Université Sorbonne Paris Cité

université PARIS DIDEROT PARIS 7

Thèse de doctorat de l'Université Sorbonne Paris Cité Préparée à l'Université Paris Diderot

ÉCOLE DOCTORALE DE SCIENCES MATHÉMATIQUES DE PARIS CENTRE (ED 386)

Institut de Recherche en Informatique Fondamentale (IRIF)

Spécialité Informatique

Contributions to Arithmetic Complexity and Compression

Par

Guillaume LAGARDE

Directeurs de thèse : **Sophie LAPLANTE** et **Sylvain PERIFEL**

Soutenue publiquement à Paris le 5 juillet 2018 devant le jury constitué de :

Hervé FOURNIER	MC	IMJ-PRG	*Examinateur*
Sophie LAPLANTE	PU	IRIF	*Directrice*
Meena MAHAJAN	PU	Institute of Mathematical Sciences	*Rapporteuse*
Claire MATHIEU	DR	CNRS & ENS	*Examinatrice*
Dieter van MELKEBEEK	PU	University of Wisconsin-Madison	*Rapporteur*
Sylvain PERIFEL	MC	IRIF	*Directeur*
Olivier SERRE	DR	CNRS & IRIF	*Président du jury*
Tatiana STARIKOVSKAYA	MC	ENS	*Examinatrice*

Abstract

This thesis explores two territories of computer science: complexity and compression. More precisely, in a first part, we investigate the power of non-commutative arithmetic circuits, which compute multivariate non-commutative polynomials. For that, we introduce various models of computation that are restricted in the way they are allowed to compute monomials. These models generalize previous ones that have been widely studied, such as algebraic branching programs. The results are of three different types. First, we give strong lower bounds on the number of arithmetic operations needed to compute some polynomials such as the determinant or the permanent. Second, we design some deterministic polynomial-time algorithm to solve the white-box polynomial identity testing problem. Third, we exhibit a link between automata theory and non-commutative arithmetic circuits that allows us to derive some old and new tight lower bounds for some classes of non-commutative circuits, using a measure based on the rank of a so-called Hankel matrix. A second part is concerned with the analysis of the data compression algorithm called Lempel-Ziv. Although this algorithm is widely used in practice, we know little about its stability. Our main result is to show that an infinite word compressible by LZ'78 can become incompressible by adding a single bit in front of it, thus closing a question proposed by Jack Lutz in the late 90s under the name "one-bit catastrophe". We also give tight bounds on the maximal possible variation between the compression ratio of a finite word and its perturbation—when one bit is added in front of it.

Keywords: algebraic complexity, lower bounds, polynomial identity testing, arithmetic circuits, Lempel-Ziv, compression.

Résumé

Cette thèse explore deux territoires distincts de l'informatique fondamentale : la complexité et la compression. Plus précisément, dans une première partie, nous étudions la puissance des circuits arithmétiques non commutatifs, qui calculent des polynômes non commutatifs en plusieurs indéterminées. Pour cela, nous introduisons plusieurs modèles de calcul, restreints dans leur manière de calculer les monômes. Ces modèles en généralisent d'autres, plus anciens et largement étudiés, comme les programmes à branchements. Les résultats sont de trois sortes. Premièrement, nous donnons des bornes inférieures sur le nombre d'opérations arithmétiques nécessaires au calcul de certains polynômes tels que le déterminant ou encore le permanent. Deuxièment, nous concevons des algorithmes déterministes fonctionnant en temps polynomial pour résoudre le problème du test d'identité polynomiale. Enfin, nous construisons un pont entre la théorie des automates et les circuits arithmétiques non commutatifs, ce qui nous permet de dériver de nouvelles bornes inférieures en utilisant une mesure reposant sur le rang de la matrice dite de Hankel, provenant de la théorie des automates. Une deuxième partie concerne l'analyse de l'algorithme de compression sans perte Lempel-Ziv. Pourtant très utilisé, sa stabilité est encore mal établie. Vers la fin des années 90s, Jack Lutz popularise la question suivante, connue sous le nom de « one-bit catastrophe » : « étant donné un mot compressible, est-il possible de le rendre incompressible en ne changeant qu'un seul bit ? ». Nous montrons qu'une telle catastrophe est en effet possible. Plus précisément, en donnant des bornes optimales sur la variation de la taille de la compression, nous montrons qu'un mot « très compressible » restera toujours compressible après modification d'un bit, mais que certains mots « peu compressibles » deviennent en effet incompressibles.

Mot-clés : complexité algébrique, bornes inférieures, test d'identité polynomiale, circuits arithmétiques, Lempel-Ziv, compression.

Contents

Acknowledgements

> « Un voyage se passe de motifs. Il ne tarde pas à prouver qu'il se suffit à lui-même. On croit qu'on va faire un voyage, mais bientôt c'est le voyage qui vous fait, ou vous défait. »
>
> Nicolas Bouvier. *L'usage du monde.*

Ce délicat voyage n'aurait pas été si agréable sans la rencontre et la présence de nombreuses personnes.

Je me dois avant tout de remercier Sylvain et Sophie, mes deux directeurs de thèse. Sylvain, merci infiniment de m'avoir guidé et accordé temps, confiance et indépendance durant ces trois années de thèse, sans jamais hésiter à mettre les mains dans le cambouis lorsqu'il le fallait. L'aventure « Lempel-Ziv » me rappellera pendant longtemps de bons souvenirs, malgré les nombreux "stack overflows" qu'elle a pu engendrer. Sophie, merci pour ton aide précieuse dans les moments qui m'étaient importants, et les nombreuses et intéressantes discussions. Je ne désespère toujours pas de trouver un lien entre circuits arithmétiques et communication.

Thanks to Meena Mahajan and Dieter van Melkebeek for having accepted the tedious and time-consuming task of reviewing this manuscript, and for the improvements you suggested. Merci aussi à Hervé Fournier, Claire Mathieu, Olivier Serre et Tatiana Starikovskaya de me faire l'honneur d'être membres du jury de soutenance.

Cette thèse et moi-même devons beaucoup aux personnes avec qui j'ai eu l'immense privilège de collaborer. Guillaume Malod et Hervé Fournier, et plus récemment Arpita Korwar, merci pour toutes ces heures passées devant le tableau blanc (ne désespérons pas, un jour ces feutres fonctionneront et s'effaceront sans rechigner !) et votre bonne humeur inaltérable. Special thanks to Nutan Limaye and Srikanth Srinivasan who were awesome hosts during my visit at IIT Bombay, whether it be on a scientific or a personal level. Merci à Nathanaël Fijalkow et Pierre Ohlmann, c'était un réel plaisir de toucher à vos fameux automates, pas

si éloignés de nos circuits après tout. Thank you Abhishek Methuku, for being incredibly enthusiastic and optimistic during that hot summer in Budapest, when we were working on combinatorial questions while drinking sparkling water. Merci à Pierre Aboulker de m'avoir fait rechercher des lignes dans des graphes. Plus récemment, merci Vincent Cohen-Addad de partager ta connaissance et tes problèmes sur le clustering hiérarchique.

Merci aussi à tous les membres de l'IRIF de créer cette atmosphère si particulière, curieux mélange d'aménité et de profondeur scientifique, mais aussi et surtout de rendre les pauses café si agréables. Aux (ex- pour certains) doctorants, postdocs et ATER, en particulier : Alex, Khaled, Bruno, Jehanne, Florent, Brieuc, Pablo, Charles, Laurent, Yassine, Simon, Clément, Alessandro. À Alexandre et Lucas, vous avez été les meilleurs co-bro imaginables, et surtout des partenaires coriaces de tournois de Non-deterministic Boolean Analysis comme rarement on en voit de nos jours. Un grand merci aussi à l'équipe administrative qui a été d'une efficacité sans faille : Houy, Laïfa, Odile, Dieneba, Etienne et Nicolas (ou, devrais-je dire, sauveteur de l'extrême dans les situations administrato-sphériques les plus compliquées).

Je dois également énormément à mes professeurs de prépa et de l'ÉNS de Lyon qui, en plus de me donner une raison d'aller en cours, m'ont fait don de leur vision si singulière des mathématiques et de l'informatique; Alain Juhel, Jean Voedts, Alexandre Miquel, Daniel Hirschkoff, Stéphan Thomassé, Pascal Koiran, merci à vous. Alexandre, un merci particulier pour ton précieux soutien lors de la préparation de mon voyage sabbatique.

Thanks also to Gyula O.H. Katona for a wonderful summer internship and winter visit; this is where it all really started.

*
* *

Enfin, ces trois années de thèse n'auraient pas eu la même saveur sans les « extra-académiques », amis dont la présence, qu'elle soit constante ou elliptique, compte énormément pour moi. Aux lyonnais d'un jour ou de toujours, désormais éparpillés dans le monde entier : Rémi l'incarnation moderne de Cyrano, Qian "pseudo-sœur" chinoise, Paul-Elliot, Simon, Gauthier et sa passion incommensurable de la poésie allemande, Jean-Florent, Fabrice, Gabrielle, Florent mon faux-jumeau scientifique et littéraire (©), Arnaud, Tot, Lucas, Nicoo. À la team *jeux* ∪ *badminton* : Romain, Raphaël, Mathilde, Mathieu et Véronique. Aux anciens de prépa : Damien hipster avant l'heure, Colin, Pierre-Etienne, Souheila. À ceux que je connais depuis si longtemps que j'ai arrêté de compter les années : Mathieu, Emilie, Claire. À ceux qui ne rentrent pas dans les cases : Amélie pour les soirées surréalistes et les tribulations garguantuesques. Maud pour les footings

matinaux et les exquises quiches[1] que vous pourrez probablement tester d'ici 1h45 environ, à ce stade.

*
* *

Merci à ma belle-famille pour leur chaleureux accueil et la quantité astronomique de marocchinos consommée.

Un immense merci à ma famille, et en particulier à mes parents, pour leur soutien inconditionnel et leurs conseils avisés depuis bon nombre d'années maintenant.

Enfin, merci à Anne d'être infiniment patiente et de rayonner au quotidien.

[1] Prononciation rapide non triviale.

Prelude

> « Par ma foi ! il y a plus de quarante ans que je dis de la prose sans que j'en susse rien, et je vous suis le plus obligé du monde de m'avoir appris cela. »
>
> Molière. *Le Bourgeois gentilhomme.*

Just as *Monsieur Jourdain* was surprised and delighted to learn that he had been using prose for years without knowing it, most of us are not conscious that we are instinctively practising *computer science* on a daily basis. Whenever you wonder if that very hard climbing route could even be physically feasible by a mere human being, you are trying to solve what is called a *computability* problem. You—a *model of computation*—try to solve a given task—*climbing the route.* After a proof that it is indeed the case—you, or a more experienced friend, finally climbed it!—, you might wonder if the huge number of moves you made were really necessary and try to decrease this number to be as close to the minimum theoretically attainable as possible, in order to save your energy next time. At that point, you changed the point of view from computability to *algorithmics* and *complexity*. Ultimately, you get an efficient way of climbing this route, and it is now the right time to explain your solution to someone else. If the sequence of moves you did was the following[2]:

- *move right hand*, *move left foot*, *move left hand*, *move right foot*, *move right hand*, *move left foot*, *move left hand*, *move right foot*, *move right hand*, *move left foot*, *move left hand*, *move right foot*, *move right hand*, *move left foot*, *move left hand*, *move right foot*, *move right hand*, *move left foot*, *move left hand*, *move right foot*, *move right hand*, *move left foot*, *move left hand*, *move right foot*, *move right hand*, *move left foot*, *move left hand*, *move right foot*, *move right hand*, *move left foot*, *move left hand*, *move right foot*, *move right hand*, *move left foot*, *move left hand*, *move right foot*,

[2] Don't ask me what kind of crazy route it can be.

then you would probably say "it consists of a sequence of "*move right hand*, *move left foot*, *move left hand*, *move right foot*" repeated 9 times", without enumerating—or so I hope, for your friend—the whole sequence move by move. Congratulation, this time, you just made what is called a *compression*: you described your action with a clever and compact method.

This short and rather informal story introduced the two main territories studied in this thesis: *complexity* and *compression*.

In the first part of this manuscript, we focus on complexity questions in a domain called arithmetic complexity, while in the second part we concentrate on compression questions with the study of a particular and well-used data compression method called *Lempel-Ziv*.

Part one: Non-commutative Arithmetic Circuits

The aim of *complexity theory* is to understand what are the problems we can solve with a limited amount of resources. These ressources are quite often *time* and *space*, as they are the most natural ones when dealing with computers. But these notions are not immutable and can be exemplified in various ways depending on the model of computation and the measure that is suitable for what is wanted. Two examples among others: by "time", we can denote the number of steps of a Turing machine, but we can also denote the depth of a boolean circuit.

Arithmetic complexity is the algebraic analogue of boolean complexity with an algebraic flavor. Boolean functions are replaced by *multivariate polynomials*, which are the core objects of the theory. The way we compute them is through a model called *arithmetic circuits*, the definition of which is similar to that of boolean circuits and is stated later in this manuscript. The motivation for this is twofold: first, the study of polynomial computations arises in many places in computer science and naturally leads to such questions, and second, by moving to a more structured world than boolean functions, it might be possible to use tools from mathematics like *linear algebra, algebraic geometry, etc.*, to tackle the problem of finding good *lower bounds*, a problem known to be hard.

The contribution of this thesis to *arithmetic complexity* lies in the non-commutative setting, a world where variables do not commute. In this setting, we explore circuits that are restricted in the way they are allowed to compute monomials (a more precise definition will be stated later in the manuscript, with the notion of *parse trees*). The results are of three different types:

- We give *lower bounds* for various models of non-commutative computations, that is, we show that some polynomials require a large number of arithmetic operations to be computed:
  - for circuits with a unique parse tree ("UPT circuits") through a measure

that characterises exactly their complexity. This extends a work by Nisan [38] for algebraic branching programs.

  - for circuits that allow up to an exponential number of parse trees ("rot-PT circuits" and "k-PT circuits"). The aim of this is to get lower bounds for models of computation that are closer to general non-commutative circuits.

  - for homogeneous formulas that allow slightly less parse trees than the maximum possible (that is, that allow up to $2^{o(d)}$ parse trees, where d corresponds to the degree of the polynomial) computing $\mathrm{IMM}_{n,d}$—a polynomial that corresponds to matrix multiplication and that is complete for the important model of computation called *arithmetic branching programs*. This makes some progress towards a separation between non-commutative formulas and algebraic branching programs, a famous open problem in the non-commutative setting.

- We provide deterministic polynomial time algorithms to solve the important *white-box polynomial identity testing* problem for some classes of circuits; that is, we design algorithms to decide efficiently whether a circuit within a given class computes the formal zero polynomial:

  - for UPT circuits, through two adapations and extensions of previous algorithms for algebraic branching programs, due to Raz and Shpilka [42] and Arvind, Joglekar and Srinivasan [4].

  - for constant sum of UPT circuits. This generalises a similar result that was obtained for sum of *read once algebraic branching programs (ROABPs)* by Gurjar, Korwar, Saxena and Thierauf [15].

- We construct a bridge between *automata theory* and *arithmetic circuits*. More precisely, we show that non-commutative algebraic branching programs are equivalent to acyclic weighted automata over words, and that non-commutative unique parse tree circuits are equivalent to layered weighted automata over trees. Subsequently, this correspondence—together with the use of fundamental theorems from automata theory—allow us to derive some old and new tight lower bounds for some classes of non-commutative arithmetic circuits.

Figures 1 and 2 (roughly) represent what was known before and what is known now in the restricted context of this thesis.

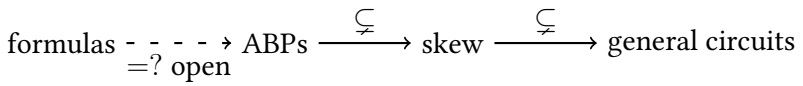


Figure 1: Before

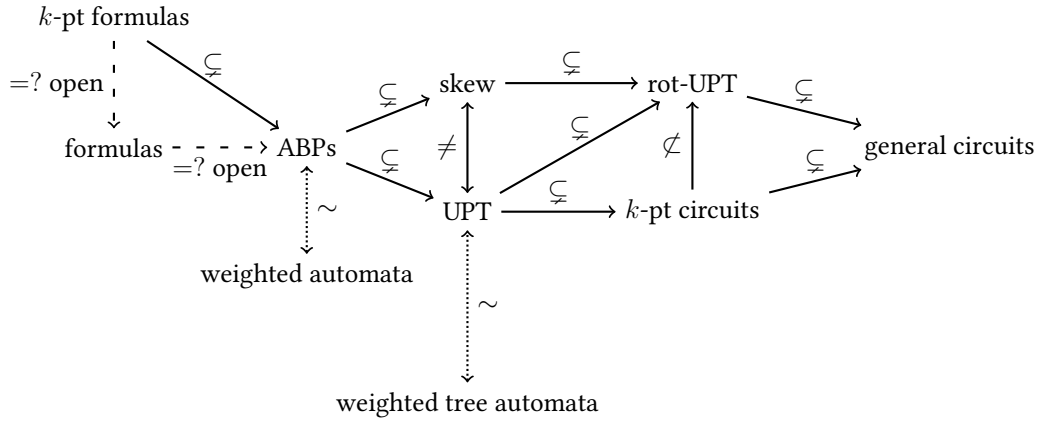


Figure 2: Now

Part two: Lempel-Ziv, a "One-bit catastrophe" but not a tragedy

Data compression is the art of finding a concise bit description of the information in order to save storage resources or to reduce the use of the network bandwidth for example. All the methods are not equivalent, some are without loss of information—lossless—, as for the text, others slightly deteriorate the quality—lossy— as for musics or videos for example, in exchange for a better compression ratio.

Lempel-Ziv algorithms are among the most popular compression algorithms. They refer to a series of lossless techniques—all based on a notion of dictionary which is constructed during the compression process—that can work on any file format. Introduced by Abraham Lempel and Jacob Ziv in 1977 and 1978, starting with two methods called *LZ'77* and *LZ'78*, they are widely used in practice as key ingredients in various places such as *deflate*, *gif*, *gzip*, etc, but were also the starting point of a long line of theoretical research—some references of which can be found in Chapter 7. Yet, their behavior and robustness are still not well understood. While it is reasonable to expect a certain stability from a data compression algorithm against small perturbation on the input, Jack Lutz, in the late '90s, asked the following: "When using LZ'78, is it possible to change the compression ratio of an infinite word by adding a single bit in front of it?". This question, known as "one-bit catastrophe" question was still unanswered.

The main contribution of this thesis to compression is to give a positive answer

to this question. But before proving that, we investigate the behavior of LZ'78 on finite words and get the following:

- We give an *upper bound* on the maximal variation possible between the compression ratio of a finite word and its variant—when one bit is added in front of it.
- We give *constructions* that show that the previous upper bound is tight up to a multiplicative constant.

A catastrophe for infinite word—that is, a compressible word that becomes incompressible after we add one bit in front of it—is then derived from the results on finite words.

Publications

The results presented in this thesis can be found in the following papers:

- Guillaume Lagarde, Guillaume Malod, and Sylvain Perifel. Non-commutative computations: lower bounds and polynomial identity testing. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:94, 2016
- Guillaume Lagarde, Nutan Limaye, and Srikanth Srinivasan. Lower bounds and PIT for non-commutative arithmetic circuits with restricted parse trees. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, pages 41:1–41:14, 2017
- Guillaume Lagarde, Nutan Limaye, and Srikanth Srinivasan. Lower bounds and PIT for non-commutative arithmetic circuits with restricted parse trees (extended version). *To appear in Computational Complexity*
- Nathanaël Fijalkow, Guillaume Lagarde, and Pierre Ohlmann. Tight bounds using hankel matrix for arithmetic circuits with unique parse trees. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:38, 2018
- Guillaume Lagarde and Sylvain Perifel. Lempel-ziv: a "one-bit catastrophe" but not a tragedy. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1478–1495, 2018

Part I

Non-commutative Arithmetic Circuits

Overview

This part is dedicated to lower bounds for non-commutative arithmetic circuits.

- In Chapter 1, we introduce the standard definitions related to arithmetic circuits that will be used throughout the entire part.
- Chapter 2 is a complete study of circuits with a unique parse tree ("UPT circuits").
- Chapter 3 shows lower bounds for circuits with up to an exponential number of parse trees ("rot-PT circuits" and "k-PT circuits").
- Chapter 4 makes progress towards a separation between formulas and algebraic branching programs in the non-commutative setting. More precisely, using similar ideas to that of Chapter 3, we show some lower bounds for formulas—with a restricted number of parse trees—computing the *iterated matrix multiplication* polynomial.
- Chapter 5 is devoted to deterministic and polynomial time algorithms for PIT—for UPT circuits, as well as for constant sum of UPT circuits—a decision problem closely related to lower bounds.
- Chapter 6 makes a bridge between some non-commutative classes of circuits and weighted automata. This bridge gives a way to derive or improve already known results in non-commutative lower bounds as consequences of theorems from automata theory based on *Hankel matrices*.

Chapter 1

Preliminaries

« O tôt matin du commencement !
O souffle du vent, qui vient
Des rives nouvelles ! »

Bertolt Brecht. *La vie de Galilée.*

1.1 Arithmetic complexity

Arithmetic circuits

The most natural strategy to compute a target polynomial is to use the operations from the algebra $\mathbb{F}[X]$, that is $+$ and $\times$, together with the use of the constants from the field $\mathbb{F}$. This strategy is exactly captured by *arithmetic circuits*. More formally:

Definition 1.1: Arithmetic circuit

An arithmetic circuit is an acyclic directed graph where gates of in-degree greater than zero are labeled by $+$ or $\times$ and gates of in-degree zero, called the *inputs* of the circuit, are labeled by either a variable or a constant coming from $\mathbb{F}$. Each gate Φ represents in the natural way a formal polynomial that is denoted by P_Φ. The unique gate of out-degree zero is called the *output* of the circuit and we say that the polynomial computed (or represented) by the circuit is the polynomial computed at this gate; for a particular circuit C, this polynomial is denoted by P_C.

The *size* of an arithmetic circuits is defined as the number of wires. Sometimes, it will be more convenient to consider instead the number of nodes; in this case,

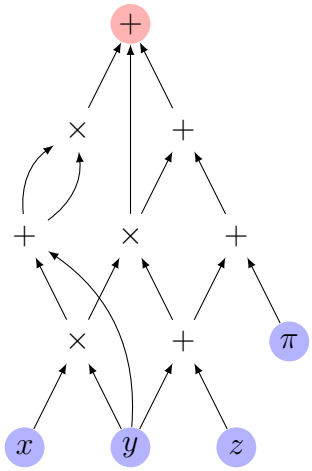


Figure 1.1: Example of an arithmetic circuit of depth 5. The blue gates are the inputs. The red gate is the output. The number of nodes is 12. The number of edges is 17.

we will explicitly mention that the measure we take is the number of nodes. Observe that the two measures are equivalent up to a constant factor as long as the gates are of bounded fan-in. The *depth* of a circuit is the size of the longest path from the output to an input: this can be seen as a measure of how well you can parallelize the computation of the polynomial represented by your circuit.

What are we looking for?

The questions are mainly of two kinds:

- **Polynomials' point of view** "Given a polynomial f, is there a circuit that computes f with some properties on the circuit?" In particular, the **lower bound** question falls in this category: find explicit[1] polynomials that require large circuits to be computed (i.e., the number of arithmetic operations needed to compute this polynomial is large). Large meaning most often superpolynomial in the number of variables and the degree of the polynomial. This is the quest of looking for intractable polynomials.

- **Circuits' point of view** Circuits can be used as a compact representation for polynomials since an arithmetic circuit C can compute a polynomial that has an exponential number of non-zero monomials in the size of C. See Figure 1.2 for an example. A natural question is: how to handle

[1] *explicit* meaning in general that coefficients are computable by a reasonable algorithm (in P for example)

them efficiently; in others words: "Given a circuit C, does the polynomial computed by C satisfy a particular property?" Eg: Is the degree of the polynomial greater than 42? Is the polynomial divisible by $x^7 - y$, or by another polynomial also given by an arithmetic circuit? Of course, to answer this kind of questions, you can always develop explicitly the polynomial computed by the circuit but this would not yield an efficient algorithm since polynomial-sized circuits can represent polynomials with an exponential number of non-zero monomials, as the previous example shows. The most representative question that falls within this category is the famous **polynomial identity testing problem (PIT)**: given a circuit C, decide if the polynomial computed by C is formally zero. PIT will be considered in more detail in Chapter 5

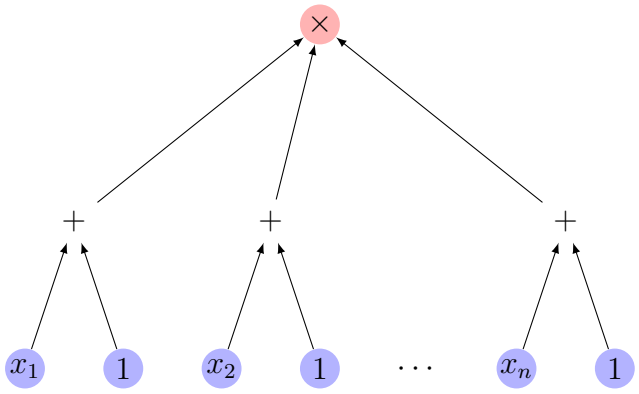


Figure 1.2: Arithmetic circuit of size $3n$ that computes $\prod_{i=1}^{n}(x_i + 1)$, which is a polynomial with 2^n non-zero monomials.

But reality is cruel. Although the ultimate goal for the first kind of questions is to find explicit polynomials for which we can prove superpolynomial lower bounds against general circuits, the best up to now is the following theorem:

Theorem 1.2: Baur and Strassen, 1983

For all $n, d \in \mathbb{N}$, $\sum_{i=1}^{n} x_i^d$ requires circuits of size $\Omega(n \log d)$ to be computed.

The situation is not better for PIT, for which there is a polynomial time randomized algorithm that follows from the Schwartz-Zippel lemma but no subexponential time deterministic algorithm is known for general circuits.

In fact, these two questions (lower bounds and PIT) are strongly related; solving PIT is similar to killing two birds with one stone. Indeed, in 2003, Kabanets

and Impagliazzo [19] proved a beautiful result showing—more or less—that "Derandomizing polynomial identity tests means proving circuit lower bounds". More precisely, a deterministic polynomial-time algorithm to solve PIT implies either a superpolynomial lower bound on the size of arithmetic circuits computing the permanenent or $\mathsf{NEXP} \not\subset \mathsf{P/poly}$. A pessimistic way to interpret this theorem is that this is also a hint that partially explains why PIT seems very hard to solve in P and out of reach so far: "because lower bounds are".

Restrictions

Although lower bounds for general circuits seem out of reach so far, by making some natural restrictions on the circuits, we sometimes obtain stronger lower bounds—even exponential ones. These restrictions are useful to better understand the computation of polynomials by capturing some models of computation that are more suitable for a particular situation. For example, in [18], it is proved that every monotone circuits (that is, arithmetic circuits that use only positive elements from an ordered field, such as $\mathbb{R}$) computing the permanent of an $n \times n$ matrix has size $2^{\Omega(n)}$.

The more the circuits are constrained, the easier it is to provide lower bounds and design good algorithms for PIT. We give below some of the restrictions that will be used in this manuscript, but notice that there are quite a lot of others such as *multilinear* circuits, *syntactically multilinear* circuits, *monotone* circuits, ...

- **Formulas** Circuits where the underlying graph is a tree. Equivalently, a formula is a circuit where the fan-out of each gate is at most 1. Intuitively, this corresponds to polynomial computations where a computation step can be used at most one.
- **Skew circuits** Circuits where the $\times$ gates have at most one non input child.
- $\Sigma\Pi\Sigma$ **circuits** Circuits of depth three, starting with a layer of $+$ gates, then a layer of $\times$ gates, and a final $+$ gate.
- **Constant depth circuits** Circuits where the length of any path from the output to an input is bounded by a constant.

Interesting polynomials

Some polynomials receive more attention than others; a first reason that explains this phenomenon is the fact that a polynomial can capture completely the "complexity" of an arithmetic class of polynomials/circuits; a second one is due to

the important consequences that would follow from a large enough lower bound proof for them. Below is the presentation of three famous polynomials; we will see some others later in the manuscript.

- **Determinant and Permanent**

  For $X = \{x_{1,1}, x_{1,2}, \ldots, x_{1,n}, \ldots, x_{n,1}, \ldots, x_{n,n}\}$ a set of n^2 variables, we define the determinant and the permanent as

  $$\mathrm{DET}_n(X) = \sum_{\sigma \in S_n} sgn(\sigma) \prod_{i=1}^{n} x_{i,\sigma(i)}$$

  and

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

  where the sums are over all permutations σ of the set $\{1, 2, \ldots, n\}$ and $sgn(\sigma)$ stands for the signature of the permutation σ. Although the two definitions are very close, the permanent has much fewer properties than the determinant. The determinant has a beautiful geometric interpretation in terms of volumes, but the permnanent seems to have only a combinatorial flavor. It is believed that the permanent is not computable by circuits of polynomial size, whereas it is well known that the determinant is computable by small circuits, for example by using the Gaussian elimination algorithm and deleting the divisions that appear in the process.

  In fact, proving the two polynomials are of distinct complexity can roughly be seen as an algebraic variant of the famous $\mathsf{P} = \mathsf{NP}?$ question, namely $\mathsf{VP} = \mathsf{VNP}?$[2]

- **Iterated Matrix Multiplication**

  Assume that $N = n^2 \cdot d$ for positive $n, d \in \mathbb{N}$ and let $\mathrm{IMM}_{n,d}(X)$ denote the following polynomial (called the *Iterated Matrix Multiplication polynomial* of parameters n and d) in N variables. Assume X is partitioned into d sets of variables $X_1, \ldots, X_d$ of size n^2 each and let $M_1, \ldots, M_d$ be $n \times n$ matrices such that the entries of M_i ($i \in [d]$) are distinct variables in X_i. Let $M = M_1 \cdot M_2 \cdots M_d$ be the multiplication of the d matrices; each entry of M is a homogeneous polynomial of degree d from $\mathbb{F}[X]$. We define the polynomial $\mathrm{IMM}_{n,d}$ to be the sum of the diagonal entries of M[3].

[2] In fact, the VP versus VNP question is closer to the LOGCFL versus $\#\mathsf{P}$ question.

[3] This is not exactly the standard definition of $IMM_{n,d}$ which is in general, defined as the polynomial in the first row and column of the matrix $M_1 \cdot M_2 \cdots M_d$. However, taking the trace of the matrix gives a more symetric definition and help in writing cleaner statements.

As we shall see later, this important polynomial is completely captured by the model of computation called *Algebraic Branching Programs (ABPs in short)*.

Example 1.3

For $n = 2, d = 3$ and sets $X_i = \{x^i_{1,1}, \ldots, x^i_{2,2}\}$

$$\mathrm{IMM}_{2,3}(X_1, X_2, X_3) = \mathrm{Tr}\left(\begin{bmatrix} x^1_{1,1} & x^1_{1,2} \\ x^1_{2,1} & x^1_{2,2} \end{bmatrix} \times \begin{bmatrix} x^2_{1,1} & x^2_{1,2} \\ x^2_{2,1} & x^2_{2,2} \end{bmatrix} \times \begin{bmatrix} x^3_{1,1} & x^3_{1,2} \\ x^3_{2,1} & x^3_{2,2} \end{bmatrix}\right)$$

$$= \mathrm{Tr}\left(\begin{bmatrix} \sum_{i_1=1}^{2}\sum_{i_2=1}^{2} x^1_{1,i_1}x^2_{i_1,i_2}x^3_{i_2,1} & \star \\ \star & \sum_{i_1=1}^{2}\sum_{i_2=1}^{2} x^1_{2,i_1}x^2_{i_1,i_2}x^3_{i_2,2} \end{bmatrix}\right)$$

$$= \sum_{i_1=1}^{2}\sum_{i_2=1}^{2} x^1_{1,i_1}x^2_{i_1,i_2}x^3_{i_2,1} + \sum_{i_1=1}^{2}\sum_{i_2=1}^{2} x^1_{2,i_1}x^2_{i_1,i_2}x^3_{i_2,2}$$

1.2 Non-commutative setting

From now on, we work over the non-commutative setting, in which x_1x_2 and x_2x_1 are two distinct monomials. The motivation for this is twofold: first, the study of polynomial computations over non-commutative algebras (e.g. when the polynomials are evaluated over the algebra of $k \times k$ matrices over $\mathbb{F}$; or over any non-commutative field such as the quaternions) naturally leads to such questions [10, 9], and second, computing any polynomial non-commutatively is at least as hard as computing it in the commutative setting and thus, the lower bound question should be easier to tackle in this setting.

1.2.1 Non-commutative polynomials

We use $X = \{x_1, \ldots, x_n\}$ to denote the set of variables. Unless explicitly stated, we work over the algebra of *non-commutative polynomials* (also known as *free algebra*), written $(\mathbb{F}\langle X\rangle, +, \times, .)$ – or just $\mathbb{F}\langle X\rangle$ in short.

- $\mathbb{F}$ is a commutative field.

- $(\mathbb{F}\langle X\rangle, +, .)$ is the vector space of formal and finite linear combinations of strings (called *monomials*) over the alphabet X. Observe that x_1x_2 and x_2x_1 are two distinct monomials.
- $\times$ is a bilinear product defined for two monomials m_1 and m_2 as their concatenation m_1m_2. It is then extended bilinearly to any pairs of polynomials from $\mathbb{F}\langle X\rangle$.

The set of monomials over the alphabet X is written $\mathcal{M}(X)$. Given a polynomial f and m a monomial, we say that m is a *non-zero* monomial if the coefficent associated to m in f is non-zero. Most often, α_m will denote the coefficient associated to the monomial m.

Degree and related definitions

The *degree* of a monomial m, written $\deg(m)$, is the length of the corresponding string. By extension, the degree of a polynomial $f \in \mathbb{F}\langle X\rangle$, written $\deg(f)$, is the maximal degree of a non-zero monomial of f. For $d \in \mathbb{N}$, $\mathcal{M}_d(X)$ will denote the set of monomials of degree exactly d.

A polynomial is said to be *homogeneous* if all the non-zero monomials are of same degree. The *homogeneous component* of degree i of a polynomial f, written $f^{[i]}$, is the sum of all monomials of degree i appearing in f.

Example 1.4

$\mathcal{M}(\{x\}) = \{x^i, \forall i \in \mathbb{N}\}$
$\mathcal{M}_3(\{x, y\}) = \{x^3, x^2y, xy^2, y^3, y^2x, yx^2, xyx, yxy\}$
$f = x_1x_2x_3^3 + x_1^2 + x_1^4 + x_2x_3$ is a non homogeneous polynomial of degree 5 with four non-zero monomials. Its homogeneous component of degree 2 is $f^{[2]} = x_1^2 + x_2x_3$.
$g = x_1x_2x_3 + x_1^3$ is a homogeneous polynomial of degree 3.

j-product

The following notion will be useful to decompose and factor polynomials.

Definition 1.5: j-product of two polynomials

Given homogeneous polynomials $g, h \in \mathbb{F}\langle X\rangle$ of degrees d_g and d_h respectively and an integer $j \in [0, d_h]$, we define the *j-product* of g and h — denoted $g \times_j h$ — as follows:

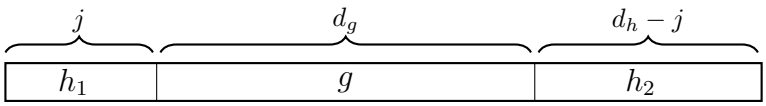


Figure 1.3: j-product of two monomials g and h.

- If g and h are two monomials, then h is uniquely factorised as a product of two monomials $h_1 \cdot h_2$, with $\deg(h_1) = j$ and $\deg(h_2) = d_h - j$. In this case we define $g \times_j h$ to be $h_1 \cdot g \cdot h_2$.
- The map is extended bilinearly to general homogeneous polynomials g, h. Formally, let g, h be general homogeneous polynomials, where $g = \sum_\ell g_\ell$, $h = \sum_i h_i$ and g_ℓ, h_i are monomials of g, h respectively. For $j \in [0, d_h]$, each h_i can be factored uniquely into h_i^1, h_i^2 such that $\deg(h_i^1) = j$ and $\deg(h_i^2) = d_h - j$. And $g \times_j h$ is defined to be $\sum_i \sum_\ell h_i^1 g_\ell h_i^2 = \sum_i h_i^1 g h_i^2$.

Observe that $g \times_0 h = g \cdot h$ and $g \times_{d_h} h = h \cdot g$.

Example 1.6

If $g = x_1 x_2^2$ and $h = x_3 x_1 x_2$, then:

$$g \times_2 h = x_3 x_1^2 x_2^3$$

If $g = x_1 x_2 x_3 + x_2^2 x_1$ and $h = x_3 x_2 + x_2^2$, then:

$$\begin{aligned} g \times_1 h &= x_1 x_2 x_3 \times_1 x_3 x_2 + x_1 x_2 x_3 \times_1 x_2^2 + x_2^2 x_1 \times_1 x_3 x_2 + x_2^2 x_1 \times_1 x_2^2 \\ &= x_3 x_1 x_2 x_3 x_2 + x_2 x_1 x_2 x_3 x_2 + x_3 x_2^2 x_1 x_2 + x_2^3 x_1 x_2 \end{aligned}$$

1.2.2 Non-commutative circuits

In order to capture non-commutativity, we need to slightly change our model of computation. A *non-commutative arithmetic circuit* is an arithmetic circuit where the children of any multiplication gate have been ordered. In this way, a non-commutative arithmetic circuit represents a non-commutative polynomial: the polynomial computed by a $\times$ gate is the product of the polynomials computed by its children, where the product is taken in the given order.

Further, unless mentioned otherwise, we allow both $+$ and $\times$ gates to have unbounded fan-in and $+$ gates to compute arbitrary linear combinations of their

inputs (the input wires to the $+$ gate are labelled by the coefficients of the linear combination). The size of a circuit will be the number of edges. We always assume that the output gate is a $+$ gate and that the input gates feed into $+$ gates. We also assume that $+$ and $\times$ gates alternate on any path from the output gate to an input gate. The reason for this is that any circuit can be converted to one of this form with at most a constant-factor blow-up in size and depth; however, it will be more convenient to work with circuits of this form.

Homogeneity

Most often, our circuits and formulas will be *homogeneous* in the following sense. Define the *formal degree* of a gate in the circuit as follows: the formal degree of an input gate is 1, the formal degree of a $+$ gate is the maximum of the formal degrees of its children, and the formal degree of a $\times$ gate is the sum of the formal degrees of its children. We say that a circuit is homogeneous if each gate computes a homogeneous polynomial and any gate computing a non-zero polynomial computes one of degree equal to the formal degree of the gate. Note, in particular, that every input node is labelled by a variable only (and not by constants from $\mathbb{F}$).

Homogeneity is not a strong assumption on the circuit thanks to the following well known lemma (stated here for multiplication fan-in 2, but any circuit can be converted to have this additional property with a small blow-up in size and by possibly increasing logarithmically the depth).

Lemma 1.7: Homogenization

Any homogeneous polynomial of degree d computed by a non-commutative circuit C of size s with $\times$ fan-in 2 can be computed by a homogeneous circuit of size $O(s.d^2)$.

Proof. We construct a homogeneous circuit C' for f as follows.

- **The gates** of C' are denoted by pairs of the form (Φ, i). For each gate $\Phi \in C$ and for each $i \in [0, d]$, we add a gate (Φ, i) to the circuit C'. We then add edges and additional gates in such a way that $P_{(\Phi,i)}$ will be the homogeneous component of degree i of the polynomial computed by Φ in the circuit C, namely $P_\Phi^{[i]}$. If Γ is the output gate of C, then (Γ, d) is the output gate of C'.

- **Edges:**

- If $\Phi \in C$ is an addition gate with children $\Psi_1, \ldots, \Psi_t$, then for each i, (Φ, i) is an addition gate with children $(\Psi_1, i), \ldots, (\Psi_t, i)$.
- If $\Phi \in C$ is a multiplication gate with children Ψ_1, Ψ_2 (in this order), then $P_\Phi^{[i]} = \sum_{j \leq i} P_{(\Psi,j)} \times P_{(\Psi,i-j)}$. Therefore, we set (Φ, i) to be an addition gate. We then add $i+1$ multiplication gates to the circuit, each corresponding to one of the $i+1$ products $P_{(\Psi,j)} \times P_{(\Psi,i-j)}$, and we add these gates as children of (Φ, i).

By induction, it is easy to see that C' computes f and is of size $O(s.d^2)$.

□

1.3 Parse trees restriction

If we pretend the multiplication to be non associative, a non-commutative monomial can be computed in different ways that depend on how the parentheses are set. For example, if we restrict the fan-in to be 2, the monomial $x_1x_2x_3x_4$ can be basically computed by five different non-commutative circuits, one for each of the following possible setting of the parentheses:

- $(x_1 \cdot x_2) \cdot (x_3 \cdot x_4)$
- $(x_1 \cdot (x_2 \cdot x_3)) \cdot x_4$
- $x_1 \cdot ((x_2 \cdot x_3) \cdot x_4)$
- $((x_1 \cdot x_2) \cdot x_3) \cdot x_4$
- $x_1 \cdot (x_2 \cdot (x_3 \cdot x_4))$

see Figure 1.4. Often, the circuits we consider will be restricted in the ways they are allowed to compute monomials. The reason for this is a decomposition lemma of the polynomials according to monomials computation, see Lemma 1.8; although quite trivial, this decomposition is at the core of many others, more complex ones, that will arise in this thesis.

To make this precise we need the notion of a "parse tree" of a circuit, which has been considered in many previous works [18, 2, 36].

Parse trees

Fix a homogeneous non-commutative circuit C. A *parse formula* of C is a non-commutative *formula* C' obtained from C as follows:

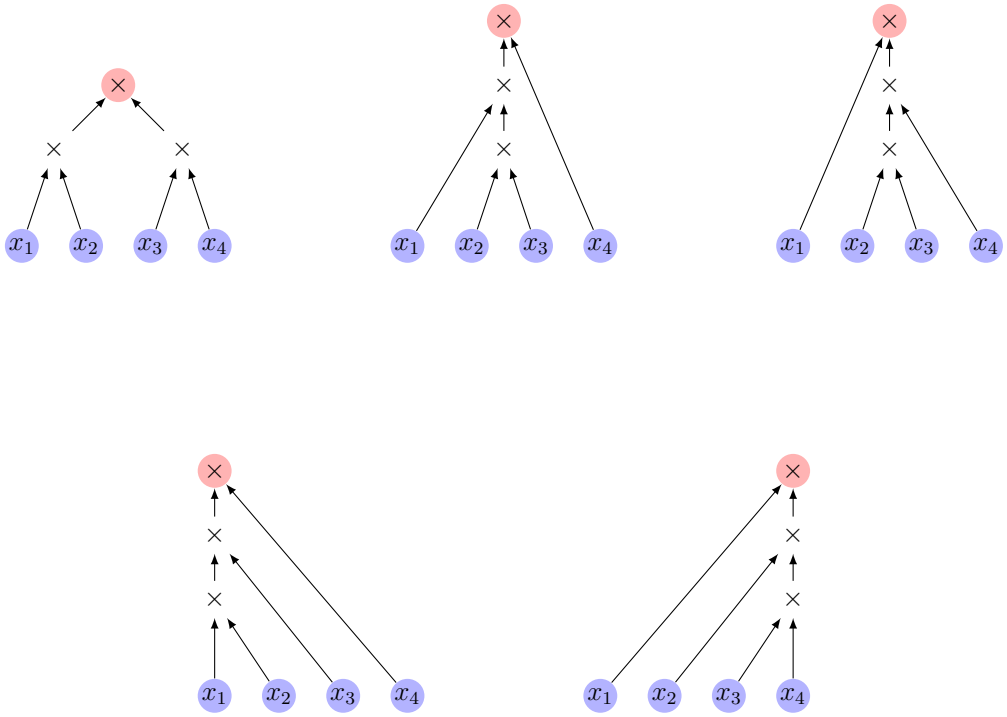


Figure 1.4: Five essential ways to compute $x_1x_2x_3x_4$ by non-commutative arithmetic circuit with only $\times$ gates of fan-in 2.

- Corresponding to the output $+$ gate of C, we add an output $+$ gate to C',
- For every $+$ gate Φ' added to C' corresponding to a $+$ gate Φ in C, we choose exactly one child Ψ of Φ in C and add a copy Ψ' to C' as a child of Φ'. The constant along the wire from Ψ' to Φ' remains the same as in C.
- For every $\times$ gate Φ' added to C' corresponding to a $\times$ gate Φ in C and *every wire* from a child Ψ to Φ in C, we add a copy of Ψ' to C' and make it a child of Φ. The order of the various gates Ψ' added to C' is the same as the order of the corresponding wires in C.

Any such parse formula C' computes a *monomial* (with a suitable coefficient) that is denoted by $val(C')$. As the following lemma shows—whose a proof can be foundd in [35] for example—the polynomial computed by C is the sum of all monomials computed by parse formulas C' of C.

Lemma 1.8: Monomials decomposition

Let $f \in \mathbb{F}\langle X \rangle$ be a polynomial computed by a non-commutative arithmetic

circuit C. Then

$$f(X) = \sum_{C'} val(C'),$$

where C' runs over all parse formulas of the circuit C.

A parse tree of C is a rooted, ordered tree obtained by taking a parse formula C' of C, "short circuiting" the $+$ gates (i.e., we remove the $+$ gates and connect the gates that were connected to it directly), and deleting all labels of the gates and the edges of the tree. See Figure 1.5 for an example. Note that in a homogeneous circuit C, each such tree has exactly d leaves, where d is the degree of the polynomial computed by C. We say that the tree T is the *shape* of the parse formula C'.

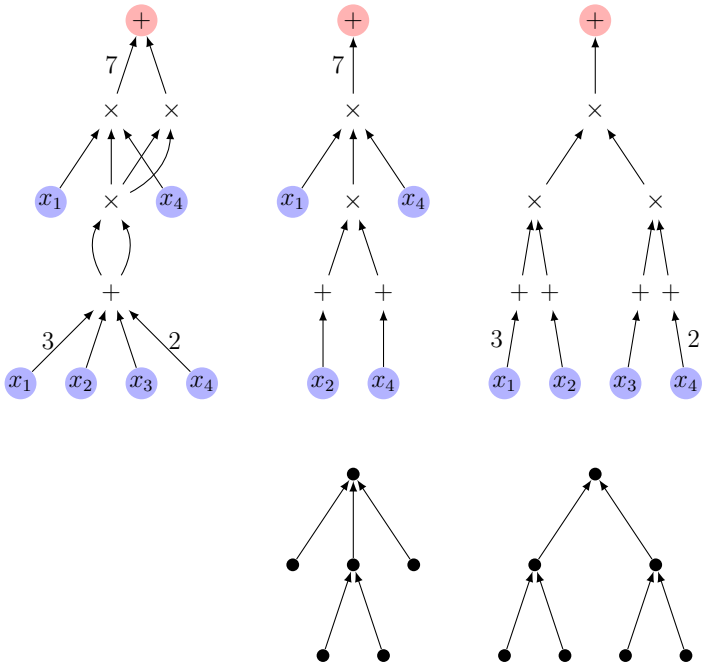


Figure 1.5: First row: from left to right, a non-commutative arithmetic circuit and two parse formulas in the circuit. Second row: the corresponding parse trees. To simplify the picture, the constant 1 has not been depicted along some edges. Also we have not introduced $+$ gates between the two layers of $\times$ gates; the reader should assume that the edges between the two layers carry $+$ gates of fan-in 1.

The process that converts the parse formula C' into T associates to each internal node of T a multiplication and an addition gate of C' and to each leaf of T an input and an addition gate of C'. See Figure 1.6.

Let T be a parse tree of a homogeneous circuit C with d leaves. Recall that a pre-order traversal of a tree visits first the root, and then recursively traverses

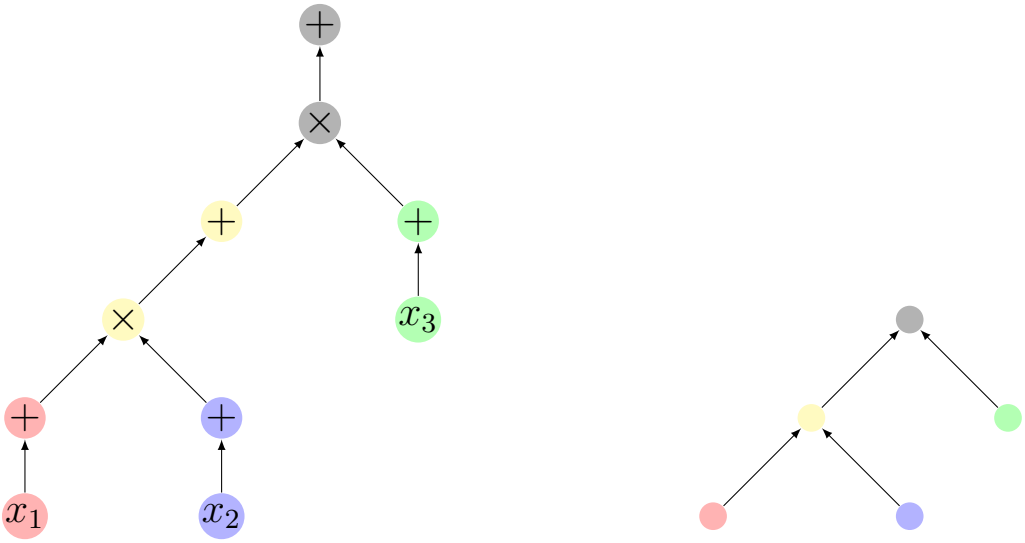


Figure 1.6: Association between gates in a parse formula and nodes in the shape. Gates and nodes of same color are associated. Left: a parse formula. Right: the corresponding shape

each child. Given a node $v \in V(T)$, we define

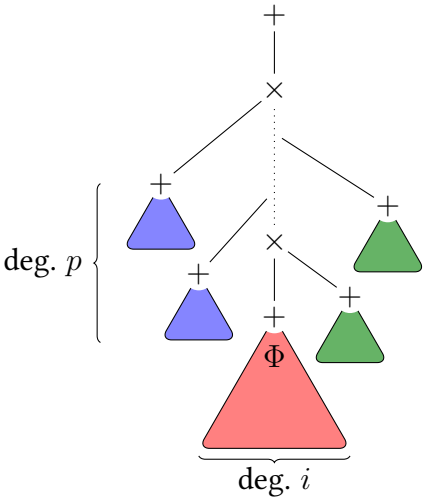
$$\deg(v) := \text{the number of leaves in the subtree rooted at } v$$

and

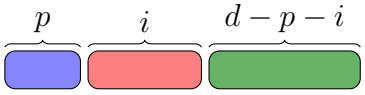
$$\mathrm{pos}(v) := (1 + \text{the number of leaves preceding } v \text{ in a pre-order traversal of } T)$$

The type of v is defined to be $\mathrm{type}(v) := (\deg(v), \mathrm{pos}(v))$. (The reason for this definition is that in any parse formula C' of shape T, the monomial computed by the addition gate, multiplication gate or input gate corresponding to v in C' computes a monomial of degree $\deg(v)$ which sits at position $\mathrm{pos}(v)$ w.r.t. the monomial computed by C'. See Figure 1.7). We also use $\mathcal{I}(T)$ to denote the set of internal nodes of T and $\mathcal{L}(T)$ to denote the set of leaves of T.

The set of parse trees that can be obtained from parse formulas of C is denoted $\mathcal{T}(C)$. We say that a homogeneous non-commutative arithmetic circuit is a *Unique Parse Tree circuit* (or *UPT circuit*) if $|\mathcal{T}(C)| = 1$ (this is equivalent to the definition of *unambiguous circuits* we have introduced in [25]). More generally if $|\mathcal{T}(C)| \leq k$, we say that C is a *k-PT* circuit. Finally, if $\mathcal{T}(C) \subseteq \mathcal{T}$ for some family $\mathcal{T}$ of trees, we say that C is *$\mathcal{T}$-PT*. Similarly, we also define UPT formulas, k-PT formulas and $\mathcal{T}$-PT formulas. If C is a UPT circuit with $\mathcal{T}(C) = \{T\}$, we say that T is the *shape* of the circuit C.



(a) A parse formula and a gate Φ which is associated to a gate $v \in T$ of type (i, p).



(b) Repartition of the variables in the monomial computed by the parse formula.

Figure 1.7: Type of a gate in a parse formula.

Remark 1.9

We can interpret some classes using this new framework. For example:

- Algebraic Branching Programs can be proved to be equivalent to UPT circuits for which the shape is a comb[a].
- Skew circuits are equivalent to circuits for which the shapes can be all possible rotations of a comb.
- General circuits are circuits for which the shapes are unrestricted.

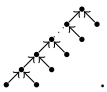
[a] A comb is a tree of this form  .

Figure 1.8 gives a parse trees' point of view of some classes that we will define later.

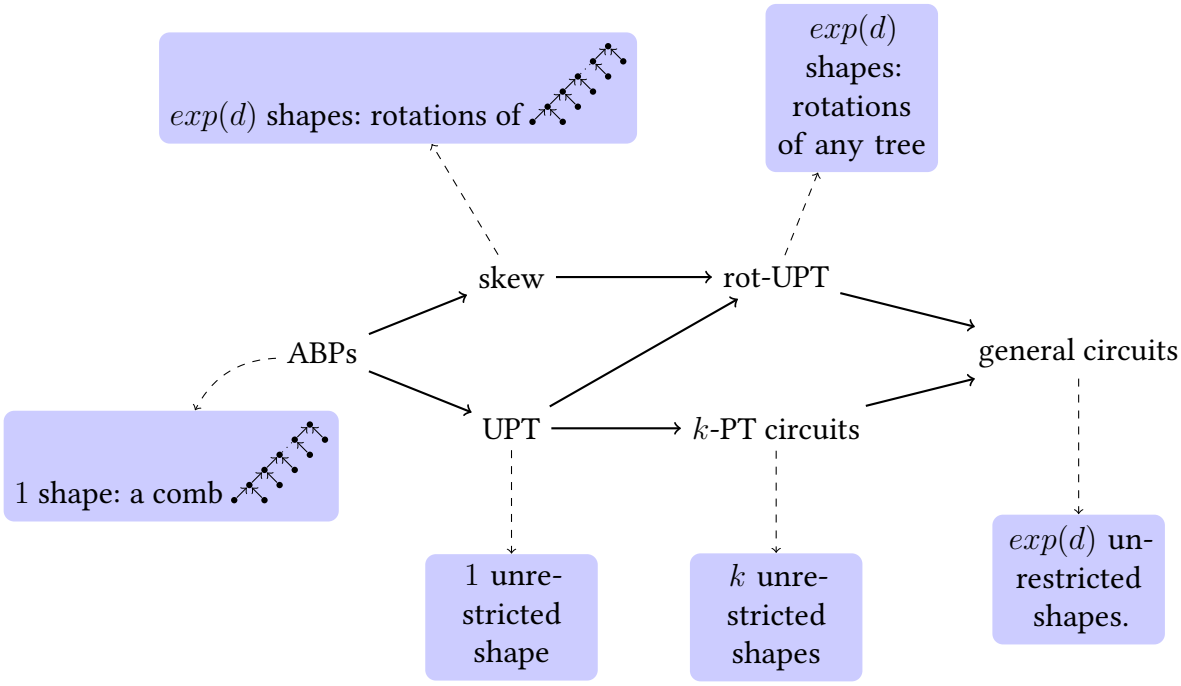


Figure 1.8: Parse trees' point of view.

1.4 Lower bound techniques

1.4.1 High level idea

The main known lower bounds in arithmetic complexity are quite often a consequence of the following (high level) steps:

1. Define a measure on polynomials $\mu : \mathbb{F}\langle X\rangle \to \mathbb{R}$. This measure is extended to circuits: $\mu(C) := \mu(P_C)$.

2. Prove that the polynomials computed by a class of circuits $\mathcal{C}$ are of low measure (typically, polynomial in the size of the circuits). In symbol:

$$\forall C \in \mathcal{C}, \mu(C) = |C|^{O(1)}.$$

3. Give a polynomial p of high measure.

Then, use the observation that a circuit $C \in \mathcal{C}$ computing p satisfies $\mu(C) = \mu(p)$, therefore $|C|^{O(1)} \geq \mu(p)$. If the measure of p is large enough, this process provides an interesting lower bound on the size of any circuit of the class $\mathcal{C}$ computing p.

Choice of the measure

Ideally, μ should be a good approximation of the "true" complexity of our polynomials, against general circuits. Such requirements seem out of reach so far. Instead, the measures used in practice are somehow weaker but more manipulable. Most often, given a class of circuits $\mathcal{C}$, a decomposition lemma for polynomials computed by circuits in the class is given: $\forall C \in \mathcal{C}, P_C = \sum_{i=1}^{k} f_i$, where f_i are simpler polynomials and k a parameter related to the size of C. This decomposition paves the way to defining a well fitted measure. Indeed, a natural strategy is to define μ such that:

- $\mu(f)$ is small for all simple polynomials f coming from the decomposition.
- μ is subadditive. That is, $\forall f, g \in \mathbb{F}\langle X \rangle, \mu(f+g) \leq \mu(f) + \mu(g)$.
- The polynomials for which we want lower bounds are of μ-measure reasonnably high.

Subadditivity gives the point 2. that was stated in the high level strategy. Indeed, for a polynomial p, we then have:

$$\mu(p) = \mu(C) \leq \sum_{i=1}^{k} \mu(f_i)$$

From this, one can usualy deduce a lower bound on k; but k being related to the size of C, we get lower bound on the size of any $C \in \mathcal{C}$ computing p.

1.4.2 Partial derivative matrix

Go back to Nisan

In an influential result, Nisan [38] made progress on the problem of proving lower bounds for non-commutative circuits by proving exponential lower bounds on the size of non-commutative *formulas*, and more generally non-commutative *algebraic branching brograms* (ABPs), computing the determinant and permanent (and also other explicit polynomials).[4] The method used by Nisan to prove this lower bound can also be seen as a precursor to the method of *partial derivatives* in arithmetic circuit complexity (introduced by Nisan and Wigderson [39]), variants of which have been used to prove a large body of lower bound results in the area [39, 44, 14, 21, 22].

[4] In contrast, we do not yet have superpolynomial lower bounds for depth-3 formulas in the commutative setting for computing any explicit polynomial.

Given a homogeneous polynomial f of degree d over a set X of n variables, Nisan considers the matrix $M[f]$ of size $n^{d/2} \times n^{d/2}$ whose rows and columns are indexed by monomials of degree $d/2$ each, where the (m_1, m_2)th entry of $M[f]$ is the coefficient of the monomial $m_1 m_2$ in f. The measure of a polynomial is then the rank of this matrix.

Nisan then proved that any polynomial computed by a formula (or an ABP) must be of relatively small rank, that is polynomial in the size of the formula (or the ABP). But it is pretty easy to see that interesting polynomials such as the determinant or the permanent are of high rank.

Here, we follow the extension of Nisan's measure that was introduced in [30], where the more general family of matrices $M_Y[f]$ where $Y \subset [d]$ is of size $d/2$ and (m_1, m_2)th entry of $M_Y[f]$ is the coefficient of the unique monomial m such that the projection of m to the locations in Y gives m_1, and the locations outside gives m_2. The formal and precise definition is stated below. This time, the hard part will be to find good subsets Y which make the rank of the polynomials computed by an arithmetic class of circuits small. Such subsets Y can therefore be seen as weaknesses for the considered class of circuits.

Generalisation

Here we recall some definitions from [38] and [30]. Let Π denote a partition of $[d]$ given by an ordered pair (Y, Z), where $Y \subseteq [d]$ and $Z = [d] \setminus Y$. In what follows we only use ordered partitions of sets into two parts. We say that such a Π is *balanced* if $|Y| = |Z| = d/2$.

Given a monomial m of degree d and a set $W \subseteq [d]$, we use m_W to denote the monomial of degree $|W|$ obtained by keeping only the variables in the locations indexed by W and dropping the others. For example, if $W = \{1, 3\} \subseteq [4]$ and $m = xyzt$, then $m_W = xz$.

Definition 1.10: Partial Derivative matrix

Let $f \in \mathbb{F}\langle X \rangle$ be a homogeneous polynomial of degree d over $n = |X|$ variables. Given a partition $\Pi = (Y, Z)$ of $[d]$, we define an $n^{|Y|} \times n^{|Z|}$ matrix $M[f, \Pi]$ with entries from $\mathbb{F}$ as follows: the rows of $M[f, \Pi]$ are labelled by monomials from $\mathcal{M}_{|Y|}(X)$ and the columns by elements of $\mathcal{M}_{|Z|}(X)$. Let $m' \in \mathcal{M}_{|Y|}(X)$ and $m'' \in \mathcal{M}_{|Z|}(X)$; the (m', m'')th entry of $M[f, \Pi]$ is the coefficient in the polynomial f of the unique monomial m such that $m_Y = m'$ and $m_Z = m''$.

Example 1.11

Consider $X = \{x_1, x_2\}$ and $f = 3x_1x_2^3 + x_1x_2x_1^2 + 7x_2x_1^2x_2$ a homogeneous polynomial of degree 4. Let us consider also the two partitions $\Pi_1 = (\{2,4\},\{1,3\})$ and $\Pi_2 = (\{2,3,4\},\{1\})$. Then we have:

$$M[f,\Pi_1] = \begin{array}{c} \begin{matrix} x_1^2 & x_1x_2 & x_2x_1 & x_2^2 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 7 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} x_1^2 \\ x_1x_2 \\ x_2x_1 \\ x_2^2 \end{matrix}$$

and

$$M[f,\Pi_2] = \begin{array}{c} \begin{matrix} x_1 & x_2 \end{matrix} \\ \begin{bmatrix} 0 & 0 \\ 0 & 7 \\ 0 & 0 \\ 3 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \end{array} \begin{matrix} x_1^3 \\ x_1^2x_2 \\ x_1x_2^2 \\ x_2^3 \\ x_2^2x_1 \\ x_2x_1^2 \\ x_1x_2x_1 \\ x_2x_1x_2 \end{matrix}$$

We will use the rank of the matrix $M[f,\Pi]$—denoted $\mathrm{rank}(f,\Pi)$—as a measure of the complexity of f. Note that since the rank of the matrix is at most the number of rows, we have for any $f \in \mathbb{F}\langle X\rangle$ $\mathrm{rank}(f,\Pi) \leq n^{|Y|}$.

Definition 1.12: Relative Rank

Let $f \in \mathbb{F}\langle X\rangle$ be a homogeneous polynomial of degree d over $n = |X|$ variables. For any $Y \subseteq [d]$, we define the *relative rank of f w.r.t.* $\Pi = (Y,Z)$—denoted $\text{rel-rank}(f,\Pi)$— to be

$$\text{rel-rank}(f,\Pi) := \frac{\mathrm{rank}(M[f,\Pi])}{n^{|Y|}}.$$

Fix a partition $\Pi = (Y,Z)$ of $[d]$ and two homogeneous polynomials g,h of degrees d_g and d_h respectively. Let $f = g \times_j h$ for some $j \in [0,d_h]$. This

induces naturally defined partitions Π_g of $[d_g]$ and Π_h of $[d_h]$ respectively in the following way. Let $I_g = [j+1, j+d_g]$ and $I_h = [d] \setminus I_g$. We define $\Pi_g = (Y_g, Z_g)$ such that Y_g corresponds to the indices of the elements of Y in I_g, that is, $Y_g = \{k \in [d_g] \mid Y \text{ contains the } k\text{th smallest element of } I_g\}$; $\Pi_h = (Y_h, Z_h)$ is defined similarly with respect to I_h. Denote $|Y_g|, |Z_g|, |Y_h|, |Z_h|$ by d'_g, d''_g, d'_h, d''_h respectively.

In the above setting, we have a simple description of the matrix $M[f, \Pi]$ in terms of $M[g, \Pi_g]$ and $M[h, \Pi_h]$. We use the observation that monomials of degree $|Y| = d'_g + d'_h$ are in one-to-one correspondence with pairs (m'_g, m'_h) of monomials of degree d'_g and d'_h respectively (and similarly for monomials of degree $|Z|$). The following appears in [30].

Lemma 1.13: Tensor Lemma

Let $f = g \times_j h$ be as above. Then, $M[f, \Pi] = M[g, \Pi_g] \otimes M[h, \Pi_h]$, where $\otimes$ stands for the tensor product.

Corollary 1.14

Let $f = g \times_j h$ be as above. We have $\operatorname{rank}(f, \Pi) = \operatorname{rank}(g, \Pi_g) \cdot \operatorname{rank}(h, \Pi_h)$. In the special case where one of Y_g or Z_g is empty and one of Y_h or Z_h is empty, the tensor product is an outer product of two vectors and hence $\operatorname{rank}(f, \Pi) \leq 1$.

Sometimes, we will associate to any partition $\Pi = (Y, Z)$ the string in $\{-1, 1\}^d$ that contains -1 in exactly the locations indexed by Y. Given partitions $\Pi_1, \Pi_2 \in \{-1, 1\}^d$, we now define $\Delta(\Pi_1, \Pi_2)$ to be the Hamming distance between the two strings, or equivalently as $|Y_1 \Delta Y_2|$ where $\Pi_1 = (Y_1, Z_1)$ and $\Pi_2 = (Y_2, Z_2)$.

Proposition 1.15

Let $f \in \mathbb{F}\langle X \rangle$ be homogeneous of degree d and $\Pi \in \{-1, 1\}^d$. Then, $\operatorname{rank}(f, \Pi) = \operatorname{rank}(f, -\Pi)$, where $-\Pi(i) = -1 \times \Pi(i)$ for all $i \in [d]$.

Proof. It follows from the fact that $M[f, -\Pi]$ is the transpose of $M[f, \Pi]$. □

Lemma 1.16: Distance lemma

Let $f \in \mathbb{F}\langle X \rangle$ be homogeneous of degree d and $\Pi_1, \Pi_2 \in \{-1,1\}^d$. Then, $\mathrm{rank}(f, \Pi_2) \leq \mathrm{rank}(f, \Pi_1) \cdot n^{\Delta(\Pi_1, \Pi_2)}$.

Proof. We prove this by induction on $\Delta(\Pi_1, \Pi_2)$. The base case of the induction is the case that $\Delta(\Pi_1, \Pi_2) = 0$, i.e., $\Pi_1 = \Pi_2$. In this case, the statement is trivial.

Now consider when $\Delta(\Pi_1, \Pi_2) = \Delta \geq 1$. Take any partition Π such that $\Delta(\Pi_1, \Pi) = \Delta - 1$ and $\Delta(\Pi, \Pi_2) = 1$. By the induction hypothesis, we know that $\mathrm{rank}(f, \Pi) \leq \mathrm{rank}(f, \Pi_1) \cdot n^{\Delta - 1}$ and so it suffices to show that $\mathrm{rank}(f, \Pi_2) \leq \mathrm{rank}(f, \Pi) \cdot n$.

Assume that $\Pi = (Y, Z)$ and $\Pi_2 = (Y_2, Z_2)$. We know that $\Delta(\Pi, \Pi_2) = |Y \Delta Y_2| = 1$. W.l.o.g. assume that $Y = Y_2 \setminus \{i\}$ for some $i \in [d]$ (the other case, when $Y = Y_2 \cup \{i\}$ is similar). Note that $Z = Z_2 \cup \{i\}$.

Consider the matrix $M_2 := M[f, \Pi_2]$. We divide M_2 into n blocks as follows. For each $x \in X$, let M_2^x be the submatrix where we only keep the rows corresponding to monomials of degree $|Y_2|$ that contain the variable x in the location "corresponding" to $i \in [d]$ (i.e., in the jth position where j is the index of i in Y_2). Clearly, we have $\mathrm{rank}(M_2) \leq \sum_{x \in X} \mathrm{rank}(M_2^x)$.

On the other hand, we also see that each M_2^x is a submatrix of $M := M[f, \Pi]$: namely, the submatrix obtained by only keeping the columns corresponding to those monomials that contain the variable x in the location corresponding to i (as above but w.r.t. Z). Hence, $\mathrm{rank}(M_2^x) \leq \mathrm{rank}(M)$ for each x.

Hence, we see that $\mathrm{rank}(M_2) \leq \sum_{x \in X} \mathrm{rank}(M_2^x) \leq n \cdot \mathrm{rank}(M)$ and this completes the induction. □

A polynomial that is full rank w.r.t. all partitions

Remember the high level steps given by Remark 1.4.1 in order to provide lower bounds. For us, we will have:

1. This is what we did in the previous subsection. Indeed, our measure μ will be the rank of the matrices $M[\star, \Pi]$ for well chosen partition Π depending on the class of circuits that is considered.

2. This point will be the core of all the next chapters. Most often, a chapter will consist of finding a good decomposition of the polynomials computed by a class of circuits. This decomposition will lead us to find partitions Π that make the rank of the computed polynomials small.

3. This point is already done by the following theorem, which was shown in [30].

Theorem 1.17

For any even d and any positive $N \in \mathbb{N}$, there is an integer $q_0(N,d)$ such that the following holds over any field $\mathbb{F}$ of size at least $q_0(N,d)$. There is an explicit homogeneous polynomial $F_{N,d} \in \mathbb{F}\langle X\rangle$ of degree d over $N = |X|$ variables such that for any balanced partition $\Pi = (Y,Z)$ of $[d]$, $\mathrm{rank}(F_{N,d},\Pi) = N^{d/2}$ (equivalently, $\text{rel-rank}(F_{N,d},\Pi) = 1$). Further, $F_{N,d}$ can be computed by an explicit homogeneous non-commutative arithmetic circuit of size $\mathrm{poly}(N,d)$.

Sketch of the proof. Fix d an even integer and let $N = |X|$ be the number of variables. Consider the complete graph where vertices are the elements of $[d]$; i.e., consider the graph $G = ([d], \binom{[d]}{2})$. Create also a new variable λ_e for every edge e in G. Then, consider the following polynomial:

$$g(X,\lambda) = \sum_{M \text{perfect matching of } G} (\prod_{e\in M} \lambda_e).g_M(X),$$

where

$$g_M(X) = \sum_{w\in[n]^d : w_i = w_j \forall \{i,j\}\in M} x_{w_1}x_{w_2}\dots x_{w_d}.$$

Then one can prove that:

- This polynomial is computable by a small arithmetic circuit (recursive construction).
- For any balanced partition Π, there is (at least) one perfect matching M_Π for which $M[g_{M_\Pi},\Pi]$ is a permutation matrix, and hence is full-rank for this partition.
- As long as the underlying field $\mathbb{F}$ is of size large enough, one can instantiate the variables λ to $\alpha \in \mathbb{F}^{d(d-1)/2}$ such that for any balanced partition Π, $\mathrm{rank}(g(X,\alpha),\Pi) = \mathrm{rank}(g_{M_\Pi}(X),\Pi) = N^{d/2}$.
- A little argument then shows that if a polynomial is full-rank with respect to any balanced partition, then this polynomial is also full-rank with respect to any partition.

□

Therefore, to get lower bounds, we just need to take care of 2., as we can always take the polynomial given by Theorem 1.17 for a polynomial of high

measure. However, sometimes, we will want a lower bound for a particular polynomial (such as $\mathrm{IMM}_{n,d}$ in Chapter 4), in this case we will need to do a proof of 3. again.

Remark 1.18

Since Theorem 1.17 provides a polynomial which is full rank with respect to any partition, it means that the rank method alone is not enough to prove superpolynomial lower bounds against general circuits.

1.5 Table of separations

The following table presents the separations between classes of non-commutative circuits that are proved in this manuscript. Each row corresponds to a separation together with the polynomial that is used to show it.

Separation	Polynomial
ABP $\subsetneq$ Skew	palindrome (already known)
ABP $\subsetneq$ UPT	palindrome (simple observation)
UPT $\not\subset$ Skew	square of the palindrome 2.4.1
Skew $\subsetneq$ rot-UPT	square of the palindrome 2.4.1
Skew $\not\subset$ UPT	moving palindrome 2.4.1
UPT $\subsetneq$ rot-UPT	moving palindrome 2.4.1
UPT $\subsetneq$ k-UPT	moving palindrome 2.4.1
k-PT $\subsetneq$ general circuits	full rank polynomial w.r.t all partitions 3.1
rot-PT $\subsetneq$ general circuits	full rank polynomial w.r.t all partitions 3.2
k-PT $\not\subset$ rot-PT	sum of palindrome and square of the palindrome 3.10
UPT formula $\subsetneq$ ABP	iterated matrix multiplication 4.2
k-PT formula $\subsetneq$ ABP	iterated matrix multiplication 4.3

Chapter 2

UPT Circuits

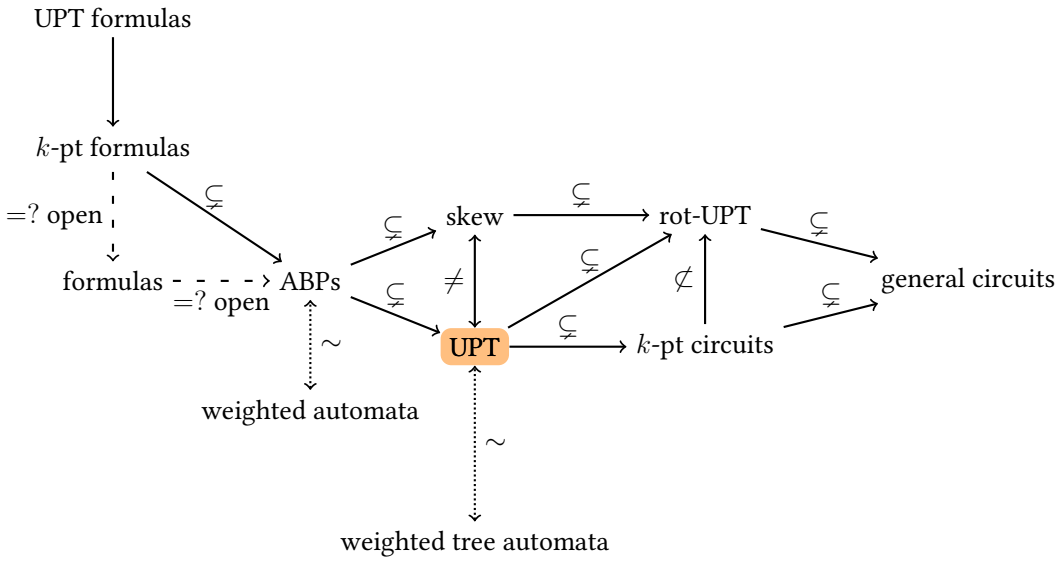


Figure 2.1: You are here.

Overview

This chapter focuses on UPT circuits, which are circuits with only one kind of parse tree, called the shape of the circuit. These circuits will be the core bricks of almost all decompositions in the next chapters. One reason for this is that we understand them almost completely, as this chapter shows.

A parse formula corresponds to a monomial computation, therefore the class of UPT circuits can be seen as circuits for which each monomial is computed in the same way, given by the underlying shape. The results contained in this

chapter can be seen as direct extensions of Nisan's work for ABPs [38]—in which he provides an exact characterisation of the complexity of ABPs, and uses it to derive lower bounds for them. Indeed, one can easily observe that the class of polynomials computed by ABPs are exactly the ones computed by UPT circuits for which the underlying shape is a comb (recall that the definition of a comb is given in Remark 1.9). Following Nisan, we show an exact characterisation of the complexity of UPT circuits and use it to give lower bounds for interesting polynomials such as the determinant or the permanent.
This chapter is based on the following publication:

- Guillaume Lagarde, Guillaume Malod, and Sylvain Perifel. Non-commutative computations: lower bounds and polynomial identity testing. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:94, 2016

However, some notation and proofs have been modified with the intention of unifying and simplifying parts of the initial paper.

Multiplication fan-in restriction

First, we observe that any UPT formula or circuit can be converted to another (of possibly different shape) where each multiplication gate has fan-in at most 2.

Lemma 2.1

Let C be a UPT circuit (resp. formula) of size s (recall that the size is the number of edges) and shape T. Then there is a tree T' and a UPT circuit (resp. formula) C' of size $\leq 3s$ and shape T' such that C' computes the same polynomial as C and every multiplication gate in C' has fan-in at most 2. (This implies that every internal node of T' also has fan-in at most 2.) Further, there is a deterministic polynomial-time algorithm which, when given C, computes C' as above.

Proof. We give the proof only for UPT circuits, since the transformation is the same in both cases. Let C a UPT circuit as in the statement. For any $\times$-gate Φ with $k > 2$ children $\Psi_0, \ldots, \Psi_{k-1}$, we replace Φ by the following gadget of $2(k-1)-1$ gates $\Phi_0, \ldots, \Phi_{2(k-2)}$. For any $i \in [0, k-3]$, Φ_{2i} is a multiplication gate with inputs Ψ_i and Φ_{2i+1}, and Φ_{2i+1} is an addition gate with input $\Phi_{2(i+1)}$. Finally, $\Phi_{2(k-2)}$ is a multiplication gate with inputs Ψ_{k-2} and Ψ_{k-1}. The new circuit is still in alternating layer form, and is clearly UPT because we apply the same process to any multiplication gate of fan-in strictly greater than 2. For any such gate, the number of edges in the corresponding gadget is $3k-4$. Therefore, the number of

edges in the final circuit increases by at most three times the number of edges in the original circuit, so that the size of the circuit obtained by this process is $\leq 3s$.

The shape T' of the new formula is simply the modified version of the shape T obtained by replacing the internal nodes of fan-in $k > 2$ by right combs with k leaves.

This completes the construction of C' from C. The construction is easily seen to be implementable by a deterministic polynomial-time algorithm. □

Remark 2.2

Without loss of generality, for the rest of this chapter and in order to simplify the proofs, we consider only UPT circuits with multiplication fan-in at most 2. Lemma 2.1 tells us this is not a strong assumption since we can efficiently transform our circuits in order to get this additional property, with a small blow-up in size that does not matter.

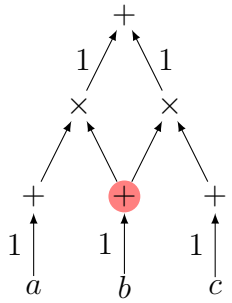
2.1 Normal form

Let C be a UPT circuit of shape T. We say that C is in *normal form* if there is a function $v : C \to T$ that associates to each gate Φ of the circuit a node $v(\Phi) \in T$ such that the following holds: if Φ is an input gate, then $v(\Phi)$ is a leaf; if Φ is a $\times$ gate with children $\Psi_1, \ldots, \Psi_t$ (in that order), then the nodes $v(\Psi_1), \ldots, v(\Psi_t)$ are the children of $v(\Phi)$ (in that order); and finally, if Φ is a $+$ gate with children $\Psi_1, \ldots, \Psi_t$ (which are all $\times$ or input gates since we assume that $+$ and $\times$ gates are alternating along each input to output path), then $v(\Phi) = v(\Psi_1) = \cdots = v(\Psi_t)$. Intuitively, what this means is that in any unravelling of a parse formula containing a (multiplication or input) gate Φ to get the parse tree T, the gate Φ always takes the position of node $v(\Phi)$. See Figure 2.2 for an example.

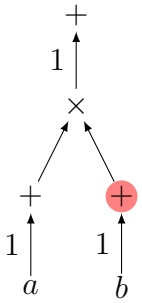
Let C be either a UPT formula or a UPT circuit of shape T in normal form. We say that a $+$ gate Φ in C is a $(u, +)$ gate if $v(\Phi) = u \in T$. Similarly, we refer to a $\times$ gate Φ in C as a $(u, \times)$ gate if $v(\Phi) = u$. For simplicity of notation, we also refer to an *input* gate Φ as a $(u, \times)$ gate if $v(\Phi) = u$. Note that the output gate is a $(u_0, +)$ gate where u_0 is the root of T.

We state and prove below some simple structural facts about UPT circuits.

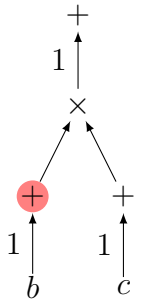
The following proposition shows that it is always possible to transform a UPT circuit into a UPT circuit in normal form. It is for these circuits that we will be able to give an exact characterisation of the complexity.



(a) A UPT circuit C not in canonical form.



(b) A parse formula of C with the red gate on the right.



(c) Another parse formula of C with the red gate on the left.

Figure 2.2: A UPT circuit that is not in normal form.

Proposition 2.3

1. Let C be a UPT *formula*. Then C is in normal form.
2. For any UPT circuit C of size s and shape T, there is another UPT circuit C' of size $O(s^2)$ and shape T in normal form computing the same polynomial as C. Further, given C and T, such a C' can be constructed in polynomial time.

Proof. • **Proof of 1.**

Let C be a UPT formula with shape T. We want to prove that C is in normal form; this is equivalent to proving that for any multiplication gate $\Phi \in C$ and for any parse formula containing the gate Φ, the gate always takes the same position in T. Let D, D' be any two parse formulas containing Φ. D (resp. D') is a formula, therefore there is a unique path p (resp. p') from the root to Φ in D (resp. D'). The crucial point is the following: as C is also a formula with D and D' as subformulas, these two paths must be equal. By definition, the position of Φ in T with respect to D is characterized by $(\deg(\Phi), \mathrm{pos}(\Phi))$; we recall that $\deg(\Phi)$ is the degree of the monomial computed at the gate Φ in D and $\mathrm{pos}(\Phi)$ equals $1 +$ the sum of the degrees of the monomials computed at the children of the multiplication gates along the path p which are on the left side of the path. As the formula is UPT, the monomials computed at a gate are all of same degree for any parse formulas containing the gate; moreover $p = p'$ so in both cases we consider the same gates in the definition of $(\deg(\Phi), \mathrm{pos}(\Phi))$ in D or D' so that the positions of Φ in T according to D or D' are equal.

- **Proof of 2.**

  The proof of this point relies on a careful inspection of the proof of [36, Lemma 2]. A circuit is called *multiplicatively disjoint* if each $\times$-gate has disjoint subcircuits as inputs. The result [36, Lemma 2] states that every circuit C of degree d can be turned efficiently into an equivalent multiplicatively disjoint circuit of size $(|C| + d)^{O(1)}$.

  The normal form is obtained by applying the algorithm to transform a general circuit into a multiplicatively disjoint circuit from [36, Lemma 2]. The resulting circuit has size $\leq 2d|C|$.

  Observe that it is not only the fact that the circuit is multiplicatively disjoint that makes the circuit in normal form, since Figure 2.2 shows a multiplicatively disjoint circuit not in normal form; the transformation itself is crucial. For the sake of completeness, we recall the construction here (modified a little bit for the needs of non-commutativity).

  For each gate $\Phi \in C$ of formal degree e, the new circuit D contains distinct gates $\Phi_1, \Phi_2, \ldots, \Phi_{d+1-e}$. Φ_k is called a clone of index k of Φ. In C, if Φ is a $\times$-gate of formal degree e with left input Ψ of formal degree e_1 and right input Γ of formal degree e_2, then in D, Φ_k has left input Ψ_k and right input Γ_{k+e_1}. In C, if Φ is a $+$-gate of formal degree e with inputs $\Psi^1, \Psi^2, \ldots, \Psi^j$ with coefficients $c_1, c_2, \ldots, c_j$, then, in D, Φ_k has inputs $\Psi^1_k, \Psi^2_k, \ldots, \Psi^j_k$ with coefficients $c_1, \ldots, c_j$.

  The proof that D is multiplicatively disjoint and computes the same polynomial as C is given in [36, Lemma 2]. There it is also proved that, in D, all gates in the subcircuit defined by a gate Φ_k of formal degree e are clones whose index lie between k and $k + e - 1$: we will call that the index property.

  We prove by contradiction that D respects the fact that in any unravelling of a parse formula containing a gate to get the parse tree, the gate always takes the same position in the (unique) parse tree. Let Φ_j be an addition gate in D and D_1 and D_2 two parse formulas which contain Φ_j but at two different positions in the shape. Let $l_1, l_2, \ldots, l_a$ (resp. $g_1, g_2, \ldots, g_b$) be the unique path in D_1 (resp. D_2) from the output gate to Φ_k (thus $l_a = g_b = \Phi_k$). Because Φ_k does not share the same position in the shape, it means that there is a minimal c such that l_c and g_c are $+$-gates with different positions. It means that l_{c-1} and g_{c-1} are two $\times$-gates (because the circuit is constituted of alternating layers) and that l_c and g_c are inputs of l_{c-1} and g_{c-1}, one as left input, one as right input (let us say in that order). As the circuit is UPT, l_c and g_c must be of same degree e. g_{c-1} and l_{c-1} are clones of same index because the path from the output gate to these gates are identical. Let us say

they are of index k. Thus l_c is a clone of index k and g_c is a clone of index $k+e$ (because of the construction and the fact that one is a left input, the other a right input of the multiplication gate). Thanks to the index property, this means that the subcircuits defined by l_c and g_c are clones whose index lies between k and $k+e-1$ for l_c and between $k+e$ and $k+2e-1$ for g_c. These two sub-circuits are thus disjoints, but this in contradiction with the fact that Φ_j belongs to both of them.

□

2.2 Decomposition lemma

In this section, we show that polynomials computed by UPT circuits in normal form can be decomposed in a way that will prove useful later to get a characterisation of the complexity. Recall that the definition of the *j-product* ($\times_j$) is given in Definition 1.5 and that of *type* in Section 1.3.

Proposition 2.4: Decomposition for UPT circuits in normal form

If a polynomial $f \in \mathbb{F}\langle X \rangle$ of degree d is computed by a UPT circuit in normal form with shape T, then for any $v \in T$ of type (i,p), f can be written as $f = \sum_{j=1}^{k_v} g_j \times_p h_j$, where:

1. k_v is the number of $(v,+)$ gates and $g_1, \ldots, g_v$ are the polynomials computed by these gates;
2. $\forall j, \deg(g_j) = i$ and $\deg(h_j) = d - i$.

Proof. Let C be a UPT circuit in normal form computing the polynomial f. By Lemma 1.8, we have: $f = \sum_{C' \in S} val(C')$, where S is the set of all parse formulas of C. Let $\Phi_1, \ldots, \Phi_{k_v}$ be the $(v,+)$ gates, computing respectively the polynomials $g_1, g_2, \ldots, g_{k_v}$. By definition of the type of the node v, the g_j are of degree i. For $1 \leq j \leq k_v$, let S_j be the set of parse formulas containing the gate Φ_j. Because a parse formula contains at most one $(v,+)$ gate, we have $S_j \cap S_k = \emptyset$ for $j \neq k$. Moreover, every parse formula must contain at least one $(v,+)$ gate. Thus the S_j are a partition of S: $S = S_1 \sqcup S_2 \sqcup \cdots \sqcup S_{k_v}$, where $\sqcup$ denotes the disjoint union. We can then rewrite the previous equality as $f = \sum_{C' \in S} val(C') = \sum_{j=1}^{k_v} \sum_{C' \in S_j} val(C')$.

Fix $j \in [1, k_{i,p}]$. Consider the circuit $C_j(y)$ obtained by changing Φ_j into an input gate labeled with a new variable y and deleting unused gates. Note that $C_j(g_j) = C$ (abusing notation and using the name of the circuit for the

computed polynomial). Let F_j be the set of parse formulas of C_j containing the input gate Φ_j. The value of any parse formula $C' \in F_j$ is of the form $y \times_p h_{C'}$ where $h_{C'}$ is a monomial of degree $(d-i)$. Then, by bilinearity of the j-product, $V_j(y) := \sum_{C' \in F_j} val(C') = y \times_p h_j$, where h_j is a polynomial of degree $(d-i)$. Note that $\sum_{C' \in S_j} val(C') = V_j(g_j)$ and therefore $\sum_{C' \in S_j} val(C') = g_j \times_p h_j$. □

2.3 Exact characterisation of the complexity

We will use the number of $+$-gates of a UPT circuit in normal form as an estimate of its size. The following lemma shows that this is a good measure of overall size.

Lemma 2.5

Let C be a UPT circuit in normal form with s $+$-gates. Then we can transform C into a new UPT circuit in normal form, without changing the shape, with s $+$-gates and at most s^2 $\times$-gates.

Proof. Recall that the circuits are of $\times$ fan-in at most two. Denote by s_i the number of $+$-gates on the i-th layer of C. If C has strictly more than s^2 $\times$-gates, then one layer i contains strictly more than s_i^2 $\times$-gates. It means that two different $\times$-gates on the same layer perform the same computation; therefore one of them can be deleted and its output replaced by the output of the other one. □

We will use this notion of size to get an exact expression of the complexity of computing a given polynomial with a UPT circuit in normal form. To do this, we create a complexity measure which is an extension for UPT circuits in normal form of the one given by Nisan [38] for algebraic branching programs. For a given homogeneous polynomial f 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 on X 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. We can rephrase this with our notation by saying that the matrix $M^i(P)$ is exactly the matrix $M[f, \Pi_i]$, where $\Pi_i = ([1,i],[i+1,d])$. Intuitively, the rank of the matrix $M^{(i)}(f)$ 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.

To make this more precise, we need to define a partition corresponding to a node in the shape of a UPT circuit computing a polynomial of degree d.

Definition 2.6

Given any integer d and any pairs (i, p) with $i + p \leq d$, we define the partition $\Pi_{(i,p)}$ of $[d]$ so that $\Pi_{(i,p)} = (Y_{(i,p)}, Z_{(i,p)})$ where:

- $Y_{(i,p)} = [p + 1, p + i]$
- $Z_{(i,p)} = [d] \setminus Y_{(i,p)} = [1, p] \cup [p + i + 1, d]$

For notational convenience, if T is a tree with d leaves, then for any $v \in T$, we define Π_v to be $\Pi_{\mathrm{type}(v)}$.

We can now express exactly the number of additions needed to compute a given polynomial by a UPT circuit in normal form (recall that the rank of a matrix $M[f, \Pi]$ is written $\mathrm{rank}(f, \Pi)$).

Theorem 2.7

Let f be a homogeneous polynomial of degree d over the set X of variables and T a shape with d leaves. Then the minimal number of addition gates needed to compute f by a UPT circuit in normal form with shape T is exactly equal to $\sum_{v \in T} \mathrm{rank}(f, \Pi_v)$.

Proof. Fix a UPT circuit C in normal form with shape T which computes f. Fix also $v \in T$ a node of type (i, p) and let $\Phi_1, \ldots, \Phi_{k_v}$ be all the $(v, +)$-gates in C. Let $f = \sum_{j=1}^{k_v} g_j \times_p h_j$ be the decomposition given by Proposition 2.4. To simplify notation, set also $k = k_v$.

First step: decomposition of the matrix $M[f, \Pi_v]$ as $L^v R^v$. We show that $M[f, \Pi_v]$ is the product of two "small" matrices L^v and R^v:

- R^v is a matrix of size $k \times n^i$. Rows are indexed by all gates $\Phi_1, \ldots, \Phi_k$. Columns are indexed by monomials $m \in \mathcal{M}_i(X)$. $R^v_{t,m}$ is the coefficient of the monomial m in the polynomial g_t computed by the gate Φ_t.
- L^v is a matrix of size $n^{d-i} \times k$. Rows are indexed by all pairs $(m_1, m_2) \in \mathcal{M}_p(X) \times \mathcal{M}_{d-p-i}(X)$. Columns are indexed by all gates $\Phi_1, \ldots, \Phi_k$.

$L^v_{(m_1,m_2),t}$ is the coefficient of the monomial $m_1 m_2$ in the polynomial computed by the circuit where Φ_t is replaced by an input gate with value 1. That is: $L^v_{(m_1,m_2),t}$ is the coefficient of the monomial $m_1 m_2$ in the polynomial h_t.

One can easily verify that $M[f, \Pi_v] = L^v R^v$.

Second step: lower bound. Since $\mathrm{rank}(f, \Pi_v) \leq \mathrm{rank}\,(L^v) \leq k$, the number k of $(v, +)$ gates must be at least $\mathrm{rank}(f, \Pi_v)$. Therefore, considering all nodes in T, we have just proved that the number of addition gates is at least $\sum_{v \in T} \mathrm{rank}(f, \Pi_v)$.

Third step: upper bound. We prove that if $\mathrm{rank}(f, \Pi_v) < k$, we can delete one $(v, +)$-addition gate in the circuit. We will possibly be increasing at the same time the number of $\times$-gates but, thanks to Lemma 2.5, this is innocuous. If $\mathrm{rank}\,(L^v) = \mathrm{rank}\,(R^v) = k$, then, by a linear algebra argument, $\mathrm{rank}(f, \Pi_v)$ should also be k. Thus, either L^v or R^v is of rank strictly less than k.

If $\mathrm{rank}\,(R^v) < k$, then one row (let us say, w.l.o.g., the first row) of R^v is a linear combination of the other rows. Going back to the meaning of the matrix, it means that the polynomial g_1 computed by the gate Φ_1 is a linear combination of the polynomials $g_2, \ldots, g_k$ computed by the gates $\Phi_2, \ldots, \Phi_k$. Let us say $g_1 = \sum_{i=2}^k c_i g_i$ for $c_i \in \mathbb{F}$. We construct a new circuit where Φ_1 is deleted. We denote by $\Psi_1, \ldots, \Psi_m$ the $\times$-gates which receive as input Φ_1. In the new circuit, we create $(k-1)$ copies of $\Psi_1, \ldots, \Psi_m$ — namely $\Psi^2_1, \ldots, \Psi^2_m, \Psi^3_1, \ldots \Psi^3_m, \ldots, \Psi^k_1, \ldots, \Psi^k_m$. Ψ^i_j does exactly the same computation as Ψ_j, but instead of taking Φ_1 as input, it takes Φ_i. Finally, an addition gate in the old circuit which took as input a Ψ_j now takes $\sum_{i=2}^k c_i \Psi^i_j$ as input.

If $\mathrm{rank}\,(L^v) < k$, then one column (let us say, w.l.o.g., the first column) of L^v is a linear combination of the other columns. This means that there are constants $c_2, \ldots, c_k$ such that $h_1 = \sum_{j=2}^k c_j h_j$. Let $\Gamma_1, \ldots, \Gamma_m$ be all the coefficients on the input edges of Φ_1 coming respectively from multiplication gates $\Psi_1, \ldots, \Psi_m$. In the new circuit, we delete Φ_1 and we add for all $1 \leq l \leq m, 2 \leq j \leq k$ an edge between Ψ_l and Φ_j with the coefficient $c_j \Gamma_l$. The new circuit computes the polynomial $\sum_{j=2}^k (g_j + c_j g_1) \times_p h_j$. By bilinearity of the j-product, this is equal to

$$\begin{aligned}
& \sum_{j=2}^k g_j \times_p h_j \quad + \quad \sum_{j=2}^k (c_j g_1) \times_p h_j \\
= \; & \sum_{j=2}^k g_j \times_p h_j \quad + \quad g_1 \times_p \left(\sum_{j=2}^k (c_j h_j) \right) \\
= \; & \sum_{j=2}^k g_j \times_p h_j \quad + \quad g_1 \times_p h_1 \\
= \; & f.
\end{aligned}$$

□

Remark 2.8

When the shape is a comb (thus corresponding to an ABP), then $p = 0$ in the proof above, and $M[f, \Pi_v]$ is the usual matrix $M^{(i)}$ of Nisan [38]. Since the number of additions gates in the circuit corresponds exactly to the number of vertices in the ABP, our result is a direct extension of Nisan's.

2.4 Comparison with other classes

2.4.1 UPT vs. Skew-circuits

In this section we show that the classes of polynomials computed by polynomial-size UPT circuits on the one hand, and by polynomial-size skew circuits on the other hand, are incomparable.

Definition 2.9: Palindrome polynomial

Assume d is even. The palindrome of degree d over the set X of variables is:

$$Pal^d(X) := \sum_{m \in \mathcal{M}_{d/2}(X)} m.\bar{m},$$

where $\bar{m}$ is the mirror of m (e.g $\bar{m} = x_3x_2x_1$ if $m = x_1x_2x_3$).

Remark 2.10

Observe that we can also define a palindrome polynomial when the integer d is odd, but this is not needed in this manuscript.

UPT $\not\subset$ Skew

It is easy to construct a small UPT skew circuit for $Pal^d(X)$ by using the following inductive formula:

$$Pal^d(X) = \sum_{i=1}^{n} x_i Pal^{d-2}(X) x_i$$

We can then use the construction for $Pal^d(X)$ to compute the square of the palindrome $(Pal^d(X))^2 = Pal^d(X) \times Pal^d(X)$ with a UPT circuit as well. But note that [29] shows that the square of the palindrome polynomial needs exponential-size skew circuits: therefore, UPT is not included in Skew. Observe that this also shows that Skew is strictly included in rot-PT since UPT is included in rot-PT.

Skew $\not\subset$ UPT

In the remainder of this section we construct a polynomial computable by a skew circuit of polynomial size but not by UPT circuits of polynomial size. The idea is the following: given a UPT circuit in normal form of degree d with any shape T, there is always a node $v \in T$ for which $\text{type}(v) = (i, p)$ where $i \in [\frac{d}{3}, \frac{2d}{3}], p \in [0, d-i]$ (see Lemma 2.11 below). We then consider a polynomial such that the associated matrices $M[f, \Pi_{(i,p)}]$ have an exponential rank for all $i \in [\frac{d}{3}, \frac{2d}{3}], p \in [0, d-i]$. According to the previous section, this means that computing the polynomial by UPT circuits requires at least an exponential number of gates.

Lemma 2.11

Given a UPT circuit in normal form computing a polynomial of degree d with shape T, there is always a node $v \in T$ for which $\text{type}(v) = (i, p)$ satisfies $i \in [\frac{d}{3}, \frac{2d}{3}], p \in [0, d-i]$.

Proof. It is sufficient to prove that there is a $+$-gate of degree $i \in [\frac{d}{3}, \frac{2d}{3}]$: the condition on p follows immediately from the definition of the type. Let Φ be a $\times$-gate of degree $> \frac{2}{3}d$ as close as possible to the leaves. Let Ψ_1, Ψ_2 be the inputs of Φ and i, j their respective degree. We have $i + j > \frac{2d}{3}, 1 \leq i \leq \frac{2d}{3}, 1 \leq j \leq \frac{2d}{3}$. These conditions force i or j to be in $[\frac{d}{3}, \frac{2d}{3}]$. □

Definition 2.12: Moving palindrome

The moving palindrome of degree n over the set $X \cup \{w\}$ of $n+1$ variables is:

$$Pal^n_{mov}(X, w) := \sum_{l \in [0, \frac{2n}{3}]} w^l Pal^{\frac{n}{3}}(X) w^{\frac{2n}{3} - l},$$

where w is a fresh variable (distinct from the X).

The first proposition below is easy and is given without proof. The second is an application of our size characterization for UPT circuits in normal form.

Proposition 2.13

$Pal^n_{mov}(X, w)$ is computable by a skew circuit of size polynomial in n.

Sketch of the proof. The palindrome is computable by a small skew circuit. Therefore, by definition, the moving palindrome is computable by a small sum of small skew circuits, hence a small skew circuit. □

Proposition 2.14

Computing $Pal^n_{mov}(X, w)$ with a UPT circuit in normal form requires at least $n^{n/6}$ gates.

Proof. Consider a UPT circuit in normal form C computing Pal^n_{mov}. Thanks to Lemma 2.11, we know that there is always a node v in the shape for which the type (i, p) satisfies $i \in [\frac{n}{3}, \frac{2n}{3}], p \in [0, n-i]$. To apply Theorem 2.7, it is enough to show that for all such (i, p), $\mathrm{rank}(Pal^n_{mov}, \Pi_{(i,p)}) \geq n^{n/6}$. This will be possible since for each such type, there is a polynomial in the sum that defines Pal^n_{mov} which has a large rank and the other polynomials will not interfere.

Let us fix a particular $(i, p), i \in [\frac{n}{3}, \frac{2n}{3}], p \in [0, n-i]$. Because $i \leq \frac{2n}{3}$ we have $p + (n - p - i) \geq \frac{n}{3}$. Then one of the two following cases occurs.
Case $p \geq \frac{n}{6}$. In this case we show first that

$$\mathrm{rank}(Pal^n_{mov}, \Pi_{(i,p)}) \geq \mathrm{rank}(w^{p-\frac{n}{6}} Pal^{\frac{n}{3}}(X) w^{n-p-\frac{n}{6}}, \Pi_{(i,p)}).$$

Indeed by definition,

$$M[Pal^n_{mov}, \Pi_{(i,p)}] = \sum_{l \in [0, \frac{2n}{3}]} M[w^l Pal^{\frac{n}{3}}(X) w^{\frac{2n}{3}-l}, \Pi_{(i,p)}];$$

And note then that, if (a, b) is a coordinate of a non-zero coefficient of

$$M[w^{p-\frac{n}{6}} Pal^{\frac{n}{3}}(X) w^{n-p-\frac{n}{6}}, \Pi_{(i,p)}]$$

and (a', b') is a coordinate of a non-zero coefficient of

$$M[w^l Pal^{\frac{n}{3}}(X) w^{\frac{2n}{3}-l}, \Pi_{(i,p)}],$$

with $l \neq p - \frac{n}{6}$, then $a \neq a'$ and $b \neq b'$. Finally, observe that in this case, every row and column of $M[w^{p-\frac{n}{6}} Pal^{\frac{n}{3}}(X) w^{n-p-\frac{n}{6}}, \Pi_{(i,p)}]$ contains at most one non-zero coefficient and there are exactly $n^{n/6}$ non-zero coefficients. Thus:

$$\operatorname{rank}(Pal^n_{mov}, \Pi_{(i,p)}) \geq \operatorname{rank}(w^{p-\frac{n}{6}} Pal^{\frac{n}{3}}(X) w^{n-p-\frac{n}{6}}, \Pi_{(i,p)}) \geq n^{n/6}.$$

Case $n - p - i \geq \frac{n}{6}$. With similar arguments, we have this time

$$\operatorname{rank}(Pal^n_{mov}, \Pi_{(i,p)}) \geq \operatorname{rank}(w^{p+i-\frac{n}{6}} Pal^{\frac{n}{3}}(X) w^{\frac{5n}{6}-p-i}, \Pi_{(i,p)}) \geq n^{n/6}. \quad \square$$

Remark 2.15

Observe that this implies that UPT is strictly included in rot-PT since Skew is included in rot-PT. Moreover, as the moving palindrome is computable by a small sum of small skew circuits, it is also computable by a small k-PT circuit. Hence UPT is strictly included in k-PT.

2.4.2 UPT vs. Determinant and Permanent

In this small section, we sketch the proof of lower bounds for the determinant and the permanent that follow from the characterisation of the complexity given in Section 2.3. First, recall the definitions of the two polynomials:

$$\mathrm{PERM}_n = \sum_{s \in S_n} \prod_{i=1}^{n} x_{1,s(1)} \cdots x_{n,s(n)} \text{ and } \mathrm{DET}_n = \sum_{s \in S_n} \operatorname{sgn}(s) \prod_{i=1}^{n} x_{1,s(1)} \cdots x_{n,s(n)}.$$

To get lower bounds we need to estimate the ranks of certain matrices $M[\star, \Pi_{(i,p)}]$. The following lemma is proved exactly in the same way as Lemma 2 in [38].

Lemma 2.16

For all $i \leq n$, $p \leq n - i$, $\operatorname{rank}(\mathrm{PERM}_n, \Pi_{(i,p)}) = \operatorname{rank}(\mathrm{DET}_n, \Pi_{(i,p)}) = \binom{n}{i}$.

We can now obtain the following lower bounds.

Theorem 2.17

Computing PERM_n or DET_n with a UPT circuit requires at least $\binom{n}{n/3}$ gates.

Proof. By Lemma 2.11, there is a gate v in the shape for which $\mathrm{type}(v) = (i, p)$ and $i \in [\frac{n}{3}, \frac{2n}{3}]$. By Lemma 2.7 and Lemma 2.16, the number of gates needed is at least $\mathrm{rank}(\mathrm{DET}_n, \Pi_v) = \binom{n}{i} \geq \binom{n}{n/3}$. □

Chapter 3

Variations around parse trees restriction

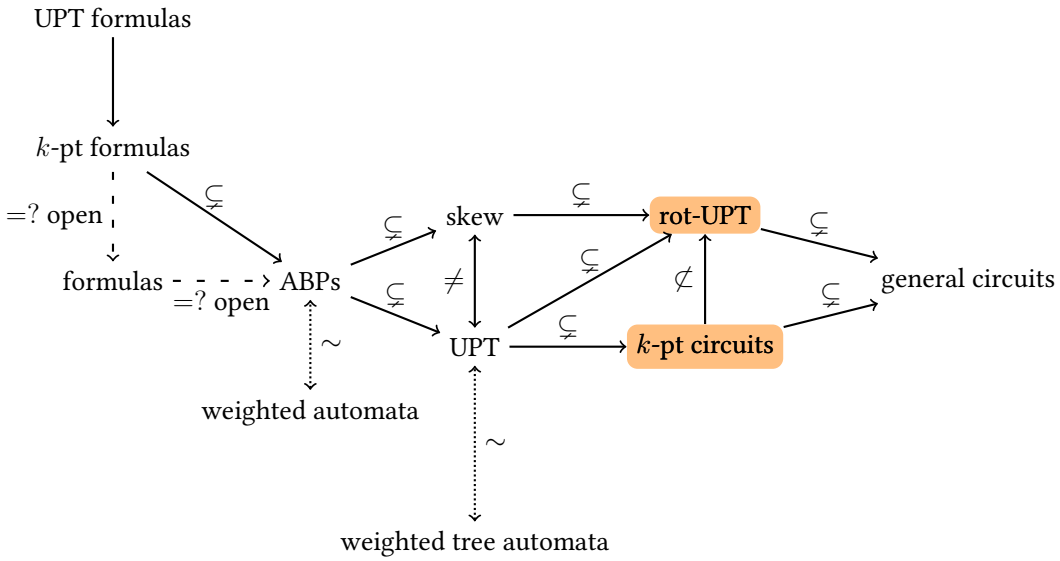


Figure 3.1: You are here.

Overview

Chapter 2 gives a pretty complete understanding of UPT circuits. But remember that the ultimate aim is to prove lower bounds for general circuits, and these ones can have an exponential number of distinct parse trees. Here, we try to get closer to this goal by increasing the number of parse trees that are allowed to appear in a circuit. We do this in two different ways: in Section 3.1, we give lower bounds

for circuits with at most $2^{d^{1/4}}$ unrestricted shapes ("$2^{d^{1/4}}$-PT circuits"). Then, in Section 3.2, we extend previous work on skew circuits to give lower bounds for circuits with an exponential number of shapes; however, this time, the shapes are constrained from the fact they all come from a fixed tree and all its rotations ("rotUPT circuits").
This chapter is based on the following publication:

- Guillaume Lagarde, Nutan Limaye, and Srikanth Srinivasan. Lower bounds and PIT for non-commutative arithmetic circuits with restricted parse trees. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, pages 41:1–41:14, 2017

- Guillaume Lagarde, Nutan Limaye, and Srikanth Srinivasan. Lower bounds and PIT for non-commutative arithmetic circuits with restricted parse trees (extended version). *To appear in Computational Complexity*

3.1 Lower bounds for k-PT circuits

In this section, we show that any k-PT circuit computing a polynomial of degree d where k is subexponential in d cannot compute the polynomial $F_{N,d}$ from Theorem 1.17. We will show that if both k and the size of the circuit are subexponential in d, then there is a Π such that $\text{rel-rank}(f, \Pi) < 1$. For large enough fields $\mathbb{F}$, Theorem 1.17 gives us a polynomial that is full rank w.r.t. all partitions and we thus get a lower bound for computing this polynomial by k-PT circuits.

Proof outline. We first show that any k-PT circuit C of size s can be written as a sum of k *UPT* circuits $C_1, \ldots, C_k$ whose shapes are the different shapes that appear in C and whose sizes are at most s^2. At this point, for each C_i, Lemma 3.2 allows us to find many partitions $\Pi_{i,1}, \ldots, \Pi_{i,t}$ such that $\text{rel-rank}(C_i, \Pi_{i,j})$ is small for each $i \in [k]$ and $j \in [t]$. However, we would like to find a common partition Π so that $\text{rel-rank}(C_i, \Pi)$ is small for each $i \in [t]$.

To do this, we use Proposition 1.15 and Lemma 1.16, which together imply that if a partition Π is somewhat close in Hamming distance to either $\Pi_{i,j}$ or $-\Pi_{i,j}$ for any $j \in [t]$, then $\text{rel-rank}(f, \Pi)$ is small as well. In fact, a close look at the parameters tells us that to obtain a superpolynomial lower bound, it suffices for the quantity $\min\{\Delta(\Pi, \Pi_{i,j}), \Delta(\Pi, -\Pi_{i,j})\}$ to be somewhat smaller than the worst case, which is $d/2$. Our aim, therefore, is to choose a Π such that for each $i \in [k]$, there exists a $j \in [t]$ so that the above occurs.

A natural choice for such a Π is to choose a uniformly random $\Pi = (Y, Z)$ of $[d]$. Analyzing the probability that $\min\{\Delta(\Pi, \Pi_{i,j}), \Delta(\Pi, -\Pi_{i,j})\}$ is close to

$d/2$ for each $j \in [t]$ is a simple probabilistic problem that we solve below. As this probability is low, we obtain a lower bound.

We start with a decomposition lemma for k-PT circuits. The lemma is stated in slightly greater generality since it is used in later sections as well.

Lemma 3.1: Decomposition for k-PT circuits

Let C be a k-PT circuit (resp. formula) of size s with $\mathcal{T}(C) = \{T_1, \ldots, T_k\}$ computing $f \in \mathbb{F}\langle X \rangle$. Then there exist normal form UPT circuits (resp. formulas) $C_1, \ldots, C_k$ of size at most s^2 each such that $\mathcal{T}(C_i) = \{T_i\}$ and $f = \sum_{i=1}^{k} f_i$, where f_i is the polynomial computed by C_i.

Proof. Let C be as in the statement. We show how to construct k UPT circuits (resp. formulas) $C_1, \ldots, C_k$ of size at most s^2, of shapes $T_1, \ldots, T_k$ respectively, computing $f_1, \ldots, f_k$ respectively such that each f_i is equal to the sum of the monomials computed by all the parse formulas of C of shape T_i. Given this, the polynomial f, which is equal to the sum of all monomials computed by all parse formulas of C, will be equal to $\sum_{i=1}^{k} f_i$ and the lemma will be proved.

Construction of C_i.

- **The gates** of C_i are denoted by pairs of the form (Φ, v). For each gate $\Phi \in C$ and for each node $v \in V(T_i)$ such that $\deg(v) = \deg(\Phi)$, we initially add a gate (Φ, v) to the circuit C_i.

- **Edges:**

  - If $\Phi \in C$ is an addition gate with children $\Psi_1, \ldots, \Psi_t$, then (Φ, v) is an addition gate in C_i with children $(\Psi_1, v), \ldots, (\Psi_t, v)$.
  - If $\Phi \in C$ is a multiplication gate with children $\Psi_1, \ldots, \Psi_t$ (in this order), then (Φ, v) is a multiplication gate with children $(\Psi_1, v_1), \ldots, (\Psi_t, v_t)$, *as long as* the children of v in T_i are exactly $v_1, \ldots, v_t$ (in this order) with $\deg(v_j) = \deg(\Psi_j)$ for each j. Otherwise, we label (Ψ, v) with 0.
  - If Φ is an input gate labelled by $x \in X$ and v a leaf node, then the gate (Φ, v) is also an input gate with the same label.

  Notice that the size of C_i is upper bounded by s^2. Further, any parse formula that does not contain any of the nodes labelled 0 has shape T_i.

We prove by induction (on any toplogical orderings of T_i and C) that for any $v \in V(T)$ and Φ in C such that $\deg(\Phi) = \deg(v)$, the gate (Φ, v) in C_i computes the sum of all parse formulas C' of C starting at Φ with the shape $T_i[v]$, where $T_i[v]$ is the subtree of i rooted at v. This will prove that the output gate of C_i computes the sum of the monomials computed by all the parse formulas of C of shape T_i. This is clearly true for the leaves.

Take now any (Φ, v) in C_i. We assume it is a multiplication gate (the other case is similar). Assume that the children of $\Phi \in C$ are $\Psi_1, \ldots, \Psi_t$ and that the children of $v \in T_i$ are $v_1, \ldots, v_r$. If either $r \neq t$ or there is an $a \in [t]$ such that $\deg(v_a) \neq \deg(\Psi_a)$, then there are no parse formulas starting at Φ of shape $T_i[v]$ and hence the gate (Φ, v) which is labelled with 0 computes the correct polynomial. So we now assume that $r = t$ and $\deg(v_a) \neq \deg(\Psi_a)$ for each $a \in [t]$.

Let us denote by S' the set of parse formulas C' of C starting at Φ with a shape $T_i[v]$, and S'_1 (respectively $S'_2, \ldots, S'_t$) the set of parse formulas starting at the gates Ψ_1 (resp. $\Psi_2, \ldots, \Psi_t$) with a shape $T_i[v_1]$ (resp. $T_i[v_2], \ldots, T_i[v_t]$).

The set S' is obtained by taking all possible combinations of parse formulas coming from $S'_1, \ldots, S'_t$. In symbols

$$\sum_{C' \in S'} val(C') = \prod_{j=1}^{t} \sum_{C'' \in S'_j} val(C'')$$

If we denote by $P(\Psi_j, v_j)$ each polynomial computed by a gate (Ψ_j, v_j) in C_i, we get by induction hypothesis that

$$\sum_{C' \in S'} val(S) = \prod_{j=1}^{t} P(\Psi_j, v_j)$$

and hence

$$\sum_{C' \in S'} val(S) = P(\Phi, v)$$

as wanted.

Finally, note that some of the leaves of the circuit are labelled by the constant 0. To eliminate this, we can repeatedly apply the following procedure. If Φ is labelled with 0 and feeds into a $\times$ gate Ψ, then remove Φ and all wires feeding into Ψ, and relabel Ψ with 0. If Φ is labelled with 0 and feeds into a $+$ gate Ψ, then simply remove Φ and if Φ has no inputs left, then relabel it with 0. This process produces a UPT circuit with shape T_i and size at most s^2. Further, since each gate is already associated with a node of T in a natural way, the circuit C_i is already in normal form. □

We now analyze the probability that $\mathrm{rank}(C, \Pi)$ is close to full for a *random* partition Π of $[d]$.

Lemma 3.2

Let C be a UPT circuit in normal form over $\mathbb{F}\langle X\rangle$ of size $s = N^c$, where $N = |X|$, and f the homogeneous polynomial of degree d computed by C. Let Π be a uniformly random partition of the variables of $[d]$ into two sets. Then for any parameter $b \in \mathbb{N}$,

$$\Pr_{\Pi}\left[\mathrm{rank}(f, \Pi) \geq N^{d/2-b}\right] \leq \exp(-\Omega(d/(b+c)^2)).$$

Postponing the proof of the above lemma to Section 3.1.1, we show that it implies the following lower bound for homogeneous non-commutative circuits with few parse trees. Note that when the field $\mathbb{F}$ is large enough, this proves a lower bound for $F_{N,d}$ from Theorem 1.17.

Theorem 3.3

Assume that $N, d \geq 2$ are growing integer parameters with d being even. Let $F \in \mathbb{F}\langle X\rangle$ be any polynomial such that for each balanced partition Π, $\mathrm{rank}(F, \Pi) = N^{d/2}$. Then, for any constant $\varepsilon \in (0, 1)$, any circuit that computes F and satisfies $|\mathcal{T}(C)| = k \leq 2^{d^{\frac{1}{3}-\varepsilon}}$ must have size at least or more than $N^{d^{\frac{1}{3}-\frac{\varepsilon}{2}}}$.

Proof. Let C be any circuit of size $s \leq N^c$ for $c = d^{1/3-\varepsilon/2}$ with $|\mathcal{T}(C)| = k \leq 2^{d^{1/3-\varepsilon}}$ and computing $f \in \mathbb{F}\langle X\rangle$. We show that there is a balanced partition Π such that $\mathrm{rank}(f, \Pi) < N^{d/2}$. This will prove the theorem.

To show this, we proceed as follows. Using Lemma 3.1, we can write $f = \sum_{i\in[k]} f_i$ where each $f_i \in \mathbb{F}\langle X\rangle$ is computed by a normal form UPT circuit C_i of size at most $s^2 \leq N^{2c}$.

Fix any $i \in [k]$. By Lemma 3.2, the number of partitions Π for which $\mathrm{rank}(f_i, \Pi) \geq N^{\frac{d}{2}-c}$ is at most $2^d \cdot \exp(-\Omega(d/c^2))$. In particular, since the number of balanced partitions is $\binom{d}{d/2} = \Theta(\frac{2^d}{\sqrt{d}})$, we see that for a random *balanced* partition Π,

$$\Pr_{\Pi \text{ balanced}}\left[\mathrm{rank}(f_i, \Pi) \geq N^{d/2-c}\right] \leq O(\sqrt{d}) \cdot \exp(-\Omega(d/c^2)) \leq \exp(-d^{1/3}).$$

Say f_i is bad for Π if $\mathrm{rank}(f_i, \Pi) \geq N^{d/2-c}$. By the above, we have

$$\Pr_{\Pi \text{ balanced}}\left[\exists i \in [k] \text{ s.t. } f_i \text{ bad for } \Pi\right] \leq k \cdot \exp(-d^{1/3}) \leq 2^{d^{1/3-\varepsilon}} \cdot \exp(-d^{1/3}) < 1.$$

In particular, there is a balanced Π such that no f_i is bad for Π. Fix such a balanced partition Π. By the subadditivity of rank, we have

$$\begin{aligned}\operatorname{rank}(f,\Pi) &\le \sum_{i\in[k]} \operatorname{rank}(f_i,\Pi) \le k\cdot N^{d/2-c} \le 2^{d^{1/3-\varepsilon}}\cdot N^{d/2-c}\\ &= N^{d/2}\cdot \exp(O(d^{1/3-\varepsilon}) - \Omega(d^{1/3-\varepsilon/2})) < N^{d/2}.\end{aligned}$$

This proves the theorem. □

3.1.1 Proof of Lemma 3.2

Notation. Recall from Chapter 1 that we identify each partition Π with an element of $\{-1,1\}^d$. Given partitions $\Pi_1,\Pi_2 \in \{-1,1\}^d$ we use $\langle \Pi_1,\Pi_2\rangle$ to denote their inner product: i.e., $\langle \Pi_1,\Pi_2\rangle := \sum_{i\in[d]} \Pi_1(i)\Pi_2(i)$. Note that the Hamming distance $\Delta(\Pi_1,\Pi_2)$ is

$$\Delta(\Pi_1,\Pi_2) = \frac{d}{2} - \frac{1}{2}\langle \Pi_1,\Pi_2\rangle. \tag{3.1}$$

Let $\mathcal{T}(C) = \{T\}$. Recall that $|\mathcal{L}(T)| = d$ and by Lemma 2.1, we can assume that the fan-in of each internal node of T is bounded by 2. For any $u \in \mathcal{I}(T)$ (recall that $\mathcal{I}(T)$ is the set of internal nodes of T), let $\mathcal{L}(u)$ denote the set of leaves of the subtree rooted at u. We identify each leaf $\ell \in T$ with $\operatorname{pos}(\ell) \in [d]$. For each $u \in \mathcal{I}(T)$, we define the partition Π_u by $\Pi_u(\ell) = -1$ iff $\ell \in \mathcal{L}(u)$.

For $\gamma > 0$, define a partition Π to be *γ-correlated to* T if for each $u \in \mathcal{I}(T)$, we have $\left|\sum_{\ell\in\mathcal{L}(u)} \Pi(\ell)\right| \le \gamma$.

For the rest of this section, let f, N, b, c, d as in the statement of Lemma 3.2.

Lemma 3.2 immediately follows from Claims 3.4 and 3.5, stated below.

Claim 3.4

Let Π be any partition of $[d]$ such that $\operatorname{rank}(f,\Pi) \ge N^{d/2-b}$. Then Π is $O(b+c)$-correlated to T.

Proof. We know from Theorem 2.7 and Proposition 1.15 that for each $u \in \mathcal{I}(T)$, $\operatorname{rank}(f,\Pi_u), \operatorname{rank}(f,-\Pi_u) \le N^c$. If Π is a partition such that either $\Delta(\Pi,\Pi_u)$ or $\Delta(\Pi,-\Pi_u)$ is strictly smaller than $\frac{d}{2} - (b+c)$ for *some* $u \in \mathcal{I}(T)$, then by Lemma 1.16 we would have $\operatorname{rank}(f,\Pi) < N^{d/2-b}$.

Thus, if $\operatorname{rank}(f,\Pi) \ge N^{d/2-b}$, we must have $\min\{\Delta(\Pi,\Pi_u),\Delta(\Pi,-\Pi_u)\} \ge \frac{d}{2} - (b+c)$ for each $u \in \mathcal{I}(T)$. By (3.1), this means that for each $u \in \mathcal{I}(T)$, $|\langle \Pi,\Pi_u\rangle| \le \gamma$ for some $\gamma = O(b+c)$.

Let v be the root of T. Note that $\Pi_v \in \{-1,1\}^d$ is the vector with all its entries being -1. Hence, we have for any $u \in \mathcal{I}(T)$,

$$\left|\sum_{\ell \in \mathcal{L}(u)} \Pi(\ell)\right| = \left|\langle \Pi, \frac{-(\Pi_u + \Pi_v)}{2}\rangle\right| \leq \frac{1}{2}(|\langle \Pi, \Pi_u\rangle| + |\langle \Pi, \Pi_v\rangle|) \leq O(\gamma).$$

This proves the claim. □

Claim 3.5

Say $\Pi \in \{-1,1\}^d$ is chosen uniformly at random and $\gamma \leq \sqrt{d}$. Then

$$\Pr_{\Pi}[\Pi \text{ is } \gamma\text{-correlated to } T] \leq \exp(-\Omega(\frac{d}{\gamma^2})).$$

The following subclaim is useful for proving Claim 3.5.

Subclaim 3.6

Assume that $r, t \in \mathbb{N}$ such that $rt \leq d/4$. Then we can find a sequence $u_1, \ldots, u_r \in \mathcal{I}(T)$ such that for each $i \in [r]$ we have $|\mathcal{L}(u_i) \setminus \bigcup_{j=1}^{i-1} \mathcal{L}(u_j)| \geq t$.

Proof. Consider the following 'greedy' procedure for choosing the u_i. Order the nodes of $\mathcal{I}(T)$ in topological order (recall that the edges of T are directed toward the root). We choose u_1 to be the least node in this order so that $|\mathcal{L}(u_1)| \geq t$ (such a node must exist since there are $d \geq t$ leaves in T). Further, having chosen $u_1, \ldots, u_i$ we choose u_{i+1} to be the least node greater than or equal to $u_1, \ldots, u_i$ in the topological order such that $|\mathcal{L}(u_{i+1}) \setminus \bigcup_{j=1}^{i} \mathcal{L}(u_j)| \geq t$.

To argue that this process produces a sequence of size at least r, note that for each $i \geq 0$, $|\mathcal{L}(u_{i+1}) \setminus \bigcup_{j=1}^{i} \mathcal{L}(u_j)| \leq 2t$. This is because the fan-in of u_{i+1} in T is at most 2 and by assumption, for each child u' of u_{i+1}, we have $|\mathcal{L}(u') \setminus \bigcup_{j=1}^{i} \mathcal{L}(u_j)| < t$. Thus for each $i \geq 0$, we have $|\bigcup_{j=1}^{i+1} \mathcal{L}(u_j)| \leq 2t(i+1)$.

In particular, if $i + 1 < r$, we have $|\bigcup_{j=1}^{i+1} \mathcal{L}(u_j)| < 2tr \leq d/2$. Thus, for v being the root of the tree, we have $|\mathcal{L}(v) \setminus \bigcup_{j=1}^{i+1} \mathcal{L}(u_j)| > d/2 \geq t$. In particular, there is at least one node u of the tree such that $|\mathcal{L}(u) \setminus \bigcup_{j=1}^{i+1} \mathcal{L}(u_j)| \geq t$. This allows us to extend the sequence further. □

Proof of Claim 3.5. We apply Subclaim 3.6 with $t = \Theta(\gamma^2)$ and $r = \Theta(d/\gamma^2)$ to get a sequence $u_1, \ldots, u_r \in \mathcal{I}(T)$ such that for each $i \in [r]$, we have $|\mathcal{L}(u_i) \setminus \bigcup_{j=1}^{i-1} \mathcal{L}(u_j)| \geq t$.

By the definition of γ-correlation, we have

$$\Pr_{\Pi}[\Pi \text{ } \gamma\text{-correlated to } T] \leq \Pr_{\Pi}\left[\forall i \in [r], \left|\sum_{\ell \in \mathcal{L}(u_i)} \Pi(\ell)\right| \leq \gamma\right]$$
$$\leq \prod_{i \in [r]} \Pr_{\Pi}\left[\left|\sum_{\ell \in \mathcal{L}(u_i)} \Pi(\ell)\right| \leq \gamma \;\middle|\; \{\Pi(\ell) \mid \ell \in \bigcup_{j<i} \mathcal{L}(u_j)\}\right] \tag{3.2}$$

Fix any $i \in [r]$ and $\Pi(\ell)$ for each $\ell \in \mathcal{L}_{<i} := \bigcup_{j<i} \mathcal{L}(u_j)$. Note that the event $|\sum_{\ell \in \mathcal{L}(u_i)} \Pi(\ell)| \leq \gamma$ is equivalent to $\sum_{\ell \in \mathcal{L}(u_i) \setminus \mathcal{L}_{<i}} \Pi(\ell) \in I$ for some interval I of length $2\gamma = O(\sqrt{t})$. This is the probability that the sum of at least t independent uniformly chosen $\{-1, 1\}$-valued random variables lies in an interval of length $O(\sqrt{t})$. By the Central Limit theorem, this can be bounded by $1 - \Omega(1)$.

By (3.2), we get

$$\Pr_{\Pi}[\Pi \text{ } \gamma\text{-correlated to } T] \leq \exp\{-\Omega(r)\}$$

which gives the statement of the claim. □

3.2 Lower bounds for circuits with rotations of one parse tree

Given two parse trees T_1 and T_2 with the same number of leaves, we say that T_1 is a *rotation* of T_2, denoted $T_1 \sim T_2$, if T_1 can be obtained from T_2 by repeatedly reordering the children of various nodes in T_2. Clearly, $\sim$ is an equivalence relation. We use $[\![T]\!]$ to denote the equivalence class of tree T. We say that a homogeneous circuit C is *rotation UPT or rotUPT* if there is a tree T such that $\mathcal{T}(C) \subseteq [\![T]\!]$. The tree T is said to be a *template* for C.

Our main result in this section is the following.

Theorem 3.7

Let C be a rotUPT circuit of size s computing a polynomial $f \in \mathbb{F}\langle X \rangle$ of degree d over N variables, then there exists a partition $\Pi = \Pi_C$ such that $\text{rel-rank}(f, \Pi)$ is at most $\text{poly}(s) \cdot N^{-\Omega(d)}$.

In particular, we get the following corollary.

Corollary 3.8

Let $N, d \in \mathbb{N}$ be parameters with d even. Let $|\mathbb{F}| > q_0(N, d)$ where $q_0(N, d)$ is as in Theorem 1.17. Then, any rotUPT circuit for $F_{N,d}$ over $\mathbb{F}$ has size $N^{\Omega(d)}$.

Proof outline. Our starting point is a decomposition lemma of Hrubeš et al. [17] and its subsequent strengthening in [30]. As noted in [17], any homogeneous polynomial f of degree d computed by a non-commutative circuit of small size can be written as a small sum of polynomials of the form $g \times_j h$, where g, h are homogeneous polynomials of degrees in the range $[d/3, 2d/3]$. It was observed in [30] that for certain special kinds of circuits, this statement can be strengthened to yield a decomposition where each term $g \times_j h$ satisfies $\deg(g) = d'$ for some fixed $d' \in [d]$. This was used, with $d' = 3d/4$, to prove a lower bound for non-commutative skew circuits in [30].

Here, we use a similar idea to show that whenever the template T has a node of degree $d' \in [(3/4)d, (11/12)d]$, then the corresponding rotUPT circuit C is low rank w.r.t. the partition $\Pi_2 = (Y_2, Z_2)$ where $Y_2 = [d/4 + 1, 3d/4]$.

It remains to handle the case when T has no such "large degree" node. To see what happens in this case, consider the extreme case when the root v of T has two children u_1 and u_2 of degree $d/2$ each. In this case, it is easily seen that every parse formula computes two monomials of degree $d/2$ which are multiplied together to compute the output monomial. From here, it is not hard to show that the polynomial computed by the circuit C can be written as a small sum of polynomials of the form $g \cdot h$ where g and h have degree $d/2$ each. As observed by Nisan [38], this implies that C is now low-rank w.r.t. the partition $\Pi_1 = (Y_1, Z_1)$ where $Y_1 = [d/2]$. The more general case is an abstraction of this argument to allow for a few "low-degree" nodes near the root and also for a single large degree node v to have many small degree children (as opposed to just two in the above example).

We now begin with the proof. First, we will need a decomposition lemma for non-commutative circuits. The following is a variant of lemmas proved in [17, 30] and Chapter 2.

Lemma 3.9: A decomposition lemma for homogeneous circuits

Let C be any homogeneous arithmetic circuit of size s computing $f \in \mathbb{F}\langle X \rangle$ of degree d. Assume that there is some $d' \in [d/2 + 1, d]$ such that every

parse formula C' of C contains a gate computing a (homogeneous) polynomial of degree d'. Let $\Phi_1, \ldots, \Phi_r$ ($r \leq s$) be the set of $\times$ gates computing polynomials of degree d' in C and let $g_1, \ldots, g_r$ be the polynomials they compute (respectively). Then, there exist homogeneous polynomials $h_{i,j}$ ($i \in [r], j \in [0, d - d']$) of degree $d - d'$ such that

$$f = \sum_{i=1}^{r} \sum_{j=0}^{d-d'} g_i \times_j h_{i,j}.$$

Proof. We will first simplify the circuit C so that each gate Φ appears in *some* parse formula of C. If Φ appears in no parse formula of C, then we can remove it from the circuit without changing the polynomial computed by the circuit.

We consider a topological ordering of the gates of the circuit C so that if the gate Φ computes a polynomial of degree at most the degree of the gate Ψ, then Φ appears before Ψ in the ordering. This can be done since C is a homogeneous circuit.

Let $\Psi_1, \ldots, \Psi_p$ ($p \leq s$) be this topological ordering of the gates and let f_k be the polynomial computed at Ψ_k ($k \in [p]$). Let $d_k = \deg(f_k)$. We prove by induction on $k \in [p]$ that if $d_k \geq d'$, then

$$f_k = \sum_{i=1}^{r} \sum_{j=0}^{d-d'} g_i \times_j h_{i,j}^{(k)}. \tag{3.3}$$

for some homogeneous polynomials $h_{i,j}^{(k)}$ of degree $d - d'$ each. Note that this is vacuously true for k such that $\deg(f_k) < d'$.

If the gate Ψ_k is a $\times$ gate of degree $d_k \geq d'$, then we have the following possibilities:

- $d_k = d'$: In this case $f_k = g_i$ for some $i \in [r]$ and hence we can take $h_{i,0}^{(k)} = 1$ and $h_{i',j'}^{(k)} = 0$ for all other i', j' pairs.
- $d_k > d'$: In this case $f_k = f_{k_1} \cdots f_{k_t}$ for some t and $k_1, \ldots, k_t < k$. We observe that one of $d_{k_1}, \ldots, d_{k_t}$ must be at least d'.

  To see this, assume that $d_{k_a} < d'$ for each $a \in [t]$ and consider any parse formula C' containing Ψ_k (such a formula must exist since otherwise Ψ_k would have been removed in the first simplification step). By our assumption on the circuit, C' must also contain some gate Ψ' computing a polynomial of degree exactly d'. Note that Ψ' does not lie in the subcircuit of C' induced by the gate Ψ_k since all the non-output gates of this subcircuit compute polynomials of degree $< d'$ and the output gate computes a polynomial

of degree $> d'$. Also, as $d_k > d'$, Ψ_k does not appear in the subcircuit of C' induced by Ψ'. Consider the parse tree T obtained by unraveling the circuit C'. By the observations above, Ψ_k gives rise to (at least) one node u in T of degree $d_k > d'$ and Ψ' gives rise to a node v in T of degree d'. Thus, the degree of the root is at least $\deg(u) + \deg(v) > 2d' > d$, which is a contradiction since in a homogeneous circuit all parse trees have exactly d leaves.

So we can assume that $\deg(f_{k_a}) \geq d'$ for some $a \in [t]$. Applying the induction hypothesis to f_{k_a} we have

$$f_{k_a} = \sum_{i=1}^{r} \sum_{j=0}^{d'} g_i \times_j h_{i,j}^{(k_a)}.$$

for suitable $h_{i,j}^{(k_a)}$ of degree $d_{k_a} - d'$ each. Thus, we have

$$\begin{aligned} f = f_{k_1} \cdots f_{k_t} &= \sum_{i=1}^{r} \sum_{j=0}^{d_{k_a}-d'} f_1 \cdots f_{k_a-1} \cdot (g_i \times_j h_{i,j}^{(k_a)}) \cdot f_{k_a+1} \cdots f_{k_t} \\ &= \sum_{i=1}^{r} \sum_{j=0}^{d_{k_a}-d'} g_i \times_{j+d_{k_1}+\cdots+d_{k_{a-1}}} (f_1 \cdots f_{k_a-1} h_{i,j}^{(k_a)} f_{k_a+1} \cdots f_{k_t}) \end{aligned}$$

where for the final equality we have used the observation that $(g \times_j h) \times_{j'} h' = g \times_{j+j'} (h \times_{j'} h')$ for any homogeneous polynomials g, h, h' and any relevant j, j'.

For any $j \in [0, d_{k_a} - d']$, we have $j' := j + d_{k_1} + \cdots + d_{k_{a-1}} \in [d_{k_1} + \cdots + d_{k_{a-1}}, d_k - d']$. Hence, setting $h_{i,j'}^{(k)} = f_1 \cdots f_{k_a-1} h_{i,j}^{(k_a)} f_{k_a+1} \cdots f_{k_t}$ for each $j' \in \in [d_{k_1} + \cdots + d_{k_{a-1}}, d_k - d']$, and 0 for all $j' < d_{k_1} + \cdots + d_{k_{a-1}}$, the above yields (3.3) in this case.

If the gate Ψ_k is a $+$ gate of degree $d_k \geq d'$, then it is a linear combination of gates $\Psi_{k_1}, \ldots, \Psi_{k_t}$ of degree d_k each. By induction, for each $a \in [t]$, we have

$$f_{k_a} = \sum_{i=1}^{r} \sum_{j=0}^{d-d'} g_i \times_j h_{i,j}^{(k_a)}.$$

Say $f_k = \sum_a \alpha_a f_{k_a}$ where $\alpha_a \in \mathbb{F}$. Then using the fact that $\times_j$ is bilinear, we get

$$f_k = \sum_{a \in [t]} \alpha_a f_{k_a} = \sum_{i=1}^{r} \sum_{j=0}^{d-d'} \sum_a \alpha_a (g_i \times_j h_{i,j}^{(k_a)}) = \sum_{i=1}^{r} \sum_{j=0}^{d-d'} g_i \times_j (\sum_a \alpha_a h_{i,j}^{(k_a)}))$$

which is of the form required in (3.3). This completes the induction. □

We now prove the main theorem of this section.

Proof of Theorem 3.7. Let C be rotUPT of size s computing a polynomial f of degree d over N variables, and let T be a template for C. Let $\Pi_i = (Y_i, Z_i)$ for $i \in [2]$ be two partitions of $[d]$ with $Y_1 = [d/2]$ and $Y_2 = [d/4 + 1, 3d/4]$. We will show that $\text{rel-rank}(f, \Pi) \leq \text{poly}(s) \cdot N^{-\Omega(d)}$ for at least one $\Pi \in \{\Pi_1, \Pi_2\}$.

We consider two cases:

Case 1: there is a node of degree $d_0 \in [\frac{3}{4}d, \frac{11}{12}d]$ in the template T. Note that every rotation $T' \in [\![T]\!]$ also has a node of degree d_0. Since every parse tree of C is a member of $[\![T]\!]$, this implies that each parse formula C' of C contains a gate of degree d_0. Applying Lemma 3.9 with $d' = d_0$, we see that there are $k \leq s$ homogeneous polynomials $g_1, \ldots, g_k$ of degree d_0 and $k \cdot (d - d_0 + 1)$ homogeneous polynomials $h_{i,j}$ ($i \in [k], j \in [0, d - d_0]$) of degree $d - d_0$ such that:

$$f = \sum_{i=1}^{k} \sum_{j=0}^{d-d_0} g_i \times_j h_{i,j} \tag{3.4}$$

We show that each term of the above decomposition has low relative rank w.r.t. the partition Π_2 defined above. Fix a term $g_i \times_j h_{i,j}$ from the decomposition above. Let $\Pi'_j = (Y'_j, Z'_j)$ where $Y'_j = [j + 1, j + d_0]$. By Corollary 1.14, we see that $\text{rank}(g_i \times_j h_{i,j}, \Pi'_j) \leq 1$.

A straightforward calculation shows that $\Delta(\Pi'_j, \Pi_2) = d_0 - \frac{d}{2} = \frac{d}{2} - \Omega(d)$ for all j. Hence, by Lemma 1.16, we see that $\text{rank}(g_i \times_j h_{i,j}, \Pi_2) \leq N^{(d/2)-\Omega(d)}$ and hence $\text{rel-rank}(g_i \times_j h_{i,j}, \Pi_2) \leq N^{-\Omega(d)}$ for each i, j.

Using (3.4) and the subadditivity of rank, we see that $\text{rel-rank}(f, \Pi_2) \leq (sd) \cdot N^{-\Omega(d)} \leq s^2 \cdot N^{-\Omega(d)}$.

Case 2: there is no gate of degree $d_0 \in [\frac{3}{4}d, \frac{11}{12}d]$ in the template T. In this case, we show that $\text{rel-rank}(f, \Pi_1)$ is small, where Π_1 is as defined above.

Let v be the node in T such that $\deg(v) > \frac{11}{12}d$ and all its children have degree $\leq \frac{11}{12}d$. Note that such a v is uniquely defined (if there were another such node v', it cannot be an ancestor or descendant of v; hence, we see that the number of leaves in T is at least $\deg(v) + \deg(v') > d$ which is a contradiction). Let $d'_0 = \deg(v)$. Let $v_1, \ldots, v_t$ be the children of v in T and assume that $\deg(v_i) = d'_i$. Note that $d'_i < \frac{3d}{4}$ for each i.

As in the previous case, we see that every parse formula contains a gate of degree d'_0 and hence applying the lemma with $d' = d'_0$ we get

$$f = \sum_{i=1}^{\ell} \sum_{j=0}^{d-d'_0} g'_i \times_j h'_{i,j} \tag{3.5}$$

where $g'_1, \ldots, g'_\ell$ ($\ell \leq s$) are the polynomials of degree d'_0 computed by multiplication gates in C. We show that for each i, j, $\text{rel-rank}(g'_i \times_j h'_{i,j}, \Pi_1) \leq N^{-\Omega(d)}$. As in the previous case, this will imply $\text{rel-rank}(f, \Pi) \leq s^2 \cdot N^{-\Omega(d)}$.

Fix any i, j. We use g and h instead of g'_i and $h'_{i,j}$. We know that g is a polynomial computed by some $\times$ gate Φ of degree d'_0 in the circuit. Consider the $+$ gates feeding into Φ. Since every parse tree T' of C is a rotation of T, it must be the case that there are exactly t such $+$ gates $\Psi_1, \ldots, \Psi_t$ computing polynomials $\tilde{g}_1, \ldots, \tilde{g}_t$ such that $g = \tilde{g}_1 \cdots \tilde{g}_t$. Assume that $\deg(\tilde{g}_a) = d''_a$ for each $a \in [t]$. Then $\{d''_1, \ldots, d''_t\} = \{d'_1, \ldots, d'_t\}$ as multisets, where $d'_a = \deg(v_a)$ as defined above; in particular, $d''_a < \frac{3}{4}d$ for each a.

Thus, $g \times_j h = (\tilde{g}_1 \cdots \tilde{g}_t) \times_j h$. For any $a \in [t]$, we note that we can also write $g \times_j h = (\tilde{g}_1 \cdots \tilde{g}_a) \times_j h_a$ where $h_a := (\tilde{g}_{a+1} \cdots \tilde{g}_t) \times_j h$. Let $\Pi''_a = (Y''_a, Z''_a)$ be the partition of $[d]$ such that $Y''_a = [j+1, j+d''_1+\cdots+d''_a]$. By Corollary 1.14, we know that $\text{rank}(g \times_j h, \Pi''_a) = \text{rank}((\tilde{g}_1 \cdots \tilde{g}_a) \times_j h_a, \Pi''_a) \leq 1$ for *each* $a \in [t]$. Therefore, by Lemma 1.16, to prove that $\text{rel-rank}(g \times_j h, \Pi_1) \leq N^{-\Omega(d)}$, it suffices to show that $\Delta(\Pi''_a, \Pi_1) \leq \frac{d}{2} - \Omega(d)$ for *some* $a \in [t]$. We do this now.

Consider the least $b \in [t]$ so that $d''_1+d''_2+\cdots+d''_b \geq \frac{1}{20}d$. Let $\delta = d''_1+\cdots+d''_b$. Since each $d''_a < \frac{3}{4}d$, we know that $\delta \in [\frac{1}{20}d, d - \frac{1}{5}d]$. Note that for partition Π''_b, we have $\Delta(\Pi''_b, \Pi_1) = |Y''_b \Delta Y_1|$ where $Y''_b = [j+1, j+\delta]$ and $Y_1 = [\frac{1}{2}d]$. We thus get

$$\begin{aligned} |Y''_b \Delta Y_1| &\leq j + |\frac{d}{2} - (j+\delta)| = j + \max\{\frac{d}{2} - (j+\delta), j + \delta - \frac{d}{2}\} \\ &= \max\{\frac{d}{2} - \delta, 2j + \delta - \frac{d}{2}\} = \frac{d}{2} - \min\{\delta, d - (2j+\delta)\}. \end{aligned}$$

Since $\delta \geq \frac{1}{20}d$ and $(2j+\delta) \leq 2 \cdot \frac{1}{12}d + d - \frac{1}{5}d < d - \Omega(d)$, we see that $\Delta(\Pi''_b, \Pi_1) = |Y''_b \Delta Y_1| = \frac{d}{2} - \Omega(d)$. This completes the proof. □

Remark 3.10

We note that the proof of the theorem yields the stronger statement that f is low-rank w.r.t. one of the two partitions Π_1 and Π_2. It is not hard to use this to prove a lower bound for an even simpler polynomial than the polynomial $F_{N,d}$ from Theorem 1.17, by taking for example $g(X) = \alpha Pal^d(X) + \beta Pal^{d/2}(X) \times Pal^{d/2}(X)$ for suitable $\alpha, \beta \in \mathbb{F}$ to ensure that

$$\mathrm{rank}(g, \Pi_1) = \mathrm{rank}(Pal^d(X), \Pi_1) = N^{d/2}$$

and

$$\mathrm{rank}(g, \Pi_2) = \mathrm{rank}(Pal^{d/2}(X) \times Pal^{d/2}(X), \Pi_2) = N^{d/2}.$$

Note also that $g(X)$ is computable by a small 2-PT circuit, hence 2-PT $\not\subset$ rot-PT.

Chapter 4

Towards a separation between Formulas and ABPs

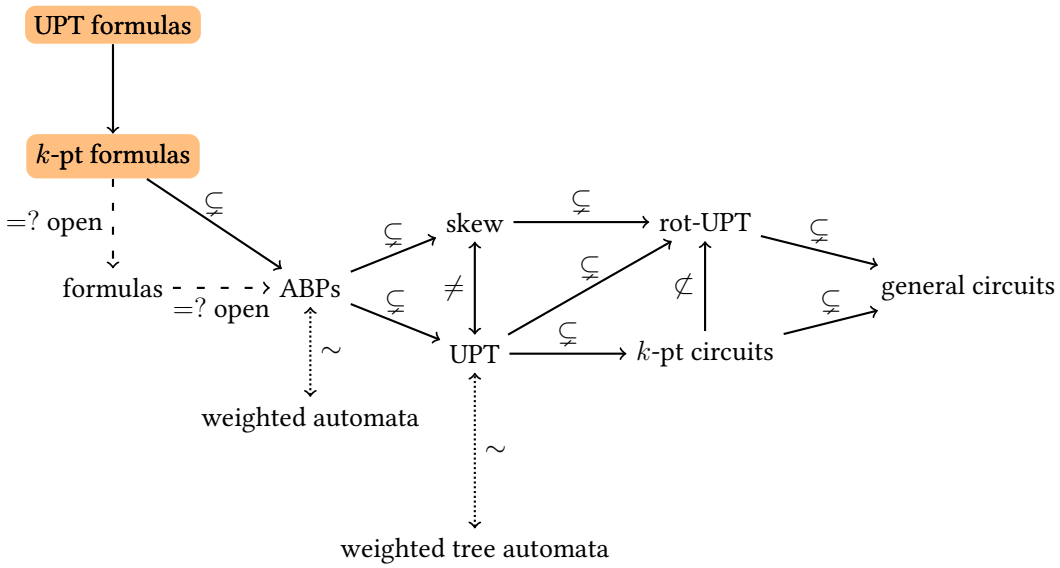


Figure 4.1: You are here.

Overview

In the previous chapters, we make some progress in proving lower bounds for circuits closer and closer to general circuits by restricting the parse trees in the circuits. In this chapter, we make progress on the "Formulas vs. ABPs" question, using the same approach: we use restrictions on the parse trees that appear in formulas. More precisely, we prove some tight lower bounds for formulas (with

some restrictions on the parse trees) computing $\mathrm{IMM}_{n,d}$, yielding separations from ABPs. More specifically, we prove in Section 4.2 a tight superpolynomial lower bound on the size of any UPT formula that computes $\mathrm{IMM}_{n,d}$. In Section 4.3, we prove a superpolynomial lower bound for any formula computing $\mathrm{IMM}_{n,d}$ as long as the number of distinct parse trees is significantly smaller than 2^d (assuming $d \leq \log n$).
This chapter is based on the following publication:

- Guillaume Lagarde, Nutan Limaye, and Srikanth Srinivasan. Lower bounds and PIT for non-commutative arithmetic circuits with restricted parse trees. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, pages 41:1–41:14, 2017

- Guillaume Lagarde, Nutan Limaye, and Srikanth Srinivasan. Lower bounds and PIT for non-commutative arithmetic circuits with restricted parse trees (extended version). *To appear in Computational Complexity*

4.1 Notation and decomposition lemma for labelled UPT formulas

In this section, we fix some notation that will be used in the rest of Chapter 4. We also prove that any polynomial computed by a *labelled* UPT formula (defined below) admits a specific decomposition, given by Lemma 4.5.

Throughout the chapter, the set X will be a set of $N = n^2 d$ variables satisfying $X = \bigcup_{i \in [d]} X_i$ where each X_i has n^2 variables. Let M_i be an $n \times n$ matrix whose entries are distinct variables from X_i. Recall from Section 1.1 that $\mathrm{IMM}_{n,d} = \mathrm{Tr}(M_1 \cdots M_d) \in \mathbb{F}\langle X \rangle$.

Algebraic Branching Programs

We also consider homogeneous algebraic branching programs (ABPs), defined by Nisan in the non-commutative context. We give here a slightly different definition that is equivalent up to polynomial factors.

A homogeneous *algebraic branching program* (ABP) for a homogenous polynomial $f \in \mathbb{F}\langle X \rangle$ of degree d is a pair (n_1, ρ) where $n_1 \in \mathbb{N}$ and ρ is a map from $X' = \{x'_1, \ldots, x'_{n_1^2 d}\}$ to homogeneous linear functions from $\mathbb{F}\langle X \rangle$ such that f can be obtained by substituting $\rho(x'_i)$ for each x'_i in the polynomial $\mathrm{IMM}_{n_1,d}(X')$. The parameter n_1 is called the *width* of the ABP.

Other notation and first properties

For $I = \{i_1 < \cdots < i_t\} \subseteq [d]$, we define the set of *$I$-monomials* to be the set of monomials of the form $x_1 \cdots x_t$ where $x_j \in X_{i_j}$. Also, we define $\mathcal{P}_I$ to be the set of those polynomials P over the variables $\bigcup_{j\in[t]} X_{i_j}$ that can be written as a linear combination of I-monomials. We define IMM_I to be $\mathrm{Tr}(M_{i_1} \cdots M_{i_t})$. Note that $\mathrm{IMM}_I \in \mathcal{P}_I$.

Given I as above and $f \in \mathcal{P}_I$, we define $M_I(f)$ to be a $n^{2\lceil \frac{t}{2} \rceil} \times n^{2\lfloor \frac{t}{2} \rfloor}$ matrix whose rows and columns are labelled by I_{odd}-monomials and I_{even}-monomials respectively, where $I_{\text{odd}} = \{i_1, i_3, i_5, \ldots\}$ and $I_{\text{even}} = \{i_2, i_4, i_6, \ldots\}$. The (m', m'')th entry of $M_I(f)$ is the coefficient in f of the I-monomial which is equal to m' when restricted to its odd locations and m'' when restricted to its even locations. Note that $\mathrm{rank}(M_I(f)) \leq n^{2\lfloor \frac{t}{2} \rfloor}$. We define $\text{rel-rank}_I(f) = \mathrm{rank}(M_I(f))/n^{2\lfloor \frac{t}{2} \rfloor}$.

We will need the following simple observations.

Observation 4.1

For any $I \subseteq [d]$ of size t and any $f \in \mathcal{P}_I$, we have $\mathrm{rank}(M_I(f)) \leq n^{2\lfloor t/2 \rfloor}$. For any $I \subseteq [d]$ of size t, we have $\mathrm{rank}(M_I(\mathrm{IMM}_I)) = n^{2\lfloor t/2 \rfloor}$, and hence $\text{rel-rank}_I(\mathrm{IMM}_I) = 1$.

Let T be a parse tree with t leaves and $I = \{i_1 < i_2 < \cdots < i_t\}$. The *$I$-labelling of T* is the function $\mathrm{lab} : T \to 2^I \setminus \{\emptyset\}$ defined as follows. For each $u \in \mathcal{L}(T)$ (recall that $\mathcal{L}(T)$ is the set of leaves of T and $\mathcal{I}(T)$ is the set of internal nodes of T), we define $\mathrm{lab}(u)$ to be $\{i_j\}$ if u is the jth leaf in the pre-order traversal of T. We will sometimes abuse notation and write $\mathrm{lab}(u) = i_j$. For each $v \in \mathcal{I}(T)$, we define $\mathrm{lab}(v)$ to be the set of labels of the leaves in the subtree rooted at v.

Let F be a UPT formula of shape T. By Proposition 2.3, we know that F is in normal form. We say that F is *I-labelled* if for each input gate Φ of F that is a $(u, \times)$-gate with $u \in \mathcal{L}(T)$, the variable labelling Φ lies in the set $X_{\mathrm{lab}(u)}$. The following is an easy observation.

Observation 4.2

If F is an I-labelled UPT formula with shape T, then it computes a polynomial from $\mathcal{P}_I$. More generally, if Φ is a $(u, +)$ or $(u, \times)$ gate of F with $u \in T$, then Φ computes a polynomial from $\mathcal{P}_{\mathrm{lab}(u)}$.

Further, for any F that is a UPT formula of size at most s computing a polynomial f, there is an I-labelled UPT formula F' of shape T and size at most s that computes the polynomial $f' \in \mathcal{P}_I$ that is obtained from f by zeroing out the coefficients of all monomials that are not I-monomials.

In the lower bound proofs, we follow a strategy used by Nisan and Wigderson [39], and show that any small formula (with suitably restricted parse trees) can be converted to a low-rank polynomial after setting a subset of the variables. To make this precise, we need the notion of a *restriction*, which we define now.

A *restriction* is formally just a subset $I \subseteq [d]$, which represents a substitution ρ_I of the set of variables in $X = \bigcup_{i \in [d]} X_i$ as follows:

$$\rho_I(x) = \begin{cases} x & \text{if } x \in \bigcup_{i \in I} X_i, \\ 0 & \text{if } x \text{ is an offdiagonal entry of } M_j \text{ for } j \notin I, \\ 1 & \text{if } x \text{ is a diagonal entry of } M_j \text{ for } j \notin I. \end{cases}$$

In other words, we substitute all the variables in $\bigcup_{j \notin I} X_j$ such that each M_j ($j \notin I$) becomes the identity matrix. All variables from the set $\bigcup_{i \in I} X_i$ are left as is.

Every polynomial $P \in \mathbb{F}\langle X \rangle$ is transformed in the natural way by such a substitution. We call this new polynomial a *restriction of* P and denote it by $P|_I$.

Let T be a parse tree with d leaves. For any restriction I, let $T|_I$ denote the tree obtained by removing all nodes $u \in T$ such that $\mathrm{lab}(u) \cap I = \emptyset$ (in particular only leaves with labels from I survive in $T|_I$). The I-labelling of the tree $T|_I$ is given by the labelling function lab_I where $\mathrm{lab}_I(u) = \mathrm{lab}(u) \cap I$.

We make the following simple observations.

Observation 4.3

1. If $P \in \mathcal{P}_{[d]}$, then $P|_I \in \mathcal{P}_I$.
2. $\mathrm{IMM}_{n,d}|_I = \mathrm{IMM}_I$.

We note that the above restrictions don't increase the complexity of the polynomial as far as UPT formulas are concerned.

Lemma 4.4

For any $[d]$-labelled UPT formula F of size s and shape T computing some f (note that $f \in \mathcal{P}_{[d]}$ by Observation 4.2), there is a UPT formula $F|_I$ of size at most s and shape $T|_I$ computing $f|_I$.

Proof. Let F be as in the statement and I be any restriction. If we replace every variable in the formula F by $\rho_I(x)$, we obtain by definition a formula F' which computes $f|_I$. F' is not a UPT formula since the leaves which were labeled by a variable from X_j ($j \notin I$) have been replaced by some constants, whereas leaves in a UPT formula have to be variables. We now transform F' to a UPT formula $F|_I$ in the following way. Let u, v be any pair of nodes in T with $\mathrm{lab}(u) \cap I = \emptyset$ and $\mathrm{lab}(v) \cap I \neq \emptyset$. Each $(u, +)$-gate Ψ in F' computes a constant (say α) and is wired to a $(v, \times)$-gate Φ in F'. We delete the subformula rooted at Ψ from F' and multiply by α the constant labelling the output wire of Φ. By doing this for every u, v and Ψ, the formula F' becomes a new formula $F|_I$ which is UPT with shape $T|_I$, and still computes $f|_I$. □

Decomposition lemma

We now state a decomposition lemma for UPT formulas which will help us choose suitable restrictions that help simplify small formulas to low-rank polynomials.

Let T be a parse tree and $\pi = (v_r, \ldots, v_0)$ a path of length r in it[1]. We say that u is an *off-path node* of π if there is an $i < r$ such that u is a child of v_i and $u \neq v_{i+1}$. The set of off-path nodes of π is denoted $\mathrm{off}(\pi)$.

[1]Recall that our trees are oriented towards the root.

Lemma 4.5

Let F be an I-labelled UPT formula of size s with shape T computing a polynomial $f \in \mathcal{P}_I$, and let $\pi = (v_r, \ldots, v_0)$ be a path in T. If we define u_j to be the jth node of $\mathrm{off}(\pi) \cup \{v_r\}$ that appears in the pre-order traversal of T, then we can decompose f as:

$$f = \sum_{i=1}^{k} \prod_{j=1}^{t} f_{i,j}$$

where

- $k \leq s$,
- $t = |\mathrm{off}(\pi) \cup \{v_r\}|$,
- $f_{i,j} \in \mathcal{P}_{\mathrm{lab}(u_j)}$.

Proof. Let F be an I-labelled UPT formula as in the statement and $\pi = (v_r, \ldots, v_0)$ be a path in T. Let $(u_1, \ldots, u_t)$ be the ordering of the set of nodes $(\mathrm{off}(\pi) \cup \{v_r\})$ given by a pre-order traversal of T. By Proposition 2.3, F is in normal form.

We say that a path in the formula F is of *signature* π if the $+$-gates along this path are successively a $(v_r, +)$ gate, a $(v_{r-1}, +)$ gate, and so on until we get a $(v_0, +)$ gate. Let k be the number of paths in F with signature π, and $p_1, p_2, ..., p_k$ be these paths. As F is a formula, the number of paths from a leaf to the root is upper bounded by s. Therefore $k \leq s$.

Each parse formula F' (which is a subformula of F) of F has shape T and further each $+$-gate in F' has fan-in 1; thus, each parse formula contains one and only one path of signature π. The set S of parse formulas of F is therefore naturally partitioned as $S = S_1 \cup \cdots \cup S_k$, where S_i is the set of parse formulas that contain the path p_i. Recall that if F' is a parse formula of F, we denote by $val(F')$ the monomial (along with its coefficient) computed by it. We have:

$$f = \sum_{F' \in S} val(F') = \sum_{i=1}^{k} \sum_{F' \in S_i} val(F')$$

From now, what remains to prove is that for each $a \in [k]$, $\sum_{F' \in S_a} val(F')$ is of the form $\prod_{j=1}^{t} f_{i,j}$ with the additional property that each $f_{i,j}$ is a polynomial in

$\mathcal{P}_{\mathrm{lab}(u_j)}$.

We fix a particular $a \in [k]$. The polynomial $\sum_{F' \in S_a} val(F')$ is nothing else than the polynomial computed by the I-labelled UPT formula G where for all $m \in \{0, \ldots, r\}$, each $(v_m, +)$ gate that is not present on the path p_a has been deleted (together with the entire subformula at that gate). Observe that for all m, the $(v_m, +)$ gate in G is of in-degree and out-degree 1, except for the output gate, which is a $(v_0, +)$ gate of in-degree 1 and is of out-degree 0.

Let $p_a = \Phi_r, \Psi_{r-1}, \Phi_{r-1}, \Psi_{r-2}, \Phi_{r-2}, \ldots, \Phi_1, \Psi_1, \Phi_0$, where Φ_0 is the root and for each m, Φ_m is the $(v_m, +)$-gate and Ψ_m is the $(v_m, \times)$-gate in p_a. By construction, the path p_a is present in G. We prove by induction that the following statement $H(m)$ holds for each $m \leq r$,

$H(m)$: If we denote by $(w_1, \ldots, w_{t_m})$ the ordering of the set $\mathrm{off}((v_r, \ldots, v_m)) \cup \{v_r\}$ given by a pre-order traversal of T, then the polynomial computed by Φ_m in G is of the form $\prod_{j=1}^{t_m} g_j$ where:

- $t_m = |\mathrm{off}((v_r, \ldots, v_m)) \cup \{v_r\}|$
- $g_j \in \mathcal{P}_{\mathrm{lab}(w_j)}$

We now prove $H(m)$ by downward induction on m. It is clearly true when $m = r$ since $t_r = 1$, as F is an I-labelled UPT formula and hence the polynomial computed by Φ_r is an element of $\mathcal{P}_{\mathrm{lab}(v_r)}$.

Assume the statement $H(m+1)$ and let $\prod_{j \in [t_{m+1}]} h_j$ be the decomposition of the polynomial computed by the gate Φ_{m+1} given by the induction hypothesis. The polynomial computed by Φ_m (a $+$-gate of fan-in 1 in the formula G) is the product of the inputs of Ψ_m: assume that these inputs are (in left-to-right order) $\Phi'_1, \ldots, \Phi'_b$ with each Φ'_ℓ being a $(w''_\ell, +)$-gate for some $w''_\ell \in T$. Let P_ℓ be the polynomial computed by Φ'_ℓ ($\ell \in [b]$). The gate Φ_{m+1} is one among $\Phi'_1, \ldots, \Phi'_b$: let us say it is Φ'_c. The polynomial computed by Φ_m is equal to $\prod_{\ell=1}^{b} P_\ell = (\prod_{\ell=1}^{c-1} P_\ell).(\prod_{j=1}^{t_{m+1}} h_j).(\prod_{\ell=c+1}^{b} P_\ell)$ by the induction hypothesis. Each P_ℓ is computed by a $(w''_\ell, +)$-gate and is thus a polynomial in $\mathcal{P}_{lab(w''_\ell)}$ (Observation 4.2), and by induction, the h_j are polynomials in $\mathcal{P}_{\mathrm{lab}(w'_j)}$, where $w'_1, \ldots, w'_{t_{m+1}}$ is the ordering of $\mathrm{off}((v_r, \ldots, v_{m+1})) \cup \{v_r\}$ given by the pre-order traversal of T. But observe that $w''_1, \ldots, w''_{c-1}, w'_1, \ldots, w'_{t_{m+1}}, w''_{c+1}, \ldots, w''_{t_m}$ is exactly the ordering of $\mathrm{off}((v_r, \ldots, v_m)) \cup \{v_r\}$ given by the pre-order traversal of T, so that the induction holds, and the lemma is proved. □

Proof outline of the two lower bounds contained in this chapter. The basic proof strategy is to choose a suitable $I \subseteq [d]$ and apply this restriction to the polynomial $\mathrm{IMM}_{n,d}$ as well as the formula computing it. By Observation 4.3, under any such restriction, $\mathrm{IMM}_{n,d}$ becomes the polynomial IMM_I which, by Observation 4.1, has reasonably large relative rank. On the other hand, for any given small formula (with some restriction on the parse trees), we show that there is a suitable choice for the restriction that makes its relative rank quite small. This will prove the lower bound.

The choice of the restriction to make a small formula low rank is dictated by Lemma 4.5. Consider first the case of a UPT formula F of shape T which has depth Δ. In this case, we show that we can find a path $\pi = (v_r, \ldots, v_0)$ in T with t many off-path nodes where t is roughly $\Delta d^{1/\Delta}$. Applying Lemma 4.5, we can find a decomposition of F of the form

$$\sum_{i=1}^{s} \prod_{j \in [t]} f_{i,j}$$

where s is the size of the formula F; it is crucial here that the sets of variables among $X_1, \ldots, X_d$ that $f_{i,j}$ depends upon is exactly the set of labels of the jth off-path node of π and in particular, is the same no matter which i we choose (this is where the formula being UPT is used). Say that $I_j \subseteq [d]$ indexes the variable sets that $f_{i,j}$ depends on. To make sure that these terms are low-rank, we use the following observation (used also by [39]): for each j such that $|I_j|$ is odd, the relative rank of the term $\prod_j f_{i,j}$ drops by a factor of $1/n$. Thus, if we restrict to a subset I such that $|I \cap I_j| = 1$ for each j, the relative rank of each term is at most n^{-t} and hence we obtain a lower bound of n^t on the size s of the formula. This gives a lower bound of roughly $n^{\Omega(\Delta d^{1/\Delta})}$ on the size of F. With no restriction on Δ, this gives a lower bound of $n^{\Omega(\log d)}$.

Now consider the case that F is a k-PT formula for $k \ll 2^d$. In this case, we first write F as a sum of k UPT formulas $F_1, \ldots, F_k$ (Lemma 3.1) and find a restriction I such that *each* F_i becomes low-rank with respect to I. To do this, we note that, by our proof in the UPT case, for some F_j to be low-rank w.r.t. I, it suffices to show that after applying the restriction we can find *some* path $\pi = (v_r, \ldots, v_0)$ in the corresponding shape such that there are many off-path nodes in π whose labels involve an odd number of variable sets among $X_1, \ldots, X_d$.

The way we do this is that we choose a restriction $I \subseteq [d]$ uniformly at random and show that for any given shape T, the probability that there is no π as above in the restricted parse tree $T|_I$ is very small. With this technical statement in hand, the proof of the lower bound follows as in the UPT case.

For the rest of this chapter and in order to avoid useless technical details, the

size of a circuit will be its number of vertices.

4.2 Lower bound for UPT formulas

We define the $\times$-*depth* of a formula to be the maximum number of $\times$-gates that one can meet on a path from the root to a leaf. Note that if a formula has alternating $+$ and $\times$ gates on each path and has depth Δ' and $\times$-depth Δ, then $\Delta \leq \lceil \frac{\Delta'}{2} \rceil$. We will state our lower bounds bounds in terms of $\times$-depth.

Throughout this section, we assume that all the UPT formulas we consider don't have any multiplication gate of fan-in 1, or equivalently, the shape of any UPT formula we consider does not have any internal node of fan-in 1. This assumption is w.l.o.g. as shown below.

Lemma 4.6

Given any UPT formula F of shape T and size s computing a polynomial f, there is another UPT formula F' of shape T' and size at most s computing f where T' has no internal nodes of fan-in 1 (and consequently F' has no $\times$-gates of fan-in 1). Further, if all internal nodes of T have fan-in at most $k \in \mathbb{N}$ with $k \geq 2$, then the same holds for T'.

Proof. The transformation process is the following: a multiplication gate of fan-in 1 does not change its input and therefore can be deleted without changing the polynomial computed. Merging the two layers of $+$-gates above and below the deleted gate ensures the formula still alternates between $+$-gates and $\times$-gates. The shape T' of the new formula is simply the shape T where the internal nodes of fan-in 1 have been removed and replaced by an edge. Clearly, the new shape T' has the required property. □

Before attacking the main theorem, we will need one more lemma.

Lemma 4.7

Let T be a tree with d leaves and depth Δ, such that all internal nodes are of in-degree strictly greater than 1. Then there is a path $\pi = (v_\ell, \ldots, v_0)$ in T such that $|\mathrm{off}(\pi) \cup \{v_\ell\}| \geq \Omega(\Delta d^{1/\Delta})$.

Proof. Let T be a tree as in the statement. We denote by $wt(v)$ the fan-in of the node v. We will prove the following equivalent conclusion: there is a path $\pi = (v_\ell, \ldots, v_0)$ from a leaf to the root such that $1 + \sum_{0 \leq i < \ell} (wt(v_i) - 1) \geq \Omega(\Delta d^{1/\Delta})$.

We consider two distinct cases:

- **Case 1:** $\Delta > \log_e(d)$. In this case, any path $p = (v_\Delta, \ldots, v_0)$ of depth Δ satisfies $1 + \sum_{0 \le i < \Delta} (wt(v_i) - 1) \ge \Delta + 1 = \Omega(\Delta d^{1/\Delta})$.

- **Case 2:** $\Delta \le \log_e(d)$. We consider the following greedy procedure to choose the path of internal nodes: starting from the root, repeatedly choose a child such that the number of leaves in the resulting subtree is maximized. Let $p = v_0, \ldots, v_\ell$ be the sequence of nodes thus obtained. Note that $\ell \le \Delta \le \log_e(d)$.

  We prove by induction on the tree T that the number of leaves of the tree is at most the product of the fan-ins of $v_0, \ldots, v_{\ell-1}$ (this fact is true for any tree T and not just trees T of depth at most $\log_e d$). If $\ell = 0$, the entire tree consists of just the root: hence the number of leaves is 1 and the (empty) product also evaluates to 1. Assume now that the root v_0 has k children corresponding to subtrees $T_1, T_2, \ldots T_k$. We assume the number of leaves in the subtree T_i is t_i. Assume the greedy algorithm chooses v_1 corresponding to the subtree rooted in T_i (thus, we must have $t_i \ge t_j$ for any $j \in [k]$). By the induction hypothesis, $\prod_{j=1}^{\ell} wt(v_j) \ge t_i$. Therefore

  $$\prod_{j=0}^{\ell-1} wt(v_j) \ge k \cdot t_i \ge \sum_{j=1}^{k} t_j = d$$

  which concludes the induction.

  By the inequality of arithmetic and geometric means, we have

  $$\frac{\sum_{0 \le i < l} wt(v)}{\ell} \ge \left(\prod_{0 \le i < \ell} wt(v) \right)^{1/\ell} \ge d^{1/\ell}.$$

  So, we have

  $$\sum_{0 \le i < \ell} (wt(v) - 1) \ge \ell(d^{1/\ell} - 1).$$

  Notice that the right part of this inequality is a decreasing function of ℓ in the regime $\ell \le \log_e(d)$, so that:

  $$\begin{aligned} \sum_{0 \le i < \ell} (wt(v) - 1) \ge \Delta(d^{1/\Delta} - 1) &= \Delta d^{1/\Delta}(1 - \frac{1}{d^{1/\Delta}}) \\ &\ge \Delta d^{1/\Delta}(1 - \frac{1}{e}) \\ &= \Omega(\Delta d^{1/\Delta}). \end{aligned}$$

□

We now prove a lower bound for any UPT formula computing $\mathrm{IMM}_{n,d}$. The lower bound is depth-dependent and stronger for smaller depths.

Theorem 4.8

Let F be a UPT formula of $\times$-depth Δ, size s, computing $IMM_{n,d} \in \mathbb{F}\langle X\rangle$. Then, $s \geq n^{\Omega(\Delta d^{1/\Delta})}$. In particular, any UPT formula for $\mathrm{IMM}_{n,d}$ must have size $n^{\Omega(\log d)}$.

By our earlier observation relating the $\times$-depth with depth, we get the lower bound stated in the introduction.

Proof. Let F be a UPT formula as in the statement. Let T be the shape of F: note that the depth of T is precisely the $\times$-depth of F which is Δ.

By Lemma 4.2, we can assume w.l.o.g that the formula F is $[d]$-labelled and hence that the variables that label any input gate Φ of F corresponding to a node $v(\Phi) \in T$ are all included in $X_{\mathrm{lab}(v(\Phi))}$ where lab is the $[d]$-labelling of T. By Lemma4.7, there is a path $\pi = (v_\ell, \ldots, v_0)$ in the shape T of F, such that $|\mathrm{off}(\pi) \cup \{v_0\}| \geq \Omega(\Delta d^{1/\Delta})$. Let us denote by t the size of $\mathrm{off}(\pi) \cup \{v_0\}$. We decompose $\mathrm{IMM}_{n,d}$ along this path by Lemma 4.5, as:

$$\mathrm{IMM}_{n,d}(X_1, X_2, ..., X_d) = \sum_{i=1}^{k} \prod_{j=1}^{t} f_{i,j}$$

with $k \leq s$. Each $f_{i,j}$ is a $\mathcal{P}_{\mathrm{lab}(u_j)}$ where $(u_1, \ldots, u_t)$ is the ordering of $\mathrm{off}(\pi) \cup \{v_0\}$ given by a pre-order traversal of T.

We now apply a restriction to this equality given by the subset I choosen in the following way: for each j, we select one element from $\mathrm{lab}(u_j)$ — we call it p_j — and add it to I. The set I is of size t. Under this restriction, each $f_{i,j}$ becomes a homogeneous linear polynomial in the variables X_{p_j}. We call these homogeneous linear polynomials $l_{i,j}$. On the other hand, we know (Observation 4.3) that $\mathrm{IMM}_{n,d}|_I = \mathrm{IMM}_I$. We thus get

$$\mathrm{IMM}_I = \sum_{i=1}^{k} \prod_{j=1}^{t} l_{i,j}.$$

It is not hard to see that for each i, $\mathrm{rank}(M_I(\prod_{j=1}^{t} l_{i,j})) \leq 1$. By Observation 4.1 and subadditivity of the rank, we get:

$$n^{2\lfloor t/2 \rfloor - 1} \leq k$$

Therefore, we get

$$s \geq k \geq n^{\Omega(\Delta t^{1/\Delta})}$$

as wanted. □

Remark 4.9

Notice that this lower bound is tight for every $\times$-depth Δ, since the standard divide and conquer approach to computing $\mathrm{IMM}_{n,d}$ gives in fact a UPT formula of size $n^{O(\Delta d^{1/\Delta})}$ and $\times$-depth Δ, for any $\Delta \leq \log d$.

4.3 Separation between k-PT formulas and ABPs

In this section, we will prove a lower bound on the size of k-PT formulas computing $\mathrm{IMM}_{n,d}$ as long as k is significantly smaller than 2^d. Recall that the total number of parse trees with d leaves is $2^{O(d)}$ (see for example [11]) and hence the results of this section may be intuitively interpreted as saying that any non-trivial upper bound on the number of parse trees appearing in the formula gives a separation between non-commutative formulas and ABPs.

The main theorem of this section is the following.

Theorem 4.10

Let n, d be growing parameters with $d \leq \log n$. Then, any k-PT formula F computing $\mathrm{IMM}_{n,d}$ has size at least n^{ℓ} where $\ell = \Omega(\log d - \log\log k)$. In particular, if $k = 2^{o(d)}$, then $\mathrm{size}(F) \geq n^{\omega(1)}$ and if $k = 2^{d^{1-\Omega(1)}}$, then $\mathrm{size}(F) \geq n^{\Omega(\log d)}$.

Remark 4.11

We say a few words about the assumption $d \leq \log n$. The standard divide-and-conquer approach computing $\mathrm{IMM}_{n,d}$ yields a (UPT) formula of size $n^{O(\log d)}$. It would be quite surprising if this standard algorithm were not optimal in terms of formula size (even for non-UPT formulas).

Intuitively, improving on the standard divide-and-conquer algorithm gets harder as d gets smaller: this is because any formula of size $n^{o(\log d)}$ for computing $\mathrm{IMM}_{n,d}$ can be straightforwardly used to recursively obtain formulas for $\mathrm{IMM}_{n,D}$ of size $n^{o(\log D)}$ for any $D > d$. Thus, the case of smaller d, which seems harder algorithmically, is a natural first candidate for lower bounds.

Let T be a parse tree. We say that a node $u \in T$ is *odd* if the number of leaves in the subtree rooted at u is odd. Given a path π, let $\mathrm{odd}(\pi)$ denote the set of odd off-path nodes of π.

Lemma 4.12

Let F be an I-labelled UPT formula of size s with shape T computing polynomial f. If T has a path $\pi = (v_r, \ldots, v_0)$ with $|\mathrm{odd}(\pi)| \geq \ell$, then

$$\text{rel-rank}_I(f) \leq \frac{s}{n^{\ell-1}}.$$

Proof. Let $(u_1, \ldots, u_t)$ be the ordering of the set of nodes $(\mathrm{off}(\pi) \cup \{v_0\})$ given by a pre-order traversal of T, and $f = \sum_{i=1}^{k} \prod_{j=1}^{t} f_{i,j}$ be a decomposition given by Lemma 4.5, where each $f_{i,j}$ is in $\mathcal{P}_{\mathrm{lab}(u_j)}$. By Observation 4.1, we know that $\mathrm{rank}(M_I(f_{i,j})) \leq n^{2\lfloor |\mathrm{lab}(u_j)|/2 \rfloor}$. Hence, by the subadditivity of rank and

Lemma 1.13, we have:

$$\begin{aligned}\operatorname{rank}(M_I(f)) &\leq \sum_{i=1}^{k}\prod_{j=1}^{t}\operatorname{rank}(M_I(f_{i,j}))\\ &\leq \sum_{i=1}^{k}\prod_{j=1}^{t} n^{2\lfloor\frac{|\mathrm{lab}(u_j)|}{2}\rfloor}\\ &= \sum_{i=1}^{k} n^{2\sum_{j=1}^{t}\lfloor\frac{|\mathrm{lab}(u_j)|}{2}\rfloor}\\ &= \sum_{i=1}^{k} n^{2(\lfloor\frac{|I|}{2}\rfloor-\lfloor\frac{|\mathrm{odd}(\pi)|}{2}\rfloor)}\end{aligned}$$

As $k \leq s$, this implies that $\text{rel-rank}_I(f) \leq \frac{s}{n^{2\lfloor\frac{|\mathrm{odd}(\pi)|}{2}\rfloor}} \leq \frac{s}{n^{\ell-1}}$. □

We now try to show that, given a small k-PT formula, there is a suitable choice of the restriction that makes its relative rank quite small. To do this, we will use Lemma 4.12, which translates the statement to a combinatorial statement about some trees. On the other hand, IMM remains high rank under arbitrary restrictions by Observation 4.1. This will prove Theorem 4.10.

The main technical lemma in the proof of Theorem 4.10 is the following.

Lemma 4.13

Let T be any tree with d leaves such that every internal node has fan-in exactly 2. Assume we choose $I \subseteq [d]$ by adding each $i \in [d]$ to I independently with probability $1/2$. Then for any $\ell \in \mathbb{N}$

$$\Pr_I[T|_I \text{ has no path } \pi \text{ such that } |\mathrm{odd}(\pi)| \geq \ell] \leq \exp(-\Omega(\frac{d}{2^{8\ell}})).$$

Postponing the proof of the above lemma to the end of this section, we can prove Theorem 4.10 as follows.

Proof of Theorem 4.10. We assume throughout that $\log d - \log\log k$ is larger than a large enough constant (to be chosen later), since otherwise the theorem is trivial. Let $\ell = \lfloor\frac{1}{10}(\log d - \log\log k)\rfloor$.

Assume that F is a k-PT formula of size s computing $\mathrm{IMM}_{n,d}$. If $s \geq n^{\ell/4}$, then we are done.

Otherwise, we argue as follows. By Lemma 3.1, there exist UPT formulas $F_1, \ldots, F_k$ of size at most s^2 each such that

$$\mathrm{IMM}_{n,d} = \sum_{i=1}^{k} f_i$$

where f_i is the polynomial computed by F_i. Let T_i denote the shape of F_i. By Lemma 2.1, we can assume that each internal node of T_i has fan-in exactly 2.

By Observation 4.2 and Lemma 4.6, for each F_i, there is a $[d]$-labelled UPT formula F_i' of shape T_i and size at most s^2 that computes the polynomial f_i' that is obtained from f_i by removing monomials that are not $[d]$-monomials. Since $\mathrm{IMM}_{n,d} \in \mathcal{P}_{[d]}$, we see that

$$\mathrm{IMM}_{n,d} = \sum_{i=1}^{k} f_i'. \tag{4.1}$$

Now, choose a random restriction I by adding each $j \in [d]$ to I independently with probability $1/2$. Consider the relative rank of the polynomials on both sides of (4.1) after the restriction. For the left hand side, we know using Observation 4.1 that for any I,

$$\text{rel-rank}_I(\mathrm{IMM}_{n,d}|_I) = \text{rel-rank}_I(\mathrm{IMM}_I) = 1. \tag{4.2}$$

We now consider the right hand side of (4.1). By Lemma 4.4, for any choice of restriction I and $i \in [k]$, the restricted polynomial $f_i'|_I$ has a UPT formula $F_i'|_I$ of size at most s^2 and shape $T_i|_I$ computing $f_i'|_I$. For each $i \in [k]$, let $\mathcal{E}_i$ denote the event that $T_i|_I$ has no path π such that $|\mathrm{odd}(\pi)| \geq \ell$. By Lemma 4.13, we know that the probability of $\mathcal{E}_i$ is at most $\exp(-\Omega(\frac{d}{2^{8\ell}}))$. Let $\mathcal{E} = \bigvee_{i=1}^{k} \mathcal{E}_i$. By a union bound we have

$$\Pr[\mathcal{E}] \leq k \cdot \exp\left(-\Omega\left(\frac{d}{2^{8\ell}}\right)\right) < 1 \tag{4.3}$$

if $(\log d - \log\log k)$ is larger than some fixed constant. If I is such that the event $\mathcal{E}$ does not occur, then for this choice of I and any $i \in [k]$, by Lemma 4.12, $\text{rel-rank}_I(f_i'|_I) \leq \frac{s^2}{n^{\ell-1}} \leq \frac{1}{n^{(\ell/2)-1}}$, where the final inequality follows from our assumption that $s < n^{\ell/4}$. Now, since $\text{rel-rank}_I(\cdot)$ is subadditive, we have

$$\text{rel-rank}_I(\sum_{i\in[k]} f_i') \leq \frac{k}{n^{(\ell/2)-1}} \leq \frac{2^d}{n^{(\ell/2)-1}} \leq \frac{1}{n^{(\ell/2)-2}} < 1$$

where the final two inequalities follow from the fact that $d \leq \log n$ and the assumption that ℓ is greater than some fixed constant. This contradicts (4.1) and (4.2) and hence concludes the proof of the theorem. □

Proof of Lemma 4.13

We impose a natural partial order on the vertices in T by saying that $u \preceq v$ for $u, v \in T$ if v is an ancestor of u. Given a set of paths $P = \{\pi_1, \ldots, \pi_r\}$ in the tree T, we say that P is *independent* if the sets $\mathrm{off}(\pi_i)$ ($i \in [r]$) are pairwise disjoint and moreover, the set $\mathrm{off}(P) := \bigcup_i \mathrm{off}(\pi_i)$ forms an antichain w.r.t. the partial order $\preceq$ (informally, no node in $\mathrm{off}(P)$ is an ancestor of another).

We show the following claim.

Claim 4.14

Assume T is as in the statement of the lemma. Then for any $\ell \geq 1$, there is an independent set P of paths in T of length ℓ such that $|P| = \Omega(d/2^{2\ell})$.

Proof. Given a tree T as in Lemma 4.13, let us define T' to be the subtree of T which contains every node of T that has height ℓ or more (here the height of a node u is the length of the longest path from a leaf from $\mathcal{L}(u)$ to u). Though every internal node in T has degree two, T' may have internal nodes of degree two as well as one. The leaves of T' are those internal nodes of T that have height exactly ℓ.

The main idea is as follows: Suppose T' has 'many' leaves, then it is easy to find many independent paths in T. This is because each leaf v of T' is a node in T with at least one path of length ℓ rooted at v. This gives us as many independent paths as the number of leaves in T'. On the other hand, if T' does not have many leaves, then it also does not have many fan-in two nodes. In this case, by throwing away all fan-in two nodes of T', we get many components. Each component is a path and not all can be of length less than ℓ. Subdividing the long paths into paths of length ℓ then gives us the set of independent paths[2]. We now work out the details.

As every internal node of T has degree two, the number of nodes at height h, for any parameter $h \geq 1$, is at least half of the number of nodes at height $h-1$. It follows that the number of leaves in T' and therefore $|T'|$ is $\geq \frac{d}{2^\ell}$. We use s to denote $|T'|$.

Case I, the number of leaves in T' is $\geq s/100\ell$: Each leaf v in T' has a subtree rooted at it in T, say T_v. For each leaf v in T', T_v has at least one path of length ℓ from v to a leaf of T. Let us call this path π_v. As the leaves of T' are all the nodes at height ℓ, for two leaves of T', say $u \neq v$, and for any vertex x

[2]Note that we only need the off-path nodes to form an antichain and not the nodes on the path itself.

in $\mathrm{off}(\pi_v)$ and any vertex y in $\mathrm{off}(\pi_u)$, neither $x \preceq y$ nor $y \preceq x$. (If one of the conditions holds then it will contradict the fact that both u, v have height ℓ.)

The number of paths thus obtained is at $\Omega(s/\ell) = \Omega(d/2^{2\ell})$, and hence we are done in this case.

Case II, the number of leaves in T' is $< s/100\ell$: It is easy to see that the number of fan-in two nodes in any tree is upper bounded by the number of leaves in the tree. Therefore, T' has at most $s/100\ell$ degree two nodes. Let F' be the forest obtained by deleting all fan-in two nodes of T'. F' is a collection of paths. As we deleted degree two nodes, the total number of components created in F' is at most twice the number of degree two nodes, i.e., at most $s/50\ell$.

We call a component *small* if it has at most ℓ nodes and *large* otherwise. The total number of nodes in small components is at most $(s/50\ell) \cdot \ell = s/50$. We will not consider such components. Since $|T'| = s$, we are left with at least $s - s/50 \geq s/2$ nodes even if we discard all the small components.

Let C be a component with r vertices, where $r > \ell$. It can be broken down into $\lfloor \frac{r}{\ell+1} \rfloor$ paths, each of length ℓ. This will give us at least $\lfloor s/2(\ell+1) \rfloor$ many paths in total. As $\lfloor s/2(\ell+1) \rfloor > d/2^{2\ell}$, if we argue that all these paths are independent, we will be done.

It is not very hard to see why these paths are independent. Suppose two paths π, π' belonged to the same large component, then consider vertices $x \in \mathrm{off}(\pi)$ and $y \in \mathrm{off}(\pi')$. As T is a tree, neither $x \preceq y$ nor $y \preceq x$. Therefore, any such two paths are independent. Now say π, π' are two paths which come from two different large components. Then for $x \in \mathrm{off}(\pi)$ and $y \in \mathrm{off}(\pi')$, the common ancestor of x, y is a degree two node, which we deleted. Again, we can see that neither $x \preceq y$ nor $y \preceq x$. □

Given Claim 4.14, we proceed as follows. Applying Claim 4.14 with 4ℓ in place of ℓ, we obtain a set $P = \{\pi_1, \ldots, \pi_r\}$ of independent paths in T with $r = \Omega(d/2^{8\ell})$. For each π_i, let $\mathrm{off}(\pi_i) = \{u_{i,1}, \ldots, u_{i,4\ell}\}$. Note that these off-path nodes all exist since each internal node of T is assumed to have fan-in 2.

We now consider the effect of the random restriction I, chosen as in the lemma statement, on the tree T. For any $i \in [r]$ and $j \in [4\ell]$, let $Z_{i,j} \in \{0, 1\}$ be the random variable that is 1 if $u_{i,j}$ is present in $T|_I$ and is an odd node, and 0 otherwise; equivalently, if lab is the $[d]$-labelling of T, then $Z_{i,j} = 1$ iff the number of leaves v such that $\mathrm{lab}(v) \in I$ is odd. Note that $\mathbf{E}[Z_{i,j}] = (1/2)$ for each $i \in [r]$ and $j \in [4\ell]$. Moreover, since P is an independent set of paths, the sets of leaves in the subtrees of $u_{i,j}$ (for different i, j) are pairwise disjoint and consequently, the random variables $Z_{i,j}$ (for various i, j) are mutually independent. In particular,

by a Chernoff bound applied to $Z := \sum_{i\in[r],j\in[4\ell]} Z_{i,j}$, we get

$$\Pr[Z \le r\ell] = \Pr\left[Z \le \frac{1}{2}\mathbf{E}[Z]\right] \le \exp(-\Omega(\mathbf{E}[Z])) \le \exp(-\Omega(r\ell)) \le \exp(-\Omega(\frac{d}{2^{8\ell}})).$$

Note that Z is the total number of nodes in $\mathrm{off}(P)$ that end up as odd nodes in $T|_I$. Hence, if $Z > r\ell$, then the number of odd nodes per (surviving) path of P in $T|_I$ is at least $r\ell/r = \ell$. In particular, there must be some path π in $T|_I$ such that $|\mathrm{odd}(\pi)| \ge \ell$. This concludes the proof of Lemma 4.13.

Chapter 5

Polynomial Identity Testing

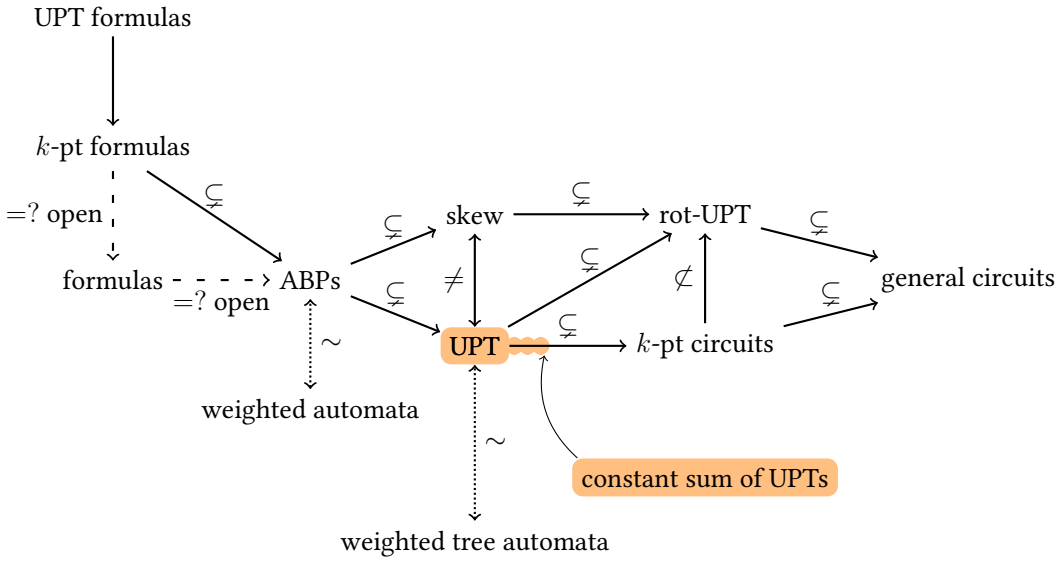


Figure 5.1: You are here.

Overview

Polynomial Identity Testing (**PIT** in short) is the important problem of determining whether a multivariate polynomial given by an arithmetic circuit equals formally zero (that is, each coefficient of the polynomial has to be zero). A very efficient and simple probabilistic algorithm is known for this problem: evaluate the polynomial at a random point over the field $\mathbb{Z}/p\mathbb{Z}$ for a large enough random prime p, and answer that the polynomial is formally zero if the evaluation equals zero. The correctness of this algorithm can be proved by using the Schwartz-

Zippel lemma (saying essentially that a non-zero low-degree polynomial does not have a lot of zeros) and the Prime Number Theorem. This places the algorithm in the complexity class coRP (the class of problems which admit probabilistic polynomial-time algorithms which never reject a good input but may accept a wrong one with small probability (false positive)). Nevertheless, no deterministic sub-exponential time algorithm for PIT is known. The main issue is to *derandomize* the problem, that is, to design a deterministic algorithm still working in polynomial time (or at least in sub-exponential time).

Testing polynomial identities is of great importance in algorithm design. For example, the best parallel algorithms for finding perfect matchings are based on testing whether a polynomial (in this case the determinant) equals formally zero. Designing a good deterministic algorithm for PIT would imply a good parallel deterministic algorithm for finding perfect matchings.

PIT is also linked with lower bounds. Indeed, in 2003, Kabanets and Impagliazzo have shown that if PIT has a deterministic polynomial-time algorithm, then either $\mathsf{NEXP} \not\subset \mathsf{P/poly}$ or the permanent does not have polynomial size arithmetic circuits. For a complete proof of this result, see [40] or [3]. This result explains partially why derandomization of PIT seems hard: it would imply lower bounds which are known hard to prove (because of a lot of barriers–diagonalization, algebraization, natural proofs, ...).

PIT is considered in mainly two ways:

- The *black-box* setting, in which one has access to a given polynomial f computed by a class of circuits only through evaluations of f over a set of points that the decider can choose.

- The *white-box* setting, in which the input of the problem is a circuit computing f itself.

We will focus on the white-box setting.

Remark 5.1

Observe that as evaluations of a polynomial f can be done efficiently given a circuit for f, a black-box PIT algorithm gives a white-box PIT algorithm working in roughly the same time as well. However, although *white-box* PIT algorithms should be easier to design, there is still no big difference on what we can solve in black-box model compared to the white-box model.

This chapter is based on the three following publications:

- Guillaume Lagarde, Guillaume Malod, and Sylvain Perifel. Non-commutative computations: lower bounds and polynomial identity testing. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:94, 2016

- Guillaume Lagarde, Nutan Limaye, and Srikanth Srinivasan. Lower bounds and PIT for non-commutative arithmetic circuits with restricted parse trees. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, pages 41:1–41:14, 2017

- Guillaume Lagarde, Nutan Limaye, and Srikanth Srinivasan. Lower bounds and PIT for non-commutative arithmetic circuits with restricted parse trees (extended version). *To appear in Computational Complexity*

5.1 PIT for UPT Circuits

In this section, we give two differents deterministic polynomial-time algorithms for PIT for the polynomials computed by UPT circuits.

- The first one 5.1.1 is based on the idea of the *Hadamard product*, used also by Arvind et al. [4] in the context of non-commutative ABPs. The idea is to show that if a polynomial f has a small UPT circuit, then so does the polynomial F each of whose coefficients is the square of the corresponding coefficient of f. When the underlying field is $\mathbb{R}$, all coefficients of F are now non-negative and hence F is the zero polynomial if and only if it evaluates to a non-zero value when each of its input variables is substitured by the scalar 1. While this method can be extended to work over $\mathbb{C}$ as well, it is not clear how to use this in the context of, say, positive characteristic.

- The second approach 5.1.2 leads a deterministic PIT algorithm for UPT circuits over general fields by adapating an older algorithm for ABPs due to Raz and Shpilka [42]. The idea behind this test is to compute, for each d', a small set of monomials of degree d' whose "coefficients"[1] determine the coefficients of all other monomials in the underlying polynomial. In particular, it reduces the problem of PIT for ABPs to computing the coefficient of a particular monomial, which can be done efficiently. A very similar idea turns out to work for UPT circuits as well.

[1] We write the underlying polynomial f of degree d computed by the ABP as a polynomial of degree d' where the coefficients are themselves polynomials of degree $d - d'$ and multiply the monomials of degree d' to the left.

5.1.1 Via Hadamard product

We will use the following binary operation over polynomials from [4], which is to polynomials what intersection is to formal languages.

Definition 5.2: Hadamard product

Given two polynomials in $\mathbb{F}\langle X\rangle$, $f = \sum_{m\in\mathcal{M}(X)} \alpha_m m$ and $g = \sum_{m\in\mathcal{M}(X)} \beta_m m$, the *Hadamard product* of f and g, written $f \odot g$, is:

$$f \odot g = \sum_{m\in\mathcal{M}(X)} \alpha_m \beta_m m$$

In [4], a logspace algorithm is given which, on input two ABPs A and B, outputs a new ABP C computing the Hadamard product of the polynomials computed by A and B. Consequently, they observed that this result gives the following derandomization for PIT.

Theorem 5.3: [4]

The problem of polynomial identity testing for non-commutative algebraic branching programs over $\mathbb{R}$ is in P.

Here, we extend this result: we give a construction to perform the Hadamard product of two UPT circuits with the same shape. In other words, we prove that the class of UPT 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 UPT circuits over $\mathbb{R}$.

Circuits will be assumed in normal form with multiplication fan-in 2, since Lemma 2.1 and Proposition 2.3 give explicit algorithms working in polynomial time to transform a UPT circuit with these two additional properties. The idea is then to create a circuit computing iteratively the Hadamard product of all pairs of addition gates of same type. The regularity of the parse tree will allow us to spread the Hadamard product layer by layer.

First, we state without proof the following easy lemma, saying essentially that if two polynomials admit two similar decomposition, then their Hadamard product respects a similar decomposition as well that can be computed from the Hadamard product of the smaller bricks.

Lemma 5.4

Let $d, d' \in \mathbb{N}$ and let $(f_i)_{1 \leq i \leq n}$ and $(g_i)_{1 \leq i \leq m}$ be families of polynomials in $\mathbb{F}\langle X \rangle$ with $\deg(f_i) = d$ and $\deg(g_i) = d'$. Set also $(\alpha_{i,j})_{1 \leq i \leq n, 1 \leq j \leq m} \in \mathbb{R}^{nm}$ and $(\beta_{i,j})_{1 \leq i \leq n, 1 \leq j \leq m} \in \mathbb{R}^{nm}$. Then:

$$\left(\sum_{(i,j)} \alpha_{i,j} f_i g_j\right) \odot \left(\sum_{(i,j)} \beta_{i,j} f_i g_j\right) = \sum_{(i,j),(k,l)} \alpha_{i,j}\beta_{k,l}(f_i \odot f_k)(g_j \odot g_l).$$

Theorem 5.5: Hadamard product of two UPT circuits

Let C and D be two UPT circuits in normal form, of same shape, and of size s and s', that compute two polynomials f and g. Then $f \odot g$ is computed by a UPT circuit of size at most ss'; moreover, this circuit can be constructed in polynomial time.

Proof. The new circuit computes the Hadamard product of all pairs $(\Phi_1, \Phi_2) \in C \times D$ of addition gates of the same type. As the output gate in C and in D are of the same type (because C and D have the same shape), the new circuit will in particular compute the Hadamard product of f and g. If the degree of Φ_1 and Φ_2 is 1, then the Hadamard product is trivial since the gates compute variables.

Assume we have constructed the circuit up to layer i (that is, for each gate of degree less than or equal to i). We now show how to construct the layer $(i+1)$. Let $\Phi_1 \in C$ and $\Phi_2 \in D$ be two addition gates of degree $(i+1)$ and of same type. Because the circuits are UPT, Φ_1 (resp. Φ_2) computes a polynomial of the form $h_1 = (\sum_{(i,j)} \alpha_{i,j} f_i g_j)$ (resp. $h_2 = (\sum_{(i,j)} \beta_{i,j} f_i g_j)$), where the f_i are all of identical types, and where the g_j are also all of identical types. Lemma 5.4 then shows how to compute $h_1 \odot h_2$ from the previously computed $f_i \odot f_j$ and $g_i \odot g_j$.

By induction, we thus construct the desired circuit layer by layer. Given a type, if there were i (resp. j) addition gates of this type in C (resp. in D), we have created exactly ij gates in the new circuit. Therefore, the total number of gates in the new circuit is no more than ss'. □

Corollary 5.6

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

Proof. Given $f(x_1, \ldots, x_n)$ computed by a UPT circuit, construct the circuit which computes $(f \odot f)(x_1, \ldots, x_n)$ and evaluate it on $(1, 1, \ldots, 1)$. The output is the sum of the squares of the coefficients of f, therefore it is equal to 0 if and only if f is equal to the zero polynomial. □

Remark 5.7

From a UPT circuit computing a polynomial $f = \sum_{m \in \mathcal{M}(X)} \alpha_m m$ over $\mathbb{C}$, it is not hard to deduce a UPT circuit of same shape for the conjugate $\bar{f} = \sum_{m \in \mathcal{M}(X)} \bar{\alpha}_m m$. Therefore, a similar algorithm works over $\mathbb{C}$, since $(f \odot \bar{f}) = \sum_{m \in \mathcal{M}(X)} |\alpha_m|^2 m$.

We also obtain another corollary that is to be compared with the results of Section 2.4.2.

Corollary 5.8

Over $\mathbb{R}$, in the non-commutative setting, computing the determinant with an UPT circuit is as hard as computing the permanent.

Proof. Observe that $\mathrm{DET} \odot \mathrm{DET} = \mathrm{PERM}$. Therefore, by Theorem 5.5, from a circuit computing the determinant, we can build in polynomial time a circuit computing the permanent. □

5.1.2 Via Raz and Shpilka

In this section, we give another deterministic PIT algorithm for UPT circuits. Our algorithm, which is an adaptation of the algorithm of Raz and Shpilka [42], is field independent.

Theorem 5.9: Whitebox PIT for UPT circuits over all fields

Let $N, s \in \mathbb{N}$ be parameters. There is a deterministic algorithm running in time $\mathrm{poly}(s)$ which, on input a UPT circuit C of size at most s over N variables, decides whether C computes the zero polynomial.

Proof. Let C be the input UPT circuit. Let T be the unique parse tree of the circuit C (it is easy to determine T from the circuit C by constructing an arbitrary parse formula of C and obtaining the parse tree corresponding to it). By Proposition 2.3 and Lemma 2.1, we can assume without loss of generality that C is in normal form and that T has fan-in bounded by 2.

For each node $v \in T$, let r_v denote the number of $(v, \times)$-gates and t_v the number of $(v, +)$-gates. We also identify the $(v, \times)$-gates with $[r_v]$ and $(v, +)$-gates with $[t_v]$ in an arbitrary way. For any $v \in T$ and any monomial $m \in \mathcal{M}_{\deg(v)}(X)$, let $\xi^v_m \in \mathbb{F}^{r_v}$ be defined so that for any $i \in [r_v]$, the ith entry $\xi^v_m(i)$ of the vector ξ^v_m is the coefficient of the monomial m in the polynomial computed at the ith $(v, \times)$ gate. Similarly, let $\chi^v_m \in \mathbb{F}^{t_v}$ be the coefficient vector of the monomial m at the $(v, +)$-gates.

The idea of the algorithm is to compute, for each $v \in T$, a set $B_{v,+} \subseteq \mathcal{M}_{\deg(v)}(X)$ of size at most t_v such that the set of vectors $\tilde{B}_{v,+} = \{\chi^v_m \mid m \in B_{v,+}\}$ is a linearly independent set of vectors that generates all the vectors in $\tilde{C}_{v,+} := \{\chi^v_m \mid m \in \mathcal{M}_{\deg(v)}(X)\} \subseteq \mathbb{F}^{t_v}$. In particular, the polynomial computed by the circuit C is non-zero iff for u being the root of T, there is a $\chi \in \tilde{B}_{u,+}$ such that the entry of χ corresponding to the output gate of the circuit is non-zero.[2]

Thus, it suffices to compute the sets $B_{v,+}$ and $\tilde{B}_{v,+}$ for each $v \in T$. In order to do so, it will also help to compute $B_{v,\times} \subseteq \mathcal{M}_{\deg(v)}(X)$ and $\tilde{B}_{v,\times} = \{\xi^v_m \mid m \in B_{v,\times}\}$ of size at most r_v each so that the set of vectors $\tilde{B}_{v,\times}$ is a linearly independent set of vectors that generates all the vectors in $\tilde{C}_{v,\times} := \{\xi^v_m \mid m \in \mathcal{M}_{\deg(v)}(X)\} \subseteq \mathbb{F}^{r_v}$.

The algorithm begins by choosing the sets $B_{v,\times}$ for each leaf node $v \in T$. This may be done efficiently since $\deg(v) = 1$ for each leaf node and hence the number of monomials $m \in \mathcal{M}_{\deg(v)}(X)$ is exactly $|X| = N$. By computing the coefficient vectors for each such monomial and performing Gaussian elimination, we can find a suitable set $B_{v,\times}$ as required in time $\mathrm{poly}(N, s) = \mathrm{poly}(s)$.

To compute these bases for nodes higher up in T we proceed inductively as follows.

Sum gates. We first describe how to construct $B_{v,+}$ and $\tilde{B}_{v,+}$ given $B_{v,\times}$ and $\tilde{B}_{v,\times}$. Since each $(v, +)$-gate computes a linear combination of the $(v, \times)$-gates, we see that there is a matrix $M_v \in \mathbb{F}^{t_v \times r_v}$ such that $\chi^v_m = M_v \xi^v_m$ for every $m \in \mathcal{M}_{\deg(v)}(X)$. In particular, given sets $B_{v,\times}$ and $\tilde{B}_{v,\times}$ as above, the set $\{\chi^v_m \mid m \in B_{v,\times}\}$ is a *spanning set* for the set $\tilde{C}_{v,+}$. By Gaussian elimination, we can choose a basis $\tilde{B}_{v,+} \subseteq \tilde{B}_{v,\times}$ in time $\mathrm{poly}(N, t_v, r_v) = \mathrm{poly}(s)$ and choose $B_{v,+}$ to be the corresponding set of monomials.

[2]Recall that the output gate of the circuit C is always assumed to be a $+$ gate, possibly of fan-in 1.

Multiplication gates. Now let $v \in T$ be an internal node with children u and w. We show how to compute $B_{v,\times}$ and $\tilde{B}_{v,\times}$ given $B_{u,+}$, $B_{w,+}$, $\tilde{B}_{u,+}$ and $\tilde{B}_{w,+}$.

Let $r = r_v$ and let Φ_i be the ith $(v,\times)$-gate in C for each $i \in [r]$. Let Φ'_i and Φ''_i be the left and right children respectively of Φ_i; note that Φ'_i is a $(u,+)$-gate and Φ''_i a $(w,+)$-gate. For monomials $m' \in \mathcal{M}_{\deg(u)}(X)$ and $m'' \in \mathcal{M}_{\deg(w)}(X)$, let $\lambda^u_{m'}$ and $\lambda^w_{m''} \in \mathbb{F}^r$ denote the coefficient vectors of m' and m'' at the gates Φ'_i ($i \in [r]$) and Φ''_i ($i \in [r]$) respectively.[3] For any monomial m', each entry of the vector $\lambda^u_{m'}$ is the coefficient of the monomial m' at some $(u,+)$-gate and hence an entry of the vector $\chi^u_{m'}$. In particular, $\lambda^u_{m'} = P_u \chi^u_{m'}$ for some linear projection P_u; a similar fact is true for the $\lambda^w_{m''}$ as well. Thus, the vectors $\{\lambda^u_{m'} \mid m' \in B_{u,+}\}$ span all the vectors in $\{\lambda^u_{m'} \mid m' \in \mathcal{M}_{\deg(u)}(X)\}$ and similarly, $\{\lambda^w_{m''} \mid m'' \in B_{w,+}\}$ spans all the vectors in $\{\lambda^w_{m''} \mid m'' \in \mathcal{M}_{\deg(w)}(X)\}$.

Now, note that for any monomial $m \in \mathcal{M}_{\deg(v)}(X)$, there is a unique pair of monomials $m' \in \mathcal{M}_{\deg(u)}(X)$ and $m'' \in \mathcal{M}_{\deg(w)}(X)$ such that $m = m'm''$. Further, the coefficient of monomial m in the polynomial computed at Φ_i is the product of the coefficients of m' at Φ'_i and m'' at Φ''_i. In other words, we have $\lambda^v_m = \lambda^u_{m'} \cdot \lambda^w_{m''}$, the pointwise product of the vectors $\lambda^u_{m'}$ and $\lambda^w_{m''}$. By linearity, it follows that the coefficient vectors corresponding to the monomials in $B_{u,w} := B_{u,+} \cdot B_{w,+} = \{m'm'' \mid m' \in B_{u,+}, m'' \in B_{w,+}\}$ span $\tilde{C}_{v,\times}$. Since $|B_{u,w}|$ has size at most s^2, both $B_{u,w}$ and the corresponding coefficient vectors can be computed in time $\mathrm{poly}(s)$. By Gaussian elimination, we can find in time $\mathrm{poly}(s)$ the sets $B_{v,\times}$ and $\tilde{B}_{v,\times}$ as required.

This completes the description of the algorithm and its analysis. From the analysis above, it is clear that the algorithm runs in time $\mathrm{poly}(s)$. We have shown Theorem 5.9. □

5.2 PIT for sum of UPT circuits

In this section we will give a deterministic polynomial time algorithm for the PIT problem for the sum of k UPT circuits. Recently a deterministic algorithm was designed by Gurjar et al. [15] for polynomial identity testing of sum of ROABPs. Our algorithm uses a similar idea for the PIT of sum of UPT circuits. Our PIT algorithm is white box, i.e., it uses the structure of the underlying UPT circuits.

[3] Note that the gates Φ'_i and Φ'_j may coincide even if $i \neq j$. This does not matter for our argument.

Theorem 5.10: theorem

Let $N, s, k \in \mathbb{N}$ with $N \leq s$. There is a deterministic algorithm running in time $s^{O(2^k)}$ which, on input $k+1$ UPT circuits $C_0, C_1, \ldots, C_k$ (of possibly differing shapes) each of size at most s over N variables, decides whether $\sum_{i=0}^{k} C_i$ computes the zero polynomial.

Proof idea

Say that circuit C_i has shape T_i for $i \in [0, k]$ (it is easy to compute T_i given each C_i as observed in Section 5.1.2). By Proposition 2.3 and Lemma 2.1, we can assume without loss of generality that each C_i is in normal form and that T_i has fan-in bounded by 2.

Let P_i be the polynomial computed by the UPT circuit C_i for each $0 \leq i \leq k$. Let $P = -P_0$ and let $Q = \sum_{i=1}^{k} P_i$. Note that in this notation, checking whether $\sum_{i=0}^{k} P_i \equiv 0$ is equivalent to checking whether $P \equiv Q$. We will present an algorithm to do this in four steps.

> **Step 1:** We show how to efficiently build a small set of *characterizing identities* for the polynomial P. We will ensure that this set of identities is of size $\mathrm{poly}(N, s, d) = \mathrm{poly}(s)$.
>
> **Step 2:** We will then check whether all the identities hold for the polynomial Q as well. This is done by a call to the PIT algorithm for the sum of k UPT circuits. We will analyze the complexity of this step and bound it by $s^{O(2^k)}$.
>
> **Step 3:** We will then show that if Q satisfies all the characterizing identities, and moreover P and Q agree on a small set of coefficents, then the two polynomials are in fact identical.
>
> **Step 4:** We will show that testing the equality of the above set of coefficients of P and Q can also be performed in time $\mathrm{poly}(s)$.

We now give a more detailed outline of the above steps with the statements of many formal claims. For the sake of exposition, we postpone the proofs of these intermediate claims to the end of this section.

Step 1: We now introduce some notation to formally define the characterizing identities for a polynomial defined by a UPT circuit. Let $I = [i, j]$ be an interval in $[d]$, i.e., $1 \leq i \leq j \leq d$. If the interval is of size 1, i.e., $I = [i, i]$, then we simply

use i to denote it. Recall that $\mathcal{M}_{|I|}(X)$ stands for all monomials of degree exactly $|I|$. For any r, let $\mathbb{F}\langle X\rangle_r$ be the set of homogeneous polynomials of degree r.

For any interval I in $[d]$ and any monomial $m \in \mathcal{M}_{|I|}(X)$, we define a map $\partial_{I,m} : \mathbb{F}\langle X\rangle_d \to \mathbb{F}\langle X\rangle_{d-|I|}$, which is defined as follows:

$$\partial_{I,m}(P) = \sum_{m_1,m_2:deg(m_1)=i-1,deg(m_2)=d-j} \alpha_{m_1,m,m_2} \cdot m_1 \cdot m_2,$$

where α_{m_1,m,m_2} is the coefficient of the monomial $m_1 \cdot m \cdot m_2$ in P. Informally, $\partial_{I,m}$ is an operator which, when applied to a polynomial P of degree d, retains only those monomials of P (along with their coefficients) which have the monomial m at exactly the positions in the interval I, while substituting the constant 1 for all the variables in positions indexed by I.

Let T be T_0, the shape of the parse tree corresponding to P. For each $v \in T$, we use I_v to denote the interval $[\mathrm{pos}(v), \mathrm{pos}(v) + \deg(v) - 1]$ where $\mathrm{type}(v) = (\mathrm{pos}(v), \deg(v))$ is as defined in Section 1.3.

Starting from the leaves, we start building identities corresponding to each of the nodes in the tree T. Formally, we show the following inductive claim.

Claim 5.11

There is an algorithm that runs in time $\mathrm{poly}(s)$ that, for every node v in T, computes a set $B_v \subseteq \mathcal{M}_{\deg(v)}(X)$ such that $|B_v| \leq s$ and also

- if v is a leaf node and $I_v = i$ then, for each $x \in X$ and for each $m \in B_v$, it computes coefficients $c^v_{x,m}$ such that $\partial_{i,x}(P) = \sum_{m \in B_v} c^v_{x,m} \cdot \partial_{i,m}(P)$,
- if v is an internal node with children u, w, then for all $m' \in B_{u,w} := B_u \cdot B_w$ and for all $m \in B_v$, it computes coefficients $c^v_{m',m}$ such that $\partial_{I_v,m'}(P) = \sum_{m \in B_v} c^v_{m',m} \cdot \partial_{I_v,m}(P)$.

The algorithm for the above is almost identical to the PIT algorithm in Section 5.1.2. Note that the size of the output of the algorithm is $\mathrm{poly}(N, s, d) = \mathrm{poly}(s)$.

We will prove this claim later.

Step 2: Let us assume that the above claim holds. Now if $P = Q$, then the same set of identities must also hold for the polynomial Q. We now describe how one can check that Q satisfies these identities (the algorithm can safely reject if some identity is not satisfied by Q). Suppose we have all the identities for P along with the sets B_v for all nodes v in T and all the coefficients $\left(c^v_{m',m}\right)_{m' \in B_{u,w}, m \in B_v}$ again for every node v in T.

In general, we need to check identities of the following form when v is an internal node with children u, w: $\partial_{I_v,\tilde{m}}(Q) = \sum_{m \in B_v} c_{\tilde{m},m} \cdot \partial_{I_v,m}(Q)$ for each $\tilde{m} \in B_{u,w}$. (A similar check has to be made when v is a leaf node.)

Recall that $Q = \sum_{i=1}^{k} P_i$. Therefore, we can rewrite the above identity as follows:

$$\sum_{i=1}^{k} \partial_{I_v,\tilde{m}}(P_i) = \sum_{m \in B_v} c_{\tilde{m},m} \cdot \sum_{i=1}^{k} \partial_{I_v,m}(P_i).$$

Rearranging this we get

$$\sum_{i=1}^{k} \left[\sum_{m \in B_v} c_{\tilde{m},m} \cdot \partial_{I_v,m}(P_i) - \partial_{I_v,\tilde{m}}(P_i) \right] \equiv 0. \tag{5.1}$$

We first show that each of the k terms in the above sum has a small UPT circuit.

Claim 5.12

For each $i \in [k]$, $\sum_{m \in B_v} c_{\tilde{m},m} \cdot \partial_{I_v,m}(P_i) - \partial_{I_v,\tilde{m}}(P_i)$ can be computed by a UPT circuit of size at most $O(s^2)$. Further, these circuits can be constructed in time $\mathrm{poly}(k, s)$.

By the above claim, Equation 5.1 reduces to an identity testing question for the sum of at most k UPT circuits, and hence can be solved recursively. Finally, when we get to the case that $k = 1$, we simply appeal to our result from Section 5.1.2. Using Claim 5.12 (which we will prove later) and the algorithm from Section 5.1.2 for a single UPT circuit, we see that this step can be performed in time $(s^2)^{O(2^{k-1})} = s^{O(2^k)}$.

Step 3: Now suppose all the above checks succeed. That is, we have been able to ensure that the following statements hold:

- For every leaf node v, $x \in X, m \in B_v$ and i such that $I_v = i$:

$$\partial_{i,x}(P) = \sum_{m \in B_v} c^v_{x,m} \cdot \partial_{i,m}(P) \text{ and } \partial_{i,x}(Q) = \sum_{m \in B_v} c^v_{x,m} \cdot \partial_{i,m}(Q). \tag{5.2}$$

- For every internal node v with children u, w, for every $m \in B_v$ and $m' \in B_{u,w}$ we have:

$$\partial_{I_v,m'}(P) = \sum_{m \in B_v} c^v_{m',m} \cdot \partial_{I_v,m}(P) \text{ and } \partial_{I_v,m'}(Q) = \sum_{m \in B_v} c^v_{m',m} \cdot \partial_{I_v,m}(Q) \tag{5.3}$$

Claim 5.13

Equations 5.2, 5.3 imply that for any node $v \in T$ and any $m' \in \mathcal{M}_{|I_v|}(X)$ and $m \in B_v$, there exist $c^v_{m',m} \in \mathbb{F}$ such that

$$\partial_{I_v,m'}(P) = \sum_{m \in B_v} c^v_{m',m} \cdot \partial_{I_v,m}(P) \text{ and } \partial_{I_v,m'}(Q) = \sum_{m \in B_v} c^v_{m',m} \cdot \partial_{I_v,m}(Q).$$

Note that (5.2) and (5.3) only give us a polynomially large set of common identities satisfied by P and Q. The content of Claim 5.13 is that we can use these to infer an *exponentially* (since the size of $\mathcal{M}_{|I|}(X)$ is exponential) large set of common identitites for P and Q.

We will present the proof of Claim 5.13 later. For now let us assume this claim.

Now, let v_0 be the root of T. We check that for each $m \in B_{v_0}$, $\partial_{[d],m}(P) = \partial_{[d],m}(Q)$ (as described in Step 4). Note that for any m of degree d, $\partial_{[d],m}(P)$ and $\partial_{[d],m}(Q)$ are simply coefficients of the monomial m in P and Q respectively. Again, if any of these coefficients are not equal, we can safely reject. However, if these checks succeed, using Claim 5.13, we can see that *all* the coefficients of polynomials P and Q are equal and hence they are the same polynomial. In this case, we accept.

Step 4: As noted above, $\partial_{[d],m}(P)$ and $\partial_{[d],m}(Q)$ are simply coefficients of the monomial m in the polynomials P and Q respectively. We use the following lemma proved in [5] to compute these coefficients.

Lemma 5.14: [5]

Given access to a non-commutative circuit C of size s which is computing the polynomial f of degree d and given a monomial m, the coefficient of m in f can be computed in time polynomial in s, d.

This completes the description of the four main steps. We now prove the claims used in these steps.

Proof sketch of Claim 5.11. We follow exactly the procedure in the PIT algorithm for UPT circuits in Section 5.1.2 and compute sets $B_{v,+}, \tilde{B}_{v,+}, B_{v,\times}$, and $\tilde{B}_{v,\times}$ exactly as in that algorithm. We will take our sets B_v to be the sets $B_{v,\times}$ for each $v \in T$. Clearly $|B_v| \leq s$ for each v.

To compute the coefficients $c^v_{m',m} \in \mathbb{F}$, we proceed as follows. For any leaf node $v \in T$, $y \in X$ and $x \in B_v$, we choose $c^v_{y,x}$ such that we have $\xi^v_y = \sum_{x \in B_v} c^v_{y,x} \xi^v_x$.

For an internal node $v \in T$ with children u and w, and any $m' \in B_u \cdot B_w$, we note that by the definition of $B_{v,\times}$ in the proof of Theorem 5.9, each $m' \in B_u \cdot B_w$, $\xi^v_{m'}$ lies in $\mathrm{Span}(\tilde{B}_{v,\times})$ and hence we can find $c^v_{m',m}$ such that $\xi^v_{m'} = \sum_{m \in B_v} c^v_{m',m} \xi^v_m$.

This concludes the description of the algorithm. To show that this works as intended, it suffices to prove the following claim.

Claim 5.15

Let $v \in T$ and $t := \deg(v)$. For any m' such that $\deg(m') = t$ and for any set $B \subseteq \mathcal{M}_t(X)$, if $\xi^v_{m'} = \sum_{m \in B} c^v_{m',m} \cdot \xi^v_m$ then $\partial_{I,m'}(P) = \sum_{m \in B} c^v_{m',m} \cdot \partial_{I,m}(P)$.

Proof. Let $v \in T$ be such that $\mathrm{type}(v) = (t, p)$, where t is $\deg(v)$ and p is $\mathrm{pos}(v)$. Let K_v be the number of nodes in C_0 corresponding to v. For any polynomial computed by a UPT circuit, Chapter 2 gives the following decomposition lemma, which we will recall and use below.

Lemma 5.16

Let P be a polynomial of degree d computed by a UPT circuit of size s with a parse tree T. Let (t, p) be the type of a node $v \in T$ and let K_v be the number of gates in C of that type. Let $f_1, f_2, \ldots, f_{K_v}$ be the polynomials computed by these gates, each of degree t. Then P can be written as

$$P = \sum_{j=1}^{K_v} f_j \times_p h_j,$$

where $\forall j, 1 \leq j \leq K_v \ \deg(h_j) = d - t$.

Using the above lemma our claim follows. Given below is the detailed proof of the claim.

$$
\begin{aligned}
\partial_{I,m'}(P) &= \partial_{I,m'}\left(\sum_{j=1}^{K_v} f_j \times_p h_j\right) && (a)\\
&= \partial_{I,m'}\left(\sum_{j=1}^{K_v} \xi^v_{m'}(j)\cdot m' \times_p h_j + \sum_{\tilde{m}\neq m'} \xi^v_{\tilde{m}}(j)\cdot \tilde{m} \times_p h_j\right)\\
&= \partial_{I,m'}\left(\sum_{j=1}^{K_v} \xi^v_{m'}(j)\cdot m' \times_p h_j\right) + \partial_{I,m'}\left(\sum_{\tilde{m}\neq m'} \xi^v_{\tilde{m}}(j)\cdot \tilde{m} \times_p h_j\right)\\
&= \sum_{j=1}^{K_v} \xi^v_{m'}(j)\cdot 1 \times_p h_j + \cancelto{0}{\partial_{I,m'}\left(\sum_{\tilde{m}\neq m'} \xi^v_{\tilde{m}}(j)\cdot \tilde{m} \times_p h_j\right)}\\
&= \sum_{j=1}^{K_v}\sum_{m\in B} c^v_{m',m}\cdot \xi^v_m(j) \times_p h_j && (b)\\
&= \sum_{m\in B} c^v_{m',m}\sum_{j=1}^{K_v} \xi^v_m(j) \times_p h_j\\
&= \sum_{m\in B} c^v_{m',m}\cdot \partial_{I,m}\left(\sum_{j=1}^{K_v} \xi^v_m(j)\cdot m \times_p h_j\right)\\
&= \sum_{m\in B} c^v_{m',m}\cdot \left(\partial_{I,m}\left(\sum_{j=1}^{K_v} \xi^v_m(j)\cdot m \times_p h_j\right) + \partial_{I,m}\left(\sum_{\tilde{m}\neq m} \xi^v_{\tilde{m}}(j)\cdot \tilde{m} \times_p h_j\right)\right)\\
&= \sum_{m\in B} c^v_{m',m}\cdot \partial_{I,m}\left(\sum_{j=1}^{K_v} \xi^v_{m'}(j)\cdot m \times_p h_j + \sum_{\tilde{m}\neq m} \xi^v_{\tilde{m}}(j)\cdot \tilde{m} \times_p h_j\right)\\
&= \sum_{m\in B} c^v_{m',m}\cdot \partial_{I,m}(P)
\end{aligned}
$$

The identity (a) holds due to Lemma 5.16. The identity (b) follows due to our assumption in the statement of the claim. The other identities follow due to the definition and/or by the linearity of $\partial_{I,m}$. □

□

Proof of Claim 5.12. We know that P_i has a UPT circuit C_i with shape T_i. Say we fix a monomial m and an interval $I = [i_1, i_2]$ such that $\deg(m) = |I|$. Let T_i' be the tree obtained from T_i by deleting all nodes u such that $I_u \subseteq I$. We claim that $\partial_{I,m}(P_i)$ is computed by a UPT circuit of size at most s and shape T_i'.

Consider any leaf node $w \in T_i$ such that $\mathrm{pos}(w) = i_1 + \ell - 1 \in I$. We consider each $(w, \times)$ gate Φ of C (note that these are input gates) and replace the gate by 0 if the variable x labelling Φ is the ℓth variable in m and 0 otherwise.

This gives a non-commutative arithmetic circuit where some leaves are labelled by constants. However, these constants are easily eliminated inductively as follows. For any $\times$ gate Φ' which has a child labelled by a constant α, we can remove the child and multiply the label of each wire leaving Φ' by α; for any $+$ gate Φ' which has a child labelled by a constant, it must be the case that *all* its children are labelled by constants (this follows from the UPT restriction) and hence the $+$ gate can now be labelled by a constant as well. Continuing this way, all the gates with constant labels are eliminated.

It can be checked that the circuit thus obtained is a UPT circuit of size at most s and shape T_i' computing $\partial_{I,m}(P_i)$. Returning to the statement of the claim, we have therefore shown that each of $\partial_{I_v,\tilde{m}}(P_i)$ and $\partial_{I_v,m}(P_i)$ can be computed by a UPT circuit of size s and shape T_i'.

Therefore, we can compute $\sum_{m \in B_v} c_{\tilde{m},m} \cdot \partial_{I_v,m}(P_i) - \partial_{I_v,\tilde{m}}(P_i)$ by a linear combination of the $O(s)$ UPT circuits computing $\partial_{I_v,m}(P)$ for $m \in B_v \cup \{\tilde{m}\}$. Overall, this gives a UPT circuit of size $O(s^2)$. Since the above proof is constructive, we can actually find this circuit in time $\mathrm{poly}(s)$. □

Proof of Claim 5.13. We will prove this claim by induction on $|I_v|$.

The base case is $|I_v| = 1$ which follows directly from Equation 5.2.

Suppose $|I_v| = t > 1$. Then v is an internal node in T. Let u, w be its two children. This implies that $I_v = I_u \cup I_w$. Let $|I_u| = t_1, |I_w| = t_2$. Note that $t_1, t_2 < t$.

We wish to prove that for any $m' \in \mathcal{M}_{|I_v|}(X)$, $\partial_{I_v,m'}(P) = \sum_{m \in B_v} c^v_{m',m} \cdot \partial_{I_v,m}(P)$ and $\partial_{I_v,m'}(Q) = \sum_{m \in B_v} c^v_{m',m} \cdot \partial_{I_v,m}(Q)$ for a suitable choice of $c^v_{m',m}$. Note that this already follows from (5.3) if $m' \in B_u \cdot B_w$. So we assume that $m' \notin B_u \cdot B_w$.

Let $m' = m'_1 \cdot m'_2$, where $\deg(m'_1) = t_1$ and $\deg(m'_2) = t_2$. Let R be either P or Q.

$$
\begin{aligned}
\partial_{I_v,m'}(R) &= \partial_{I_u \cup I_w, m'_1 m'_2}(R) \\
&= \partial_{I_u,m'_1} \circ \partial_{I_w,m'_2}(R) && (a) \\
&= \partial_{I_u,m'_1} \circ \left[\sum_{\overline{m} \in B_w} c^w_{m'_2,\overline{m}} \cdot \partial_{I_w,\overline{m}}(R) \right] && (b) \\
&= \sum_{\overline{m} \in B_w} c^w_{m'_2,\overline{m}} \cdot \partial_{I_u,m'_1} \circ \partial_{I_w,\overline{m}}(R) \\
&= \sum_{\overline{m} \in B_w} c^w_{m'_2,\overline{m}} \cdot \partial_{I_w - |I_u|,\overline{m}} \circ \partial_{I_u,m'_1}(R) && (a) \\
&= \sum_{\overline{m} \in B_w} c^w_{m'_2,\overline{m}} \sum_{\underline{m} \in B_u} c^u_{m'_1,\underline{m}} \cdot \partial_{I_w - |I_u|,\overline{m}} \circ \partial_{I_u,\underline{m}}(R) && (b) \\
&= \sum_{\overline{m} \in B_w, \underline{m} \in B_u} c^u_{m'_1,\underline{m}} \cdot c^w_{m'_2,\overline{m}} \cdot \partial_{I_u \cup I_w, \underline{m}\overline{m}}(R) && (a) \\
&= \sum_{\underline{m} \cdot \overline{m} \in B_{u,w}} c^u_{m'_1,\underline{m}} \cdot c^w_{m'_2,\overline{m}} \cdot \partial_{I_v,\underline{m}\overline{m}}(R) \\
&= \sum_{m \in B_v} \left(\sum_{\underline{m},\overline{m}} c^u_{m'_1,\underline{m}} \cdot c^w_{m'_2,\overline{m}} \cdot c^v_{\underline{m}\overline{m},m} \right) \cdot \partial_{I_v,m}(R). && (c)
\end{aligned}
$$

The equalities marked by (a) follow due to Observation 5.17 below. The equalities marked (b) follow due to the induction hypothesis. Finally, the equality marked (c) follows due to Equation 5.3.

The above implies the inductive claim with $c^v_{m',m}$ defined to be $\left(\sum_{\underline{m} \in B_u, \overline{m} \in B_w} c^u_{m'_1,\underline{m}} \cdot c^w_{m'_2,\overline{m}} \cdot c^v_{\underline{m}\overline{m},m} \right)$. Since the choice of $c^v_{m',m}$ is the same for both P and Q, we are done.

□

Observation 5.17

Let I, J be two contiguous intervals in $[d]$ such that I precedes J, i.e., if $I = [i_1, i_2]$ and $J = [j_1, j_2]$ then $1 \leq i_1, i_2 + 1 = j_1$ and $j_2 \leq d$. Then $\partial_{I \cup J, m_1 m_2} = \partial_{J - |I|, m_2} \circ \partial_{I,m_1} = \partial_{I,m_1} \circ \partial_{J,m_2}$, where for any two intervals I, J, $J - |I|$ denotes the interval $\{j - |I| \mid j \in J\} \cap [d]$.

Chapter 6

Automata, Circuits, Hankel Matrix

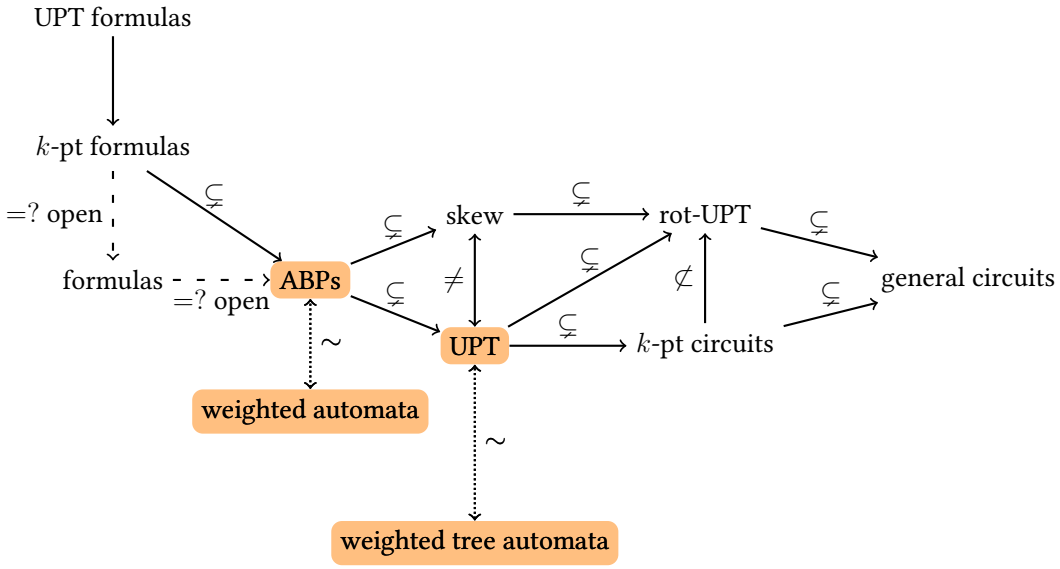


Figure 6.1: You are here.

Overview

This chapter revisits Nisan's result about ABPs and results from Chapter 2 about UPT circuits. The conceptual contribution is an exact correspondence between circuits and weighted automata: algebraic branching programs are captured by weighted automata over words, and circuits with unique parse trees by weighted automata over trees.

The key notion for understanding the minimisation question of weighted automata is the Hankel matrix: the rank of the Hankel matrix of a word or tree

series is exactly the size of the smallest weighted automaton recognising this series. For automata over words, the correspondence we establish allows us to rephrase Nisan's celebrated tight bounds for algebraic branching programs. We extend this result by considering automata over trees and obtain tight bounds for all circuits with unique parse trees (whereas tight bounds were previously obtained only for UPT circuits in some normal form, a constraint that can increase a lot the size of the circuits.).

Correspondence with weighted automata. We build a bridge between automata theory and arithmetic complexity. The correspondence is summarised in this table.

Arithmetic complexity	**Weighted automata**
variable	letter
monomial	word
polynomial	word series or tree series
algebraic branching program	layered weighted automaton over words
circuit with unique parse trees	layered weighted automaton over trees

In Section 6.1, we show that Nisan's results about algebraic branching programs follow from theorems coming from automata theory. We introduce weighted automata (WA) over words, and show the following correspondence:

- any ABP can be seen as a WA,
- under some syntactic restriction called layered, a WA can be seen as an ABP,
- for a given word series f, the size of the minimal WA recognising f is the rank of the Hankel matrix of f,
- if the word series f represents a homogeneous polynomial P, then the minimal WA recognising f is an ABP computing P.

This gives an alternative approach to state and prove Nisan's result about minimal ABPs.

Our main technical contribution is to extend this result from ABPs to UPT circuits in Section 6.2. To this end we show a more subtle and involved correspondence between UPT circuits and weighted automata over trees. Our main result is tight bounds on the size of a UPT circuit computing a given polynomial.

A similar result was obtained in Chapter 2 for UPT circuits in *normal form*; here we remove this assumption, leading to an exponential improvement in some examples as explained in Section 6.3.

This chapter is based on the following publication:

- Nathanaël Fijalkow, Guillaume Lagarde, and Pierre Ohlmann. Tight bounds using hankel matrix for arithmetic circuits with unique parse trees. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:38, 2018

6.1 Tight bounds for algebraic branching programs

We give here a definition of ABPs that slightly differs from Chapter 3. The two can easily be proved to be equivalent. However, this one is more convenient in order to state the correspondence with weighted automata.

Definition 6.1

An *algebraic branching program* (ABP) is a directed acyclic graph with a distinguished source vertex s. The vertices are partitioned into $d+1$ layers, starting with layer 0 which contains only the vertex s, and ending in layer d. Each edge is between two consecutive layers and is labeled by a homogeneous linear function over the variables X and real-valued constants. Each vertex in the last layer has a real output value. See Figure 6.2.

A path from s to a vertex in layer d induces a homogeneous polynomial of degree d obtained by multiplying all the labels of the edges and the output value. An algebraic branching program $\mathcal{C}$ computes a homogeneous polynomial $P_{\mathcal{C}}$ defined by summing the polynomials over all paths from s to vertices of the layer d.

The size of an ABP is its number of vertices.

One of the motivations for studying algebraic branching programs is that they capture matrix multiplication (cf Chapter 4). They can be proved to be equivalent to left skew circuits (*i.e.*, circuits for which the left argument of any multiplication gate is an input).

Nisan's theorem

Nisan's theorem gives for a homogeneous polynomial P the size of the smallest ABP computing P. Let d be the degree of P. For each $i \in \{0, \dots, d\}$, we define a matrix $M_{P,i}$ as follows. The rows are indexed by monomials of degree i, and the columns by monomials of degree $d-i$ (there are $|X|^i$ rows and $|X|^{d-i}$ columns). Then for u a monomial of degree i and v of degree $d-i$, define $M_{P,i}(u,v)$ to be the coefficient of uv in P. We now state Nisan's theorem (already introduced without any formal statement in Chapter 1)

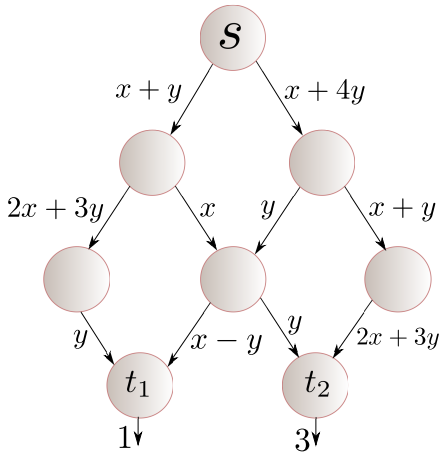


Figure 6.2: An example of an ABP with 4 layers and of size 8.

Theorem 6.2: Nisan's theorem

Let P be a homogeneous polynomial of degree d, and

$$n = \sum_{i=0}^{d} \text{rank}(M_{P,i}).$$

- Any ABP computing P has size at least n,
- There exists an ABP computing P of size exactly n.

In this section, we give an alternative proof of this theorem using a correspondence with weighted automata over finite words. The number n will appear as the rank of a single matrix called the Hankel matrix, which will in our case consist of $d+1$ independent blocks, hence the summation in Nisan's theorem.

The point of this section is to serve as an introduction to our main result in Section 6.2. Indeed, we will prove a theorem extending Nisan's theorem to a wider class of arithmetic circuits, namely circuits with unique parse trees, following the same schema. In this section we deal with weighted automata over words, in the next section we will consider weighted automata over trees.

Weighted automata over words

An element of X^* can be seen either as a monomial over the variables X, as in ABPs, or as a (finite) word over the alphabet X. A word series is a function $f : X^* \to \mathbb{R}$.

A polynomial P with variables in X can be seen as a word series $X^* \to \mathbb{R}$ which we also write P, such that $P(w)$ is the coefficient of w in P.

Definition 6.3

A *weighted automaton over words* (WA) is given by

- a finite set of states Q,
- an initial state $q_0 \in Q$,
- a transition function $\Delta : Q \times X \times Q \to \mathbb{R}$,
- an output function $F : Q \to \mathbb{R}$.

The usual definition proceeds with introducing runs, explaining that along a run the weights of the transitions are *multiplied*, and that the value of a word is the *sum* of the values of its accepting runs. We use here a more algebraic equivalent definition. Equivalently, we see the transition function as $\Delta : X \to \mathbb{R}^{Q \times Q}$, *i.e.*, $\Delta(x)$ is a matrix defined by $\Delta(x)(p, q) = \Delta(p, x, q)$. The initial state q_0 (seen as an element of $\mathbb{R}^Q$) and the transition function induce $\Delta^* : X^* \to \mathbb{R}^Q$ defined by $\Delta^*(\varepsilon) = q_0$ and $\Delta^*(wx) = \Delta^*(w) \cdot \Delta(x)$, where $\cdot$ is matrix multiplication. Similarly, we see F as a vector in $\mathbb{R}^Q$.

The weighted automaton $\mathcal{A}$ recognises the word series $f_{\mathcal{A}} : X^* \to \mathbb{R}$ defined by

$$f_{\mathcal{A}}(w) = \Delta^*(w) \cdot F,$$

where $\cdot$ is the dot product in $\mathbb{R}^Q$.

The size of a WA is its number of states.

Algebraic branching programs as weighted automata over words

ABPs form a subclass of WA over words that we define now.

Definition 6.4

A weighted automaton $\mathcal{A} = (Q, q_0, \Delta, F)$ is *d-layered* if Q can be partitioned into $d+1$ subsets $Q_0, \ldots, Q_d$ such that

(1) $Q_0 = \{q_0\}$,

(2) for all $x \in X, q, q' \in Q$, if $\Delta(q, x, q') \neq 0$ then there exists $i \in \{0, \ldots, d-1\}$ such that $q \in Q_i$ and $q' \in Q_{i+1}$,

(3) for all $q \in Q$, if $F(q) \neq 0$ then $q \in Q_d$.

Lemma 6.5

- For all ABP $\mathcal{C}$, there exists a WA over words $\mathcal{A}$ of the same size such that $f_{\mathcal{A}} = P_{\mathcal{C}}$.
- For all WA over words $\mathcal{A}$ which is layered, there exists an ABP $\mathcal{C}$ of the same size such that $P_{\mathcal{C}} = f_{\mathcal{A}}$.

Proof. Both claims are syntactic easy transformations. We explicit the construction to help the reader's intuition.

Let $\mathcal{C}$ be an ABP. We define a WA over words $\mathcal{A}$ as follows. The set of states is the set of vertices of $\mathcal{C}$, the initial state of $\mathcal{A}$ is the source vertex of $\mathcal{C}$, and the output function $F : Q \to \mathbb{R}$ is defined by $F(v)$ to be the output value of v if v is on the last layer, and 0 otherwise.For the transition function, let $\Delta(v, x, v')$ be the coefficient of x in the linear function labeling the edge (v, v') if $(v, v') \in E$, and 0 otherwise. See Figure 6.3. We have $f_{\mathcal{A}} = P_{\mathcal{C}}$.

For the second claim, the definition of d-layered WA over words exactly says that the above construction can be reversed. □

Fliess' theorem

The key notion of this chapter is the Hankel matrix of a series.

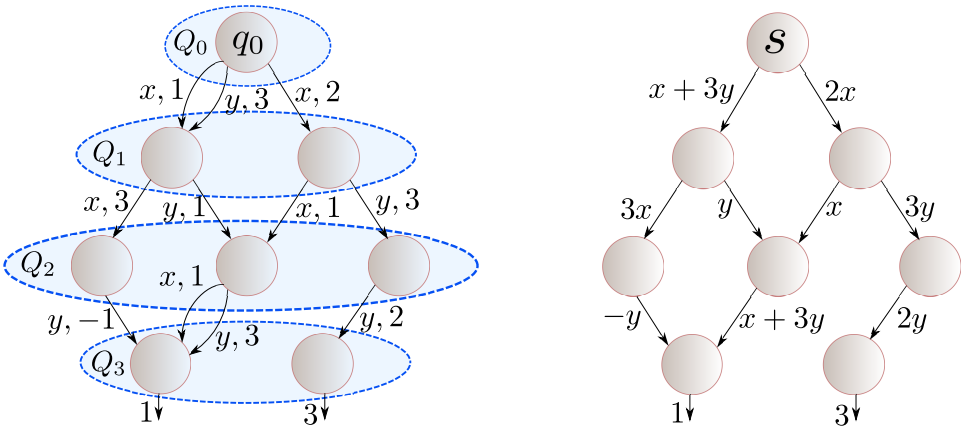


Figure 6.3: An example of 3-layered WA on the left, and its corresponding ABP on the right.

Definition 6.6

Let $f : X^* \to \mathbb{R}$, we define the (infinite) Hankel matrix $H^{(f)} \in \mathbb{R}^{X^*} \times \mathbb{R}^{X^*}$, whose rows and columns are indexed by words, by

$$H^{(f)}(u, v) = f(u \cdot v).$$

The notion of Hankel matrix and the rank of a formal non-commutative series were introduced by Carlyle and Paz [8]. One of the main results of Fliess' PhD thesis was the following theorem [13].

Theorem 6.7

Let $f : X^* \to \mathbb{R}$ be a word series such that $\text{rank}(H^{(f)})$ is finite.

- Any WA recognising f has size at least $\text{rank}(H^{(f)})$.
- There exists a WA recognising f of size exactly $\text{rank}(H^{(f)})$.

This article is in French. However, one can find great exposition of the ideas in the handbooks of Berstel and Reutenauer [6] and Sakarovitch [43]. The proof of the second item gives a construction of the WA recognising f that we detail now as we will need it to prove Theorem 6.2.

Recall that the rows of $H^{(f)}$ are indexed by words in X^*. For $u \in X^*$, let $H^{(f)}_u$ be the row corresponding to u in $H^{(f)}$, which we see as $H^{(f)}_u \in \mathbb{R}^{X^*}$. Let $Q \subseteq X^*$

such that $\left\{H_u^{(f)} \mid u \in Q\right\}$ is a basis of Span $\left\{H_u^{(f)} \mid u \in X^*\right\}$. We furthermore assume that $\varepsilon \in Q$, which is possible since $H_\varepsilon^{(f)} \neq 0$ unless f is the constant zero function.

We now construct the WA recognising f. The set of states is Q, the initial state is ε, and the output function is defined by $F(u) = f(u)$ for $u \in Q$. We now define the transition function. For $u \in Q$, there is a unique decomposition of $H_{ux}^{(f)}$ on the basis $\left\{H_u^{(f)} \mid u \in Q\right\}$:

$$H_{ux}^{(f)} = \sum_{v \in Q} \lambda(u, x, v) H_v^{(f)},$$

we define $\Delta(u, x, v) = \lambda(u, x, v)$.

Proof of Nisan's theorem

Lemma 6.8

Let P be a homogeneous polynomial of degree d. The automaton constructed above for recognising P is d-layered.

Proof. Let $\mathcal{A} = (Q, \varepsilon, \Delta, F)$ be the automaton described in the previous subsection.

Since for u of length larger than d we have $H_u^{(f)} = 0$, it implies that $Q \subseteq X^{\leq d}$. For $i \in \{0, \dots, d\}$, we let $Q_i = Q \cap X^i$. The conditions (1) and (3) are clearly satisfied, so we focus on (2).

For $i \in \{0, \dots, d\}$, let V_i denote the vector space spanned by $\left\{H_u^{(P)} \mid u \in Q_i\right\}$. Note that for $u \in Q_i$ and v of length j, if $i + j \neq d$ then $H_u^{(P)}(v) = P(u \cdot v) = 0$, hence the same is true for any $L \in V_i$: if v has length j such that $i + j \neq d$, then $L(v) = 0$.

We claim that the subspaces $V_0, V_1, \dots, V_d$ are in direct sum. Indeed, assume that $\sum_{i=0}^d L_i = 0$ with $L_i \in V_i$. Let $j \in \{0, \dots, d\}$ such that $L_j \neq 0$, and consider a word $v \in X^{d-j}$. For $i \neq j$ we have $L_i(v) = 0$ thanks to the remark above. It follows that $L_j(v) = 0$ for all $v \in X^{d-j}$, implying again with the remark above that $L_j = 0$, a contradiction. Thus the subspaces $V_0, V_1, \dots, V_d$ are in direct sum.

Let $u \in Q$ and $x \in X$. By definition

$$H_{ux}^{(P)} = \sum_{v \in Q} \Delta(u, x, v) H_v^{(P)} = \sum_{i=0}^{d} \sum_{v \in Q_i} \Delta(u, x, v) H_v^{(P)}.$$

But since $H^{(P)}_{ux} \in V_{|u|+1}$ and the vector spaces $V_0, \dots, V_d$ are in direct sum, it follows that for $v \in Q$ of length $i \neq |u|+1$ we have $\sum_{v \in Q_i} \Delta(u,x,v) H^{(P)}_v = 0$. Since the vectors $\left\{ H^{(P)}_v \mid v \in Q_i \right\}$ are linearly independent, this implies that $\Delta(u,x,v) = 0$. Thus the property (2) is satisfied. □

We now explain how to obtain the proof of Nisan's theorem (Theorem 6.2) from the correspondence. Let P be a homogeneous polynomial of degree d. Thanks to the first item of Theorem 6.7 any WA recognising P has size at least $\text{rank}(H^{(P)})$, and thanks to the first item of Lemma 6.5 this implies that any ABP computing P has size at least $\text{rank}(H^{(P)})$. Thanks to the second item of Theorem 6.7, there exists a WA recognising P of size $\text{rank}(H^{(P)})$, and thanks to Lemma 6.8, it is d-layered. Thanks to the second item of Lemma 6.5, it induces an ABP computing P of size $\text{rank}(H^{(P)})$.

Let us have a closer look at the Hankel matrix of P for a homogeneous polynomial P of degree d. Most of the matrix is filled with zeros, except for $d+1$ independent blocks, which are precisely the matrices $M_{P,i}$ for $i \in \{0, \dots, d\}$. This explains the summation in Nisan's statement of the theorem, since $\text{rank}(H^{(P)}) = \sum_{i=0}^{d} \text{rank}(M_{P,i})$.

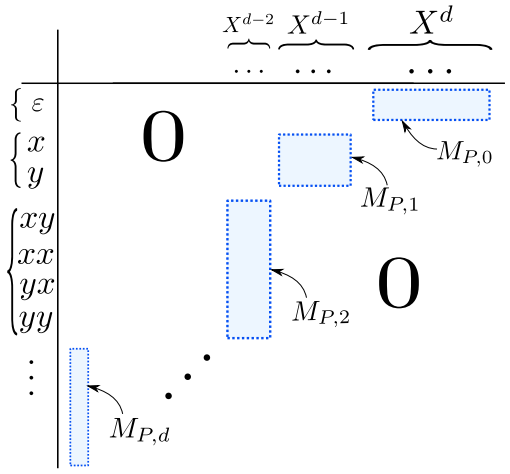


Figure 6.4: The Hankel matrix of P consists of $d+1$ independent blocks.

As we shall see in the next section, when considering UPT circuits the blocks will no longer be independent, allowing to share the result of partial computations.

6.2 Tight bounds for circuits with unique parse trees

In this section, we allow circuits to have multiple output in order to make the correspondence with weighted automata over trees cleaner. To prevent any confusion, we state below the definition that will be used in the rest of the chapter.

Definition 6.9

A circuit is a directed acyclic graph whose vertices, called gates, are of four different types.

- The *input gates* have indegree zero and are labeled with variables $x \in X$.
- The *addition gates* have unbounded fan-in and perform a linear combination of their inputs, with the associated coefficients α in $\mathbb{R}$ given on the edges.
- The *multiplication gates* have fan-in two, their arguments are ordered and the multiplication is interpreted according to this order (the left argument is multiplied before the right argument).
- The *output gates* have outdegree zero and are labeled with a real output value.

The *size* of an algebraic circuit is its number of addition gates.

While this definition allows multiple output gates, the circuits we construct only have one single output.

The reason for not taking the multiplication gates into account is because of Lemma 2.5 from Chapter 2, proving that for UPT circuits (which are the only circuits we consider), if s is the number of addition gates, then the number of multiplication gates can always be bounded by s^2. Therefore, the number of addition gates gives a pretty good idea on the total size of UPT circuits. Observe also that the situation is similar in Nisan's work for ABPs as the vertices of an ABP correspond exactly to addition gates when this ABP is converted into circuits.

We also normalise the circuits by requiring that all paths alternate between addition gates and multiplication gates and start and finish with an addition gate, which increases the size by at most a linear factor.

Statement of the result

Shapes are binary trees without any labels, we use $T, T', \dots$ for shapes. They can be built inductively: a leaf is a shape, and given two shapes T_1, T_2, the shape $T_1 \cdot T_2$ is a root with T_1 as left subtree and T_2 as right subtree.

We also consider labeled trees (later refered to as trees), which are binary trees whose leaves are labeled by variables $x \in X$. We let $\mathrm{Tree}(X)$ denote the set of trees, and use $t, t', \dots$ for them. Trees can be built inductively similarly as shapes, except that the basic trees are variables $x \in X$. A tree series is a function $f : \mathrm{Tree}(X) \to \mathbb{R}$.

For $t \in \mathrm{Tree}(X)$, we let $\widetilde{t}$ be the underlying shape of t, and for a shape T, we let $\mathrm{Tree}(X, T)$ denote the set of trees t such that $\widetilde{t} = T$.

We define the Hankel matrix for tree series. We first need the notion of contexts: a context C is an element of $\mathrm{Tree}(X \cup \{\Box\})$ with a unique leaf labeled $\Box$. We let $\mathrm{Context}(X)$ denote the set of contexts, and use $C, C', \dots$ for them. A context C and a tree t can be composed into a tree $C \circ t$ by replacing the placeholder $\Box$ by the tree t.

Definition 6.10

Let $f : \mathrm{Tree}(X) \to \mathbb{R}$, we define the (infinite) Hankel matrix $H^{(f)} \in \mathbb{R}^{\mathrm{Tree}(X)} \times \mathbb{R}^{\mathrm{Context}(X)}$ by $H^{(f)}(t, C) = f(C \circ t)$.

Our main theorem gives tight bounds on the size of UPT circuits. More precisely, we consider a homogeneous polynomial P of degree d and a shape T, and find the size of the smallest UPT circuit with shape T computing P.

A necessary condition for the existence of such a circuit is that the number of leaves of T is d. Under this condition a homogeneous polynomial P of degree d and a shape T induce a function $f_{P,T} : \mathrm{Tree}(X) \to \mathbb{R}$: for $t \in \mathrm{Tree}(X, T)$, we see t as a monomial and define $f_{P,T}(t)$ as the coefficient of this monomial in P, the function $f_{P,T}$ is zero outside of $\mathrm{Tree}(X, T)$. We refer to Figure 6.5 for an illustration of this definition.

For the sake of simplicity we use $H^{(P,T)}$ instead of $H^{(f_{P,T})}$.

Our main result is the following:

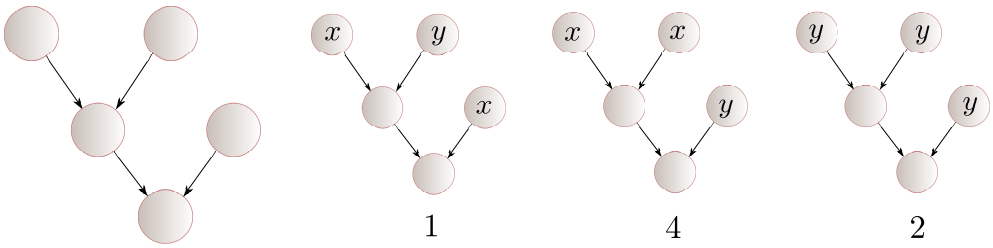


Figure 6.5: Given the shape T displayed on the left hand side, the polynomial $P = xyx + 4x^2y + 2y^3$ can be seen as the tree series $P : \mathrm{Tree}(X) \to \mathbb{R}$ associating with these three trees the values indicated below them and zero to all other trees.

Theorem 6.11

Let P be a homogeneous polynomial of degree d, and T be a shape with d leaves.

- Any UPT circuit for P with shape T has size at least $\mathrm{rank}(H^{(P,T)})$,
- There exists a UPT circuit with shape T computing P of size exactly $\mathrm{rank}(H^{(P,T)})$.

Weighted automata over trees

Definition 6.12: weighted automaton over trees

A *weighted automaton over trees* (WA) is given by

- a finite set of states Q,
- an initial function $\iota : X \times Q \to \mathbb{R}$,
- a transition function $\Delta : Q \times Q \times Q \to \mathbb{R}$,
- an output function $F : Q \to \mathbb{R}$.

Equivalently, we write the transition function as a bilinear function $\Delta : \mathbb{R}^Q \times \mathbb{R}^Q \to \mathbb{R}^Q$ defined by $\Delta(p, q)(r) = \Delta(p, q, r)$.

The initial and transition functions induce $\Delta^* : \mathrm{Tree}(X) \to \mathbb{R}^Q$ defined by $\Delta^*(x) = \iota(x) \in \mathbb{R}^Q$ and $\Delta^*(t_1 \cdot t_2) = \Delta(\Delta^*(t_1), \Delta^*(t_2))$.

The weighted automaton $\mathcal{A}$ recognises the tree series $f_{\mathcal{A}} : \mathrm{Tree}(X) \to \mathbb{R}$ defined by

$$f_{\mathcal{A}}(t) = \Delta^*(t) \cdot F,$$

where $\cdot$ is the dot product in $\mathbb{R}^Q$.

UPT circuits as weighted automata over trees

UPT circuits form a subclass of WA over trees that we define now.

Let T be a shape and v a node of T, we let T_v be the subshape of T rooted in v. We let $[T]$ denote $\{T_v \mid v \text{ node of } T\}$.

Definition 6.13

A weighted automaton $\mathcal{A} = (Q, \iota, \Delta, F)$ is *T-layered* if there exists a map $m : Q \to [T]$ such that

(1) for all $x \in X$, if $\iota(x, q) \neq 0$ then $m(q)$ is the shape reduced to a single leaf,

(2) for all $q, q_1, q_2 \in Q$, if $\Delta(q, q_1, q_2) \neq 0$, then $m(q) = m(q_1) \cdot m(q_2)$,

(3) for all $q \in Q$, if $F(q) \neq 0$ then $m(q) = T$.

Lemma 6.14

- For all UPT $\mathcal{C}$ with shape T, there exists a T-layered WA over trees $\mathcal{A}$ of the same size such that $f_{\mathcal{A}} = P_{\mathcal{C}}$.
- For all WA over trees $\mathcal{A}$ which is T-layered, there exists a UPT $\mathcal{C}$ with shape T of the same size such that $P_{\mathcal{C}} = f_{\mathcal{A}}$.

Proof. Let $\mathcal{C}$ be a UPT circuit. We define a WA over trees as follows. The set of states is the set of addition gates of $\mathcal{C}$. The initial function defined by $\iota(g, x)$ is the label of the edge coming from an input gate with label x to g, and 0 if there is no such edge. The output function defined by $F(g)$ is the label of g if g is an output gate, and 0 otherwise. The transition function is defined as follows: $\Delta(g_1, g_2, g)$ is the label of (the unique) multiplication gate g' using g_1 and g_2 as arguments and g' argument of g. Then $f_{\mathcal{A}} = P_{\mathcal{C}}$.

For the second claim, the definition of layered WA over trees exactly says that the above construction can be reverted. □

Minimisation of weighted automata over trees

The following theorem extends Fliess' theorem.

Theorem 6.15: [7]

Let $f : \mathrm{Tree}(X) \to \mathbb{R}$ be a tree series such that $\mathrm{rank}(H^{(f)})$ is finite.

- Any WA recognising f has size at least $\mathrm{rank}(H^{(f)})$.
- There exists a WA recognising f of size exactly $\mathrm{rank}(H^{(f)})$.

We detail the construction for the second item as we will need it to prove Theorem 6.11.

Recall that the rows of $H^{(f)}$ are indexed by trees in $\mathrm{Tree}(X)$. For $t \in \mathrm{Tree}(X)$, let $H_t^{(f)}$ be the row corresponding to t in $H^{(f)}$, which we see as $H_t^{(f)} \in \mathbb{R}^{\mathrm{Context}(X)}$. Let $Q \subseteq \mathrm{Tree}(X)$ such that $\left\{ H_t^{(f)} \mid t \in Q \right\}$ is a basis of $\mathrm{Span}\left\{ H_t^{(f)} \mid t \in \mathrm{Tree}(X) \right\}$. We furthermore assume that Q contains at least one tree reduced to a single variable $x \in X$, which is possible since $H_x^{(f)} \neq 0$ for some $x \in X$ unless f is the constant zero series.

We now construct the WA recognising f. The set of states is Q. The initial function is defined by $\iota(x, t) = 1$ if t is the variable $x \in X$, and 0 otherwise. We now define the transition function. For $t_1, t_2 \in Q$, there is a unique decomposition of $H_{t_1 \cdot t_2}^{(f)}$ on the basis $\left\{ H_t^{(f)} \mid t \in Q \right\}$:

$$H_{t_1 \cdot t_2}^{(f)} = \sum_{t \in Q} \lambda(t_1, t_2, t) H_t^{(f)},$$

we define $\Delta(t_1, t_2, t) = \lambda(t_1, t_2, t)$. The output function is defined by $F(t) = f(t)$ for $t \in Q$.

Proof of Theorem 6.11

Lemma 6.16

Let P be a homogeneous polynomial of degree d and T a shape with d leaves. The automaton constructed above for recognising P is T-layered.

Proof. Let $\mathcal{A} = (Q, \iota, \Delta, F)$ be the automaton described in the previous subsection.

We define $m : Q \to [T]$ by $m(t) = \widetilde{t}$. To see that indeed $\widetilde{t} \in [T]$, we remark that if $\widetilde{t} \notin [T]$ then $H_t^{(P,T)} = 0$, hence t cannot be in Q. The conditions (1) and (3) are clearly satisfied, so we focus on (2).

For $T' \in [T]$, let $V_{T'}$ denote the vector space spanned by $\left\{ H_t^{(P,T)} \mid \widetilde{t} = T' \right\}$.

We claim that the subspaces $V_{T'}$ for $T' \in [T]$ are in direct sum. It follows from the fact that if $L \in V_{T'}$, then for a context C' such that $C' \circ T' \neq T$, we have $L(C') = 0$.

Let $t_1, t_2 \in Q$. By definition

$$H_{t_1 \cdot t_2}^{(P,T)} = \sum_{t \in Q} \Delta(t_1, t_2, t) H_t^{(P,T)} = \sum_{T' \in [T]} \underbrace{\sum_{t \in Q \mid \widetilde{t} = T'} \Delta(t_1, t_2, t) H_t^{(P,T)}}_{\in V_{T'}}.$$

Since $H_{t_1 \cdot t_2}^{(P,T)} \in V_{\widetilde{t_1} \cdot \widetilde{t_2}}$ and the vector spaces $V_{T'}$ for $T' \in [T]$ are in direct sum, it follows that for $T' \in [T]$ such that $\widetilde{t_1} \cdot \widetilde{t_2} \neq T'$ we have

$$\sum_{t \in Q \mid \widetilde{t} = T'} \Delta(t_1, t_2, t) H_t^{(P,T)} = 0.$$

Since the vectors $\left\{ H_t^{(P,T)} \mid t \in Q \right\}$ are linearly independent, this implies that $\Delta(t_1, t_2, t) = 0$. Thus Property (2) is satisfied. □

We now prove our main result, Theorem 6.11. Let P be a homogeneous polynomial of degree d and T a shape with d leaves. Thanks to the first item of Theorem 6.15 any WA recognising P has size at least $\mathrm{rank}(H^{(P,T)})$, and thanks to the first item of Lemma 6.14 this implies that any UPT circuit with shape T computing P has size at least $\mathrm{rank}(H^{(P,T)})$. Thanks to the second item of Theorem 6.15, there exists a WA recognising P of size $\mathrm{rank}(H^{(P,T)})$, and thanks to Lemma 6.16, it is T-layered. Thanks to the second item of Lemma 6.5, it induces a UPT circuit with shape T computing P of size $\mathrm{rank}(H^{(P,T)})$.

6.3 Applications

In this section, we apply our main theorem to explicit polynomials. The first example illustrates the difference between general UPT circuits and UPT circuits in normal form as studied in Chapter 2, witnessing an exponential gap between the two models. Our second example is the permanent.

An exponential gap between UPT circuits and their normal restrictions

Consider the polynomial $P(x_1, x_2, x_3, x_4) = x_1x_2x_3x_4 + x_3x_4x_1x_2$ and let T be a complete binary tree with 4 leaves. Figure 6.6 shows the smallest UPT circuit with shape T computing P given by the construction of Theorem 6.11. It witnesses an interesting phenomenon: both computations x_1x_2 and x_3x_4 are shared and used twice each. This is captured in the Hankel matrix by observing that two blocks contribute only by one to the rank since the two rows are identical. Informally they correspond to isomorphic subshapes. This circuit is not in the normal form studied in Chapter 2, which does not allow such shared computations.

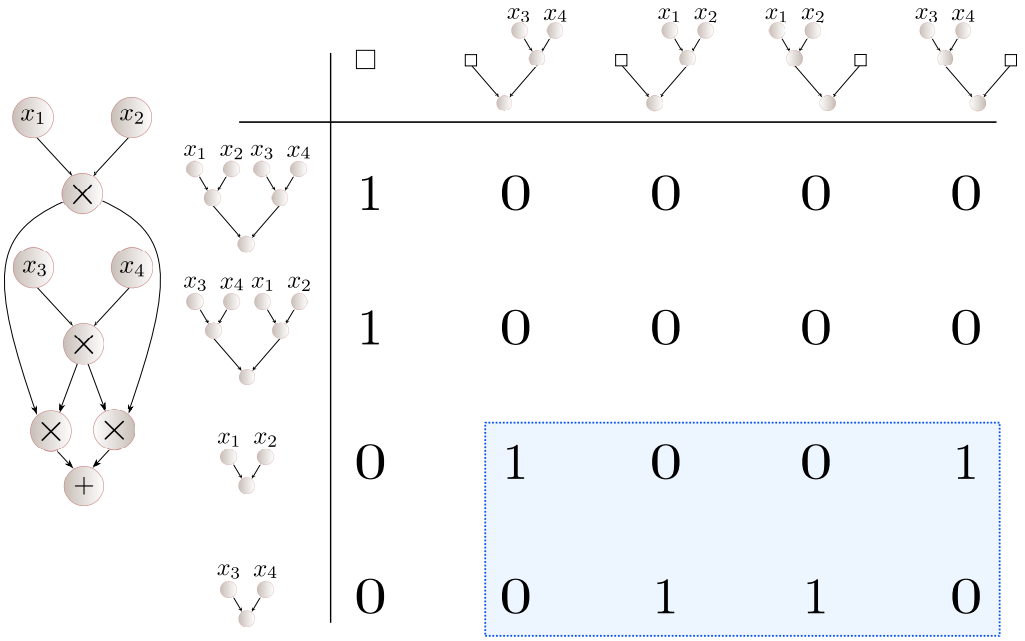


Figure 6.6: On the left hand side, the smallest UPT circuit computing $x_1x_2x_3x_4 + x_3x_4x_1x_2$. We have not depicted some addition gates to keep the figure simple. On the right hand side a sample of the corresponding Hankel matrix. The blue rectangle corresponds to an interaction between two different type of contexts.

We push this further to obtain an exponential gap between UPT circuits and UPT circuits in normal form. Let $n \in \mathbb{N}$ and T be the complete binary tree with 2^n leaves. Consider the polynomial $P(x) = x^{2^n}$. Inspecting the Hankel matrix (see Figure 6.8) yields $\text{rank}(H^{(P,T)}) = n$. Thus thanks to Theorem 6.11 the smallest UPT circuit with shape T computing P has exactly n addition gates, illustrated in Figure 6.7. The characterisation obtained in Chapter 2 shows that the smallest UPT circuit with shape T in normal form has $2^{n+1} - 1$ addition gates, yielding

an exponential gap. Note however that such a large gap can only be obtained for circuits with large degrees.

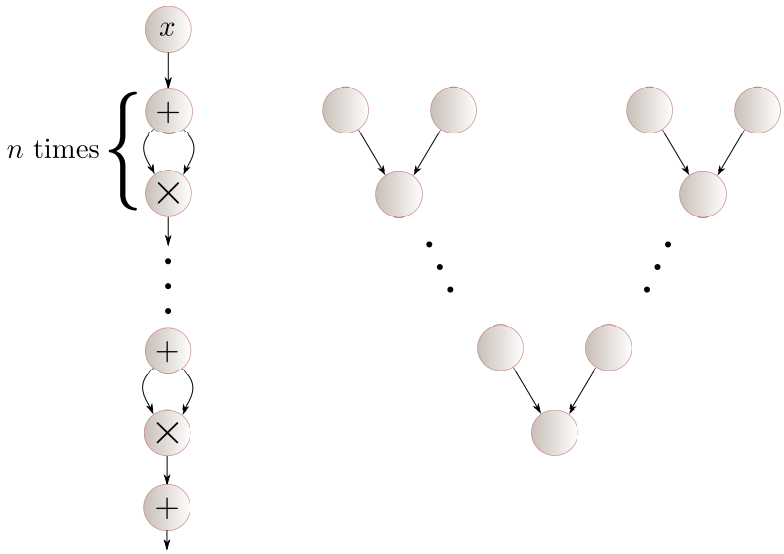


Figure 6.7: An example of a UPT circuit computing x^{2^n}. The circuit is on the left hand side and its shape on the right hand side, it is the complete binary tree of height n.

The permanent

We look at the permanent polynomial

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

over the n^2 variables $X = \{x_{i,j} \mid i,j \in [n]\}$. We examine the Hankel matrix $H^{(P,T)}$ for any shape T with n leaves and obtain the size of the smallest UPT circuit of shape T which computes the permanent.

For v a node of T, let d_v be the number of leaves in T_v.

Lemma 6.17

$$\text{rank}(H^{(P,T)}) = \sum_{v \in T} \binom{n}{d_v}.$$

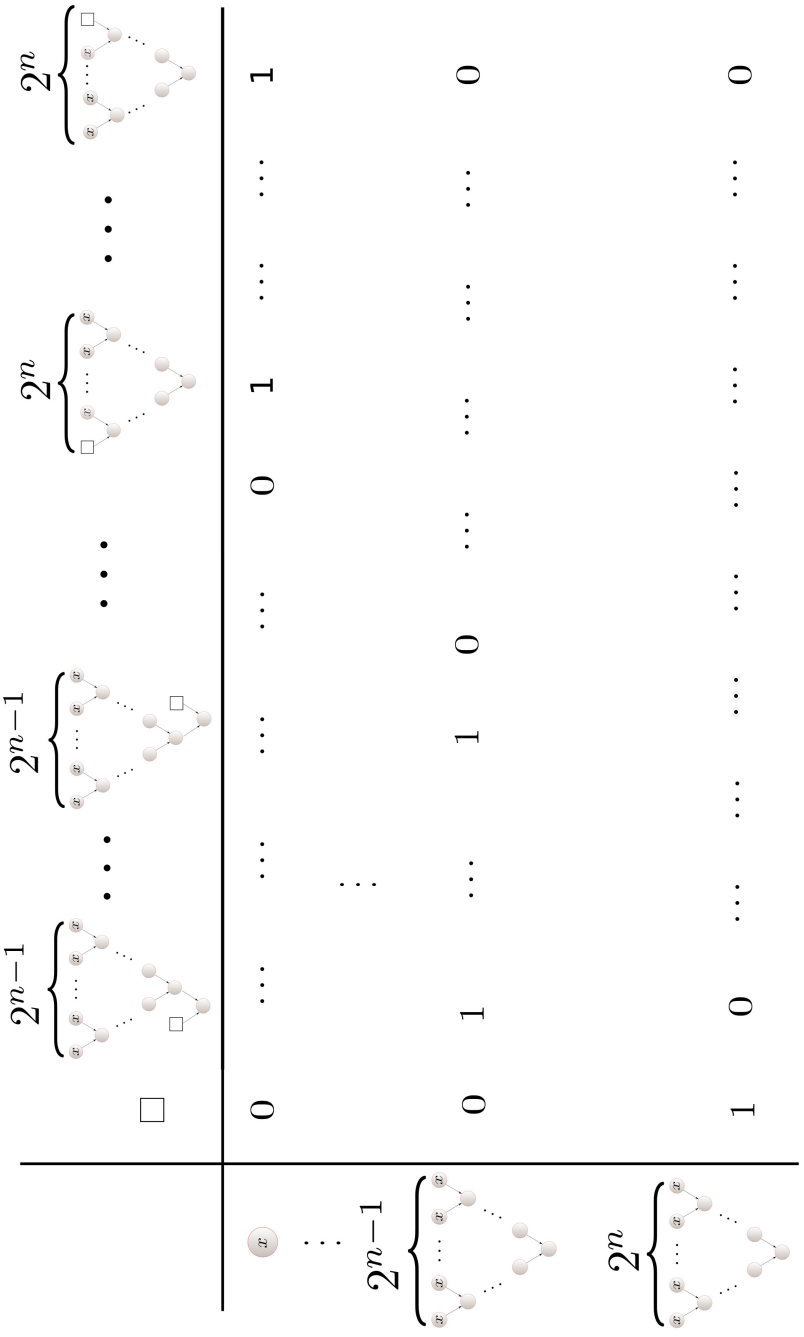


Figure 6.8: The Hankel matrix for the polynomial x^{2^n} and the shape T being the full binary tree.

Proof. Let T be a shape with n leaves $\{\ell_1, \ldots, \ell_n\}$ and $v \in T$ be a node in T. We let i_v denote the leftmost index of a leaf in T_v, *i.e.*, T_v has leaves $\ell_{i_v}, \ell_{i_v+1}, \ldots, \ell_{i_v+d_v-1}$. Moreover, let S be a subset of $\{1, \ldots, n\}$ of size d_v.

We argue that in the Hankel matrix there are $\sum_{v \in T} \binom{n}{d_v}$ independent blocks, and that the set of these blocks is in bijection with pairs (v, S) where v is a node of T and S a subset of size d_v.

Let $S = \{s_1, \ldots, s_{d_v}\}$ and its complement $\{1, \ldots, n\} \setminus S = \{q_1, \ldots, q_{n-d_v}\}$. Let U_v^S be the set of labelings of the leaves $\ell_{i_v}, \ldots, \ell_{i_v+d_v-1}$ of T_v by variables $x_{i_v,\sigma(s_1)}, \ldots, x_{i_v+d_v-1,\sigma(s_{d_v})}$ in this order, ranging over permutations σ of S.

Likewise, we let C denote the unlabeled context obtained by removing T_v from node v in T and replacing it by a placeholder $\Box$, and put B_v^S to be the set of labelings of all leaves but those in T_v of C by the variables

$$x_{1,\tau(q_1)}, \ldots, x_{i_v-1,\tau(q_{i_v-1})}, x_{i_v+d_v,\tau(q_{i_v})}, \ldots, x_{n,\tau(q_{n-d_v})},$$

ranging over permutations τ of the complement of S.

For any σ, τ, the corresponding labeled tree $t_\sigma \in U_v^S$ and labeled context $c_\tau \in B_v^S$ are such that $f_P(c_\tau[t_\sigma]) = H^{(P,T)}(t_\sigma, c_\tau) = 1$, inducing a block of 1's indexed by $U_v^S \times B_v^S$ in $H^{(P,T)}$.

Conversely, we see that any $(t, c) \in \mathrm{Tree}(X) \times \mathrm{Context}(X)$ such that $H^{(P,T)}(t, c) = 1$ is in some $U_v^S \times B_v^S$, and that for any two distinct $(S, v), (S', v')$, both U_v^S and $U_{v'}^{S'}$ and B_v^S and $B_{v'}^{S'}$ are disjoint, hence the blocks cover all 1's in $H^{(P,T)}$ and are independent. This concludes. $\square$

We instantiate this result for two shapes:

- If T is a comb, this yields that the smallest ABP computing the permanent has size $\sum_{i=1}^{n} \binom{n}{i} + \sum_{i=1}^{n} \binom{n}{0} = 2^n + n$,
- If T is a full binary tree of depth $k = \log(n)$, this yields that the smallest UPT circuit with this shape computing the permanent has size $\sum_{i=0}^{k} 2^i \binom{2^k}{2^{k-i}} = \Theta(\frac{2^n}{\sqrt{n}})$.

Hence the latter UPT circuit is more efficient.

However, recall that in our model circuits have unbounded fan-in on addition gates. In this setting a natural and a more accurate estimation of the size of the circuit (or number of operations that are performed) is to count directly the total number of arguments of addition gates. Examining more closely the automata and the circuits we construct, we obtain the following formula that gives the number of such edges for a UPT circuit with shape T computing the permanent

$$\sum_{v \in T} \binom{d_v}{f_v} \binom{n}{d_v},$$

where f_v is the number of leaves in $T_{v'}$, with v' an argument of v (indeed, it does not depend upon which argument is chosen). Note that our optimality result does not apply to this new measure, but we can still consider the size of the circuits constructed by Theorem 6.11. It yields an ABP of size $n2^{n-1} + n$, which asymptotically matches the well known optimal ABP for the permanent that is asymptotically as fast as Ryser formula. For the case of the full binary tree, we obtain a UPT circuit of size $\Theta(2^{\frac{3}{2}n}/\log(n))$, worse than the ABP.

Part II

Lempel-Ziv: a “One-bit catastrophe” but not a tragedy

Overview

This part gives a positive answer to the "one-bit catastrophe" question—introduced by Jack Lutz in the late '90s—that asks whether an infinite word compressible by LZ'78 can become incompressible by adding a single bit in front of it.

In Chapter 7 we introduce all the notions related to LZ'78 and state our main results. Chapter 8 is devoted to the proof of the upper bound (the "not a tragedy" part), whereas the next chapters are about lower bounds. In Chapter 9 we explicitly give a word, based on de Bruijn sequences, whose compression ratio is optimal but the addition of a single bit deteriorates the compression ratio as much as the aforementioned upper bound allows to. That is a particular case of the result of Chapter 10 but we include it anyway for three reasons: it illustrates the main ideas without obscuring them with too many technical details; the construction is more explicit; and the bounds are better.

In Chapter 10 we prove our main theorem on finite words (Theorem 7.10). It requires the existence of a family of "de Bruijn-style" words shown in Section 10.1 thanks to the probabilistic method. Finally, Chapter 11 uses the previous results to prove the "original" one-bit catastrophe, namely on infinite words (Theorem 7.6). The whole part is based on the following publication:

- Guillaume Lagarde and Sylvain Perifel. Lempel-ziv: a "one-bit catastrophe" but not a tragedy. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1478–1495, 2018

Chapter 7

Introduction

Suppose you compressed a file using your favorite compression algorithm, but you realize there was 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 this second part of the thesis, 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 [27], Pierce II and Shields [41], as well as more recently by López-Valdés [31].

Ziv-Lempel algorithms

In the paper [47] where they introduce their second compression algorithm LZ'78, Ziv and Lempel analyse its performance in terms of finite-state lossless compressors and show it achieves the best possible compression ratio. Together with its cousin algorithm LZ'77 [46], this generic lossless compressor has 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 the internet based on Ziv-Lempel algorithms have been proposed [16].

Besides its pratical interest, the algorithm LZ'78 was the starting point of a long line of theoretical research, triggered by the optimality result among finite-state compressors proved by Ziv and Lempel. In recent work, for instance, a

comparison of pushdown finite-state compressors and LZ'78 is made in [37]; the article [20] studies Lempel-Ziv and Lyndon factorisations of words; or the efficient construction of absolutely normal numbers of [34] 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. [45]), or to measure the complexity of a discrete signal [1] for instance.

Actually, both in theory and in practice, Ziv-Lempel algorithms are undoubtedly among the most studied compression algorithms and we have chosen only a very limited set of references: we do not even claim to be exhaustive in the list of fields where LZ'77 or LZ'78 play a role.

Robustness

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 [27], 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 [33] can be defined in terms of compression (see [32]), this question is linked to finite-state and polynomial-time dimensions as [31] 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.

Here, we show the existence of an infinite sequence w which is compressible by LZ'78, but the addition of a single bit in front of it makes it incompressible (the compression ratio of $0w$ is non-zero, see Theorem 7.6), thus we settle the "one-bit catastrophe" question. To that end, we study the question over finite words, which enable stating more precise results. For a word w and a letter a, we first prove in Theorem 7.7 that the compression ratio $\rho(aw)$ of aw cannot deviate too much from the compression ratio $\rho(w)$ of w:

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

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

However, our results are tight up to a constant factor, as we show in Theorem 7.10: 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

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

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

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

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

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

that is to say that the compression ratio of $0w$ is much worse than that of w (which in that case is optimal). To give a concrete idea, the bounds given by our Theorem 9.1 for words of size 1 billion ($|w| = 10^9$) yield a compression for w of size at most $d \log d \leq 960{,}000$ (where $d = 1.9\sqrt{|w|}$), whereas for $0w$ the compression size is at least $d' \log d' \geq 3{,}800{,}000$ (where $d' = 0.039|w|^{3/4}$).[1]

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.

Lempel-Ziv, compression and results

Before turning to the description of LZ'78 algorithm, let us recall standard notation on words.

7.1 Basic notation

The *binary alphabet* is the set $\{0, 1\}$. A *word* w is an element of $\{0, 1\}^\star$, that is, a finite ordered sequence of letters 0 or 1, whose *length* is denoted by $|w|$. The

[1] Actually, throughout the manuscript we preferred readability over optimality and thus did not try to get the best possible constants; simulations show that there is a lot of room for improvement, since already for small words the difference is significant (using notation introduced in Section 7 and Chapter 9, for $w = \mathrm{Pref}(x)$ with $x \in \mathrm{DB}(12)$, $|w| \simeq 8.10^6$ and w is parsed in about 4100 blocks, whereas $0w$ is parsed in more than 200,000 blocks).

empty word is denoted by λ. For a word $w = x_0 \cdots x_{n-1}$ (note that the indices begin at zero), where $x_i \in \{0, 1\}$, $w[i..j]$ will denote the *substring* $x_i \cdots x_j$ of w (or λ if $j < i$); $w[i]$ or w_i will denote the letter x_i; and $w_{\leq i}$ (respectively $w_{<i}$) will denote $w[0..i]$ (resp. $w[0..i-1]$). We say that a word m is a *factor* of w if m is any substring $w[i..j]$. In the particular case of $i = 0$ (respectively $j = n-1$), m is also called a *prefix* (resp. a *suffix*) of w. The set of factors of w is denoted by $\mathscr{F}(w)$, and its set of prefixes $\mathscr{P}(w)$. By extension, for a set M of words, $\mathscr{F}(M)$ will denote $\cup_{w \in M} \mathscr{F}(w)$ and similarly for $\mathscr{P}(M)$. If u and w are two words, we denote by $\mathrm{Occ}_w(u)$ the number of occurrences of the factor u in w.

The "length-lexicographic order" on words is the lexicographic order where lengths are compared first.

An *infinite word* is an element of $\{0, 1\}^{\mathbb{N}}$. The same notation as for finite words apply.

All logarithms will be in base 2. The size of a finite set A is written $|A|$.

7.2 LZ'78

7.2.1 Notions relative to LZ

A *k-partition* (or just *partition*) of a word w is a sequence of k non-empty words $m_1, \ldots, m_k$ such that $w = m_1.m_2.\cdots.m_k$. The *LZ-parsing* (or just *parsing*) of a word w is the unique partition of $w = m_1 \cdots m_k$ such that:

- $m_1, \ldots, m_{k-1}$ are all distinct[2];
- $\forall i \leq k$, $\mathscr{P}(m_i) \subseteq \{m_1, \ldots, m_i\}$.

The words $m_1, \ldots, m_k$ are called *blocks*. The *predecessor* of a block m_i is the unique m_j, $j < i$, such that $m_i = m_j a$ for a letter a. The compression algorithm LZ'78 parses the word w and encodes each block m_i as a pointer to its predecessor m_j together with the letter a such that $m_i = m_j a$. For instance, the word $w = 00010110100001$ is parsed as

Blocks	0	00	1	01	10	100	001
Block number	0	1	2	3	4	5	6

and thus encoded as

$$(\lambda, 0); (0, 0); (\lambda, 1); (0, 1); (2, 0); (4, 0); (1, 1).$$

[2]The last word m_k might be equal to another m_i.

The *dictionary* of w is the set $\mathrm{Dic}(w) = \{m_1, \ldots, m_k\}$ (in the example, $\{0, 1, 00, 01, 10, 001, 100\}$). Remark that, by definition, $\{\lambda\} \cup \mathrm{Dic}(w)$ is prefix-closed.

The *parse tree* of w is the unique rooted binary tree whose $(k+1)$ vertices are labeled with $\lambda, m_1, \ldots, m_k$, such that the root is λ and if a vertex m_i has a left child, then it is $m_i 0$, and if it has a right child, then it is $m_i 1$.[3] See Figure 7.1. Remark also that the depth of a vertex is equal to the size of the corresponding block.

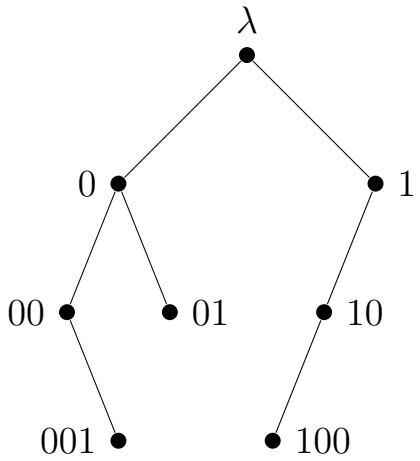


Figure 7.1: Parse tree of 00010110100001.

By abuse of language, we say that a block b "increases" or "grows" in the parsing of a word w when we consider one of its successors, or when we consider a path from the root to the leaves that goes through b. Indeed, going from b to its successor amounts to add a letter at the end of b (hence the "increase").

7.2.2 Compression ratio

As in the example above, given a word w and its LZ-parsing $m_1 \cdots m_k$, the *LZ-compression* of w 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 $m_j = m_i[0..(|m_i| - 2)]$, and a_i the last letter of m_i (that is, the unique letter such that $m_i = m_j a_i$). When the LZ-compression is given, one can easily reconstruct the word w.

[3] Note that, in order to recover the parsing from the parse tree, the vertices must also be labeled by the order of apparition of each block, but we do not need that in the sequel.

Remark 7.1

- If x is a word, we define $\mathrm{Pref}(x)$ the concatenation of all its prefixes in ascending order, that is,

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

  Then the parsing of the word $w = \mathrm{Pref}(x)$ 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

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

  Actually, it is easy to see that this optimal compression is attained only for the words w of the form $\mathrm{Pref}(x)$.

  In Chapter 10 we will need the concatenation of all prefixes of x starting from a size $p+1$, denoted by $\mathrm{Pref}_{>p}(x)$, that is,

$$\mathrm{Pref}_{>p}(x) = x_0x_1\cdots x_p.x_0x_1\cdots x_{p+1}.\cdots .x_0\cdots x_{n-1}.$$

- On the other hand, if w is the concatenation, in length-lexicographic order, of all words of size $\leq n$ ($w = 0.1.00.01.10.11.000.001\ldots$), then it has size

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

  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

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

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

The number of bits needed in the LZ-compression is $\Theta(\sum_{i=1}^{k}(|p_i|+1)) = \Theta(k\log k)$. As the two previous extremal cases show, $k\log k = \Omega(\sqrt{|w|}\log|w|)$ and $k\log k = O(|w|)$.

Definition 7.2

The *compression ratio* of a word w is

$$\rho(w) = \frac{|\mathrm{Dic}(w)| \log |\mathrm{Dic}(w)|}{|w|}.$$

As Remark 7.1 shows,

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

A sequence of words (w_n) is said LZ-compressible if $\rho(w_n)$ tends to zero (i.e., $k_n \log k_n = o(|w_n|)$), and consistently it will be considered *LZ-incompressible* if $\liminf_{n\to\infty} \rho(w_n) > 0$ (in other terms, $k_n \log k_n = \Omega(|w_n|)$).

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

Definition 7.3

The *size of the LZ-compression* of w, or *compression size*, is defined as the size of $\mathrm{Dic}(w)$, that is, the number of blocks in the LZ-parsing of w.

Remark that $|\mathrm{Dic}(w)| = \Omega(\sqrt{|w|})$ and $|\mathrm{Dic}(w)| = O(|w|/\log(|w|))$. We can now restate the definition of incompressibility of a sequence of words in terms of compression size instead of the number of bits in the LZ-compression.

Definition 7.4

A sequence of words (w_n) is said to be *incompressible* iff

$$|\mathrm{Dic}(w_n)| = \Theta\left(\frac{|w_n|}{\log(|w_n|)}\right).$$

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 $\liminf$ or $\limsup$ of the compression ratios of the prefixes.

Definition 7.5: compression ratios for infinite words

Let $w \in \{0,1\}^{\mathbb{N}}$ be an infinite word.

$$\rho_{\inf}(w) = \liminf_{n\to\infty} \rho(w_{<n}) \quad \text{and} \quad \rho_{\sup}(w) = \limsup_{n\to\infty} \rho(w_{<n}).$$

The word w is called *incompressible* if $\rho_{\inf}(w) > 0$.

7.3 One-bit catastrophe and results

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). In Chapter 11 we will answer that question positively:

Theorem 7.6

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

$$\rho_{\sup}(w) = 0 \quad \text{and} \quad \rho_{\inf}(0w) \geq \frac{1}{6\,075}.$$

Remark that the $\liminf$ is considered for the compression ratio of $0w$ and the $\limsup$ for w, which is the hardest possible combination as far as asymptotic compression ratios are concerned.

But before proving this result, most of the work will be on finite words (only in Chapter 11 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 Chapter 8, we show that the compression ratio of aw cannot be much worse than that of w. In particular, all words "sufficiently" compressible (compression size $o(|w|/\log^2|w|)$) 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 7.11).

Theorem 7.7

For every word $w \in \{0,1\}^\star$ and any letter $a \in \{0,1\}$,

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

Remark 7.8

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

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

We also show in Chapter 9 that this result is tight up to a multiplicative constant, since Theorem 9.1 implies the following result.

Theorem 7.9

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

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

More generally, we prove in Chapter 10 our main result:

Theorem 7.10

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:

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

This shows that the upper bound is tight (up to a multiplicative constant) for any possible compression size. This also provides an example of compressible words that become incompressible when a letter is added in front (see Remark 7.11), thus showing the one-bit catastrophe for finite words.

Remark 7.11

In particular:

- Theorem 7.7 implies that, if an increasing sequence of words (w_n) satisfies $|\mathrm{Dic}(w_n)| = o(|w_n|/\log^2|w_n|)$, then for any letter $a \in \{0,1\}$, aw_n remains fully compressible ($|\mathrm{Dic}(aw_n)| = o(|w_n|/\log|w_n|)$);
- however, by Theorem 7.10, there is an increasing sequence of words (w_n) such that $|\mathrm{Dic}(w_n)| = \Theta(|w_n|/\log^2|w_n|)$ (compressible) but $|\mathrm{Dic}(0w_n)| = \Theta(|w_n|/\log|w_n|)$ (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 $|\mathrm{Dic}(w_n)| = \Theta(\sqrt{|w_n|})$ (optimal compression) but $|\mathrm{Dic}(0w_n)| = \Theta(|w_n|^{3/4})$. This special case is treated extensively in Theorem 9.1. More generally, we can get $|\mathrm{Dic}(w_n)| = \Theta(|w_n|^{\alpha})$ and $|\mathrm{Dic}(0w_n)| = \Theta(|w_n|^{(1+\alpha)/2})$ for any $\alpha \in [1/2, 1]$.

7.4 Parsings of w and aw

We will often compare the parsing of a word w and the parsing of aw for some letter a: let us introduce some notation (see Figure 7.2).

- 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[4]:
  - 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 7.2).
  - 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.

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

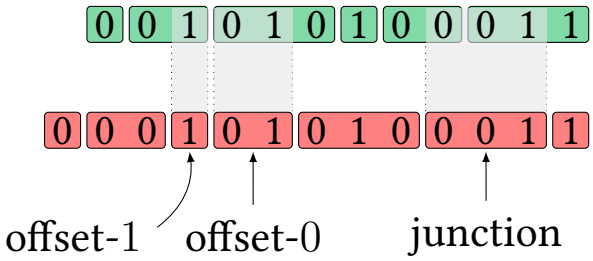


Figure 7.2: The green blocks of w and red blocks of $0w$ for $w = 001010100011$.

Chapter 8

Upper bound

This chapter is devoted to the proof of Theorem 7.7 giving an upper bound on the compression ratio of aw, for any letter a, as a function of the compression ratio of the word w. In their 1998 paper [27], Lathrop and Strauss ask the following question: "Consider optimally compressed sequences: Will such sequences compress reasonably well if a single bit is removed or added to the front of the sequence?" We give a positive and quantified answer: indeed, a word w compressed optimally has a compression size $O(\sqrt{n})$, thus by Theorem 7.7, the word aw has a compression size $O(n^{3/4})$. (And we shall complete this answer with the matching lower bound in the next chapter.)

The first lemma bounds the size of the partition of a word w if the partitioning words come from a family with a limited number of words of same size. In its application, the partition will be a subset of the LZ-parsing, and Lemma 8.3 below will give the required bound on the number of factors of a given size.

Lemma 8.1

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 different words of $\mathcal{F}$. Then the number of words used in the partition is at most $2\sqrt{N|w|}$.

Proof. Let $m(i)$ be the number of words of size i occurring in the partition of w, and k the size of the largest words used. We want to prove that

$$\sum_{i=1}^{k} m(i) \leq 2\sqrt{N|w|}.$$

We have:

$$|w| = \sum_{i=1}^{k} im(i) \geq \sum_{i \geq \sqrt{\frac{|w|}{N}}} im(i) \geq \sqrt{\frac{|w|}{N}} \sum_{i \geq \sqrt{\frac{|w|}{N}}} m(i)$$

hence

$$\sum_{i \geq \sqrt{\frac{|w|}{N}}} m(i) \leq \sqrt{N|w|}.$$

On the other hand, since $m(i) \leq N$:

$$\sum_{i < \sqrt{\frac{|w|}{N}}} m(i) < N\sqrt{\frac{|w|}{N}} = \sqrt{N|w|}.$$

□

Remark 8.2

Note that if, for all $i \geq 1$, $\mathcal{F}$ contains exactly $\min(2^i, N)$ words of size i, the concatenation of all the words of $\mathcal{F}$ up to size s gives a word w of size

$$|w| = \sum_{i=1}^{\log N} i2^i + \sum_{i > \log N}^{s} iN \leq 2N \log N + (s - \log N)(s + \log N + 1)N/2$$

partitioned into m blocks, where

$$m = \sum_{i=1}^{\log N} 2^i + \sum_{i > \log N}^{s} N \geq (s - \log N)N.$$

Thus $m \geq \sqrt{2}\sqrt{N|w|}$ if $s >> \log N$. This shows the optimality of Lemma 8.1 up to a factor $\sqrt{2}$.

We now come to the lemma bounding the number of factors of a given size in a word w as a function of its LZ-parsing.

Lemma 8.3

Let T be the parse tree of a word w. Then the number of different factors of size i in the blocks of w is at most $|T| - i$ (that is, $|\mathscr{F}(\mathrm{Dic}(w)) \cap \{0,1\}^i| \leq |T| - i$).

Proof. A factor of size i in a block b corresponds to a subpath of size i in the path from the root to b in the parse tree. The number of such subpaths is bounded by the number of vertices at depth at least i. □

Actually, below we will use Lemma 8.3 sub-optimally since we will ignore the parameter i and use the looser bound $(|T| - 1)$.

Let us turn to the proof of Theorem 7.7, the main result of the present chapter.

Proof of Theorem 7.7. Let T be the parse tree of w and $D = \mathrm{Dic}(aw)$ 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 for D_1: The number of junction blocks is less than the number of green blocks, therefore $|D_1| \leq |\mathrm{Dic}(w)| \leq \sqrt{|\mathrm{Dic}(w)|.|w|}$ (recall that $|\mathrm{Dic}(w)| \leq |w|$).
- Bound for D_2: Consider $\tilde{w}$ the word w where all the junction blocks have been replaced by the empty word λ. We know that $\tilde{w}$ is partitioned into different words by D_2. But $D_2 \subset \mathcal{F}$, where $\mathcal{F} = \mathscr{F}(\mathrm{Dic}(w))$ (the set of factors contained in the green blocks). By Lemma 8.3, the number of words of size i in $\mathcal{F}$ is bounded by $|T| - i$, which is at most $|\mathrm{Dic}(w)|$. Finally, Lemma 8.1 tells us that the number of words in any partition of $\tilde{w}$ by words of $\mathcal{F}$ is bounded by $2\sqrt{|\mathrm{Dic}(w)|.|\tilde{w}|} \leq 2\sqrt{|\mathrm{Dic}(w)|.|w|}$.

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

Remark 8.4

Instead of a single letter, we can add a whole word z in front of w. With the same proof, it is easy to see that

$$|\mathrm{Dic}(zw)| \leq |\mathrm{Dic}(z)| + 3\sqrt{|w|.|\mathrm{Dic}(w)|}.$$

Alternately, if we remove the first letter of $w = aw'$ (or any prefix) we get the same upper bound:

$$|\mathrm{Dic}(w')| \leq 3\sqrt{|aw'|.|\mathrm{Dic}(aw')|}.$$

Chapter 9

"Weak catastrophe" for the optimal compression ratio

Before the proof of Theorem 7.10, we first present a "weak catastrophe", namely the third item of Remark 7.11 in which the compression size 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 7.7.

Theorem 9.1

For an infinite number of words w:

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

Remark 9.2

The "true" values of the constants that we will get below are as follows:

$$|\mathrm{Dic}(w)| \leq 3\sqrt{\frac{2}{5}}\sqrt{|w|} \text{ and } |\mathrm{Dic}(0w)| \geq \frac{1}{36}\left(\frac{8}{5}\right)^{3/4}|w|^{3/4} - o(|w|^{3/4}).$$

Observe that this weak catastrophe is a special case of Theorem 7.10 (with better constants, though). The aim of this chapter is twofold: first, it will be a constructive proof, whereas the main theorem will use the probabilistic method; second, this chapter 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.

9.1 De Bruijn sequences

A *de Bruijn sequence* of order k (or $\mathrm{DB}(k)$ in 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 circuits 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.

(In symbols, $\mathrm{Occ}_x(u) = 2^{k-|u|}$.) 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 begining 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 [28], as well as later [27] and [41] for example.

9.2 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 7.1). Thus we are looking for an x such that $0\mathrm{Pref}(x)$ has the worst possible compression ratio. In Chapter 8 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 dictionnary 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 $i \leq \gamma k$) 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 (called i-violation in the sequel), 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, Lemma 9.5 shows that at most half of the prefixes of x can contain offset-i blocks for any fixed $i \leq \gamma k$. This is due to the insertion of gadgets that "kill" some starting positions i if necessary, by "resynchronizing" the parsing at a different position.

- On the other hand, red blocks starting at position $i > \gamma k$ are shown to be of small size by Proposition 9.7. This is implied by Lemma 9.6 claiming that, due to the structure of the $\mathrm{DB}(k)$ (few repetitions of factors), 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 size is $\Omega(n^{3/4})$ (Theorem 9.1).

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 adaptive, which is the reason why we give an algorithm to describe the word w.

Let us summarize the organisation of the lemmas of this chapter:

- Lemma 9.3 is necessary for the algorithm: it shows that, in $0\mathrm{Pref}(x)$, there can be at most one position i such that the number of i-violations is too high.

- Lemma 9.4 shows that the parsing of w is not disturbed by gadgets and therefore the compression size of w is $O(\sqrt{n})$.

- Lemma 9.5 shows that gadgets indeed remove i-violations as required, for $i \leq \gamma k$.

- Lemma 9.6 uses the property of the $\mathrm{DB}(k)$ to prove that junction blocks cannot create too many i-violations if $i > \gamma k$.

- Finally, Proposition 9.7 uses Lemma 9.6 to show that the offset-i red blocks are small if i is large.

9.3 Construction and first properties

Let γ be any constant greater than or equal to 3. Let x be a $\mathrm{DB}(k)$ beginning by 01. We denote its size by $s = 2^k + k - 1$. Suppose for convenience that k is odd, so that s is even.[1] For $i \in [0, s-1]$, let $w_i = x_{\leq i}$, so that $\mathrm{Pref}(x) = w_0.w_1 \dots w_{s-1}$.

The word w that we will construct is best described by an algorithm. It will merely be $\mathrm{Pref}(x)$ in which we possibly add "gadgets" (words) between some of the w_j in order to control the parsings of w and $0w$. The letter in front that will provoke the "catastrophe" is the first letter of w, that is, 0.

The gadgets g_i^j (for $i \in [0, \gamma k]$ and $j \geq 0$) are defined as follows (where $\bar{x}_i$ denotes the complement of x_i):

- $g_0^j = 10^j$;
- and for $i > 0$, $g_i^j = x_{<i}.\bar{x}_i.1^j$.

Recall that the green blocks are those of the parsing of w, whereas the red ones are those of the parsing of $0w$. We call "regular" the green blocks that are not gadgets (they are of the form w_j for some j). For $i \in [0, s-1]$, we say that a regular green block in w is i-violated if there is an offset-i (red) block in it. Note that gadgets do not count in the definition of a violation.

Lemma 9.3

For $i \in [0, s-1]$, let l_i be the number of i-violated blocks in $\mathrm{Pref}(x) = w_0.w_1 \dots w_{s-1}$. Then for all $i \neq i'$, $l_i + l_{i'} \leq s$.

In particular, there can be at most one i such that the number of i-violated blocks is $> s/2$.

Proof. Let i and i' be such that $0 \leq i < i' < s$.

Consider the red blocks starting at position i and i' in any green block.

No green block in $w_0 \dots w_{i'-1}$ is i'-violated since they are too small to contain position i'. Let a be the number of i-violated blocks in $w_0 \dots w_{i'-1}$. In $w_{i'} \dots w_{s-1}$, let b be the number of green blocks that are both i-violated and i'-violated, and let c (respectively d) be the number of i-violated (resp. i'-violated) blocks that are not i'-violated (resp. i-violated) blocks.

[1] This is to avoid dealing with the fractional part of $s/2$, but the construction also works in the case where k is even.

The number of i-violations is $l_i = a + b + c$ and the number of i'-violations is $l_{i'} = b + d$. But $b + c + d \leq s - i'$ and $b \leq i' - i - a$ (since a red block starting at position i can only be increased $(i' - i)$ times before it overlaps position i', and it has already increased a times in the first i' green blocks), so that $l_i + l_{i'} = (b + c + d) + (a + b) \leq (s - i') + (i' - i) \leq s$. Therefore, l_i or $l_{i'}$ has to be $\leq s/2$. □

The algorithm constructing w, illustrated in Figure 9.1, is as follows.

1. If the number of i-violations in $w_0.w_1 \dots w_{s-1}$ is $\leq s/2$ for all $i \in [0, \gamma k]$, then output $w = w_0.w_1 \dots w_{s-1}$.

2. Otherwise, let i be the (unique by Lemma 9.3) integer in $[0, \gamma k]$ for which the number of i-violations is $> s/2$. Let $c = 0$ (counter for the number of inserted gadgets) and $d = s/2 + 1$ (counter for the place of the gadget to be inserted).

3. For all $j \in [0, s - 1]$, let $z_j = w_j$.

4. While the number of i-violations in $z_0.z_1 \dots z_{s-1}$ is $\geq d$, do:

   (a) let j be such that w_j is the d-th i-violated green block;

   (b) $z_j \leftarrow g_i^c w_j$ (we add the gadget g_i^c before the block w_j);

   (c) $c \leftarrow c + 1$;

   (d) if w_j is still i-violated, then $d \leftarrow d + 1$.

5. Return $w = z_0.z_1 \dots z_{s-1}$.

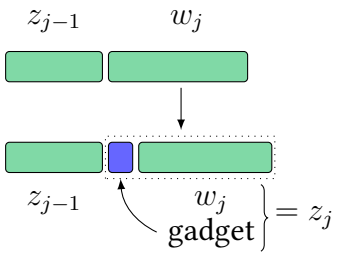


Figure 9.1: Illustration of Step 4(b) of the algorithm.

Some parts of the algorithm might seem obscure, in particular the role of the counter d. The proof of the following properties should help understand this construction, but let us first explain the intuition behind the algorithm. Below (Proposition 9.7) we will have a generic argument (i.e., true without gadgets)

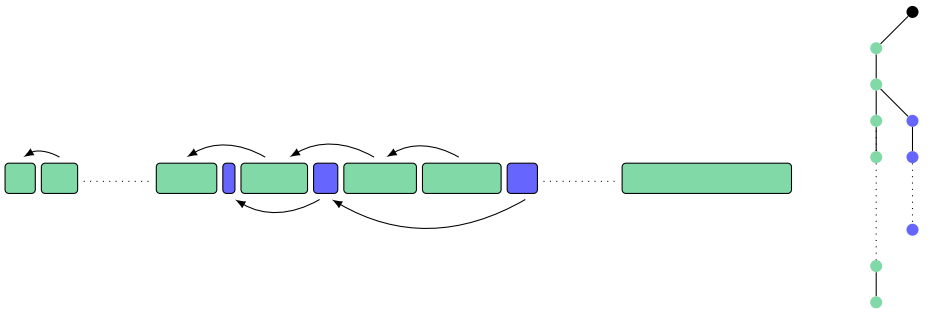


Figure 9.2: Left: Form of the word w. The blocks in green are the regular blocks, the blocks in blue are the gadgets. The arcs represent the relation of paternity. Right: The shape of the parse tree of w.

to deal with the i-violations for $i > \gamma k$, therefore for now we only care about i-violations for $i \leq \gamma k$. They are not problematic if there are at most (roughly) $s/2$ of them. Thanks to Lemma 9.3, there is therefore at most one i_0 which can be problematic. To guarantee the upper bound of (roughly) $s/2$ for the number of i_0-violations, every time it is necessary we insert between two regular green blocks one gadget to kill the $(s/2+1)$-th, $(s/2+2)$-th, etc., i_0-violations. But gadgets are guaranteed to work as expected only if at least $1+(\gamma+1)k$ of them have already been inserted (see Lemma 9.5), hence the counter d is useful to avoid inserting two gadgets in front of the same regular block.

From now on, we call w the word output by the algorithm. We first evaluate the size of w. Its minimal size is obtained when no gadgets are added during the algorithm:

$$|w| \geq \frac{s(s+1)}{2}.$$

On the other hand, if $s/2$ gadgets $g_{\gamma k}^c$ of size $\gamma k + 1 + c$ are added, we obtain an upper bound on $|w|$:

$$|w| \leq \frac{s(s+1)}{2} + \sum_{c=0}^{s/2-1} (\gamma k + 1 + c) = \frac{5s^2}{8} + o(s^2).$$

Let us show that the word w is nearly optimally compressible (upper bound).

Lemma 9.4

The compression size of w is at most

$$3\sqrt{\frac{2}{5}}\sqrt{|w|}.$$

Proof. If the algorithm stops at step 1, then $w = \mathrm{Pref}(x)$ and it is compressed optimally (see Remark 7.1): the compression size is

$$\sqrt{2}\sqrt{|w|} + O(1).$$

Otherwise, we add at most one gadget for each w_j, and only for $j > s/2$. Therefore, there are at most $s/2$ gadgets. Remark that, for the i fixed in the algorithm, the gadgets $(g_i^j)_j$ are prefixes one of each other, and none of them are prefixes of x. Thus the parse tree of w consists of one main path of size s (corresponding to $w_0, w_1, \ldots, w_{s-1}$), together with another path of size $\leq s/2$ (corresponding to the gadgets $(g_i^j)_j$) starting from a vertex of the main path. See Figure 9.2.

The worst case for the compression size is when the second path is of size $s/2$ and starts at the root. Then the size of w is

$$|w| \geq \frac{s(s+1)}{2} + \frac{(s/2)(1+s/2)}{2} \geq \frac{5s^2}{8}$$

and the size of the dictionary is $3s/2$, yielding the compression size stated in the lemma. □

Let us now turn to the lower bound on the compression size of $0w$. The next lemma shows that, for $i \leq \gamma k$, there are not too many i-violations thanks to the gadgets.

Lemma 9.5

For all $i \in [0, \gamma k]$, the number of i-violations in w is at most

$$s/2 + (1+\gamma)k + 1.$$

Proof. If no gadgets have been added during the algorithm, then for all $i \in [0, \gamma k]$, the number of i-violations in w is $\leq s/2$.

Otherwise, first remark that Lemma 9.3 remains valid even when the gadgets are added. We need to distinguish on the type ($i = 0$ or $i > 0$) of the most frequent violations in w.

- Case 1: the most frequent violations are 0-violations. In that case, we claim that whenever a gadget is inserted before a block w_i, the 0-violation in w_i disappears. It is enough to prove that whenever a gadget g_0^j is added, it was already in the dictionary of $0w$, so that the next word in the dictionary will begin by $g_0^j 0$ and the parsing will overlap position 0 of the next green block.

  We proceed by induction: for $j = 0$, $g_0^0 = 1$, and this word is the third block in the parsing of $0w$, because x starts with 01. For $j > 0$: when g_0^{j-1} was parsed, by induction it was already in the dictionary, so that the block added in the dictionary of $0w$ starts with $g_0^{j-1}0 = g_0^j$.

  After at most $s/2$ iterations of the while loop, there is no more $(s/2+1)$-th 0-violation: the number of 0-violations is exactly $s/2$. Observe that violations for $i > 0$ have been created, but by Lemma 9.3, for each $i > 0$, the number of i-violations remains $\leq s/2$.

- Case 2: the most frequent violations are i-violations for some $i > 0$. In that case, the first few times when a gadget is inserted, it may fail to kill the corresponding i-violation. But we claim that the number of such fails cannot be larger than $(\gamma+1)k+1$ (equivalently, in the algorithm the counter d remains $\leq s/2 + (\gamma+1)k + 2$).

  Indeed, since we add a gadget only before an i-violation, the parsing splits the gadget $g_i^j = x_{<i}\bar{x}_i.1^j$ between $x_{<i}$ and $\bar{x}_i.1^j$. Furthermore, by induction, $\bar{x}_i.1^j$ is not split by the parsing. But for the gadget g_i^{k+1}, $\bar{x}_i.1^{k+1}$ is parsed in exactly one block because this factor does not appear anywhere in w before g_i^{k+1}. From that moment on, each i-violation creates through the gadget a 0-violation. The number of blocks that are both 0-violated and i-violated is at most i (due to the growth of the block at position 0). Thus, at most i more gadgets may fail to kill position i. The total number of "failing" gadgets is $\leq k+1+i \leq (\gamma+1)k+1$.

□

9.4 The weak catastrophe

This chapter is devoted to the proof of the lower bound: the compression size of $0w$ is $\Omega(|w|^{3/4})$. Thanks to Property $(\star)$ from page 130, Lemma 9.6 below bounds the number of junction blocks ending at a fixed position $(i-1)$ by a decreasing function of i. The proof is quite technical and requires to distinguish three categories among (red) junction blocks:

- Type 1: junctions over consecutive factors w_a and w_{a+1} (no gadget between two regular green blocks);

- Type 2: junctions starting in a gadget $g_j^{j'}$ and ending in the following regular green block;
- Type 3: junctions starting in a regular green block and ending in the following gadget $g_j^{j'}$.

Lemma 9.6

Let $i \geq 2k+3$. Let uu' be a junction block of type 1 over $w_a w_{a+1}$ ending at position $i-1$ in w_{a+1}, with u being the suffix of w_a and u' the prefix of w_{a+1}. Then $|u| \leq k - \log(i-2k-1)$.

In particular, the number of such blocks is upper bounded by the number of words of size $\leq k - \log(i-2k-1)$, that is, $\frac{2^{k+1}}{i-2k-1}$.

Proof. Let v be the prefix of size $2k$ of u' (which is also the prefix of x). All the prefixes of uu' of size $\geq |uv|$ have to be in the dictionary of $0w$: we call M the set of these prefixes ($|M| = i-2k$). We claim that these blocks are junction blocks of type 1 or 3 only (except possibly for one of type 2), with only u on the left side of the junction. Indeed, let us review all the possibilities:

1. uv cannot be completely included in a regular block, otherwise $v[0..k-1]$ would appear both at positions 0 and $p > 0$ in x, which contradicts Property $(\star)$ (page 130);
2. uv cannot be completely included in a gadget:
   - if the gadget is $g_0^j = 10^j$, impossible because v cannot have more than k zeroes since it is a factor of x,
   - if the gadget is $g_b^j = x_{<b}.\bar{x}_b.1^j$, by the red parsing of gadgets, either uv is in $x_{<b}$ (impossible because v would appear at a position $\geq |u|$ in x), or uv is in $\bar{x}_b.1^j$ (impossible because v cannot contain more than k ones);
3. if uv is a type 1 junction but not split between u and v, it is impossible because the three possible cases lead to a contradiction:
   - if u goes on the right, then v would appear at another position $p > 0$ in x,
   - if v goes on the left by at least k, then $v[0..k-1]$ would again appear at two different positions in x,

- if v goes on the left by less than k, then it goes on the right by more than k and $v[k..2k-1]$ would again appear at two different positions in x;

4. if uv is a type 2 junction but not split between u and v, it is again impossible:
   - if u goes on the right, then v would appear at another position $p > 0$ in x,
   - if v goes on the left by at least 2, then $v[0..1]$ would be either 00 or 11 (depending on the gadget), but we know it is $x_0x_1 = 01$,
   - otherwise, v goes on the right by $2k-1$, and $v[1..k]$ would appear at positions 0 and 1 in x;
5. if uv is a type 3 junction, first remark that the gadget is of the form g_b^j for $b > 0$ (because, for gadgets of the form g_0^j, the red parsing starts at position 0 of the gadget). If uv is not split between u and v, it is once again impossible:
   - if u goes on the right, the red parsing of the gadget stops after $x_{<b}$ and v would appear in x at a non-zero position,
   - similarly, if v goes on the left by less than k, then $v[k..2k-1]$ would again appear at two different positions in x,
   - if v goes on the left by at least k, then $v[0..k-1]$ would again appear at two different positions in x.

Remark finally that all parsings of type 2 junctions have different sizes on the left. Therefore, at most one can contain u on the left. The claim is proved.

Thus, at least $|M|-1$ regular green blocks have u as suffix. Remark that, since $|M| \geq 3$, there are at least two such green blocks, therefore $|u| \leq k$. Hence by Property $(\star)$ (page 130) we have:

$$\begin{aligned} |M| - 1 &\leq 2^{k-|u|} \\ i - 2k - 1 &\leq 2^{k-|u|} \\ |u| &\leq k - \log(i-2k-1). \end{aligned}$$

□

As a consequence, in the next proposition we can bound the size of offset-i blocks. Along with the role of gadgets, this will be a key argument in the proof of Theorem 9.1. The idea is the following: for a red block u starting at a sufficiently large position i, roughly $|u|$ other red blocks have to end at position $(i-1)$, and in the red parsing $\Omega(|u|^2)$ prefixes of these blocks must appear in different green blocks (and in the dictionary), giving the bound $s = \Omega(|u|^2)$.

Proposition 9.7

For any $i > \gamma k$, the size of an offset-i block included in a regular green block is at most

$$2\sqrt{s} + 5k + \frac{2^{k+1}}{i - 2k - 1}.$$

Proof. Let u be an offset-i block of size $\geq 2k$.

We claim that the red blocks predecessors of u of size at least $2k+1$ have to start at position i in regular green blocks. Indeed, let v be a prefix of size $\geq 2k+1$ of u; let us analyse as before the different cases:

- If v is included in a regular green block, then it has to start at position i by Property $(\star)$ (page 130);
- v cannot be included in a gadget since it would lead to a contradiction:
  - in gadgets of type g_0^j, v would contain 0^{2k},
  - in gadgets of type $g_a^j = x_{<a}\bar{x}_a 1^j$ (for $a \in]0, \gamma k]$), either v goes into $x_{<a}$ by at least k and $v[0..k-1]$ would appear at two positions in x, or v goes into $\bar{x}_a 1^j$ by at least $k+2$ and v would contain 1^{k+1};
- If v is included in a junction block of type 1, then v starts at position i in the left regular block, otherwise either $v[0..k-1]$ would be in the left regular block at a position different from i, or $v[k+1..2k]$ would be in the right regular block at a position $\leq \gamma k < i$;
- v cannot be included in a junction block of type 2: indeed, by the red parsing, the left part of the junction (included in a gadget) is either 10^j or $a1^j$ for some letter $a \in \{0, 1\}$, thus v cannot go on the left by $\geq k+2$ and hence has to go on the right by at least k leading to a contradiction with Property $(\star)$;
- If v is included in a junction block of type 3, then v starts at position i in the left regular block, otherwise either $v[0..k-1]$ would be in the left (regular) block at a position different from i, or, by the red parsing, the gadget is of type g_a^j (for $a > 0$) and $v[k+1..2k]$ would be included in $x_{<a}$ at a position $\leq \gamma k < i$.

Thus, at least $|u| - 2k$ red blocks end at position $i-1$.

By Lemma 9.6, at most $\frac{2^{k+1}}{i-2k-1}$ of them are junctions of type 1. Note furthermore that, as shown during the proof of Lemma 9.5, if $a \geq k+1$, the a-th junction

of type 2 stops at position 0 or 1, hence at most k of the blocks ending at position $i-1$ are junctions of type 2. Finally, there is, by definition, no junction of type 3. Therefore, there are at least $|u|-3k-\frac{2^{k+1}}{i-2k-1}$ offset blocks ending at position $i-1$. We call M the set of such blocks. See Figure 9.3. Remark that $|u|-3k-\frac{2^{k+1}}{i-2k-1}$ is a lower bound on the number of offset blocks ending at position $i-1$. But the number of such blocks is at most i. Therefore

$$|u|-3k-\frac{2^{k+1}}{i-2k-1}\leq i$$

We distinguish two cases in the proof:

First case: $i\in[\gamma k+1,2\sqrt{s}]$. Then

$$|u|-3k-\frac{2^{k+1}}{i-2k-1}\leq i\leq 2\sqrt{s}$$

so that

$$|u|\leq 2\sqrt{s}+3k+\frac{2^{k+1}}{i-2k-1}\leq 2\sqrt{s}+5k+\frac{2^{k+1}}{i-2k-1}.$$

Second case: $i>2\sqrt{s}$.

All the words in M are in the dictionary and are of different size, since two offset blocks ending at the same position and of same size would be identical, which is not possible in the LZ-parsing. The words of $\mathscr{P}(M)$ (the set of prefixes of the words in M) are also in the dictionary. Let

$$A=\frac{|u|-5k-\frac{2^{k+1}}{i-2k-1}}{2}.$$

Observe that $i-A-2k\geq\frac{i}{2}-k$ as $|u|-3k-\frac{2^{k+1}}{i-2k-1}\leq i$. Therefore $i-A-2k\geq\sqrt{s}-k$, which is large against γk. Consider the words of $\mathscr{P}(M)$ containing $x[i-A-2k..i-A]$: they must start at a position $\leq i-A-2k$ and end at a position $\in[i-A,i-1]$. The number of such words is at least the product of the number of blocks in M starting at position $\leq i-A-2k$ and of the number of possible ending points, that is, at least

$$\left(\left(|u|-3k-\frac{2^{k+1}}{i-2k-1}\right)-\left(A+2k\right)\right)A.$$

Remark that these words contain a part of a regular green block of size at least $2k+1$ starting at position $i-A-2k>\gamma k$. Hence, by the same case analysis as before, for these words, the part corresponding to the factor $x[i-A-2k..i-$

$A - k - 1]$ must appear included in a regular green block, so that two such words cannot appear in the same regular green block by Property $(\star)$ (page 130). But there are at most s distinct regular green blocks, thus:

$$\left(|u| - 5k - \frac{2^{k+1}}{i - 2k - 1} - A\right) A \leq s.$$

The value of A gives:

$$\left(\frac{|u| - 5k - \frac{2^{k+1}}{i-2k-1}}{2}\right)^2 \leq s$$

$$|u| \leq 2\sqrt{s} + 5k + \frac{2^{k+1}}{i - 2k - 1}.$$

□

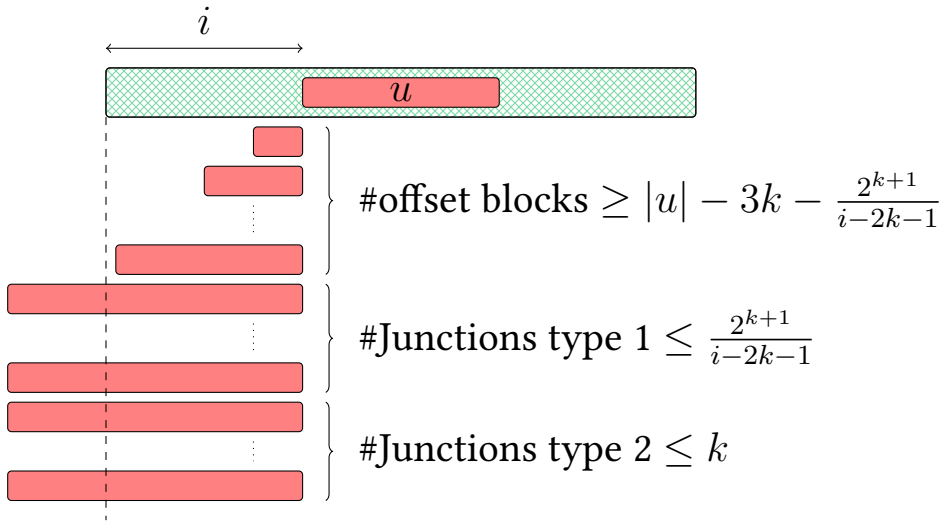


Figure 9.3: Blocks ending at position $i - 1$ for the proof of Proposition 9.7.

We are ready for the proof of the main theorem of this chapter.

Proof of Theorem 9.1. The intuition is the following: by Proposition 9.7, the red blocks starting at position j, for $j = \Omega(\sqrt{s})$, are of size $\Theta(\sqrt{s}) = \Theta(|w|^{1/4})$, so if we prove that a portion of size $\Theta(|w|)$ of the word $0w$ is covered by offset-j blocks for j large enough, then the compression size will be $\Omega(|w|^{3/4})$. To that purpose, we prove that for large enough regular green blocks, there is an interval of positions $[2s/3 - l, 2s/3]$ (with $l = 2\sqrt{s} + 5k + 3$), such that there is at least one offset-i block for $i \in [2s/3 - l, 2s/3]$.

In every regular green block of size larger than $2s/3$, let us show that there is an offset-i red block, for $i \in [2s/3 - l, 2s/3]$. Indeed, for every $i < 2s/3 - l$, the maximal size $f(i)$ of a red block starting at position i satisfies $i + f(i) \leq 2s/3$: in the case where $i > \gamma k$ we use the bound given by Proposition 9.7, and in the case where $i \leq \gamma k$, the $(t - \gamma k)$ predecessors of size $\geq \gamma k + 1$ of a red block of size t starting at position i start at position i as well (since $x_{\leq \gamma k}$ is not a factor of a gadget, it cannot be seen anywhere by a red block except at position i in a regular green block), hence the size of an offset-i block in that case is at most γk plus the number of i-violations. Therefore by Lemma 9.5 red blocks starting at position i have their size upper bounded by $\gamma k + s/2 + 1 + (1 + \gamma)k$.

Therefore, since a red block starting at position $i \geq 2s/3 - l$ is of size at most $B = 2\sqrt{s} + 5k + \frac{2^{k+1}}{2s/3-l-2k-1}$ by Proposition 9.7, each green block of size $h \geq 2s/3$ is covered by at least

$$\frac{h - 2s/3}{B} \geq \frac{h - 2s/3}{2\sqrt{s} + O(k)}$$

red blocks. Thus, the total number of red blocks is at least

$$\frac{1}{2\sqrt{s} + O(k)} \sum_{h=2s/3}^{s} (h - 2s/3) = \frac{1}{36} s^{3/2} + o(s^{3/2}).$$

With the gadgets, the size of w is at most $(5/8)s^2 + o(s^2)$, therefore the total number of red blocks is at least:

$$\frac{1}{36} \left(\frac{8}{5}\right)^{3/4} |w|^{3/4} - o(|w|^{3/4}) \geq 0.039|w|^{3/4}.$$

□

Remark 9.8

Despite the fact that $1\mathrm{Pref}(x)$ compresses optimally, this is not at all the case with the gadgets, since Theorem 9.1 remains valid with the new word w output by the algorithm even when we put 1 instead of 0 in front of w.

Chapter 10

General case

In this chapter we prove Theorem 7.10. The proof 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 9.1. The existence of this family is shown using the probabilistic method in Section 10.1: 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 Chapter 9. (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 Chapter 9 will apply individually. By choosing appropriately the number of chains and their length, we can obtain any compression size for w up to $\Theta(n/\log^2 n)$ and the matching bound for $0w$ (see Theorem 7.10).

The organisation of the chapter is as follows: Section 10.1 is devoted to the proof of existence of the required family of words. Section 10.2 defines the gadgets, describes the construction of w via an algorithm, and gives the upper bound on the compression size of w. Finally, Section 10.3 shows the lower bound on the compression size of $0w$ via a series of results in the spirit of Section 9.4.

Throughout the present chapter, we use parameters with some relations between them that are worth being stated once and for all in Figure 10.1 for reference.

In particular, note that we have the following relations:

$$0 \leq p \leq \sqrt{l} \quad \text{and} \quad \frac{\gamma \log n}{3} \leq m \leq \frac{\sqrt{l}}{9}.$$

n sufficiently large (the size of w)

$\gamma \geq 10$ (an absolute constant)

$l \in [(9\gamma)^2 \log^2 n, \sqrt{n}]$ (size of the x^j)

$p = \log \frac{n}{l^2}$ (2^p is the number of chains)

$k = \frac{\log l}{2}$ (parameter in (P1))

$m = \max(\gamma p, \gamma \log l)$ (parameter in (P2)).

Figure 10.1: Parameters used throughout Chapter 10.

10.1 Family of de Bruijn-type words

We need two properties for a family F of 2^p words $x^1, \ldots, x^{2^p}$ of size l (the parameters n, l, p, k, γ and m are those given in Figure 10.1): the first is a relaxed version of Property $(\star)$ on "true" de Bruijn words; the second guarantees that the words of F are "independent".

- (P1) For all $x \in F$, for all words u of size $\leq k$,
$$\mathrm{Occ}_x(u) \leq \frac{kl}{2^{|u|}}.$$
- (P2) Any factor u of size m appears in at most one word of the family F, and within that word at only one position.

Note that in Chapter 9, we did not need (P2) since only one word was concerned, but still (P2) was true for the same value k as in $(\star)$, instead of m here.

The following lemmas show that (P1) and (P2) hold with high probability for a random family F. We first recall the well-known Chernoff bound.

Theorem 10.1: Chernoff bound

Let $X_1, \ldots, X_n$ be independent random variables over $\{0, 1\}$, and $X = \sum X_i$. Denote by μ the expectation of X. Let $\delta > 1$. Then:

$$\Pr(X > \delta\mu) < 2^{-\frac{(\delta-1)\mu \log \delta}{2}}.$$

For (P1), we need to consider positions separated by a distance k in order to obtain the independence required for the Chernoff bound; then a union bound will complete the argument for the other positions.

Lemma 10.2: (P1) holds whp

Let p and l be positive integers such that $p \leq \sqrt{l}$. Let F be a family of 2^p words $x^1, \ldots, x^{2^p}$ of size l chosen uniformly and independently at random. Then F satisfies Property (P1) with probability $2^{-\Omega(\sqrt{l}\log\log l)}$.

Proof. Fix a word u of size $\leq k$ and take x of size l at random. For $i \in [0, k-1]$ and $j \in [0, l/k - 1]$, let

$$X_j^i = \begin{cases} 1 & \text{if } u \text{ occurs at position } i + jk \text{ in } x \\ 0 & \text{otherwise.} \end{cases}$$

For a fixed i, the X_j^i are independent. Let $\mu_i = E(\sum_j X_j^i)$. We have:

$$\mu_i = \frac{l}{k2^{|u|}}.$$

By the Chernoff bound (Theorem 10.1):

$$\Pr\left(\sum_j X_j^i > k\mu_i\right) < 2^{-\frac{(k-1)(\log k)\mu_i}{2}},$$

that is,

$$\log\Pr\left(\sum_j X_j^i > \frac{l}{2^{|u|}}\right) < -(k-1)(\log k)\frac{l}{k2^{k+1}}.$$

By union bound over all the words u of size at most k, all the words of F and all the moduli $i \in [0, k-1]$, we have:

$$\begin{aligned}\log\Pr\left(\sum_{i,j} X_j^i > \frac{kl}{2^{|u|}}\right) &< (k+1) + p + \log k - (k-1)(\log k)\frac{l}{k2^{k+1}} \\ &= -\Omega\big(\sqrt{l}\log\log l\big).\end{aligned}$$

□

The analysis for (P2) does not use Chernoff bounds, but instead it uses a slight "independence" on the occurrences of a factor u obtained by showing that u can be supposed "self-avoiding" (the precise meaning of these ideas will be clear in the proof).

Lemma 10.3: (P2) holds whp

Let p and l be positive. Recall that $m = \max(\gamma p, \gamma \log l)$ in (P2). Let F be a family of 2^p words $x^1, \ldots, x^{2^p}$ of size l chosen uniformly and independently at random. Then F satisfies Property (P2) with probability a least $1 - 2/l$.

Proof. Let us first show that we can assume with high probability that factors of w_i are not overlapping too much. We say that a word u of size m is "bad" if it overlaps itself at least by half, that is:

$$\exists i \in [|u|/2, |u| - 1] : u[0..i-1] = u[|u| - i..|u| - 1] \text{ (we say that } u \text{ is } i\text{-bad).}$$

(Remark that a word u can be both i-bad and j-bad for $i \neq j$.) Let us first bound the number of bad words. If u is i-bad, then for each $j < i$, $u_{j+|u|-i} = u_j$. Therefore, specifying the $|u| - i$ first bits specifies the whole word u, meaning that there are at most $2^{|u|-i}$ i-bad words. In total, there are at most

$$\sum_{i=|u|/2}^{|u|-1} 2^{|u|-i} = 2^{1+|u|/2} - 2$$

bad words, that is, a fraction $< 2^{-m/2+1}$ of all words of size m.

Now, we say that a word x^j of size l is "good" if it contains no bad factor. Let us show that, with high probability, all the words $x^j \in F$ are good (Property (G)). Fix $j \in [1, 2^p]$. If x^j is not good, then there is at least one position where a bad factor u occurs:

$$\Pr(x^j \text{ is not good}) \leq |x^j| \Pr_{|u|=m}(u \text{ is bad}) \leq l 2^{-m/2+1}.$$

We use the union bound over all 2^p words $x^j \in F$ to obtain:

$$\Pr(G) \geq 1 - l 2^{p-m/2+1}.$$

Since property G has very high probability, we will only show that (P2) holds with high probability when G is satisfied. Let $x = x^1 \ldots x^{2^p}$ (the size of x is therefore $l2^p$). Let u be a word of size m, which is not bad. Let X_u be the number of occurences of u in x. In order to get at least two occurrences of u, we have to choose two positions, the $|u|$ bits of the first occurrence, and the bits of the second occurrence that are not contained in the first; but u can't overlap itself by more than $m/2$ bits, thus:

$$\Pr(X_u \geq 2) \leq \frac{|w|}{2^m} \frac{|w|}{2^{m/2}} \leq l^2 2^{2p - \frac{3}{2}m}.$$

Using the union bound over all good words u of size m, of which there are at most 2^m, we get:

$$\Pr(\forall \text{ good } u, X_u \leq 1) \geq 1 - 2^m l^2 2^{2p-\frac{3}{2}m} = 1 - l^2 2^{2p-m/2}.$$

Now, the probability that F respects Property (P2) can be lower bounded by the probability that F contains no bad words, and that the number of occurences of good words is at most 1, which gives:

$$\begin{aligned}\Pr(F \text{ satisfies (P2)}) &\geq \Pr(G \wedge \forall \text{ good } u, X_u \leq 1) \\ &\geq 1 - l^2 2^{2p-m/2} - l 2^{p-m/2+1} \\ &> 1 - \frac{2}{l} \text{ since } \gamma \geq 10\end{aligned}$$

(for the last line, consider the two following cases: $p \geq \log l$ where $m = \gamma p$ and $l \leq n^{1/3}$; and $p \leq \log l$ where $m = \gamma \log l$ and $l \geq n^{1/3}$). □

Corollary 10.4

For all sufficiently large l and $p \leq \sqrt{l}$, there exists a family F of 2^p words $x^1, \ldots, x^{2^p}$ of size l satisfying Properties (P1) and (P2), and where the first bit of x^1 is 1.

10.2 Construction

(Recall the choice of parameters n, l, p, k, γ and m defined in Figure 10.1.)

For n sufficiently large and $l \in [(9\gamma)^2 \log^2 n, \sqrt{n}]$, we will construct a word w of size n whose compression size is $\Theta(n/l)$ whereas the compression size of $0w$ is $\Theta(n/\sqrt{l})$ (thus matching the upper bound of Theorem 7.7). Let F be a family as in Corollary 10.4. For some integers q_j (defined below), the word w will merely be the concatenation of $\text{Pref}_{>q_j}(x^j)$ (see Remark 7.1 for the definition of $\text{Pref}_{>q}(x)$) for all the 2^p words x^j of the family F, with possibly some gadgets added between the prefixes of x^j (each $\text{Pref}_{>q_j}(x^j)$ together with the possible gadgets will be denoted z^j and called a "chain"), and a trailing set of zeroes so as to "pad" the length to exactly n. The integer q_j will be chosen so that the first occurrence of $x^j[0..q_j]$ is parsed in exactly one green block.

Each chain z^j (with gadgets) is of size $\Theta(l^2)$ and is fully compressible in w (compressed size $\Theta(l)$) since it is made of prefixes (plus gadgets that won't impede much the compression ratio). Thus the total compression size of w is $\Theta(l2^p)$, compared to $|w| = n = \Theta(l^2 2^p)$ for a compression size of $\Theta(n/l)$.

On the other hand, due to the properties of F and similarly to Theorem 9.1, in $0w$ each chain will compress only to a size $\Theta(l^{3/2})$, thus the total compression size of $0w$ is $\Theta(l^{3/2}2^p)$, for a compression size of $\Theta(n/\sqrt{l})$.

Remark 10.5

- If we take the smallest possible l, that is, $l = (9\gamma)^2 \log^2 n$, then we obtain compression sizes of $\Theta(n/\log^2 n)$ and $\Theta(n/\log n)$, thus showing the one-bit catastrophe.
- On the other hand, if we take the largest possible l, that is, $l = \sqrt{n}$, then we obtain $\Theta(\sqrt{n})$ and $\Theta(n^{3/4})$ as in Theorem 9.1.

Let us now start the formal description of the word w. As previously, we will call green the blocks in the parsing of w and red those in the parsing of $0w$. The green blocks in each chain z^j that are not gadgets will be called "regular blocks" (they are of the form $x^j[0..q]$ for some q). Recall that the chain z^j will be of the form $\mathrm{Pref}_{>q_j}(x^j)$ with possibly some gadgets between the prefixes. We can already define the integers q_j:

$$q_j = \min\{i \geq 0 : x^j[0..i] \text{ is not a prefix of } x^1, \ldots, x^{j-1}\}.$$

In that way, we guarantee that the first green block in each z^j is exactly $x^j[0..q_j]$. Remark that, by Property (P2), $q_j \in [0, m]$. For all j we will denote by $s_j = |x^j| - q_j = l - q_j$ the number of regular green blocks in z^j.

Fix n and $l = l(n) \in [(9\gamma)^2 \log^2 n, \sqrt{n}]$, and let $k = (\log l)/2$ and $p = \log(n/l^2)$. As in Property (P2), call $m = \max(\gamma p, \gamma \log l)$. Here are the new gadgets that will (possibly) be inserted in the chain z^j ($j \in [1, 2^p]$), for $i \in [0, 2k\sqrt{l}]$:

- for $c \geq 0$: $g_0^c(j) = ua^c$, where $a = x^j[0]$ is the first letter of x^j, and u is the smallest word in

  $$\mathrm{Dic}(0z^1 \ldots z^{j-1}\mathrm{Pref}_{>q_j}(x^j[0..|x^j|/2]))$$

  but not in

  $$\mathrm{Dic}(z^1 \ldots z^{j-1}\mathrm{Pref}_{>q_j}(x^j)) :$$

  this is a word which is in the parsing of $0w$ up to the insertion of $g_0^0(j)$ but not in the corresponding parsing of w (Lemma 10.7 below guarantees the existence of such a word and proves it is of size $\leq m$);

- for $i > 0$ and $c \geq 0$, let $m' = \max(i, m)$ and $v = x^j[0..m-1]1^l$. Then:

$$g_i^c(j) = x^j[0..m'-1]\bar{x}^j_{m'}v[0..c-1]$$

where $\bar{x}^j_{m'}$ denotes the complement of $x^j[m']$.

We define i-violations in each chain z^j as previously, that is, a regular green block is i-violated if it contains an offset-i red block. The following lemma is proved in the exact same way as Lemma 9.3.

Lemma 10.6

For $j \in [1, 2^p]$ and $i \in [0, s_j - 1]$, let l_i^j be the number of i-violated blocks in z^j. Then for all j and all $i \neq i'$, $l_i^j + l_{i'}^j \leq s_j$.

In particular, for each z^j there can be at most one i such that the number of i-violated blocks is $> s_j/2$.

The formal construction of the word w is once again best described by an algorithm taking as parameters n and l:

1. For all $j \in [1, 2^p]$ and $i \in [q_j + 1, l]$, $z_i^j \leftarrow x^j[q_j..i-1]$. Throughout the algorithm, z^j will denote $z^j_{q_j+1} \dots z^j_l$ (and thus will vary if one of the z_i^j varies).

2. For $j = 1$ to 2^p do:

   (a) if there is $i \in [0, 2k\sqrt{l}]$ (unique by Lemma 10.6) such that the number of i-violations in the chain z^j is $> s_j/2$, then:

      i. let $c = 0$ (counter for the number of inserted gadgets in z^j) and $d = s_j/2 + 1$ (counter for the place of the gadget to be inserted),

      ii. while the number of i-violations in the chain z^j is $\geq d$, do:

         A. let r be such that z_r^j is the d-th i-violated green block in z^j,

         B. $z_r^j \leftarrow g_i^c(j)z_r^j$ (we add the gadget $g_i^c(j)$ before the block $x^j[0..r-1]$),

         C. $c \leftarrow c + 1$,

         D. if z_r^j is still i-violated, then $d \leftarrow d + 1$.

3. Let $w' = z^1.z^2 \dots z^{2^p}$. Return $w = w'0^{n-|w'|}$ (padding to obtain $|w| = n$).

Remark that we have the following bounds on the size of w'. Its size is minimal if no gadgets are added:

$$|w'| \geq 2^p \left(\sum_{i=m}^{l} i \right) = \frac{n}{l^2} \cdot \frac{(l-m+1)(l+m)}{2} \geq \frac{n}{2} - o(n)$$

and its size is maximal if each chain contains $l/2$ gadgets g_i^c (whose size is at most $2k\sqrt{l}+1+c$):

$$\begin{aligned}|w'| &\leq 2^p\left(\sum_{i=1}^{l} i + \sum_{c=0}^{l/2-1}(2k\sqrt{l}+1+c)\right)\\ &\leq \frac{n}{l^2}\left(\frac{5l^2}{8}+2kl^{3/2}\right)\leq n.\end{aligned}$$

Therefore at the end of the algorithm it is legitimate to pad w' with at most $n/2+o(n)$ zeroes to obtain the word w of size precisely n.

The following lemma justifies the existence of the gadgets $g_0^c(j)$.

Lemma 10.7

There is a constant $C>0$ such that, for all n, for all $l\in[(9\gamma)^2\log^2 n,\sqrt{n}]$ with $l>C$, for all $j\in[1,2^p]$ (where $p=\log(n/l^2)$) there exists a word u of size $\leq m$ in

$$\mathrm{Dic}(0z^1\dots z^{j-1}\mathrm{Pref}_{>q_j}(x^j[0..|x^j|/2]))$$

but not in $\mathrm{Dic}(z^1\dots z^{j-1}\mathrm{Pref}_{>q_j}(x^j))$.

This implies that we can insert the gadgets $g_0^c(j)$ in a chain z^j whenever we need to.

Proof. For $j=1$: the first red block in z^1 is 0, but all the regular green blocks in z^1 begin with 1 (cf. Corollary 10.4). Therefore there exists a word u of size 1 in

$$\mathrm{Dic}(0\mathrm{Pref}_{>q_1}(x^1[0..|x^1|/2]))$$

not in $\mathrm{Dic}(\mathrm{Pref}_{>q_1}(x^1))$.

For $j>1$: as we shall see in the proof of Theorem 7.10 below[1], in z^{j-1} there is an interval of $A=3\sqrt{l}$ positions that contains the starting position of a red block in at least $l/3$ regular green blocks. One of the positions of the interval is the starting point of a branch of at least

$$\frac{l}{3A}=\frac{\sqrt{l}}{9}\geq m$$

red blocks. Thus there is a red block of size m which appears nowhere in $z^1,\dots,z^{j-1}$ by (P2) and is not a prefix of z^j, thus it is a word of $\mathrm{Dic}(0z^1\dots z^{j-1})$ not in $\mathrm{Dic}(z^1\dots z^{j-1}\mathrm{Pref}_{>q_j}(x^j))$. □

[1]That is not a circular argument because we only need the result up to z^{j-1} to claim the existence of gadgets for z^j.

We can now show that w has compression size $O(|w|/l)$ by giving an upper bound on the size of the dictionary of w.

Lemma 10.8

The compression size $|\mathrm{Dic}(w)|$ of w is at most

$$\frac{3+\sqrt{3}}{2} \cdot \frac{|w|}{l}.$$

Proof. The definition of the integers q_j guarantees that the parsing resynchronizes at each beginning of a new chain z^j.

In a chain z^j, the definition of $g_0^0(j)$ guarantees that this gadget, if present, will be parsed in exactly one green block, and after that the subsequent gadgets $g_0^c(j)$ also.

Similarly, (P2) together with the fact that gadgets are only inserted in the second half of a chain (thus, after more than m green blocks) imply that the possible gadgets $g_i^c(j)$ for $i > 0$ are also parsed in exactly one green block.

For each chain z^j, the parse tree consists in a main path of size l (regular green blocks) together with another path of size $\leq l/2$ corresponding to the gadgets $g_i^c(j)$. The compression size cannot be worse than in the (hypothetical) case where these two paths begins at depth 0, for all j. In that case, there are $\leq (3/2)l$ green blocks for each chain, and a size

$$|z^j| \geq \frac{l(l+1)}{2} + \frac{\frac{l}{2}(1+\frac{l}{2})}{2} \geq \frac{5}{8}l^2.$$

Since the number of chains is $2^p = n/l^2$, in that (hypothetical) worst case the number of green blocks in w' is at most $3n/2l$ and $|w'| \geq 5n/8$. The $\leq 3n/8$ trailing zeroes of w are parsed in at most $\sqrt{3n}/2 \leq (\sqrt{3}/2)n/l$ green blocks. Hence the compression size of w is at most $(3/2 + \sqrt{3}/2)(n/l)$. □

10.3 Proof of the main theorem

(Recall the choice of parameters n, l, p, k, γ and m defined in Figure 10.1.)

We now prove the lower bound of Theorem 7.10. Recall that z^j denote the j-th chain of w. We will write w_i^j the i-th regular block of the chain z^j. As in the previous chapter, we will distinguish junctions over two consecutive regular blocks (type 1); junctions starting in a gadget and ending in a regular block (type 2); and junctions starting in a regular block and ending in a gadget (type 3).

The next proposition is the core of the argument, and Theorem 7.10 will follow easily. The proposition is a corollary of lemmas that we will show afterwards.

Proposition 10.9

Let $f(i)$ be the maximal size of an offset-i (red) block included in a regular green block.

- If $i \leq 2k\sqrt{l}$ then $f(i) \leq \frac{l}{2} + 4k\sqrt{l} + 2m + 1$.
- Otherwise, $f(i) \leq 2\sqrt{l} + 3k + 7m + \frac{2kl}{i-4m-2}$.

Proof. The first point is a consequence of Lemmas 10.10 and 10.11. The second point is exactly Lemma 10.13. □

With Proposition 10.9 in hand, let us prove the main theorem.

Proof of Theorem 7.10. We will show that each chain z^j in $0w$ is parsed in at least $\frac{1}{54}l^{3/2}$ blocks, thus

$$|\mathrm{Dic}(0w)| \geq 2^p \frac{1}{54} l^{3/2} = \frac{1}{54} \cdot \frac{|w|}{\sqrt{l}}.$$

Fix an index j. In order to prove that the chain z^j is parsed in at least $\frac{1}{54}l^{3/2}$ red blocks, we first prove that in every regular green block of size larger than $2l/3$ in the chain z^j, there is an interval of positions $[2l/3 - A, 2l/3]$ (with $A = 3\sqrt{l}$), such that there is at least one offset-i (red) block for $i \in [2l/3 - A, 2l/3]$. Indeed, by Proposition 10.9, for any $i < 2l/3 - A$, the maximal size $f(i)$ of a red block starting at position i satisfies $i + f(i) \leq 2l/3$.

Therefore, since the red blocks starting at position $i \geq 2l/3 - A$ are of size at most $f(2l/3 - A) \leq 3\sqrt{l}$, a regular green block of z^j of size h is covered by at least $(h - 2l/3)/(3\sqrt{l})$ red blocks. Thus the number of red blocks in the parsing of z^j is at least

$$\sum_{h=2l/3}^{l} \frac{h - 2l/3}{3\sqrt{l}} \geq \frac{1}{54} l^{3/2}.$$

□

Now we prove Proposition 10.9 thanks to the next four lemmas. The first two show that the gadgets do their job: indeed, for small i (the indices i covered by the gadgets), the offset-i blocks are not too large. For that, we first bound the number of violations.

Lemma 10.10

For any $i \in [0, 2k\sqrt{l}]$ and $j \in [1, 2^p]$, the number of i-violations in the chain z^j is at most $\frac{l}{2} + 2m + 1 + 2k\sqrt{l}$.

Proof. We fix j and focus on the number of i-violations in the chain z^j. Recall that s_j denotes the number of regular blocks in the chain z^j ($s_j \leq l$).

If no gadgets have been added during the execution of the algorithm, then for all $i \in [0, 2k\sqrt{l}]$, the number of i-violations is $\leq s_j/2 \leq l/2$.

Otherwise, we distinguish on the type of the most frequent violation ($i = 0$ or $i > 0$).

- Case 1: the most frequent violations are 0-violations. In that case, a proof similar to the case 1 of Lemma 9.5 (with 0 replaced by $a = x^j[0]$ the first letter of x^j) shows that the number of i-violations for any $i \in [0, 2k\sqrt{l}]$ is $\leq s_j/2 \leq l/2$.

- Case 2: the most frequent violations are i-violations for some $i > 0$. Let us see how the parsing of $0w$ splits the gadgets. As in the definition of the gadgets, let $m' = \max(i, m)$ and $v = x^j[0..m-1]1^l$. When the first gadget $g_i^0(j) = x^j[0..m'-1]\bar{x}^j_{m'}$ is added, the red parsing splits the gadget $g_i^0(j)$ between $x^j_{<i}$ and $x^j[i..m'-1]\bar{x}^j_{m'}$, because the gadget is added before a regular block with an i-violation. Furthermore, $x^j[i..m'-1]\bar{x}^j_{m'}$ is not split by the parsing, because at that moment in the algorithm, the number of i-violations in the previous regular green blocks is $s_j/2 \geq m' - i$, so that, as the position i has been seen $\geq m' - i$ times, the word $x^j[i..m'-1]$ is already in the dictionary of $0w$. Similarly, the gadget $g_i^c(j) = x^j[0..m'-1]\bar{x}^j_{m'}v[0..c-1]$ is split by the red parsing between $x^j_{<i}$ and $x^j[i..m'-1]\bar{x}^j_{m'}v[0..c-1]$, with the additional property that this second part is not split by the parsing.

  But for the gadget $g_i^{2m+1}(j)$, the second part $x^j[i..m'-1]\bar{x}^j_{m'}x^j[0..m-1]1^{m+1}$ is parsed in exactly one block because this factor does not appear anywhere in a regular block because of 1^{m+1} (cf. (P2)) nor in a gadget of a preceding chain because of $x^j[0..m-1]$ (cf. (P2) again). From that moment on, each i-violation creates a 0-violation. The number of green blocks that are both 0-violated and i-violated is at most $i \leq 2k\sqrt{l}$. Thus, at most $2k\sqrt{l}$ more gadgets fail to kill the corresponding i-violation. The total number of "failing" gadgets in the chain z^j is at most $2m + 1 + 2k\sqrt{l}$.

□

If the number of i-violations is not too large, then the same is true for the size of offset-i blocks, as the following easy result states.

Lemma 10.11

If the number of i-violations in the chain z^j is b, then any offset-i block u in a regular block of z^j is of size at most $b + 2k\sqrt{l}$.

Proof. The $|u| - 2k\sqrt{l}$ predecessors of u of size at least $2k\sqrt{l} + 1$ cannot appear in gadgets, hence by Property (P2) they must appear at position i in the regular green blocks of the chain z^j. Therefore, each such predecessor contributes to an i-violation in z^j, so that $|u| - 2k\sqrt{l} \leq b$. □

Now, the next two results show that, for large i, the size of offset-i blocks is small. First, we need to bound the number of junction blocks ending at position $i - 1$.

Lemma 10.12

Let j be fixed and $i > 2k\sqrt{l}$. Let uu' be a junction block of type 1 between two regular green blocks w_a^j and w_{a+1}^j, ending at position $i - 1$ in w_{a+1}^j (thus $|u'| = i$). Then $|u| \leq \log(kl) - \log(i - 4m - 2)$.

In particular, the number of such blocks is upper bounded by $\frac{2kl}{i-4m-2}$.

Proof. Let v be the prefix of size $4m + 1$ of u' (which is also the prefix of x^j). We claim that all the prefixes of uu' of size $\geq |uv|$ are junction blocks of type 1 or 3 only (except possibly for one of type 2), with only u on the left side of the junction. Indeed, recalling the red parsing of gadgets explained in the proof of Lemma 10.10 and Property (P2), we distinguish the following cases:

1. uv cannot be completely included in a regular block, otherwise $v[0..m-1]$ would appear both at positions 0 and $p > 0$ in x^j, which contradicts Property (P2);

2. uv cannot be completely included in a gadget:

   - if the gadget is $g_0^c(j)$, then v would contain a^{m+1} for some $a \in \{0, 1\}$,
   - if the gadget is $g_b^c(j)$ for $b > 0$, let $m' = \max(b, m)$: the red parsing splits this gadget between $x^j[0..b-1]$ and $x^j[b..m'-1]\bar{x}_{m'}^j x^j[0..m-1]1^d$. Then uv is not contained in the first part by (P2), nor in the second part since it cannot contain 1^{m+1};

3. if uv is a type 1 junction but not split between u and v, it is impossible because the three possible cases lead to a contradiction:

   - if u goes on the right, then v would appear at another position $p > 0$ in x^j,
   - if v goes on the left by at least m, then $v[0..m-1]$ would again appear at two different positions in x^j,
   - if v goes on the left by less than m, then it goes on the right by more than m and $v[3m+1..4m]$ would again appear at two different positions in x^j;

4. if uv is a type 2 junction but not split between u and v, it is again impossible:

   - if u goes on the right, then v would appear at another position $p > 0$ in x^j,
   - if v goes on the left by at least $3m+1$, in case of $g_0^c(j)$ then v would contain a^{2m+1} and in case of $g_b^c(j)$ then v would contain 1^{m+1} (recall where the red parsing splits this gadget),
   - otherwise, v goes on the right by at least $m+1$, and $v[3m+1..4m]$ would appear at two different positions x^j;

5. if uv is a type 3 junction, first remark that the gadget is of the form $g_b^c(j)$ for $b > 0$ (because, for gadgets of the form $g_0^c(j)$, the red parsing starts at position 0 of the gadget). If uv is not split between u and v, it is once again impossible:

   - if u goes on the right, the red parsing of the gadget stops after $x^j_{<b}$ and v would appear in x^j at a non-zero position,
   - similarly, if v goes on the left by less than m, then $v[m..2m-1]$ would again appear at two different positions in x^j,
   - if v goes on the left by at least m, then $v[0..m-1]$ would again appear at two different positions in x^j.

Remark finally that all parsings of type 2 junctions have different sizes on the left. Therefore, at most one can contain u on the left. The claim is proved. Thus u appears at least $i - 4m - 2$ times as a suffix of a regular green block.

Remark that Property (P1) implies that factors of size more than k appear at most $k\sqrt{l}$ times in x^j. Thus, since $i - 4m - 2 > k\sqrt{l}$, we have $|u| \leq k$. Hence by Property (P1), the number of occurrences of u is upper bounded by $kl/2^{|u|}$.

Therefore

$$\begin{aligned} i - 4m - 2 &\leq \frac{kl}{2^{|u|}} \\ |u| &\leq \log(kl) - \log(i - 4m - 2), \end{aligned}$$

which proves the first part of the lemma.

The number of such blocks is then upper bounded by the number of words of size $\leq \log(kl) - \log(i - 4m - 2)$, that is, $\frac{2kl}{i-4m-2}$. □

The last lemma completes the preceding one: if an offset-i block is large, then a lot of blocks have to end at position $i - 1$ and too many of their prefixes would have to be in different green blocks.

Lemma 10.13

For any $j \in [1, 2^p]$ and any $i > 2k\sqrt{l}$, the size of an offset-i block included in a regular green block of the chain z^j is at most

$$2\sqrt{l} + 3k + 7m + \frac{2kl}{i - 4m - 2}.$$

Proof. We argue as in Proposition 9.7. Let u be an offset-i block included in a regular green block of the chain z^j. We show as before that the $|u| - 3m$ predecessors of u of size $\geq 3m$ have to start at position i in regular green blocks. Indeed, let v be a prefix of size $\geq 3m$ of u; let us analyse the different cases:

- If v is included in a regular green block, then it has to start at position i by Property (P2);
- v cannot be included in a gadget since it would lead to a contradiction:
  - in gadgets of type $g_0^c(j)$, v would contain a^{m+1} for some letter $a \in \{0, 1\}$,
  - in gadgets of type $g_b^c(j)$ (for $b \in]0, 2k\sqrt{l}]$), either v would contain 1^{m+1} or a factor of x^j of size m and at a position different from i;
- If v is included in a junction block of type 1, then v starts at position i in the left regular block, otherwise either $v[0..m-1]$ would be in the left regular block at a position different from i, or $v[m-1..2m-2]$ would be in the right regular block at a position $\leq 3m < i$;

- v cannot be included in a junction block of type 2. Indeed, it cannot go by $\geq m$ on the right (by (P2)), thus it goes on the left by at least $2m+1$: for $g_0^c(j)$ it would contain a^{m+1} (for some $a \in \{0,1\}$), and for $g_b^c(j)$ ($b>0$), it would either contain 1^{m+1}, or a factor of x^j of size m at a position $< m' \leq i$;

- If v is included in a junction block of type 3, then v starts at position i in the left regular block, otherwise either $v[0..m-1]$ would be in the left (regular) block at a position different from i, or, by the red parsing, the gadget is of type $g_b^c(j)$ (for $b>0$) and $v[m-1..2m-2]$ would be included in x^j at a position $\leq 3m < i$.

Thus, at least $|u|-3m$ red blocks end at position $i-1$ in the regular blocks of z^j. Among them:

- By Lemma 10.12, at most $\frac{2kl}{i-4m-2}$ of them are junctions of type 1.

- At most $2m$ of them are junctions of type 2, since from the $(2m+1)$-th gadget on, the type 2 junctions end at position 0 (in case of gadgets $g_b^c(j)$ for $b>0$, see the proof of Lemma 10.10) or $\leq m+1 \leq i-1$ (in case of gadgets $g_0^c(j)$).

- There is no junction of type 3 by definition.

Overall, at least $|u|-5m-\frac{2kl}{i-4m-1}$ of them are offset blocks, ending at position $i-1$. We call the set of such blocks M. Let

$$A = \frac{|u|-7m-\frac{2kl}{i-4m-2}}{2}$$

and

$$S = \{u \in \mathscr{P}(M) \text{ containing } x^j[i-A-2m..i-A-1]\}.$$

We say that a red block w is *problematic* if $w \in S$ but the part of w corresponding to the factor $x^j[i-A-2m..i-A-m-1]$ is not completely included in a regular green block. We show that the number of problematic blocks is at most $2k\sqrt{l}+2m+1$.

1. The number of problematic blocks that overlap a gadget $g_0^c(j) = ua^c$ in the red parsing is at most $m+1$. Indeed, ua^c is never split by the red parsing, therefore for $c \geq m+1$, a red block that overlaps $g_0^c(j)$ would contain a^{m+1}, which is not a factor of x^j.

2. For $b>0$ (recall that $b \leq 2k\sqrt{l}$), note that the red parsing splits the gadget $g_b^c(j)$ after $x^j[0..b-1]$.

- Observe first that the number of problematic blocks that overlap the second part of the gadget ($g_b^c(j)_{\geq b}$) is at most m. Indeed, this part is not split by the red parsing, therefore for $c \geq m$ a red block that overlaps this part would contain $\bar{x}^j_{m'}x^j[0..m-1]$, which is not possible since the position of the word $x^j[0..m-1]$ should be 0 by Property (P2)
- The number of problematic blocks that appear completely included in the first part of a gadget ($g_b^c(j)_{<b}$) is at most b. Otherwise, the red parsing creates at least one red block completely included in the first part $x^j[0..b-1]$ of the gadget, and we claim that this can happen at most b times. Indeed, each time the parsing falls in this case, the last red block included in the first part of the gadget has to end at position $b-1$, but the size of this block has to be different each time, so that this second case can occur at most b times. Finally, each time a gadget is parsed, at most one of the red blocks included in the first part of the gadget can be a word of S by Property (P2).

3. There is no problematic blocks that are junction blocks of type 1 or 3. Indeed, if it were the case, the right part of the junction would be of size $\leq m-1$ since otherwise the problematic block would contain $ax^j[0..m-1]$ for some letter a, which is not possible. Therefore, within the problematic block, the factor $x^j[i-A-2m..i-A-m-1]$ appears on the left side of the junction and is thus included in a regular block.

4. The number of problematic blocks that are junction blocks of type 2 has already been considered when considering the gadgets.

All the red blocks corresponding to words of S and that are not problematic have to appear in distinct regular green blocks by Property (P2). As before, a word of S is obtained by choosing its beginning before the interval and its end after, so that

$$|S| \geq \left(|u| - 5m - \frac{2kl}{i-4m-2} - (A+2m)\right) \cdot A.$$

Therefore:

$$\begin{aligned} |S| - (2k\sqrt{l} + 2m + 1) &\leq l \\ \left(\frac{|u| - 7m - \frac{2kl}{i-4m-2}}{2}\right)^2 - (2k\sqrt{l} + 2m + 1) &\leq l, \end{aligned}$$

so that

$$\begin{aligned} |u| &\leq 2\sqrt{l + 2k\sqrt{l} + 2m + 1} + 7m + \frac{2kl}{i-4m-2} \\ &\leq 2\sqrt{l} + 3k + 7m + \frac{2kl}{i-4m-2}. \end{aligned}$$

□

Chapter 11

Infinite words

The techniques on finite words developed in the preceding chapters can almost be used as a black box to prove the one-bit catastrophe for infinite words (Theorem 7.6). 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 Chapter 10, we concatenated the bricks obtained in Chapter 9; now, we concatenate an infinite number of bricks of Chapter 10 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 size will be locally the same everywhere.

We will need an infinite sequence of families $(F_i)_{i\geq 0}$ of words similar to that of Chapter 10: thus we will need infinite sequences of parameters to specify them.

- For $i \geq 0$, the size of words in F_i will be $l_i = l_0.2^i$, for l_0 sufficiently large.
- Let $p_i = \sqrt{l_i}/(9\gamma) - 2\log l_i$, where $\gamma \geq 10$ is a constant. For $i > 0$, the number of words in F_i will be $|F_i| = 2^{p_i} - 2^{p_{i-1}}$ (and $|F_0| = 2^{p_0}$). Remark that $\sum_{j=0}^{i} |F_j| = 2^{p_i}$ and $|F_{i+1}| \sim |F_i|^{\sqrt{2}}$.
- The parameter $k_i = (\log l_i)/2$ will be the maximal size of words in Property P1(i) below.
- The parameter $m_i = \gamma p_i$ will be the size of words in Property P2(i) below.

We shall later show that there exists an infinite sequence $\mathcal{F} = (F_i)_{i\geq 0}$ matching these parameters and satisfying some desired properties (generalized versions of Properties (P1) and (P2), see below). But from an arbitrary sequence $(F_i)_{i\geq 0}$, let us first define the "base" word from which w will be constructed.

Definition 11.1

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 $q_x^i = \max\{a \,:\, x_{<a} \text{ is a prefix of a word in } \cup_{j<i} F_j\}$.

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 chapters. The sequence $\mathcal{F}$ that we shall consider will be a sequence of families of random words which will satisfy the following properties (Lemma 11.2 below shows that these properties are true with high probability).

P1(i): For all $x \in F_i$, for all words u of size at most k_i, $\mathrm{Occ}_x(u) \leq k_i l_i / 2^{|u|}$.

(P1'): For all $i \geq 0$, P1(i).

P2(i): Any factor u of size m_i appears in at most one word of $\cup_{j\leq i} F_j$, and within that word at only one position.

(P2'): For all $i \geq 0$, P2(i).

Again, (P2') guarantees a kind of "independence" of the families F_0, F_1, ..., whereas (P1') is a de Bruijn-style "local" property on each word of each family F_i.

Our first lemma shows that there exists a sequence $\mathcal{F} = (F_i)_{i\geq 0}$ satisfying (P1') and (P2').

Lemma 11.2

For every $i \geq 0$, let F_i be a set of $2^{p_i} - 2^{p_{i-1}}$ words of size l_i (and 2^{p_0} words of size l_0 for F_0) taken uniformly and independently at random. Then the probability that $\mathcal{F}$ satisfies Properties (P1') and (P2') is non-zero.

Proof. Let us show that the probability that $\mathcal{F}$ satisfies (P1') is $> 1/2$, and similarly for (P2'). We only show it for (P2'), as an analogous (and easier) proof gives the result for (P1') as well.

By Lemma 10.3, the probability that $\mathcal{F}$ does not satisfy P2(i) is less than $2/l_i = 2^{1-i}/l_0$. Thus, by union bound, the probability that all P2(i) are satisfied is larger than

$$1 - \sum_{i=0}^{\infty} \frac{2^{1-i}}{l_0} = 1 - \frac{4}{l_0}.$$

□

From now on, we consider a sequence of families $\mathcal{F} = (F_i)_{i\geq 0}$, with parameters (l_i) and (p_i), that has both Properties (P1') and (P2') for the parameters (m_i) and (k_i) defined above. Remark that the integers q_x^i defined in Definition 11.1 satisfy $q_x^i \leq m_i$ thanks to Property P2(i).

The word w that we consider is the word $w_{\mathcal{F}}$ (Definition 11.1) where gadgets have possibly been added between the regular green blocks exactly as in the algorithm of Chapter 10. Since $\mathcal{F}$ satisfies (P1') and (P2'), and the parameters l_i, p_i fall within the range of Theorem 7.10, it can be shown as in Chapter 10 that a chain of w coming from F_i will be parsed in $\geq l_i^{3/2}/54$ red blocks in $0w$ but in only $\leq 3l_i/2$ green blocks. The following two lemmas show Theorem 7.6, i.e., that w satisfies the one-bit catastrophe. We begin with the upper bound on the compression ratio of w, before proving the lower bound for $0w$ in Lemma 11.4.

Lemma 11.3

$\rho_{\sup}(w) = 0$.

Proof. By definition (Definition 7.5),

$$\rho_{\sup}(w) = \limsup_{n\to\infty} \rho(w_{<n}),$$

therefore we need to show that $\rho(w_{<n}) = G(\log G)/n$ tends to zero, where $G = |\mathrm{Dic}(w_{<n})|$ is the number of green blocks in the parsing of $w_{<n}$. Let us evaluate this quantity for a fixed n.

Let j and q be the integers such that the n-th bit of w belongs to the q-th chain of the j-th family, or in other terms, that $w_{<n}$ is the concatenation of the chains coming from $\cup_{i<j} F_i$ and of the first $q-1$ chains of F_j, together with a piece of the q-th chain of F_j.

We first give a lower bound on n as a function of the different parameters. A chain coming from Family F_i is of the form $\mathrm{Pref}_{>q_x^i}(x)$ together with possible gadgets, where $q_x^i \leq m_i \leq \sqrt{l_i}$. Therefore, the size of such a chain is at least

$$\sum_{j=m_i}^{l_i} j \geq \frac{l_i^2 - m_i^2}{2} = \frac{l_i^2}{2} - o(l_i^2).$$

Thus, using $l_j = 2l_{j-1}$, we get:

$$n + o(n) \geq \sum_{i=0}^{j-1} |F_i| \frac{l_i^2}{2} + (q-1)\frac{l_j^2}{2} \geq \frac{l_{j-1}^2}{2}(|F_{j-1}| + q)$$

as soon as $2(q-1) \geq q/2$, that is, $q \geq 2$. (We shall take care of the case $q = 1$ below.)

On the other hand, when all possible gadgets are added, each chain has a size at most $5l_i^2/8 + o(l_i^2)$ (see Section 10.2). Using the fact that $|F_{i+1}| \sim |F_i|^{\sqrt{2}}$ (i.e., the growth of the sequence $(|F_i|)_{i\geq 0}$ is more than exponential), we obtain the following upper bound:

$$n - o(n) \leq \frac{5}{8}\left(\sum_{i=0}^{j-1} |F_i| l_i^2 + q l_j^2\right) \leq \frac{5}{8}(2|F_{j-1}|l_{j-1}^2 + q l_j^2) = \frac{5}{4}(|F_{j-1}| + 2q) l_{j-1}^2.$$

In particular,

$$\log G \leq \log n \leq 2 \log |F_{j-1}|.$$

Let us now bound the number of green blocks. A chain coming from F_i, with gadgets, is parsed in at most $3l_i/2$ blocks. Hence

$$G \leq \frac{3}{2}\left(\sum_{i=0}^{j-1} |F_i| l_i + q l_j\right) \leq \frac{3}{2}(2|F_{j-1}|l_{j-1} + 2q l_{j-1}) = 3l_{j-1}(|F_{j-1}| + q).$$

We can now bound the compression ratio of $w_{<n}$:

$$\rho(w_{<n}) = \frac{G \log G}{n} \leq \frac{6}{l_{j-1}} \cdot 2 \log |F_{j-1}| \xrightarrow[n\to\infty]{} 0.$$

Finally, for the case $q = 1$, looking back at the inequalities above we have: $n + o(n) \geq |F_{j-1}| l_{j-1}^2/2$ and $G \leq 3l_{j-1}(|F_{j-1}| + 1)$, thus $\rho(w_{<n})$ again tends to zero. □

Finally we turn to the lower bound on the compression ratio of $0w$.

Lemma 11.4

$\rho_{\inf}(0w) \geq 2/(1215\gamma)$.

Proof. Define j and q as in the proof of Lemma 11.3: we want to give a lower bound on $(R \log R)/n$, where $R = |\mathrm{Dic}(0w_{<n})|$ is the number of red blocks in the parsing of $0w_{<n}$. The upper bound for n given there still hold:

$$n - o(n) \leq \frac{5}{4}(|F_{j-1}| + 2q) l_{j-1}^2.$$

Let us now give a lower bound on R. Suppose for now that $q \geq 4$, so that $2\sqrt{2}(q-1) \geq 2q$. The proof of Theorem 7.10 in Chapter 10 shows that each chain

coming from a family F_i is parsed in at least $\epsilon l_i^{3/2}$ red blocks, where $\epsilon = 1/54$. Hence:

$$\begin{aligned} R &\geq \sum_{i=0}^{j-1} \epsilon |F_i| l_i^{3/2} + \epsilon(q-1) l_j^{3/2} \\ &\geq \epsilon(|F_{j-1}| l_{j-1}^{3/2} + 2\sqrt{2}(q-1) l_{j-1}^{3/2}) \\ &\geq \epsilon(|F_{j-1}| + 2q) l_{j-1}^{3/2}. \end{aligned}$$

Therefore,

$$\frac{R \log R}{n} \geq \frac{4\epsilon}{5} \cdot \frac{\log |F_{j-1}|}{\sqrt{l_{j-1}}} \sim \frac{4\epsilon}{5 \times 9\gamma}.$$

In the case $q \leq 3$, we have $q << |F_{j-1}|$ and the same bound holds. □

Conclusion and perspectives

> « Le monde a commencé sans l'homme et il s'achèvera sans lui. Les institutions, les mœurs et les coutumes, que j'aurai passé ma vie à inventorier et à comprendre, sont une efflorescence passagère d'une création par rapport à laquelle elles ne possèdent aucun sens, sinon peut-être de permettre à l'humanité d'y jouer son rôle. »
>
> Claude Lévi-Strauss. *Tristes Tropiques.*

We conclude this thesis by giving some perspectives and open questions for future works, that are linked to what has been presented throughout the manuscript.

Part one: non commutative arithmetic circuits

- **Lower bounds.** We have shown some lower bounds using the partition method—that is, by considering $\mathrm{rank}[f, \Pi]$ for some polynomial f and partition Π. This method alone is not sufficient to prove any superpolynomial bound for general non commutative circuits, as there is a polynomial computable by small circuit that is full rank with respect to any partition (see Theorem 1.17). However, the Hankel method—that is, based on the rank of Hankel matrices—does not suffer from such a barrier yet. One can—by using similar ideas to the ones in Chapter 6—show that the rank of the Hankel matrix captures exactly the complexity of general non associative circuits computing a non associative polynomial. Therefore, proving a lower bound on the rank of the Hankel matrix on any non associative polynomial $\tilde{P}$ that is equal to a fixed non commutative polynomial P when viewed in the

associative world, yields a lower bound on P. This is the same as giving a lower bound on the rank of a parametrized Hankel matrix; it might be a way of tackling the problem of finding lower bounds for general non commutative circuits.

- **Polynomial Identity testing.** To the best of our knowledge, all known polynomial time algorithms for PIT are designed for classes of circuits that have at most a constant number of distinct parse trees. While it is intuitive[1] to understand why such a parse trees restriction is a convenient assumption to design good algorithms, we believe that in this phenomenon lies a technical barrier that has to be studied deeper in order to bypass the problem and to design new kinds of algorithms that could work for more general classes of circuits.

Part two: Lempel-Ziv, a "One-bit catastrophe" but not a tragedy

We have privileged "clarity" over optimality, hence constants can undoubtedly be improved rather easily. In that direction, a (seemingly harder) question is to obtain $\rho_{\sup}(w) = 0$ and $\rho_{\inf}(0w) = 1$ in Theorem 7.6. But even if such a catastrophe is not possible, it means there is a maximal constant $\alpha \in]0, 1[$ for which it is possible to get $\rho_{\sup}(w) = 0$ and $\rho_{\inf}(0w) = \alpha$ for some word w; what is this threshold α?

The main challenge though, to our mind, is to remove the gadgets in our constructions. Remark that the construction of Chapter 9 can also be performed with high probability with a random word instead of a de Bruijn sequence (that is what we do in Chapter 10 in a more general way). Thus, if we manage to get rid of the gadgets using the same techniques as presented here, this would mean that the "weak catastrophe" is the typical case for optimally compressible words. Simulations seem to confirm that conclusion.

Another possible direction is to understand the typical expected variation for a fixed compression ratio. For this question, even getting some estimations by simulations seems hard, as sampling uniformly at random from the words of a fixed LZ'78 compression ratio is a challenging task.

[1] Indeed, such a restriction allows to compare "submonomials"—which is an easier task—in order to say something on the "full monomials".

Index

Bibliography

[1] Mateo Aboy, Roberto Hornero, Daniel E. Abásolo, and Daniel Álvarez. Interpretation of the Lempel-Ziv complexity measure in the context of biomedical signal analysis. *IEEE Trans. Biomed. Engineering*, 53(11):2282–2288, 2006.

[2] Eric Allender, Jia Jiao, Meena Mahajan, and V. Vinay. Non-commutative arithmetic circuits: Depth reduction and size lower bounds. *Theor. Comput. Sci.*, 209(1-2):47–86, 1998.

[3] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[4] 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*, pages 25–36, 2009.

[5] Vikraman Arvind, Partha Mukhopadhyay, and Srikanth Srinivasan. New results on noncommutative and commutative polynomial identity testing. *Computational Complexity*, 19(4):521–558, 2010.

[6] Jean Berstel and Christophe Reutenauer. *Noncommutative Rational Series with Applications*. Cambridge University Press, 2011. Second Edition of the English translation. Original title: Les séries rationnelles et leurs langages, Masson 1984.

[7] Symeon Bozapalidis and Olympia Louscou-Bozapalidou. The rank of a formal tree power series. *Theoretical Computer Science*, 27:211–215, 1983.

[8] Jack W. Carlyle and Azaria Paz. Realizations by stochastic finite automata. *Journal of Computer System Science*, 5(1):26–40, 1971.

[9] Steve Chien, Lars Eilstrup Rasmussen, and Alistair Sinclair. Clifford algebras and approximating the permanent. *J. Comput. Syst. Sci.*, 67(2):263–290, 2003.

[10] Steve Chien and Alistair Sinclair. Algebras with polynomial identities and computing the determinant. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 352–361, 2004.

[11] Kenneth Church and Ramesh Patil. Coping with syntactic ambiguity or how to put the block in the box on the table. *Comput. Linguist.*, 8(3-4):139–149, July 1982.

[12] Nathanaël Fijalkow, Guillaume Lagarde, and Pierre Ohlmann. Tight bounds using hankel matrix for arithmetic circuits with unique parse trees. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:38, 2018.

[13] Michel Fliess. Matrices de Hankel. *Journal de Mathématiques Pures et Appliquées*, 53:197–222, 1974.

[14] Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Approaching the chasm at depth four. *J. ACM*, 61(6):33:1–33:16, 2014.

[15] Rohit Gurjar, Arpita Korwar, Nitin Saxena, and Thomas Thierauf. Deterministic identity testing for sum of read-once oblivious arithmetic branching programs. In *30th Conference on Computational Complexity, CCC 2015, June 17-19, 2015, Portland, Oregon, USA*, pages 323–346, 2015.

[16] Christopher Hoobin, Simon J. Puglisi, and Justin Zobel. Relative Lempel-Ziv factorization for efficient storage and retrieval of web collections. *Proc. VLDB Endow.*, 5(3):265–273, November 2011.

[17] Pavel Hrubeš, Avi Wigderson, and Amir Yehudayoff. Non-commutative circuits and the sum-of-squares problem. *Journal of the American Mathematical Society*, 24(3):871–898, 2011.

[18] Mark Jerrum and Marc Snir. Some exact complexity results for straight-line computations over semirings. *J. ACM*, 29(3):874–897, 1982.

[19] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 355–364, 2003.

[20] 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*, pages 45:1–45:13, 2017.

[21] Neeraj Kayal, Nutan Limaye, Chandan Saha, and Srikanth Srinivasan. An exponential lower bound for homogeneous depth four arithmetic formulas. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 61–70, 2014.

[22] Mrinal Kumar and Shubhangi Saraf. On the power of homogeneous depth 4 arithmetic circuits. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 364–373, 2014.

[23] Guillaume Lagarde, Nutan Limaye, and Srikanth Srinivasan. Lower bounds and PIT for non-commutative arithmetic circuits with restricted parse trees (extended version). *To appear in Computational Complexity*.

[24] Guillaume Lagarde, Nutan Limaye, and Srikanth Srinivasan. Lower bounds and PIT for non-commutative arithmetic circuits with restricted parse trees. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, pages 41:1–41:14, 2017.

[25] Guillaume Lagarde, Guillaume Malod, and Sylvain Perifel. Non-commutative computations: lower bounds and polynomial identity testing. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:94, 2016.

[26] Guillaume Lagarde and Sylvain Perifel. Lempel-ziv: a "one-bit catastrophe" but not a tragedy. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1478–1495, 2018.

[27] 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*, SEQUENCES '97, pages 123–135, Washington, DC, USA, 1997. IEEE Computer Society.

[28] Abraham Lempel and Jacob Ziv. On the complexity of finite sequences. *IEEE Trans. Information Theory*, 22(1):75–81, 1976.

[29] Nutan Limaye, Guillaume Malod, and Srikanth Srinivasan. Lower bounds for non-commutative skew circuits. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:22, 2015.

[30] Nutan Limaye, Guillaume Malod, and Srikanth Srinivasan. Lower bounds for non-commutative skew circuits. *Theory of Computing*, 12(1):1–38, 2016.

[31] María Lopéz-Valdés. Lempel-Ziv dimension for Lempel-Ziv compression. In *Proceedings of the 31st International Conference on Mathematical Foundations of Computer Science*, MFCS'06, pages 693–703, Berlin, Heidelberg, 2006. Springer-Verlag.

[32] María López-Valdés and Elvira Mayordomo. Dimension is compression. *Theory Comput. Syst.*, 52(1):95–112, 2013.

[33] Jack H. Lutz. Dimension in complexity classes. *SIAM J. Comput.*, 32(5):1236–1259, 2003.

[34] Jack H. Lutz and Elvira Mayordomo. Computing absolutely normal numbers in nearly linear time. *CoRR*, abs/1611.05911, 2016.

[35] Guillaume Malod. *Polynômes et coefficients. (Polynomials and coefficients)*. PhD thesis, Claude Bernard University Lyon 1, France, 2003.

[36] Guillaume Malod and Natacha Portier. Characterizing valiant's algebraic complexity classes. *J. Complexity*, 24(1):16–38, 2008.

[37] Elvira Mayordomo, Philippe Moser, and Sylvain Perifel. Polylog space compression, pushdown compression, and Lempel-Ziv are incomparable. *Theory Comput. Syst.*, 48(4):731–766, 2011.

[38] Noam Nisan. Lower bounds for non-commutative computation (extended abstract). In *STOC*, pages 410–418, 1991.

[39] Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity*, 6(3):217–234, 1997.

[40] Sylvain Perifel. *Complexité algorithmique*. Ellipses, 1st edition, 2014. `http://www.liafa.univ-paris-diderot.fr/~sperifel/livre_complexite.html`.

[41] Larry A. Pierce II and Paul C. Shields. *Sequences Incompressible by SLZ (LZW), Yet Fully Compressible by ULZ*, pages 385–390. Springer US, Boston, MA, 2000.

[42] Ran Raz and Amir Shpilka. Deterministic polynomial identity testing in non-commutative models. *Computational Complexity*, 14(1):1–19, 2005.

[43] Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

[44] Amir Shpilka and Avi Wigderson. Depth-3 arithmetic circuits over fields of characteristic zero. *Computational Complexity*, 10(1):1–27, 2001.

[45] Yi Zhang, Junkang Hao, Changjie Zhou, and Kai Chang. Normalized Lempel-Ziv complexity and its application in bio-sequence analysis. *Journal of Mathematical Chemistry*, 46(4):1203–1212, November 2009.

[46] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theor.*, 23(3):337–343, September 1977.

[47] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theor.*, 24(5):530–536, September 1978.