Randomness Through the Lens of Polynomials and Compression

Dissertation for
habilitation à diriger des recherches

by Sylvain Perifel

Université Paris Cité

December 4, 2023

Il n'y a pas de hasard,
il n'y a que des rendez-vous

("There are no coincidences,
only appointments")

Attributed to Paul Éluard
(1895-1952)

Acknowledgements

I wholeheartedly thank all the members of the jury for accepting to assess this work, Claire Mathieu, Damiano Mazza, Olivier Carton and in particular Meena Mahajan, Santiago Figueira and Olivier Bournez for moreover reviewing this manuscript.

Research is not a lonely process, and my thanks also go to all my co-authors and colleagues for their dedication to science. Without them the results presented here would not have seen the light of day.

Particular thoughts go to Olivier Carton who encouraged and coached me through this journey of writing and defending my *habilitation à diriger des recherches*, and to Thomas Colcombet who rightfully insisted on using his wonderful `knowledge` LaTeX package for this document (if you can see blue links everywhere that is because of him).

To my son.

Table of contents

Introduction

I have tried to write a consistent story based on some highlights taken from my results since my PhD thesis. The story begins with a question: what is randomness, and tries to give some answers from different perspectives.

A first perspective dates back to concepts from theory of information: according to Martin-Löf and others, a sequence is random if no part of it can be predicted by an algorithm. In this document we shall adopt a point of view coming from computational complexity: hence the predicting algorithm will be required to be "efficient".

But an efficient algorithm that predicts parts of a sequence can *compress* it at least a little. Thus a sequence may be considered random if it cannot be compressed by an efficient algorithm. This is our first answer and the object of the first chapter, where we will study and compare classical compression algorithms.

Of course, this study is not only for the beauty of defining randomness. Compression algorithms have wide applications in practice and it is essential to understand their abilities. They also have links with dimension of complexity classes among others, as we will see.

The other perspective I wanted to develop is more about *using* randomness. With a central motivation: does it speed algorithms up, or on the contrary is it always possible do get rid of random bits and obtain deterministic algorithms that run as fast as probabilistic ones? The latter option is called derandomisation. This question is particularly interesting for one of the few natural problems that are not derandomised yet, namely polynomial identity testing.

Studying this problem goes through understanding what kind of polynomials can efficiently be computed by arithmetic circuits. This is the goal of algebraic complexity, which has enjoyed an increase of interest over the last fifteen years. This is due in part to a general strategy, called *geometric complexity theory*, to tackle the main open question of complexity, P versus NP, via algebraic means (see the survey [Bür+11]). One of its intermediate goals is to prove that computing the permanent cannot be efficiently reduced to computing the determinant. This question can naturally be seen as an analogue of P versus NP when using arithmetic circuits as model of computation.

Interest in arithmetic circuits was also sparked by a string of applications of a measure based on *partial derivatives* that allowed to prove encouraging lower bounds (see the surveys [SY10b; CKW11]). Lower bounds are actually deeply connected to derandomisation, and these two topics are the objects of our second chapter.

The purpose of the present document is not to give all the details of the theorems and the proofs: the interested reader is suggested to read the original articles for technicalities. Instead, the goal is to look at these different statements and proof techniques with hindsight, and to tell the story of selected results and domains that might seems far apart but finally draw together a consistant picture. Thus I have chosen to write a high-level review, but to include nevertheless some ideas of the proofs, since it is part of the journey and without these ideas the significance of the results would seriously decrease.

Chapter 1

Randomness Through the Lens of Compression

In order to be *random*, a sequence has to be somehow *impredictible*. But of course this notion depends on the power of the "predictor", thus a variety of randomness notions have emerged. Instead of impredictibility, we can also think of *compression*, since predicting the bits of a sequence enables to compress it. In other words, different notions of randomness arise from different classes of compressors. And comparing these notions amounts to comparing the compressors. This is what we shall do in Section 1.1 for three resource-limited compressors, namely pushdown transducers, polylog-space compression, and the Lempel-Ziv algorithm (LZ'78).

Then, as another way to understand its power, we shall focus on LZ'78 to study its *robustness* to small perturbations, and solve the so-called "one-bit catastrophe" question (Section 1.2).

Finally, our perspectives (Section 1.3) will center on a particular type of (arguably) *pseudo-random* sequences, namely normal sequences: what resources do they need to be compressed? And what about the normality of the famous Ehrenfeucht-Mycielski sequence?

Randomness, Compression, Dimension

As mentioned above, compression can be used to define notions of randomness. Martin-Löf randomness, for instance, can be defined in terms of Kolmogorov complexity: an infinite sequence u is random iff there exists no algorithm that compresses its prefixes. In symbols:

$$\exists c > 0, \forall n, K(u[0..n-1]) > n - c.$$

Here, the compressors need only be computable and have no further restriction (i.e. no resource bounds).

But unbounded compressors do not allow to discriminate between computable sequences: all computable sequences are indeed fully compressible. Thus, other complexity measures have emerged, like for instance Lempel-Ziv complexity that measures the "complexity" of a sequence in a similar way as Ziv and Lempel algorithm LZ'78 works and that we shall present later in this chapter.

As a wide field of application of these considerations, Lutz [Lut03] proposed the notion of *effective Hausdorff dimension* to study the structure of complexity classes. It has subsequently been developed into a rich theory by him and others (see, e.g., [LM13] for the links with compression and the Lempel-Ziv algorithm). Here, computation resource bounds are imposed on the compressor (or more or less equivalently on the "betting strategies") in order to define a fractal dimension adapted to (decidable) complexity classes.

We see that (resource bounded) compressors are central in this picture, and that is what we propose to study in the following.

Why Resource-Limited Compressors?

In what follows, we will focus on compressors with very limited resources (like polylog space computation or pushdown automata). As we said, this is necessary so as to hope capture the dimension of small complexity classes for instance. But this is obviously not the only reason. On the practical side, the compression algorithms need to be efficient so that it does not take five weeks to compress your latest video. This is all the more true that data are always more massive and, as of 2022, each year dozens of zettabytes (10^{21} bytes) are produced worldwide. That's why inefficient compression algorithm is not an option (has it ever been?).

Some other restrictions, such as pushdown compression or polylog-space compression that we will study below, are also motivated by the format of data: a stack may prove useful to parse and compress XML-like document, while small space is necessary for data streams for example.

In this perspective, the present chapter will focus on three efficient compressors: Lempel-Ziv algorithm LZ'78, a general-purpose lossless compressor; pushdown transducers that may use a stack but otherwise have only finite memory; and polylog-space compressors that prove useful in the streaming model. So as to understand the power, strengths and weaknesses of these compressors and which sequences they can indeed compress, in the next two sections we will:

- pairwise compare these algorithms by exhibiting sequences compressible by one but not the other;

- focus on Lempel-Ziv algorithm (LZ'78) and study its robustness to small perturbations.

Most of the results of this chapter come from [MMP11] (Propositions 2-7 below) and [LP18] (Section 1.2), the details can be found in these papers.

1.1 Three Resource-Limited Compressors

1.1.1 Lempel-Ziv and Automata

Lempel-Ziv Algorithm

Introduction LZ'78 is a generic lossless compressor, that is, an algorithm to compress files of any type without losing information. In [ZL78] where it is introduced, Ziv and Lempel compare its performance to finite-state lossless compressors and show it achieves the best possible compression ratio, as we will do in Section 1.1.1 below. Together with its cousin algorithm LZ'77 [ZL77], they have paved the way to many dictionary coders, some of them still widely used in practice today. For instance, the `deflate` algorithm at the heart of the open source compression program `gzip` uses a combination of LZ'77 and Huffman coding; or the image format GIF is based on a version of LZ'78. As another example, methods for efficient access to large compressed data on internet based on Ziv-Lempel algorithms have been proposed [HPZ11].

Besides its pratical interest, the algorithm LZ'78 was the starting point of a long line of theoretical research, triggered by the aforementioned optimality result among finite-state compressors. In particular, we will compare it with pushdown finite-state compressors in Section 1.1.2. And for instance in other recent works, the article [Kär+17] studies Lempel-Ziv and Lyndon factorisations of words; or the efficient construction of absolutely normal numbers of [LM21] makes use of the Lempel-Ziv parsing. Some works of bioinformatics have also focussed on Ziv-Lempel algorithms, since their compression scheme makes use of repetitions in a sequence in a way that proves useful to study DNA sequences (see e.g. [Zha+09]), or to measure the complexity of a discrete signal [Abo+06] for instance.

For convenience, often in what follows we shall merely write LZ to refer to the LZ'78 algorithm.

The algorithm Let us now describe this general purpose compression algorithm *LZ'78*. "Files" to be compressed will be words u over an alphabet Σ. The idea is to take advantage of repetitions in u in order to build a compressed representation[1]

[1]Of course, only some sequences will have a shorter representation since it is impossible to compress all sequences.

of u. LZ splits u in many blocks $u = u_1 \cdots u_k$ so that u_i is the extension of a previous block u_j by only one letter ($j < i$, $u_i = u_j a$ for some $a \in \Sigma$). Thus we can encode u_i by giving j in binary and the letter a. If the block sizes keep growing, j will not be too large: j in binary will take much less bits than u_i and we will save space. Consider the following example over the alphabet $\Sigma = \{a, b\}$: the word $u = aaababbabaaaab$ is parsed as

Blocks	a	aa	b	ab	ba	baa	aab
Block number	0	1	2	3	4	5	6
Extension of block #	ε	0	ε	0	2	4	1
by letter	a	a	b	b	a	a	b

and thus is encoded as

$$(\varepsilon, a); (0, a); (\varepsilon, b); (0, b); (2, a); (4, a); (1, b).$$

Thus LZ parses a word u into k blocks $u_1, \dots, u_k$ that constitute the *dictionary* (in the example, $\text{Dict}(u) = \{a, b, aa, ab, ba, aab, baa\}$). The *LZ-compression* of u is the ordered list of k pairs (p_i, a_i), where p_i is the binary representation of the unique integer $j < i$ such that $u_j = u_i[0..(|u_i| - 2)]$, and a_i the last letter of u_i (that is, the letter such that $u_i = u_j a_i$). Of course when the LZ-compression is given, the word u can easily be reconstructed.

Remark 1. • *If x is a word, we define* Pref(x) *the concatenation of all its prefixes in ascending order, that is,*

$$\text{Pref}(x) = x_0.x_0x_1.x_0x_1x_2.\cdots.x_0\cdots x_{n-2}x_{n-1}.$$

Then the parsing of the word $u = \text{Pref}(x)$ is exactly the prefixes of x, thus the size of the blocks increases each time by one: this is the optimal compression. In that case, the number of blocks is

$$k = |x| = \sqrt{2}\sqrt{|u|} - O(1).$$

Actually, it is easy to see that this optimal compression is attained only for the words u of the form Pref(x).

• *On the other hand, if u is the concatenation, in length-lexicographic order, of all words of size $\leq n$ ($u = a.b.aa.ab.ba.bb.aaa.aab\dots$), then it has size*

$$|u| = \sum_{i=1}^{n} i2^i = (n-1)2^{n+1} + 2,$$

and its parsing consists of all the words up to size n, therefore that is the worst possible case and the number of blocks is

$$k = 2^{n+1} - 2 = \frac{|u|}{\log |u|} + O\left(\frac{|u|}{\log^2 |u|}\right).$$

(And that is clearly not the only word achieving this worst compression.)

The *number of bits* needed in the LZ-compression is

$$\sum_{i=1}^{k} (|p_i| + 1) + O(k) = k \log k + O(k),$$

where k is the number of blocks. As the two previous extremal cases show (Remark 1),

$$k \log k = \Omega(\sqrt{|w|} \log |w|) \quad \text{and} \quad k \log k = O(|w|).$$

Definition 1. *The* compression ratio *of a word u is*

$$\rho^{LZ}(u) = \frac{\#\mathrm{Dict}(u) \log \#\mathrm{Dict}(u)}{|u|}.$$

Again, as Remark 1 shows,

$$\rho^{LZ}(u) = \Omega\left(\frac{\log |u|}{\sqrt{|u|}}\right) \quad \text{and} \quad \rho^{LZ}(u) \leq 1 + O\left(\frac{1}{\log |u|}\right).$$

A sequence of words (u_n) is said LZ-compressible if $\rho^{LZ}(u_n)$ tends to zero, i.e.,

$$k_n \log k_n = o(|u_n|),$$

and consistently it will be considered LZ-incompressible if $\liminf_{n\to\infty} \rho^{LZ}(u_n) > 0$, or in other terms,

$$k_n \log k_n = \Omega(|u_n|).$$

Actually, the $(\log k)$ factor is not essential in the analysis of the algorithm, therefore we shall drop it in what follows (moreover, most of the time we will focus directly on the size of the dictionary rather than the compression ratio).

Definition 2. *The* size of the LZ-compression *of u (or* compression size, *or also* compression speed *when speaking of a sequence of words) is defined as the size of* Dict(u), *that is, the number of blocks in the LZ-parsing of u.*

Recall (Remark 1) that

$$\#\mathrm{Dict}(u) = \Omega(\sqrt{|u|}) \quad \text{and} \quad \#\mathrm{Dict}(u) = O(|u|/\log(|u|)).$$

We can now restate the definition of incompressibility of a sequence of words in terms of compression speed instead of the number of bits in the LZ-compression.

Definition 3. *A sequence of words (u_n) is said incompressible iff*

$$\#\mathrm{Dict}(u_n) = \Theta\left(\frac{|u_n|}{\log(|u_n|)}\right).$$

In those definitions, we have to speak of sequences of finite words since the asymptotic behaviour is considered. That is not needed anymore for infinite words, of course, but then two notions of compression ratio are defined, depending on whether we take the lim inf or lim sup of the compression ratios of the prefixes. In this document, for simplicity we shall only consider one of them arbitrarily (the results in [MMP11] and [LP18] take into account both definitions in the strongest possible combination).

Definition 4 (LZ compression ratio for infinite words)**.** *If $u \in \{0,1\}^{\mathbb{N}}$ is an infinite word,*

$$\rho^{LZ}(u) = \liminf_{n\to\infty} \rho^{LZ}(u[0..n-1]).$$

An easy lemma We begin with a lemma relating the parsing of LZ to the number of distinct factors in a word. This lemma will be used later (for the proof of Propositions 2 and 4) but more importantly it is an illustration of how to bound the compression size of LZ on a given word.

Lemma 1. *Let m be an integer, Σ an alphabet and $u \in \Sigma^*$ a word. Let $M = \sqrt{2|u|/m} + \log_{|\Sigma|} m$. Suppose the number of distinct factors in u of each size $i \leq M$ is bounded by m. Then LZ parses u in at most $mM = \sqrt{2m|u|} + m\log_{|\Sigma|} m$ blocks.*

Proof. LZ parses u into pairwise distinct blocks. The largest number of blocks is obtained when the blocks are of minimal size. All the possible blocks of size $\leq M$ add up to a length

$$\sum_{i=0}^{M} \min(|\Sigma|^i, m) i \geq \sum_{i=\log_{|\Sigma|} m}^{M} mi \geq m(M^2 - \log^2_{|\Sigma|} m)/2 \geq |u|.$$

Thus the number of blocks in the LZ parsing is at most $mM = \sqrt{2m|u|} + m\log_{|\Sigma|} m$. □

LZ'78 Beats Finite-State Compressors

In their paper [ZL78], Ziv and Lempel showed their seminal result that their algorithm is asymptotically at least as good as finite-state compressors. This contributed to the reputation of LZ'78 algorithm. We include this result here as a natural first step to the next section. First we need to define finite-state compressors.

Definition 5 (informal)**.** *A (finite-state) transducer is a finite-state automaton with output words on each transition. While reading a word u, an output w is built step by step following the transitions as in the example of Figure 1.1. Thus, in addition to the closure of the transition function*

$$\delta^* : Q \times \Sigma^* \to Q$$

giving the last state $\delta^*(q_0, u) \in Q$ reached after reading a word u, a transducer T over an alphabet Σ also computes a (possibly partial) function $f_T : \Sigma^* \to \Sigma^*$.

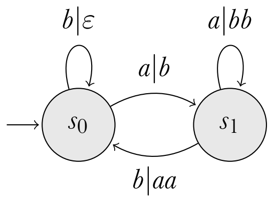


Figure 1.1: A transducer over the alphabet $\{a, b\}$. The image of the word *aabba* is the word $b.bb.aa.\varepsilon.b = bbbaab$.

Obviously, when a transducer is used to "compress" words, it should stall on no word and we would like to be able to recover the initial word from its (hopefully compressed) image and the last state reached: compressors must be total and injective.

Definition 6. *A transducer T is information lossless (IL) if the following function is total and injective:*

$$\begin{array}{rccc} f\colon & \Sigma^* & \to & \Sigma^* \times Q \\ & u & \mapsto & (f_T(u), \delta^*(q_0, u)). \end{array}$$

In that case we say that T is an information lossless finite-state transducer (ILFST).

We need now define in a natural way the size of the compression of an ILFST.

Definition 7. *The compression size of a word u by an ILFST T is merely the size of the output, i.e.,*

$$C^T(u) = |f_T(u)|.$$

The compression size of a word u by ILFST with s states is the shortest image obtained by an ILFST with s states, namely

$$C_s^{FS}(u) = \min_{T \text{ ILFST with } s \text{ states}} (C^T(u)).$$

The following proposition shows that ILFST do not compress well sequences that can be cut in many pairwise distinct blocks. Even though the proof comes from [ZL78], we include it here for the sake of completeness and because it illustrates the kind of techniques we can use for dealing with LZ parsings. From this result will immediately follow that LZ performs at least as well as ILFST.

Proposition 1 (Ziv, Lempel [ZL78])**.** *Suppose $u \in \Sigma^*$ is parsed into k* pairwise distinct *blocks $u = u_1 \dots u_k$. Then*

$$C_s^{FS}(u) \geq k(\log k - 2\log s - 2).$$

Proof. Fix an ILFST T with s states. Let b_i the number of blocks u_j whose output by T is of size i. Since the transducer is IL, two distinct blocks starting from the same state can output the same word only if they reach different states. Hence

$$b_i \leq s^2 2^i.$$

The best case for the compression is when the outputs are the smallest possible, that is, when there exists l such that $b_0, \dots, b_l$ are "full" ($b_i = s^2 2^i$ for $i \leq l$), and $b_{l+2}, \dots$ are "empty" ($b_i = 0$ for $i \geq l+2$). For convenience, let us call $b = b_{l+1}$, $0 \leq b < s^2 2^{l+1}$ (remark that $b \leq k/2$). In that case,

$$C_s^{FS}(u) = \sum i b_i = \sum_{i=0}^{l} i s^2 2^i + (l+1)b \geq s^2(l-1)2^{l+1} + (l+1)b. \tag{1.1}$$

But of course

$$k = \sum b_i = \sum_{i=0}^{l} s^2 2^i + b \leq s^2 2^{l+1} + b,$$

that is,

$$l \geq \log\left(\frac{k-b}{s^2}\right) - 1. \tag{1.2}$$

Combining Equations (1.1) and (1.2) we obtain

$$\begin{aligned} C_s^{FS}(u) &\geq 2b + k(\log(k-b) - 2\log s - 2) \\ &= k(2x + \log k + \log(1-x) - 2\log s - 2) \end{aligned}$$

where $x = b/k \in [0, 1/2]$. Since $2x + \log(1-x) \geq 0$ over $[0, 1/2]$, we obtain the desired result. $\square$

In other words, for all parsings of u into k pairwise distinct blocks, for fixed $s \in \mathbb{N}$ we have

$$C_s^{FS}(u) \geq k \log k - O(k).$$

Since LZ parses u in pairwise distinct blocks and the number of bits in the LZ-compression is $k \log k + O(k)$, the result follows.

Corollary 1. *Asymptotically, LZ performs at least as well as IL finite-state transducers.*

1.1.2 Pushdown Compression

Now that LZ performs better than finite-state transducers, the next natural question is whether it also outperforms pushdown transducers. In this section we show that it is not the case, as actually both compression methods are incomparable. But we first have to (informally) define what kind of pushdown transducers we use. We refer the reader to [MMP11] for the details of the definitions and the proofs.

Definition 8 (informal)**.** *A* pushdown transducer *(PDT) is a finite-state transducer equipped with a* stack*: transitions thus depend on the input symbol as well as the stack's top symbol. At each transition, the top symbol is removed and new symbols can be added to the stack. Our transducers will be* deterministic*, meaning that only one transition is possible whatever the combination of input and top stack symbols.*

The stack proves useful for compressing structured documents like, e.g., XML files: tags are indeed nested, and this nesting can be parsed thanks to the stack. It is for instance well known that the language

$$\{(\langle a\rangle)^n b(\langle /a\rangle)^n : n \in \mathbb{N}\}$$

over the alphabet $\{\langle, \rangle, /, a, b\}$ is not regular but can be recognized by a simple deterministic pushdown automaton that keeps pushing a symbol X on its stack while reading $\langle a\rangle$, and then pops X when reading $\langle /a\rangle$, thus ensuring that there are as many opening as closing tags.

During the run of a PDT, symbols are outputted on each transition: once concatenated, together with the last state reached they form the *image* of the transducer on the input word. For instance, consider the following transducer T with state set $Q = \{q_0, q_1\}$ (q_0 being the initial state), over input/output alphabet $\Sigma = \{a, b\}$ and stack alphabet $\Gamma = \{z_0, X\}$ (where z_0 is the bottom stack symbol):

- in state q_0:
  - if the input symbol is a, push X onto the stack (and output nothing, or more formally the empty word ε),

  - else output b and go to state q_1;

- in state q_1:
  - if the input symbol is b, then output b,
  - if the input symbol is a and the stack's top symbol is X, then output a and pop X,
  - otherwise stop.

Then the image of $a^m b^n a^k$, is

$$T(a^m b^n a^k) = \begin{cases} (b^n a^k, q_1) & \text{if } k \leq m \\ \text{undefined} & \text{otherwise} \end{cases}$$

Thus, a PDT can be seen as a (partial) function $T : \Sigma^* \to \Sigma^* \times Q$.

Of course, a "valid" compressor must be a total function and cannot send two distinct words on the same image, otherwise the image could not be decompressed.

Definition 9. *We say that a PDT T is information lossless (ILPDT) if the function $T : \Sigma^* \to \Sigma^* \times Q$ is total and injective. That way the input can be recovered ("decompressed") from the output and the final state. We shall write T_Σ for the projection of T on Σ^*, that is, we only keep the output word and not the final state.*

Note that our example is not IL for two reasons: it is not total on the one hand, and it is not injective on the other ($T(ab) = T(aab) = (b, q_1)$ for example).

Definition 10. *As for IL finite-state transducers, we naturally define the compression size of an ILPDT T on a word u as:*

$$C_T^{PD}(u) = |T_\Sigma(u)|.$$

We finally define the compression ratio as a measure of how well the word is compressed.

Definition 11. *The compression ratio of an ILPDT T on a finite word u is:*

$$\rho_T^{PD}(u) = \frac{C_T^{PD}(u)}{|u|}.$$

If u is infinite, we take the limit of the compression ratios of its prefixes:

$$\rho_T^{PD}(u) = \liminf_{n\to\infty} \rho_T^{PD}(u[0..n-1]).$$

Note that in the above definition we could take lim sup instead of lim inf. For simplicity we have arbitrarily taken the lim inf but both options are valid and in [MMP11] we always show our results with the least favourable of the two.

LZ vs Pushdown

Comparing ILPDT to LZ amounts to reviewing the strengths and weaknesses of each method.

- LZ compresses well *repeated factors*, even if they are not consecutive. For example, for integers n, s satisfying $\log n < s \leq \sqrt{n}$, if u is the concatenation of n words $x_{i_1}, \ldots, x_{i_n}$ among $\sqrt{2n/s}$ possible words $x_1, \ldots, x_{\sqrt{2n/s}}$ of size s, then u has at most $2n/s$ distinct factors of each length $i \leq s$ (since such a factor overlaps at most two consecutive x_j). Thus for $m = 3n/s$ and $M = \sqrt{2ns/(3n/s)} + \log(3n/s)$, we have $M < s$ and hence Lemma 1 teaches that u (of size ns) is parsed into at most $mM < ms = 3n$ blocks. Therefore u is highly compressible provided s is sufficiently large.

- But in the preceding construction the words x_j are arbitrary and can be chosen Kolmogorov random and independent. Then by carefully choosing their order $x_{i_1}, \ldots, x_{i_n}$, we can actually diagonalise over all ILPDT with $\log n$ states so that none of them can compress u at all.

This gives the following result that LZ can compress sequences that ILPDT cannot.

Proposition 2 ([MMP11, Thm 1])**.** *There is an infinite word u such that $\rho^{LZ}(u) = 0$ but $\rho^{PD}(u) = 1$.*

On the other hand, LZ also has weaknesses that ILPDT don't have.

- A *palindrome* $u\#\bar{u}$ can always be compressed at least to half its size ($C^{PD}(u\#\bar{u}) \leq |u|$ for any u) since we can use the stack to remember u and then check that the second part is indeed $\bar{u}$.

- There is no such guarantee for LZ. Suppose indeed that at some point in the parsing of a word, the dictionary contains exactly (all) the words of size $< k$. Now among the words of size k, consider the palindromes A_k and split the rest into B_k and $\bar{B}_k$ such that words in $\bar{B}_k$ are exactly the mirrors of words in B_k. Then the concatenation of B_k, then A_k and finally $\bar{B}_k$ (in the order symmetric to that of B_k), is a palindrome whose parsing by LZ adds to the dictionary exactly all words of size k. We can therefore build that way an infinite sequence made of palindromes that cannot be compressed by LZ.

Formally we obtain the following proposition.

Proposition 3 ([MMP11, Thm 5])**.** *For all $\varepsilon > 0$, there exists an infinite word u such that $\rho^{LZ}(u) > 1 - \varepsilon$ but $\rho^{PD}(u) \leq 1/2$.*

Propositions 2 and 3 together show that LZ and ILPDT are incomparable.

1.1.3 Polylog-Space Compressors

We can now do the same study for another class of resource-limited compressors, namely polylog-space compressors. Here we want to be able to compress a *large stream*: the input can only be read once from left to right, and so as to avoid running out of memory, memory usage is limited to a number of bits polylogarithmic in the length of the input. We shall call this class of algorithms *plogon transducers*.

Definition 12 ([HIM78]). *A plogon transducer is a Turing machine M with the following properties:*

- *M reads its input u from left to right (no turning back);*
- *$M(u)$ is given $|u|$ in binary on a special tape;*
- *$M(u)$ writes the output from left to right on a write-only tape;*
- *$M(u)$ uses memory bounded by $\log^c |u|$ for some constant c.*

As for finite-state transducers and pushdown transducers, our compressors must be information lossless.

Definition 13. *A plogon transducer M is information lossless if it computes a total and injective function. In that case we say that M is an information lossless plogon transducer (ILplog).*

And of course we can define the compression size and compression ratio of ILplog.

Definition 14. *The compression size of a finite word u by an ILplog M is*

$$C_M^{Plog}(u) = |M(u)|,$$

and its compression ratio is

$$\rho_M^{Plog}(u) = \frac{C_M^{Plog}(u)}{|u|}.$$

If u is an infinite word, we take the limit of the compression ratios of its prefixes:

$$\rho_M^{Plog}(u) = \liminf_{n\to\infty} \rho_M^{Plog}(u[0..n-1]).$$

The same remark as for LZ and ILPDT applies here on the arbitrary choice of lim inf instead of lim sup in the definition of the compression ratio of infinite words.

While having a polylog memory may seem rather powerful in comparison to PDT or even LZ, we sketch the ideas why it is actually incomparable with these two compressors.

LZ vs Plogon

- As we have already seen in Section 1.1.2, LZ compresses very well repetitions. Indeed, Lemma 1 implies that for $v = u^n$ for any word u of size n (let's say a Kolmogorov random one), $C^{LZ}(v) = O(n^{3/2}) = O(|v|^{3/4})$.

- But with only $\log^{O(1)} |v| = \log^{O(1)} n$ bits of memory, an ILplog L won't be able to compress v. Even more, it won't compress any of the copies of u inside v. Indeed, call c_i the configuration of L after processing u^{i-1}. Then c_i is of size $\log^\alpha n$ for some α. Suppose for contradiction that the output of L while processing the i-th copy of u is a word z_i of size $< n - 2\log^\alpha n$. If, on an input y of size n, L starting from c_i outputs z_i, then $y = u$ (otherwise the total output of L would be the same on $u^{i-1}y$ and on u^i, which is impossible since L is IL). Hence the following program of size $|z_i| + |c_i| < n - \log^\alpha n$ would describe the Kolmogorov random word u of size n, a contradiction: find by enumeration the (only) word y of size n on which L starting from c_i outputs z_i. Hence, no copy of u in v can be compressed to a word of size $< n - 2\log^\alpha n$.

This shows that LZ sometimes outperforms ILplog.

Proposition 4 ([MMP11, Thm 2])**.** *There exists an infinite word u such that $\rho^{LZ}(u) = 0$ but $\rho^{Plog}(u) = 1$.*

But the opposite is true as well.

- LZ does not compress the concatenation v of all words of size $\leq n$ in lexicographic order, because each block in the parsing is then exactly one of these words.

- But there is a simple ILplog that can compress this sequence: it can keep in memory the preceding word (of logarithmic size) and simply check that the current word is indeed its successor in the lexicographic order. If so, it outputs, say, 01, otherwise it outputs the current word with all its bits doubled, followed by 10. It is clearly injective and compresses v very well.

Thus ILplog can also outperform LZ.

Proposition 5 ([MMP11, Thm 6])**.** *There exists an infinite word u such that $\rho^{Plog}(u) = 0$ but $\rho^{LZ}(u) = 1$.*

Plogon vs Pushdown

Finally, we can compare ILPDT and ILplog.

- If u is a Kolmogorov random word of size n, then $C^{PD}(u\#\bar{u}) \leq n$, thus the compression ratio is at most $1/2$.

- But for the same reason as above (second item leading to Proposition 4), an ILplog cannot compress $u\#\bar{u}$.

Hence we obtain the following.

Proposition 6 ([MMP11, Thm 3]). *For all $\varepsilon > 0$ there exists an infinite word u such that $\rho^{PD}(u) \leq 1/2$ but $\rho^{Plog}(u) > 1 - \varepsilon$.*

And again we can do a last time the same exercise in the opposite direction.

- As in the second item leading to Proposition 2, out of n^2 Kolmogorov random words $x_1, \ldots, x_{n^2}$ of size n, we can choose 2^n indices $i_j \in [1, n^2]$ such that the sequence
$$v = x_1 \ldots x_{n^2} i_1 x_{i_1} \ldots i_{2^n} x_{i_{2^n}},$$
where the indices i_j are written in binary, diagonalises against all ILPDT with $\log n$ states, so that none can compress it.

- But this sequence is well compressed by an ILplog, since it is enough to keep in memory $x_1, \ldots, x_{n^2}$ (of size polylog compared to $|v|$) and to check whether the index i_j indeed corresponds to the word x_{i_j}.

This last result ends the comparison of our three resource-limited compressors.

Proposition 7 ([MMP11, Thm 7]). *There exists an infinite word u such that $\rho^{Plog}(u) = 0$ but $\rho^{PD}(u) = 1$.*

1.2 A One-Bit Catastrophe

1.2.1 The Question and an Overview of the Results

Suppose you compressed a file using your favorite compression algorithm, but you realize there were a typo that makes you add a single bit to the original file. Compress it again and you get a much larger compressed file, for a one-bit difference only between the original files. Most compression algorithms fortunately do not have this strange behaviour; but if your favorite compression algorithm is called LZ'78, one of the most famous and studied of them, then this surprising scenario might well happen… In rough terms, that is what we show in [LP18] and what we will sketch in this section, thus closing a question advertised by Jack Lutz under the name "one-bit catastrophe" and explicitly stated for instance in papers of Lathrop and Strauss [LS97], Pierce II and Shields [PS00], as well as more recently by López-Valdés [Lop06].

Actually, both in theory and in practice, Ziv-Lempel algorithms are undoubtedly among the most studied compression algorithms. Yet, the robustness of LZ'78 remained unclear: the question of whether the compression ratio of a sequence could vary by changing a single bit appears already in [LS97], where the authors also ask how LZ'78 will perform if a bit is added in front of an optimally compressible word. Since the Hausdorff dimension of complexity classes introduced by Lutz [Lut03] can be defined in terms of compression (see [LM13]), this question is linked to finite-state and polynomial-time dimensions as [Lop06] shows. As a practical illustration of the issue the (lack of) robustness can cause, let us mention that the `deflate` algorithm tries several starting points for its parsing in order to improve the compression ratio.

In this section we will show the existence of an infinite sequence w which is compressible by LZ'78, but the addition of a single bit (the alphabet now is $\{0, 1\}$) in front of it makes it incompressible (the compression ratio of $0w$ is non-zero, see Theorem 1), thus we settle the "one-bit catastrophe" question. To that end, we study the question over finite words, which enables stating more precise results. For a word w and a letter $a \in \{0, 1\}$, we first prove in Theorem 2 that the compression ratio $\rho^{LZ}(aw)$ of aw cannot deviate too much from the compression ratio $\rho^{LZ}(w)$ of w:

$$\rho^{LZ}(aw) \leq 3\sqrt{2}\sqrt{\rho^{LZ}(w) \log |w|}.$$

In particular, aw can only become incompressible ($\rho^{LZ}(aw) = \Theta(1)$) if w is already poorly compressible, namely $\rho^{LZ}(w) = \Omega(1/\log n)$. This explains why the one-bit catastrophe cannot be "a tragedy" as we point out in the title of [LP18].

However, our results are tight up to a constant factor, as we show in Theorem 4: there are constants $\alpha, \beta > 0$ such that, for any $l(n) \in [90^2 \log^2 n, \sqrt{n}]$, there are infinitely many words w satisfying

$$\rho^{LZ}(w) \leq \alpha \frac{\log |w|}{l(|w|)} \quad \text{whereas} \quad \rho^{LZ}(0w) \geq \beta \frac{\log |w|}{\sqrt{l(|w|)}}.$$

In particular, for $l(n) = 90^2 \log^2 n$, these words satisfy

$$\rho^{LZ}(w) \leq \frac{1}{\log |w|} \quad \text{and} \quad \rho^{LZ}(0w) \geq \frac{\beta}{90}$$

(this is the one-bit catastrophe over finite words). But actually the story ressembles much more a tragedy for well-compressible words. Indeed, for $l(n) = \sqrt{n}$ we obtain:

$$\rho^{LZ}(w) \leq \alpha \frac{\log |w|}{\sqrt{|w|}} \quad \text{whereas} \quad \rho^{LZ}(0w) \geq \beta \frac{\log |w|}{|w|^{1/4}},$$

that is to say that the compression ratio of $0w$ is much worse than that of w (which in that case is optimal).

This "catastrophe" shows that LZ'78 is not robust with respect to the addition or deletion of bits. Since a usual good behaviour of functions used in data representation is a kind of "continuity", our results show that, in this respect, LZ'78 is not a good choice, as two words that differ in a single bit can have images very far apart.

1.2.2 More Details

The one-bit catastrophe question is originally stated only on infinite words. It asks whether there exists an infinite word w whose compression ratio changes when a single letter is added in front of it. More specifically, a stronger version asks whether there exists an infinite word w compressible (compression ratio equal to 0) for which $0w$ is not compressible (compression ratio > 0). At Section 1.2.6 we will answer that question positively:

Theorem 1. *There exists $w \in \{0,1\}^{\mathbb{N}}$ such that*

$$\rho^{LZ}(w) = 0 \quad \textit{and} \quad \rho^{LZ}(0w) \geq \frac{1}{6\,075}.$$

But before proving this result, most of the work will be on finite words (only in Section 1.2.6 will we show how to turn to infinite words). Let us therefore state the corresponding results on finite words. Actually, on finite words we can have much more precise statements and therefore the results are interesting on their own (perhaps even more so than the infinite version).

In the following sections we give the ideas of the proofs (the complete proofs can be found in [LP18]). In Section 1.2.3, we show the easy direction: the compression ratio of aw cannot be much more than that of w. In particular, all words "sufficiently" compressible (compression speed $o(|w|/\log^2|w|)$) cannot become incompressible when a letter is added in front (in some sense, thus, the one-bit catastrophe cannot happen for those words, see Remark 3).

Theorem 2. *For all word $w \in \{0,1\}^{\star}$ and any letter $a \in \{0,1\}$,*

$$\#\mathrm{Dict}(aw) \leq 3\sqrt{|w|.\#\mathrm{Dict}(w)}.$$

Remark 2. *When stated in terms of compression ratio, using the fact that* $\#\mathrm{Dict}(w) \geq \sqrt{|w|}$*, this result reads as follows:*

$$\rho^{LZ}(aw) \leq 3\sqrt{2}\sqrt{\rho^{LZ}(w) \log |w|}.$$

Then in Section 1.2.4 we show that this result is tight up to a multiplicative constant, since Theorem 5 implies the following result.

Theorem 3. *For an infinite number of words* $w \in \{0, 1\}^\star$*,*

$$\#\mathrm{Dict}(0w) \geq \frac{1}{35}\sqrt{|w|.\#\mathrm{Dict}(w)}.$$

This is actually a warm-up for the more general result proved in Section 1.2.5:

Theorem 4. *Let* $l : \mathbb{N} \to \mathbb{N}$ *be a function satisfying* $l(n) \in [(90 \log n)^2, \sqrt{n}]$*. Then for an infinite number of words w:*

$$\#\mathrm{Dict}(w) \leq \frac{3+\sqrt{3}}{2} \cdot \frac{|w|}{l(|w|)} \text{ and } \#\mathrm{Dict}(0w) \geq \frac{1}{54} \cdot \frac{|w|}{\sqrt{l(|w|)}}.$$

This shows that the upper bound is tight (up to a multiplicative constant) for any possible compression speed. This also provides an example of compressible words that become incompressible when a letter is added in front (see Remark 3), thus showing the one-bit catastrophe for finite words.

Remark 3. *In particular, the following three cases are of interest (the last two being the two "extremal" applications of Theorem 4):*

- *Theorem 2 implies that, if an increasing sequence of words* (w_n) *satisfies*

  $$\#\mathrm{Dict}(w_n) = o(|w_n|/\log^2 |w_n|),$$

  then for any letter $a \in \{0, 1\}$*,* aw_n *remains fully compressible:*

  $$\#\mathrm{Dict}(aw_n) = o(|w_n|/\log |w_n|).$$

- *however, by Theorem 4, there is an increasing sequence of words* (w_n) *such that*

  $$\#\mathrm{Dict}(w_n) = \Theta(|w_n|/\log^2 |w_n|) \text{ (compressible)}$$

  but

  $$\#\mathrm{Dict}(0w_n) = \Theta(|w_n|/\log |w_n|) \text{ (incompressible)},$$

  which is the one-bit catastrophe on finite words;

- *the following interesting case is also true: there is an increasing sequence of words* (w_n) *such that*
$$\#\mathrm{Dict}(w_n) = \Theta(\sqrt{|w_n|}) \text{ (optimal compression)}$$
but
$$\#\mathrm{Dict}(0w_n) = \Theta(|w_n|^{3/4}).$$
This special case is treated extensively in Theorem 5.

In what follows we will often compare the parsing of a word w and the parsing of aw for some letter a: let us introduce some notations (see Figure 1.2).

- The blocks of w will be called the *green blocks*.
- The blocks of aw will be called the *red blocks* and are split into two categories[2]:
  - The *junction blocks*, which are red blocks that overlap two or more green blocks when we align w and aw on the right (that is, the factor w of aw is aligned with the word w, see Figure 1.2).
  - The *offset-i blocks*, starting at position i in a green block and completely included in it. If not needed, the parameter i will be omitted.

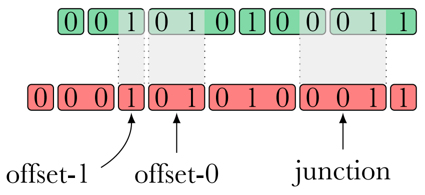


Figure 1.2: The green blocks of w and red blocks of $0w$ for $w = 001010100011$.

1.2.3 Upper Bound

In order to show Theorem 2, we need two simple lemmas. The first is obtained by examining the parsing tree of the word.

Lemma 2. *For any i, the number of different words of size i that are factors of some blocks of the LZ-parsing of a word w is at most* $\#\mathrm{Dict}(w)$.

The second lemma is obtained by counting in a similar way as in Lemma 1. It claims that, in an LZ-parsing, if there are few different blocks of same size, then the size of the blocks must grow fast hence the total number of blocks is small.

[2]Except the first block of aw, which is the word a and which is just called a red block.

Lemma 3. *Let $\mathcal{F}$ be a family of distinct words such that for each i, the number of words of size i in $\mathcal{F}$ is bounded by a constant N. Suppose that a word w is partitioned into distinct words from $\mathcal{F}$. Then the number of words used in the partition is at most $2\sqrt{N|w|}$.*

These lemmas enable to show Theorem 2.

Proof of Theorem 2. Let $D = \mathrm{Dict}(aw)$ be the set of red blocks. We partition D into D_1 and D_2, where D_1 is the set of junction blocks together with the first red block (consisting only of the letter a), and D_2 is the set of offset blocks.

- Bound on D_1: The number of junction blocks is less than the number of green blocks, therefore

  $$\#D_1 \leq \#\mathrm{Dict}(w) \leq \sqrt{\#\mathrm{Dict}(w).|w|}$$

  (recall that $\#\mathrm{Dict}(w) \leq |w|$).

- Bound on D_2: Consider $\tilde{w}$ the word w where all the junction blocks have been replaced by the empty word ε. We know that $\tilde{w}$ is partitioned into distinct words by D_2. But $D_2 \subset \mathcal{F}$, where $\mathcal{F}$ is the set of factors of $\mathrm{Dict}(w)$ (the set of factors contained in the green blocks). By Lemma 2, the number of words of size i in $\mathcal{F}$ is bounded by $\#\mathrm{Dict}(w)$. Finally, Lemma 3 tells us that the number of words in any partition of $\tilde{w}$ by words of $\mathcal{F}$ is bounded by

  $$2\sqrt{\#\mathrm{Dict}(w).|\tilde{w}|} \leq 2\sqrt{\#\mathrm{Dict}(w).|w|}.$$

In the end, $\#D = \#D_1 + \#D_2 \leq 3\sqrt{|w|.\#\mathrm{Dict}(w)}$. □

1.2.4 Optimally Compressible Sequences

Before the proof of Theorem 4, we first present a "weak catastrophe", namely the third item of Remark 3 in which the compression speed of a sequence changes from $O(\sqrt{n})$ (optimal compression) to $\Omega(n^{3/4})$ when a letter is added in front, thus matching the upper bound of Theorem 2.

Theorem 5. *For an infinite number of words w:*

$$\#\mathrm{Dict}(w) \leq 1.9\sqrt{|w|} \text{ and } \#\mathrm{Dict}(0w) \geq 0.039|w|^{3/4}.$$

Observe that this weak catastrophe is a special case of Theorem 4 (with better constants, though). The aim of this section is twofold: first, it will be a constructive proof, whereas the main theorem will use the probabilistic method; second, this section will set up the main ideas and should help understand the general proof.

A main ingredient in the construction is de Bruijn sequences, that we introduce shortly before giving the overview of the proof.

De Bruijn Sequences

A *de Bruijn sequence* of order k (or $\mathrm{DB}(k)$ for short, notation that will also designate the set of all de Bruijn sequences of order k) is a word x of size $2^k + k - 1$ in which every word of size k occurs exactly once as a substring. For instance, `0001011100` is an example of a $\mathrm{DB}(3)$. Such words exist for any order k as they are, for instance, Eulerian paths in the regular directed graph whose vertices are words of size $(k-1)$ and where there is an arc labeled with letter a from u to v iff $v = u[1..k-2]a$.

Given any $x \in \mathrm{DB}(k)$, the following well-known (and straightforward) property holds:

$(\star)$ Any word u of size at most k occurs exactly $2^{k-|u|}$ times in x.

Thus, a factor of size $l \leq k$ in x will identify exactly 2^{k-l} positions in x (the i-th position is the beginning of the i-th occurence of the word).

The use of de Bruijn sequences is something common in the study of this kind of algorithms: Lempel and Ziv themselves use it in [LZ76], as well as later [LS97] and [PS00] for example.

Overview of the Proof

Recall that a word w is optimally compressed iff it is of the form $w = \mathrm{Pref}(x)$ for some word x (Remark 1). Thus we are looking for an x such that $0\mathrm{Pref}(x)$ has the worst possible compression ratio. In Section 1.2.3 the upper bound on the dictionary size came from the limitation on the number of possible factors of a given size: it is therefore natural to consider words x where the number of factors is maximal, that is, de Bruijn sequences.

Although we conjecture that the result should hold for $w = \mathrm{Pref}(x)$ whenever x is a de Bruijn sequence beginning with 0, we were not able to show it directly. Instead, we need to (possibly) add small words, that we will call "gadgets", between the prefixes of x.

For some arbitrary k, we fix $x \in \mathrm{DB}(k)$ and start with the word $w = \mathrm{Pref}(x)$ of size n. The goal is to show that there are $\Omega(n^{3/4})$ red blocks (i.e that the size of the dictionary for $0w$ is $\Omega(n^{3/4})$): this will be achieved by showing that a significant (constant) portion of the word $0w$ is covered by "small" red blocks (of size $O(n^{1/4})$). Let $s = |x|$, so that $n = \Theta(s^2)$. More precisely, we show that, in all the prefixes y of x of size $\geq 2s/3$, at least the last third of y is covered by red blocks of size $O(\sqrt{s}) = O(n^{1/4})$.

This is done by distinguishing between red blocks starting near the beginning of a green block (offset-i for $i \leq \gamma k$, where $\gamma \geq 3$ is a constant) and red blocks starting at position $i > \gamma k$:

- For the first, what could happen is that by coincidence the parsing creates most of the time an offset-i red block, which therefore would increase until it covers almost all the word w. To avoid this, we introduce gadgets: we make sure that this happens at most half of the time (and thus cannot cover more than half of w). More precisely, the insertion of gadgets "kills" some starting positions i if necessary, by "resynchronizing" the parsing at a different position, thus ensuring that at most half of the prefixes of x contain offset-i blocks for any fixed $i \leq \gamma k$.
- On the other hand, red blocks starting at position $i > \gamma k$ are shown to be of small size. This is due to the structure of the $\mathrm{DB}(k)$ (few repetitions of factors) which implies that few junction red blocks can go up to position $(i-1)$ and precede an offset-i block.

Since all large enough prefixes of x have a constant portion containing only red blocks of size $O(n^{1/4})$, the compression speed is $\Omega(n^{3/4})$ (Theorem 5).

Gadgets must satisfy two conditions:

- they must not disturb the parsing of w;
- the gadget g_i must "absorb" the end of the red block ending at position $(i-1)$, and ensures that the parsing restarts at a controlled position different from i.

The insertion of gadgets in w is not trivial because we need to "kill" positions without creating too many other bad positions, that is why gadgets are only inserted in the second half of w. Moreover, gadget insertion depends on the parsing of $0w$ and must therefore be adaptative, which is the reason why in [LP18] we give an algorithm to describe the word w.

1.2.5 General Case

The proof of Theorem 4 first goes through the existence of a family F of "independent" de Bruijn-style words which will play a role similar to the de Bruijn word x in the proof of Theorem 5. The existence of this family is shown using the probabilistic method: with high probability, a family of random words satisfies a relaxed version (P1) of the "local" property $(\star)$, together with a global property (P2) that forbids repetitions of large factors throughout the whole family.

The word w that we will consider is the concatenation of "chains" roughly equal to $\mathrm{Pref}(x)$ for all words $x \in F$, with gadgets inserted if necessary as in Section 1.2.4. (The construction is actually slightly more complicated because in each chain we must avoid the first few prefixes of x in order to synchronise the parsing of w; and the gadgets are also more complex.) Properties (P1) and (P2) guarantee that each of the chains of w are "independent", so that the same kind of argument as in Section 1.2.4 will apply individually. By choosing appropriately the number of chains and their length, we can obtain any compression speed for w up to $\Theta(n/\log^2 n)$ and the matching bound for $0w$ (see Theorem 4).

1.2.6 Infinite Words

The techniques on finite words developed in the preceding sections can almost be used as a black box to prove the one-bit catastrophe for infinite words (Theorem 1). Our aim is to design an infinite word $w \in \{0,1\}^{\mathbb{N}}$ for which the compression ratios of the prefixes tend to zero, whereas the compression ratios of the prefixes of $0w$ tend to $\epsilon > 0$. In Section 1.2.5, we concatenated the bricks obtained in Section 1.2.4; now, we concatenate an infinite number of bricks of Section 1.2.5 of increasing size (with the parameters that gave the one-bit catastrophe on finite words). As before, each chain of size l will be parsed in $\Theta(l)$ green blocks and $\Theta(l^{3/2})$ red blocks. To guarantee that the compression ratio always remains close to zero in w and never goes close to zero in $0w$, the size of the bricks mentioned above will be adjusted to grow neither too fast nor too slow, so that the compression speed will be locally the same everywhere.

The "base" word from which w will be constructed is of the following type.

Definition 15. *Given a sequence $\mathcal{F} = (F_i)_{i\geq 0}$ where each F_i is a family of words, we denote by $w_{\mathcal{F}}$ the word*

$$w_{\mathcal{F}} = \prod_{i=0}^{\infty} \prod_{x \in F_i} \mathrm{Pref}_{>q_x^i}(x)$$

where $\mathrm{Pref}_{>q}(x)$ denotes the concatenation of all prefixes of x starting from size $q+1$, and

$$q_x^i = \max\{a : x[0..a-1] \text{ is a prefix of a word in } \cup_{j<i} F_j\}.$$

For a particular sequence $\mathcal{F} = (F_i)$, the word w will be equal to $w_{\mathcal{F}}$ with some gadgets inserted between the prefixes as in the previous sections. The sequence $\mathcal{F}$ that we shall consider will be a sequence of families of random words which will satisfy two properties (generalisation of (P1) and (P2) above):

(P2') guarantees a kind of "independence" of the families $F_0, F_1, \ldots$;

(P1') is a de Bruijn-style "local" property on each word of each family F_i.

The existence of a sequence $\mathcal{F}$ satisfying these two properties is shown by a probabilistic method. This shows the one-bit catastrophe. In this exposition we have only sketched ideas and many details remain hidden: they are given in [LP18].

We now present some future work linked to the present chapter.

1.3 Perspective: Normality

When you cast a die a large number of times, any value i among $\{1, \dots, 6\}$ is expected to be obtained with frequency one sixth. More generally, for any $x_1, \dots, x_k \in \{1, \dots, 6\}$, the sequence of consecutive values $x_1 \dots x_k$ is expected to appear with frequency 6^{-k}.

This particular behaviour of random series is what Borel [Bor09] named "normality". More precisely, an infinite word u over an alphabet Σ is called *normal* if for any finite word $v \in \Sigma^*$, v appears as a factor in u with frequency $(\#\Sigma)^{-|v|}$. Such a word u thus behaves like a random word if we only consider the frequency of its factors.

Historically this notion concerned the expansion of real numbers in a given base, and has drawn a lot of attention in the last hundred years, but it is still unknown whether π, e or even $\sqrt{2}$ are normal numbers. Many characterisations have been proposed, see [BC18]. It was of course only natural to look at it in the broader context of arbitrary infinite words and not only real numbers, enabling in particular to make a bridge between this number-theoretic notion and automata theory, as we shall see.

1.3.1 Can Automata Compress Normal Sequences?

"True" random sequences cannot be compressed at all by algorithms. What about normal sequences? They indeed have a "random flavour": does it prevent them from being compressible?

One of the most well-known normal sequence is Champernowne number [Cha33], defined in base 10 (and accordingly in any other base) as:

$$0.1\,2\,3\,4\,5\,6\,7\,8\,9\,10\,11\,12\,13\,14\,15\,16\,17\,18\,19\,20\,21\dots99\,100\,101\dots$$

This sequence does not look exactly random and is obviously compressible since it has this simple description: "concatenate in ascending order all the natural numbers written in base 10".

But, as mentioned in the introduction, the notion of randomness actually depends on the power of the compressor. It can be shown that weak compressors such as finite-state transducers are not able to compress this sequence. This is in fact a more general result: it is a characterisation of normal sequences (the result follows from [SS72] and [Dai+04], see [BC18] for a survey and a proof).

Theorem 6. *An infinite word is normal iff it cannot be compressed by a finite-state transducer.*

In other words, if we define randomness as incompressibility, normal sequences are the random sequences for finite-state transducer, and therefore are among the most basic "random" sequences since the compressors involved in Theorem 6 are among the weakest.

What if our compressors had more power, would they then be able to compress normal sequences? The answer is *no* if we only allow nondeterminism: nondeterministic finite-state transducers can compress no normal sequences. But if moreover we allow for a stack, then the answer is *yes*: nondeterministic pushdown transducers can compress some normal sequences. Before Proposition 8 below, the state of our knowledge concerning the compressibility of normal sequences by various finite-state transducers was like in Table 1.1 (see [BCH15]).

Finite-state transducer	det.	non-det.	non-real-time
No extra memory	N	N	N
One counter	N	N	N
More than one counter	N	N	Y
One stack	**?**	Y	Y
One stack and one counter	Y	Y	Y

Table 1.1: Compressibility of normal sequences by different kinds of transducers (before this work).

In an ongoing work with Olivier Carton, we complete the missing result in this table. More precisely, we come up with a normal sequence that can be compressed by a deterministic pushdown transducer.

Proposition 8. *There exist a deterministic ILPDT T and an infinite word w such that:*

- *w is normal;*
- $\rho_T^{PD}(w) < 1$.

Indeed, it turns out that the naïve way to use the stack is enough to compress a palindromic variant of Champernowne sequence: roughly speaking, if the input letter and the top stack letter coincide, then pop and, only every two steps, output a particular symbol (thus keeping injectivity); if the input letter and the top stack letter do not coincide, then output and push the input letter (no compression occurs here). That way, consecutive pops are "shrunk" by half, and this is enough to compress a little when the input sequence contains a lot of large palindromes. This is in particular the case of the following sequence:

$$w = w_1\tilde{w}_1 w_2\tilde{w}_2 w_3\tilde{w}_3 \dots$$

where w_i is the concatenation of all words of size i, and $\tilde{w}_i$ is the mirror of w_i. This sequence is a variation on Champernowne's and happens to remain normal.

1.3.2 Ehrenfeucht-Mycielski Sequence

As Borel [Bor09] showed, almost all real numbers are normal (and even *absolutely normal*, meaning normal in any base). We can construct some absolutely normal numbers efficiently, in "nearly linear time", see [LM21]. Despite these rather encouraging facts, apart perhaps from (variations on) Champernowne sequence almost no "natural" normal number is known. Of course π, e and $\ln 2$ are good candidates but any proof seems elusive. It has furthermore been conjectured by Borel himself [Bor50] that irrational algebraic numbers are absolutely normal, but it is not even known for example whether the asymptotic frequency of zeroes and ones is $1/2$ in the binary expansion of $\sqrt{2}$ (i.e., whether $\sqrt{2}$ is *simply normal* in base 2).

Open question 1. *Is $\sqrt{2}$ simply normal in base 2?*

This question seems out of reach. But the quest for "natural" normal numbers does not restrict to famous mathematical constants. The following well-known sequence might be a simpler candidate, at least with a more combinatorial flavour. In particular, its definition reminds of the LZ parsings. It has been introduced in [EM92] and has drawn a lot of attention since then, due to its seemingly "pseudo-random" behaviour.

Definition 16. *The Ehrenfeucht-Mycielski sequence is the infinite word EM over the alphabet $\{0,1\}$ that starts with* 010 *and whose $(i+1)$-th bit x_i ($i > 2$) is defined iteratively as follows.*

Find the largest k such that the suffix $x_{i-k}\cdots x_{i-1}$ already appears as a factor in $x_0\cdots x_{i-2}$. Call $j < i-k$ the largest position where it appears ($x_j\cdots x_{j+k-1} = x_{i-k}\cdots x_{i-1}$).

Then $x_i = 1 - x_{j+k}$ (i.e. we flip the bit following the previous occurrence of the longest possible suffix). See Figure 1.3.

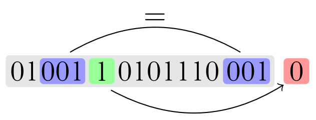


Figure 1.3: The beginning of *EM* and how to define the next bit.

This sequence seems to be somehow "pseudo-random", and it is conjectured to be normal, but so far only the following two weak evidences of this fact have been shown.

Proposition 9.
- *Every finite word over the alphabet $\{0,1\}$ appears infinitely many times as a factor of EM ([EM92]).*

- *If $M_n(1)$ denotes the number of ones in $EM[0..n-1]$, then*

$$\liminf \frac{M_n(1)}{n} \geq 1/4 \quad \textit{and} \quad \limsup \frac{M_n(1)}{n} \leq 3/4.$$

  In particular, the asymptotic frequencies of zeroes and ones cannot be too "unbalanced" and lie in the interval $[1/4, 3/4]$ ([KS07]).

Thanks to the similarity of *EM* with LZ parsing, the understanding gained from Section 1.2 on the one-bit catastrophe might be helpful to strengthen Kieffer and Szpankowski's result above (Proposition 9, second item) to $1/2$ instead of $[1/4, 3/4]$ and finally settle the following question known as the *balance conjecture*.

Open question 2. *Is EM simply normal in base 2?*

Chapter 2

Randomness Through the Lens of Algebraic Complexity

The link between randomness and polynomials is perhaps less obvious than with compression in the previous chapter. Nevertheless, on the one hand randomness plays a crucial rôle in computational complexity at large and in algebraic complexity in particular; and on the other hand, we shall focus on the problem of polynomial identity testing (PIT) with the underlying motivation of understanding whether randomness is necessary to solve it efficiently.

In Section 2.1 we shall review some of the results obtained in [FPV15] (Section 2.1.2 below), [Dvi+12] (Section 2.1.3), [LMP19] (Section 2.1.4) and [Bal+21] (Section 2.1.5) concerning PIT and lower bounds on the circuit complexity of polynomials, without going into much technical details. Then in Section 2.2 we shall focus on a problem that we think is worth being studied, namely the power of PIT.

Polynomial Identity Testing

In what follows, we shall encounter randomness used in three different ways. First in the proofs of course, with the use of, e.g., a probabilistic method in Section 2.1.3. Second in complexity classes like MA in Section 2.1.2. But the third is more important and more obvious: it will appear in algorithms for PIT and will be at the heart of many results in this chapter as well as of Section 2.2.

Actually this last usage of randomness is connected to the (arguably) main task in algebraic complexity and computational complexity at large, namely proving lower bounds. Consider indeed the following example. Suppose your task is to decide whether a polynomial $p(x)$ computed by an arithmetic circuit C is zero (this is the problem PIT). You might come up with the following (black box) algorithm evaluating p at well-chosen points:

- compute some integers $a_1, \ldots, a_k$;
- if $p(a_i) \neq 0$ for some i, then reject; otherwise accept.

If your algorithm works for all circuits of size s, this means that the polynomial

$$q(x) = \prod_{i=1}^{k}(x - a_i)$$

does not have circuits of size s, because the algorithm incorrectly accepts such a non-zero polynomial. Hence you have proved a lower bound.

This easy consideration is just the tip of the iceberg: the links between derandomisation and circuit lower bounds are indeed much deeper. A long line of works by Yao, Nisan, Wigderson, Sudan, Impagliazzo, Kabanets among others (see, e.g., the book [AB09] for an exposition) proved that (strong) lower bounds on Boolean circuits imply derandomisation, and conversely that derandomising PIT gives (weak) lower bounds.

That is why this chapter focusses on some aspects of lower bounds in algebraic complexity and of polynomial identity testing.

2.1 Lower Bounds and PIT

2.1.1 Arithmetic Circuits and PIT

Arithmetic Circuits and Valiant's Classes

We first need to define the basic concepts. As a basic model of computation, we use *arithmetic circuits* (see the survey [SY10b]): that is, directed acyclic graphs in which vertices of indegree zero are called *input gates* and are labeled by a variable x_i or a constant α_i from the underlying field $\mathbb{F}$; all the other vertices (*gates*) are of indegree two and labeled with $+$ or $\times$; and the unique vertex of outdegree zero is called the *output gate*. Each gate computes a polynomial in $\mathbb{F}[x_1, \ldots, x_n]$ in a natural way, and the polynomial computed by the circuit is the polynomial computed by the output gate. When the only constants allowed are $\{-1, 0, 1\}$, we say that the circuit is *constant-free*.

The size of a circuit is its number of gates (i.e. the number of vertices of the graph). Remark that, by repeated squaring, polynomials of exponential degree, or with an exponential number of monomials, or with integer coefficients of exponential bitsize can be computed by arithmetic circuits: for instance,

$$p(x) = (1 + x)^{2^n}$$

has the arithmetic circuit of size $n+3$ of Figure 2.1.

$$\begin{tikzcd}
1 \arrow[dr] & & x \arrow[dl] \\
& + \arrow[d, bend right] \arrow[d, bend left] & \\
& \times \arrow[d, bend right] \arrow[d, bend left] & \\
& \times & \\
& \vdots \arrow[d, bend right] \arrow[d, bend left] & \\
& \times &
\end{tikzcd}$$

Figure 2.1: Arithmetic circuit of size $n+3$ for the polynomial $p(x) = (1+x)^{2^n}$.

As for languages, we are usually not interested in a single polynomial, but rather in *families* of polynomials like, e.g., the *determinant* family $(\det_n)_{n\in\mathbb{N}}$ defined by

$$\det_n(x_{1,1},\ldots,x_{n,n}) = \sum_{\sigma\in S_n} \varepsilon(\sigma) \prod_{i=1}^{n} x_{i,\sigma(i)},$$

($\det_n$ is a polynomial in n^2 variables), or its "cousin" the *permanent*

$$\mathrm{per}_n(x_{1,1},\ldots,x_{n,n}) = \sum_{\sigma\in S_n} \prod_{i=1}^{n} x_{i,\sigma(i)}.$$

That way we can speak of the asymptotic size of circuits to compute such polynomials, and we can define complexity classes, in a way that is reminiscent of P and NP (see, e.g., the book of Bürgisser [Bür00a]).

Definition 17 (Valiant's classes)**.**

- VP *is the class of families of (multivariate) polynomials* $(p_n)_{n\in\mathbb{N}}$ *where:*
  - $\deg(p_n)$ *is polynomially bounded,*
  - (p_n) *has polynomial-size arithmetic circuits.*
- *A family* $(q_n)_{n\in\mathbb{N}}$ *is in* VNP *if there exists* $(p_n)\in\mathsf{VP}$ *and a polynomial* $r(n)$ *such that:*

$$q_n(x_1,\ldots,x_{r(n)}) = \sum_{a_1,\ldots,a_{r(n)}\in\{0,1\}} p_n(x_1,\ldots,x_{r(n)},a_1,\ldots,a_{r(n)}),$$

  that is, (q_n) *is an exponential sum of a* VP *family.*

Note that the number of variables of families in VP is polynomially bounded (since it must have circuits of polynomial size). As in the Boolean world, the question VP $=$ VNP? is wide open and is central to the domain. Actually, the permanent family (per_n) is VNP-complete, thus the question is whether the permanent has arithmetic circuits of polynomial size.

These classes are non-uniform, but we could require that the circuits (C_n) for (p_n) can be built by a Turing machine in time $n^{O(1)}$, and obtain the *uniform* versions of VP and VNP.

If (p_n) is in uniform VNP, then the function which, on input (m, n), computes the coefficient of the monomial m in p_n, is in GapP. This is a large class (Toda's theorem [Tod91] shows that $\mathsf{P}^{\mathsf{GapP}}$ contains the whole polynomial hierarchy PH). In Section 2.1.2 we will consider polynomials whose coefficients are in MA: let us define that now.

The Boolean class of complexity MA (Merlin-Arthur) is a kind of probabilistic version of NP and lies "just above" NP.

Definition 18. MA *is the class of languages L for which there exists $A \in \mathsf{P}$ and a polynomial $q(n)$ such that:*

- *if $x \in L$ then $\exists y \in \{0,1\}^{q(|x|)} \Pr_{r \in \{0,1\}^{q(|x|)}}((x, y, r) \in A) \geq 2/3$;*
- *if $x \notin L$ then $\forall y \in \{0,1\}^{q(|x|)} \Pr_{r \in \{0,1\}^{q(|x|)}}((x, y, r) \in A) \leq 1/3$.*

We say that a family of polynomials (p_n) has its *coefficients in* MA if p_n has a polynomial number of variables and a polynomial degree, and its coefficients are integers computable in MA, that is, the language

$$\{(n, m, i, b) \mid \text{the } i\text{-th bit of the coefficient of the monomial } m \text{ is } b\}$$

is in MA.

Polynomial Identity Testing

A central problem in (Boolean) complexity theory is Polynomial Identity Testing (PIT), in particular because it is one of the few problems in BPP (actually in coRP) not (yet) known to be in P. Finding a polynomial-time deterministic algorithm for problems for which only probabilistic ones are known is called *derandomisation*.

As its name suggests, PIT is about finding whether polynomials given by arithmetic circuits are identical, or equivalently whether one polynomial is zero:

- *input:* a (constant-free) arithmetic circuit C computing a polynomial

$$p \in \mathbb{Z}[x_1, \ldots, x_k];$$

- *question:* is p identically zero?

Of course the answer depends on the field (more precisely on its characteristic). In this chapter we can safely assume the field is $\mathbb{R}$.

As mentioned above, a circuit can compute polynomials of exponential degree, with exponentially many monomials and with coefficients of exponential bitsize. Thus writing down p explicitly takes exponential time. The following usual randomised algorithm avoids all these traps:

- pick at random $a_1, \dots, a_k \in \{0, \dots, 2^{n^2}\}$ and $r \in \{2, \dots, 2^{n^2}\}$;
- accept iff $p(a_1, \dots, a_k) \equiv 0 \bmod r$.

Overflows are avoided thanks to the modulo r, and the strategy of evaluating at random points works thanks to Schwartz-Zippel lemma [Zip79; Sch80]:

Lemma 4 (Schwartz-Zippel)**.** *Let $p \in \mathbb{R}[x_1, \dots, x_k]$ be a nonzero polynomial of degree d and $S \subseteq \mathbb{R}$ a finite set. Then*

$$\mathrm{Pr}_{a_1,\dots,a_k \in S}(p(a_1, \dots, a_k) = 0) \leq \frac{d}{\#S}.$$

Actually, PIT is equivalent to a simpler problem that we call *Integer Identity Testing* (IIT) concerning circuits computing integers (i.e. they have no variables):

- *input:* a (constant-free) arithmetic circuit C computing an integer $N \in \mathbb{Z}$;
- *question:* $N = 0$?

Obviously, IIT reduces to PIT. The converse is true as well: in a circuit C computing a polynomial $p(x_1, \dots, x_k)$, it is enough to replace the variables by integers growing sufficiently fast thanks to the following lemma.

Lemma 5. *Let $p \in \mathbb{Z}[x_1, \dots, x_k]$ be a polynomial of degree d whose coefficients are bounded in absolute value by M. Let $\alpha_1, \dots, \alpha_k \in \mathbb{N}$ be integer satisfying:*

$$\alpha_1 \geq M + 1 \quad \textit{and} \quad \alpha_{i+1} \geq 1 + M(d+1)^i \alpha_i^d.$$

Then:

$$p \equiv 0 \iff p(\alpha_1, \dots, \alpha_k) = 0.$$

2.1.2 Fixed-Polynomial Lower Bounds on "Explicit" Polynomials

General Lower Bounds

Before trying to separate complexity classes, the first natural step is probably to show that there at least exist some polynomials that are "hard". As in the Boolean case, it might at first seem easy to show that for every k there exist polynomial families (p_n) that do not have arithmetic circuits of size n^k. This is indeed trivial in some cases:

- if p_n has degree $> 2^{n^k}$;
- if p_n has more than n^k variables;
- if the coefficients of p_n generate a field of transcendence degree $> n^k$ over $\mathbb{Q}$.

But these all are "bad" (trivial) reasons. Fortunately, Schnorr [Sch78] shows a more satisfactory result.

Theorem 7. *For all $t \in \mathbb{N}$, there exist $a_1, \ldots, a_{t^2} \in \{0, 1\}$ such that the univariate polynomial*

$$p(x) = \sum a_i x^i$$

does not have arithmetic circuits over $\mathbb{C}$ of size $\varepsilon \frac{t}{\log t}$ (here, $\varepsilon > 0$ is a constant independent of t).

Note that the proof is more involved than in the Boolean case, because arbitrary complex constants can be used even if the coefficients are in $\{0, 1\}$, and thus a simple counting argument or diagonalisation is not enough. Schnorr remarks that the coefficients of a polynomial computed by a circuit using constants $\alpha = (\alpha_1, \ldots, \alpha_k)$ is given by a polynomial mapping in α. Hence, finding hard polynomials reduces to finding a point outside the image of the mapping associated to some circuit which is universal for a given size.

However, this is a nonconstructive result and we have no idea of what the polynomial looks like. Its coefficients are computable in exponential time, which is far from efficient. We would rather have an "explicit" polynomial, like e.g.

$$p(x_1, \ldots, x_n) = x_1^d + \cdots + x_n^d$$

for which Baur and Strassen [BS83] have shown an $n \log d$ lower bound. Alas, this is currently the best known lower bound for an "explicit" polynomial.

Explicitness

Before presenting the results we obtained in [FPV15] on that problem, we should say what an "explicit" polynomial is. Unfortunately there is no clear consensus. Some consider that being able to compute "efficiently" the coefficients of the polynomial is enough to be explicit. Others would insist that the polynomial has uniform circuits. We consider both variants in the following results.

Uniform Circuits

The first result shows that, if our circuits are not allowed to use arbitrary complex constants, then there is an explicit hard polynomial, where explicit here means uniform in VNP.

Theorem 8. *For all $k > 0$, there is a uniform family of polynomials (p_n) in VNP, $p_n \in \mathbb{Z}[x_1, \ldots, x_n]$, such that p_n has no constant-free arithmetic circuits of size n^k.*

The proof goes along the following lines. If the permanent polynomial family, which is VNP-complete, has no circuits of size n^k (as is widely conjectured), then the result is true. Otherwise, the small circuits for the permanent enable to collapse the whole counting hierarchy to MA (see [FPV15] based on [Lun+92]). But in the counting hierarchy, thus in MA, we are able to diagonalise against all "sign conditions" realisable by circuits of size n^k. The MA protocol can then be used to define a VNP family whose sign condition is different.

Easy Coefficients

For the other "definition" of explicitness, we shall consider here the case of circuits using arbitrary constants from $\mathbb{C}$. This is obviously more entangled and we need to assume the generalised Riemann hypothesis (GRH) to tackle these constants. That way we were able to generalise the result of Jansen and Santhanam [JS12] that there exist polynomials with coefficients in MA (thus, explicit in some sense) but not computable by arithmetic circuits of size n^k over $\mathbb{Z}$. Assuming GRH, we extend their result to the case of circuits over the complex field.

Theorem 9. *Assume GRH is true. For any constant k, there is a family (p_n) of polynomials on n variables, of degree $n^{O(1)}$, with coefficients in $\{0, 1\}$ computable in MA, and such that p_n has no arithmetic circuits of size n^k over $\mathbb{C}$.*

The idea is to find coefficients that cannot be obtained by a *universal* arithmetic circuit $U(a, x)$ simulating circuits of size n^k. Here, $a \in \mathbb{C}^N$ (for $N \sim n^{2k}$) encodes the structure and the constants of the simulated circuit. If we find values $\gamma_1, \ldots, \gamma_d \in \{0, 1\}$ such that $U(a, x) \neq \sum \gamma_i x^i$ for all $a \in \mathbb{C}^N$, then the polynomial $p_\gamma(x) = \sum \gamma_i x^i$ has no circuits of size n^k.

But deciding whether a system of equations has no solution over $\mathbb{C}$ can be done in the polynomial hierarchy thanks to [Koi96]. With a few more quantifiers, we can actually find values $\gamma_1, \ldots, \gamma_d \in \{0, 1\}$ for which there are no solutions over $\mathbb{C}$, and hence for which p_γ has no circuits of size n^k.

This gives a polynomial p_γ with coefficients in PH. To bring them down to MA, we use the same kind of trick as for Theorem 8: if the permanent has no circuits of polynomial size, then it is itself the desired polynomial. Otherwise, $\mathsf{PH} = \mathsf{MA}$ thanks to [Bür00b] and [Lun+92], and we are done.

Here, GRH is used in Koiran's result [Koi96] to reduce finding the solutions of a system of polynomial equations over $\mathbb{C}$ to solutions over $\mathbb{F}_p$, for a small prime p, and hence place the problem in PH.

Other variants of these results can be found in [FPV15]. The next sections are devoted to the study of restricted models of computation.

2.1.3 ABPs vs Formulas

As can be seen from the previous section, known circuit lower bounds are very limited and separations of classes like VP and VNP seem completely out of reach for the moment. That's why a large area of research focusses on *restricted* frameworks where lower bounds are of course easier to prove. The present section and the next study some usual restrictions and present lower bounds.

We can weaken the framework in two directions: considering weaker models of computation on the one hand instead of arithmetic circuits, and consider computations on "limited" algebraic structures on the other hand, like non-commutative ones. It is hoped that exploring restricted computations and obtaining lower bounds will help to get results in the general case.

Restricting the Model of Computation

Here we shall consider *multilinear* ABPs and formulas, terms that we all define now.

Definition 19. *A polynomial is* multilinear *if it has degree at most one in each variable.*

Many important polynomials are multilinear, e.g., the determinant, the permanent and matrix product. A natural restricted model for computing multilinear polynomials is *multilinear computation*, in which all intermediate stages of the computation are required to be multilinear as well.

There is a large body of research devoted to multilinear computation, in particular to proving lower bound for multilinear formulas (for which the underlying computation graph is a tree). The first result in this direction was the breakthrough paper of Raz [Raz04a] showing that multilinear formulas for both the permanent and the determinant must be of super-polynomial size. Later, in [Raz04b], Raz showed that multilinear *circuits* are super-polynomially stronger than multilinear formulas (see [RY08] for a simpler proof). Exponential lower bounds for *constant depth* multilinear circuits, as well as strong separations based on circuit-depth, were proved in [RY09]. Super-linear lower bounds for the size of arithmetic circuits were proved in [RSY08].

Here we further extend this line of work by proving a super-polynomial separation between multilinear algebraic branching programs (ABPs) and multilinear formulas. As multilinear circuits can efficiently simulate multilinear ABPs, this strengthens the mentioned results of [Raz04b; RY09]. We first need to define ABPs and formulas.

Definition 20.

- *An algebraic branching program (ABP) is a directed acyclic graph with two special nodes in it: a start-node and an end-node. The edges of the ABP are labeled by either variables or field elements. Every directed path γ from the start-node to the end-node computes the monomial f_γ which is the product of all labels on the path γ. The ABP computes the polynomial $f = \sum_\gamma f_\gamma$, where the sum is over all paths γ from start-node to end-node. The size of an ABP is the number of nodes in the graph.*

- *A formula is a rooted directed binary tree (the edges are directed toward the root). The leaves of the formula are labeled by either variables or field elements. The inner nodes have in-degree two and are labeled by either $+$ or $\times$. A formula computes a polynomial in the obvious way. The size of a formula is the number of nodes.*

Both ABPs and formulas have natural restrictions to the multilinear world. An ABP is *multilinear* if on every directed path from start-node to end-node no variable appears more than once. A formula is *multilinear* if every sub-formula in it computes a multilinear polynomial.

ABPs capture the computational power of iterated matrix product: For every ABP of size s, there are $\mathrm{poly}(s)$ many matrices $A_1, A_2, \ldots$ of dimensions $\mathrm{poly}(s) \times \mathrm{poly}(s)$ with entries that are either variables or field elements, so that the polynomial computed by the ABP is the $(1, 1)$ entry in the matrix $A_1 A_2 \cdots$. In the other direction, for every s matrices of dimensions $s \times s$, there is a (multi-start-node and multi-end-node) ABP of size $\mathrm{poly}(s)$ computing the product of the matrices. In fact, ABPs also capture the computational power of the determinant: For every ABP of size s, there is a matrix A of dimension $\mathrm{poly}(s)$ with entries that are either variables or field elements, so that the determinant of A is the polynomial the ABP computes [Val79; MP08], and the determinant can be computed by a polynomial-size ABP [Ber84; Sam42; MV97]. The link between the determinant and ABPs was first shown by Toda [Tod92], using the equivalent model of skew circuits. However, the known polynomial-size ABPs for the determinant are not multilinear, so the lower bound of [Raz04a] does not yield our result (by current knowledge).

Formulas, on the other hand, capture a computational model in which every sub-computation can be used only once (as the underlying computation graph is a tree). Since formulas can be parallelized to have depth which is logarithmic in their size, they also capture the parallel time it takes to perform the computation.

Separating ABPs and Formulas

It is known that ABPs can efficiently simulate formulas [Val79]. Similar ideas show that *multilinear ABPs* can efficiently simulate *multilinear formulas*. A natural question is thus whether the other direction holds as well. In the multilinear setting, this question was raised in particular by Jansen in [Jan08]. We show that in the multilinear world it does not (a similar separation is believed to hold for general algebraic computation).

Theorem 10. *For every positive integer n, there is a multilinear polynomial $F = F_n$ in n variables with zero-one coefficients so that the following holds:*

(i) There is a uniform algorithm that, given n, runs in time $O(n)$ and outputs a multilinear ABP computing F.

(ii) Over any field, every multilinear formula computing F must be of size $n^{\Omega(\log n)}$.

Our lower bound of $n^{\Omega(\log n)}$ is tight since any polynomial-size multilinear ABP can be simulated by a multilinear formula of size $n^{O(\log n)}$ (see, e.g., [RY08]).

The proof of Theorem 10 consists of two parts: (i) constructing a small multilinear ABP computing some polynomial F and (ii) showing that any multilinear formula computing F is of super-polynomial size. The two parts have conflicting demands: In part (i) we wish to make the polynomial F simple enough so that a small ABP can compute it, whereas in part (ii) we will need to rely on the hardness of F to prove a lower bound. To succeed in both parts we need to take full advantage of the expressive power that ABPs grant us. Below we give a high-level description of the proof, focusing on part (ii), which is considerably more complicated. Along the way we will highlight ideas from previous works that are used in the proof.

The lower bound part of the proof uses several ideas introduced in previous works [Raz04a; Raz04b; RY08]. Of particular importance is the notion of a *full-rank* polynomial. Following a notion defined by Nisan [Nis91] in the case of non-commutative computations and that we shall also encounter in Section 2.1.4, a polynomial f can be used to define a family of matrices $\{M(f_\Pi)\}_\Pi$, where Π ranges over all partitions of the variables X into two sets of variables Y, Z of equal size (these are the so-called *partial derivative* matrices). The polynomial f is said to have *full-rank* if the rank of $M(f_\Pi)$ is full for every such Π. This property turns out to be useful in showing complexity lower bounds for f. Indeed, Raz showed that every full-rank polynomial f cannot have polynomial-size multilinear formulas [Raz04a; Raz04b].

To the best of our knowledge, full-rank polynomials may also require super-polynomial-size ABPs. Thus, in order to prove our separation we will look for a property which is *weaker* than being full-rank and is still useful for proving lower bounds. One of the main new ideas in our proof is a construction of a special *subset* of partitions, called *arc-partitions*, which is sufficiently powerful to carry through the lower bound proof and, at the same time, simple enough to carry part (i) of the proof. The number of arc-partitions is much smaller than the total number of partitions. Nevertheless, we are still able to show that every *arc-full-rank* polynomial f (i.e., $M(f_\Pi)$ has full rank for all arc-partitions Π) does not have polynomial-size multilinear formulas.

We now go into more details as to how this family of partitions is defined and what makes it useful. We will start by describing a *distribution* over partitions. The partitions that will have positive probability of being obtained in this distribution will be called arc-partitions. The distribution is defined according to the following (iterative) sampling algorithm (see Figure 2.2). Position the n variables on a cycle with n nodes so that there is an edge between i and $i+1$ modulo n. Start with the *arc* $[L_1, R_1] = \{0, 1\}$ (an arc is a connected path on the cycle). At step $t > 1$ of the process, maintain a partition of the arc $[L_t, R_t]$. "Grow" this partition by first picking a pair uniformly at random out of the three possible pairs $(L_t - 2, L_t - 1), (L_t - 1, R_t + 1), (R_t + 1, R_t + 2)$, and then defining the partition Π on this pair to map to a random permutation of the two variables y_{t+1}, z_{t+1}. After $n/2$ steps, we have chosen a partition of the n variables into two disjoint, equal-size sets of variables Y, Z.

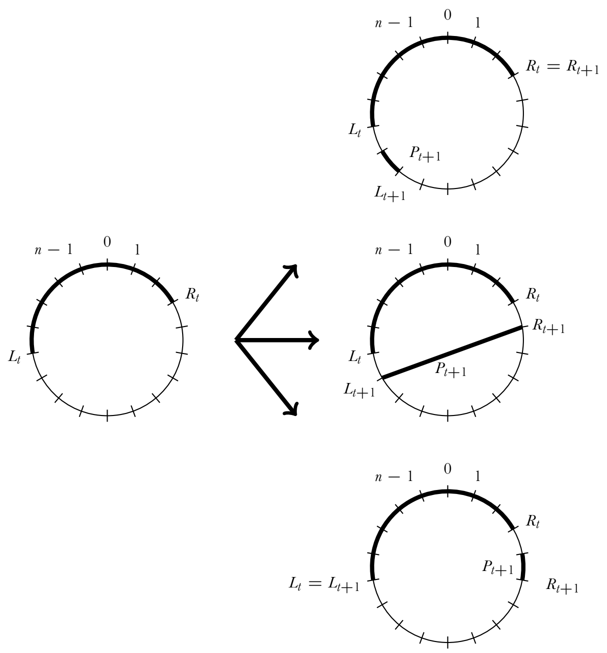


Figure 2.2: *Incremental definition of a pairing. On the left the arc $[L_t, R_t]$ in the n-cycle. On the right the three options for the next pair P_{t+1} and the corresponding L_{t+1}, R_{t+1}.*

The arc-partitions allow us to adapt the key argument in [Raz04a]. Let us remind roughly how this argument works after the simplifications from [SY10a, Section 3.6]. Every multilinear formula computes a polynomial which can be seen as a small sum of polynomials, each polynomial defining a "non-redundant" K-coloring of the n-variables with $K \sim \log n$. This is simply a mapping $C : [n] \mapsto [K]$ so that the pre-image of every color $k \in [K]$ is not too small. A color k is said to be "balanced" with respect to a partition Π if the number of Y variables of color k is roughly the same as the number of Z variables of color k. Now, for a given coloring C, if we choose a random partition Π from the set of *all* partitions, simple properties of the hyper-geometric distribution imply that the probability that all colors in C are "balanced" is at most $p = n^{-\Omega(K)} = n^{-\Omega(\log n)}$. This bound, in turn, proves a roughly $1/p = n^{\Omega(\log n)}$ lower bound for the size of multilinear formulas.

Following a similar strategy, we show that for any "non-redundant" K-coloring C, for a random arc-partition, the probability that all colors in C are "balanced" is at most $n^{-\Omega(K)}$ as well. This turns out to be significantly more difficult than showing it for a random partition (from the set of all partitions). The hardest part of the proof is analyzing a random walk on a two-dimensional "distorted chessboard" where we need to prove certain anti-concentration results.

Let us now turn to another kind of restriction, different from multilinearity.

2.1.4 Unambiguous Circuits

Nisan [Nis91] proved in 1991 exponential lower bounds for non-commutative arithmetic formulas and more generally for non-commutative algebraic branching programs. In this section we extend this result to "unambiguous circuits".

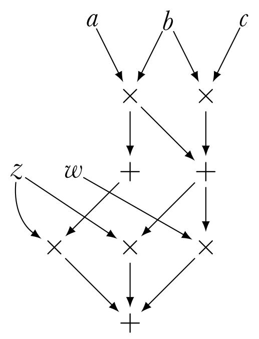
Noncommutativity and Unambiguity

Let $\mathbb{F}$ be a field. In the *non-commutative* ring $\mathbb{F}\langle x_1, \ldots, x_n\rangle$, variables do not commute so that $x_i x_j$ and $x_j x_i$ $(i \neq j)$ are distinct monomials[1]. Non-commutative computations arise naturally for instance when computing over matrices, but also can have applications for *commutative* computations (see [CRS03; Bar00], in particular the use of non-commutative determinants to approximate the commutative permanent).

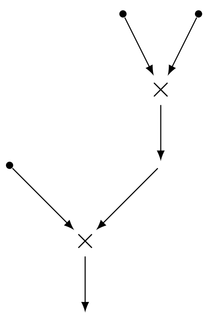
Given a (non-commutative) circuit, we can look at the set of monomials it produces (before any grouping/cancellations). If we pretend that the computation is also non-associative, a monomial comes with parentheses to indicate the "way" in which it was computed. The pattern of parentheses for a given monomial (the structure of the monomial in a sense) can also be seen as a tree. We will focus on circuits

[1]Nevertheless addition is still commutative and the rules for the constants do not change, according to the underlying field $\mathbb{F}$.

where this structure (the *shape*) is the same for all the monomials computed by the circuit, and we will call these circuits *unambiguous*.



(**a**) An unambiguous circuit C.



(**b**) The shape of the circuit C, corresponding to the pattern of parentheses $\bullet \times (\bullet \times \bullet)$.

Figure 2.3: An unambiguous circuit and its shape.

If one computes an algebraic branching program as a circuit, then monomials are all obtained by successive multiplication on the right, and they all have the same structure. Our model is thus more general than the one considered by Nisan.

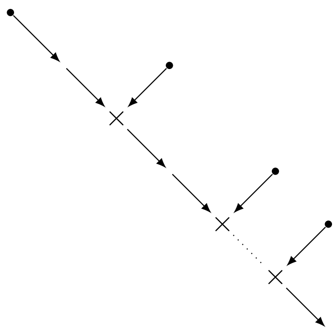


Figure 2.4: Shape of an ABP turned into an unambiguous circuit.

Let us emphasize that the class of polynomials computable by unambiguous circuits of polynomial size is quite large and natural: it contains all the ABPs as already explained (cf. Figure 2.4), as well as for instance the palindrome polynomial used in [Nis91; LMS15]. It is rich enough to contain, for all k, the polynomial f_k defined in [LMS15] which requires exponential-size circuits of skew-depth k, thus creating a hierarchy of increasing power inside general non-commutative circuits. A final example: the $\Theta(n2^n)$ computation of the permanent, tersely explained by Nisan in [Nis91], is also unambiguous and is asymptotically as fast as Ryser's formula (but has the advantage of being monotone and non-commutative).

Lower Bound

Nisan's result was extended in [LMS15] to a more powerful model, so-called *skew circuits*, arithmetic circuits where every multiplication involves at most one argument which is not a variable or a constant. There, non-commutative skew circuits were shown to be exponentially more powerful than non-commutative branching programs, but exponentially less powerful than general non-commutative circuits. This model and some extensions are the strongest model of non-commutative computation for which we have superpolynomial lower bounds.

Here, we again extend Nisan's result but in a different direction, namely for unambiguous circuits. Perhaps the most striking aspect of Nisan's paper, more than its elegance, is that it gives an *exact* expression for the complexity of *any* polynomial. More precisely, the minimal size of a branching program computing a polynomial f is expressed via the ranks of a family of matrices defined by f, for all branching programs in a certain "canonical" form. We prove a generalization of his theorem, characterizing the minimal size of a "canonical" unambiguous circuit computing any polynomial f, also in terms of ranks of matrices. This exact characterization also yields exponential lower bounds, making unambiguous circuits another "strongest" model of non-commutative computation for which we have superpolynomial lower bounds (it is incomparable with the models of [LMS15]).

For a homogeneous polynomial P of degree d and each integer $i \leq d$, Nisan defined the *partial derivative matrix* $M^{(i)}(P)$, which is a $n^{d-i} \times n^i$ matrix whose rows are indexed by monomials of degree $(d-i)$ and columns by monomials of degree i. The entry (m_1, m_2) of the matrix is defined to be the coefficient of the monomial $m_1 m_2$ in P. Intuitively speaking, the rank of the matrix $M^{(i)}(P)$ is a measure of how "correlated" the prefix of length i of a monomial appearing in P is to the rest of the monomial. Small ABPs have "information bottlenecks" at each degree i, and hence the amount of correlation in the computed polynomial must be low. In our case the correlation will be between the prefix of degree p and the suffix of degree $(d-p-i)$ on the one hand, and the middle part of degree i on the other hand. In that case we say the "type" of the gate is (i, p) (see Figure 2.5).

We can then define, for a homogeneous polynomial P of degree d and a particular type (i, p), the analogue of Nisan's partial derivative matrix: $M^{(i,p)}(P)$ is a matrix of size $n^{d-i} \times n^i$, rows are indexed by all pairs of monomials

$$(m_1, m_3) \in \{x_1, \dots, x_n\}^p \times \{x_1, \dots, x_n\}^{d-p-i},$$

columns are indexed by monomials $m_2 \in \{x_1, \dots, x_n\}^i$, and $M^{(i,p)}(P)_{(m_1,m_3),m_2}$ is the coefficient of the monomial $m_1 m_2 m_3$ in P.

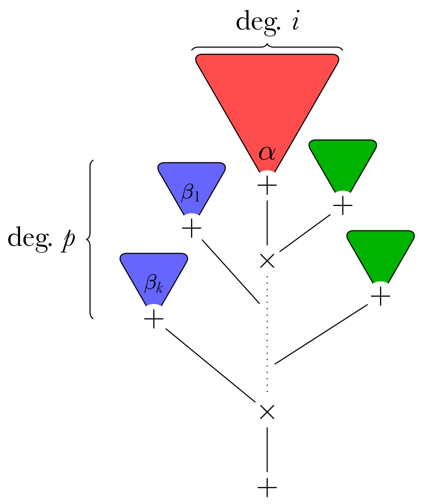


Figure 2.5: A shape and a gate α of type (i, p).

Our result expresses the minimal number of addition gates in an unambiguous circuit for a polynomial P as a function of the ranks of the matrices $M^{(i,p)}(P)$.

Theorem 11. *Let P be a homogeneous polynomial of degree d and $\mathcal{T}$ a shape with d leaves. Then the minimal number of addition gates needed to compute P by a (canonical) unambiguous circuit with shape $\mathcal{T}$ is exactly equal to*

$$\sum_{(i,p)\in S} \operatorname{rank}\left(M^{(i,p)}(P)\right),$$

where S is the set of all existing types of $+$-gates in the shape $\mathcal{T}$.

As an application, we obtain the following exponential lower bound for the computation of the permanent and the determinant.

Theorem 12. *Computing the permanent or the determinant on $n \times n$ variables with an unambiguous circuit requires at least $\binom{n}{n/3}$ gates.*

PIT

Finally we consider the PIT for our model of non-commutative unambiguous circuits. Note that, as in the commutative setting, there is a polynomial-time randomized algorithm for the non-commutative PIT [BW05] (for *polynomial-degree* circuits only). But here derandomisation has some significant success. A first efficient white-box[2] deterministic algorithm for non-commutative ABPs was given by Raz & Shpilka [RS05]. Here we use ideas from a simpler construction given by Arvind et al. [AJS09; AMS10] to get a polynomial-time deterministic PIT algorithm for unambiguous circuits over $\mathbb{R}$ or $\mathbb{C}$. This seems to be the strongest non-commutative model so far for which PIT has been derandomised.

The idea of the algorithm over $\mathbb{R}$ is simple and uses the Hadamard product.

Definition 21. *Given two polynomials $P = \sum_m a_m m$ and $Q = \sum_m b_m m$ (where m denotes a monomial), the Hadamard product of P and Q is*

$$P \odot Q = \sum_m a_m b_m m.$$

We first give a construction to perform the Hadamard product of two unambiguous circuits with the same shape. In other words, we prove that the class of unambiguous circuits of a given shape is *stable under Hadamard product*. As in the case of ABPs, it will provide a deterministic polynomial-time algorithm for PIT over unambiguous circuits.

Theorem 13 (Hadamard product of two unambiguous circuits)**.** *Let $\mathcal{C}$ and $\mathcal{D}$ be two unambiguous circuits (in canonical form), of the same shape, and of size s and s', that compute two polynomials P and Q. Then $P \odot Q$ is computed by an unambiguous circuit of size at most ss'; moreover, this circuit can be constructed in polynomial time.*

Corollary 2. *There is a deterministic polynomial-time algorithm for PIT for polynomials computed by non-commutative unambiguous circuits over $\mathbb{R}$.*

Proof. Given $P(x_1, \dots, x_n)$ computed by an unambiguous circuit, construct the circuit which computes $(P \odot P)(x_1, \dots, x_n)$ and evaluate it on $(1, 1, \dots, 1)$. The output is the sum of the squares of the coefficients of P, therefore it is equal to 0 if and only if P is equal to the zero polynomial. $\square$

2.1.5 PIT on Roots of Unity

Our last highlight is taken from [Bal+21] and is a variant of the usual PIT. Arithmetic circuits over $\mathbb{C}$ can take arbitrary constants as inputs, but of course, so as to be able to encode the problem in binary, for PIT we only consider constant-free arithmetic circuits. There is at least one case of interest where we could add complex constants, though: when these constants are roots of unity $\zeta_n = e^{\frac{2i\pi}{n}}$ since such a

[2] A PIT algorithm is *white-box* if it can use the structure of the computation model; it is *black-box* if it only requires an evaluation oracle.

value can simply be encoded as n in binary. Indeed, identity testing in number fields is a fundamental problem in algorithmic algebra that has been studied in relation to solving systems of polynomial equations [Ge93; Koi96] and polynomial identity testing [CK00].

Among number fields, cyclotomic fields, i.e., those generated by roots of unity, play a central role. Our aim is to design an efficient algorithm for the following problem Cyclotomic Identity Testing (CIT):

- *input:* an algebraic circuit C computing a (univariate) polynomial $g(x)$, together with an integer n given in binary;
- *question:* determine whether $g(\zeta_n) = 0$, where $\zeta_n = e^{2\pi i/n}$ (a complex primitive n-th root of unity).

Observe that CIT is at least as hard as PIT obviously.

Plaisted [Pla84] gave a randomised polynomial-time algorithm for the problem where the polynomial f is given in sparse representation, and subsequently, two different deterministic polynomial-time algorithms were given by Cheng et al. [CTV10; Che07]. The conclusion of [CTV10] raises the question of the complexity of CIT. The authors note that this problem lies in the counting hierarchy based on results of [All+09]. Our result substantially improves on [CTV10].

Theorem 14. CIT *is in* BPP *assuming GRH, and in* coNP *unconditionally.*

The algorithm works by computing in $\mathbb{F}_p$ where $\mathbb{F}_p$ has a *primitive n-th root of unity*, namely

$$\omega_n \in \mathbb{F}_p \text{ such that } (\omega_n)^n = 1 \text{ but } (\omega_n)^i \neq 1 \text{ for all } 0 < i < n.$$

A number α is a *cyclotomic integer* if it is in $\mathbb{Z}[\zeta_n]$. For $s \in \mathbb{N}$, we will say that the cyclotomic integer $\alpha = g(\zeta_n)$ has *a description of size s* if it is computed by a circuit C where s is the sum of the size of C and the bit-length of n.

The ring of cyclotomic integers is equipped with a *norm* $\mathcal{N}(\alpha) \in \mathbb{Z}$ having the following properties.

Proposition 10. *Let α be a cyclotomic integer with a description of size s. Then*

$$|\mathcal{N}(\alpha)| \leq 2^{2^{2s}}.$$

Proposition 11. *Let $p \in \mathbb{Z}$ be a prime such that $\mathbb{F}_p$ contains a primitive n-th root of unity ω_n. Given $g(x) \in \mathbb{Z}[x]$, denoting by $\bar{g} \in \mathbb{F}_p[x]$ the reduction of g modulo p, we have:*

1. *if $g(\zeta_n) = 0$ then $\bar{g}(\omega_n) = 0$, and*

2. *if $\bar{g}(\omega_n) = 0$ then $p \mid \mathcal{N}(g(\zeta_n))$.*

Proposition 11 suggests a natural test for CIT: evaluate the circuit in a finite field $\mathbb{F}_p$ that contains a primitive n-th root of unity. Since the multiplicative group $\mathbb{F}_p^*$ is cyclic, it is clear that $\mathbb{F}_p^*$ contains a primitive n-th root of unity just in case

$$n \mid (p-1), \quad \text{i.e.,} \quad p \equiv 1 \bmod n.$$

Using good bounds on the density of primes in arithmetic progressions under GRH (see, e.g. [DM13, Chapter 20, page 125]), we can obtain the following result.

Proposition 12. *Let α be a non-zero cyclotomic integer whose description has size at most s. Suppose that p is chosen uniformly at random from*

$$\{q \in \mathbb{N} : q \leq 2^{5s} \;\; \textit{and} \;\; q \equiv 1 \bmod n\}.$$

Assuming GRH,

1. *p is prime with probability at least $\frac{1}{6s}$, and*

2. *given that p is prime, the probability that it divides $\mathcal{N}(\alpha)$ is at most 2^{-s}.*

Altogether these considerations enable to show that the algorithm of Figure 2.6 places CIT in BPP. Without GRH, known bounds on the density of primes in arithmetic progressions are not quite as good (see [IK04, Corollary 18.8]), and in Proposition 12 we would only obtain the *existence* of a suitable prime. Then by guessing instead of randomising, our problem can be placed in coNP instead of BPP.

2.2 Perspective: Could PIT be EXP-Complete?

If, in this chapter, we have seen variants and restrictions of PIT, that is because designing an efficient deterministic algorithm for the original PIT remains elusive. The situation is actually so dramatic that we cannot rule out, for the moment, that PIT might be surprisingly powerful:

Open question 3. *Could PIT be EXP-complete?*

It could even be NEXP-complete for polynomial-time Turing reductions, that is,

$$\mathsf{P}^{\mathrm{PIT}} = \mathsf{NEXP}?$$

Of course this possibility seems incredible since most believe that the usual algorithm for PIT will be derandomised some day and that actually PIT $\in \mathsf{P}$…

Cyclotomic Identity Testing

Input: an algebraic circuit C and an integer n, written in binary, of combined size s.

Output: whether $g(\zeta_n) = 0$ for the polynomial $g(x)$ computed by C.

```
1: Pick $p$ uniformly at random from
   $$\{q \in \mathbb{N} : q \leq 2^{5s},\ q \text{ prime},\ \text{and } q \equiv 1 \bmod n\}.$$
2: Pick $h$ uniformly at random from
   $$\{a : a \in \mathbb{F}_p^*, \bigwedge_{\substack{2 \leq q < 10 \log(p-1) \\ q | p-1}} a^{\frac{p-1}{q}} \neq 1\}.$$
3: Set $\omega_n := h^{\frac{p-1}{n}} \in \mathbb{F}_p^*$.
4: Output 'Zero' if $\bar{g}(\omega_n) = 0$ where $\bar{g}$ is the reduction of $g$ modulo $p$;
   otherwise output 'Non-Zero'.
```

Figure 2.6: Algorithm for the CIT problem

Local Reductions

However, Question 3 seems out of reach for the moment. What we shall propose here is a weaker version together with some ideas to settle it. We must first explain what local reductions are [PY86].

Definition 22. *A local reduction is a reduction that is computable in polylogarithmic time.*

More precisely, a problem A locally reduces to a problem B, written $A \leq_{\log} B$, if there exists a function f computable in polynomial time which, on input i (written in binary, and thus of logarithmic size compared to x) and with oracle access to x, outputs the i-th bit y_i of y such that

$$x \in A \iff y \in B.$$

Of course, local reductions are less powerful than usual polynomial-time (many-one) reductions, thus it is harder to be complete under $\leq_{\log}$ and giving a negative answer to the following question must be easier (or rather less hard, let's say) than to Question 3.

Open question 4. *Could PIT be* EXP*-complete for local reductions?*

Note that most NP-complete problems remain complete under local reductions (see [PY86]), showing that local reductions are actually not so weak. This explains why Question 4 remains relevant. Still, that PIT is so powerful seems completely impossible, and yet this question remains open. Before sketching some ideas, we present an alternative framework which might be easier to grasp.

Succinct Problems

We say that a Boolean circuit C of size s *represents* a word x of size n (where $s << n \leq 2^s$) if for all $i \leq n$ given in binary, $C(i) = x_i$ (that is, $C(i)$ computes the i-th bit of x). Note that C can be exponentially smaller than x. For a problem A, its *succinct version* $\mathsf{succ}\,A$ is the following problem:

- *input:* a Boolean circuit C representing a word x;
- *question:* does x belong to A?

We can think of C as a "compressed" representation of x. Since the input can be exponentially smaller, $\mathsf{succ}\,A$ might be exponentially more difficult. However, only a negligible fraction of the initial inputs x can be encoded that way of course, therefore $\mathsf{succ}\,A$ will be difficult only if some hard instances are indeed compressible. It is thus rather surprising that most NP-complete problems see their succinct version become NEXP-complete…

Maybe the link with local reductions is clear now: succinct problems work on inputs that are exponentially "compressed", and local reductions work in polylogarithmic time. Thus, on the "uncompressed" input, the local reduction will work in polynomial time. This observation is summarised by the following proposition (see [PY86]).

Proposition 13. *If A is NP-complete under local reductions, then $\mathsf{succ}\,A$ is NEXP-complete (under usual, polynomial-time many-one reductions).*

Similarly, if A is EXP-complete for local reductions, then $\mathsf{succ}\,A$ is EEXP-complete (for usual, polynomial-time many-one reductions). Here, EEXP is double-exponential time, that is,

$$\mathsf{EEXP} = \mathsf{DTIME}(2^{2^{n^{O(1)}}}).$$

For the problem A we will consider PIT, or rather IIT which is equivalent as we saw in Section 2.1.1. Thus succPIT is the following problem:

- *input:* a Boolean circuit C of size s representing an arithmetic circuit D of size $\leq 2^s$ computing an integer $N \in \mathbb{Z}$;
- *question:* $N = 0$?

Since our goal is to give a negative answer to Question 4, thanks to Proposition 13 we can rephrase it in terms of the succinct version of PIT.

Goal: prove that succPIT is not EEXP-complete.

We shall see below the interest of working with succPIT instead of PIT.

Some Ideas

Here we only sketch informally ideas that might contribute to settle Question 4. For a contradiction, suppose succPIT is EEXP-complete. Let $t(n) = 2^{2^{n^{o(1)}}}$ be a sufficiently large function (to be fixed later). By the time hierarchy theorem, succPIT then cannot have an algorithm working in time $t(n)^{O(1)}$. In particular, if $a_1, \ldots, a_k \in \mathbb{N}$, $k \leq t(n)$, are prime numbers computable in time $t(n)$, then the following algorithm for succPIT:

- compute $a_1, \ldots, a_k$
- accept iff $D \equiv 0 \bmod a_i$ for all i

does not correctly decide succPIT since it works in time $t(n)^{O(1)}$. Hence there must exist infinitely many inputs C (call n their size) representing arithmetic circuits D (of size $\leq 2^n$) such that $D \neq 0$ but $D \equiv 0 \bmod a_i$ for all i. In other words,

$$D = \lambda \prod_i a_i. \qquad (\star)$$

Let us ponder this situation: a product of $t(n) = 2^{2^{n^{o(1)}}}$ "arbitrary" integers can be computed by an arithmetic circuit of size $\leq 2^n$ having a description C of size n. This means that quasi-exponential size products are easy to compute, even by circuits having a logarithmic description. The problem is that we only compute a multiple of $\prod a_i$ and we do not control the factor λ in Equation $(\star)$.

Pretend for a moment that $\lambda = 1$ in Equation $(\star)$, that is, D computes $\prod a_i$ exactly. This is not a reasonable assumption but it enables us to show the machinery we can develop.

Fix an encoding of Boolean formulas ϕ into prime integers p_ϕ, and consider the encoding p_i of the i-th formula ϕ of size $\log t(n)$ that is satisfiable. Then take $a_i = p_i$, and the arithmetic circuit $D = \prod a_i$ of size 2^n having a description C of size n. The following algorithm with advice C of size n, working in time $2^{O(n)}$ on input ϕ of size $\log t(n)$:

$$\text{accept iff } D \equiv 0 \bmod p_\phi$$

accepts exactly those formulas ϕ of size $\log t(n)$ that are satisfiable. In other words,

$$\text{SAT} \in \mathsf{DTIME}(2^{O(t^{(-1)}(2^n))})/t^{(-1)}(2^n).$$

Here is the place where we need that D has a logarithmic description, and hence where it is useful to consider succPIT instead of PIT:

Lemma 6 (folklore)**.** $\mathsf{NP} \subset \mathsf{DTIME}(2^{f(n)})/f(n)$ *implies* $\mathsf{NP} \subseteq \mathsf{DTIME}(2^{f(n)})$.

Thanks to this lemma we get:

$$\mathsf{NP} \subseteq \mathsf{DTIME}(2^{O(t^{(-1)}(2^n))}).$$

In particular, since PIT $\in \mathsf{coRP} \subseteq \mathsf{coNP}$, we have

$$\text{PIT} \in \mathsf{DTIME}(2^{O(t^{(-1)}(2^n))})$$

and thus

$$\mathsf{succPIT} \in \mathsf{DTIME}(2^{O(t^{(-1)}(2^{2^n}))}).$$

For t sufficiently large, this is a contradiction with the assumption that succPIT is EEXP-complete because

$$2^{O(t^{(-1)}(2^{2^n}))} = 2^{2^{n^{o(1)}}}.$$

Obviously, this attempts fails to work since the factor λ in Equation $(\star)$ has no reason to be 1. Still, along the same lines we can show, for example, that SP_{SAT} is not EEXP-complete, where SP_{SAT} is the following problem:

- *input:* a Boolean circuit C of size s representing an arithmetic circuit D of size $\leq 2^s$ computing an integer $N \in \mathbb{Z}$;
- *question:* is it true that $\forall \phi \in \text{SAT}^{=\log t(n)}, N \equiv 0 \bmod p_\phi$, **and** $\exists \psi \notin \text{SAT}^{=\log t(n)}$ such that $|\psi| \leq 2^n$ and $N \equiv 0 \bmod p_\psi$?

Can we push further and generalise the ideas above to succPIT?

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009 (cit. on p. 30).

[Abo+06] Mateo Aboy, Roberto Hornero, Daniel E. Abásolo, and Daniel Álvarez. “Interpretation of the Lempel-Ziv Complexity Measure in the Context of Biomedical Signal Analysis”. In: *IEEE Trans. Biomed. Engineering* 53.11 (2006), pp. 2282–2288 (cit. on p. 5).

[AJS09] Vikraman Arvind, Pushkar S. Joglekar, and Srikanth Srinivasan. “Arithmetic Circuits and the Hadamard Product of Polynomials”. In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*. 2009, pp. 25–36 (cit. on p. 44).

[All+09] E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. “On the complexity of numerical analysis”. In: *SIAM J. Comput.* 38.5 (2009), pp. 1987–2006 (cit. on p. 45).

[AMS10] Vikraman Arvind, Partha Mukhopadhyay, and Srikanth Srinivasan. “New Results on Noncommutative and Commutative Polynomial Identity Testing”. In: *Computational Complexity* 19.4 (2010), pp. 521–558 (cit. on p. 44).

[Bal+21] Nikhil Balaji, Sylvain Perifel, Mahsa Shirmohammadi, and James Worrell. “Cyclotomic Identity Testing and Applications”. In: *ISSAC 2021: International Symposium on Symbolic and Algebraic Computation*. 2021, pp. 35–42 (cit. on pp. 29, 44).

[Bar00] A. Barvinok. “New Permanent Estimators via Non-Commutative Determinants”. In: *ArXiv Mathematics e-prints* (2000) (cit. on p. 40).

[BC18] V. Becher and O. Carton. “Normal numbers and Computer Science”. In: *Sequences, Groups, and Number Theory*. 2018, pp. 233–269 (cit. on p. 25).

[BCH15] V. Becher, O. Carton, and P. A. Heiber. "Normality and Automata". In: *Journal of Computer and System Sciences* 81 (2015), pp. 1592–1613 (cit. on p. 26).

[Ber84] Stuart J. Berkowitz. "On Computing the Determinant in Small Parallel Time Using a Small Number of Processors". In: *Inf. Process. Lett.* 18.3 (1984), pp. 147–150 (cit. on p. 37).

[Bor09] É. Borel. "Les probabilités dénombrables et leurs applications arithmétiques". In: *Rendiconti Circ. Mat. Palermo* 27 (1909), pp. 247–271 (cit. on pp. 25, 27).

[Bor50] É. Borel. "Sur les chiffres décimaux de $\sqrt{2}$ et divers problèmes de probabilités en chaîne". In: *Comptes rendus de l'Académie des Sciences de Paris* 230 (1950), pp. 591–593 (cit. on p. 27).

[BS83] Walter Baur and Volker Strassen. "The Complexity of Partial Derivatives". In: *Theor. Comput. Sci.* 22 (1983), pp. 317–330 (cit. on p. 34).

[Bür+11] Peter Bürgisser, J. M. Landsberg, Laurent Manivel, and Jerzy Weyman. "An Overview of Mathematical Issues Arising in the Geometric Complexity Theory Approach to $\mathsf{VP} \neq \mathsf{VNP}$". In: *SIAM J. Comput.* 40.4 (2011), pp. 1179–1209 (cit. on p. 2).

[Bür00a] Peter Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory*. Ed. by Springer. 2000 (cit. on p. 31).

[Bür00b] Peter Bürgisser. *Completeness and reduction in algebraic complexity theory*. Vol. 7. Springer-Verlag, 2000, pp. xii+168 (cit. on p. 36).

[BW05] Andrej Bogdanov and Hoeteck Wee. "More on Noncommutative Polynomial Identity Testing". In: *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*. 2005, pp. 92–99 (cit. on p. 43).

[Cha33] D. G. Champernowne. "The construction of decimals normal in the scale of ten". In: *J. London Math. Soc.* 8 (1933), pp. 254–260 (cit. on p. 25).

[Che07] Q. Cheng. "Derandomization of sparse cyclotomic integer zero testing". In: *FOCS'07*. IEEE. 2007, pp. 74–80 (cit. on p. 45).

[CK00] Z.Z. Chen and M.Y Kao. "Reducing randomness via irrational numbers". In: *SIAM J. Comput.* 29.4 (2000), pp. 1247–1256 (cit. on p. 45).

[CKW11] Xi Chen, Neeraj Kayal, and Avi Wigderson. "Partial Derivatives in Arithmetic Complexity and Beyond". In: *Foundations and Trends in Theoretical Computer Science* 6.1-2 (2011), pp. 1–138 (cit. on p. 2).

[CRS03] Steve Chien, Lars Eilstrup Rasmussen, and Alistair Sinclair. "Clifford algebras and approximating the permanent". In: *J. Comput. Syst. Sci.* 67.2 (2003), pp. 263–290 (cit. on p. 40).

[CTV10] Q. Cheng, S. P. Tarasov, and M. N. Vyalyi. "Efficient algorithms for sparse cyclotomic integer zero testing". In: *Theory of Computing Systems* 46.1 (2010), pp. 120–142 (cit. on p. 45).

[Dai+04] J. Dai, J. Lathrop, J. Lutz, and E. Mayordomo. "Finite-State Dimension". In: *Theoretical Computer Science* 310 (2004), pp. 1–33 (cit. on p. 25).

[DM13] H. Davenport and H.L. Montgomery. *Multiplicative Number Theory*. Springer New York, 2013 (cit. on p. 46).

[Dvi+12] Zeev Dvir, Guillaume Malod, Sylvain Perifel, and Amir Yehudayoff. "Separating multilinear branching programs and formulas". In: *STOC 2012, Proceedings of the 44th Symposium on Theory of Computing Conference*. 2012, pp. 615–624 (cit. on p. 29).

[EM92] A. Ehrenfeucht and J. Mycielski. "A pseudorandom sequence—how random is it?" In: *Amer. Math. Monthly* 99 (1992), pp. 373–375 (cit. on p. 27).

[FPV15] Hervé Fournier, Sylvain Perifel, and Rémi de Verclos. "On fixed-polynomial size circuit lower bounds for uniform polynomials in the sense of Valiant". In: *Inf. Comput.* 240 (2015), pp. 31–41 (cit. on pp. 29, 34–36).

[Ge93] G. Ge. "Testing equalities of multiplicative representations in polynomial time". In: *FOCS'93*. IEEE. 1993, pp. 422–426 (cit. on p. 45).

[HIM78] Juris Hartmanis, Neil Immerman, and Stephen R. Mahaney. "One-Way Log-Tape Reductions". In: *19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16-18 October 1978*. 1978, pp. 65–72 (cit. on p. 14).

[HPZ11] Christopher Hoobin, Simon J. Puglisi, and Justin Zobel. "Relative Lempel-Ziv Factorization for Efficient Storage and Retrieval of Web Collections". In: *Proc. VLDB Endow.* 5.3 (2011), pp. 265–273 (cit. on p. 5).

[IK04] H. Iwaniec and E. Kowalski. *Analytic number theory*. Vol. 53. AMS, 2004 (cit. on p. 46).

[Jan08] Maurice J. Jansen. "Lower Bounds for Syntactically Multilinear Algebraic Branching Programs". In: *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*. Vol. 5162. 2008, pp. 407–418 (cit. on p. 38).

[JS12] Maurice J. Jansen and Rahul Santhanam. “Stronger Lower Bounds and Randomness-Hardness Trade-Offs Using Associated Algebraic Complexity Classes”. In: *STACS*. 2012, pp. 519–530 (cit. on p. 35).

[Kär+17] Juha Kärkkäinen, Dominik Kempa, Yuto Nakashima, Simon J. Puglisi, and Arseny M. Shur. “On the Size of Lempel-Ziv and Lyndon Factorizations”. In: *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*. Vol. 66. 2017, 45:1–45:13 (cit. on p. 5).

[Koi96] Pascal Koiran. “Hilbert’s Nullstellensatz is in the Polynomial Hierarchy”. In: *J. Complexity* 12.4 (1996), pp. 273–286 (cit. on pp. 35, 36, 45).

[KS07] John C. Kieffer and W. Szpankowski. “On the Ehrenfeucht-Mycielski Balance Conjecture”. In: *Discrete Mathematics & Theoretical Computer Science* DMTCS Proceedings vol. AH, 2007 Conference on Analysis of Algorithms (AofA 07) (2007) (cit. on p. 28).

[LM13] María López-Valdés and Elvira Mayordomo. “Dimension Is Compression”. In: *Theory Comput. Syst.* 52.1 (2013), pp. 95–112 (cit. on pp. 4, 17).

[LM21] Jack H. Lutz and Elvira Mayordomo. “Computing absolutely normal numbers in nearly linear time”. In: *Inf. Comput.* 281.104746 (2021) (cit. on pp. 5, 27).

[LMP19] Guillaume Lagarde, Guillaume Malod, and Sylvain Perifel. “Non-commutative computations: lower bounds and polynomial identity testing”. In: *Chic. J. Theor. Comput. Sci.* 2019 (2019) (cit. on p. 29).

[LMS15] Nutan Limaye, Guillaume Malod, and Srikanth Srinivasan. “Lower bounds for non-commutative skew circuits”. In: *Electronic Colloquium on Computational Complexity (ECCC)* 22 (2015), p. 22 (cit. on pp. 41, 42).

[Lop06] María Lopéz-Valdés. “Lempel-Ziv Dimension for Lempel-Ziv Compression”. In: *Proceedings of the 31st International Conference on Mathematical Foundations of Computer Science*. 2006, pp. 693–703 (cit. on p. 17).

[LP18] Guillaume Lagarde and Sylvain Perifel. “Lempel-Ziv: a “one-bit catastrophe” but not a tragedy”. In: *SODA 2018, Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. 2018, pp. 1478–1495 (cit. on pp. 5, 8, 16–18, 23, 24).

[LS97] J. Lathrop and M. Strauss. “A Universal Upper Bound on the Performance of the Lempel-Ziv Algorithm on Maliciously-Constructed Data”. In: *Proceedings of the Compression and Complexity of Sequences 1997*. 1997, pp. 123–135 (cit. on pp. 17, 22).

[Lun+92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. "Algebraic Methods for Interactive Proof Systems". In: *J. ACM* 39.4 (1992), pp. 859–868 (cit. on pp. 35, 36).

[Lut03] Jack H. Lutz. "Dimension in Complexity Classes". In: *SIAM J. Comput.* 32.5 (2003), pp. 1236–1259 (cit. on pp. 4, 17).

[LZ76] Abraham Lempel and Jacob Ziv. "On the Complexity of Finite Sequences". In: *IEEE Trans. Information Theory* 22.1 (1976), pp. 75–81 (cit. on p. 22).

[MMP11] Elvira Mayordomo, Philippe Moser, and Sylvain Perifel. "Polylog Space Compression, Pushdown Compression, and Lempel-Ziv Are Incomparable". In: *Theory Comput. Syst.* 48.4 (2011), pp. 731–766 (cit. on pp. 5, 8, 11–13, 15, 16).

[MP08] Guillaume Malod and Natacha Portier. "Characterizing Valiant's algebraic complexity classes". In: *J. Complex.* 24.1 (2008), pp. 16–38 (cit. on p. 37).

[MV97] Meena Mahajan and V. Vinay. "Determinant: Combinatorics, Algorithms, and Complexity". In: *Chic. J. Theor. Comput. Sci.* 1997 (1997) (cit. on p. 37).

[Nis91] Noam Nisan. "Lower Bounds for Non-Commutative Computation (Extended Abstract)". In: *Proceedings of the 23rd ACM Symposium on Theory of Computing, ACM Press*. 1991, pp. 410–418 (cit. on pp. 38, 40, 41).

[Pla84] D. A. Plaisted. "New NP-hard and NP-complete polynomial and integer divisibility problems". In: *TCS* 31.1-2 (1984), pp. 125–138 (cit. on p. 45).

[PS00] Larry A. Pierce II and Paul C. Shields. "Sequences Incompressible by SLZ (LZW), Yet Fully Compressible by ULZ". In: *Numbers, Information and Complexity*. 2000, pp. 385–390 (cit. on pp. 17, 22).

[PY86] Christos H. Papadimitriou and Mihalis Yannakakis. "A Note on Succinct Representations of Graphs". In: *Inf. Control.* 71.3 (1986), pp. 181–185 (cit. on pp. 47, 48).

[Raz04a] R. Raz. "Multi-linear formulas for permanent and determinant are of super-polynomial size". In: *Proceedings of the 36th Annual STOC*. 2004, pp. 633–641 (cit. on pp. 36–38, 40).

[Raz04b] R. Raz. "Multilinear $\mathrm{NC}_1 \neq$ Multilinear NC_2". In: *Proceedings of the 45th Annual FOCS*. 2004, pp. 344–351 (cit. on pp. 36–38).

[RS05] Ran Raz and Amir Shpilka. "Deterministic Polynomial Identity Testing in Non-commutative Models". In: *Comput. Complex.* 14.1 (2005), pp. 1–19 (cit. on p. 44).

[RSY08] R. Raz, A. Shpilka, and A. Yehudayoff. "A lower bound for the size of syntactically multilinear arithmetic circuits". In: *SIAM J. on Computing* 38.4 (2008), pp. 1624–1647 (cit. on p. 36).

[RY08] R. Raz and A. Yehudayoff. "Balancing syntactically multilinear arithmetic circuits". In: *Computational Complexity* 17.4 (2008), pp. 515–535 (cit. on pp. 36, 38).

[RY09] R. Raz and A. Yehudayoff. "Lower Bounds and Separations for Constant Depth Multilinear Circuits". In: *Computational Complexity* 18.2 (2009), pp. 171–207 (cit. on pp. 36, 37).

[Sam42] Paul A. Samuelson. "A Method of Determining Explicitly the Coefficients of the Characteristic Equation". In: *Annals of Mathematical Statistics* 13 (1942), pp. 424–429 (cit. on p. 37).

[Sch78] Claus-Peter Schnorr. "Improved Lower Bounds on the Number of Multiplications/Divisions which are Necessary of Evaluate Polynomials". In: *Theor. Comput. Sci.* 7 (1978), pp. 251–261 (cit. on p. 34).

[Sch80] Jack T. Schwartz. "Fast Probabilistic Algorithms for Verification of Polynomial Identities". In: *J. ACM* 27.4 (1980), pp. 701–717 (cit. on p. 33).

[SS72] C. P. Schnorr and H. Stimm. "Endliche Automaten und Zufallsfolgen". In: *Acta Informatica* 1 (1972), pp. 345–359 (cit. on p. 25).

[SY10a] A. Shpilka and A. Yehudayoff. "Arithmetic circuits: A survey of recent results and open questions". In: *Found. Trends Theor. Comput. Sci.* 5 (2010), pp. 207–388 (cit. on p. 40).

[SY10b] Amir Shpilka and Amir Yehudayoff. "Arithmetic Circuits: A survey of recent results and open questions". In: *Foundations and Trends in Theoretical Computer Science* 5.3-4 (2010), pp. 207–388 (cit. on pp. 2, 30).

[Tod91] Seinosuke Toda. "PP is as Hard as the Polynomial-Time Hierarchy". In: *SIAM J. Comput.* 20.5 (1991), pp. 865–877 (cit. on p. 32).

[Tod92] Seinosuke Toda. "Classes of Arithmetic Circuits Capturing the Complexity of Computing the Determinant". In: *IEICE Transactions on Information and Systems* 75 (1992), pp. 116–124 (cit. on p. 37).

[Val79] L. G. Valiant. “Completeness Classes in Algebra”. In: *STOC ’79: Proceedings of the eleventh annual ACM symposium on Theory of computing*. 1979, pp. 249–261 (cit. on pp. 37, 38).

[Zha+09] Yi Zhang, Junkang Hao, Changjie Zhou, and Kai Chang. “Normalized Lempel-Ziv complexity and its application in bio-sequence analysis”. In: *Journal of Mathematical Chemistry* 46.4 (2009), pp. 1203–1212 (cit. on p. 5).

[Zip79] Richard Zippel. “Probabilistic algorithms for sparse polynomials”. In: *EUROSAM*. Vol. 72. 1979, pp. 216–226 (cit. on p. 33).

[ZL77] J. Ziv and A. Lempel. “A Universal Algorithm for Sequential Data Compression”. In: *IEEE Trans. Inf. Theor.* 23.3 (1977), pp. 337–343 (cit. on p. 5).

[ZL78] Jacob Ziv and Abraham Lempel. “Compression of individual sequences via variable-rate coding”. In: *IEEE Trans. Inf. Theory* 24.5 (1978), pp. 530–536 (cit. on pp. 5, 9, 10).

Index