# Tightening the Compression Hierarchies

**MSc Thesis** (*Afstudeerscriptie*)

written by

**Navid Talebanfard**
(born May, 12th 1987 in Mashhad, Iran)

under the supervision of **Prof.dr. Harry Buhrman**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

**MSc in Logic**

at the *Universiteit van Amsterdam.*

| **Date of the public defense:** | **Members of the Thesis Committee:** |
|---|---|
| *July, 25th 2011* | Prof.dr. Dick de Jongh (chair) |
| | Dr. Leen Torenvliet |
| | Prof.dr. Harry Buhrman |

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

## Abstract

Fix $t$, $k$ and $n$ such that $k < n \leq t$ and let $\mathsf{EASY}^n_{t,k}$ be the set of all strings of length $n$ that are generated by programs of size $k$ in at most $t$ steps. It is not hard to see that for sufficiently large $t'$ (in fact for $t' > t \log t$ for technical reasons) we have $\mathsf{EASY}^n_{t,k} \subseteq \mathsf{EASY}^n_{t',k}$. But can we get a strict inclusion, or equivalently, is there an string $x$ that is generated by some $k$-bit program in $t'$ steps but cannot be generated in $t$ steps from $k$-bit programs? If so what is the smallest $t'$ for which a strict inclusion holds? Consider "the first (in lexicographic order) $x \in \{0, 1\}^n$ that is not generated in $t$ steps by any program of size $k$". The statement in quotations is already a description of that $x$ and it takes $2^k t \log t$ steps to output $x$. This shows that $\mathsf{EASY}^n_{t,k} \subset \mathsf{EASY}^n_{2^k t \log t,k}$. But is this exponential gap really needed? This question that still remains open will be the central topic of this thesis. We will examine different variants of it and will demonstrate its connection with deterministic simulations of randomized computation.

# Contents

# Chapter 1

# Introduction

Randomness has been the subject of scrutiny since antiquity, some thinkers taking it as an inherent property of objects, others viewing the world as governed by deterministic laws considering randomness as a subjective assessment of a phenomenon caused by the inability of men to understand the order of events. Democritus presented the example of two men sending their servants to get water and causing them to meet. The servants would see this meeting as random, as they are unaware of their masters' plan. Epicurus on the other hand believed that the world consisted of atoms, which swerve randomly along their paths independent of our observation.

After the advent of computers, researchers opened new perspectives into this issue in their attempts to quantify randomness. Consider the following game taken from [BM84] where Alice flips a coin and Bob wins if he guesses the outcome correctly. The game is played in the following ways.

1. Bob has to make his guess before the coin is flipped. In this case we do not expect him to win with more than 50% of chance.

2. Bob can announce his guess while the coin is spinning in the air. His chance of winning is the same even though he can in principle solve the equations of motion.

3. Same as the previous one with Bob having a pocket calculator. If he is fast enough he can solve the equations and hence increase his chance to say 51%.

4. This time Bob has a super computer and motion sensors that can record the initial velocity of the coin very quickly. In this case he can win with very high probability say 90%.

What outcome Bob counts as random depends on what he knows and what resources he has access to. The more he has the less randomness he finds, up to a point that in the last case he considers the outcome as almost non-random.

From this viewpoint what matters is how things look to us and to compuational devices with different capabilities, and thus the question whether there exists genuine randomness independent of observation becomes irrelevant.

Now is it true that an object $x$ that looks random to an observer $A$ looks non-random to a more powerful observer $B$? If so how much more power is needed to achieve this? We saw that the more resources Bob had access to, the less random he found the outcome of the game, and how much more power he needed depended on the difficulty of solving the equations of motion. Assume that $A$ and $B$ are computers and their power is bounded by the amount of time that they can work, in which case $B$ being more powerful than $A$ means that it can run for longer. We now restate the question: how much more time does a computer need to see some object as non-random which looked previously random to it?

To be able to give an acurate answer to this question we should first state what we mean by "random". When we fail to find patterns in something we usually blame this on the thing being random, and the less patterns we spot the more random we find it. Look at this carpet below. It consists of a few number of shapes that are repeated in a regular way all over the carpet. It is really hard to call it random looking.

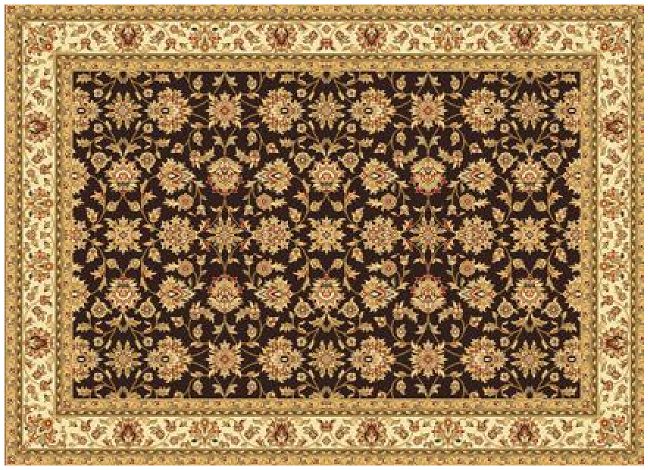

Figure 1.1: A picture of a carpet

On the other hand it is very unlikely that anyone can find any concise pattern in the following picture from Wolfram's "A New Kind of Science". It is important to make it clear what kind of patterns we are looking for and who wants to find these patterns. In the above pictures we searched for visual patterns, but only those that could be perceived by human beings. Even among humans two people need not have the same visual perception, different cultural backgrounds, education, etc. can effect one's experience. I think only the connoisseurs of postmodern art can find hidden patterns in this painting of Jackson

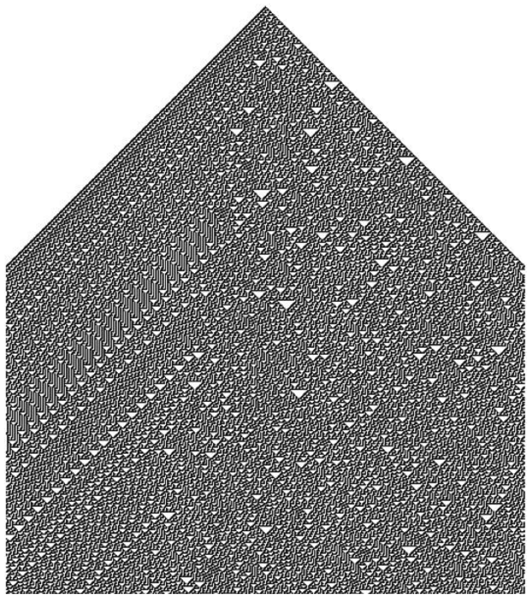

Figure 1.2: A random looking picture made by a simple program

Pollock (Figure 1.3).

Figure 1.2 is the output of a very simple program, and thus a computer that finds such a program sees the picture as quite regular. Taking computer program as the main observers of randomness was the core idea of an approach initiated by Solomonoff, Kolmogorov and Chaitin, who took objects whose smallest effective description has size at least the size of the object itself as random. An effective description is a process that transforms a short description of an object to the explicit object. These procedures are taken to be Turing machines and descriptions are programs. Hence if we define the Kolmogorov complexity of a string $x$ as $C(x) = \min\{|p| : U(p) = x\}$ where $U$ is a universal machine, then $x$ is random if $C(x) \geq |x|$. By introducing time bounds in this definition we will be able to express our question formally. For a function $t$ let $C^t(x) = \min\{|p| : U(p) = x \text{ in } t(|x|) \text{ steps}\}$. Now we have the following question.

**Question.** Fix any $n$ and consider all $n$-bit strings for which there is no program of length at most $k < n$ that generates them in time $t(n)$, i.e., for all such strings $x$ we have $C^t(x) > k$. How much more time do we need to get a shorter description of at least one of those $x$'s, i.e., what is the smallest[1] $t'$ such that there exists $x$ of length $n$ with $C^t(x) > k$ and $C^{t'}(x) \leq k$.

Considering time bounds moves us immediately into the realm of computa-

[1]For two functions $f$ and $g$ we say that $f$ is smaller than $g$ if $f = o(g)$.

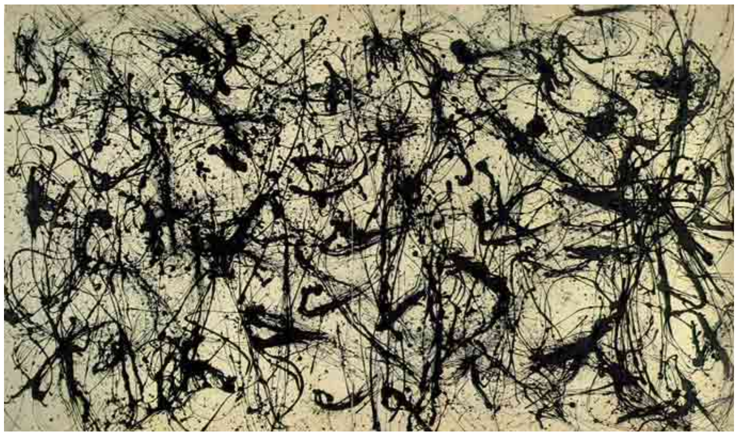

Figure 1.3: No. 32, Jackson Pollock

tional complexity which is the topic of Chapter 2 where we overview some basic concepts and relevant topics to our main problem. In Chapter 3 we state the problem formally and consider different variants of it, how it relates to imporant concepts such as circuit lower bounds and derandomization. Before discussing our main problem, we will first see two examples of hierarchy phenemenon in computational complexity. In Chapter 4 we discuss some possible further work and how additive combinatorics might be useful for this problem.

# Chapter 2

# Preliminaries

In this chapter we review some basics of computational complexity theory and Kolmogorov complexity required to study our main problem. For a gentle introduction to the theory of computation and an in-depth treatment of computational complexity we refer the reader to [AB09] or [Gol08].

## 2.1 Algorithms and Turing Machines

An *algorithm* is a set of instructions that if followed a certain goal is achieved. This includes primitive goals such as lighting a match or more sophisticated ones such as deciding whether an integer is a prime. For the former wikiHow.com says:

1. Hold the match firmly at about its middle with your pointer (index) finger and your thumb.

2. Put the match head at the end of the striker.

3. Firmly pressing the match head to the striker all the while, slide the match quickly along the striker. Remember, too much pressure will break the match while not enough will not light the match.

To test if an integer $n$ is a prime we will proceed as follows.

1. Write all numbers from 2 to $\lfloor \frac{n}{2} \rfloor$ in a row.

2. Choose some uncrossed number in the list. If it divides $n$ then $n$ is not prime, otherwise cross that number and repeat this step.

3. If all numbers are crossed then $n$ is prime.

Even though some take these two procedures fundamentally different[1], they both consist of steps each depending on previous ones, including some easy or comprehensible course of actions (holding the match for instance), and that it is guaranteed that if the instructions are followed flawlessly the desired outcome will be reached.

Hilbert, in his address in the International Congress of Mathematicians in 1900, asked for an algorithm that can decide or compute if a polynomial over integer coefficients has an integer root. In 1928 he posed a more general problem known as *Entscheidungsproblem* (German for "decision problem") which asked if mathematics was decidable, i.e., if there is an algorithm which could decide at least in principle whether any statement is true or not. To give a negative answer to this, one needs an explicit definition of an algorithm (holding the match between fingers simply wouldn't work). By 1936 several definitions were proposed, most imporant of which are those of Church and Turing. But all the proposed definitions turned out to be equivalent, in the sense that any algorithm in any model had an equivalent algorithm in any other model doing the same thing. This strenghened the thought that if bright minds such as those of mathematicians came up with the same thing when defining algorithm, maybe that is what algorithms actually are. This is the content of what is now known as the Church-Turing thesis.

**Church-Turing Thesis.** What could be intuitively computed, can be computed in the computation model of Turing (this model shall be defined shortly).

Turing's model of computation was based on how a human does basic arithmetic computations where one usually has a piece of paper and a pen at his disposal and can write new numbers and eliminate old ones following the laws of arithmetic to get the result. A Turing machine consists of an infinite tape (sometimes more than one tape) which is a collection of memory cells in linear order and a head that can point to these memory cells and scan them or write on them, a set of internal states and symbols, and a control program. Each step of the computation consists of scanning the memory cell to which the head is pointing, and taking the action which is instructed by the control program based on the internal state and the scanned symbol. The actions usually consist of a movement of the head either to the right or to the left, writing some symbol, and a change in the internal state. Before we give the formal definition we need some notations. The length of a string $x$ is denoted by $|x|$. For two string $x$ and $y$, $xy$ represents their concatenation and their pairing is $\langle x, y\rangle = 1^{|x|}0xy$. The set of binary strings of length $n$ is denoted by $\{0,1\}^n$. The set of binary strings of finite length is $\{0,1\}^* = \cup_n\{0,1\}^n$. All the sets that we consider in this thesis are subsets of $\{0,1\}^*$. A way to represent a set is via their characteristic functions. The characteristic function of $A$ denoted by $\chi_A$ is an infinite binary string whose $n$th bit is 1 if and only if the $n$th binary sequence in lexicographic

[1]Cleland [Cle93] argues that causality plays a defining role in the first procedure yet this is not so in the prime testing algorithm. The friction *caused* by pressing the match *causes* heat which in turn *causes* the match to light. There is no such relationship among the constituents of prime testing procedure.

order is a member of $A$. We can now define Turing machines formally.

**Definition 2.1.** A Turing machine $M$ is a tuple $(\Sigma, Q, \delta, q_{\mathsf{start}}, q_{\mathsf{reject}}, q_{\mathsf{accept}})$ where:

- $\Sigma$ is a finite set of symbols. In this thesis we only consider $\Sigma = \{0,1\}$.
- $Q$ is the set of internal states with three designated members $q_{\mathsf{start}}$, $q_{\mathsf{reject}}$ and $q_{\mathsf{accept}}$ such that $q_{\mathsf{reject}} \neq q_{\mathsf{accept}}$.
- The control program or transition function is $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{\mathsf{Left}, \mathsf{Right}\}$.

The computation of $M$ on $x$ denoted by $M(x)$ starts with having $x$ written down on the tape and $M$ being in state $q_{\mathsf{start}}$ and it continues by applying $\delta$ step by step. If at some point $M$ reaches $q_{\mathsf{accept}}$ we say that $M$ *accepts* $x$ and if it reaches $q_{\mathsf{reject}}$ we say that it *rejects* $x$. But in both cases we say that the computation *halts*, otherwise it does not halt and runs for ever. For a machine $M$, $L(M)$ is the set of all strings that $M$ accepts. If for $A \subseteq \{0,1\}^*$ there exists some Turing machine $M$ such that $A = L(M)$ we say that $A$ is *decidable* or *computable* by $M$. A striking result (by the time of its discovery) was that there exist *undecidable* sets. No matter how sophisticated a Turing machine we pick, it cannot compute any of those sets. To see this we note that each Turing machine $M$ can be encoded or represented by a binary string. Then as there are countably many finite binary strings, there will be only countably many Turing machines and hence countably many decidable sets. Since there are uncountably many sets of binary strings, there is some set that is not decidable.

We can similarly define computability of functions. For this we assume that in $M$ instead of two states of $q_{\mathsf{accept}}$ and $q_{\mathsf{reject}}$, there is one halting state called $q_{\mathsf{halt}}$. On input $x$ after $M$ reaches this state, what is left on the tape will be regarded as the output which is denoted by $M(x)$[2]. In this setting we can model acceptance and rejection of inputs by outputting 1 and 0, respectively. A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is called *partial computable* if there is some machine $M$ such that $f(x) = M(x)$ whenever $f(x)$ is defined. If such $f$ is a total function we say that it is *total computable.*

Turing proved that in his model there are *universal* machines, those that can simulate all other machines. This was the first theoretical proof of the possibility of general purpose computers (this came out as a surprise as it was proved long before digital computers were built). In formal terms a universal Turing machine is a machine $U$ such that for all $M$ and $x$ we have $U(\langle M, x\rangle) = M(x)$. This allows us to use arguments of the form: "construct a machine $M_1$ as follows: on input $x$ run $M_2(x)$ and do ...", which is always part of proofs in computability theory. Having this in mind and applying a *diagonalization method* we prove that the halting set defined by $\textbf{HALT} = \{\langle M, x\rangle : M \text{ halts on } x\}$ is undecidable. Assume to get a contradiction that it is decidable and thus there exists

[2]It will be clear from the context which definition of $M(x)$ we mean (computation of $M$ on $x$ or the output of $M$ on $x$).

some Turing machine $H$ such that **HALT** $= L(H)$. We take another machine $D$ that on input $x$ runs $H(\langle x, x\rangle)$. If $H$ rejects $D$ accepts, and if $H$ rejects $D$ falls into a loop and does not halt. If $D(D)$ halts then $H(\langle D, D\rangle)$ accepts, but then the construction of $D$ implies that $D(D)$ does not halt. If $D(D)$ does not halt then $H(\langle D, D\rangle)$ rejects and hence by construction $D(D)$ halts. This is a contradiction and thus **HALT** is undecidable.

Hilbert triggerred the birth of a whole new theory with his question which opened a new framework in mathematics and a novel methodology for even other fields of science. But it does not mean that his question was left unresolved. In 1970 Yuri Matiyasevich finally showed that Hilbert's desired algorithm does not exist, or now in computability terminology the set of polynomials over integer coefficients with integral roots is undecidable.

## 2.2 Computation under Time Bounds

The definition of decidability does not put any restriction on the computation, it only requires that the set could "in principle" be decided which means that the best machine deciding that set might require say one million years to halt, but as long as we can prove that it halts it is okay. But such a set is at least in practice undecidable, for it is very unlikely that human kind can witness the halting of a machine deciding it. To make decidability more down to earth we bring some efficiency measure into play: time!

We denote by $t_M(x)$ the running time of $M$ on input $x$ which is the number of steps that it takes $M$ to halt on $x$. We then define $t_M(n) = \max_{x\in\{0,1\}^n}\{t_M(x)\}$. We can now define our first class of sets decidable under time bounds. For any $t : \mathbb{N} \to \mathbb{N}$, $\mathsf{DTIME}(t(n))$ is the class of sets computable by machines running in time $t(n)$.

**Definition 2.2.** Important classes of such are:

1. $\mathsf{P} = \bigcup_{c\geq 1} \mathsf{DTIME}(n^c + c)$
2. $\mathsf{SUB\text{-}EXP} = \bigcap_{\epsilon>0} \mathsf{DTIME}(2^{n^\epsilon})$
3. $\mathsf{E} = \bigcup_{c\geq 1} \mathsf{DTIME}(2^{cn})$
4. $\mathsf{EXP} = \bigcup_{c\geq 1} \mathsf{DTIME}(2^{n^c})$

Here are some examples of problems in these classes.

**Graph 2-Colorability.** A coloring of a graph is an assignment of colors to the vertices of a graph, such that no two adjacent vertices have the same colors. For a graph $G$ the minimum number of colors for which a coloring is possible is denoted by $\chi(G)$. Let $\texttt{2-COL} = \{G : \chi(G) \leq 2\}$. It is easy to see that $\texttt{2-COL} \in \mathsf{P}$: start with any vertex $v$, color it with 0. Now take the neighbors of $v$ color them with 1. Continuing this if at some point we get a vertex that we want to color with $x$ but it has already been colored with $1-x$ we reject the graph. Otherwise we obtain a 2-coloring of $G$. The running time of this algorithm is bounded by the number of vertices in $G$.

**Satisfiability.** A Boolean formula over variables $x_1, \ldots, x_n$ is an expression consisting of variables and logical connectives AND ($\wedge$), OR ($\vee$) and NOT ($\neg$). A formula is *satisfiable* if there exists a valuation of its variables which makes the formula true. It is well known that each formula has an equivalent conjunctive normal form (CNF), i.e. a formula of the form $\bigwedge_i (\bigvee_j x_{i_j})$. Define

$$\mathbf{SAT} = \{\varphi : \varphi \text{ is a CNF and there exists } x \text{ such that } \varphi(x) = 1\}.$$

We have $\mathbf{SAT} \in \mathsf{E}$ since we can try all assignments of its variables and check whether any of them makes $\varphi$ true.

As in computability theory where one tries to show that some sets are decidable some are not, in computational complexity we would like to prove that some sets are decidable in time $t(n)$ or undecidable in time $t'(n)$. In the former case after proving that $A \in \mathsf{DTIME}(t_1(n))$ for instance, then one wants to get even smaller time bounds $t_2(n) = o(t_1(n))$ with $A \in \mathsf{DTIME}(t_2(n))$. The latter case on the other hand has resisted attempts; we are not yet able to prove the undecidability of specific sets in some time bounds. The most notorious example is $\mathbf{SAT}$ where the question whether $\mathbf{SAT} \in \mathsf{P}$ remains maybe the most famous problem in computational complexity. We will get back to this later again.

The reader should not despair for we might not be able to prove that specific sets are not decidable in certain time bounds, but we can show that there exist such sets. This is reminiscient of proving the existence of undecidable sets in the last section. This issue will be discussed in the next chapter. However we will need an important concept which goes as follows. Remember that in the last section using universal Turing machines we could run machines inside each other as subroutines. In time bounded computation we can of course do the same, but we might also need in some situations to run a subroutine machine for a certain number of steps, say $t(n)$ steps. For this to be possible $t(n)$ should be in some sense “nice”. Here is an example of how $t(n)$ could be not nice: take $t(n)$ to be some uncomputable function (which we know exists). There is no way that we can implement “run this for $t(n)$ steps”, as $t(n)$ does not make sense to any machine. As time is precious $t(n)$ should also be computable in short time, otherwise running the subroutine would not make that much sense. We can summarize this niceness requirement as follows.

**Definition 2.3.** A function $t : \mathbb{N} \to \mathbb{N}$ is called *time-constructible* if there exists a Turing machine $M$ such that $M(n) = t(n)$ and it is computed in time $t(n)$.

Another issue in running machines inside each other is the time overheads caused by simulation. Recall that when we defined universal Turing machines this was of no concern as what we needed was just the correct simulation and not the amount of time. Fortunately there exits an efficient universal Turing machine which will form the basis of some of our main concepts and results.

**Theorem 2.4. (Hennie-Stearns [HS66])** *There exists a universal Turing machine $U$ such that $U(\langle M, x\rangle) = M(x)$ for every $M$ and $x$ and there exists* *a constant $c$ depending on $M$ such that if $M$ halts on $x$ within $t$ steps, then $U(\langle M, x\rangle)$ halts within $ct\log t$ steps.*

Let us now get back to **SAT** as promised. The definition of **SAT** implies that if $\varphi \in$ **SAT** then there exists $x$ such that $\varphi(x) = 1$, and if $\varphi \notin$ **SAT** then for all $x$ we have $\varphi(x) = 0$. When $\varphi \in$ **SAT** a satisfying assignment is called a *witness*. Note that given any $\varphi$ and any assignment we can compute $\varphi(x)$ in linear time. Now we can say that the members of **SAT** have a witness and the correctness of the witness could be checked efficiently, whereas the non-members of **SAT** do not have any witness. This property of **SAT** is called having an *efficiently verifiable proof system* defined as follows.

**Definition 2.5.** A set $L$ has an efficiently verifiable proof system if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial time machine $A$ such that

1. Completeness: For every $x \in L$ there exists $y \in \{0,1\}^{p(|x|)}$ with $A(x, y) = 1$. Such $y$ is a proof or a witness for $x \in L$ and thus this condition says that every $x \in L$ has a proof.

2. Soundness: For every $x \notin L$ and every $y \in \{0,1\}^{p(|x|)}$ we have $A(x, y) = 0$ which means that no $x \notin L$ has a proof.

The class of sets having this property is called $\mathsf{NP}$. There is an alternative way of defining this class as follows. We change the Turing machines so that they have two transition functions $\delta_1$ and $\delta_2$. At each step it can choose which rule to follow. Note that we do not make any assumptions on how this choice is made, our machine is at ultimate liberty in making its decision. The running time of such a machine $M$ on an input is the maximum number of steps it takes before it halts over all sequences of non-deterministic choices. The machine accepts $x$ if at least one of its computation paths accepts $x$. We can now define time-bounded complexity classes in this model.

**Definition 2.6.** $\mathsf{NTIME}(t(n))$ is the class of sets accepted by a non-deterministic machine running in time $t(n)$. Furthermore $\mathsf{NP} = \bigcup_{c\geq 1} \mathsf{NTIME}(n^c + c)$.

Now let us see why these two definitions are equivalent. Take any set $L$ with an efficiently verifiable proof system. We can make a machine that on input $x$ non-deterministically tries to find a witness. If $x \in L$ there is some witness by assumption and our non-deterministic machine can find it. Otherwise all the computation paths will reject and hence $x$ will be rejected. Conversely, take $L \in \mathsf{NP}$ with its second definition. Now a witness for $x \in L$ is just an accepting computation path of a corresponding non-deterministic machine.

It is not hard to see that $\mathsf{P} \subseteq \mathsf{NP}$. The reason is simple: any polynomial time machine $M$ deciding a set $L \in \mathsf{P}$ is also an efficient proof verifier where the witnesses are empty. It was mentioned earlier that whether or not **SAT** $\in \mathsf{P}$ is the most famous problem in complexity. But this is not exactly true, in fact "the" problem is whether or not $\mathsf{P} = \mathsf{NP}$ and since we already know one direction, the question is if $\mathsf{NP} \subseteq \mathsf{P}$. Let us now see how a solution to **SAT** $\in ?\, \mathsf{P}$

is a solution to $\mathsf{NP} \subseteq ?\, \mathsf{P}$, i.e., $\mathbf{SAT} \in \mathsf{P} \Leftrightarrow \mathsf{NP} \subseteq \mathsf{P}$. Of course if $\mathbf{SAT} \notin \mathsf{P}$ then $\mathsf{NP} \not\subseteq \mathsf{P}$ as **SAT** is a member of NP. For the other direction we should prove that if $\mathbf{SAT} \in \mathsf{P}$ then all members of NP are in P. In fact we can show that if there is some polynomial time machine deciding **SAT**, for any set $A \in \mathsf{NP}$ the same machine could be slightly modified to decide $A$ in polynomial time. This means that **SAT** captures the difficulty of NP. Let us see what it exactly means.

The history of mathematics is replete with reductions, the process of transforming one problem into another, where a solution to the latter could be converted back into a solution to the original problem. The same theme also exists in computational complexity. We say that $A$ many-one reduces to $B$ denoted by $A \leq_{\mathrm{m}}^{\mathsf{P}} B$, if there exists a polynomial time computable function $f$ such that $x \in A \Leftrightarrow f(x) \in B$. A set $B$ is called $\mathcal{C}$-hard for some class $\mathcal{C}$ if for every $A \in \mathcal{C}$ we have $A \leq_{\mathrm{m}}^{\mathsf{P}} B$. A $\mathcal{C}$-complete set is a $\mathcal{C}$-hard set $B \in \mathcal{C}$. Intuitively a $\mathcal{C}$-complete set captures the hardness of $\mathcal{C}$, once we get an efficient algorithm for it, we have an algorithm for all other members of $\mathcal{C}$. Here is an example:

**Reduction from SAT to 3-SAT**: 3-**SAT** is a subset of **SAT** where each conjunct in its members consists only of three variables. Given any CNF formula $\varphi$ on $n$ variables and $m$ clauses we make a new CNF formula $\psi$ with each clause having three variables such that $\varphi$ is satisfiable if and only $\psi$ is. The idea is to substitute each clause $C$ with $k > 3$ variables with a pair of clauses $C_1$ on $k-1$ variables and $C_2$ with three variables and that $C$ is satisfiable if and only if $C_1 \wedge C_2$ is. Repeating this we will get the desired result. Let $C = x_1 \vee x_2 \vee \ldots \vee x_k$ be a clause in $\varphi$. Note that some of $x_i$'s might be negated, but this does not effect our argument. Introduce a new variable $y$ and define $C_1 = x_1 \vee \ldots \vee x_{k-2} \vee y$ and $C_2 = x_{k-1} \vee x_k \vee \neg y$. It is easy to see that $C_1$ and $C_2$ have the required properties. For the running time of the procedure note that we should repeat this for at most all of the clauses, and for each clause it is repeated at most $n$ times. Thus the running time is $O(mn)$ which is polynomial as desired.

As the reader might now guess **SAT** is NP-complete which was proved independently by Levin [Lev73] and Cook [Coo71]. This shows that 3-**SAT** is also NP-complete. In fact if $A$ is NP-complete and $A \leq_{\mathrm{m}}^{\mathsf{P}} B$ for some $B \in \mathsf{NP}$ then $B$ is also NP-complete. This follows from transitivity of $\leq_{\mathrm{m}}^{\mathsf{P}}$. Using this observation hundreds or even thousands of NP-complete problems have been found.

So far we have seen several classes of sets each capturing a certain computational property, e.g., P corresponds to efficient decidability and NP specified the efficient verifiability. We can unite this efficiency requirement and generalize these two classes as follows.

**Definition 2.7.** For $i \geq 1$, $\Sigma_i^{\mathsf{P}}$ is the class of sets $L$ for which there exists a deterministic poly-time machine $M$ and a polynomial $p$ such that

$$x \in L \Leftrightarrow Q_1 w_1 \in \{0,1\}^{p(|x|)} Q_2 w_2 \in \{0,1\}^{p(|x|)} \ldots Q_i w_i \in \{0,1\}^{p(|x|)} \quad \text{such that } M(x, w_1, \ldots, w_i) = 1,$$

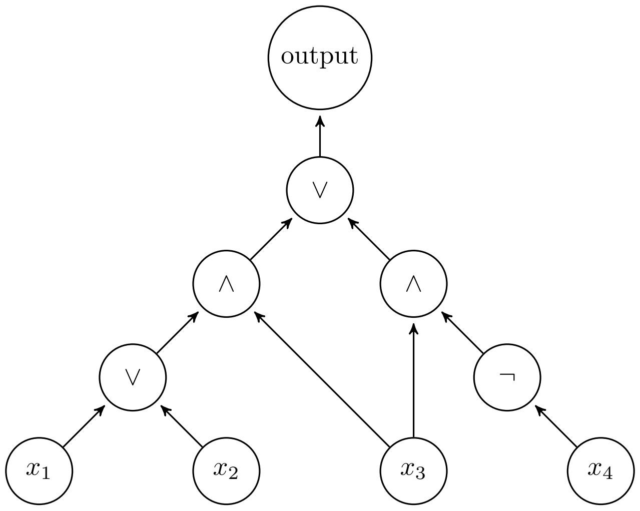

Figure 2.1: A Boolean circuit computing $f(x_1, x_2, x_3, x_4) = ((x_1 \vee x_2) \wedge x_3) \vee (x_3 \wedge \neg x_4))$.

where $Q_i$ denotes $\exists$ if $i$ is odd and $\forall$ if it is even. The *polynomial hierarchy* is defined as $\mathsf{PH} = \cup_i \Sigma_i^{\mathsf{P}}$.

A stronger form of $\mathsf{NP} \neq \mathsf{P}$ conjecture states that $\mathsf{PH}$ is indeed a hierarchy and its layers are distinct sets.

## 2.3 Non-Uniformity

The machines that we considered so far were designed so that they could deal with inputs of every length, or they were *uniform.* In this section we introduce the notion of non-uniform computation and in particular Boolean circuits. For each $n \in \mathbb{N}$ a *Boolean circuit* is an acyclic digraph with $n$ specified vertices as inputs and one or more vertices representing the outputs. The other vertices which are called *gates* are labeled by Boolean connectives. The number of edges going out of a vertex is called the *fan-out* of that vertex. Similarly the number of edges coming in to a vertex is called the *fan-in* of that vertex. The gates labeled with $\wedge$ and $\vee$ have fan-in 2, and gates with label $\neg$ have fan-in 1 and there is no restriction on the fan-out of any vertex. The size of a circuit $C$ is the number of its gates and is denoted by $|C|$. The gates apply their corresponding operations on their inputs and store the result (see figure 2.1).

**Definition 2.8.** Let $s : \mathbb{N} \to \mathbb{N}$ be a function. We say that a set $L$ is in $\mathsf{SIZE}(s(n))$ if there exists a sequence of circuits $C_1, C_2, \ldots$ with $|C_n| \leq s(n)$ such that for all $n$ and $x \in \{0,1\}^n$ it holds that $x \in L \Leftrightarrow C(x) = 1$. Furthermore $\mathsf{P}/\mathsf{poly} = \bigcup_{c \geq 1} \mathsf{SIZE}(n^c + c)$.

A nice and simple construction that simulates the Turing machine states, head movements and the tape information shows that $\mathsf{P} \subseteq \mathsf{P}/\mathsf{poly}$. An alternative definition of non-uniformity is given by the notion of advice functions. Let $\mathcal{C}$ be a class of sets and $\alpha : \mathbb{N} \to \mathbb{N}$ some function. A set $L$ is in $\mathcal{C}/\alpha$ if there exists a sequence of strings $a_1, a_2, \dots$ with $|a_i| \leq \alpha(i)$ and some $L' \in \mathcal{C}$ such that for all $x \in \{0,1\}^*$ it holds that $x \in L \Leftrightarrow \langle x, a_{|x|} \rangle \in L'$. It is not hard to see that $\mathsf{P}/\mathsf{poly} = \bigcup_{c,d} \mathsf{DTIME}(n^d)/n^d + c$. We say that a language $L$ has *circuit complexity* at least $s(n)$ if $L \notin \mathsf{SIZE}(s'(n))$ for all $s' = o(s)$.

Boolean circuits are more structured than Turing machine and for this reason people thought that it is probably easier to prove lower bounds for them. In fact using circuit lower bounds we can seperate uniform classes, e.g., if $\mathsf{NP} \not\subseteq \mathsf{P}/\mathsf{poly}$ then $\mathsf{P} \neq \mathsf{NP}$ since we know that $\mathsf{P} \subseteq \mathsf{P}/\mathsf{poly}$. This is a plausible approach since we believe that $\mathsf{NP}$ does not have polynomial sized circuits for the following reason.

**Theorem 2.9. (Karp-Lipton [KL80])** *If* $\mathsf{NP} \subseteq \mathsf{P}/\mathsf{poly}$ *then* $\mathsf{PH} = \Sigma_2^{\mathsf{p}}$.

But it is not always about complexity classes whose equality remains unknown. Take $\mathsf{EXP}$ for instance, we know that $\mathsf{P} \subset \mathsf{EXP}$ (see Theorem 3.1) and yet we cannot prove $\mathsf{EXP} \not\subseteq \mathsf{P}/\mathsf{poly}$. We only believe that is very unlikely that $\mathsf{EXP}$ could be simulated by polynomial sized circuits by the following theorem.

**Theorem 2.10. (Meyer, see [AB09])** *If* $\mathsf{EXP} \subseteq \mathsf{P}/\mathsf{poly}$ *then* $\mathsf{EXP} = \Sigma_2^{\mathsf{p}}$.

## 2.4 Randomized Computation

Sometimes we have to make random choices in real life. This happens usually when there is no obvious preference over any choice, or when we are in a game-like situation and we should keep our behavior as unpredictable as possible in order to win (or not to lose). Consider a simple game of rock-paper-scissors. If we are known as someone who plays paper most of the time, it is very likely that we are beaten in the next game by the opponent playing scissors. The most reliable thing to do is not to leave any trace, to play as patternless as possible, to play randomly.

We can now imagine algorithms that use some sort of randomness. For instance if we want to teach someone how to play rock-paper-scissors we use a phrase of the sort "play randomly". As in the beginning of the chapter that we tried to formalize the notion of algorithm, it is desirable to mathematically represent such algorithms. To model this class of algorithms we can assume that the machine has an extra tape where it keeps a string of random bits which are either given from external world or the machine itself has some internal function which can produce those random bits (where these bits come from does not have any impacts on our model). The output then depends not only on the input but also on the random string. Therefore on a single input the machine might give multiple answers and hence make errors. Based on how the error is made we can distinguish several classes of randomized computation. Let $A$ be some set and let $M$ be a randomized algorithm trying to decide $A$. If $M$ makes errors on both members and non-members of $A$ we say that it has *two-sided* error.

**Definition 2.11.** Let $t : \mathbb{N} \to \mathbb{N}$ be any function. For any set $A \subseteq \{0,1\}^*$ we say $A \in \mathsf{BPTIME}(t(n))$ if there exists a machine $M$ that receives inputs of the form $(x, r)$ and runs in time $t(|x|)$ such that $\Pr_{r\in\{0,1\}^{t(|x|)}}[M(x,r) = A(x)] \geq \frac{2}{3}$. The *probabilistic polynomial time* class is then defined as $\mathsf{BPP} = \bigcup_{c\geq 1} \mathsf{BPTIME}(n^c + c)$.

If we allow the machine to make error on only the members of a set, we get a *one-sided* error.

**Definition 2.12.** Let $t : \mathbb{N} \to \mathbb{N}$ be any function. $\mathsf{RPTIME}(t(n))$ is the class of sets $A$ for which there exists a machine $M$ that receives inputs of the form $(x, r)$ and runs in time $t(|x|)$ such that

1. $x \in A \Rightarrow \Pr_{r\in\{0,1\}^{t(|x|)}}[M(x,r) = 1] \geq \frac{2}{3}$
2. $x \notin L \Rightarrow \Pr_{r\in\{0,1\}^{t(|x|)}}[M(x,r) = 0] = 1$.

Similar to $\mathsf{BPP}$ we can define $\mathsf{RP} = \bigcup_{c\geq 1} \mathsf{RPTIME}(n^c + c)$.

We can easily reduce the error of probabilistic algorithms by simply repeating them. For $L \in \mathsf{RP}$ let $M$ be a machine witnessing this that uses random strings of length $p(n)$. We modify the algorithm by making a random string $r_1 r_2 \ldots r_k$ with $|r_i| = p(n)$. We then run $M(x, r_i)$ for each $i$ and accept the input if and only if $M(x, r_i)$ accepts for some $i$. By a simple union bound we can see that there error is reduced to $2^{-k}$. We can follow a similar strategy for $\mathsf{BPP}$ sets: we repeat the algorithm and take a majority vote among the answers. Then by Chernoff bounds the error reduction follows. Both of these procedures require $k \cdot p(n)$ random bits but it is desirable to use fewer bits. This is possible for both $\mathsf{BPP}$ and $\mathsf{RP}$. Let $\mathsf{Strong\,RP}$ be a set of $\mathsf{RP}$ sets for which there exists an algorithm usign $r(n)$ random bits with error bounded by $2^{-r(n)^\lambda}$ for some $\lambda > 0$. We then have the following result.

**Theorem 2.13. (Saks, et al. [SSZ95])** $\mathsf{RP} = \mathsf{Strong\,RP}$.

It is not hard to see that randomized computation could be simulated deterministically: just take all the possible random choices and take the majority of the answers. This clearly yields $\mathsf{BPP} \subseteq \mathsf{EXP}$. But can we do the simulation in less time, or going through all the coin flips is the best possible strategy? Given the ubiquitous applications of randomized algorithms and the fact that there are problems whose only known efficient algorithms are randomized (e.g. polynomial identity testing, see [KI04]), one might conjecture that $\mathsf{P} \subset \mathsf{BPP}$. But after the seminal work of Nisan and Widgerson [NW94] it is widely believed that $\mathsf{BPP} = \mathsf{P}$. In the remaining of this section we overview the main results in this field.

By *derandomization* we mean deterministic simulations of randomized computation, in particular if such a simulation yields $\mathsf{BPP} = \mathsf{P}$ we call it a *full* derandomization of $\mathsf{BPP}$. The main ingredient of a derandomization procedure is a pseudorandom generator. These are deterministic functions whose output can fool a certain class of algorithms, in the sense that they will be regarded as random. The first kind of pseudorandom generators were studied in the context of cryptography in several works for instance by Yao [Yao82] who proved that under the strong assumption of existence of what he called secure pseudorandom generators sub-exponential simulations of $\mathsf{BPP}$ is possible. For a detailed study of these pseudorandom generators we refer the reader to [Gol01].

**Definition 2.14.** Two sequences of random variables $\{X_i\}_{i\in\mathbb{N}}$ and $\{Y_i\}_{i\in\mathbb{N}}$ are called *indistinguishable in polynomial time* if there exit a function $l : \mathbb{N} \to \mathbb{N}$ such that $|X_n| = |Y_n| = l(n)$ and for all $n$ and for every randomized polynomial time algorithm $A$, every positive polynomial $p$ and sufficiently large $n$'s we have

$$|\Pr[A(X_n) = 1] - \Pr[A(Y_n) = 1]| < \frac{1}{p(l(n))},$$

where the probability is taken over the distributions of $X_i$'s and $Y_i$'s and the random coins of the algorithm.

**Definition 2.15.** A *secure pseudorandom generator* (PRG) is a deterministic polynomial time algorithm $G$ such that

1. there exists a function $l : \mathbb{N} \to \mathbb{N}$ called the *stretch function* for $G$ with $|G(x)| = l(|x|)$ for all $x \in \{0,1\}^*$,
2. and that $\{G(U_n)\}_{n\in\mathbb{N}}$ is indistinguishable in polynomial time from $\{U_{l(n)}\}_{n\in\mathbb{N}}$, where $U_k$ is a $k$-bit string chosen uniformly at random.

We have now enough tools to state the first conditional derandomization result:

**Theorem 2.16. (Yao [Yao82])** *If secure pseudorandom generators exist then* $\mathsf{BPP} \subseteq \mathsf{SUB\text{-}EXP}$.

*Proof.* (sketch) It is known that if secure pseudorandom generators exist, they also exist for arbitrary stretch functions (see [Gol01]). Let $A$ be a $\mathsf{BPP}$ algorithm that uses $r(n)$ random bits on inputs of length $n$. Fix $0 < \epsilon < 1$ and let $G$ be a pseudorandom generator that streches inputs of length $n^\epsilon$ to $n$. On an $n$ bit input $x$ for all $y \in \{0,1\}^{r(n)^\epsilon}$ we compute $A(x, y)$ and output the majority of answers. This clearly runs in $O(2^{r(n)^\epsilon} \cdot \mathsf{poly}(n))$ and it is not hard to see that it accepts the same language as $A$ does. $\square$

The crucial observation of Nisan and Widgerson was that the PRGs need not run in polynomial time, since in the derandomization procedure we already need to check all the seeds which costs us exponential time. Thus they considered the following kind which they coined quick PRGs.

**Definition 2.17.** A function $G$ computable in time $2^{O(n)}$ that maps strings of length $l(n)$ to strings of length $n$ is called *quick PRG* if for any circuit $C$ of size $O(n)$

$$\left|\Pr[C(G(U_{l(n)})) = 1] - \Pr[C(U_n) = 1]\right| < \frac{1}{n}.$$

The shorter the seed is the better derandomization we get, in particular if $l(n)$ is logarithmic we get a full derandomization of $\mathsf{BPP}$. Using an elegant construction, Nisan and Widgerson made PRGs out of functions which cannot be computed by circuits of certain size on average. This kind of hardness is defined formally below.

**Definition 2.18.** A Boolean function $f : \{0,1\}^* \to \{0,1\}$ is $(\epsilon, S)$-*hard* at $n$ if for any circuit $C$ of size at most $S$ we have

$$\left|\Pr_{x\in\{0,1\}^n}[C(x) = f(x)] - \frac{1}{2}\right| < \frac{\epsilon}{2}.$$

The *average hardness* of $f$ at $n$ denoted by $H_f(n)$ is the largest integer $h$ such that $f$ is $(\frac{1}{h}, h)$-hard at $n$.

**Theorem 2.19.** *If there exists $f \in \mathsf{E}$ with $H_f(n) = 2^{\Omega(n)}$ then there exists a PRG that stretches strings of length $c\log n$ to length $n$ and thus $\mathsf{BPP} = \mathsf{P}$.*

People then tried to get still weaker assumptions, an example of which is the following where average-case hardness is substituted by worse-case hardness.

**Theorem 2.20. (Impagliazzo-Widgerson [IW97])** *If there exists $L \in \mathsf{E}$ with circuit complexity $2^{\Omega(n)}$ then $\mathsf{BPP} = \mathsf{P}$.*

Until this point circuit lower bounds proved to be sufficient for derandomization, there are several results obtaining deranomization result from hardness assumption (see [AB09]). But these results all assumed specific circuit lower bounds, for instance $2^{\Omega(n)}$. One asks whether any circuit lower bound implies some derandomization which turned out to be the case in the following result.

**Theorem 2.21.** **[SU05]** *For any function $s$ if $\mathsf{E} \not\subseteq \mathsf{SIZE}(s)$ then $\mathsf{BPTIME}(t) \subseteq \mathsf{DTIME}(2^{O(s^{-1}(t^{O(1)}))})$, where $t$ is the running time of the probabilistic algorithm.*

## 2.5 Relativization

Oracle machines are Turing machines with access to an oracle that can solve a certain decision problem. We can model this by adding a read-only tape to the machine which contains the characteristic sequence of a set $O \subseteq \{0,1\}^*$. The machine can make queries of the form "is $q \in O$?" and these queries will be answered in a single step. We can then define complexity classes with respect to an oracle in a similar way. For instance $\mathsf{P}^O$ is the class of sets computable in polynomial time given access to oracle $O$. The usual complexity classes will then be special cases of such definitions when the oracle is $\emptyset$.

Take some complexity theoretic statement and its proof, for instance $\mathsf{P} \subseteq \mathsf{NP}$. Any poly-time machine $M$ deciding a set $L \in \mathsf{P}$ could be regarded as a verifier when the witnesses are empty strings. Notice that we are not making any assumptions on the oracle to which $M$ has access and thus we can say, $\mathsf{P}^O \subseteq \mathsf{NP}^O$ for any oracle $O$. This is an example of a *relativizing* proof. Many or maybe even most of the results in complexity indeed relativize, which is probably because relativizing techniques are natural approaches to algorithmic problems. This notion could be applied to assess the difficulty of an open problem. For instance if some statement $A$ is conjectured, and one is able to prove the existence of an oracle with respect to which $A$ does not hold, a proof for $A$ requires a non-relativizing technique. A key example of this kind is the following result which implies that any solution to $\mathsf{P}$ versus $\mathsf{NP}$ problem should not relativize.

**Theorem 2.22. (Baker-Gill-Solovay [BGS75])** *There exist oracles $A$ and $B$ such that $\mathsf{P}^A = \mathsf{NP}^A$ and $\mathsf{P}^B \neq \mathsf{NP}^B$.*

## 2.6 Kolmogorov Complexity

For a comprehensive discourse on the contents of this section we refer the reader to [LV08]. To measure the amount of information in an object, we try to describe it. The description is considered useful if we can fully reconstruct the object. Then the smallest description size would be regarded as the quantity of information in that object. Let $\mathcal{F} = \{f | f : \{0,1\}^* \to \{0,1\}^*\}$ be a class of functions. For $f \in \mathcal{F}$ and all $x \in \{0,1\}^*$ let $C_f(x) = \min\{|p| : f(p) = x\}$. In this setting $f$ is a description method and $p$ is a description of $x$ under $p$. We say that $f \in \mathcal{F}$ is a *minimal element* of $\mathcal{F}$ if for all $g \in \mathcal{F}$ there exists some constant $c$ depending on $g$ such that $C_f(x) \leq C_g(x) + c$ for all $x$. It turns out that if we let $\mathcal{F}$ to contain simply all functions, there will not be any minimal element. But if we take $\mathcal{F}$ to be the class of partial computable functions, we can prove the existence of minimal elements.

**Theorem 2.23. (Invariance Theorem)** *The class of partial computable functions has a minimal element.*

*Proof.* Let $\varphi_1, \varphi_2, \ldots$ be some effective enumeration of partial computable functions. Let $U$ be a universal Turing machine that expects inputs of the form $\langle n, p\rangle$, where $n$ encodes the $n$th Turing machine and $p$ is the literal program to be run on that machine. Let $\varphi_U$ be the function computed by $U$. We show that $C_{\varphi_U}(x) \leq C_{\varphi_i}(x) + O(1)$ for all $i$ and $x$. Fix some $i$ and for all $x$ let $p_x$ be the shortest program such that $\varphi_i(p_x) = x$. Since $\varphi_U$ is universal we have $\varphi_U(\langle i, p_x\rangle) = x$. But then $\langle i, p_x\rangle$ is a description of $x$ under $\varphi_U$ and its length is $2|i| + 1 + |p_x|$. Since this holds for every $x$ and $i$ was chosen arbitrarily the result follows. $\square$

From here on we simply write $C(x) = C_{\varphi_U}(x)$ for some choice of universal machine $U$ and by invariance theorem we are guaranteed that the description size is at most a constant longer than any other description. In case $U$ is given access to some oracle $A$, we denote the Kolmogorov complexity of $x$ with respect to $A$ by $C^A(x)$. However, it is important to note that this definition does not allow any measure on how difficult it is to generate a string from its shortest description. A fundamental measure of difficulty is the amount of time needed to compute something. There are strings that are generated by very short strings but in many steps. Hence we also add the factor of time and for a partial computable function $\psi$ and a time-constructible function $t : \mathbb{N} \to \mathbb{N}$ we define $C^t_\psi(x) = \min\{|p| : \psi(p) = x$ in at most $t(|x|)$ steps$\}$. Unfortunately we cannot get the same result as Invariance Theorem since the simulation costs us time. Yet using efficient universal Turing machines as in Theorem 2.4 we can get the following result.

**Theorem 2.24. (Time-Bounded Invariance Theorem)** *There exists a universal parital computable function $\varphi_U$ such that for every other partial computable function $\varphi$, there exists a constant $c$ such that $C^{ct\log t}_{\varphi_U}(x) \leq C^t_\varphi(x) + c$ for all $x$.*

Let $\varphi_U$ be the universal partial computable function of Theorem 2.24. We can now define the *time-bounded Kolmogorov complexity* of $x$ as $C^t(x) = \min\{|p| : U(p) = x$ in at most $t(|x|)$ steps$\}$. We can similarly write $C^{A,t}(x)$ when $U$ has $A$ as an oracle.

# Chapter 3

# Hierarchy Theorems

We are now ready to study our main problem. Recall from Chapter 1 that we want to examine the effects of bounded resources (time, in particular) on the ability to find patterns in objects. We will formulate the problem in the language of time-bounded Kolmogorov complexity and will present new results on the problem which are based on the works of Luís Antunes, Harry Buhrman, Lance Fortnow and André Souto. This is the first time that these results appear in written format.

In Chapter 1 we tried to motivate the problem from a philosophical point of view. But now that we have the required background we can argue that this problem is interesting from also a technical point of view. In fact this is a recurring theme in both theory of computation and computational complexity to see what we can do given a certain amount of resources. In such an approach the focus is mostly on decidability, in loose terms we can ask are there sets that look undecidable to a class of algorithms with certain power but look decidable to other algorithms with more power? Results of such are known as *hierarchy* theorems in computational complexity. To get familiar with this theme, we will first look at some such results before we get to our main proble

## 3.1 Decidability

Let $t : \mathbb{N} \to \mathbb{N}$ be some function and consider the class of sets computable in time $\mathsf{DTIME}(t(n))$. Of course this class does not contain all sets for the simple reason that there exist undecidable sets. But what about decidable sets? Are they all contained in $\mathsf{DTIME}(t(n))$? Intuitively we expect that if we are given more time we can compute more, and as Hartmanis and Stearns proved this is indeed the case.

**Theorem 3.1. (Time Hierarchy Theorem [HS65])** *For any time constructible functions $t$ and $t'$ satisfying $t(n) \log t(n) = o(t'(n))$, then $\mathsf{DTIME}(t(n)) \subset \mathsf{DTIME}(t'(n))$.*

*Proof.* (sketch) We should construct a set $L \in \mathsf{DTIME}(t')$ but not in $\mathsf{DTIME}(t)$. Consider algorithm $A$ which on input $x$ runs $M_x(x)$ for $t(|x|)$ steps and outputs $1 - M_x(x)$ ($M_x$ is the Turing machine described by $x$). Let $L$ be the set decided by $A$. We claim that $L \notin \mathsf{DTIME}(t)$. Assume to get a contradiction that this is the case and there is a $t$-time machine $M_z$ deciding $L$. On input $z$ by the construction of $A$ we get $M_z(z) = 1 - M_z(z)$ and this is a contradicition. It remains to prove that $L \in \mathsf{DTIME}(t'(n))$. We will not go into the details of this fact, but we note that we should bound the running time of $A$ which follows from efficient simulations of Turing machines. $\square$

We now address the same question for probabilistic classes, whether we can compute more having more time. First we argue why we believe that such a hierarchy exists. We mentioned earlier that it is believed that $\mathsf{BPP} = \mathsf{P}$ which implies that for every $c$ there exists $d$ such that $\mathsf{BPTIME}(n^c) \subseteq \mathsf{DTIME}(n^d)$. But by Theorem 3.1 there is some $d'$ such that $\mathsf{DTIME}(n^d) \subset \mathsf{DTIME}(n^{d'})$. This implies that $\mathsf{BPTIME}(n^c) \subset \mathsf{DTIME}(n^{d'}) \subseteq \mathsf{BPTIME}(n^{d'})$, where the last inclusion follows from the fact that deterministic computation is special case of randomized computation with zero error. Proving hierarchy theorems for probabilistic classes is not as straightforward as for deterministic classes. In fact it is still an open problem whether $\mathsf{BPTIME}(n^c) \subset \mathsf{BPTIME}(n^d)$ for $c < d$. Standard techniques such as diagonalization do not seem to be applicable here. Notice that the machines $M$ over which we should diagonalize must have a certain property, namely for all $x \in \{0,1\}^*$ it must hold either $\Pr[M(x) = 1] > \frac{2}{3}$ or $\Pr[M(x) = 1] < \frac{1}{3}$, and of course there exist machines for which this probability lies between $\frac{1}{3}$ and $\frac{2}{3}$ and it is not an easy task to decide this.

However, starting with the paper of Barak [Bar02], several results towards establishing a hierarchy were discovered. To apply a diagonalization method, Barak introduces a notion of reduction between sets which is slightly different from the standard reduction. He allows the complexity of reduction to depend on the complexity of the reduced set. For instance if $L$ reduces to $L'$ then the time that this reduction takes is somehow similar to the time that $L$ requires.

**Definition 3.2.** A set $B$ is called $\mathsf{BPTIME}$-hard if there exists $c$ such that for all time-constructible functions $t : \mathbb{N} \to \mathbb{N}$ and and any $A \in \mathsf{BPTIME}(t(n))$ there exists a function computable in $t(|x|)^c$ deterministic time such that $x \in A \Leftrightarrow f(x) \in B$ for all $x \in \{0,1\}^*$. Furthermore, if $B \in \mathsf{BPP}$ then it is said to be $\mathsf{BPP}$-complete.

**Theorem 3.3. (Barak [Bar02])** *If* $\mathsf{BPP}$ *has a complete set, then there exists $c$ such that for every time-constructible time bound $t : \mathbb{N} \to \mathbb{N}$ we have* $\mathsf{BPTIME}(t(n)) \subset \mathsf{BPTIME}(t(n)^c)$.

*Proof.* Let $L$ be a $\mathsf{BPP}$-complete set with a probabilistic machine $N$ deciding $L$ running in time $n^c$. Given that $L$ is $\mathsf{BPTIME}$-hard we know that there exists a constant $d$ such that for every time-constructible $t$, every $\mathsf{BPTIME}(t(n))$ set is reducible to $L$ in $t(n)^d$ determinisitc time. Fix some time-constructible function $t$ and an enumeration $M_1, M_2, \ldots$ of Turing machines and define the following

set
$$L' = \{x : M_x^{t(|x|)^d}(x) \notin L\},$$
where $M_x^y$ is just $M_x$ run for $y$ steps. We now show that

(i) $L' \in \mathsf{BPTIME}(t(n)^{bcd})$ for some constant $b$

(ii) $L' \notin \mathsf{BPTIME}(t(n))$

To prove the former we give the following algorithm $D$: on input $x$ we just output $1 - N(M_x^{t(|x|)^d}(x))$. Since $N$ is a $\mathsf{BPP}$ machine deciding $L$ we obtain

1. $M_x^{t(|x|)^d}(x) \notin Ł \Rightarrow \Pr[D(x) = 1] \geq \frac{2}{3}$
2. $M_x^{t(|x|)^d}(x) \in L \Rightarrow \Pr[D(x) = 1] < \frac{1}{3}$,

which proves (i). Now assume that $L' \in \mathsf{BPTIME}(t(n))$. Since $L$ is complete there is a reduction from $L'$ to $L$. Let this be witnessed by $M_i$ and hence $x \in L' \Leftrightarrow M_i^{t(|x|)^d}(x) \in L$ and in particular $i \in L' \Leftrightarrow M_i^{t(|i|)^d}(i) \in L$, which is a contradiction by definition of $L'$ and we are done. □

Barak also shows that if $\mathsf{NP} \subseteq \mathsf{BPP}$ then $\mathsf{BPP}$ has a complete problem and thus has a hierarchy. By a padding argument he concludes that $\mathsf{BPTIME}(n) \neq \mathsf{NP}$. In the same paper Barak follows another strategy to get hierarchy results. He considers *optimal* probabilistic algorithms for certain sets, optimal in the sense that they are slower to only a polynomial factor than any other algorithm deciding that set. For instance if there is an optimal algorithm running in time $t(n)$, then there is some constant $c$ such that any other algorithm deciding that set runs in time at least $t(n)^{\frac{1}{c}}$. Using this approach he was able to prove a hierarchy result when there exists a little bit of non-uniformity.

**Theorem 3.4. (Barak [Bar02])** *For all constants $c \geq 1$ there is some constant $d$ such that $\mathsf{BPTIME}(n^d)/\log\log n \not\subseteq \mathsf{BPTIME}(n^c)/\log n$.*

Improving the optimal algorithm of Barak, Fortnow and Santhanam were able to reduce the number of advice bits to 1.

**Theorem 3.5. (Fortnow-Santhanam [FS04])** *For all constants $1 < c < d$ it holds that $\mathsf{BPTIME}(n^c)/1 \subset \mathsf{BPTIME}(n^d)/1$.*

## 3.2 Randomness

Let $f$ and $t$ be functions such that $f(n) \leq n \leq t(n)$ for all $n$. Consider the following class of strings [Har83]:
$$C[f, t] = \{x : C^t(x) \leq f(|x|)\},$$
the set of strings that are generated by some program of size $f(n)$ in time $t(n)$. Let $t'$ be such that $t'(n) > t(n) \log t(n)$. It is obvious that $C[f, t] \subseteq C[f, t']$,

the same program witnessing $C^t(x) \leq f(|x|)$ yields $C^{t'}(x) \leq f(|x|)$. A natural question is if we can find a smaller description of such strings if we had more time than $t$ or in other words if this inclusion is strict? The intuition suggests that it should be the case, since if we are given more time we can find more patterns and hence we can express the string with fewer bits. By a simple argument we can prove that if we are given exponentially more time, i.e., if $t'(n) \geq c2^{f(n)}t(n)\log t(n)$, then the inclusion is strict. This is formally stated below.

**Theorem 3.6. (Longpré [Lon86])** *Let $f$ and $t$ be functions computable in time $t$ with $f$ not bounded by any constant and $f(n) < n$. There is some constant $c$ such that $C[f,t] \subset C[f, c2^f t \log t]$.*

*Proof.* For any $k$ consider the following program $p_k$:

1. find the smallest $l$ with $f(l) > k$
2. output the first $x$ of length $l$ with $C^t(x) > f(l)$.

As $f$ is not bounded by any constant, the first step halts successfully by finding an appropriate $l$. For the second step we simply run all the programs of length at most $2^{f(l)}$ for $t(l)$ steps and output the lexicographically first $x$ that was not generated by any of those. Such $x$ exists since $f(l) < l$[1], and hence the program halts. The running time is clearly $c2^{f(l)}t(l)\log t(l)$, where $t(l)\log t(l)$ is incurred by the simulation of programs. For the size of $p_k$ notice that it contains $k$ with some constant size program. So we can assume $|p_k| = d + \log k$ for some constant $d$. Choosing $k$ large enough we get $|p_k| \leq k$. For such $k$ the output of $p_k$ is generated by a program of size at most $k < f(|x|)$ in time $c2^{f(|x|)}t(|x|)$ and as the second step guarantees that $C^t(x) > f(|x|)$ the result follows.

□

Now the question is whether we can improve the result of Theorem 3.6 in the sense that we can compress better if we are given only polynomially more time. We will refer to this problem as the *compression hierarchy problem.* This is reminiscient of the Time Hierarchy Theorem where having more time we can compute more. But things are not as easy with Kolmogorov complexity. Buhrman and Fortnow considered a similar question and asked for strings that do not have short descriptions in a certain time, but with a little bit more time we can give almost optimal descriptions[2] of them. Their approach to this problem was to consider strings of length $2^n$ as the characteristic strings of sets corresponding to the $n$-bit members. They showed that the time-bounded Kolmogorov complexity of such strings is in some sense equivalent to their circuit complexity, and hence a set whose characteristic sequence is composed of strings with high Kolmogorov complexity does not have small circuits. The following theorem is a formalization of this idea.

[1]Each program of size $f(l)$ can generate at most one string. Therefore the number of strings $x$ of length $l$ with $C(x) \leq f(l)$ is at most $2^{f(l)} < 2^l$. Thus there exists some string of length $l$ with Kolmogorov complexity at least $f(l)$.

[2]For a string $x$ of length $n$ a description of size $O(\log n)$ could be regarded as optimal, since we need to specify at least the length of $x$.

**Theorem 3.7. (Buhrman and Fortnow [BF11])** *The following statements are equivalent:*

1. *There exist constants $d, c, d'$ and $c'$ such that for all sufficiently large $n$ there exists $x$ with $|x| = N = 2^n$ satisfying $C^{N^d}(x) \leq c \log N$ and $C^{N^{d'}}(x) \geq N^{c'}$,*

2. *There exists $L \in \mathsf{E}$ with circuit complexity at least $2^{\Omega(n)}$, i.e., there is some constant $b > 0$ such for all sufficiently large $n$ we have $s(L^{=n}) \geq 2^{bn}$.*

*Proof.* (1) $\to$ (2): Define the following set

$$L = \{x : \lfloor (1+c)n \rfloor \leq |x| < \lfloor (1+c)(n+1) \rfloor, x = pqr, |p| = \lfloor cn \rfloor, |r| = n, \\ U^{2^{dn}}(p)_r = 1\}.$$

We first check that $L \in \mathsf{E}$. On an input $x$ with $|x| = n$ we cut it into $x = pqr$ with proper lenghts. Then we run $U(p)$ for $2^{d|r|}$ steps and output the $r$th bit. Hence the running time is $O(n) + 2^{d|r|} = 2^{O(n)}$.

Assume now to get a contradiction that for all $b > 0$, there are infinitely many $n$'s such that $s(L^{=n}) < 2^{bn}$. For any such $n$ let $C_n$ be a circuit witnessing this. Take $n'$ such that $\lfloor (1+c)n' \rfloor \leq n < \lfloor (1+c)(n'+1) \rfloor$. Take some $x$ with $|x| = N = 2^{n'}$ with $C^{N^d}(x) \leq cn'$ witnessed by $|p| = \lfloor cn' \rfloor$ and $C^{N^{d'}}(x) \geq N^{c'}$. The following algorithm outputs $x$:

```
for $i = 1$ to $2^{n'}$ ($i$ represents the $i$th string of length $n'$) do
    OUTPUT $C_n(p0^{n-n'-\lfloor cn' \rfloor}i)$
end for
```

This algorithm generates $x$ from at most $n' + 2^{bn} < 2^{\frac{bc}{c+1}(n'+1)+1} < 2^{\frac{2bc}{c+1}n'}$ bits in time $2^{n'} + 2^{bn} < 2^{\frac{2bc}{c+1}n'}$. Thus if we choose $b$ small enough so that $\frac{2bc}{c+1} < d'$ we get a contradiction.

(2) $\to$ (1): Let $L$ be a set in $\mathsf{E}$ computable in time $2^{dn}$ by some algorithm $A$ with circuit complexity $2^{\Omega(n)}$. Therefore there exists a constant $b$ such that for sufficiently large $n$, $L^{=n}$ requires circuits of size at least $2^{bn}$. For such $n$'s let $x_n$ be the string of length $2^n$ corresponding to the characteristic sequence of $L$ confined to the strings of length $n$. Program $p_n$ defined as follows clearly outputs $x_n$: for $i \in \{0,1\}^n$, the $i$th bit of output is $A(i)$. Notice that $p_n$ runs in time $2^n \times 2^{dn} = 2^{(d+1)n}$ and to represent it we just need a description of $n$ and $A$ with some constant size program which will not cost us more than $O(\log n)$ bits. This proves that $C^{N^{d+1}}(x_n) \leq O(\log N)$. Assume for the sake of contradiction that for all $c'$ and $d'$ we have $C^{N^{d'}}(x_n) < N^{c'}$ with some program $p$ witnessing this. Let $A(p,i)$ be the algorithm that runs the program $p$ and outputs the $i$th bit of the result. This clearly decides $L^{=n}$ and since any algorithm running in time $t$ could be converted to a circuit of size $t^2$ deciding the same language, by hardwiring $p$ we get a circuit for $L^{=n}$ of size $2^{c'n} + 2^{2d'n}$. Choosing $c'$ and $d'$ small enough this sum will be smaller than $2^{bn}$ contradicting the assumption that $s(L^{=n}) \geq 2^{bn}$. □

It is not hard to see that the proof of Theorem 3.7 relativizes since the universal machine $U$ used in the proof could safely be substituted by $U^O$ for any oracle $O$ without any impact on the argument. A result of Wilson [Wil85] states that there exists an oracle $A$ such that $\mathsf{EXP}^A \subseteq \mathsf{P}^A/\mathsf{poly}$ and consequently $\mathsf{E}^A \subseteq \mathsf{P}^A/\mathsf{poly}$. But then by Theorem 3.7 there exists a relativized world $A$ where for all constants $d$, $c$, $d'$ and $c'$ there are infinitely many $n$'s such that for all $x$ with $|x| = N = 2^n$ we have either $C^{A,N^d}(x) > c \log N$ or $C^{A,N^{d'}}(x) < N^{c'}$, which means that proving a hierarchy of the form as in statement (1) of Theorem 3.7 requires non-relativizing techniques. Also note that applying Theorem 3.7 and the derandomization result of Impagliazzo and Widgerson (Theorem 2.20) proving Theorem 3.7 (1) implies the full derandomization of $\mathsf{BPP}$. This is stated formally as follows.

**Corollary 3.8.** *If there exist constants $d, c, d'$ and $c'$ such that for all sufficiently large $n$ there exists $x$ with $|x| = N = 2^n$ satisfying $C^{N^d}(x) \leq c \log N$ and $C^{N^{d'}}(x) \geq N^{c'}$, then $\mathsf{BPP} = \mathsf{P}$.*

Let us get back to the compression hierarchy problem. Note that a negative answer to this problem implies that for all time bounds that are only polynomially far from each other, the respective Kolmogorov complexity of all strings is (almost) constant. Antunes, et al. took this approach and applying a similar argument as in Theorem 3.7, proved the following theorem.

**Theorem 3.9. (Antunes et al. [ABFS11])** *For every $f : \mathbb{N} \to \mathbb{N}$ with $f(n) \leq n$ for sufficiently large $N = 2^n$ the followings are equivalent:*

1. *$\exists c \forall bd \exists a$ such that for all $x$ with $|x| = N$ and*

$$C^{N^d}(x) \leq f(N) + b\log(N) \Rightarrow C^{N^c}(x) \leq f(N) + a\log(N).$$

2. *$\exists c' \forall bd \exists a$ such that*

$$\mathsf{DTIME}(2^{dn})/f(2^n) + bn \subseteq \mathsf{DTIME}(2^{c'n})/f(2^n) + an.$$

*Proof.* (1) $\to$ (2): Set $c' = c + 1$. Take any $A \in \mathsf{DTIME}(2^{dn})/f(2^n) + bn$ and for any $n$ consider $\chi_A^{=n}$. Let $M$ be the $\mathsf{DTIME}(2^{dn})$ machine deciding $A$ with the advice sequence $\alpha_1, \alpha_2, \ldots$ such that $|a_i| \leq f(2^i) + bi$. Now consider the following program:

```
for $i = 1$ to $2^n$ do
    $\chi_i = M(x_i, \alpha_n)$
end for
```

It is clear it runs in time $2^n \cdot 2^{dn} = 2^{(d+1)n}$ and the output $\chi$ is just $\chi_A^{=n}$. This description contains $n$, $\alpha_n$ and some constant size program corresponding to $M$ and the for loop. The size of this description is $|\alpha_n| + \log n + C$ for some constant $C$ and consequently there is some $b'$ such that $f(2^n) + bn + \log n + C \leq f(2^n) + b'n$. We just showed that

$$C^{|\chi_A^{=n}|^{d+1}}(\chi_A^{=n}) \leq f(|\chi_A^{=n}|) + b' \log(|\chi_A^{=n}|).$$

We can thus apply 1 and get constants $a$ and $c$ such that

$$C^{|\chi_A^{=n}|^c}(\chi_A^{=n}) \leq f(|\chi_A^{=n}|) + a\log(|\chi_A^{=n}|).$$

Let $\alpha_n'$ be such a description of $\chi_A^{=n}$. We construct a machine $M'$ as follows: on an $n$-bit input $x$ which is the $i$th string in the lexicographic ordering of $n$-bit strings, it runs $U(\alpha_n)$. If the $i$th bit of the output is 1 it accepts $x$, otherwise it rejects. By assumption $U(\alpha_n)$ runs in time $2^{cn}$ and we need at most $2^n$ more steps to check the $i$th bit of the output. So the total running time is $2^{cn} + 2^n \leq 2^{c+1}n = 2^{c'n}$. This shows that $A \in \mathsf{DTIME}(2^{c'n})/f(2^n) + an$.

(2) $\rightarrow$ (1) For all $n$ let $x_n$ with $|x| = n$ be any string such that $C^{n^d}(x) \leq f(n) + b\log n$ witnessed by a program $p_n$. Now take a set $A$ whose characteristic sequence is $x_1x_2x_3\ldots$, i.e., the contcatenation of all $x_i$'s. We claim that $A \in \mathsf{DTIME}(2^{dn})/f(2^n) + bn$. To show this weuse the advice sequence $p_1, p_2, \ldots$ and a machine $M$ running in time $2^{dn}$ such that $z \in A \Leftrightarrow M(z, p_{|z|}) = 1$. Machine $M$ works as follows: on input $(z, p_{|z|})$ with $|z| = n$ it runs $U(p_n)$ for $2^{dn}$ steps and outputs the $z$th bit of the result. This clearly satisfies our requirement. Now by assumption we get $A \in \mathsf{DTIME}(2^{cn})/f(2^n) + an$. Let this be witnessed by a machine $M'$ and an advice sequence $p_1', p_2', \ldots$. We can now generate $\chi_A^{=n}$ from $M'$ as follows: we run $M'(x, p_n')$ and write the output in turn for all $x$ of length $n$ in lexicographic order. This sequence of outputs is clearly $\chi_A^{=n}$ and it is generated in time $2^n \times 2^{cn} = 2^{c'n}$. This description requires $M'$, $n$ and $p_n'$ and thus its size is bounded by $|p_n'| + n + O(1) \leq f(2^n) + a'n$ for some constant $a'$. Since $x_i$'s were chosen arbitrarily the result is established. □

Similar to Theorem 3.7, all the simulations in the proof of Theorem 3.9 could be done with respect to any oracle and hence Theorem 3.9 relativizes. The following result says that there exist relativized worlds where the second statement in Theorem 3.9 holds. This result and the fact that Theorem 3.9 relativizes imply that a proving that the first statement (1) in Theorem 3.9 is false, requires non-relativizing techniques.

**Theorem 3.10. (Antunes et al. [ABFS11])** *For every $f : \mathbb{N} \rightarrow \mathbb{N}$ with $f(n) \leq n$ for all $n$, there exists $A$ such that $\exists c \forall bd \exists a$ such that*

$$\mathsf{DTIME}^A(2^{dn})/f(2^n) + bn \subseteq \mathsf{DTIME}^A(2^{cn})/f(2^n) + an.$$

*Proof.* Let $g(n) = f(2^n) + bn$. We construct $A$ in stages by adding new elements to the already constructed part. Let $A_n$ denote the part of $A$ that has been constructed by the end of stage $n$. Then $A = \lim_{n\rightarrow\infty} A_n$.
Stage 0: At this stage we set $A_0 = \emptyset$.
Stage $n$: Assume that $A_0, \ldots, A_{n-1}$ have been constructed. Let $M$ be any machine running in time $2^{dn}$.

Step 1: For all $x \in \{0,1\}^{\leq n}$ and all $y$ with $|y| \leq g(|x|)$, consider the computation of $M^{A_{n-1}}(x, y)$ and let $Q_n$ be the set of all queries made on these computations.

Step 2: For all $y \in \{0,1\}^{g(n)}$ we choose a distinct $z_y$ with $|z_y| \leq g(n) + an$ with $a$ to be chosen later such that for all $x \in \{0,1\}^n$ it holds that $\langle x, \langle M, z_y \rangle\rangle \notin Q_n$. We add $\langle x, \langle M, z_y \rangle\rangle$ to $A$ if and only if $M^{A_{n-1}}(x,y) = 1$ (the fact that $\langle x, \langle M, z_y \rangle\rangle \notin Q_n$ guarantees that after adding this element to $A$, the computation of $M^A(x,y)$ is the same as $M^{A_{n-1}}(x,y)$).

**Fact 1.** Step 2 succeeds by finding $2^{g(n)}$ many $z_y$'s satisfying

$$(*) \forall y \in \{0,1\}^{g(n)} \forall x \in \{0,1\}^n \langle x, \langle M, z_y \rangle\rangle \notin Q_n.$$

*Proof.* Consider a 01-matrix in which columns are labeled with strings of length $n$ and rows are labeled with strings of length $g(n) + an$. In the $(x,z)$ entry we write 1 iff $\langle x, \langle M, z \rangle\rangle \in Q_n$. We say that a column $z$ is *spoiled* if for some $x$ the $(x,z)$ entry is 1. Since each query in $Q_n$ can spoil at most one column, the number of spoiled columns is at most $|Q_n|$. We can bound $|Q_n|$ as

$$\begin{aligned} |Q_n| &\leq 2^{n+1} \cdot 2^{g(n)} \cdot 2^{dn} \\ &\leq 2^{(d+2)n+g(n)}, \end{aligned}$$

which follows from the fact that there are $2^{n+1}$ and $2^{g(n)}$ choices for $x$ and $y$, respectively, and for each pair there are at most $2^{dn}$ queries. Setting $a = d+3$ this means that the number of unspoiled columns is at least $2^{g(n)+an} - 2^{(d+2)n+g(n)} \geq 2^{g(n)}$. Now we can choose $2^{g(n)}$ unspoiled columns and the strings that correspond to these columns satisfy (*) and we are done.

It follows from the construction that

$$M^A(x,y) = 1 \Leftrightarrow \langle x, \langle M, z_y \rangle\rangle \in A \tag{3.1}$$

We now show that this establishes the result: take any $L \subseteq \mathsf{DTIME}^A(2^{dn})/g(n)$ witnessed by $M$ and advice sequence $\{\alpha_n\}_{n\in\mathbb{N}}$. Now take a new advice sequence $\{\beta_n\}_{n\in\mathbb{N}}$ such that $\beta_n = \langle M, z_{\alpha_n}\rangle$. By 3.1 we can just query whether $\langle x, \beta_{|x|}\rangle$ is in $A$. This takes linear time and as $\beta_n$'s have the required size, the result follows.

□

Theorem 3.10 implies for any function $f : \mathbb{N} \to \mathbb{N}$ there is some oracle $A$ with respect to which if $x$ is describable in almost $f(|x|)$ bits then for all polynomial time bounds it has a description of size roughly $f(|x|)$. This means that if we want to get descriptions with strictly less than $f(|x|)$ bit we need at least exponential time. This is the opposite of what we wanted to show in the compression hierarchy problem. Therefore a positive anwer to this problem requires non-relativizing techniques, a strong evidence that it is indeed a hard problem.

Note that the oracle made in Theorem 3.10 depends on the function $f$ which means that it is possible that in any relativized world there is some hierarchy with polynomial gap for some function $f$ while this is not so for another function $f'$. Now the question is if there exists an oracle that works for every function.

**Problem (Buhrman).** Does there exist an oracle $A$ such that for all $f : \mathbb{N} \to \mathbb{N}$ with $f(n) \leq n$ for all $n$ it holds that $\exists c \forall bd \exists a$ such that

$$\mathsf{DTIME}^A(2^{dn})/f(2^n) + bn \subseteq \mathsf{DTIME}^A(2^{cn})/f(2^n) + an.$$

An affirmative answer to this question implies that a proof of the existence of a function for which a hierarchy with polynomial gap holds would require non-relativizing techniques. The method of Theorem 3.10 does not seem to be applicable here. Recall that to each string $y$ we associated a distinct string $z_y$. If we want to use the same argument here, we will have to find a distinct $z$ for each function $f$ and each string $y$. But since there are too many $f$'s we cannot use the counting argument to show that there exist such distinct $z$'s. We should thus look for other techniques to construct such an oracle.

# Chapter 4

# Future Work

We introduced the time-bounded Kolmogorov complexity and studied the hierarchy given by this definition. We investigated whether it is possible to prove hierarchies with polynomial gaps and that it relates to circuit lower bounds and hence derandomization.

For a possible further attack on the compression hierarchy problem, we now argue that using results from additive combinatorics (sum-product theorem in particular) it might be possible to get hierarchy results. As we will explain below additive combinatorics provides a nice setting for proving the existence of strings with high time-bounded Kolmogorov complexity that are somehow made of strings with low time-bounded Kolmogorov complexity. Using this property we might be able to generate those strings with fewer bits.

Let $\mathbb{F}$ be a field and let $A$ be a subset of $\mathbb{F}$. Consider the sets $A+A = \{a+b : a, b \in A\}$ and $A \times A = \{a \times b : a, b \in A\}$. We ask how large could these two sets be compared to the cardinality of $A$. Erdős and Szemerédi proved that at least one of them is large if $\mathbb{F} = \mathbb{R}$.

**Theorem 4.1. (Erdős, Szemerédi [ES83])** *For $\mathbb{F} = \mathbb{R}$ there exists $\epsilon > 0$ such that for all $A \subset \mathbb{F}$ we have* $\max\{|A+A|, |A \times A|\} \geq |A|^{1+\epsilon}$.

We now ask if some similar statement holds for finite fields. Notice that if $|A| \geq \Omega(|\mathbb{F}|)$ or if $A$ is a strict subfield of $A$ the statement is not true. To rule out these possibilities we assume that $A$ is not large and that $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ for some prime $p$ which guarantees that it does not have any strict subfield. We then have the following theorem.

**Theorem 4.2. (Bourgain, Katz and Tao [BKT04])** *For $\mathbb{F} = \mathbb{F}_p$ and every $A \subset \mathbb{F}$ with $|A| \leq |F|^{0.9}$ we have* $\max\{|A+A|, |A \times A|\} \geq |A|^{1+0.001}$.

This theorem has been used to construct randomness extractors in [BKS$^+$05] and also in later works. Exctractors could in some sense be considered as close cousins of procedures that increase Kolmogorov complexity (which is what we wanted to do). It is interesting to see if one can obtain any strong result in this respect using Theorem 4.2 or other tools in additive combinatorics. The following is the most naive approach. Let $n = 2^p - 1$ be some Mersenne prime and consider $\mathbb{F}_n$ and the set $A = C[f,t]^{=p} \setminus \{1^p\}$ for some $f$ and $t$. We will look at binary strings of length $p$ as member of $\mathbb{F}_n$. By Theorem 4.2 we have either $|A + A| > |A|$ or $|A \times A| > |A|$. Assume that the former is true. This means that there are $x$ and $y$ in $A$ such that $x + y \notin A$. Since $|A + A|$ is large enough we can also assume that $x + y \neq 1^p$. Thus for $z = x + y$ we have $C^t(z) > f(p)$, but $x + y$ is a description of $z$ of length $2f(p)$ running in polynomial time in $f$. This bound is far from what we wanted to prove, but it is interesting to see if more subtle applications of the sum-product theorem would give any better bound.

# Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity : A Modern Approach*. Cambridge University Press, 2009.

[ABFS11] Luís Antunes, Harry Buhrman, Lance Fortnow, and André Souto. personal communication, 2011.

[Bar02] Boaz Barak. A probabilistic-time hierarchy theorem for slightly non-uniform algorithms. In Jos Rolim and Salil Vadhan, editors, *Randomization and Approximation Techniques in Computer Science*, volume 2483 of *Lecture Notes in Computer Science*, pages 952–953. Springer Berlin / Heidelberg, 2002.

[BF11] Harry Buhrman and Lance Fortnow. personal communication, 2011.

[BGS75] Theodore Baker, John Gill, and Robert Solovay. Relativizations of the $\mathsf{P} =? \mathsf{NP}$ question. *SIAM Journal on Computing*, 4(4):431–442, 1975.

[BKS$^+$05] Boaz Barak, Guy Kindler, Ronen Shaltiel, Benny Sudakov, and Avi Wigderson. Simulating independence: new constructions of condensers, ramsey graphs, dispersers, and extractors. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, STOC '05, pages 1–10, New York, NY, USA, 2005. ACM.

[BKT04] Jean Bourgain, Nets Katz, and Terence Tao. A sum-product estimate in finite fields, and applications. *Geometric And Functional Analysis*, 14:27–57, 2004. 10.1007/s00039-004-0451-1.

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[Cle93] Carol E. Cleland. Is the church-turing thesis true? *Minds and Machines*, 3:283–312, 1993. 10.1007/BF00976283.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[ES83] P. Erdős and E. Szemerédi. *Studies in Pure Mathematics: To the Memory of Paul Turán*, chapter On Sums and Products of Integers, pages 213–218. Birkhuser, 1983.

[FS04] L. Fortnow and R. Santhanam. Hierarchy theorems for probabilistic polynomial time. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 316 – 324, oct. 2004.

[Gol01] Oded Goldreich. *Foundations of Cryptography*, volume 1. Cambridge University Press, 2001.

[Gol08] Oded Goldreich. *Computational Complexity: A Conceptual Perspective.* Cambridge University Press, 2008.

[Har83] Juris Hartmanis. Generalized kolmogorov complexity and the structure of feasible computations. In *Proc. 24th IEEE Symp. Foundations of Computer Science*, pages 439–445, 1983.

[HS65] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:pp. 285–306, 1965.

[HS66] F. C. Hennie and R. E. Stearns. Two-tape simulation of multitape turing machines. *J. ACM*, 13:533–546, October 1966.

[IW97] Russell Impagliazzo and Avi Wigderson. $\mathsf{P} = \mathsf{BPP}$ if $\mathsf{E}$ requires exponential circuits: derandomizing the XOR lemma. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 220–229, New York, NY, USA, 1997. ACM.

[KI04] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13:1–46, 2004. 10.1007/s00037-004-0182-6.

[KL80] Richard M. Karp and Richard J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the twelfth annual ACM symposium on Theory of computing*, STOC '80, pages 302–309, New York, NY, USA, 1980. ACM.

[Lev73] L. A. Levin. Universal sequential search problems. *Probl. Peredachi Inf.*, 9:265–266, 1973.

[Lon86] Luc Longpré. *Resource Bounded Kolmogorov Complexity, A Link Between Computational Complexity and Information Theory.* PhD thesis, Cornell University, 1986.

[LV08] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications.* Springer, 3rd edition, 2008.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149 – 167, 1994.

[SSZ95] Michael Saks, Aravind Srinivasan, and Shiyu Zhou. Explicit dispersers with polylog degree. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, STOC '95, pages 479–488, New York, NY, USA, 1995. ACM.

[SU05] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *J. ACM*, 52:172–216, March 2005.

[Wil85] Christopher B. Wilson. Relativized circuit complexity. *Journal of Computer and System Sciences*, 31(2):169 – 181, 1985.

[Yao82] A. C. Yao. Theory and application of trapdoor functions. In *IEEE Symposium on Foundations of Computer Science*, 1982.