INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Kolmogorov Generic Bases

Ana Luísa Mendes Lima Pereira

Dissertação para obtenção do Grau De Mestre em
Matemática e Aplicações

Júri

Presidente:	Professora Doutora Cristina Sales Viana Serôdio Sernadas
Orientador:	Professor Doutor João Filipe Quintas dos Santos Rasga
Vogais:	Professor Doutor Paulo Alexandre Carreira Mateus

Outubro 2011

Acknowledgments

First of all, I would like to thank my thesis adviser, Professor João Rasga, for all his time and care for my work. His guidance was crucial to the development of this thesis, which would not had been the same without our many discussions.

I would also like to thank my family, particularly my mother and father, for all their support and patience. I realize I am not always the easiest person but they never give up.

Finally, a very big thank you to all my friends, for keeping me happy and inspired and being there for me through anything. In the words of James Murphy, "We set controls for the heart of the sun, one of the ways that we show our age.".

Resumo

Começamos por estudar a complexidade algorítmica de números e de strings (o tamanho da menor descrição do objecto em questão). Provamos que existe uma correspondência entre estas duas complexidades e seguimos para estudar complexidade de prefixo (uma restrição da complexidade algorítmica simples). Também definimos e estudamos uma forma de complexidade de demonstrações lógicas baseada no número mínimo de linhas necessário para provar uma fórmula. Aqui, adaptamos resultados conhecidos e apresentamos alguns novos. Seguidamente, usamos as semelhanças entre estes quatro tipos de complexidade para definir uma nova estrutura: a base genérica de Kolmogorov. Verificamos que instâncias desta estrutura coincidem com as complexidades previamente mencionadas. Por fim, apresentamos alguns resultados novos, nomeadamente que existem classes de bases genéricas de Kolmogorov que induzem funções não computáveis (algumas das quais podem ser aproximadas por uma função computável), funções computáveis e funções ilimitadas.

Palavras-Chave: complexidade de Kolmogorov, complexidade de Kolmogorov de prefixo, complexidade de demonstrações lógicas, base genérica de Kolmogorov.

Abstract

We start by studying the algorithmic complexity of numbers and of strings (the size of the smallest description of the object in question). We prove that there is a correspondence between these two complexities and go on to study algorithmic prefix complexity (a restriction of plain algorithmic complexity). We also define and study a form of logical proof complexity based on the minimum number of lines necessary to demonstrate a formula. Here, we adapt some known results and present some new ones. We then use the similarities between these four types of complexity to define a new structure: the Kolmogorov generic basis. We verify that instances of it coincide with the complexities previously mentioned. Finally, we present some new results, namely that there are classes of Kolmogorov generic bases that induce: incomputable functions (some of which can be approximated by a computable function), computable functions and unbounded functions.

Keywords: Kolmogorov complexity, prefix Kolmogorov complexity, logical proof complexity, Kolmogorov generic basis.

Contents

Acknowledgments **iii**

Resumo **v**

Abstract **vii**

Contents **ix**

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**

1.1 Context 1

1.2 Computability and Complexity Theory 2

1.3 Coding Theory 5

1.4 Classical Propositional Logic 6

2 Kolmogorov Based Complexities **11**

2.1 Kolmogorov Complexity 11

2.1.1 Kolmogorov Complexity Based on Partial Recursive Functions . . . 12

2.1.2 Kolmogorov Complexity Based on Turing Machines 20

2.2 Prefix Kolmogorov Complexity 24

2.3 Kolmogorov Proof Complexity 29

2.3.1 Frege Systems 31

3 Kolmogorov Generic Bases **41**

3.1 Definition and Examples 41

3.2 Results 48

4 Conclusion **55**

Bibliography **57**

List of Figures

1.1 A Turing machine. 2

3.1 Some Kolmogorov generic bases notable classes: $\mathcal{C}_1$ has non computable functions, $\mathcal{C}_2$ has computable functions, $\mathcal{C}_3$ has unbounded functions and $\mathcal{C}_4$ has functions that can be approximated by a computable function. 54

List of Tables

3.1 Examples of Kolmogorov generic bases. 48

Chapter 1

Introduction

1.1 Context

One of the main issues in information theory, a branch of applied mathematics and electrical engineering involving the quantification of information, is how to describe an object using the minimal amount of information possible. But what properties should that description have? And minimal by what standard? These are some of the questions Kolmogorov complexity sets out to investigate and the reason why we will look at it and some of its various versions in Sections 2.1 and 2.2. Here, we will define the complexity of an object as the length of its smallest possible description.

While studying this subject, we came across the idea of somehow bringing these concepts into the realm of logic. We did this by choosing formulas as our objects and derivations as our descriptions, therefore defining the complexity of a formula as the length of its shortest proof. In Section 2.3, we investigate the proper way to define this concept, as, for instance, we try to find the best deductive system to work with. Also, since proof complexity is not a new subject, some known results are adapted and some new results are proven.

Looking at these two settings together, we started to realize how similar they were and considered how they could be described by the same structure. For instance, they are both minimization functions of a measure of size of a method for arriving at a given object. With this and more in mind, we were able to develop in Chapter 3 the wider concept of Kolmogorov generic basis, instances of which can in fact generate the complexities we studied in Chapter 2. We were then able to arrive at a concept for which we proved some results, even if not applicable to all Kolmogorov generic bases as we had first hoped for.

Finally, in Chapter 4, we present some ideas on how to improve the work done for this thesis and possibly come up with some more general results. The background we will

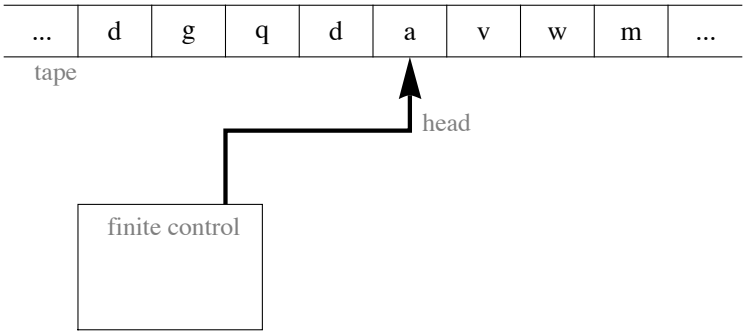

Figure 1.1: A Turing machine.

need on computability, complexity and coding theory, as well as propositional logic, can be found in this chapter.

1.2 Computability and Complexity Theory

In this section, we will present several well known notions and results related to computability theory. The proofs of these results can be found in [13, Section 1.7]. However, as we will not focus our attention too much on these matters. For a more comprehensive study of them, the reader is referred to any textbook on computability theory, for instance [16].

First, let us consider the Turing machine, the theoretical device presented by Alan Turing in 1936. A *Turing machine* consists of a finite program, called the *finite control*, capable of manipulating a linear list of cells, called the *tape*, using one access pointer, called the *head*. An illustration of this can be seen in Figure 1.1.

It uses an infinite tape as its memory, therefore unlimited. The head can read and write symbols as well as move around on the tape. Initially, the tape contains only the input string and is blank everywhere else. If the machine needs to store information, it may write it on the tape. To read the information it has written, the machine can move the head back over it. The outputs *accept* and *reject* are obtained by entering designated accepting or rejecting states. If it doesn't enter an accepting or rejecting state, it will go on forever, never halting.

Furthermore, Turing machines can be divided into two categories, according to the number of possibilities for the result of each step:

- *deterministic Turing machines* (DTM)- the set of rules prescribes at most one action to be performed for any given situation;

- *non-deterministic Turing machines* (*NDTM*)- the set of rules may prescribe more than one action for a given situation.

A Turing machine may also have more than one tape. In this case, it usually has an input tape (where the input is written, usually a read-only tape), an output tape (where the output is written, usually a write-only tape) and a finite number of working tapes (where it does any calculations needed).

We will be focused on deterministic Turing machines so when we say "Turing machine" we will mean a deterministic one.

Definition 1.2.1. A *partial recursive function* is a function computed by a Turing machine, from n-tuples of integers onto the integers, $n \geq 1$. If the function computed is defined for all arguments, i.e., the Turing machine halts for all inputs, we call it a *total recursive function*, or just *recursive function*.

A *predicate* is a function with range $\{0, 1\}$. We can interpret this as: a predicate of an n-tuple of values is 'true' if the function assumes value 1 when it has that n-tuple of values as its arguments; otherwise, it is 'false' or 'undefined'. Hence, we can talk about *partial* and *total recursive predicates*. ∇

The **Church-Turing thesis**, name after mathematicians Alonzo Church and Alan Turing, states that *"every effectively calculable function is a computable function"*. As such, an "effective procedure" is simply a computable one.

Another important result is that it's possible to give an *effective enumeration* of Turing machines. By this we mean that it's possible to construct a computable one-to-one pairing linking each natural number to a specific Turing machine and vice-versa. One simple way to achieve this enumeration is to encode, in a canonical way, the table of rules of each Turing machine in binary. This effective enumeration implies that there is also one for partial recursive functions, since each partial recursive function is computed by a unique Turing machine. Hence, it's also linked to a unique natural number.

Definition 1.2.2. A *universal* Turing machine U is a Turing machine that can simulate the behavior of an arbitrary Turing machine for any input. ∇

All that is needed to make U simulate machine T is an adequate description of T's finite program such that, when the input is entered on U's tape, it uses the description of T to imitate its actions on a representation of its tape contents. There are infinitely many universal Turing machines.

The partial recursive function ϕ computed by a universal Turing function U is called a *universal partial recursive function*.

Partial and total recursive functions are also useful in characterizing sets.

Definition 1.2.3. A set A is *recursively enumerable (or computably enumerable, semi-decidable, provable or Turing-recognizable)* if there exists a partial recursive function ϕ such that $\phi(x) = \begin{cases} 1, & x \in A \\ \text{undefined}, & x \notin A \end{cases}$.

A set A is *recursive* if there exists a recursive characteristic function, i.e., a recursive function ϕ such that $\phi(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases}$. ∇

An alternative definition of recursive set and a useful property of recursively enumerable sets can be seen next.

Lemma 1.2.4. *A set A is recursive if and only if A and its complement $\overline{A}$ are recursively enumerable.*

Every infinite recursively enumerable set contains an infinite recursive subset. $\diamondsuit$

Finally, we define the notion of computability.

Definition 1.2.5. A partial function $f : \mathbb{Q} \to \mathbb{R}$ is *upper semi-computable* if it can be computably approximated from above, i.e., if it is defined by a rational-valued partial recursive function $\phi(x, k)$ with x a rational number and k a non-negative integer such that $\phi(x, k + 1) < \phi(x, k)$ for every k and $\lim_{x\to\infty}\phi(x, k) = f(x)$. Moreover, a function f is called:

- *lower semi-computable* if $-f$ is upper semi-computable;
- *semi-computable* if it is lower or upper semi-computable or both;
- *computable* (or *recursive* if the domain is integer or rational) if it's both lower and upper semi-computable in its domain. ∇

For a total function, similar definitions are used.

We now turn our attention to complexity theory. A *language* over an *alphabet* Σ (a set of symbols, usually finite) is simply a subset of Σ^* (set of all finite strings over the alphabet Σ). A Turing machine *accepts a language* L if the machine computes a total recursive predicate over the language, that is, if it returns 1 when the input is a member of L and 0 otherwise.

Definition 1.2.6. (*Time and space computational complexity*) Let T be a Turing machine and t and s two functions over the natural numbers. For each input of length n, if T makes at most $t(n)$ moves before it stops, then we say that T runs in time t (has *time complexity* t). If T uses at most $s(n)$ space tape cells in the aforementioned computation, we say that T uses s space (has *space complexity* s). ∇

Given a function $f : \mathbb{N} \to \mathbb{N}$, we will use the notation $O(f)$ to denote the class of functions that from a certain point on do not exceed f by more than a fixed multiplicative factor.

Definition 1.2.7. There are several well known examples of complexity classes:

- DTIME$[t(n)]$ is the set of languages accepted by multi-tape deterministic Turing machines in time $O(t(n))$;
- NTIME$[t(n)]$ is the set of languages accepted by multi-tape non-deterministic Turing machines in time $O(t(n))$;
- P is the complexity class $\cup_{c\in\mathbb{N}}$ DTIME$[n^c]$;
- NP is the complexity class $\cup_{c\in\mathbb{N}}$ NTIME$[n^c]$. ∇

Languages accepted in *polynomial time*, that is, the languages in P, are considered to be *feasibly* computable. The extensively studied "P versus NP" question can be understood as whether problems for which it is easy to certify the answer are the same problems for which it is easy to find the answer. This is a matter that has sparked much discussion and research and still has no definite answer.

1.3 Coding Theory

The field of information theory was first developed by Claude E. Shannon under the name "mathematical theory of communication". It consists of a theory where the meaning of a message is ignored and interest lies only in the problem of communicating a message between a sender and receiver, under the assumption that the universe of possible messages is known to both parties. The set of possible messages from which the selection takes place is an *essemble*. We will only consider countable ensembles.

We will focus our attention on the subfield of coding theory, more specifically prefix-codes and their advantages. The proofs of the results in this section will be omitted but can be found in [13, Section 1.11].

Let Σ be a finite alphabet (commonly $\{0, 1\}$) and $D : \Sigma^* \to \mathbb{N}$ our *decoding function* (its domain is the set of *code-words* and its range is the set of *source-words*). If $D(y) = x$, y is a code-word for the source-word x.

Define the *encoding relation* E for a source-word $x \in \mathbb{N}$ *as* $E(x) = S \subseteq \Sigma^*$, where S is the set of all code-words y such that $D(y) = x$. If D is injective, then E is a function and it is called the *encoding function*.

Consider the natural extension E' of E defined as follows:

- $E'(\varepsilon) = \varepsilon$ (where ε is the empty string);
- $(x, y) \in \mathbb{N}^2 \Rightarrow E'(xy) = \{ab : a \in E(x), b \in E(y)\}$.

From this point on, the difference between E and E' will be ignored and both will be addressed as E.

A *source-sequence* is a sequence of strings $x_1 ... x_n$, with $x_i \in \mathbb{N}$ for $i = 1, ..., n$ that we want to encode and a *code-sequence* is a sequence of strings $y_1 ... y_n$, with $y_i = E(x_i)$ for $i = 1, ..., n$, the result of an encoding. We say a code is *uniquely decodable* if E is injective, i.e., if for any source-sequence of finite length, there is no other sequence that has the same corresponding code-sequence.

Let $l : \Sigma^* \to \mathbb{N}$ be a function that receives a code-word and returns its length, i.e., the number of symbols in the string. If the source-word alphabet is finite and there is a constant c such that $l(y) = c$ for all code-words y, then we say that D is a *fixed-length* code. On the other hand, when the set of source-words is infinite, we resort to *variable-length* codes.

Code-word $x_1 ... x_n$ is a *prefix* of $y_1 ... y_m$, with $n \leq m$, if $x_i = y_i$, for $i = 1, ..., n$. If no code-word is the prefix of another code-word, then each code-word is uniquely decodable, which is why we are interested in prefix-codes, defined below.

Definition 1.3.1. A code is a *prefix-code* (or *instantaneous code*) if the set of code-words is prefix-free, that is, no code-word is a prefix of another code-word. ∇

The main advantage is that to decode a sequence from a prefix-code, we just have to start at the beginning and decode one code-word at a time, making it simpler than having to take the whole sequence into account.

A uniquely decodable code is *complete* if adding a new code-word to the code-word alphabet makes it no longer uniquely decodable.

Definition 1.3.2. A prefix-code is *self-delimiting* if there is a Turing machine that decides whether a given word is a code-word, receiving only the word as input, and moreover, computes the decoding function. (With respect to Turing machines, each code-word has an implicit end marker.) ∇

This type of code has the advantage that we can recognize each code-word in a sequence, a very useful property.

1.4 Classical Propositional Logic

In this section, we introduce some notation and the concepts in classical propositional logic we will need further on, based on [1, 6, 15].

Assume a fixed countably infinite set of propositional variables Ξ, whose elements will be denoted by $\xi_1, \xi_2, ...$ The set of *propositional formulas over* Ξ, denoted by L, contains the formulas of the form

$$\varphi ::= \xi | (\neg\varphi) | (\varphi \to \varphi) | (\varphi \wedge \varphi),$$

where $\xi \in \Xi$.

The intuitive meaning of the symbol $\neg$ is *negation*, that is, $\neg\varphi$ is true when φ is not true. The intuitive meaning of the symbol $\to$ is *implication*, that is, $\varphi_1 \to \varphi_2$ is true when φ_1 being true implies that φ_2 is also true. Lastly, the intuitive meaning of the $\wedge$ is *conjunction*, i.e., $\varphi_1 \wedge \varphi_2$ is true when φ_1 and φ_2 are both true.

Other boolean connectives can be defined as abbreviations using the ones above. For instance, $\varphi \vee \psi$ is defined as $(\neg\varphi) \to \psi$.

From now on, when we say φ is a formula, we will mean that it is a propositional formula over Ξ and when no confusion arises, we will omit some parentheses.

A *valuation* v is a map that assigns a truth value (0 or 1) to each propositional variable, i.e., it is a total function $v : \Xi \longrightarrow \{0, 1\}$. As usual, 0 corresponds to false and 1 corresponds to true.

Definition 1.4.1. A formula φ is said to be *satisfied by valuation* v if $v \Vdash \varphi$. We inductively define the relation $\Vdash$, with regards to a valuation v, as:

- $v \Vdash \xi$ if $v(\xi) = 1$;
- $v \Vdash (\neg\varphi)$ if $v \nVdash \varphi$;
- $v \Vdash (\varphi_1 \to \varphi_2)$ if $v \Vdash \varphi_2$ or $v \nVdash \varphi_1$;
- $v \Vdash (\varphi_1 \wedge \varphi_2)$ if $v \Vdash \varphi_1$ and $v \Vdash \varphi_2$. ∇

Definition 1.4.2. Given a set of formulas Γ and a formula ψ, we write $\Gamma \models \psi$ if ψ is a semantic consequence of Γ, i.e., if for all γ in Γ, $v \Vdash \gamma$, then $v \Vdash \varphi$.

A formula φ is called a *tautology* if $\models \varphi$. ∇

Definition 1.4.3. A *substitution* σ is a mapping from the propositional variables in Ξ to the formulas in L. If φ is a formula, $\sigma\,\varphi$ denotes the result of applying the substitution to φ, i.e., simultaneously replacing each propositional variable in φ by its image under σ. We call $\sigma\,\varphi$ a substitution instance of φ. ∇

Definition 1.4.4. A *Frege rule* is a pair $\langle\{\psi_1, ..., \psi_k\}, \varphi\rangle$, represented by $\frac{\psi_1, ..., \psi_k}{\varphi}$, where $\psi_1, ..., \psi_k$ and φ are formulas and $\psi_1, ..., \psi_k \models \varphi$. The formulas $\psi_1, ..., \psi_k$ are called

premises and the formula φ is called the *conclusion*. If $k = 0$, then the rule is called an *axiom schem*a. ∇

We call a substitution instance of an axiom schema an *axiom* and we say that formula ϕ_{k+1} is *inferred by Frege rule* $\langle\{\psi_1, ..., \psi_k\}, \psi_{k+1}\rangle$ *from formulas* $\phi_1, ..., \phi_k$ if there exists a substitution σ such that $\sigma\,\psi_i = \phi_i$, for $i = 1, ..., k+1$.

An example of a Frege rule is Modus Ponens, where given the premises $\xi_1 \to \xi_2$ and ξ_1, we conclude ξ_2. This rule can be represented by the pair $\langle\{\xi_1 \to \xi_2, \xi_1\}, \xi_2\rangle$.

Definition 1.4.5. A *deductive system* consists of a set of Frege rules. ∇

An example of a deductive system is given next.

Example 1.4.6. Let $\Xi = \{\xi_1, ..., \xi_n\}$ be a set of propositional variables. For each variable $\xi_i \in \Xi$, let $\overline{\xi_i}$ denote the negation of ξ_i. A *literal* l_i can either be a variable ξ_i or its negation $\overline{\xi_i}$, with the convention that if $l_i = \overline{\xi_i}$, then $\overline{l_i} = \overline{\overline{\xi_i}} = \xi_i$. A *clause* is a disjunction of literals $\{l_{i_1}, ..., l_{i_k}\}$, possibly empty. A *conjunctive normal form* (CNF) formula is a conjunction of clauses.

Resolution is a deductive system for formulas in CNF form based on the resolution algorithm of [14]. It uses the following *resolution rule:*

$$\frac{\xi_1 \vee \xi_2 \quad \overline{\xi_2} \vee \xi_3}{\xi_1 \vee \xi_3}.$$

□

Definition 1.4.7. Consider a deductive system D and a set $\Gamma \cup \{\varphi\}$ of formulas. A *derivation* for φ in D with hypothesis in Γ is simply a finite sequence of formulas $(\phi_1, ..., \phi_n)$ where each ϕ_i, $i = 1, ..., n$, is either one of the hypothesis, a substitution instance of an axiom schema in D or is inferred by a Frege rule in D from previous formulas in the sequence.

If a derivation has no hypothesis, it is called a *proof*. ∇

Denote by L^+ the set of all finite sequences over L, except for the empty sequence and L^* the set of all finite sequences over L including the empty sequence, denoted by ε.

Consider two distinct propositional variables, $\xi_1, \xi_2 \in \Xi$. We shall encode an element of L^* and one of L^+ into one of L^+ by the function $\langle\cdot,\cdot\rangle : L^* \times L^+ \to L^+$ such that $\langle\psi_1...\psi_m, \phi_1...\phi_n\rangle = \xi_1^m\,\xi_2\,\psi_1...\psi_m\,\phi_1...\phi_n$. Observe that the result of this pairing is in fact an element of L^+ since it is simply a sequence with the formula ξ_1 written m times, followed by the formula ξ_2 and then the formulas $\psi_1, ..., \psi_m$ and $\phi_1, ..., \phi_n$.

Definition 1.4.8. Consider a deductive system D. The *propositional proof system for L induced by D* is a function $\Delta_D : L^+ \to L$ that expects an input of the form $\langle \psi_1 ... \psi_m, \phi_1 ... \phi_n \rangle$, where $\psi_1 ... \psi_m \in L^*$and $\phi_1 ... \phi_n \in L^+$, and returns ϕ_n if $(\phi_1, ..., \phi_n)$ is a derivation sequence for ϕ_n in D with hypothesis in $\{\psi_1, ..., \psi_m\}$. Otherwise, Δ_D is undefined. ∇

When considering a derivation sequence $d = (\phi_1, ..., \phi_n)$ and a finite set of formulas $\Gamma = \{\psi_1, ..., \psi_m\}$, we will write $\langle \Gamma, d \rangle$ when we actally mean $\langle \psi_1 ... \psi_m, \phi_1 ... \phi_n \rangle$. Observe that we must consider the possibility of the empty sequence in the first part of $\langle \cdot, \cdot \rangle$ since we may be trying to prove a formula without the use of hypothesis. In this case ($S = \{\}$), we set the convention that $\langle S, d \rangle = \langle \varepsilon, d \rangle = \xi_2 \, \phi_1 ... \phi_n$, where ε is the empty sequence.

Furthermore, observe that each deductive system D induces only one Δ_D.

Notation 1.4.9. Let φ be a formula, Γ a (possibly infinite) set of formulas, d a derivation sequence and Δ_D the propositional proof system induced by a deductive system D. We use the notation $\Gamma \vdash^d_{\Delta_D} \varphi$ when there exists an $\Gamma' \subseteq \Gamma$ finite such that $\Delta_D(\langle \Gamma', d \rangle) = \varphi$.

We write $\Gamma \vdash_{\Delta_D} \varphi$ if there is a $d \in L^+$ such that $\Gamma \vdash^d_{\Delta_D} \varphi$.

We should point out that even if Γ is an infinite set of formulas, when creating a derivation sequence for φ in D with hypothesis in Γ, only a finite set of hypotheses is actually used.

Chapter 2

Kolmogorov Based Complexities

2.1 Kolmogorov Complexity

The concept of Kolmogorov complexity has its origins in probability theory, information theory and philosophical notions of randomness but belongs to the field of algorithmic information theory, a discipline of computer science. It is closely linked to an attempt to mend problems in both probability theory and information theory, namely that:

- The notion of randomness of an individual sequence cannot be expressed in classic probability theory;
- Even though the description of some strings can be compressed considerably if some regularity is exhibited, without such a pattern, it becomes more complex to express large numbers in a compressed manner.

As such, the notion of *Kolmogorov complexity* (or *descriptive complexity*, *stochastic complexity*, *algorithmic entropy*, *program-size complexity* or *algorithmic information content*) of an object came to be seen as the measure of the computational resources needed to specify the object. A more accurate definition follows.

Definition 2.1.1. The Kolmogorov complexity of an object is the length of the shortest program from which the object can be effectively reconstructed. ∇

Instead of defining the quantity of information of an object in relation to a set from which it may be picked, it is more natural to look only at the individual object. One way to measure the information it contains is to count the number of bits required to loosely describe it. For instance, the number 3^{101} can be very easily described in a few bits but there are numbers with the same amount of digits whose shortest description will just be the number itself, since there is not enough regularity to describe it with less bits.

However, a description is useful only if we can completely reconstruct the object from it. Furthermore, even though different methods of description may favor different objects, all should be described in a similar way and we should be able to describe any object from the universe we are considering.

As a matter of fact, the notion of information content of individual objects can only be practical if the quantity of information is an attribute of the object alone and is independent of the means of description. As such, we will require an agreed-upon universal description method and an agreed-upon mechanism to reproduce the object from its description.

For the rest of this section, we will look at two different ways of presenting the concept of Kolmogorov complexity: one, the standard definition and the other based on more intuitive notions connected to computer science. Along the way, we will prove they are equivalent and present some results regarding the two presentations of this complexity.

2.1.1 Kolmogorov Complexity Based on Partial Recursive Functions

This first presentation of Kolmogorov Complexity is based on Li-Vitányi's book [1], which has become an international standard on the matter.

Let X denote the set of object we will be considering, one that should be recursively enumerable. As such, let us define $i(x)$ as the natural number corresponding to the object $x \in X$.

To be able to compare different methods of specifying an object, we will assume each one is a partial function f over the naturals. At this point, we consider something as general as a partial function to try not to restrict our domain of specification methods. Further on, we will test if this is an appropriate class of functions or not.

Let $\Sigma = \{0, 1\}$ be the alphabet we will be using throughout the rest of this thesis. As such, when we refer to a string we will mean one over Σ, i.e., an element of Σ^*. We shall use the symbol ε to denote the empty string.

We will also need to consider the following one-to-one map between the natural numbers and strings: associate each string with its index in the length-increasing lexicographic ordering, as shown below.

$$(\varepsilon, 0), (0, 1), (1, 2), (00, 3), (01, 4), (10, 5), (11, 6), ...$$

Keeping this in mind, throughout this thesis, whenever we say "the natural number corresponding to string s", or the other way around, we will mean according to the map-

ping above.

We also define $l : \mathbb{N} \to \mathbb{N}$ as the function which returns the length (number of bits) of the string corresponding to the natural number in question, i.e., $l(p)$ is the length of the string in position $p+1$ of the ordering above. As such, for all p, $l(p) = \log(p) + O(1)$. Note that by log we have in mind the base 2 logarithm. This will be consistent throughout this thesis and we will say so if we mean a different base.

For each object x in X, the complexity C_f^F with respect to specifying method f is defined by

$$C_f^F(x) = \min\{l(p) : f(p) = i(x), p \in \mathbb{N}\}$$

and $C_f^F(x) = \infty$ if such a p does not exist.

The complexity $C_f^F(x)$ is, therefore, the length of the smallest string that corresponds to a natural number which outputs the rank of x when inputted into f. This may sound like a very unnatural definition (reason why we later look at a different presentation of this complexity) but in computer science terminology we would say that p is a program and f a computer, so that $C_f^F(x)$ is the minimal length of a program for f to compute x without additional input. The F in superscript stands for the fact that we are using functions as description methods.

If we consider distinct methods $f_1, f_2, ..., f_n$ of specifying the objects of X, it is easy to construct a new one, f, that assigns to each $x \in X$ a complexity $C_f^F(x)$ that exceeds only by a constant c the minimum of $C_{f_1}^F(x), C_{f_2}^F(x), ..., C_{f_n}^F(x)$. This could be done by reserving the first $\log(r)$ bits of p to identify the method f_i that should be followed, using as a program the remaining bits.

Definition 2.1.2. We say that a method f *minorizes* a method g (additively) if there is a positive constant c dependent only on f and g such that for all x in X,

$$C_f^F(x) \leq C_g^F(x) + c.$$

Also, we say that two methods f and g are *equivalent* if they minorize each other. ∇

Throughout this thesis, we will prove many results where we use additive or multiplicative constants. These will always be positive, even if we don't specify this.

We can now consider the hierarchy of equivalence classes of methods with regard to minorization. As Kolmogorov remarked, the idea of looking at the length of a description would have no use if the constructed hierarchy didn't possess certain appealing characteristics. In particular, we would like for there to be a unique minimal element in such a hierarchy: the equivalence class of description methods that minorizes all other description

methods. As it turns out, some sets of description methods do have a unique minimal element, while others don't.

Definition 2.1.3. Let $\mathcal{C}$ be a subclass of the partial functions over the naturals. A function f is *additively optimal* (a special type of universality) *for* $\mathcal{C}$ if it belongs to $\mathcal{C}$ and if for every function $g \in \mathcal{C}$, f minorizes g. ∇

Thus, if we chose an additively optimal method of specification, we can rest assure that the complexity of an object x, from an asymptotic point of view, does not depend on accidental peculiarities of the chosen method since any two additively optimal methods f and g are equivalent in the following way: $|C_f^F(x) - C_g^F(x)| \leq c_{f,g}$, for all $x \in X$.

Proposition 2.1.4. *The class of partial functions over the naturals does not have an additively optimal element.* $\diamondsuit$

Proof. First, we show that any additively optimal function f for this class must be unbounded. Suppose that f is an additively optimal function for this class and assume by contradition that it is bounded. Then, it wouldn't be able to describe any natural number above its bound, i.e., there would be an $M \in \mathbb{N}$ such that for all x with $i(x) > M$, $C_f^F(x) = \infty$. However, there are unbounded partial functions, so considering one such function g such that for y with $i(y) > M$, there exists a natural number p such that $g(p) = i(y)$, we have that $C_g^F(y) < \infty$. Hence, the inequality $C_f^F(x) \leq C_g^F(x) + c_{f,g}$ would not be satisfied.

Going back to the main statement, suppose f is an additively optimal function for the class of all partial functions over the naturals. Take an infinite sequence of elements of X, $x_1, x_2, ...$, satisfying $C_f^F(x_n) \geq n$ for all n, which we can do because f is unbounded. Consider the specifying method g defined by $g(n) = i(x_n)$. Clearly, $C_g^F(x_n) = \min\{l(p) : g(p) = i(x_n)\} = \min\{l(p) : i(x_p) = i(x_n)\} \leq l(n) = \log(n) + O(1) \ll n \leq C_f^F(x_n)$ for all n in $\mathbb{N}$. Consequently, f cannot be additively optimal and there is no additively optimal partial function. $\square$

With this proposition, we realize that the class of partial functions is not ideal for our purposes, since the hierarchy of complexities with respect to the partial functions does not have any minimal element. By restricting it a bit, we will discover that actually the class of partial *recursive* functions does have a minimal element, allowing us to obtain a well-behaved hierarchy of complexities.

From now on, we consider the particular problem of describing objects consisting of natural numbers ($X = \mathbb{N}$) in terms of programs consisting of finite strings of 0's and 1's. As such, we will use $i(n) = n$.

We shall encode two strings $x, y \in \Sigma^* = \{0,1\}^*$ into one by the function $[\cdot,\cdot] : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that $[x,y] = 1^{l(x)}0\,x\,y$. Here, l receives a string and not a natural number as before (and returns the length of the string): we will use the same notation for the two functions and assume the reader can distinguish between the two settings. Also, when considering a finite set of strings $S = \{x_1, x_2, ..., x_n\}$ and $x \in \Sigma^*$, by $[S,x]$ we shall mean $[[x_1,[x_2,[...,x_n]]],x]$ and consider $[\{\},x] = [\varepsilon,x] = 0x$.

Lemma 2.1.5. *There is an additively optimal universal partial recursive function.* $\diamondsuit$

Proof. Let ϕ_0 be the function computed by a universal Turing machine U expecting input of the form $[n,p]$. The total program $[n,p]$ should be interpreted as follows: n consists of a self-delimiting encoding (as explained in Section 1.3) of T_n (the n-th element in a standard ordering of all Turing machines) and p is the program U simulates on T_n. So if ϕ_n is the function computed by T_n, this means $\phi_0([n',p']') = \phi_n(p)$, where n' and p' are the strings corresponding to natural numbers n and p and $[n,p]'$ is the natural number corresponding to string $[n,p]$.

From now on, we will not be using the $[n',p']'$ notation anymore to keep the text cleaner and will assume the reader can tell that when we write a string as the argument for a partial recursive function, we actually mean the natural number corresponding to it.

Furthermore, by convention, we set that $U = T_0$ so $\phi_0([0,p]) = \phi_0(p)$.

Suppose $C^F_{\phi_0}(x) = \min\{l(p) : \phi_0(p) = x\} = l(p_1)$ with $\phi_0(p_1) = x$ and $C^F_{\phi_n}(x) = \min\{l(p) : \phi_n(p) = x\} = l(p_2)$ with $\phi_n(p_2) = x$. Since $\phi_0(p_1) = x = \phi_n(p_2) = \phi_0([n,p_2]) = \phi_0\left(1^{l(n)}0\,n\,p_2\right)$, we have that $l(p_1) \leq l\left(1^{l(n)}0\,n\,p_2\right) = l(p_2) + 2l(n) + 1$ because p_1 is by definition a smallest possible description for x. This is to say that

$$C^F_{\phi_0}(x) \leq C^F_{\phi_n}(x) + c_{\phi_n}, \text{where } c_{\phi_n} = 2l(n) + 1.$$

As such, ϕ_0 is an additively optimal universal partial recursive function. $\square$

Definition 2.1.6. Let x be a natural number, $S \subset \mathbb{N}$ a finite set of natural numbers and $\phi : \mathbb{N} \to \mathbb{N}$ a partial recursive function. The *complexity* C^F_ϕ *of* x *conditional to* S is defined by

$$C^F_\phi(x|S) = \min\{l(p) : \phi([S,p]) = x, p \in \mathbb{N}\}$$

and $C^F_\phi(x|S) = \infty$ if there is no such p. Any triple (ϕ, S, p) satisfying $\phi([S,p]) = x$ is called a *description of* x and p is called a *program to compute* x *by* ϕ, given S. ∇

Essentially, $C^F_\phi(x|S)$ gives the size of the smallest natural number p which when inputted into function ϕ, along with the set of natural numbers S, outputs x. Note that this

complexity is defined only for finite sets of natural numbers S so whenever we refer to the conditional version, it will be conditional to a finite set.

It should also be noted that we are again simplifying the notation by writing $\phi([S,p])$ when p and S are, respectively, a natural number and a set of natural numbers. What we actually mean is that p is converted to its corresponding string and the elements of S as well and then the function $[\cdot,\cdot]$ is applied to these. The result of this function is then converted back to a natural number and applied to ϕ.

Theorem 2.1.7. *(Invariance theorem) There is an additively optimal universal partial recursive function ϕ_0 for the class of partial recursive functions to compute x given S. Therefore, $C^F_{\phi_0}(x|S) \leq C^F_{\phi}(x|S) + c_\phi$ for all partial recursive functions ϕ and all x and S, where c_ϕ is a constant depending on ϕ but not on x or S.* $\diamondsuit$

Proof. Let S be s set of natural number $\{y_1, ..., y_m\}$ and, again, let ϕ_0 be the function computed by a universal Turing machine U. This time, when U receives input $[n,[S,p]]$ it simulates Turing machine T_n on input $[S,p]$. So if ϕ_n is the function computed by T_n, this means $\phi_0([n,[S,p]]) = \phi_n([S,p])$.

Like before, let p_1 and p_2 be such that $C^F_{\phi_0}(x|S) = \min\{l(p) : \phi_0([S,p]) = x\} = l(p_1)$ with $\phi_0([S,p_1]) = x$ and $C^F_{\phi_n}(x|S) = \min\{l(p) : \phi_n([S,p]) = x\} = l(p_2)$ with $\phi_n([S,p_2]) = x$. Since $\phi_0\left(1^{l(y_1)}0\, y_1 \ldots 1^{l(y_m)}0\, y_m\, p_1\right) = \phi_0([S,p_1]) = x = \phi_n([S,p_2]) = \phi_0([n,[S,p_2]]) = \phi_0\left(1^{l(n)}0\, n\, 1^{l(y_1)}0\, y_1 \ldots 1^{l(y_m)}0\, y_m\, p_2\right)$, we have that $l\left(1^{l(y_1)}0\, y_1 \ldots 1^{l(y_m)}0\, y_m\, p_1\right) = 2l(y_1) + \ldots + 2l(y_m) + m + l(p_1) \leq 2l(y_1) + \ldots + 2l(y_m) + m + l(p_2) + 2l(n) + 1 = l\left(1^{l(n)}0\, n\, 1^{l(y_1)}0\, y_1 \ldots 1^{l(y_m)}0\, y_m\, p_2\right)$because p_1 is by definition a smallest possible description for x given S. This is to say that $l(p_1) \leq l(p_2)+2l(n)+1$, which implies that $C^F_{\phi_0}(x) \leq C^F_{\phi_n}(x) + c_{\phi_n}$, where $c_{\phi_n} = 2l(n) + 1$. □

We can now use the fact that there exist these additively optimal universal partial recursive functions to fix one of them and simplify our notation. This can be done because any two additively optimal universal partial recursive functions are equal up to a fixed constant. In fact, using the theorem above, we can prove that for every pair ϕ, ϕ' of such functions, there is a fixed constant $c_{\phi,\phi'}$, depending only on ϕ and ϕ', such that for all x and S we have that

$$|C^F_{\phi}(x|S) - C^F_{\phi'}(x|S)| \leq c_{\phi,\phi'}.$$

Definition 2.1.8. Fix an additively optimal universal ϕ_0, which we shall call the *reference function for C^F*, and dispense with the subscript by defining the *conditional Kolmogorov complexity based on partial recursive functions* $C^F(\cdot|\cdot)$ as

$$C^F(x|S) = C^F_{\phi_0}(x|S), \text{ for all } x \text{ and } S.$$

We also fix the Turing machine U that computes ϕ_0 and call it the *reference machine*. The *unconditional Kolmogorov complexity based on partial recursive functions* $C^F(\cdot)$ is defined by

$$C^F(x) = C^F(x|\{\}) \text{ for all } x. \nabla$$

Results

Having finally settled on a clear definition for this complexity, we can now present some results. This first theorem shows some of its most basic properties. The first inequality is based on the fact that you can always describe a string by just using it directly and so its length is an upper bound for the unconditional Kolmogorov complexity. The second takes into account that describing a number conditional to a set of numbers can be done by just ignoring the set, so the conditional complexity has the unconditional one as an upper bound.

Theorem 2.1.9. *There exist constants c_1 and c_2 such that for all $x \in \mathbb{N}$ and $S \subset \mathbb{N}$,*

$$C^F(x) \leq l(x) + c_1 \text{ and } C^F(x|S) \leq C^F(x) + c_2. \diamondsuit$$

Proof. Proving the first statement is quite simple: consider a Turing machine T that, for input x, outputs x and let ϕ be the partial recursive function it computes. Then, for all naturals, we have that $C^F_\phi(x) = \min\{l(p) : \phi(p) = x\} = \min\{l(p) : p = x\} = l(x)$. By the invariance theorem (Theorem 2.1.7), we get that $C^F_{\phi_0}(x) \leq C^F_\phi(x) + c_\phi = l(x) + c$.

Now we prove the second inequality: construct a Turing machine T that for all S and z computes output x on input $[S, z]$ if and only if the universal reference machine U computes output x for input z, i.e., $T([S, z]) = x \Leftrightarrow U(z) = x$. Let ϕ be the partial recursive function computed by T. Then, $C^F_\phi(x|S) = \min\{l(p) : \phi([S, p]) = x\} = \min\{l(p) : \phi_0(p) = x\} = C^F_{\phi_0}(x)$. Again, by applying the invariance theorem, we get that $C^F_{\phi_0}(x|S) \leq C^F_\phi(x|S) + c_\phi = C^F_{\phi_0}(x) + c_\phi$. □

As we've mentioned before, some numbers will be quite easy to describe in a short program, i.e., are easily compressible, whereas others look so irregular that it seems there is no way to compress them to arrive at a description shorter than themselves. In fact, for each n there are 2^n possible binary strings of length n but only $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ possible shorter descriptions.

It follows from the above that for each length, there is at least one incompressible number and, for any set of naturals S, there is a number x of corresponding string length n such that $C^F(x|S) \geq l(x)$.

We can actually improve this results by using a counting argument, as shown in the theorem below.

Theorem 2.1.10. *(Incompressibility theorem) Let c be a positive integer. For each fixed S, every finite set A of cardinality m has at least $m(1-2^{-c})+1$ elements x with $C^F(x|S) \geq log(m) - c$.* $\diamondsuit$

Proof. The number of programs of length less than $\log(m) - c$ is $\sum_{i=0}^{\log(m)-c-1} 2^i = 2^{\log(m)-c} - 1$. As such, there are at least $m - \left(2^{\log(m)-c} - 1\right) = m - m2^{-c} + 1$ elements in A that have no program of length less than $\log(m) - c$. $\square$

Example 2.1.11. Consider the set $A = \{x : l(x) = n\}$, which has 2^n elements. By Theorem 2.1.9, we know that $C^F(x) \leq n + c$ for some fixed constant c and all $x \in A$. Applying the incompressibility theorem for any S, we get that $C^F(x|S) \geq n - c$ for another constant c and applying the second inequality of Theorem 2.1.9 we get that $C^F(x) \geq n - c'$ for some c'. As such, we realize that this trivial estimate is actually quite sharp. The deeper reason is that since there are few short programs, there can be only few objects of low complexity. $\square$

Note that for most applications of Kolmogorov complexity, Theorems 2.1.7, 2.1.9 and 2.1.10 give us all the information we need.

Now, we will look at the unconditional version of C^F as an integer function $C^F : \mathbb{N} \to \mathbb{N}$ and study its behaviour. As we have mentioned, Theorem 2.1.9 gives us an upper bound: $C^F(x) \leq l(x) + c$. It is easy to see that $l(x) + c$ is actually computable and grows monotonically to infinity. It is also the least so upper bound for $C^F(x)$. However, we will next prove that the greatest monotonic lower bound also grows to infinity but is incomputable, as is C^F.

Theorem 2.1.12. *Define function $m : \mathbb{N} \to \mathbb{N}$ by $m(x) = min\{C^F(y) : y \geq x\}$. That is, m is the greatest monotonic increasing function bounding C^F from below.*

1. *The function m is unbounded.*

2. *The function C^F is unbounded.*

3. *For any partial recursive function ϕ that goes monotonically to infinity from some x_0 onward, we have $m(x) < \phi(x)$ except for finitely many x. In other words, although m goes to infinity, it does so more slowly than any unbounded partial recursive function.* $\diamondsuit$

Proof. 1. Since the number of possible programs with length i is finite, the number of natural numbers with complexity i is also finite. Hence, the number of natural numbers with complexity equal or greater to i is infinite. As such, for each i there is a least an x_i such that for all $x > x_i$, $C^F(x) \geq i$. Clearly, for all i we have $x_{i+1} \geq x_i$. Consider an $x_i < x \leq x_{i+1}$. Since $x > x_i$, $C^F(x) \geq i$. Also, since $x \leq x_{i+1}$, $C^F(x) < i + 1$, i.e., $C^F(x) \leq i$. Hence, $C^F(x) = i$ and $m(x) = \min\{C^F(y) : y \geq x\} = C^F(x) = i$ so m is unbounded.

2. Since m is C^F's greatest monotonic lower bound and it is unbounded, so is C^F.

3. Start by assuming that the statement is false: there is a monotonic non-decreasing unbounded partial recursive function ϕ satisfying $\phi(x) \leq m(x)$ for infinitely many x. Consider the domain of definition of ϕ, $A = \{x \in \mathbb{N} : \phi(x) < \infty\}$. It is an infinite recursively enumerable set so, by Theorem 1.2.4, it contains an infinite recursive subset B. Now, define the function

$$\psi(x) = \begin{cases} \phi(x), & x \in B \\ \phi(y) \text{ with } y = \max\{z : z \in B, z < x\}, & \text{otherwise} \end{cases}.$$

This function is total recursive and goes monotonically to infinity. Also, $\psi(x) \leq \phi(x) \leq m(x)$ for infinitely many x.

Define another function M such that $M(a) = \max\{x : C^F(x) \leq a\}$. $M(a) + 1 = \min\{x : m(x) > a\}$ since $C^F(M(a)) = a$ and by definition $C^F(M(a) + 1) > a$. It is also easy to verify that the function $G(a) = \max\{x : \psi(x) \leq a + 1\}$ is total recursive and $G(a) \geq \min\{x : m(x) > a\} = M(a) + 1 > M(a)$ for infinitely many a's, i.e., $G(a) > M(a)$ for infinitely many a's. In other words, $C^F(G(a)) > a$ for infinitely many a's. However, by Theorems 2.1.7 and 2.1.9, $C^F(G(a)) \leq C^F_G(G(a)) + c \leq l(a)+c' = \log(a)+c''$ for some constants c, c' and c''. This implies that $\log(a)+c'' > a$ for infinitely many a's, which is obviously impossible.

□

Now we turn our attention to proving that C^F is incomputable.

Theorem 2.1.13. *(Incomputability theorem) No partial recursive function ϕ defined on an infinite set of points can coincide with $C^F(x)$ over the whole of its domain of definition. Hence, C^F is not a computable function.* ♢

Proof. This proof is related to that of the third item from the previous theorem. Suppose there is a partial recursive function ϕ defined on an infinite set of points that coincides

with C^F over the whole of its domain of definition. Then by Theorem 1.2.4, there is an infinite recursive subset A of the domain of definition of ϕ. The function ψ such that $\psi(m) = \min\{x \in A : C^F(x) \geq m\}$ is partial recursive because $C^F(x) = \phi(x)$ on A. In addition, by definition of ψ, $C^F(\psi(m)) \geq m$. However, by the invariance theorem (Theorem 2.1.7, which we can apply because ψ is a partial recursive function),

$$C^F(\psi(m)) \leq C^F_\psi(\psi(m)) + c \leq l(m) + c' = \log(m) + c'',$$

for some constants c, c' and c''.

Hence, $m \leq \log(m)$ up to a constant independent of m, which is false from some m onward and shows that our assumption was incorrect. □

Even though we cannot compute this complexity function, we can approximate it with a computable function, as shown below.

Proposition 2.1.14. *There is a total recursive function $\phi(t, x)$, monotonic and decreasing in t, such that $lim_{t\to\infty}\phi(t, x) = C^F(x)$.* ♢

Proof. Construct ϕ as follows:

- For each x, it's known that the Kolmogorov complexity of x is at most $l(x) + c$ (Theorem 2.1.9) so run the reference Turing machine U in the proof of Theorem 2.1.7 for t steps on each program p of length at most $l(x) + c$;
- If for any such input p the computation halts with output x, then define the value of $\phi(t, x)$ as the length of the shortest such p, otherwise define it as equal to $l(x) + c$.

It is easy to observe that $\phi(t, x)$ is recursive, total, and monotonically decreasing with t (for all x, $\phi(t_1, x) \leq \phi(t_2, x)$ if $t_1 > t_2$). Looking at $\lim_{t\to\infty}\phi(t, x)$, the limit exists, since for each x there exists a t such that U halts with output x after computing t steps starting with input p and satisfying $l(p) = C^F(x)$.

□

As one would imagine, there are plenty more results regarding algorithmic complexity we could present. However, these are the ones we felt were relevant to our studies so others are not included here. For more, the reader is referred to [13, Chapter 2].

2.1.2 Kolmogorov Complexity Based on Turing Machines

We continue our study of Kolmogorov complexity by now looking at a presentation based on the work of Lance Fortnow, as seen for instance in [8]. This time, the concept of

Kolmogorov complexity is presented using the ideas from computer science that originated it, creating a perhaps more intuitive definition. The main difference in this subsection is that instead of using natural numbers or partial recursive functions, we use strings and Turing machines.

Definition 2.1.15. Let $x \in \Sigma^*$ be a string, $S \subset \Sigma^*$ a finite set of strings and T a Turing machine. The *complexity* C_T^M *of* x *conditional to* S is defined as

$$C_T^M(x|S) = \min\{l(p) : T([S,p]) = x, p \in \Sigma^*\}$$

and $C_T^M(x|S) = \infty$ if there is no such p. ∇

Essentially, $C_T^M(x)$ gives the size of the smallest string p which when inputted into Turing machine T, outputs x. So, with this presentation, we can see that we get a definition closer to the idea which originated this complexity (if p is a program and f a computer, then $C_f^M(x)$ is the minimal length of a program for f to compute output x). The M in superscript stands for the fact that we are now using Turing machines as our description methods.

Note that for this definition, we use the function $[\cdot,\cdot]$ as it was originally defined, receiving two strings and returning another one. Also, by $T([S,p]) = x$, we mean that Turing machine T outputs x when it receives input $[S,p]$.

It is easy to prove that the two presentations of Kolmogorov complexity we have looked at are equivalent, as shown in the two propositions below. We just need to keep in mind that there is a unique correspondence between partial recursive functions and Turing machines (as discussed in Section 1.2) and between natural numbers and strings (as presented in Subsection 2.1.1).

For the next two propositions, we will use the notation x' to denote the natural number corresponding to string x, if x is a string, or the string corresponding to natural number x, if x is a natural number. This can be extended to sets: $S' = \{s_1, ..., s_n\} = \{s_1', ..., s_n'\}$.

Furthermore, note that, depending on the complexity we are using, we will use the function l in one of its two forms: receiving a string and returning its length or receiving a natural number and returning the length of the string corresponding to it.

Proposition 2.1.16. *For any Turing machine T, string x and finite set of strings S, there is a partial recursive function ϕ such that $C_T^M(x|S) = C_\phi^F(x'|S')$.* $\diamondsuit$

Proof. We simply have to consider ϕ as the partial recursive function computed by T. Indeed, $C_\phi^F(x'|S') = \min\{l(p) : \phi([S',p]) = x'\} = \min\{l(p') : T([S,p']) = x\} = C_T^M(x|S)$. $\square$

Proposition 2.1.17. *For any partial recursive function ϕ , natural number x and finite set of natural numbers S, there is a Turing Machine T such that $C_\phi^F(x|S) = C_T^M(x'|S')$.* $\diamondsuit$

Proof. Consider T as the Turing Machine that computes ϕ. Then, $C_T^M(x'|S') = \min\{l(p) : T([S', p]) = x'\} = \min\{l(p') : \phi([S, p']) = x\} = C_\phi^F(x|S)$. □

Taking these two results into account, we will be able to prove any statement regarding the Kolmogorov complexity based on partial recursive functions for the one based on Turing machines and vice-versa. Therefore, we will write the statements equivalent to the ones in Subsection 2.1.1 but omit the proofs as they all come directly from the application of Proposition 2.1.17. However, first we must define a special type of Turing machine.

Definition 2.1.18. Let $\mathcal{C}$ be a subclass of the Turing machines. A machine T is *additively optimal* (a special type of universality) for $\mathcal{C}$ if it belongs to $\mathcal{C}$ and if for every Turing machine $V \in \mathcal{C}$ there is a constant $c_{T,M}$ such that for all x,

$$C_T^M(x) \leq C_V^M(x) + c_{T,V}.$$

As such, $c_{T,V}$ depends only on T and V but not x. ∇

Theorem 2.1.19. *There is an additively optimal universal Turing machine T_0 for the class of Turing machines that compute x given S. Therefore, $C_{T_0}^M(x|S) \leq C_T^M(x|S) + c_T$ for all Turing machines T and all x and S, where c_T is a constant depending on T but not on x or S.* $\diamondsuit$

The theorem above is essential to fix one additively optimal universal Turing machine and simplify our notation. This can be done because, as with additively optimal partial recursive functions, any two additively optimal universal Turing machines are equal up to a fixed constant. In fact, using Theorem 2.1.7, we can prove that for every pair T, T' of such machines, there is a fixed constant $c_{T,T'}$, depending only on T and T', such that for all x and S we have that

$$|C_T^M(x|S) - C_{T'}^M(x|S)| \leq c_{T,T'}.$$

Definition 2.1.20. Fix an additively optimal universal T_0, which we shall call the *reference machine for C^M*, and dispense with the subscript by defining the *conditional Kolmogorov complexity based on Turing machines $C^M(\cdot|\cdot)$* as

$$C^M(x|S) = C_{T_0}^M(x|S), \text{ for all } x \text{ and } S.$$

The *unconditional Kolmogorov complexity based on Turing machines* $C^M(\cdot)$ is defined as

$$C^M(x) = C^M(x|\{\}) \text{ for all } x. \nabla$$

Again, note that this complexity is defined only for finite sets of strings S so whenever we refer to the conditional version, it will be conditional to a finite set.

Results

Theorem 2.1.21. *There exist constants* c_1 *and* c_2 *such that for all* $x \in \Sigma^*$ *and* $S \subset \Sigma^*$,

$$C^M(x) \leq l(x) + c_1 \text{ and } C^M(x|S) \leq C^M(x) + c_2. \diamondsuit$$

Theorem 2.1.22. *Let* c *be a positive integer. For each fixed* S*, every finite set* A *of cardinality* m *has at least* $m(1 - 2^{-c}) + 1$ *elements* x *with* $C^M(x|S) \geq log(m) - c$. $\diamondsuit$

In the sequel, given a natural number x, by $C^M(x)$ we will mean $C^M(x')$, where x' is the binary string corresponding to x according to the mapping in the beginning of Subsection 2.1.1.

Theorem 2.1.23. *Define a function* m *by* $m(x) = min\{C^M(y) : y \geq x\}$. *That is,* m *is the greatest monotonic increasing function bounding* C^M *from below.*

1. *The function* m *is unbounded.*

2. *The function* C^M *is unbounded.*

3. *For any partial recursive function* ϕ *that goes monotonically to infinity from some* x_0 *onward, we have* $m(x) < \phi(x)$ *except for finitely many* x*. In other words, although* m *goes to infinity, it does so more slowly than any unbounded partial recursive function.* $\diamondsuit$

Theorem 2.1.24. *No partial recursive function* ϕ *defined on an infinite set of points can coincide with* $C^M(x)$ *over the whole of its domain of definition. Hence,* C^M *is not a computable function.* $\diamondsuit$

Proposition 2.1.25. *There is a total recursive function* $\phi(t, x)$*, monotonically decreasing in* t*, such that* $lim_{t\to\infty}\phi(t, x) = C^M(x)$. $\diamondsuit$

2.2 Prefix Kolmogorov Complexity

Although Kolmogorov complexity has its strengths, it fails to fulfil on some aspects, such as:

- It is not *subadditive*: $C^F([x,y]) \leq C^F(x) + C^F(y) + c$, for some constant c, does not always hold;
- It is not *monotonic over prefixes*: $C^F(x) \leq C^F(xy)$ is also not always true (here, by xy, we mean the natural number associated with the concatenation of the strings corresponding to natural numbers x and y);
- In the development of the theory of *random infinite sequences,* it would be natural to identify infinite random sequences with infinite sequences of which all finite initial segments are random. As it turns out, no infinite sequence satisfies this criteria, due to complexity oscillations.

As such, a new type of complexity, designed to answer some of these needs, was developed. Again, we will present it based on [13]. Obviously, years of work and different approaches went into trying to find the best way to remedy the shortcomings of Kolmogorov complexity. However, since it is not the focus of this thesis to specify these details, we will simply say that it was discovered that a complexity based on prefix-codes (as introduced in Section 1.3) solved the problems above (except for nonmonotonicity) and some others. As a result, this new type of complexity is regarded by many as a sort of standard algorithmic complexity. Nevertheless, one should keep in mind that different applications require different versions of algorithmic complexity, and all of these are natural for their own purposes.

For this Kolmogorov complexity, we return to the natural numbers, as in C^F, but will consider a particular kind of partial recursive function aimed at capturing the advantages of prefix-codes, where no code-word is the prefix of another code-word.

Definition 2.2.1. A *partial recursive prefix function* $\phi : \mathbb{N} \to \mathbb{N}$ is a partial recursive function such that if $\phi(p) < \infty$ and $\phi(q) < \infty$, then p' is not a proper prefix of q', where p' and q' are, respectively, the strings associated with p and q. ∇

In short, a partial recursive prefix function is a partial recursive function with a prefix-free domain of definition.

To construct the Turing machines that compute these functions, first consider the standard enumeration of Turing machines $T_1, T_2, ...$, constructed as in Section 1.2. Also, let $\phi_1, \phi_2, ...$ be the corresponding enumeration of partial recursive functions, where every

partial recursive function is present. We will now construct a new enumeration for *prefix Turing machines*, machines that compute prefix partial recursive functions.

The way to do this is to change every Turing machine T computing ϕ to a machine M computing a partial recursive prefix function φ, where $\phi = \varphi$ if ϕ was already a partial recursive prefix function. This is done using the procedure below. Contrary to T, which is presented with an input from Σ^* delimited by end markers, machine M is presented with a potentially infinite binary input sequence $b_1 b_2 ...$, which it reads from left to right without backing up.

Definition 2.2.2. A *halting input*, or *program*, of prefix Turing machine M is an initial segment $b_1 b_2 ... b_m$ of the input sequence $b_1 b_2 ...$ such that M halts after reading b_m and before reading b_{m+1}. ∇

There are no other possible inputs for a prefix Turing machine. Due to this, the set of programs for a prefix Turing machine M is prefix-free and self-delimiting (see Section 1.3), since M determines the end of each program without reading the next symbol.

The computation of M on input $b_1 b_2 ...$ is given by the following procedure that modifies the operation of the original T:

1. **Set** $p = \varepsilon$.

2. Dovetail all computations of T computing $\phi(pq)$, for all $q \in \Sigma^*$. (Let q_j be the j-th string of Σ^* according to a fixed enumeration of Σ^*. Dovetailing here means executing consecutive stages $i = 1, 2, ...$, such that in the i-th stage we do one next step of the computation of $\phi(pq_j)$ for all j with $j \leq i$.)

   If $\phi(pq) < \infty$ is the first halting computation **then** go to step 3.

3. **If** $q = \varepsilon$ **then** output $\phi(p)$ and halt (these p are already self-delimiting) **else** read the next input bit (call it b) and set $p = pb$ (if this case happens for every initial segment of the input, then the machine never halts).

   Go to step 2.

This construction produces an effective enumeration of all prefix Turing machines $M_1, M_2, M_3, ...$ The corresponding list of partial recursive functions $\varphi_1, \varphi_2, \varphi_3, ...$ has all, and only, partial recursive prefix functions. This allows us to write the prefix equivalent to the invariance theorem (Theorem 2.1.7).

From now on, whenever we are using a partial recursive prefix function φ, we will write K_φ instead of C_φ^F. This is the standard terminology for this complexity.

Theorem 2.2.3. *There exists an additively optimal universal partial recursive prefix function φ_0 such that for every partial recursive prefix function φ there is a constant c_φ such that*

$$K_{\varphi_0}(x|S) \leq K_\varphi(x|S) + c_\varphi, \text{ for all } x \in \mathbb{N}, S \subset \mathbb{N}. \diamondsuit$$

Proof. First, we must prove that there exists a universal prefix machine. Using techniques similar to those in Section 1.2, we can arrive at a Turing machine U such that $U([n,[S,p]]) = M_n(S,p)$, where M_n is the n-th machine in the enumeration we constructed above. Let S be the set of natural numbers $\{y_1, ..., y_m\}$ and let φ_0 be the function computed by U.

Consider $K_{\varphi_0}(x|S) = \min\{l(p) : \varphi_0([S,p]) = x\} = l(p_1)$ with $\varphi_0([S,p_1]) = x$ and $K_{\varphi_n}(x|S) = \min\{l(p) : \varphi_n([S,p]) = x\} = l(p_2)$ with $\varphi_n([S,p_2]) = x$. Since $\varphi_0\left(1^{l(y_1)}0\, y_1 \ldots 1^{l(y_m)}0\, y_m\, p_1\right) = \varphi_0([S,p_1]) = x = \varphi_n([S,p_2]) = \varphi_0([n,[S,p]]) = \varphi_0\left(1^{l(n)}0\, n\, 1^{l(y_1)}0\, y_1 \ldots 1^{l(y_m)}0\, y_m\, p_2\right)$, we have that $l\left(1^{l(y_1)}0\, y_1 \ldots 1^{l(y_m)}0\, y_m\, p_1\right) = 2l(y_1) + \ldots + 2l(y_m) + m + l(p_1) \leq l\left(1^{l(n)}0\, n\, 1^{l(y_1)}0\, y_1 \ldots 1^{l(y_m)}0\, y_m\, p_2\right) = 2l(y_1) + \ldots + 2l(y_m) + m + l(p_2) + 2l(n) + 1$ because p_1 is by definition a smallest possible description for x given S. This is to say that $l(p_1) \leq l(p_2) + 2l(n) + 1$, which implies that $K_{\varphi_0}(x) \leq K_{\varphi_n}(x) + c_{\varphi_n}$, where $c_{\varphi_n} = 2l(n) + 1$. □

Just as for additively optimal universal partial recursive functions, any two φ_1, φ_2 additively optimal universal partial recursive prefix functions are equivalent in the sense that, for all naturals x and finite set of naturals S, there is a positive constant c_{φ_1,φ_2} such that

$$|K_{\varphi_1}(x|S) - K_{\varphi_2}(x|S)| \leq c_{\varphi_1,\varphi_2}.$$

Due to this, we can, as before, simplify our notation for prefix complexity.

Definition 2.2.4. Fix an additively optimal partial recursive prefix function φ_0 as the standard reference and U in the proof of Theorem 2.2.3 as the *reference prefix machine*. Define the *conditional prefix complexity* $K(\cdot|\cdot)$ as

$$K(x|S) = K_{\varphi_0}(x|S), \text{ for all } x \text{ and } S.$$

Also, we define the *unconditional prefix complexity* as $K(x) = K(x|\{\})$ for all natural numbers x. ∇

Results

Proposition 2.2.5. K *is a subadditive function: for all natural numbers x and y,*

$$K([x,y]) \leq K(x) + K(y) + c \text{ for some constant } c. \diamondsuit$$

Proof. Let φ_0 be the reference additively optimal partial recursive prefix function and consider strings p_1 and p_2 such that $K(x) = l(p_1)$ and $K(y) = l(p_2)$ with $\varphi_0(p_1) = x$ and $\varphi_0(p_2) = y$, i.e., they are minimal descriptions of x and y. Since the set of programs of U is a prefix-code, we can modify U to a machine V that first reads p_1 and computes x, then reads p_2 and computes y, and finally computes and outputs $[x, y]$. This means that, if $V = M_n$ in the enumeration of prefix Turing machines and M_n computes the partial recursive prefix function φ, then $\varphi_0([n,[p_1,p_2]]) = \varphi([p_1,p_2]) = [x,y]$. Hence, $K([x,y]) \leq l([n,[p_1,p_2]]) = l\left(1^{l(n)}0\, n\, 1^{l(p_1)}0\, p_1 p_2\right) = 2l(n) + 2l(p_1) + l(p_2) + 2 = K(x) + K(y) + c$, with $c = 2l(n) + l(p_1) + 2$. □

So we have proved that K has a property which C^F doesn't but we should better investigate the relation between these two types of complexity. Next, we present two of the most important results in this regard. Namely, that C^F and K are asymptotically equal and how to define C^F using K.

Proposition 2.2.6. *For all x and S,*

$$C^F(x|S) \leq K(x|S) \leq C^F(x|S) + 2log(C^F(x|S)), \text{ up to additive constant terms. } \diamondsuit$$

Proof. To prove the first inequality, we simply remark that the prefix partial recursive functions are a restriction on the notion of partial recursive functions, hence applying Theorem 2.1.7 suffices.

To prove the second inequality, suppose $C^F(x|S) = l(p)$ for some program p such that $\varphi_0([S,p]) = x$ and recall that $\overline{l(p)}p$ is a self-delimiting encoding for p, where $\overline{l(p)} = [l(p),\varepsilon] = 1^{l(l(p))}0\, l(p)$. Hence, there is a prefix Turing machine M such that $M\left(\left[S,\overline{l(p)}p\right]\right) = x$, i.e., $\overline{l(p)}p$ is a description of x conditional to S. Since $K(x|S)$ is the smallest such description, we know that $K(x|S) \leq l\left(\overline{l(p)}p\right) = 2l(l(p)) + l(p) + 1 = 2l(C^F(x|S)) + C^F(x|S) + 1 \leq C^F(x|S) + 2\log(C^F(x|S)) + c$, for some $c \in \mathbb{R}^+$. □

The upper bound for K using C^F could be to improved to $K(x|S) \leq C^F(x|S) + l^*(CL(x|S)) + c$, where $l^*(x) = \begin{cases} l(x) + l^*(l(x)), & l(x) > 1 \\ l(x), & l(x) = 0,1 \end{cases}$. However, this improvement

would be irrelevant for our purposes so we will keep the simpler estimate given by the proposition above.

Lemma 2.2.7. *C^F is the unique (up to an additive constant) function satisfying*

$$C^F(x) = min\{i : K(x|i) \leq i\} + O(1) = K(x|C^F(x)) + O(1). \diamondsuit$$

Proof. First, we prove that $K(x|C^F(x)) + O(1) \leq C^F(x)$. Simply note that the shortest Turing machine program p for x has length $C^F(x)$ and there is a prefix machine that given $C^F(x)$, computes x from p.

Next, it is obvious that $\min\{i : K(x|i) \leq i\} + O(1) \leq K(x|C^F(x)) + O(1)$ (we just need to consider $i = C^F(x)$ and use the inequality previously proven).

Now, we prove that $C^F(x) \leq \min\{i : K(x|i) \leq i\} + O(1)$. This is the same as proving that if $K(x|i) \leq i$, then $C^F(x) \leq i + O(1)$ (it is true for the minimum if and only if if it true for all possible values). Assume that x is computed by the reference prefix machine U, given i, from an input p, with $l(p) \leq i$. Consider another Turing machine T (not necessarily of the prefix type) such that, if $i - l(p) - 1 > 0$, when presented with input $0^{i-l(p)-1}1p$, extracts i from the input length and simulate U on p. If $i - l(p) - 1 < 0$, then it receives just p as input. By Theorem 2.1.7, $C^F(x) \leq C^F_\phi(x) \leq i$, up to additive constants, where ϕ is the function computed by machine T.

Hence, we have proved that $\min\{i : K(x|i) \leq i\} + O(1) \leq K(x|C^F(x)) + O(1) \leq C^F(x) \leq \min\{i : K(x|i) \leq i\} + O(1)$, which implies the statement from the lemma. □

We are now capable of writing some results about prefix complexity inspired by the ones for Kolmogorov complexity. The first two results are quite simple to prove, as they are a direct consequence of Proposition 2.2.6 and the corresponding results for Kolmogorov complexity based on partial recursive functions.

Proposition 2.2.8. *There exist constants c_1 and c_2 such that for all $x \in \mathbb{N}$ and $S \subset \mathbb{N}$,*

$$K(x) \leq l(x) + 2log(l(x)) + c_1 \text{ and } K(x|S) \leq K(x) + 2log(K(x)) + c_2. \diamondsuit$$

Proof. By Proposition 2.2.6, we know that $K(x|S) \leq C^F(x|S) + 2\log(C^F(x|S)) + c$ and by Theorem 2.1.9, we know that $C^F(x) \leq l(x) + c_1$ and $C^F(x|S) \leq C^F(x) + c_2$. Hence, we get that $K(x) \leq l(x) + 2\log(l(x)) + c'_1$ and $K(x|S) \leq C^F(x) + 2\log(C^F(x)) + c'_2$. However, we also know by Proposition 2.2.6 that $C^F(x) \leq K(x) + c'$ so applying that to the second inequality gives us $K(x|S) \leq K(x) + 2\log(K(x)) + c_3$. □

The quantitative relations between K and C^F show that there must be many incompressible strings with respect to K. For C^F, for each n, the maximal complexity of strings

of length n equals n, up to a fixed constant. With K, as we have seen, there is not such a simple result.

As such, we cannot arrive at a theorem which mimics the incompressibility theorem (Theorem 2.1.10) for C^F. There are some results regarding the maximal prefix complexity of a string of length n and the distribution of description lengths, but these are not revelant for this thesis so the reader is referred to [13, Section 3.3] for more information on this topic.

As an integer function, $K : \mathbb{N} \to \mathbb{N}$ has the same behaviour as $C^F : \mathbb{N} \to \mathbb{N}$. Namely, it is unbounded and incomputable but we are able to approximate it with a total recursive function. We omit the proofs of the following theorems because they would be the similar to the proofs of Theorems 2.1.12, 2.1.13 and 2.1.14 but with C^F replaced by K.

Theorem 2.2.9. *Define a function m by $m(x) = min\{K(y) : y \geq x\}$. That is, m is the greatest monotonic increasing function bounding K from below.*

1. *The function m is unbounded.*
2. *The function K is unbounded.*
3. *For any partial recursive function ϕ that goes monotonically to infinity from some x_0 onward, we have $m(x) < \phi(x)$ except for finitely many x. In other words, although m goes to infinity, it does so more slowly than any unbounded partial recursive function.* $\diamondsuit$

Theorem 2.2.10. *No partial recursive function ϕ defined on an infinite set of points can coincide with $K(x)$ over the whole of its domain of definition. Hence, K is not a computable function.* $\diamondsuit$

Theorem 2.2.11. *There is a total recursive function $\phi(t, x)$, monotonic decreasing in t, such that $lim_{t\to\infty}\phi(t, x) = K(x)$.* $\diamondsuit$

Next, we could have a section regarding a presentation of prefix Kolmogorov complexity based on Turing machines, as we did in Subsection 2.1.2. However, we believe this would add nothing new to our studies since, like with the two presentations of Kolmogorov complexity, we would be able to prove they are equivalent.

2.3 Kolmogorov Proof Complexity

In this section, we will look at a fairly unexplored presentation of proof complexity, one inspired by Kolmogorov complexity. Recall that the Kolmogorov complexity of an object

is initially defined as the length of the shortest program from which the object can be effectively reconstructed. As such, we give the following initial definition of *proof complexity*:

Definition 2.3.1. The proof complexity of a formula is the length of the shortest derivation in a given deductive system that effectively proves the formula. ∇

As with Kolmogorov complexity, it will be important to prove every formula in our universe in a similar way to keep coherence. By this we mean using the same deductive system (and, therefore, the same induced propositional proof system, as defined in Section 1.4) or using a class of deductive systems that doesn't allow for much difference between the size of proofs for the same formula within its members.

It should be noted that we will only be concerning ourselves with classical propositional logic since, firstly, we want to restrict our attention to one specific field, so as to try to get the most interesting results possible and, secondly, propositional logic is general enough to be applied in a wide range of situations but also simple enough as to not make our task too difficult. It remains to be investigated how far one could go with the concept of proof complexity in another field of logic.

As in Section 1.4, fix a set of propositional variables Ξ and denote by L the set of propositional formulas over Ξ. Recall that L^+ is the set of all finite sequences over the elements of L, except for the empty sequence. As such, define $|\cdot| : L^+ \to \mathbb{N}$ as the length of the sequence in question, i.e., the number of formulas it has.

Also recall that for a given deductive system D, we denote by Δ_D its induced propositional proof system. We are now ready to define the complexity CL_D of a formula φ in L with respect to the deductive system D as

$$CL_D(\varphi) = \min\{|d| : \vdash^d_{\Delta_D} \varphi, d \in L^+\}$$

and $CL_D(\varphi) = \infty$ if there is no such d.

In logic terminology, $CL_D(\varphi)$ is the minimal length of a proof d for formula φ in deductive system D. As such, the case of $CL_D(\varphi) = \infty$ can be seen as trying to prove a formula that is not derivable without hypothesis in D. Naming this function CL comes from the fact that it is a complexity measure for logic.

Without further specifying these concepts, we can prove one simple result, stated below.

Proposition 2.3.2. *$CL_D(\varphi) = 1$ if and only if φ is a substitution instance of an axiom in D.* $\diamondsuit$

Proof. Consider a formula φ and a deductive system D.

($\Rightarrow$) Let $CL_D(\varphi) = 1$. This means that the smallest proof for φ over D is just one line long and, as such, that line has to be a formula ψ that is an axiom (the use of a Frege rule with premises would imply the existence of previous lines). But the final line in a derivation always has to be the formula we want to derive, in this case φ. Hence, $\psi = \varphi$ and φ is an axiom.

($\Leftarrow$) Let φ be an axiom. The smallest derivation for φ over D is just one line with φ itself, i.e., $CL_D(\varphi) = 1$. □

2.3.1 Frege Systems

We now move on to detailing the kind of deductive systems we will be working with. As we mentioned before, we need a class of systems which has some kind of relation between the length of the derivations a formula can have for each of its members. We will see that Frege systems satisfy this requirement. These systems also have the advantage of being very commonly used, which allows us to benefit from all the research that has already been done about them.

This subsection is based on the definitions and results of [1, 3, 6].

Definition 2.3.3. Let $\psi_1, ..., \psi_k, \varphi$ be formulas. A deductive system $\mathcal{F}$ is *complete* if $\psi_1, ..., \psi_k \vdash_{\Delta_{\mathcal{F}}} \varphi$ whenever $\psi_1, ..., \psi_k \models \varphi$.

A *Frege system* is a finite and complete deductive system. ∇

Example 2.3.4. A common example of a Frege system is the *Hilbert Propositional Calculus*. It has as its axiom schemas the formulas below and has Modus Ponens as its only Frege rule.

Axiom schemas:

- $(\xi_1 \wedge \xi_2) \rightarrow \xi_1$;
- $(\xi_1 \wedge \xi_2) \rightarrow \xi_2$;
- $\xi_1 \rightarrow (\xi_2 \rightarrow (\xi_1 \wedge \xi_2))$;
- $\xi_1 \rightarrow (\xi_2 \rightarrow \xi_1)$;
- $(\xi_1 \rightarrow \xi_2) \rightarrow (\xi_1 \rightarrow (\xi_2 \rightarrow \xi_3)) \rightarrow (\xi_1 \rightarrow \xi_3)$;
- $\xi_1 \rightarrow \neg\neg\xi_1$;
- $\neg\neg\xi_1 \rightarrow \xi_1$;

- $\xi_1 \to (\xi_1 \vee \xi_2)$;
- $\xi_2 \to (\xi_1 \vee \xi_2)$;
- $(\xi_1 \to \xi_3) \to ((\xi_2 \to \xi_3) \to ((\xi_1 \vee \xi_2) \to \xi_3))$;
- $(\xi_1 \to \xi_2) \to ((\xi_1 \to \neg\xi_2) \to \neg\xi_2)$,

where ξ_1 and ξ_2 are propositional variables. □

Note that from now on, when we consider a Frege system $\mathcal{F}$ and write

$$S \vdash^d_{\mathcal{F}} \varphi,$$

we actually mean $S \vdash^d_{\Delta_{\mathcal{F}}} \varphi$, where $\Delta_{\mathcal{F}}$ is the propositional proof system induced by $\mathcal{F}$.

Next, we will see exactly why Frege systems are appropriate for our situation in two results adapted from [6, Theorem 2.3], where a slightly stronger proposition is proved.

Proposition 2.3.5. *Consider a Frege system $\mathcal{F}$, a substitution σ and formulas $\psi_1, ..., \psi_k, \varphi$. If $\psi_1, ..., \psi_k \vdash^d_{\mathcal{F}} \varphi$ for some derivation sequence d, then $\sigma\,\psi_1, ..., \sigma\,\psi_k \vdash^{\sigma(d)}_{\mathcal{F}} \sigma\,\varphi$, where $\sigma(d)$ is the sequence composed of the application of σ to each of the formulas in d.* ♢

Proof. This is done by complete induction on the length of d. Suppose we have $|d| = n$.

- $n = 1$: The derivation is composed of a single line, i.e., $d = \{\varphi\}$ (the last line of a derivation has to be the formula being proved) and $\sigma(d) = \{\sigma\,\varphi\}$. Since d is a derivation one line long, this means one of two things:
  - Either φ is an axiom: Then, $\vdash^d_{\mathcal{F}} \varphi$ and $\sigma\,\varphi$ is also an axiom since, by the definition of Frege rule, any substitution instance of an axiom is an axiom. Hence, $\vdash^{\sigma(d)}_{\mathcal{F}} \sigma\,\varphi$, which implies that $\sigma\,\psi_1, ..., \sigma\,\psi_k \vdash^{\sigma(d)}_{\mathcal{F}} \sigma\,\varphi$.
  - Or $\varphi = \psi_i$ for some $i = 1, ..., k$: Then, $\psi_i \vdash^d_{\mathcal{F}} \varphi$ and $\sigma\,\varphi = \sigma\,\psi_i$. Hence, $\sigma\,\psi_i \vdash^{\sigma(d)}_{\mathcal{F}} \sigma\,\varphi$, which implies that $\sigma\,\psi_1, ..., \sigma\,\psi_k \vdash^{\sigma(d)}_{\mathcal{F}} \sigma\,\varphi$.
- $n > 1$: The derivation is composed of n lines, i.e., $d = \{\varphi_1, ..., \varphi_{n-1}, \varphi\}$ and $\sigma(d) = \{\sigma\,\varphi_1, ..., \sigma\,\varphi_{n-1}, \sigma\,\varphi\}$. Now, consider the shorter derivation sequence $d' = \{\varphi_1, ..., \varphi_{n-1}\}$. It's obvious that $\psi_1, ..., \psi_k \vdash^{d'}_{\mathcal{F}} \varphi_{n-1}$ so, by the induction hypotheses, $\sigma\,\psi_1, ..., \sigma\,\psi_k \vdash^{\sigma(d')}_{\mathcal{F}} \sigma\,\varphi_{n-1}$, where $\sigma(d') = \{\sigma\,\varphi_1, ..., \sigma\,\varphi_{n-1}\}$.

  Since $\psi_1, ..., \psi_k \vdash^d_{\mathcal{F}} \varphi$, we know that there is a Frege rule $R = \frac{\phi_1, ..., \phi_m}{\phi}$ in $\mathcal{F}$, with $m \leq n - 1$, such that φ follows from earlier steps in derivation d by R and some substitution σ', i.e., $\sigma'\,\phi = \varphi$ and $\sigma'\,\phi_j = \varphi_{i_j}$ for $j = 1, ..., m$ and $\{i_1, ..., i_m\} \subseteq$

$\{1, ..., n-1\}$. As such, we can also say that $\sigma\,\varphi$ follows from earlier $\sigma\,\varphi_{i_j}$'s by R and substitution $\sigma \circ \sigma'$. In fact, $\sigma \circ \sigma'\,\phi = \sigma\,\varphi$ and $\sigma \circ \sigma'\,\phi_j = \sigma\,\varphi_{i_j}$ for $j = 1, ..., m$, as we wanted. Hence, adding $\sigma\,\varphi$ to derivation sequence $\sigma(d')$ creates derivation $\sigma(d)$, which witnesses $\sigma\,\psi_1, ..., \sigma\,\psi_k \vdash_{\mathcal{F}} \sigma\,\varphi$.

□

Theorem 2.3.6. *For any two Frege systems $\mathcal{F}_1$ and $\mathcal{F}_2$, there is a function g and a constant c such that for all formulas φ and set of formulas S, if $S \vdash^{d}_{\mathcal{F}_1} \varphi$, then $S \vdash^{g(d)}_{\mathcal{F}_2} \varphi$ and $|g(d)| \leq c\,|d|$, where c depends only on $\mathcal{F}_1$.* ♢

Proof. Assume $\mathcal{F}_1$ and $\mathcal{F}_2$ are Frege systems. For each rule $R = \frac{\psi_1,...,\psi_m}{\psi}$ in $\mathcal{F}_1$, let d_R be a derivation of ψ from $\psi_1, ..., \psi_m$ in $\mathcal{F}_2$. Furthermore, suppose d is a derivation of φ from S in $\mathcal{F}_1$ and $d = (\phi_1, ..., \phi_k)$, for some formulas $\phi_1, ..., \phi_k$. To construct the $\mathcal{F}_2$-derivation $g(d)$ from d, if ϕ_i follows from earlier ϕ_j's by the $\mathcal{F}_1$-rule R_i and substitution σ_i, we replace ϕ_i by the derivation $\sigma_i(d_{R_i})$ (with hypotheses deleted). According to Proposition 2.3.5, $\sigma_i(d_{R_i})$ is a derivation of ϕ_i from the same earlier ϕ_j's. Finally, it is clear that $|g(d)| \leq c\,|d|$, where c is the maximum of $|d_R|$, as R ranges over the finite set of rules of $\mathcal{F}_1$. □

From now on, we consider the class of deductive systems $\{\mathcal{F}\colon \mathcal{F}$ is a Frege system$\}$ and restrict our definition of proof complexity to them.

Definition 2.3.7. Let φ be a formula, S a set of formulas and $\mathcal{F}$ a Frege system. The *complexity $CL_{\mathcal{F}}$ of φ conditional to S* is defined by

$$CL_{\mathcal{F}}(\varphi|S) = \min\{|d| :\ S \vdash^{d}_{\mathcal{F}} \varphi, d \in L^{+}\}$$

and $CL_{\mathcal{F}}(\varphi|S) = \infty$ if there is no such d. ∇

As such, $CL_{\mathcal{F}}(\varphi|S)$ is the size of the smallest derivation for φ in a Frege system $\mathcal{F}$, using the elements of S as hypotheses.

As we noted in Section 1.4, a derivation can be created from an infinite set of hypotheses so note that for this complexity, the set S can be infinite, even if only a finite number if its formulas actually appear in each derivation.

Theorem 2.3.8. *Let $\mathcal{F}_1, \mathcal{F}_2$ be any two Frege systems, φ a formula and S a set of formulas. Then, there are constants c_1 and c_2, dependent only on $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively, such that*

$$c_1\,CL_{\mathcal{F}_2}(\varphi|S) \leq CL_{\mathcal{F}_1}(\varphi|S) \leq c_2\,CL_{\mathcal{F}_2}(\varphi|S).\ ♢$$

Proof. We shall prove that $CL_{\mathcal{F}_1}(\varphi|S) \leq c_2\, CL_{\mathcal{F}_2}(\varphi|S)$. The other inequality is proven in a similar fashion.

Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two Frege systems and d a derivation sequence that satisfies $|d| = CL_{\mathcal{F}_2}(\varphi|S)$ and $S \vdash^{d}_{\mathcal{F}_2} \varphi$. It is a smallest derivation of φ using hypotheses S. By Theorem 2.3.6, there is a $g \in \mathrm{PF}$ and $c \in \mathbb{R}^+$ such that $g(d)$ witnesses $S \vdash_{\mathcal{F}_1} \varphi$ and $|g(d)| \leq c\,|d|$. Therefore, $|g(d)| \leq c\, CL_{\mathcal{F}_2}(\varphi|S)$.

However, since $S \vdash^{g(d)}_{\mathcal{F}_1} \varphi$, it follows from the definition of $CL_{\mathcal{F}_1}(\varphi|S)$ that $|g(d)| \geq CL_{\mathcal{F}_1}(\varphi|S)$. Putting the two inequalities together, we arrive at the wanted result, that $CL_{\mathcal{F}_1}(\varphi|S) \leq c\, CL_{\mathcal{F}_2}(\varphi|S)$. □

Using this theorem, we can now stop talking about the complexity of a formula with regard to a specific Frege system and move on to define it as using a fixed Frege system, for instance the one from Example 2.3.4, therefore simplifying our notation.

Definition 2.3.9. Fix a Frege system $\mathcal{F}_0$, which we shall call the *reference Frege system for CL*, and dispense with the subscript by defining the *conditional Kolmogorov proof complexity* $CL(\cdot|\cdot)$ as

$$CL(\varphi|S) = CL_{\mathcal{F}_0}(\varphi|S), \text{ for all } \varphi \text{ and } S.$$

The *unconditional Kolmogorov proof complexity* $CL(\cdot)$ is defined as $CL(\varphi) = CL(\varphi|\{\})$ for all formulas φ, where $\{\}$ represents having no hypotheses. ∇

When S is a singleton ($S = \{\psi\}$), we abbreviate $CL(\varphi|\{\psi\})$ to $CL(\varphi|\psi)$. We will also from now on write $\vdash$ instead of $\vdash_{\mathcal{F}_0}$.

It should be noted that while proof complexity has been thoroughly studied before, it just hadn't been defined in this manner or studied through this perspective before, as far as we know. In [1], the concept of proof complexity is described but no significant results for our purposes can be found. In [2], some theorems relevant to our purposes are established. We adapt them to our terminology later on, as Theorems 2.3.16 to 2.3.18. Some research on proof complexity for certain specific classes of tautologies has also been done (for instance in [9]) and in [3] lower bounds for the length of proofs are discussed.

Results

First, we present some simple results that relate the Kolmogorov proof complexity of different formulas.

Proposition 2.3.10. *Let φ and ψ be any two formulas and Γ a set of formulas. Then:*

1. $CL(\psi) \leq CL(\varphi) + CL(\psi|\varphi)$;

2. $CL(\psi) \leq CL(\varphi) + CL(\varphi \to \psi) + 1$;

3. $CL(\psi|\varphi) \leq CL(\varphi \to \psi) + 2$;

4. $CL(\psi|\Gamma) \leq CL(\psi)$. $\diamondsuit$

Proof. For all these proofs, consider $CL(\varphi) = |d_1|$ with $\vdash^{d_1} \varphi$, $CL(\psi) = |d_2|$ with $\vdash^{d_2} \psi$, $CL(\psi|\varphi) = |d_3|$ with $\varphi \vdash^{d_3} \psi$ and $CL(\varphi \to \psi) = |d_4|$ with $\vdash^{d_4} \varphi \to \psi$ for some derivations $d_1, ..., d_4$.

1. Since d_1 is a derivation for φ and d_3 is a derivation for ψ given φ, then putting the two together in a new derivation d composed of the elements of d_1 followed by the elements d_3 (except the line where φ is assumed) gives us a derivation for ψ. Then, $|d| = |d_1| + |d_3| - 1 \leq |d_1| + |d_3|$ and obviously, $|d| \geq |d_2|$ since d_2 is the shortest derivation for ψ.

2. Considering that d_1 is a derivation for φ and d_4 is a derivation for $\varphi \to \psi$, putting the two together in a new derivation d composed of the elements of d_1 followed by the elements d_4 and then a new line with just ψ (which we can get by applying Modus Ponens) gives us a derivation for ψ. Then, $|d| = |d_1| + |d_4| + 1$ and obviously, $|d| \geq |d_2|$ since d_2 is the shortest derivation for ψ.

3. Since d_4 is a derivation for $\varphi \to \psi$ and we want to prove ψ given φ, we can create a new derivation d that has the elements of d_4 followed by a line with φ (assumption) and another with ψ (derived by Modus Ponens). This gives a us a derivation sequence for ψ given φ. Then, $|d| = |d_4| + 2$ and obviously, $|d| \geq |d_3|$ since d_3 is the shortest derivation for ψ given φ.

4. Any derivation for a formula given no hypotheses is also a derivation for the same formula given any set of hypotheses. As such, d_2 also witnesses $\Gamma \vdash \psi$ and $|d_2| \geq CL(\psi|\Gamma)$.

□

We now present some new definitions that will allow us to arrive at an upper bound for the Kolmogorov proof complexity of a given formula, the idea of which came from [12]. Note that the definition below will give us a new kind complexity for formulas in L. However, we only introduce it to arrive at the aforementioned upper bound and will not explore what else can be proven with this new complexity.

Definition 2.3.11. Let d be a derivation. We define $\rho(d)$ as the number of distinct subformulas in d.

Let φ be a formula in L and $\mathcal{F}$ a Frege system. Define $\tau_{\mathcal{F}}(\varphi) = \min\{\rho(d) : \vdash^d_{\mathcal{F}} \varphi\}$ and $\tau_{\mathcal{F}}(\varphi) = \infty$ if there is no such d. ∇

Proposition 2.3.12. *For any two Frege systems $\mathcal{F}_1$ and $\mathcal{F}_2$, there is a function g and a constant c such that for all formulas φ, if $\vdash^d_{\mathcal{F}_1} \varphi$, then $\vdash^{g(d)}_{\mathcal{F}_2} \varphi$ and $\rho(g(d)) \leq c\,\rho(d)$, where c depends only on $\mathcal{F}_1$.* $\diamondsuit$

Proof. By Theorem 2.3.6, there exists such a function g. If we look at the way $g(d)$ is constructed in the proof of this theorem, we realize that the number of subformulas it has is at most c times the number of subformulas d has, where c is the maximum of $\rho(d_R)$, as R ranges over the finite set of rules of $\mathcal{F}_1$. □

Corollary 2.3.13. *Let $\mathcal{F}_1, \mathcal{F}_2$ be any two Frege systems and φ a formula. Then, there are constants c_1 and c_2, dependent only on $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively, such that*

$$c_1\,\tau_{\mathcal{F}_2}(\varphi) \leq \tau_{\mathcal{F}_1}(\varphi) \leq c_2\,\tau_{\mathcal{F}_2}(\varphi).\ \diamondsuit$$

Proof. We will prove that $\tau_{\mathcal{F}_1}(\varphi) \leq c_2\,\tau_{\mathcal{F}_2}(\varphi)$. The other inequality is proven the same way.

Consider two Frege systems $\mathcal{F}_1$ and $\mathcal{F}_2$ and a derivation sequence d that satisfies $\rho(d) = \tau_{\mathcal{F}_2}(\varphi)$ and $\vdash^d_{\mathcal{F}_2} \varphi$. By Corollary 2.3.12, there is a function g and constant c such that $g(d)$ witnesses $\vdash_{\mathcal{F}_1} \varphi$ and $\rho(g(d)) \leq c\,\rho(d)$. Therefore, $\rho(g(d)) \leq c\,\tau_{\mathcal{F}_2}(\varphi)$. Since $g(d)$ is a derivation satisfying $\vdash^{g(d)}_{\mathcal{F}_1} \varphi$, it follows from the definition of $\tau_{\mathcal{F}_1}(\varphi)$ that $\rho(g(d)) \geq \tau_{\mathcal{F}_1}(\varphi)$. Putting the two inequalities together, we get $\tau_{\mathcal{F}_1}(\varphi) \leq c\,\tau_{\mathcal{F}_2}(\varphi)$. □

As such, we can proceed as with CL and fix one Frege system. This allows us to make the notation simpler and arrive at the upper bound we were looking for.

Definition 2.3.14. Let $\mathcal{F}_0$ be the reference Frege system for CL. For any formula φ in L, define $\tau(\varphi) = \tau_{\mathcal{F}_0}(\varphi)$. ∇

Proposition 2.3.15. *For every formula φ in L, we have that $CL(\varphi) \leq \tau(\varphi)$.* $\diamondsuit$

Proof. For any derivation d' such that $\vdash^{d'} \varphi$, note that there is at least one new subformula for each line in d'. As such, $\rho(d') \geq |d'|$. Let d be a derivation sequence such that $|d| = CL(\varphi)$ and $\vdash^d \varphi$. It follows from this that $|d| \leq |d'| \leq \rho(d')$. Hence, $|d|$ is smaller or equal to the number of subformulas in any derivation that witnesses $\vdash \varphi$, which means

that it is smaller or equal to the minimum number of subformulas in any derivation that witnesses $\vdash \varphi$, i.e., $|d| = CL(\varphi) \leq \tau(\varphi)$. □

The next three results and proofs are based on [2, Theorems 1 to 3]. Even though they are not translatable to the other complexities we are studying, we felt it was important to show them because they are the adaptation of previously demonstrated theorems to this new context of Kolmogorov proof complexity.

Theorem 2.3.16. *(Deduction theorem)* $CL(\psi \to \varphi) \leq c\, CL(\varphi|\psi)$*, for some constant* c*.*

Proof. Suppose we have a derivation $d = (\phi_1, ..., \phi_n)$ such that $CL(\varphi|\psi) = n$ and $\psi \vdash^d \varphi$. From this derivation, we can create another d' such that $\vdash^{d'} \psi \to \varphi$.

For this, we just need to substitute each ϕ_i with $\psi \to \phi_i$ and fill in the gaps according to how ϕ_i was derived:

- ϕ_i is ψ: we can create a derivation for $\psi \to \psi$ with a constant number of lines;
- ϕ_i is an axiom: a derivation for $\psi \to \phi_i$ can be created with a constant number of lines;
- ϕ_i is derived by Modus Ponens from ρ and $\rho \to \phi_i$: we can create a derivation for $\psi \to \phi_i$ from $\psi \to \rho$ and $\psi \to (\rho \to \phi_i)$ with a constant number of lines.

This allows us to arrive at a proof d' such that $|d'| = c\,|d|$ and since $|d'| \geq CL(\psi \to \varphi)$, we get the intended result. □

By adapting the proof of Theorem 2.3.16 for m hypotheses, we get that: If $\psi_1, ..., \psi_m \vdash \varphi$, then $CL(\psi_1 \to (\psi_2 \to ... \to (\psi_m \to \varphi)...)) \leq c^m CL(\varphi|\{\psi_1, ..., \psi_m\})$. However, we can substantially improve this bound, as shown below.

Theorem 2.3.17. *(Simple deduction theorem)* $CL(\psi_1 \to (\psi_2 \to ... \to (\psi_m \to \varphi)...)) \leq c'(CL(\varphi|\{\psi_1, ..., \psi_m\}) + m)$*, for some constant* c'*.*

Proof. Given an n line derivation d of φ using the set of assumptions $\{\psi_1, ..., \psi_m\}$ such that $n = CL(\varphi|\{\psi_1, ..., \psi_m\})$, we construct a derivation d' of φ from the single assumption $\psi_1 \wedge ... \wedge \psi_m$ (where the conjunction is to be associated from left-to-right). For any Frege proof system, there is a constant k such that $\phi \wedge \rho \vdash^{d_1} \phi$ and $\phi \wedge \rho \vdash^{d_2} \rho$ and $|d_1| = |d_2| = k$. Thus, d' can be constructed to:

1. first derive each of $\psi_1, ..., \psi_m$ in $2k(m-1)$ lines and
2. then, derive φ in at most n lines (using d).

Hence, we have created a derivation d' such that $\psi_1 \wedge ... \wedge \psi_m \vdash^{d'} \varphi$ and $|d'| \leq 2k(m-1)+n$, i.e., $|d'| = O(m+n)$.

By applying Theorem 2.3.16, there is a derivation d_3 of $\psi_1 \wedge ... \wedge \psi_m \rightarrow \varphi$ with $O(m+n)$ lines.

Finally, it can be shown by induction on m that there is a derivation d_4 of $(\psi_1 \wedge ... \wedge \psi_m \rightarrow \varphi) \rightarrow (\psi_1 \rightarrow (\psi_2 \rightarrow ... \rightarrow (\psi_m \rightarrow \varphi)...))$ with $|d_4| = O(m)$. As such, by combining these two derivations and applying Modus Ponens, we get what we were after: a derivation d_5 witnessing $\vdash^{d_5} \psi_1 \rightarrow (\psi_2 \rightarrow ... \rightarrow (\psi_m \rightarrow \varphi)...)$. For this derivation, $|d_5| = |d_3| + |d_4| + 1 = O(m+n)$. □

We show one last result from [2], a generalization of the previous one.

Corollary 2.3.18. *Let φ be any conjunction of $\psi_1, ..., \psi_m$ in that order but associated arbitrarily. Let $i_1, ..., i_n$ be any sequence from $\{1, ..., m\}$ and let ϕ be any conjunction of $\psi_{i_1}, ..., \psi_{i_n}$ again in the indicated order and associated arbitrarily. Then, $CL(\varphi \rightarrow \phi) \leq O(n+m)$.*

Proof. By Theorem 2.3.16, it suffices to show that $CL(\phi|\varphi) \leq O(n+m)$. A derivation sequence for ϕ from φ proceeds as follows:

1. from the assumption φ deduce each subformula of φ and, in particular, each of the formulas $\psi_1, ..., \psi_m$ and

2. then, deduce each subformula of ϕ from the smallest to the largest.

Since there is a constant k such that $\omega \wedge \rho \vdash^{d_1} \omega$, $\omega \wedge \rho \vdash^{d_2} \rho$ and $\{\omega, \rho\} \vdash^{d_3} \omega \wedge \rho$ and $|d_1| = |d_2| = |d_3| = k$, for all formulas ρ and ω , it is clear that this new derivation contains $O(m+n)$ lines. □

Lastly, we prove a result for this kind of complexity that is an adaptation of the incompressibility theorem (Theorem 2.1.10). However, we are not able to prove it for a language as general as L, as we have done for the other results so far. We will have to consider a setting where the number of possible formulas is finite. Also, let us clarify that by $\log_N$ we mean the base N logarithm.

Theorem 2.3.19. *Consider a language where there are at most N possible formulas and let c be a positive integer. For each fixed set of formulas S, every finite set B of cardinality m has at least $\frac{m(N-N^{-c}-1)+N}{N-1}$ formulas φ with $CL(\varphi|S) \geq log_N(m) - c$.* ♢

Proof. The number of derivations of length less than $\log_N(m) - c$ is $\sum_{i=1}^{\log_N(m)-c-1} N^i = \frac{N\left(N^{\log_{N(m)-c-1}-1}\right)}{N-1} = \frac{N^{-c}m-N}{N-1}$. As such, there are at least $m - \frac{N^{-c}m-N}{N-1} = \frac{m(N-N^{-c}-1)+N}{N-1}$ formulas in B that have no derivation of length less than $\log_N(m) - c$. □

Even though B is a finite set of formulas, it is not enough to arrive at the result we want because a derivation for a formula in B can be composed of formulas not in B and what we need here is a limited amount of possibilities to construct such a derivation.

Keeping in line with the previous sections, now we would study CL as an integer function or, better yet, study $CL \circ i_T^{-1}$ as an integer function, where $i_T^{-1}(n)$ is the n^{th} element of a enumeration of L, the formulas we are working with. However, we were unable to determine whether $CL \circ i_T^{-1}$ is computable or bounded without adding some constraints to our language L (similarly to the theorem above). We will expand on this in the next chapter.

Chapter 3

Kolmogorov Generic Bases

3.1 Definition and Examples

After carefully studying the four types of complexity found in the previous chapter, we noticed there are some quite evident similarities between them.

For instance, for all four complexities, which specific additively optimal partial recursive function/additively optimal universal Turing machine/additively optimal partial recursive prefix function/Frege system is being used only affects the value of the complexity by an additive or multiplicative constant. Also, they all have an upper bound for the unconditional version and the conditional complexity has an upper bound which is a function of the unconditional complexity.

As such, we developed the concept of Kolmogorov Generic Basis, a structure designed to embody all four types of complexity we studied but that is not limited to these.

Definition 3.1.1. A *Kolmogorov generic basis* is a tuple (A, D, M, F, v) such that:

- A and D are two non-empty recursively enumerable sets[1];
- $M : D \to \mathbb{N}$ is a computable function[2];
- F is a non-empty subset of the functions from D to A that can be simulated by deterministic Turing machines[3];
- $v : A \times 2^A \times F \times D \to \{0, 1\}$ is a function[4].

[1] A is the set of objects we want to describe and D is the set of objects we use to describe the elements of A.

[2] M is the function of D we want to minimize.

[3] F is the set of description methods we use to interpret each element of D and arrive at an element of A.

[4] v is the function that tells us whether the element of D in question effectively describes the inputted element of A.

This tuple induces another function $g : A \times 2^A \times F \to \mathbb{N} \cup \{\infty\}$ such that

$$g(a, S, f) = \min\{M(d) : d \in D, v(a, S, f, d) = 1\},$$

also written as $g_f(a|S)$. If no such d exists ($v(a, S, f, d) = 0$ for all $d \in D$), then we define $g_f(a|S)$ as ∞. Also, we will abbreviate $g_f(a|\{\})$ as $g_f(a)$.

A Kolmogorov generic basis must also have the following properties:

1. There exists an $f_1 \in F$ such that for each $f_2 \in F$, there exist constants c_1 and c_2 (dependent only on f_2) such that for all finite $S \in 2^A$ and $a \in A$, $g_{f_1}(a|S) \leq c_1\, g_{f_2}(a|S) + c_2$. Therefore, we can fix one such $f_1 \in F$ and abbreviate $g_{f_1}(a|S)$ as $g(a|S)$ and $g_{f_1}(a)$ as $g(a)$, keeping in mind that we are always talking about the same f_1.[5]

2. There exist constants c_1 and c_2 and a function $h : A \to \mathbb{N}$ such that for all $a \in A$, $g(a) \leq c_1\, h(a) + c_2$.[6]

3. There exist constants c_1 and c_2 such that for all $a \in A$ and finite $S \in 2^A$, $g(a|S) \leq c_1 g(a) + c_2$.[7]

We call $g(a|S)$ the *conditional Kolmogorov generic function for basis* (A, D, M, F, v) and $g(a)$ the *unconditional* version. ∇

Looking at this definition, we can see that a Kolmogorov generic function g returns the smallest possible value for an element of D over M satisfying verification function v. This is how C^F, C^M, K and CL work.

We will now see what instances of Kolmogorov generic basis induce the complexities we studied earlier and justify some of our choices for this definition along the way.

Example 3.1.2. To get the Kolmogorov complexity based on partial recursive functions (defined in Subsection 2.1.1) we have to consider:

- A and D as $\mathbb{N}$;
- F as the set of all partial recursive functions from $\mathbb{N}$ to $\mathbb{N}$;
- $M(d) = l(d)$, the length of the string corresponding to natural number d;

[5] This properties ensures that the particular specification method f we are using only affects the value of $g_f(a|S)$ by additive and multiplicative constants.

[6] This properties ensures that the unconditional Kolmogorov generic function has a function of its input as an upper bound.

[7] This properties ensures that the conditional Kolmogorov generic function has the unconditional version as an upper bound.

- $v(a,S,f,d) = \begin{cases} 1, & S \text{ is finite and } f([S,d]) = a \\ 1, & S \text{ is infinite and } f([S',d]) = a \text{ for some finite } S' \subset S\,, \\ 0, & \text{otherwise} \end{cases}$

  where $[\cdot,\cdot]$ is the pairing defined in Subsection 2.1.1. □

Even though the Kolmogorov complexities C^F, C^M and K are not defined for an infinite set S, this case is taken into account because, as we noted in Section 2.3, for Kolmogorov proof complexity CL, S can be infinite. As a matter of fact, in propositional logic, we can have an infinite set of hypotheses, even though we can actually only use a finite set in a derivation.

Also, recall that we are simplifying the notation by writing $\phi([S,p])$ when p and S are, respectively, a natural number and a set of natural numbers. What we actually mean is that p is converted to its corresponding string and the elements of S as well and then the function $[\cdot,\cdot]$ is applied to these. The result of this function is then converted back to a natural number and applied to ϕ.

Proposition 3.1.3. *Using the Kolmogorov generic basis defined in Example 3.1.2,*

$$g_f(a|S) = \begin{cases} C_f^F(a|S), & S \textit{ is finite} \\ min\{C_f^F(a|S') : S' \subset S \textit{ is finite}\}, & S \textit{ is infinite} \end{cases}$$

and $g_f(a) = C_f^F(a)$. ♢

Proof. For the basis from Example 3.1.2, $g_f(a|S) = \min\{l(d) : d \in \mathbb{N},\, v(a,S,f,d) = 1\}$. We should first observe that A and D are non-empty recursively enumerable sets and F is the set of partial recursive functions from $\mathbb{N}$ to $\mathbb{N}$. Furthermore, $l(d)$ is clearly a computable function, since it just needs to find the string corresponding to d and see how many bits it has.

Let's consider a finite S. Then, $v(a,S,f,d) = 1$ if and only if $f([S,d]) = a$ and so $g_f(a|S) = \min\{l(d) : d \in D,\, f([S,d]) = a\} = C_f^F(a|S)$.

Now, let's consider the more complicated case of an infinite S. For this case, $g_f(a|S) = \min\{l(d) : d \in D,\, f([S',d]) = a \text{ for some finite } S' \subset S\}$. This is to say $g_f(a|S)$ is the smallest of the possible values of $l(d)$, for each $S' \subset S$ finite. In other words, $g_f(a|S) = \min\{C_f^F(a|S') : S' \subset S \text{ is finite}\}$.

Observe that if we came across an $S' \subset S$ such that no $d \in D$ satisfies $f([S',d]) = a$, then $C_f^F(a|S') = \infty$ so those cases would become irrelevant when considering the minimum (unless $C_f^F(a|S') = \infty$ for all S', in which case $g_f(a|S) = \infty$, as it should be).

It is obvious that $g_f(a) = C_f^F(a)$ since $g_f(a) = g_f(a|\{\})$ is a particular case of S being finite. Also, regarding the properties in Definition 3.1.1:

- property 1 is a consequence of the invariance theorem (Theorem 2.1.7), with $c_1 = 1$;
- property 2 is equivalent to the first statement of Theorem 2.1.9, with $h(a) = l(a)$ and $c_1 = 1$;
- property 3 translates into the second statement of Theorem 2.1.9, considering $c_1 = 1$.

□

It should be noted that for property 1 of the definition of Kolmogorov generic basis, we could had written a stronger condition, one that would be equivalent to the invariance theorem: There exists an $F' \subseteq F$ such that for all $f_1 \in F'$, $f_2 \in F$, there exist positive constants c_1 and c_2 such that for all $S \in 2^A$ and $a \in A$, $g_{f_1}(a|S) \leq c_1\, g_{f_2}(a|S) + c_2$.

However, this condition would had forced us to define the set F', the subset of F formed of the elements that have that property (for instance, it would be the set of additively optimal prefix partial recursive functions, in the case of prefix Kolmogorov complexity). This seemed unnecessary since the main purpose of that condition is to say that we can fix an f and always use it as a reference for g, something that we can also do with the weaker condition we decided to use.

Example 3.1.4. To arrive at the Kolmogorov complexity based on Turing machines (Subsection 2.1.2), we have to consider:

- A and D equal to $\{0,1\}^*$;
- F equal to the set of deterministic Turing Machines over $\{0,1\}^*$;
- $M(d) = l(d)$, the length of string d;
- $v(a, S, f, d) = \begin{cases} 1, & S \text{ is finite and } f([S,d]) = a \\ 1, & S \text{ is infinite and } f([S',d]) = a \text{ for some finite } S' \subset S, \\ 0, & \text{otherwise} \end{cases}$

  where, as before, $[\cdot,\cdot]$ is the function defined in Subsection 2.1.1. □

Proposition 3.1.5. *Under the conditions above,*

$$g_f(a|S) = \begin{cases} C_f^M(a|S), & S \textit{ is finite} \\ min\{C_f^M(a|S') : S' \subset S \textit{ is finite}\}, & S \textit{ is infinite} \end{cases}$$

and $g_f(a) = C_f^M(a)$. $\diamondsuit$

Proof. This proof is very similar to the previous one, we need only to adjust a few details to this new setting. For this basis, $g_f(a|S) = \min\{l(d) : d \in \{0,1\}^*, v(a,S,f,d) = 1\}$. Also, observe that A and D are obviously non-empty recursively enumerable sets and F is the set of deterministic Turing machines from $\{0,1\}^*$ to $\{0,1\}^*$. Furthermore, $l(d)$ is computable, since it just needs to return how many bits the string d has.

Let's consider a finite S. Then, $v(a,S,f,d) = 1$ if and only if $f([S,d]) = a$. As such, $g_f(a|S) = \min\{l(d) : d \in D,\ f([S,d]) = a\} = C_f^M(a|S)$.

Now, let's consider an infinite S. This means $g_f(a|S) = \min\{l(d) : d \in D,\ f([S',d]) = a$ for some finite $S' \subset S\}$. This means $g_f(a|S)$ is the smallest of the possible values of $l(d)$, for each $S' \subset S$ finite. In other words, $g_f(a|S) = \min\{C_f^M(a|S') : S' \subset S \text{ is finite}\}$.

It is obvious that $g_f(a) = C_f^M(a)$ since $g_f(a) = g_f(a|\{\})$ is a particular case of S being finite. Also, regarding the properties in Definition 3.1.1:

- property 1 is a consequence of Theorem 2.1.19, with $c_1 = 1$;
- property 2 is equivalent to the first statement of Theorem 2.1.21, with $h(a) = l(a)$ and $c_1 = 1$;
- property 3 translates into the second statement of Theorem 2.1.21, considering $c_1 = 1$.

□

Observe that in Definition 3.1.1, for properties 1 and 3, we only consider finite sets S because otherwise, for C^F, C^M and K we would not be able to verify these conditions. In fact, suppose we try to prove that property 1 holds for C^F considering an infinite S, i.e., that there exists an $f_1 \in F$ such that for each $f_2 \in F$, there exist constants c_1 and c_2 such that for all infinite $S \in 2^A$ and $a \in A$, $C_{f_1}^F(a|S) \leq c_1 C_{f_2}^F(a|S) + c_2$. Defining S' as a finite set verifying $C_{f_1}^F(a|S') = \min\{C_{f_1}^F(a|S') : S' \subset S \text{ is finite}\}$ and S'' as an infinite set verifying $C_{f_2}^F(a|S'') = \min\{C_{f_2}^F(a|S') : S' \subset S \text{ is finite}\}$, the inequality above holds only if $C_{f_1}^F(a|S') \leq c_1 C_{f_2}^F(a|S'') + c_2$. However, the invariance theorem is only applicable here if $S' = S''$ and we cannot guarantee this. We would be in a similar situation if we tried to prove property 3 for an infinite S.

Example 3.1.6. To get the prefix Kolmogorov complexity (Section 2.2) we have to consider:

- A and D as $\mathbb{N}$;

- F as the set of all partial recursive prefix functions from $\mathbb{N}$ to $\mathbb{N}$;
- $M(d) = l(d)$, the length of the string corresponding to natural number d;
- $v(a, S, f, d) = \begin{cases} 1, & S \text{ is finite and } f([S, d]) = a \\ 1, & S \text{ is infinite and } f([S', d]) = a \text{ for some finite } S' \subset S, \\ 0, & \text{otherwise} \end{cases}$

  where once more $[\cdot, \cdot]$ is as defined in Subsection 2.1.1. □

Proposition 3.1.7. *Taking into account the Kolmogorov generic basis defined above,*

$$g_f(a|S) = \begin{cases} K_f(a|S), & S \textit{ is finite} \\ min\{K_f(a|S') : S' \subset S \textit{ is finite}\}, & S \textit{ is infinite} \end{cases}$$

and $g_f(a) = K_f(a)$. ♢

Proof. The first part of this proof is the same as that of Proposition 3.1.3 (changing C^F for K and considering this new F, whose elements are restrictions of partial recursive functions) so we will omit it and skip ahead to the verification of the properties from Definition 3.1.1:

- property 1 is a consequence of Theorem 2.2.3, with $c_1 = 1$;
- property 2 comes from the first statement of Proposition 2.2.8, with $h(a) = l(a)$ and $c_1 = 3$;
- property 3 is a consequence of the second statement of Proposition 2.2.8, considering $c_1 = 3$.

□

We should point out that properties 2 and 3 could had been tailored to translate exactly into Proposition 2.2.8. However, then the counter domain of h would have to be the real numbers. So, using the fact that $\log(n) \leq n$ for all natural numbers, we decided to impose a condition which is a little weaker than the aforementioned proposition but keeps the structure of the basis a little simpler, dismissing the unwanted log term.

Example 3.1.8. To arrive at the Kolmogorov proof complexity (studied in Section 2.3), we use the following basis:

- A is the set of propositional formulas over a set of propositional variables Ξ;
- $D = A^+$ is the set of all possible derivations over A;

- F is the set of all propositional proof systems induced for A by Frege systems;
- $M(d) = |d|$, the length of derivation sequence d;
- $v(a, S, f, d) = \begin{cases} 1, & S \text{ is finite and } f(\langle S, d\rangle) = a \\ 1, & S \text{ is infinite and } f(\langle S', d\rangle) = a \text{ for some finite } S' \subset S\,, \\ 0, & \text{otherwise} \end{cases}$

  where $\langle \cdot, \cdot \rangle$ is as defined in Section 1.4. □

Proposition 3.1.9. *Using the basis from the previous example, we get* $g_f(a|S) = CL_f(a|S)$ *and* $g_f(a) = CL_f(a)$. ♢

Proof. Note that, according to Notation 1.4.9, we can write the condition for $v(a, S, f, d) = 1$ simply as $S \vdash_f^d a$. As such, we get that $g_f(a|S) = min\{|d| : d \in A^+, S \vdash_f^d a\} = CL_{\mathcal{F}}(a|S)$, where $\mathcal{F}$ is the Frege system that induced f. Also, $g_f(a) = g_f(a|\{\}) = CL_{\mathcal{F}}(a|\{\}) = CL_{\mathcal{F}}(a)$.

Furthermore, A and D are non-empty recursively enumerable sets and F is a non-empty subset of the set of partial recursive functions from A to D since every propositional proof system induced by a Frege system is a partial recursive function. In fact, since a Frege system only has a finite set of rules, we can construct a deterministic Turing machine that computes the propositional proof system induced by it. Also, note that $|d|$ is a computable function, since it just needs to see how many lines derivation d has.

As such, we only have left to remark that, regarding the properties in Definition 3.1.1:

- property 1 is a consequence of Theorem 2.3.8, using $c_2 = 0$;
- property 2 translates into Proposition 2.3.15, with $h(a) = \tau(a)$ and $c_1 = 1$ and $c_2 = 0$;
- property 3 is equivalent to the forth statement from Proposition 2.3.10, with $c_1 = 1$ and $c_2 = 0$.

□

Above, we present an upper bound for the unconditional version of Kolmogorov proof complexity (Proposition 2.3.15) with a property that is not ideal: it depends on the reference Frege system. Although we proved that it varies only by a multiplicative factor with the Frege system being used (Corollary 2.3.13), this is not an ideal situation since, for the other examples of Kolmogorov generic bases we looked at, the upper bound only depends on the object whose complexity we are calculating.

	$\mathbf{C_f^F(a|S)}$	$\mathbf{C_f^M(a|S)}$	$\mathbf{K_f(a|S)}$	$\mathbf{CL_f(a|S)}$
$\mathbf{A}$	$\mathbb{N}$	$\{0,1\}^*$	$\mathbb{N}$	formulas
$\mathbf{D}$	$\mathbb{N}$	$\{0,1\}^*$	$\mathbb{N}$	derivations
$\mathbf{M(d)}$	$l(d)$	$l(d)$	$l(d)$	$\|d\|$
$\mathbf{F}$	partial recursive functions	deterministic Turing machines	partial recursive prefix functions	Frege systems
$\mathbf{v(a,S,f,d)}$	$f([S,d]) = a$	$f([S,d]) = a$	$f([S,d]) = a$	$S \vdash_f^d a$
$\mathbf{h(a)}$	$l(a)$	$l(a)$	$l(a)$	$\tau(a)$

Table 3.1: Examples of Kolmogorov generic bases.

As such, it is a slight abuse of the definition to say $h(a) = \tau(a)$ and $h : A \to \mathbb{N}$. However, we kept the definition of property 2 of a Kolmogorov generic basis as is in part because we believe a better upper bound, truly dependent only on a could probably be found with more time and research. In fact, in [3], Buss mentions that "The best know upper bounds on the size of Frege proofs are exponential...".

One approach might be to find such an upper bound for another type of propositional proof system and then adapt it for Frege systems if the size of proofs in the two systems are related ([10] is a good survey on the relationships between different propositional proof systems).

Table 3.1 summarizes Kolmogorov generic bases that induce the examples in this section. It should be taken into account that the information in the table is not complete, for instance the description of function v in the table is just the condition for it to be 1 and only in the case of S being finite for C^F, C^M and K.

3.2 Results

Having defined this new structure, we now work on proving some results for it. First we look at whether, given a Kolmogorov generic basis, its induced unconditional Kolmogorov generic function is computable or not. Recall that C^F, C^M and K are not computable. However, for this new type of complexity, we were not able to prove such a general result. In fact, we prove that there is a class of Kolmogorov generic basis for which the induced g is computable and one for which it isn't.

For any recursively enumerable set S, let $i_S : S \to \mathbb{N}$ be the function such that $i_S(s)$ is the index of s in a fixed enumeration of S. As such, $i_S^{-1}(n)$ returns the n^{th} element of the enumeration.

Define $\mathcal{C}_1$ as the class of Kolmogorov generic bases (A, D, M, F, v) such that, for any enumeration of A, there exist constants k_1 and k_2 such that, for any a in A, $g(a) \leq$

$k_1 C^F(i_A(a)) + k_2$. Recall that C^F is the Kolmogorov complexity based on partial recursive functions from Subsection 2.1.1.

Theorem 3.2.1. *For any basis in $\mathcal{C}_1$, the induced unconditional Kolmogorov generic function g is not computable. Moreover, no partial recursive function ϕ defined on an infinite set of points can coincide with $g(i_A^{-1}(n))$ over the whole of its domain of definition, no matter what enumeration of A is chosen.* $\diamondsuit$

Proof. This proof was inspired by the proof of the incomputability theorem (Theorem 2.1.13).

For each basis (A, D, M, F, v) in class $\mathcal{C}_1$, suppose there exists a partial recursive function ϕ defined on an infinite set of points that does coincide with the induced $g \circ i_A^{-1}$ over the whole of its domain of definition, for some enumeration of A. Consider an infinite recursive subset of its domain of definition $B \subseteq \mathbb{N}$ (which exists by Lemma 1.2.4) and consider the function $\psi : \mathbb{N} \to B$ such that $\psi(m) = \min\{n \in B : g(i_A^{-1}(n)) \geq m\}$. Observe that this function is a partial recursive function since $g \circ i_A^{-1}$ coincides with ϕ in B.

It is easy to see that, by the definition of ψ, $g(i_A^{-1}(\psi(m))) \geq m$. Also, by the definition of $\mathcal{C}_1$, $g(i_A^{-1}(\psi(m))) \leq k_1\, C^F(\psi(m)) + k_2$ for some constants k_1 and k_2. Since ψ is a partial recursive function, we can apply the invariance theorem (Theorem 2.1.7) and get that $C^F(\psi(m)) \leq C_\psi^F(\psi(m)) + c \leq l(m) + c' = \log(m) + c''$, for some constants c, c' and c''. As such, we get that there exist constants c_1 and c_2 such that for all $m \in \mathbb{N}$, $m \leq c_1 \log(m) + c_2$, which is false from some m onward no matter what constants we use. Hence, our assumption was incorrect and g is not computable. $\square$

Looking at the definition of $\mathcal{C}_1$, it is obvious that the Kolmogorov generic basis that induces C^F is in this class. Furthermore, the fact that the bases for C^M and K are also in $\mathcal{C}_1$ follows from, respectively, Propositions 2.1.16 and 2.2.6, which specify the relation each of these complexities has with C^F.

However, Kolmogorov proof complexity may not to be a part of this class since we cannot specify a relation between CL and C^F. In fact, it seems unlikely that there is a relation between the minimal size of a proof in a Frege system for a certain formula and the minimal size of a natural number that when inputted into an additively optimal partial recursive function outputs the rank of the formula in an enumeration of all possible formulas.

Define $\mathcal{C}_2$ as the class of Kolmogorov generic bases such that:

- for all f in F, f is defined for all inputs;
- for all $a \in A$, $f \in F$ and $d \in D$, $v(a, \{\}, f, d) = 1$ if and only if $f(d) = a$;

- for all $i \in \mathbb{N}$, the set $\{d \in D : M(d) = i\}$ is finite.

Note that in the case that the elements of f are Turing machines, by $f(d) = a$ we mean that Turing machine f outputs a on input d.

Theorem 3.2.2. *Any Kolmogorov generic function g induced by a basis in class $\mathcal{C}_2$ is computable, i.e., $g \circ i_A^{-1}$ is a partial recursive function, no matter what enumeration of A is chosen.* $\diamondsuit$

Proof. Since D is a recursively enumerable set and, for all $i \in \mathbb{N}$, the set $\{d \in D : M(d) = i\}$ is finite, we can define an enumeration of D such that its elements are organized in ascending order of $M(d)$, i.e., first come the elements of D such that $M(d) = 0$, then $M(d) = 1$ and so on. Let i_D be the indexes of the elements of D in one such enumeration.

Also, because $v(a, \{\}, f, d) = 1$ if and only if $f(d) = a$, $g_f(i_A^{-1}(n)) = min\{M(d) : v(i_A^{-1}(n), \{\}, f, d) = 1\} = min\{M(d) : f(d) = i_A^{-1}(n)\}$ and since any f in F is defined for all inputs, we can just test whether for a given n and d $f(d) = i_A^{-1}(n)$ and always have an answer.

Note that, since the elements of F are either partial recursive functions or deterministic Turing machines, each can be uniquely linked to a Turing machine (either the Turing machine that computes it or itself). As such, each f in F is associated to the index in an enumeration of all Turing machines of the one it is linked to. Denote this index by $i_F(f)$.

Now, consider a Turing machine that receives the strings corresponding to an $n \in \mathbb{N}$ and $i_F(f)$ for an $f \in F$ and follows the algorithm below in order to return $g_f(i_A^{-1}(n))$.

Set $j = 0$, $r = 0$;

While $(r == 0)$ **do** {

If {the Turing machine associated with f outputs $i_A^{-1}(n)$ on input $i_D^{-1}(j)$}

then {set $r = 1$}

otherwise {set $j = j + 1$}};

Return $M(i_D^{-1}(j))$.

It's easy to see that this Turing machine computes $g \circ i_A^{-1}$ and, as such, this function is partial recursive. □

Class $\mathcal{C}_2$ doesn't have any of the examples we studied: for C^F, C^M and K the first condition doesn't hold and for CL the second and third conditions don't hold. However, inside $\mathcal{C}_2$ there is a simple Kolmogorov generic basis described below.

Example 3.2.3. An example of a Kolmogorov generic function in $\mathcal{C}_2$ is the one induced by the basis (A, D, M, F, v) where:

- $A = D = \mathbb{N}$;
- $F = \{f(x) = x\}$;
- $M(d) = d$;
- $v(a, S, f, d) = \begin{cases} 1, & d = a \\ 0, & \text{otherwise} \end{cases}$.

Under these conditions, $g_f(a|S) = \min\{d : d \in \mathbb{N}, d = a\} = a$, which verifies all three properties of a Kolmogorov generic basis:

1. $g_f(a|S) = a \leq a = g_f(a|S)$, so $c_1 = 1$ and $c_2 = 0$;
2. $g(a) = a \leq a$, so $c_1 = 1$, $c_2 = 0$ and $h(a) = a$;
3. $g(a|S) = a \leq a = g(a)$, so $c_1 = 1$ and $c_2 = 0$.

Furthermore, A and D are obviously recursively enumerable, $M(d) = d$ is computable and F is a non-empty subset of the partial recursive function between $\mathbb{N}$. □

Proposition 3.2.4. *The Kolmogorov generic basis from Example 3.2.3 is in* $\mathcal{C}_2$. ♢

Proof. The three properties to belong to $\mathcal{C}_2$ are met since:

- the only f in F is $f(x) = x$, which is defined for all input;
- for all $a \in \mathbb{N}, f \in F$ and $d \in \mathbb{N}$, $v(a, \{\}, f, d) = 1$ if and only if $d = a$ if and only if $f(d) = a$;
- for each $i \in \mathbb{N}$, the set $\{d \in D : d = i\}$ is finite (in fact, it only has one element).

As such, the induced g is computable. □

For the purposes of Figure 3.1, we shall call the induced Kolmogorov generic function for this basis g^*.

We now consider the class $\mathcal{C}_3$ of Kolmogorov generic bases such that for all $i \in \mathbb{N}$, the set $\{d \in D : M(d) = i\}$ is finite. Obviously, $\mathcal{C}_2 \subset \mathcal{C}_3$. We will prove a result for this class inspired by Theorem 2.1.12.

Theorem 3.2.5. *For any Kolmogorov generic basis in $\mathcal{C}_3$, define a function $m : \mathbb{N} \to \mathbb{N}$ such that $m(x) = min\{g(i_A^{-1}(y)) : y \geq x\}$ for some enumeration of A. That is, m is the greatest monotonic increasing function bounding $g \circ i_A^{-1}$ from below. The function m is unbounded.* $\diamondsuit$

Proof. Since the number of possible elements of D with $M(d) = i$ is finite, the number of natural numbers with $g \circ i_A^{-1}$ equal to i is also finite. Hence, the number of natural numbers with $g \circ i_A^{-1}$ equal or greater to i is infinite.

As such, for each i there is a least an x_i such that for all $x > x_i$, $g(i_A^{-1}(x)) \geq i$. Clearly, for all i we have $x_{i+1} \geq x_i$. Consider an $x_i < x \leq x_{i+1}$. Since $x > x_i$, $g(i_A^{-1}(x)) \geq i$. Also, since $x \leq x_{i+1}$, $g(i_A^{-1}(x)) < i + 1$, i.e., $g(i_A^{-1}(x)) \leq i$. Hence, $g(i_A^{-1}(x)) = i$ and $m(x) = \min\{g(i_A^{-1}(y)) : y \geq x\} = g(i_A^{-1}(x)) = i$ so m is unbounded. □

As a corollary of this result, we now know that for $\mathcal{C}_3$ the induced Kolmogorov generic functions are unbounded. Within this class, we obviously have the Kolmogorov complexities based on partial recursive functions and Turing machines and prefix Kolmogorov complexity but not the Kolmogorov proof complexity. However, a modified version of the basis for CL can also be found in $\mathcal{C}_3$.

Consider a Kolmogorov generic basis for CL, like the one in Example 3.1.8, but with the difference that the number of propositional variables is finite and there is a maximum size for a formula. By size, we mean the number of connectives and propositional variables. Since the number of propositional connectives is already finite and so is the number of Frege rules for any Frege system, then there would be a finite number of derivation sequences for each length. For purposes of Figure 3.1, we shall call the induced Kolmogorov generic function for this basis CL^*.

Furthermore, since $\mathcal{C}_2$ is a subset of $\mathcal{C}_3$, g^* belongs to $\mathcal{C}_3$ as well.

So far, we know that $\mathcal{C}_3$ has Kolmogorov generic bases that induce unbounded functions that can be both computable (Theorem 3.2.2) or incomputable (Theorem 3.2.1). We will see below that some of those functions can be approximated by a computable function.

Define $\mathcal{C}_4$ as the class of Kolmogorov generic bases (A, D, M, F, v) in $\mathcal{C}_3$ such that:

- the function h from property 2 of Definition 3.1.1 is a total recursive function;
- for all $a \in A, f \in F$ and $d \in D$, $v(a, \{\}, f, d) = 1$ if and only if $f(d) = a$.

Theorem 3.2.6. *For any Kolmogorov generic basis in $\mathcal{C}_4$, there is a total recursive function $\phi(t, a)$, monotonic and decreasing in t, such that $lim_{t\to\infty}\phi(t, a) = g(a)$.* $\diamondsuit$

Proof. Consider a Kolmogorov generic basis in $\mathcal{C}_4$. Since for any f in F, its behaviour can be simulated by a deterministic Turing machine, consider one such f and the machine

that simulates it, M. Also, the induced unconditional Kolmogorov function will be $g(a) = g(a|\{\}) = \min\{M(d) : d \in D, v(a, \{\}, f, d) = 1\} = \min\{M(d) : d \in D, f(d) = a\}$, since v is such that $v(a, \{\}, f, d) = 1$ if and only if $f(d) = a$.

As such, we can construct ϕ on input t and a as follows:

- For each a in A, we know from property 2 of a Kolmogorov generic basis that $g(a)$ is at most $c_1\, h(a) + c_2$, where the constants c_1 and c_2 don't depend on a. As such, run machine M for t steps on each element of $\{d \in D : M(d) \leq c_1\, h(a) + c_2\}$ (this set if finite because of the defining properties of $\mathcal{C}_4$).

- If for any such input d the computation halts with output a, then define the value of $\phi(t, a)$ as the smallest value of $M(d)$ for those d's. Otherwise, define it as equal to $c_1\, h(a) + c_2$ (which we can calculate since h is total recursive).

It is easy to observe that $\phi(t, a)$ is recursive, total, and monotonically decreasing with t (for all a, $\phi(t_1, a) \leq \phi(t_2, a)$ if $t_1 > t_2$). Looking at $\lim_{t\to\infty}\phi(t, a)$, the limit exists, since for each a there exists a t such that M halts with output a after computing t steps starting with input d and satisfying $M(d) = g(a)$.

□

Inside this new class, we can only find the basis that induces g^*, since obviously the identity function is total recursive and $v(a, \{\}, f, d) = 1$ if and only if $f(d) = d = a$. However, the bases that induce C^F, C^M and K are not in $\mathcal{C}_4$ since the second condition is not verified for all f in F. In the case of C^F, this condition is true only for partial recursive additively optimal functions that satisfy $f(0p) = f(p)$ for all p. A similar condition is required for C^M and K. Furthermore, the bases that induce CL and CL^* are not in this class since the second condition isn't verified. In fact, in this case, $v(a, \{\}, f, d) = 1$ if and only if $\vdash_f^d a$ if and only if $f(\langle\{\}, d\rangle) = a$ if and only if d is a proof for a in the Frege system that induces f.

Of the results we proved for each type of complexity in Chapter 2, there is still one that we haven't discussed in this general setting: the incompressibility theorem (Theorem 2.1.10). For all the types of complexity we looked at, there is a common thread among the different D's which allowed us to prove some version of this theorem: each element of D is constructed by putting together the elements of some other set (for C^M, it is that each string can only be 0 or 1 in each position and for CL it is that each derivation is a sequence of formulas). However, since this property is not explicit in the definition of Kolmogorov generic bases, we did not prove any version of this theorem.

Figure 3.1 summarizes the results we presented in this section and shows how these classes are related with each other. However, one should take into account that these

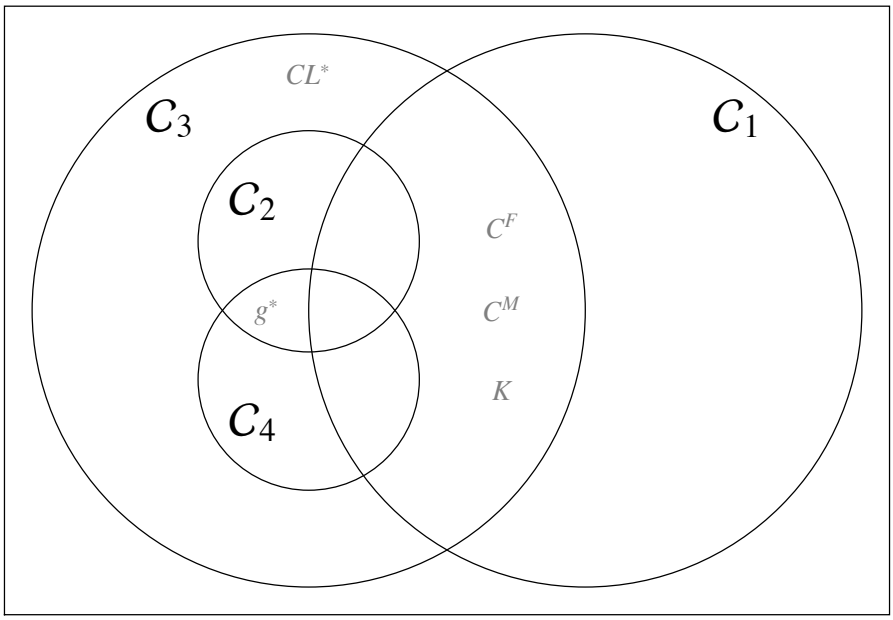

Figure 3.1: Some Kolmogorov generic bases notable classes: $\mathcal{C}_1$ has non computable functions, $\mathcal{C}_2$ has computable functions, $\mathcal{C}_3$ has unbounded functions and $\mathcal{C}_4$ has functions that can be approximated by a computable function.

theorems prove only one side of each implication, i.e., we did not prove for instance that class $\mathcal{C}_3$ is the class of all Kolmogorov generic bases that induce unbounded functions, nor do we suspect this is the case.

When looking at the figure, one should keep in mind that, for brevity's sake, we did not write the bases we have talked about but the induced functions. Also, in the legend, we obviously don't mean that, for instance $\mathcal{C}_2$ is a class of computable functions. We mean that $\mathcal{C}_2$ is a class of Kolmogorov generic bases that induce computable Kolmogorov generic functions.

Chapter 4

Conclusion

Over the course of this thesis, we tried to incorporate the most significant examples we could find of Kolmogorov generic bases. However, there could be plenty more to add to our studies. For instance, regarding Kolmogorov complexity, we could also look at the *length conditional Kolmogorov complexity* of a natural number x: $C^F(x|l(x))$.

Furthermore, there is the *uniform Kolmogorov complexity* of a finite string x with respect to a partial recursive function φ, defined as $C^F_\varphi(x; l(x)) = \min\{l(p) : \varphi([m, p]) = x_{1:m} \text{ for all } m \leq l(x)\}$. This complexity has the advantage of being monotonic over prefixes, something all the other versions of Kolmogorov complexity we have looked at are not.

Both these complexities can be found in [13] and it is likely that they could be seen as examples of Kolmogorov generic functions or, if not, the definition of Kolmogorov generic basis could be tweaked to accommodate them.

In the field of logic, a couple of other possible examples of Kolmogorov generic functions also come to mind. Firstly, we could consider the complexity $\tau(\varphi)$, as we have defined in Section 2.3 (the minimal number of subformulas for a derivation of formula φ). We have already proven a version of the invariance theorem for this complexity (Corollary 2.3.13) and an upper bound for it can be found in [12] so including it as an example seems like an easy task.

The last example we will mention, also related to Kolmogorov proof complexity, is a complexity where not the number of steps in the derivation is measured but the number of symbols when it is regarded as a string. We could call this $CL^{symb}(\varphi) = \min\{l(d) : d \text{ derives } \varphi\}$, where *symb* stands for symbol and $l(d)$ is the size of string d.

It may seem unimportant to study more examples of Kolmogorov generic basis but we believe that incorporating these instances will allow for a better understanding of how this structure works and facilitate reaching truly general results. At the moment, it seems

like a hard goal to reach due to how generally this structure is defined (the fact that it is able to accommodate both computable and incomputable Kolmogorov generic functions is a good example of this).

As such, it looks as though to be capable of proving a result applicable to any Kolmogorov generic basis, its definition should be improved, perhaps by finding a way to describe the commonalities in the structure of the different D's we studied (as explained at the end of Chapter 3), something we were not able to do.

However, no matter how much the definition is improved, at some point, one should abstract himself from the known examples of the Kolmogorov generic bases and focus solely on the definition at hand to see what can be proven with it.

In closing, we believe this thesis represents a first attempt at describing a structure that, as we have shown, can be successfully applied to a great deal of situations and, as such, could turn out to be extremely useful.

Bibliography

[1] In Samuel R. Buss, editor, *Handbook of Proof Theory*. Elsevier, 1998.

[2] Maria Luisa Bonet and Samuel R. Buss. The deduction rule and linear and near-linear proof simulations. *Journal of Symbolic Logic*, 58:688–709, 1993.

[3] Samuel R. Buss. Some remarks on lengths of propositional proofs. *Archive for Mathematical Logic*, 34:377–394, 1995.

[4] Samuel R. Buss. Towards NP-P via proof complexity and search. Technical report, 2011. To appear.

[5] Stephen Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. Cambridge University Press, 1st edition, 2010.

[6] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36–50, 1979.

[7] Lance Fortnow. Kolmogorov complexity and computational complexity. In *Complexity of Computations and Proofs. Quaderni di Matematica*, 2004.

[8] Lance Fortnow, John M. Hitchcock, A. Pavan, N. V. Vinodchandran, and Fengming Wang. Extracting kolmogorov complexity with applications to dimension zero-one laws. In *In procedings of the 33rd International Colloquium on Automara, Languages and Programing*, pages 335–345. Springer-Verlag, 2006.

[9] Nicola Galesi. On the complexity of propositional proof systems. Technical report, 2000. PhD Thesis.

[10] Alexander Hertel. Propositional proof complexity: A depth oral survey. Technical report, 2005. Depth Oral (PhD Candidacy) Paper.

[11] Jan Krajíček. Speed-up for propositional frege systems via generalizations of proofs. *Commentationes Mathematicae Universitatis Carolina*, 30:137–140, 1989.

[12] Jan Krajíček. *Bounded arithmetic, propositional logic, and complexity theory.* Cambridge University Press, 1995.

[13] Ming Li and Paul Vitányi. *An introduction to Kolmogorov Complexity and Its Applications.* Springer, 2008.

[14] John Alan Robinson. A machine oriented logic based on the resolution principle. *Journal of the Association for the Computing Machinery*, 12:627–631, 1965.

[15] Amílcar Sernadas and Cristina Sernadas. *Foundations of Logic and Theory of Computation.* College Publications, 2008.

[16] Michael Sipser. *Introduction to the Theory of Computation.* International Thomson Publishing, 1st edition, 1996.

[17] Osamu Watanabe. *Kolmogorov Complexity and Computational Complexity.* Springer-Verlag, 1992.