a little our universe of functions F. We consider only the functions that take all the values in $[0, 1]$, i.e., the functions F such that $F([0, 1]) = [0, 1]$.

Theorem 2.22. *Every set decided by a generalized probabilistic Turing machine with bounded error probability in polynomial time is decided by an AD machine with 'yes' probability $F(\sigma)$ in polynomial time.*

Proof. Let A be a set decided by a generalized probabilistic Turing machine $\mathcal{M}$ with transition probability p in polynomial time and with error probability bounded by $\gamma < 1/2$.

Let $\mathcal{M}'$ be an AD machine with 'yes' probability $F(\sigma)$. We set the unknown parameter σ to $F^{-1}(p)$, i.e., the unique value in $[0, 1]$ such that $F(F^{-1}(p)) = p$.[9] Thus, in each oracle call that $\mathcal{M}'$ makes, it gets the outcome 'yes' with probability p and 'no' with probability $1 - p$. Therefore, $\mathcal{M}'$ can simulate the machine $\mathcal{M}$, by making a call to the oracle, for every computation step of $\mathcal{M}$. $\mathcal{M}$ takes a polynomial number of steps, thus, $\mathcal{M}'$ makes a polynomial number of oracle calls and, since the time schedule of the experiment that it uses as oracle is constant, it runs in polynomial time. □

Using the results, we obtain the following important result about generalized probabilistic machines.

Theorem 2.23. *The class of sets which are decidable in polynomial time by generalized probabilistic Turing machines with bounded error probability is exactly $BPP//\log\star$.*

Proof. From Theorems 2.21 and 2.22, we get that generalized probabilistic machines have exactly the same computational power than, for example, AD machines with 'yes' probability $F(\sigma) = \sigma$. By Corollary 2.16, we know that these machines decide exactly $BPP//\log\star$. □

Note that Theorem 2.23 could have been proved using the techniques of Sections 2.1 to 2.6. Although, we chose to do it this way , in order to highlight the similarities, in practical terms, between generalized probabilistic machines and the AD machines of this chapter.

After seeing that generalized probabilistic Turing machines can compute more in polynomial time than conventional probabilistic machines, one might find surprising that the real probabilities boost the computational power of Turing machines, since the machine does not have direct access to the probabilities of transition. Although, by making a guess, i.e., by a long sequence of tosses of its internal coin, the Turing machine can pass the probability of transition for its tape, approximating the real value with high probability.

[9] As F is one-to-one and $F([0, 1]) = [0, 1]$, we know that $F^{-1}(p)$ is well defined.

Recall that the transition function of a Turing machine is a function

$$\delta : Q \times \Sigma^{k+1} \to Q \times \Sigma^k \times \{R, N, L\}^{k+1},$$

where Q is the finite set of internal states, Σ is the working alphabet (finite), k is the number of working tapes and R, N and L correspond to the possible movements of the tape heads (R means moving one cell to the right, N means do not move and L means moving one cell to the left). To sum up, the transition function is finite. Therefore, we can assign different probabilities to the transitions of the Turing machine, obtaining a finite set of real numbers. The machines with different real probabilities of transition generalize our generalized probabilistic machines and have the same computational power. We know that every set in $BPP//\log\star$ is decided by a generalized probabilistic machine, thus, it is decided by a Turing machine with probabilities of transition that are real numbers. The upper bound can be proved the same techniques than Theorem 2.15 but the advice is given by shuffling the real numbers in the following way: we list the first bit of every real number, then the second bit, then the third, and so on. In other words, if the finite set of real numbers was $\{p_1, p_2, ..., p_k\}$, then the advice would be a logarithmic number of bits of $p_{1,1}p_{2,1}...p_{k,1}p_{1,2}p_{2,2}...p_{k,2}...$, where $p_{i,j}$ is the j-th bit of the infinite binary expansion of p_i.

Chapter 3

Random walks and inner structure of $BPP//\log\star$

3.1 Unidimensional random walk

3.1.1 The experiment

Despite all the applications already mentioned, in this chapter we present an experiment that fulfils the conditions of Chapter 2.

Consider the experiment of having a particle moving along an axis. The particle is sent from position $x = 0$ to position $x = 1$. Then, at each positive integer coordinate, the particle moves right, with probability σ, or left, with probability $1 - \sigma$, as outlined in Figure 3.1. If the particle ever returns to its initial position, $x = 0$, it is absorbed (it remains there).

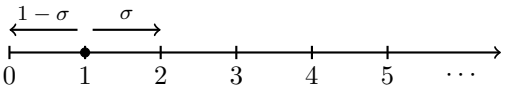

Figure 3.1: Random walk on the line with absorption at $x = 0$.

In this process, the particle takes steps of one unit, at time intervals also of one unit, thus, our experiment is a discrete unidimensional random walk with one absorbing barrier (see [40]).

In this chapter, we will study this experiment as an oracle to a Turing machine. Coupling discrete time experiments, like the random walk, with Turing machines is interesting, because, unlike experiments with continuous time, we can combine the Turing machine and the oracle in a much more elegant way. In this case, it is as if the oracle was also a mechanism, in which the transitions are the random steps.

We are interested in the probability that the particle is absorbed and we discuss it as in [34]. Let p_i be the probability of absorption (at $x = 0$) when the particle starts at $x = i$. In our model, the particle is launched from $x = 0$ but it only starts its random walk at $x = 1$, so, we want to know the value of p_1. It's

easy to see that

$$p_1 = (1-\sigma) + \sigma p_2,$$

because $1-\sigma$ is the probability that the particle is absorbed in one step (actually two, since the first step was from $x = 0$ to $x = 1$) and σp_2 represents the probability that the particle is absorbed later.

From $x = 2$, to be absorbed, the particle must initially move from $x = 2$ to $x = 1$ (not necessarily in one step), and then from $x = 1$ to $x = 0$ (again, not necessarily in one step). Both movements are made, independently, with probability p_1, thus, p_2 is just p_1^2. More generally, we have $p_k = p_1^k$. Therefore, the equation for the unidimensional random walk with absorption at $x = 0$ is given by

$$p_1 = (1-\sigma) + \sigma p_1^2,$$

and we have a quadratic equation in p_1, with solutions $p_1 = 1$ and $p_1 = \frac{1-\sigma}{\sigma}$. For $\sigma = \frac{1}{2}$, the solutions coincide and $p_1 = 1$. For $\sigma < \frac{1}{2}$, the second solution is impossible, because $\frac{1-\sigma}{\sigma} > 1$, so, we must have $p_1 = 1$. For $\sigma = 1$, the particle always moves to the right, so $p_1 = 0$. Thus, for the sake of continuity of p_1, for $\sigma > \frac{1}{2}$, we must choose $p_1 = \frac{1-\sigma}{\sigma}$. Consequently, we get

$$p_1 = \begin{cases} 1 & \text{if } \sigma \leq \frac{1}{2} \\ \frac{1-\sigma}{\sigma} & \text{if } \sigma > \frac{1}{2} \end{cases},$$

So, if $\sigma \leq \frac{1}{2}$, with probability 1, the particle always returns, although the number of steps taken is unbounded.

In Figure 3.2 we illustrate this situation, for the case $\sigma = 1/4$, giving the possible locations of the particle, and the respective probabilities, after the first steps.

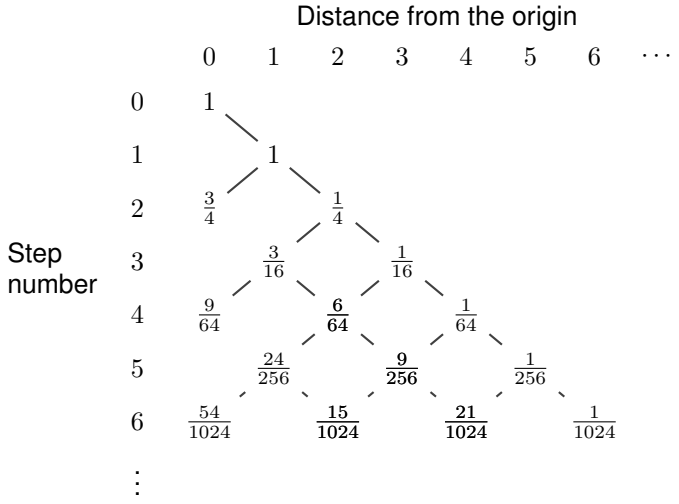

Figure 3.2: Diagram showing probabilities of the particle being at various distances from the origin, for the case of $\sigma = 1/4$.

3.1.2 Results

Consider a Turing machine coupled with a random walk experiment. This machine was introduced in [26]. To use the RWE as an oracle, we admit that the probability that the particle moves forward, σ, encodes some advice.

Unlike previously studied experiments, such as the scatter machines, both sharp and smooth cases (see [7], [1] and Section 1.1), and the balance experiments, both threshold and two-sided cases (see [10] and Section 2.7), the RWE does not need any parameters to be initialized, so, the Turing machine does not provide the oracle with any dyadic rational, it just "pulls the trigger" to start the experiment, i.e., the query corresponds to the impulse given to the particle at position $x = 0$ so it moves to $x = 1$. As we saw before, for every $\sigma \in (0, 1)$ the time that the particle takes to return is unbounded, thus, we need to define a time schedule. If the particle is absorbed during the time schedule, the finite control of the Turing machine changes to the 'yes' state, otherwise, the finite control changes to the 'no' state. Defining the time schedule is an important and complex matter. For a fixed $\sigma \in (0, \frac{1}{2})$ (in this interval we know that the particle always returns), depending on the time schedule being too small or too long, we can have always the same outcome ('yes' or 'no'). For now, let us consider that the time schedule is any constant.

So, we have an experiment with two possible outcomes, that does not need any initial information and with a constant time schedule. To fulfil the conditions of the oracles considered in Chapter 2, we only need to analyse the probability of 'yes'.

A path of the random walk is a possible sequence of moves that the particle makes that leads it to absorption. The path length is the number of steps that the particle took before the absorption. Note that there are no paths such that the length is an odd number.

Paths of the random walk along the positive x-axis with absorption at $x = 0$ are isomorphic to a specific set of well-formed sequences of parentheses. For instance, in a random walk of length 6, the particle could well behave as $((()))$ or $(()())$, where a movement to the right is represented by "$($" and a movement to the left is represented by "$)$". The first opening parenthesis corresponds to the first move of the particle from $x = 0$ to $x = 1$. The probability of answer in 6 steps is the sum of two probabilities corresponding to the two possible paths.

All paths of a certain length have the same probability, namely for even number n, the probability of each path of length n is

$$\sigma^{\frac{n}{2}-1}(1-\sigma)^{\frac{n}{2}}.$$

Therefore, we only need to know the number of possible paths for each length, i.e., the number of well-formed sequences of parentheses satisfying some properties.

The i-th Catalan number, given by

$$C_i = \frac{1}{i+1}\binom{2i}{i},$$

corresponds to the number of well-formed sequences of parentheses, i.e., sequences of parentheses with an equal number of left parentheses and right parentheses and such that any initial segment con-

tains at least as many left parentheses as right parentheses.

In [19], the authors generalize the Catalan numbers and prove the following result:

Proposition 3.1 (Blass and Braun [19])**.** *For integers $\ell \geq w \geq 0$, let X be the number of strings consisting of ℓ left and ℓ right parentheses, starting with w consecutive left parentheses, and having the property that every nonempty, proper, initial segment has strictly more left than right parentheses. Then*

$$X = \frac{w}{2\ell - w} \begin{pmatrix} 2\ell - w \\ \ell \end{pmatrix}$$

When $w = \ell = 0$, the undefined fraction $w/(2\ell - w)$ is to be interpreted as 1, since this gives the correct value $X = 1$, corresponding to the empty string of parentheses.

From this proposition, we derive the probability $q(t)$ that the particle is absorbed in even time $t+1$, for $t \geq 1$. It suffices to take $\ell = (t+1)/2$ and $w = 1$:

$$q(t) = \frac{1}{t} \binom{t}{\frac{t+1}{2}} (1-\sigma)^{\frac{t+1}{2}} \sigma^{\frac{t+1}{2}-1} .$$

Therefore, the probability that the particle is absorbed during the time schedule T is given by

$$\sum_{\substack{t=1 \\ t \ odd}}^{T-1} \frac{1}{t} \binom{t}{\frac{t+1}{2}} (1-\sigma)^{\frac{t+1}{2}} \sigma^{\frac{t+1}{2}-1} .$$

This is the probability of getting the outcome 'yes' from the oracle.

For analytical reasons, we will consider only $\sigma \in [\frac{1}{2}, 1]$, corresponding to a variation of p_1 from 1 to 0. Note that we could consider any interval contained in $[0, 1]$.

For every T, this probability is a function of σ that satisfies the desired conditions: $F \in C^1([\frac{1}{2}, 1])$, $F'(\sigma) \neq 0, \forall \sigma \in [\frac{1}{2}, 1]$ and we can compute n bits of F in time $O(2^n)$. Figure 3.3 allows us to understand the behaviour of the probability $F(\sigma)$ as a function of σ.

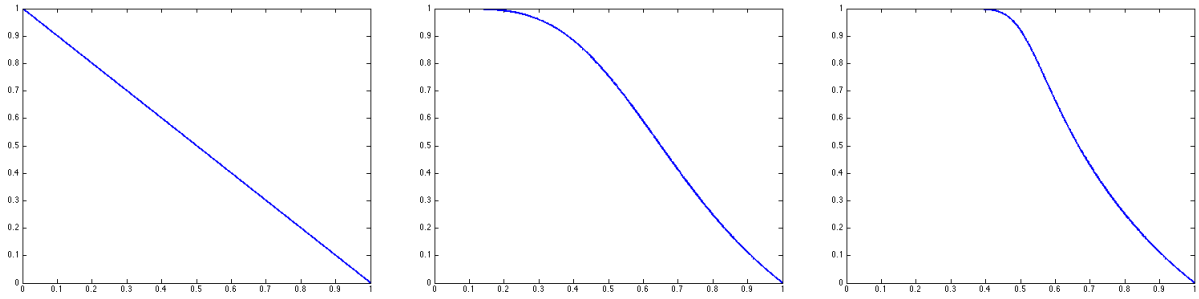

Figure 3.3: Graphs of $F(\sigma)$ for $T = 2$, $T = 10$ and $T = 100$.

We can see that as T approaches infinity, $F(\sigma)$ approaches the probability p_1 that the particle is absorbed, which makes sense, since p_1 represents the probability of absorption with unbounded time.

As we have seen above, RWE is an experiment with two possible outcomes, that does not need any initial information, with a constant time schedule and with the desired analytical properties of the probability function of the outcome 'yes'. Therefore, RW machines are in the conditions of our AD machines from Chapter 2.

Applying to RW machines the results of Chapter 2, we get the following theorems:

Theorem 3.2. *Every set in $BPP//\log\star$ is decidable by a RW machine in polynomial time.*

Theorem 3.3. *Every set decided by a RW machine in polynomial time is in $BPP//\log\star$.*

With these two theorems, we get the following result:

Corollary 3.4. *The class of sets which are decidable in polynomial time by deterministic random walk machines is exactly $BPP//\log\star$.*

3.2 Counting the calls to the oracle

In this section we focus in one computational resource of the RW machine, the number of calls to the oracle, and use it to build an hierarchy in $BPP//\log\star$. The number of times that the Turing machine consults the oracle has already been study as a resource. For example, in [4, 27, 17], relativisation problems are studied, bounding the number of oracle calls. Although, here, the oracle we are considering is not the classical one, but the random walk experiment.

3.2.1 The computational resources trade-off

Consider that we have an upper bound in the number of particles that we can launch, i.e., a bound in the number of oracle calls that our RW machine can make. So, in order to reach the desired precision when obtaining the bits of σ, we must determine the time schedule, that is, the time that we wait for the particles to return. In Chapter 2, we had any fixed time schedule and, determining how many queries we should make, we could get the desired precision. Here, since we cannot perform an arbitrary number of oracle calls anymore (with the obvious bound of the total time that the machine had to execute its computations), we need to consider the time schedule. To do so, we study how the probability of absorption varies, as a function of the time schedule T.

As we have seen before, the probability of absorption in time T is given by

$$F(\sigma,T)=\sum_{\substack{t=1\\ t\ odd}}^{T-1}\frac{1}{t}\binom{t}{\frac{t+1}{2}}(1-\sigma)^{\frac{t+1}{2}}\sigma^{\frac{t+1}{2}-1}\ .$$

For now, it suffices to note that, as T increases, $F(\sigma,T)$ increases as well, since the sum includes more terms, and all terms are positive. This result is intuitive, since the longer we wait, the more likely it is that the particle is absorbed. With this in mind and based on Lemma 2.2, we can state and prove the following:

Lemma 3.5. *A RW machine that can make up to $\xi(n)$ calls to the oracle, given input x of size $|x| = n$, can read $O(\log(\xi(n)))$ bits of the unknown parameter σ in polynomial time in n.*

Proof. We know that each particle has probability of absorption in time T of $F(\sigma, T)$. Thus, if we make $\xi(n)$ oracle calls, where n is the input size, the number of times that the experiment returns 'yes', α, is a random variable with binomial distribution. Let us consider $X = \alpha/\xi(n)$, the random variable that represents the relative frequency of absorption ('yes'). We have the expected value $\mathbb{E}[X] = \mathbb{E}[\alpha]/\xi(n) = \xi(n)F(\sigma,T)/\xi(n) = F(\sigma,T)$ and the variance $\mathbb{V}[X] = \mathbb{V}[\alpha]/\xi(n)^2 = \xi(n)F(\sigma,T)(1-F(\sigma,T))/\xi(n)^2 = F(\sigma,T)(1-F(\sigma,T))/\xi(n)$. Chebyshev's inequality states that, for every $\delta > 0$,

$$P(|X - \mathbb{E}[X]| > \delta) \leq \frac{\mathbb{V}[X]}{\delta^2} \leq \frac{F(\sigma,T)(1-F(\sigma,T))}{\xi(n)\delta^2} \leq \frac{F(\sigma,T)}{\xi(n)\delta^2}.$$

By setting $\delta = 2^{-k-5}$, where k is the number of bits of σ that we intend to read, we get

$$P(|X - F(\sigma,T)| > 2^{-k-5}) \leq \frac{2^{2k+10}F(\sigma,T)}{\xi(n)}$$

and if we want an error probability of at most γ, we get

$$\frac{2^{2k+10}F(\sigma,T)}{\xi(n)} \leq \gamma.$$

Applying logarithms to this expression, we get

$$2k + 10 + \log(F(\sigma,T)) - \log(\xi(n)) \leq \log(\gamma),$$

therefore,

$$k \leq \frac{\log(\xi(n)) + \log(\gamma) - \log(F(\sigma,T)) - 10}{2}.$$

As $F(\sigma,T) \leq 1$, its logarithm is not positive, thus, it is more intuitive to have

$$k \leq \frac{\log(\xi(n)) + \log(\gamma) + \log\left(\frac{1}{F(\sigma,T)}\right) - 10}{2}.$$

For every σ, as T increases, the probability of absorption in time T, $F(\sigma,T)$, obviously increases, approaching 1. Thus the fraction $1/F(\sigma,T)$ decreases, such as its logarithm, so, contrary to what one might first think, for each input word w of size n and its corresponding number of particle releases, the longer we wait for the particles to return, the less precision we can obtain for the bits of σ. After all, this makes sense, since if we wait too long, we will lose information about the instant when the particle was absorbed. So, we can establish that, in every oracle call, the machine will wait exactly two time steps for the particle to return (to be absorbed, the particle has to make an even number of moves). With time schedule $T = 2$, the RW experiment becomes much more simple, since $F(\sigma,2) = (1-\sigma)$. Now, we have

$$P(|X - (1-\sigma)| > 2^{-k-5}) \leq \gamma,$$

for $k \in O(\log(\xi(n)))$. Furthermore, $|(1-X)-\sigma| = |X-(1-\sigma)|$, so,

$$P(|(1-X)-\sigma| > 2^{-k-5}) \leq \gamma,$$

and we used $1-X$ to estimate σ. □

With this result, we are led to think that there may exists a hierarchy of classes identified by RW machines, induced by the number of queries to the oracle that are allowed. In the next section, we try to associate a non-uniform complexity class with a RW machine, for a given number of oracle calls.

3.2.2 Lower and upper bounds

In Chapter 2, we have studied one method for encoding advice functions $f \in \log\star$ into real numbers. Here, we show how we can encode other types of advice functions, in order to compare RW machines with other non-uniform classes.

We define the iterated logarithmic functions $\log^{(k)}(n)$:

- $\log^{(0)}(n) = n$;
- $\log^{(k+1)}(n) = \log(\log^{(k)}(n))$.

Similarly, we define the iterated exponential $\exp^{(k)}(n)$:

- $\exp^{(0)}(n) = n$;
- $\exp^{(k+1)}(n) = 2^{\exp^{(k)}(n)}$.

The iterated exponential is a well known function, since it appears as a bound on the number of computation steps for elementary functions, for example in [28] and [35].

It is trivial to see that, for every k, $\log^{(k)}$ and $\exp^{(k)}$ are inverse functions. Let $\log^{(k)}$ also denote the class of advice functions f such that $|f(n)| \in O(\log^{(k)}(n))$.

Recall the encoding presented in Section 2.1. Keeping the encoding $c(w)$ for a single word w, we define the encoding $y(f) = \lim y(f)(n)$ for an advice function $f \in \log^{(k)}\star$ in the following way:

- $y(f)(0) = 0.c(f(0))$
- if $f(n+1) = f(n)s$, then

$$y(f)(n+1) = \begin{cases} y(f)(n)c(s) & \text{if } n+1 \text{ is not of the form } \exp^{(k)}(m) \\ y(f)(n)c(s)001 & \text{if } n+1 \text{ is of the form } \exp^{(k)}(m) \end{cases}$$

So, for example, if we want to encode a function $f \in \log\log\star$, we just have to place the separator 001 when $n+1$ is of the form 2^{2^m}, for some $m \in \mathbb{N}$.

The encoding presented in Section 2.1, for functions in $\log\star$, is the particular case of this one when $k = 1$.

For every k and for every $f \in \log^{(k)}\star$, we have that $y(f) \in \mathcal{C}_3$. Also, for every n, in order to extract the value of $f(n)$, we only need to find the number $m \in \mathbb{N}$ such that $\exp^{(k)}(m-1) < n \leq \exp^{(k)}(m)$ and then read $y(f)$ in triplets, until we find the $(m+1)$-th separator. Then, it is only needed to ignore the separators and replace each 100 triplet by 0 and each 010 triplet by 1. Since $f \in \log^{(k)}\star$, we know that $|f(\exp^{(k)}(m))| = O(\log^{(k)}(\exp^{(k)}(m))) = O(m)$, so, we need $3O(m) + 3(m+1) = O(m)$ bits to get the value of $f(\exp^{(k)}(m))$ and, consequently, $O(\log^{(k)}(n))$ bits to get the value of $f(n)$.

Now that we know how to encode several types of advice functions, and having in mind Lemma 3.5, we can state and prove the following lower bound result:

Theorem 3.6. *For every k, every set in $BPP//\log^{(k+1)}\star$ is decidable by a RW machine that can make up to $\xi(n) = O(\log^{(k)}(n))$ calls to the oracle, where n is the input size, in polynomial time.*

Proof. Let A be an arbitrary set in $BPP//\log^{(k+1)}\star$ and $\mathcal{M}$ a probabilistic Turing machine with advice $f \in \log^{(k+1)}\star$, which decides A in polynomial time with error probability bounded by $\gamma_1 \in (0, 1/2)$.

Let $\mathcal{M}'$ be a RW machine with unknown parameter $y(f)$, where $y(f)$ represents the encoding of f into Cantor set C_3 and let γ_2 be a positive real number such that $\gamma_1 + \gamma_2 < 1/2$. Let w be a word such that $|w| \leq n$. We know, by Lemma 3.5, that $\mathcal{M}'$ can estimate $O(\log(\xi(n))) = O(\log(\log^{(k)}(n))) = O(\log^{(k+1)}(n))$ bits of $y(f)$, and, thus, read $f(n)$, in polynomial time, with an error probability bounded by γ_2.

Now we need to simulate independent unbiased coin tosses. Let γ_3 be such that $\gamma_1 + \gamma_2 + \gamma_3 < 1/2$. According to the proof of Lemma 3.5, in every oracle call, $T = 2$, thus we have $P(\text{`}yes\text{'}) = 1 - \sigma$ and $P(\text{`}no\text{'}) = \sigma$, so, by Lemma 2.3, $\mathcal{M}'$ can produce a sequence of fair coin tosses in time linear on the size of the sequence, to within a probability of γ_3, thus it can behave probabilistically. Therefore, $\mathcal{M}'$ can decide A in polynomial time, with error probability bounded by $\gamma_1 + \gamma_2 + \gamma_3 < 1/2$. □

In order to state and prove the upper bound theorem, we need the following auxiliary lemma:

Lemma 3.7. *Let $\mathcal{M}$ be a RW machine with unknown parameter σ that can make up to $\xi(n)$ calls to the oracle, where n is the input size, deciding some set A with error probability bounded by $\gamma < 1/4$. Let $\mathcal{M}'$ be an identical AD machine, with the exception that the unknown parameter is $\tilde{\sigma}$ and the probability that the oracle returns 'yes' is given by $\tilde{F}$. If*

$$|F(\sigma, T) - \tilde{F}(\tilde{\sigma}, T)| < \frac{1}{8\xi(n)},$$

where $F(\sigma, T)$ is the probability of absorption from the random walk, then, for any word of size $\leq n$, the probability of $\mathcal{M}'$ making an error when deciding A is $\leq 3/8$.

Proof. We know that $\mathcal{M}$ and $\mathcal{M}'$ make at most $\xi(n)$ calls to the oracle, so the query tree associated to both, T has maximum depth $\xi(n)$. Let w be a word such that $|w| \leq n$. Let $D \in \rho(T)$ be the assignment of probabilities to the edges of T corresponding to the unknown parameter σ and 'yes' probability $F(\sigma, T)$ and $D' \in \rho(T)$ the probabilities given by unknown parameter $\tilde{\sigma}$ and 'yes' probability $\tilde{F}(\tilde{\sigma}, T)$.

As $|F(\sigma,T) - \tilde{F}(\tilde{\sigma},T)| < 1/8\xi(n)$, the difference between any particular probability is at most

$$k = \frac{1}{8\xi(n)}.$$

To calculate the probability of an incorrect result for the machine with unknown parameter $\tilde{\sigma}$ and 'yes' probability given by $\tilde{F}$, we have to consider two different cases.

- $w \notin A$: In this case, an incorrect result corresponds to $\mathcal{M}'$ accepting w. The probability of acceptance for $\mathcal{M}'$ is

$$\begin{aligned} P_A(T,D') &\leq P_A(T,D) + |P_A(T,D') - P_A(T,D)| \\ &\leq \gamma + \xi(n)k \\ &\leq \gamma + \xi(n)\frac{1}{8\xi(n)} \\ &= \frac{1}{4} + \frac{1}{8} = \frac{3}{8} \end{aligned}$$

- $w \in A$: In this case, an incorrect result corresponds to $\mathcal{M}'$ rejecting w. The probability of rejection for $\mathcal{M}'$ is

$$\begin{aligned} P_R(T,D') &\leq P_R(T,D) + |P_R(T,D') - P_R(T,D)| \\ &\leq \gamma + \xi(n)k \\ &\leq \gamma + \xi(n)\frac{1}{8\xi(n)} \\ &= \frac{1}{4} + \frac{1}{8} = \frac{3}{8} \end{aligned}$$

In both cases, the error probability is bounded by $3/8$. □

Corollary 3.8. *Let $\mathcal{M}$ be a RW machine with unknown parameter σ that can make up to $\xi(n)$ calls to the oracle, where n is the input size, deciding some set in time $t(n)$ with error probability bounded by $\gamma < 1/4$. Let $\mathcal{M}_n$ be an identical AD machine, also with unknown parameter σ, but with the exception that the probability that the oracle returns 'yes' is given by $F(\sigma,T)|_{\log \xi(n)+3}$, where $F(\sigma,T)$ is the probability of absorption from the random walk. Then, $\mathcal{M}_n$ decides the same set as $\mathcal{M}$, also in time $t(n)$, but with error probability bounded by $3/8$.*

Now we state and prove the upper bound result.

Theorem 3.9. *For every k, every set decided by a RW machine that can make up to $\xi(n) = O(\log^{(k)}(n))$ calls to the oracle, where n is the input size, in polynomial time is in $BPP//\log^{(k+1)}\star$.*

Proof. Let A be a set decided by a RW machine $\mathcal{M}$ with unknown parameter σ, that can make up to $\xi(n) = O(\log^{(k)}(n))$ calls to the oracle, in polynomial time $t(n)$ and with error probability bounded by 1/4.

We want to construct a probabilistic Turing machine with advice, $\mathcal{M}'$, which decides A.

Let us use the advice function $f(n) = F(\sigma, T)\downarrow_{\log \xi(n)+3}$. We have $f \in \log^{(k+1)}\star$

By Corollary 3.8, we know that an AD machine with 'yes' probability $f(n)$ decides the same for words of size $\leq n$ than $\mathcal{M}$, but with error probability $\leq 3/8$.

Thus, M' must simulate the behaviour of an AD machine with unknown parameter σ and with 'yes' probability $f(n)$. We do that with fair coin tosses.

$f(n) = F(\sigma)\downarrow_{\log \xi(n)+3}$ is a dyadic rational with denominator $2^{\log \xi(n)+3}$. Thus, $m = 2^{\log \xi(n)+3} f(n) \in [0, 2^{\log \xi(n)+3})]$ is an integer. Now, make $k = \log \xi(n) + 3$ fair coin tosses, interpreting the results τ_i (taken in order) as either 0 or 1. Then, test if $\tau_1\tau_2...\tau_k < m$, where $\tau_1\tau_2...\tau_k$ is viewed as the binary representation of an integer. If the test is true, return 'yes', otherwise return 'no'. The probability of returning 'yes' is $m/2^k = f(n)$, as required. The time taken is polynomial in n. □

3.2.3 The hierarchy

From Theorem 3.6 and Theorem 3.9, we get the following corollary:

Corollary 3.10. *The class of sets which are decidable in polynomial time by deterministic random walk machines that can make up to $O(\log^{(k)}(n))$ calls to the oracle, where n is the input size, is exactly $BPP//\log^{(k+1)}\star$.*

As we want the RW machines to run in polynomial time, the maximum number of oracle calls that we can allow is polynomial. For that bound, the corresponding class is $BPP//\log\star$.

Thus, if we restrict more and more the number of queries to the oracle, we can obtain a fine structure of $BPP//\log\star$. Observe that if k is a very large number, the machine is allowed to make only few calls to the oracle, but the advice is smaller, so the number of bits that the machine needs to read is also smaller.

To establish the hierarchy, since we are working with non-uniform complexity classes, we need some kind of relation between advice classes. Let us explore some properties of these classes, that have already been explained in [5], [39] and [8].

If $f : \mathbb{N} \to \Sigma^*$ is an advice function, then we use $|f|$ to denote its size, i.e., the function $|f| : \mathbb{N} \to \mathbb{N}$ such that $|f|(n) = |f(n)|$, for every $n \in \mathbb{N}$. For a class of functions, $\mathcal{F}$, $|\mathcal{F}| = \{|f| : f \in \mathcal{F}\}$.

The following definition is an adaptation from [5], [39] and [8].

Definition 3.11. A class of advice functions is said to be a class of reasonable advice functions if:

(a) for every $f \in \mathcal{F}$, $|f|$ is computable in polynomial time;

(b) for every $f \in \mathcal{F}$, $|f|$ is bounded by a polynomial;

(c) for every $f \in \mathcal{F}$, $|f|$ is increasing;

(d) $|\mathcal{F}|$ is closed under addition and multiplication by positive integers;

(e) for every polynomial p of positive integer coefficients and every $f \in \mathcal{F}$, there exists $g \in \mathcal{F}$ such that $|f| \circ p \leq |g|$.

Definition 3.12. Let r and s be two total functions. We say that $r \prec s$ if $r \in o(s)$.

Let $\mathcal{F}$ and $\mathcal{G}$ be classes of advice functions. We say that $\mathcal{F} \prec \mathcal{G}$ if there exists a function $g \in \mathcal{G}$ such that, for every $f \in \mathcal{F}$, $|f| \prec |g|$.

Focusing on the classes that we are interested in, we have $\log^{(k+1)} \prec \log^{(k)}$, for all $k \geq 0$. Now, we just need to know the relation between the non-uniform complexity classes of BPP, induced by the relation $\prec$ in the advice classes.

To be able to present such result, we need to introduce the concept of tally set and its characteristic function.

Definition 3.13. A set is said to be tally if it is a language over an alphabet of a single symbol (we take this alphabet to be $\{0\}$).

Definition 3.14. Let the set of finite sequences over the alphabet Σ be ordered first by size, then alphabetically. The characteristic function of a language $A \subseteq \Sigma^*$ is the unique infinite sequence $\chi_A \colon \mathbb{N} \to \{0,1\}$ such that, for all n, $\chi_A(n)$ is 1 if, and only if, the n-th word in that order is in A.

The characteristic function of a tally set A is a sequence where the i-th bit is 1 if, and only if, the word 0^i is in A.

The following theorem is a new result, but it and its proof are inspired in [5], [39] and [8], where it is proved for the deterministic case. Here we focus on the probabilistic case.

Theorem 3.15. *If $\mathcal{F}$ and $\mathcal{G}$ are two classes of reasonable sublinear advice functions*[1] *such that $\mathcal{F} \prec \mathcal{G}$, then $BPP//\mathcal{F} \subsetneq BPP//\mathcal{G}$.*

Proof. Trivially, $BPP//\mathcal{F} \subseteq BPP//\mathcal{G}$.

Let $linear$ be the set of advice functions of size linear in the size of the input and $\eta.linear$ be the class of advice functions of size ηn, where n is the size of the input and η is a number such that $0 < \eta < 1$.There is an infinite sequence γ whose set of prefixes is in $BPP//linear$ but not in $BPP//\eta.linear$ for some η sufficiently small.[2] Let $g \in \mathcal{G}$ be a function such that, for every $f \in \mathcal{F}$, $|f| \prec |g|$. We prove that there is a set in $BPP//g$ that does not belong to $BPP//f$, for any $f \in \mathcal{F}$.

A tally set T is defined in the following way: for each $n \geq 1$,

$$\beta_n = \begin{cases} \gamma|_{|g|(n)}\ 0^{n-|g|(n)} & \text{if } |g|(n) \leq n \\ 0^n & \text{otherwise} \end{cases}.$$

T is the tally set with characteristic string $\beta_1\beta_2\beta_3$.... With advice $\gamma_{|g|(n)}$, it is easy to decide T, since we can reconstruct the sequence $\beta_1\beta_2...\beta_n$, with $\frac{n^2+n}{2}$bits, and then we just have to check if its n-th bit is 1 or 0. So, $T \in P/g \subseteq BPP//g$.

We prove that the same set does not belong to $BPP//f$. Suppose that some probabilistic Turing machine $\mathcal{M}$ with advice f , running in polynomial time, decides T with probability of error bounded by

[1] $\mathcal{F}$ is a class of reasonable sublinear advice functions if it is a class of reasonable advice functions such that, for every $f \in \mathcal{F}$, $|f| \in o(n)$.

[2] We can take for γ the Chaitin Omega number, Ω (See Appendix C).

$2^{-\log(4|g|(n))} = \frac{1}{4|g|(n)}$ (we can use this value for the error probability, due to Proposition B.2). Since $|f| \in o(|g|)$, then, for all but finitely many n, $|f|(n) < \eta|g|(n)$, for arbitrarily small η, meaning that we can compute, for all but finitely many n, $|g|(n)$ bits of γ using an advice of length $\eta.|g|(n)$, contradicting the fact that the set of prefixes of γ is not in $BPP//\eta.linear$. The reconstruction of the binary sequence $\gamma\downarrow_{|g|(n)}$ is provided by the following procedure:

$\mathcal{M}'$ procedure:

begin

input n;

$x :=\lambda$;

compute $|g|(n)$;

for $i := \frac{n^2-n}{2}$ to $\frac{n^2-n}{2} + |g|(n)$ do

query 0^i to T using advice $f(i)$;

if "YES" then $x := x1$; else $x := x0$;

end for;

output x;

end.

The queries are made simulating machine $\mathcal{M}$ which is a probabilistic Turing machine with error probability bounded by $2^{-\log(4|g|(n))} = \frac{1}{4|g|(n)}$. Thus, the probability of error of $\mathcal{M}'$ is bounded by

$$\frac{1}{4|g|(\frac{n^2-n}{2})} + ... + \frac{1}{4|g|(\frac{n^2-n}{2} + |g|(n))}.$$

As $|g|$ is increasing, the error probability is bounded by

$$\frac{1}{4|g|(\frac{n^2-n}{2})} \times |g|(n),$$

which, for $n \geq 3$ is less or equal than

$$\frac{1}{4|g|(n)} \times |g|(n) = \frac{1}{4}.$$

□

As we are considering prefix advice classes, it is useful to derive the following corollary:

Corollary 3.16. *If $\mathcal{F}$ and $\mathcal{G}$ are two classes of reasonable sublinear advice functions such that $\mathcal{F} \prec \mathcal{G}$, then $BPP//\mathcal{F}\star \subsetneq BPP//\mathcal{G}\star$.*

Proof. The proof of 3.15 is also a proof that $BPP//\mathcal{F} \subsetneq BPP//\mathcal{G}\star$, because the advice function used is $\gamma\downarrow_{|g|(n)}$, which is a prefix advice function. Since $BPP//\mathcal{F}\star \subseteq BPP//\mathcal{F}$, the statement follows. □

We have already seen that, for all $k \geq 0$, $\log^{(k+1)} \prec \log^{(k)}$. In particular, this is true for $k \geq 1$ and we have the following infinite descending chain

$$\cdots \prec \log^{(4)} \prec \log^{(3)} \prec \log^{(2)} \prec \log .$$

Therefore, by Corollary 3.16, we have the descending chain of sets

$$\cdots \subsetneq BPP//\log^{(4)}\star \subsetneq BPP//\log^{(3)}\star \subsetneq BPP//\log^{(2)}\star \subsetneq BPP//\log\star,$$

that coincide with the sets decided by RW machines that can make up to

$$\cdots \subsetneq O(\log^{(3)}(n)) \subsetneq O(\log^{(2)}(n)) \subsetneq O(\log(n)) \subsetneq O(n)$$

calls to the oracle, respectively, where n is the input size.

Note that, although we have defined $\log^{(0)}(n) = n$, the non-uniform complexity class $BPP//\log\star$ corresponds to the class of sets decided by RW machines with a polynomial number of oracle calls, and not only $O(n)$ calls.

In [8], although the probabilistic case is not considered, the descending chain of iterated logarithms is presented and completed with the class $\log^{(\omega)} = \bigcap_{k\geq 1} \log^{(k)}$, which is not trivial, since the function $\log^*$, defined by $\log^*(n) = \min\{k : \log^{(k)}(n) \leq 1\}$, is in $\log^{(\omega)}$ (See [25]). In fact, the descending chain can be continued even further and it is an open problem to know where it stops. In [25], Costa examines more closely the classes of this chain. The descending chain presented in [8] and [25] is

$$\log^{(2\omega)} \prec ... \prec \log^{(\omega+2)} \prec \log^{(\omega+1)} \prec \log^{(\omega)} \prec ... \prec \log^{(3)} \prec \log^{(2)} \prec \log,$$

where, for $k \geq 1$, $\log^{(\omega+k)}$ is the class generated by $\log^{(k)} \circ \log^*$ and $\log^{(2\omega)} = \bigcap_{k\geq 1} \log^{(\omega+k)}$. [3]

So, by Theorem 3.15 and Corollary 3.16, we have the following descending chains of non-uniform complexity classes:

$$BPP//\log^{(2\omega)} \subsetneq \cdots \subsetneq BPP//\log^{(\omega+1)} \subsetneq BPP//\log^{(\omega)} \subsetneq \cdots \subsetneq BPP//\log^{(2)} \subsetneq BPP//\log$$

and

$$BPP//\log^{(2\omega)}\star \subsetneq \cdots \subsetneq BPP//\log^{(\omega+1)}\star \subsetneq BPP//\log^{(\omega)}\star \subsetneq \cdots \subsetneq BPP//\log^{(2)}\star \subsetneq BPP//\log\star.$$

However, this hierarchy is independent of RW machines, since we only have proved the correspondence for advice classes of the form $\log^{(k)}$, with $k \in \mathbb{N}$.

[3] The chain could be continued in this way, until $\log^{(\omega^2)} = \bigcap_{k\geq 1} \log^{(k\omega)}$. In [25], we can find an interesting discussion about how it could be continued beyond that.

Chapter 4

Conclusion

To complete this work, we summarize the results achieved and point some directions to a future work.

4.1 Results

Conventionally, the oracle is an external device that the Turing machine can consult in one time step and that contains either non-computable information, or just computable information provided to speed up the computations. This classical oracle is essentially a set. Although, if we consider replacing the oracle by an analogue device, such as a physical experiment, the oracle consultation is not any more an one time step computation. Instead, it is a device that takes time to be consulted and it can only be consulted up to some accuracy. The communication between the Turing machine and the physical oracle is now something complex, and, as such, it may involve errors. There are three protocols that define how the information that is exchanged between the machine and the oracle is processed: infinite precision, unbounded precision and fixed precision. In this thesis, although we have worked mainly with a slightly different type of oracle, we have abstracted some of the work previously done in the case of fixed precision.

We considered a Turing machine coupled with a generic abstract oracle, with two possible outcomes, e.g. 'yes' or 'no', and such that no parameters are needed to be given in order to initialize the experiment, and, if the query time is less than the scheduled time, then the finite control of the Turing machine changes to the 'yes' state; otherwise the finite control changes to the 'no' state. We established some conditions on this machine, like a constant time schedule and some analytical properties of the distribution $F(\sigma)$ that corresponds to the probability of getting the outcome 'yes'. Then, we proved lower and upper bounds on its computational power, obtaining, in both cases, the non-uniform class $BPP//\log\star$.

As we are trying to establish as lower bound the non-uniform complexity class $BPP//\log\star$, we must find a way to encode the advice information into the parameter σ. In order to prove the lower bound, we used the Cantor set $\mathcal{C}_3$ to encode the advice information into the parameter σ, as in [7] and [10]. Then, we used some probabilistic and some numerical methods, to prove that our AD machine could read the

advice from the unknown parameter. After that, we showed that the machine could use the oracle to simulate fair coin tosses, so it could behave probabilistically.

To prove the upper bound, we focused on probabilistic trees, clarifying some of the results that had already been proved in [9] and [10]. Then, we just had to prove that a small variation in the probability of 'yes' did not affect the decision of the machine, concluding that a number of bits of the probability of 'yes' logarithmic in the size of the input was sufficient to preserve the decision of the machine.

After proving that our AD machines compute exactly $BPP//\log\star$, we used this result to reconstruct the proofs of the lower bound theorems for the broken balance machine (see [10, 37]) and for the balance scale machine (see [11, 37]).

Then, we studied a generalization of the probabilistic machine, introducing the generalized probabilistic machine which is a probabilistic machine such that the transition probability p takes a real value in $(0, 1)$. Showing that generalized probabilistic machines have the same computational power than our AD machines, we proved that generalized probabilistic machines compute exactly $BPP//\log\star$, too.

In Chapter 3, we concretized our AD machine, with the random walk experiment. We made an analogy between paths of the random walk and the well formed sequences of parentheses, and used some results about a generalization of Catalan numbers, to obtain that the probability that one particle is absorbed during the scheduled time T is given by

$$\sum_{\substack{t=1 \\ t\ odd}}^{T-1} \frac{1}{t}\binom{t}{\frac{t+1}{2}}(1-\sigma)^{\frac{t+1}{2}}\sigma^{\frac{t+1}{2}-1}\ .$$

This is the probability of getting the outcome 'yes' from the RWE oracle.

With the RWE in mind, we proved the class of sets which are decidable in polynomial time by deterministic random walk machines that can make up to $O(\log^{(k)}(n))$ calls to the oracle, where n is the input size, is exactly $BPP//\log^{(k+1)}\star$, where the iterated logarithmic functions $\log^{(k)}(n)$ is such that:

- $\log^{(0)}(n) = n$,
- $\log^{(k+1)}(n) = \log(\log^{(k)}(n))$,

and $\log^{(k)}$ is the class of advice functions f such that $|f(n)| \in O(\log^{(k)}(n))$.

Thus, we saw that random walk machines with a bounded number of oracle calls could induce a fine structure of $BPP//\log\star$. Then, we presented various definitions of concepts related to non-uniform complexity classes, such as the concept of reasonable advice functions, the relation $\prec$ between advice classes or the characteristic function of a tally set. All these tools were needed to prove the important result that states that, if $\mathcal{F}$ and $\mathcal{G}$ are two classes of reasonable sublinear advice functions such that $\mathcal{F} \prec \mathcal{G}$, then $BPP//\mathcal{F} \subsetneq BPP//\mathcal{G}$. Although this result was already discussed for the deterministic case in [5, 39, 8], and Beggs et al. presented a similar hierarchy of deterministic classes in [8], the probabilistic case of the theorem and the hierarchy remained unproven. In this work, we proved the

probabilistic case of the theorem and presented the hierarchy that we found in $BPP//\log\star$:

$$\cdots \subsetneq BPP//\log^{(4)}\star \subsetneq BPP//\log^{(3)}\star \subsetneq BPP//\log^{(2)}\star \subsetneq BPP//\log\star,$$

that coincide with the sets decided by RW machines that that can make up to

$$\cdots \subsetneq O(\log^{(3)}(n)) \subsetneq O(\log^{(2)}(n)) \subsetneq O(\log(n)) \subsetneq O(n)$$

calls to the oracle, respectively, where n is the input size.

Note that, although we have defined $\log^{(0)}(n) = n$, the non-uniform complexity class $BPP//\log\star$ corresponds to the class of sets decided by RW machines with a polynomial number of oracle calls, and not only $O(n)$ calls.

With the descendent chain of advice classes presented in [8] and [25], we continued our descendent chain of non-uniform classes, although without knowing if there is a correspondence with the classes decided by RW machines:

$$BPP//\log^{(2\omega)}\star \subsetneq \cdots \subsetneq BPP//\log^{(\omega+1)}\star \subsetneq BPP//\log^{(\omega)}\star \subsetneq \cdots \subsetneq BPP//\log^{(2)}\star \subsetneq BPP//\log\star.$$

4.2 Future work

There are a lot of issues that we left behind, some more relevant than other.

First, once we used the random walk experiment to illustrate the AD machine that we presented, one can think of continuing to explore that experiment as an oracle to Turing machines, modifying some details. We can think about the behaviour of the oracle. Instead of returning only 'yes' or 'no' if the particle returns, or not, during the scheduled time, the Turing machine could receive from the analogue device the time that the particle took to arrive. Thus, we could not only do frequency analysis, but also other statistical and probabilistic reasoning. We can also consider modifying the experiment itself, and use high-dimensional random walks, as well as continuous random walks, and study if those experiments origin different results from the ones we obtained.

Thinking about the hierarchy that we presented, we said that the descent chain of non-uniform classes can be continued, considering $BPP//\log^{(\omega)}\star$, where $\log^{(\omega)} = \bigcap_{k\in\mathbb{N}} \log^{(k)}$ and that it can it be continued even further. However, we do not know if there is a correspondence between these complexity classes and the classes decided by RW machines with bounded number of oracle calls, since we only proved the correspondence for advice classes of the form $\log^{(k)}$, with $k \in \mathbb{N}$. One of the questions that we left for future thinking is how we could encode a function $f \in \log^{(\omega)}\star$ into a real number.

References

[1] Tânia Ambaram, Edwin Beggs, José Félix Costa, Diogo Poças, and John V. Tucker. An analog-digital model of computation. 2015. Submitted.

[2] Tânia Filipa Nascimento Ambaram. Theory of two-sided experiments. Master's thesis, Instituto Superior Técnico, July 2014.

[3] José Luis Balcázar, Josep Días, and Joaquim Gabarró. *Structural Complexity I*. Springer-Verlag, 2nd edition, 1988, 1995.

[4] José Luis Balcázar, Josep Días, and Joaquim Gabarró. *Structural Complexity II*. Springer-Verlag, 1990.

[5] José Luis Balcázar, Ricard Gavaldà, and Hava T. Siegelmann. Computational power of neural networks: a characterization in terms of kolmogorov complexity. *Information Theory, IEEE Transactions on*, 43(4):1175–1183, Jul 1997.

[6] José Luis Balcázar and Montserrat Hermo. The structure of logarithmic advice complexity classes. *Theoretical Computer Science*, 207(1):217–244, 1998.

[7] Edwin Beggs, José Félix Costa, Bruno Loff, and John V. Tucker. Computational complexity with experiments as oracles. *Proceedings of the Royal Society, Series A (Mathematical, Physical and Engineering Sciences)*, 464(2098):2777–2801, 2008.

[8] Edwin Beggs, José Félix Costa, Bruno Loff, and John V. Tucker. Oracles and advice as measurements. In Cristian S. Calude, José Félix Costa, Rudolf Freund, Marion Oswald, and Grzegorz Rozenberg, editors, *Unconventional Computation (UC 2008)*, volume 5204 of *Lecture Notes in Computer Science*, pages 33–50. Springer-Verlag, 2008.

[9] Edwin Beggs, José Félix Costa, Bruno Loff, and John V. Tucker. Computational complexity with experiments as oracles II. Upper bounds. *Proceedings of the Royal Society, Series A (Mathematical, Physical and Engineering Sciences)*, 465(2105):1453–1465, 2009.

[10] Edwin Beggs, José Félix Costa, Diogo Poças, and John V. Tucker. Oracles that measure thresholds: the Turing machine and the broken balance. *Journal of Logic and Computation*, 23(6):1155–1181, 2013. Special issue on The Incomputable, Editors S. Barry Cooper and Mariya I. Soskova.

[11] Edwin Beggs, José Félix Costa, and John V. Tucker. Computational models of measurement and hempel's axiomatization. In A. Carsetti, editor, *Causality, Meaningful Complexity and Embodied Cognition*, volume 46 of *Theory and Decision Library A*, pages 155–183. Springer, 2010.

[12] Edwin Beggs, José Félix Costa, and John V. Tucker. Limits to measurement in experiments governed by algorithms. *Mathematical Structures in Computer Science*, 20(06):1019–1050, 2010. Special issue on Quantum Algorithms, Editor Salvador Elías Venegas-Andraca.

[13] Edwin Beggs, José Félix Costa, and John V. Tucker. Physical oracles: The Turing machine and the Wheatstone bridge. *Studia Logica*, 95(1-2):279–300, 2010.

[14] Edwin Beggs, José Félix Costa, and John V. Tucker. The impact of models of a physical oracle on computational power. *Mathematical Structures in Computer Science*, 22(5):853–879, 2012. Special issue on Computability of the Physical, Editors Cristian S. Calude and S. Barry Cooper.

[15] Edwin J. Beggs, José Félix Costa, Diogo Poças, and John V. Tucker. An analogue-digital church-turing thesis. *Int. J. Found. Comput. Sci.*, 25(4):373–390, 2014.

[16] Edwin J. Beggs, José Félix Costa, and John V. Tucker. Axiomatizing physical experiments as oracles to algorithms. *Philosophical Transactions of the Royal Society, Series A (Mathematical, Physical and Engineering Sciences)*, 370(12):3359–3384, June 2012.

[17] Edwin J. Beggs, José Félix Costa, and John V. Tucker. A natural computation model of positive relativisation. *IJUC*, 10(1-2):111–141, 2014.

[18] Edwin J. Beggs, José Félix Costa, and John V. Tucker. Three forms of physical measurement and their computability. *The Review of Symbolic Logic*, 7:618–646, 12 2014.

[19] Andreas Blass and Gábor Braun. Random orders and gambler's ruin. *Electr. J. Comb.*, 12:R23, 2005.

[20] Olivier Bournez and Michel Cosnard. On the computational power of dynamical systems and hybrid systems. *Theoretical Computer Science*, 168(2):417–459, 1996.

[21] Cristian S. Calude and Michael J. Dinneen. Exact approximations of omega numbers. *I. J. Bifurcation and Chaos*, 17(6):1937–1954, 2007.

[22] Cristian S. Calude and Michael A. Stay. Natural halting probabilities, partial randomness, and zeta functions. *Information and Computation*, 204(11):1718 – 1739, 2006.

[23] Gregory Chaitin. *How Much Information Can There Be in a Real Number?*, volume 7160 of *Lecture Notes in Computer Science*, chapter 19, pages 247–251. Springer Berlin Heidelberg, Berlin, Heidelberg, February 2012.

[24] Gregory J. Chaitin. A theory of program size formally identical to information theory. *J. ACM*, 22(3):329–340, July 1975.

[25] José Félix Costa. Incomputability at the foundations of physics (A study in the philosophy of science). *J. Log. Comput.*, 23(6):1225–1248, 2013.

[26] José Félix Costa. Uncertainty in time. *Parallel Processing Letters*, 25(1), 2015.

[27] José Félix Costa and Raimundo Leong. The ARNN model relativises P=NP and P≠NP. *Theor. Comput. Sci.*, 499:2–22, 2013.

[28] Nigel Cutland. *Computability*. Cambridge University Press, 1980.

[29] Martin Davis. The myth of hypercomputation. In Christof Teuscher, editor, *Alan Turing: the life and legacy of a great thinker*, pages 195–212. Springer, 2006.

[30] Martin Davis. Why there is no such discipline as hypercomputation. *Applied Mathematics and Computation*, 178(1):4–7, 2006.

[31] Kevin T. Kelly. *The Logic of Reliable Inquiry (Logic and Computation in Philosophy)*. Oxford University Press, USA, January 1996.

[32] Pascal Koiran, Michel Cosnard, and Max Garzon. Computability with low-dimensional dynamical systems. *Theoretical Computer Science*, 132:113–128, 1994.

[33] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

[34] Frederick Mosteller. *Fifty Challenging Problems in Probability with Solutions*. Dover Publications, 1987.

[35] Piergiorgio Odifreddi. *Classical Recursion Theory II*. Studies in Logic and the Foundations of Mathematics. North Holland, 1999.

[36] Dinis D. Pestana and Sílvio F. Velosa. *Introdução à Probabilidade e à Estatística*, volume 1. Fundação Calouste Gulbenkian, third edition, 2008.

[37] Diogo Miguel Ferreira Poças. Complexity with costing and stochastic oracles. Master's thesis, Instituto Superior Técnico, July 2013.

[38] Uwe Schöning. *Complexity and Structure*, volume 211 of *Lecture Notes in Computer Science*. Springer, 1986.

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

[40] Salvador Elias Venegas-Andraca. *Quantum Walks for Computer Scientists*. Morgan and Claypool Publishers, 2008.

[41] John von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In *Automata Studies*, pages 43–98. Princeton University Press, 1956.

[42] Arthur Steven Younger, Emmett Redd, and Hava T. Siegelmann. Development of physical super-turing analog hardware. In *Unconventional Computation and Natural Computation - 13th International Conference, UCNC 2014, London, ON, Canada, July 14-18, 2014, Proceedings*, pages 379–391, 2014.

Appendix A

Non-uniform complexity

Non-uniform complexity refers to Turing machines that have different behaviours for inputs of different sizes. A way to connect non-uniform complexity with uniform complexity (the usual, where the algorithm is the same for inputs of every size) is to assume that exists a unique algorithm for inputs of any size, that is aided by a certain information, the advice, which may vary for inputs of different sizes.

A.1 Turing machines with advice

A pairing function is a homomorphism $\langle .,. \rangle : \Sigma^* \times \Sigma^* \to \Sigma^*$ that allows to encode two words over the same alphabet in a single word in the same alphabet. A well known pairing function, computable in linear time, consists in duplicating bits and inserting a separation symbol "01". This is the pairing function used, for example, in [39] and [6]. Its inverses, $\langle . \rangle_1 : \Sigma^* \to \Sigma^*$ and $\langle . \rangle_2 : \Sigma^* \to \Sigma^*$, are also computable in linear time.

An advice function is any total function $f : \mathbb{N} \to \Sigma^*$. As noted in [3], the main property of advice functions is that the value of the advice does not fully depend on the particular instance we want to decide, but just on its size. Hence, advice functions provide external information to the algorithms, as oracles do, but the information provided by an oracle may depend on the actual input, whereas the information provided by an advice function does not: just the size of the input matters.

An advice Turing machine receives inputs of the form $\langle w, f(|w|) \rangle$, for some $w \in \Sigma^*$. Therefore, it can read both the "input" w and the advice $f(|w|)$, and its time resource is a function of $|\langle w, f(|w|) \rangle|$.

Now, we define non-uniform classes.

Definition A.1. Let $\mathcal{C}$ be a class of sets and $\mathcal{F}$ a class of advice functions. The class $\mathcal{C}/\mathcal{F}$ is the class of sets A such that there exists a set $C \in \mathcal{C}$ and an advice $f \in \mathcal{F}$ such that, for every word $w \in \Sigma^*$, $w \in A$ if and only if $\langle w, f(|w|) \rangle \in C$.

If $\mathcal{C}$ is a complexity class defined by a bound on some computational resource, $\mathcal{C}/\mathcal{F}$ is the class of all sets A such that some function of $\mathcal{F}$ provides enough additional information to decide A within the bounds specified by $\mathcal{C}$.

Common classes of advice functions are polynomially and logarithmically long advices, i.e., $poly$ and $\log$, where

$$poly = \{f : \mathbb{N} \to \Sigma^* | \text{ for some polynomial } p, |f(n)| \in O(p(n))\}$$

and

$$\log = \{g : \mathbb{N} \to \Sigma^* | \ |g(n)| \in O(\log(n))\}.$$

When exponentially long advice is allowed, every set can be decided in polynomial time, taking as advice the concatenation of all words of length n that belong to the set (at most 2^n words), thus this case is not very interesting.

In this work, we focus on the special case of prefix non-uniform classes. A prefix function is a function f such that, for every n, $f(n)$ is a prefix of $f(n+1)$. Therefore, if the advice function is a prefix function, the advice is useful not only for inputs of size n, but for all inputs with size less or equal than n.

Definition A.2. Let $\mathcal{C}$ be a class of sets and $\mathcal{F}$ a class of advice functions. The prefix non-uniform class $\mathcal{C}/\mathcal{F}\star$ is the class of sets A such that there exists a set $C \in \mathcal{C}$ and a prefix function $f \in \mathcal{F}$ such that, for every length n and every word w with $|w| \leq n$, $w \in A$ if and only if $\langle w, f(n)\rangle \in C$.

There are cases in which $\mathcal{C}/\mathcal{F}\star$ coincides with $\mathcal{C}/\mathcal{F}$, like, for example, $P/poly\star = P/poly$. However, for logarithmic advices, the equality does not hold.

A.2 $BPP/\log\star$ versus $BPP//\log\star$

Applying Definition A.1 to the probabilistic complexity class of BPP, we get that $BPP/\mathcal{F}$ is the class of sets A such that there exists a set $C \in BPP$ and an advice $f \in \mathcal{F}$ such that, for every word $w \in \Sigma^*$, $w \in A$ if and only if $\langle w, f(|w|)\rangle \in C$. As $C \in BPP$, there is a probabilistic Turing machine that decides C with error probability bounded by γ, for any possible advice. However, instead of forcing the advice to be chosen after the Turing machine, it is more intuitive to first choose the correct advice function and, only then, establish the bound for the probability of error.

Definition A.3. Let $\mathcal{C}$ be a class of sets and $\mathcal{F}$ a class of advice functions. $\mathcal{C}//\mathcal{F}$ is the class of sets A for which, given an advice function $f \in \mathcal{F}$, there exists a set $C \in \mathcal{C}$ such that, for every $w \in \Sigma^*$, $w \in A$ if and only if $\langle w, f(|w|)\rangle \in C$.

Definition A.4. Let $\mathcal{C}$ be a class of sets and $\mathcal{F}$ a class of advice functions. $\mathcal{C}//\mathcal{F}\star$ is the class of sets A for which, given a prefix advice function $f \in \mathcal{F}$, there exists a set $C \in \mathcal{C}$ such that, for every length n and every word w with $|w| \leq n$, $w \in A$ if and only if $\langle w, f(n)\rangle \in C$.

The main difference between these two definitions and the ones that we presented in Section A.1 is that, while in Definitions A.1 and A.2 the advice was chosen after fixing the Turing machine, here we first choose the advice function f and, given that function, we choose the Turing machine and the corresponding error probability. To make it clearer, we study the case of $BPP/\log\star$ and $BPP//\log\star$.

Definition A.5. $BPP/\log\star$ is the class of sets A for which there exist a probabilistic Turing machine $\mathcal{M}$, a prefix advice function $f \in \log$ and a constant $\gamma < 1/2$ such that, for every length n and every

word w with $|w| \leq n$, $\mathcal{M}$ 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 $w \notin A$.

Definition A.6. $BPP//\log\star$ is the class of sets A for which, given a prefix advice function $f \in \log$, there exist a probabilistic Turing machine $\mathcal{M}$ and a constant $\gamma < 1/2$ such that, for every length n and every word w with $|w| \leq n$, $\mathcal{M}$ 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 $w \notin A$.

For polynomial advices, we know that $P/poly = BPP/poly = BPP//poly$. However, for the logarithmic prefix case, it is known only that $P/\log\star \subseteq BPP/\log\star \subseteq BPP//\log\star$.

A.3 Non-uniform complexity and the halting problem

In this work, we deal with many non-uniform complexity classes, in particular the ones of the form $BPP//\log^{(k)}\star$, for a natural number k. One may wonder if these classes have any interest. It is well known that the halting problem is undecidable. Here we show that all these non-uniform complexity classes contain some halting set.

The halting set $\{0^n : n \text{ is the encoding of a Turing machine that halts on input } 0\}$ is in $P/linear$. Thus, $\{0^{2^n} : n \text{ is the encoding of a Turing machine that halts on input } 0\} \in P/\log$ and, more generally, using the functions defined in Section 3.2.2, $\{0^{\exp^{(k)}(n)} : n \text{ is the encoding of a Turing machine that halts on input } 0\} \in P/\log^{(k)}$. In fact, we know that we can use a prefix function as advice,[1] therefore, $\{0^{\exp^{(k)}(n)} : n$ is the encoding of a Turing machine that halts on input $0\} \in P/\log^{(k)}\star \subseteq BPP//\log^{(k)}\star$, for every natural number k.

[1] Take as advice the first n bits of Chaitin Omega number Ω (see Appendix C).

Appendix B

Bounded error probability

Inspired in [4] and [38], we state the following auxiliary lemma and prove the Proposition B.2, that says that, for a set in BPP, we can get the probability error to be as small as we want.

Lemma B.1. *Let E be an event that occurs with probability greater than $1/2 + \varepsilon$, for $0 < \varepsilon < 1/2$. Then, for any odd t, the probability that E occurs $t/2$ times, within t independent trials is greater than*

$$1 - \frac{1}{2}(1 - 4\varepsilon^2)^{t/2}.$$

Proposition B.2. *If a set A is in BPP, then, for every polynomial q, there is a probabilistic Turing machine $\mathcal{M}$, running in polynomial time, such that, for every word w:*

$$P(\mathcal{M}(w) = \chi_A(w)) > 1 - 2^{-q(|w|)}.$$

In this proposition, we use $\mathcal{M}(w) = \chi_A(w)$ to say that $\mathcal{M}$ accepts w if $w \in A$ and $\mathcal{M}$ rejects w if $w \notin A$. Therefore, by saying that for every w, $P(\mathcal{M}(w) = \chi_A(w)) > 1 - 2^{-q(|w|)}$, we mean that $\mathcal{M}$ decides A with error probability bounded by $2^{-q(|w|)}$.

Proof. Let $A \in BPP$. Thus, there is a Turing machine $\mathcal{M}$ and a positive constant $\varepsilon < 1/2$ such that, for every word w,

$$P(\mathcal{M}(w) = \chi_A(w)) > \frac{1}{2} + \varepsilon.$$

For each odd t, consider the machine $\mathcal{M}_t$ that iterates $\mathcal{M}$ t times and, for every input word w, $\mathcal{M}_t$ accepts if and only if $\mathcal{M}$ accepts w more than $t/2$ times. By Lemma B.1,

$$P(\mathcal{M}_t(w) = \chi_A(w)) > 1 - \frac{1}{2}(1 - 4\varepsilon^2)^{t/2}.$$

Therefore, we only need to choose t such that

$$1 - \frac{1}{2}(1 - 4\varepsilon^2)^{t/2} \geq 1 - 2^{-q(|w|)}.$$

Changing signs and inverting the terms, we get

$$\frac{2}{(1-4\varepsilon^2)^{t/2})} \geq 2^{q(|w|)},$$

which is equivalent to

$$t \geq \frac{2}{\log(\frac{1}{1-4\varepsilon^2})}(q(|w|)-1).$$

Then, it suffices to take $t = t(|w|) = cq(|w|)$, where

$$c = \frac{2}{\log(\frac{1}{1-4\varepsilon^2})}.$$

The machine $\mathcal{M}_t$ runs in time $O(tp(|w|)) = O(q(|w|)p(|w|))$, where p is the time resource of $\mathcal{M}$. □

Note that although this proposition is stated for polynomials, it applies to every function asymptotically bounded by a polynomial.

Appendix C

Chaitin Omega number

Chaitin Omega number, introduced by Chaitin in [24], is the halting probability, i.e., the probability that a universal prefix-free Turing machine halts for a program chosen at random.

A set $A \in \Sigma^*$ is prefix-free if, for every $x, y \in A$, if x is a prefix of y, then $x = y$. A prefix-free Turing machine is a Turing machine C such that its domain, $dom(C) = \{x \in \Sigma^* | C(x) \text{ halts}\}$, is prefix-free.

Definition C.1. The halting probability of a prefix-free Turing machine C is

$$\Omega_C = \sum_{p \in dom(C)} 2^{-|p|}.$$

This is the probability that the machine C halts for a program whose bits are generated one by one by independent tosses of a fair coin. Each k-bit program that halts contributes exactly $1/2^k$ to the halting probability Ω_C.

A universal Turing machine is a Turing machine that can simulate any other Turing machine. More formally, we have:

Definition C.2 (Calude and Stay [22])**.** The Turing machine U is universal for a class $\mathcal{R}$ of Turing machines if for every Turing machine $C \in \mathcal{R}$ there exists a fixed constant $c \geq 0$ (depending upon U and C) such that for every $x \in dom(C)$ there is a string $p_x \in dom(U)$ with $|p_x| \leq |x| + c$ and $U(p_x) = C(x)$. In case $U \in \mathcal{R}$, we simply say that the machine $U \in R$ is universal.

We also know the following result:

Theorem C.3 (Chaitin [24])**.** *We can effectively construct a universal prefix-free Turing machine.*

When $C = U$ is a universal prefix-free machine, its halting probability $\Omega = \Omega_U$ is called a Chaitin Omega number.

This probability obviously depends on the machine, thus, we have several different Omega numbers, but all with the same properties, so it is often considered that there is only one Omega number, Ω.

Knowing the number Ω allows us to solve the halting problem. In fact, according to [23], knowing the first n bits of Ω would enable us to solve the halting problem for all programs up to n bits in size. Therefore, Ω is uncomputable. Actually, it is more than uncomputable.

Ω is incompressible, i.e., we cannot compute n bits of Ω with a program that has less than $n - O(1)$ bits. In terms of ZFC theory (Zermelo-Fraenkel set theory with the axiom of choice), we have the following result:

Theorem C.4 (Calude and Dinneen [21])**.** *Assume that ZFC is arithmetically sound, that is, any theorem of arithmetics proved by ZFC is true. Then, for every universal machine U, ZFC can determine the value of only finitely many bits of Ω_U .*

Thus, for η sufficiently small, we cannot predict n bits of Ω from $\eta.n$ bits. Therefore, the set of Ω prefixes is in $BPP//linear$, but not in $BPP//\eta.linear$.

Index