UNIVERSITÉ PARIS XIII SORBONNE PARIS NORD
École doctorale Sciences Technologies Santé Galilée

Théorie de la complexité : pot-pourri

Complexity theory: medley

THÈSE DE DOCTORAT
présentée par

Ulysse Léchine
Laboratoire d'Informatique Paris Nord

pour l'obtention du grade de
DOCTEUR EN Informatique

soutenue le 3 décembre 2024 devant le jury dexamen constitué de:

Présidente du jury:
CREIGNOU Nadia – Université Aix-Marseille
Rapporteurs:
SANTHANAM Rahul – Oxford University
ROMASHCHENKO Andrei – CNRS
Examinateurs:
LIMAYE Nutan – IT university Copenhagen
HIRAHARA Shuichi – National Institute of Informatics
PERIFEL Sylvain – Université Sorbonne Paris Nord
BOURNEZ Olivier – Ecole Polytechnique
Directeur :
SEILLER Thomas – LIPN
Co-encadrant :
COUTEAU Geoffroy – IRIF

Abstract

In this thesis we study power of computation in different models, our approach pertains to complexity theory. Our contributions span multiple topics. We show that probabilistic automaton may not ouput non-normal number from normal inputs. We analyze possible extensions of Agafonov′ theorem. We establish a new time hierarchy for Kolmogorov time bounded complexity. In doing so we analyze a new kind of combinatorial game called novelty games. We present a new proof of Mulmuley′ theorem that NC is different from P in the algebraic setting.

Résumé

Dans cette thèse, nous étudions la puissance de calcul de différents modèles, notre approche se rattache à la théorie de la complexité. Nos contributions couvrent plusieurs sujets. Nous montrons que les automates probabilistes peuvent ne pas produire de nombres non normaux à partir d'entrées normales. Nous analysons les extensions possibles du théorème d'Agafonov. Nous établissons une nouvelle hiérarchie pour la complexité de Kolmogorov temporelle. Pour ce faire nous introduisons un nouveau type de jeux combinatoriaux appelés jeux de trouvailles. Nous présentons une nouvelle preuve du théorème de Mulmuley selon lequel NC est différent de P dans les modèles de calcul algébrique.

Remerciements

J'aimerais remercier par ordre chronologique les personnes suivantes : mon père pour l'amour des mathématiques, ma mère pour l'amour de ce qui n'est pas mathématique, Mamé pour les moments de pacha loin de Paris, mon professeur de mathématiques de sup M. Gilles pour m'avoir musclé mathématiquement, mon professeur d'option info M. Morcrette, Léonard sans qui l'informatique à l'ENS aurait été un long moment, mes professeurs de l'ENS et particulièrement Pascal Koiran, Omar Fawzi, Eugenia Oshurko pour les cours de complexité, Shuichi Hiarahara pour le stage, Damien Woods for the very interesting work outside complexity theory before the thesis, Damien Stehlé qui m'a encouragé à faire une thèse, Cosmo qui m'a montré un monde au delà de la thèse, Thomas Seiller et Geoffroy Couteau qui m'ont dit oui de suite pour la thèse, je les remercie aussi pour les conseils et le travail effectué ensemble au cours de ces trois ans, Jakob pour l'accueil et le travail effectué au Danemark, Esaie qui ne peut pas supporter mes remerciements les plus sincères, Eliana, Guillaume et Will pour les moments de camaraderie. Corentin Henriet et Tim Kuniskyi pour leurs idées sur les jeux de trouvailles, les rapporteurs de cette thèse pour leur temps et leur intérêt. Enfin Emma pour m'avoir accompagné sur la fin de cette thèse.

All STEM thesis must start with a concise description of the problem, an on-the-nose jest and some meta comment. I've never been a funny guy but I do have 20 dollars[1].

> **Author:** *ChatGPT, explain concisely what is complexity theory and add a little joke.*
>
> **ChatGPT:** Complexity theory is a branch of theoretical computer science that studies the resources required to solve computational problems, such as time and space. By classifying problems into complexity classes, such as P, NP, and PSPACE, complexity theory helps us understand which problems are feasible to solve and which are inherently intractable. It provides a framework for analyzing the efficiency of algorithms and the boundaries of what can be computed efficiently. In essence, complexity theory asks: "How hard is it to solve this problem?" And as any computer scientist might quip, sometimes the answer is, "It's so complex, even our best algorithms need a coffee break!"

Now we can start with the thesis.

According to Merriam Webster computing means "to determine especially by mathematical means". Computability theory asks what functions can be computed. Such a question is only relevant within the context of a specific computation model: my table calculator can probably compute less functions than my Macbook pro[2] can. According to the Church-Turing (CT) thesis, which is a property true if and only if the physical laws of our universe are computable by Turing machines, my Macbook pro can actually compute all functions that can ever be computed by any device in the universe. Stuff my macbook can compute is for instance : the maximal element of a list, the n-th prime number, the star-height of a regular expression. Stuff my macbook cannot compute is for instance: any function of $\{0,1\}^* \mapsto \{0,1\}$ with probability 1. Also it cannot compute if a given set of Wang tiles paves the 2D plane. Actually the set of computable functions is not specific to my Macbook, any device which is Turing complete has the exact same set of functions. As a matter of fact, consumer computers have been Turing complete since many decades. This then begs the question: why do people buy new computers ?

Maybe you yourself know the answer, the catch is that modern computers run faster than older ones. People care about the time taken to run a program but computability theory does not, hence the need for a new field: Complexity theory. The time taken to compute a function, or more generally the resources needed to compute a function is the object of study of Complexity theory. It is obvious that the time taken to compute a function will depend on the machine you use to compute it. To level the playing field we thus devise theoretical models and we only compare different algorithms. The efficiency of an algorithm can be measured as the number of steps it takes to finish on our theoretical model. This number of steps may obviously depend upon the input size, the running time of an algorithm is therefore a function of $\mathbb{N} \mapsto \mathbb{N}$. Now given a specific problem we may ask what is the best algorithm for this problem (in the sense of which one runs the fastest). Maybe for some problems the best algorithm run in time $O(n)$ and for some others in $O(2^n)$. Thus given a problem we may speak of its *inherent* difficulty. Now remember that the difficulty of a problem only makes sense within a certain framework/theoretical model. The task of complexity theorists is to study the difficulty of problems in various models. It shall hence be what we try to do in this thesis.

---

[1] not funded by the lab

[2] funded by the lab

We present the three main contributions of this PhD.

- In chapter 1 we introduce notions and give relevant background to the reader.
- In part I we extend Agafonov' theorem to probabilistic selectors and analyze possible extensions of the Agafonov theorem to non-regular languages. **[What is randomness when you have finite memory ?]**
- In part II we show an amelioration of the time hierarchy theorem for time bounded Kolmogorov complexity and we introduce novelty games. **[How does more time allow writing more complex strings?]**
- In part III we establish a simple proof of Mulmuley' theorem stating that $\mathsf{P}^{alg} \neq \mathsf{NC}^{\mathsf{alg}}$. **[How does parallelism help in algebraic models?]**

There were 4 papers written during this thesis , the content of part I and III were published [LSS24], [Léc23] (MFCS, FTTCS), the content of II is pending publication.

Contents

1 Complexity preliminaries 9
1.1 Boolean computation . . . 9
1.1.1 Turing machines . . . 9
1.1.2 RAM model . . . 13
1.1.3 Circuits . . . 14
1.2 Algebraic computation . . . 16

I Extensions and limitations of the Agafonov theorem 18
1.3 Introduction to automata and normal sequences . . . 20
1.3.1 Deterministic automata . . . 20
1.3.2 Probabilistic automata . . . 22
1.3.3 Normal sequences . . . 24

2 Extension of Agafonov′ theorem to probabilistic automatas [LSS24] 27
2.1 Introduction . . . 27
2.2 Preliminaries and notation . . . 29
2.2.1 Technical lemmas about normality . . . 29
2.3 Dyadic case . . . 32
2.4 Rational selector . . . 36
2.4.1 Determinisation . . . 36
2.4.2 Rational selectors preserve normality . . . 39

3 Non-extensions of the Agafonov theorem 46
3.1 Return time . . . 47
3.2 Languages of unbounded *ERT* . . . 50
3.3 Almost regular languages of finite moments . . . 55
3.4 Languages of bounded return time . . . 57

II Kolmogorov complexity 59

4 Time hierarchy for Kolmogorov complexity 60
4.1 Introduction . . . 60
4.1.1 Outline . . . 66
4.2 Definitions . . . 66
4.2.1 Notations . . . 66
4.2.2 Kolmogorov related notions . . . 67
4.3 Goal and general strategy . . . 70
4.4 Novelty games . . . 72
4.4.1 Definition . . . 73

4.4.2 General remarks about Novelty games 73
4.4.3 Relation between novelty games and time hierarchy for K^t 75
4.4.4 Resolution of $\mathcal{G}(2,2)$ case 76
4.4.5 Graph strategies 77
4.5 General Strategy for (p,k) 82
4.5.1 Strategy for (3,2) 82
4.5.2 General strategy for (p,k) 85
4.6 Formal description and analysis of the programs 87
4.6.1 One try variant 87
4.6.2 Multiple tries variant 88
4.7 Results 89
4.8 Additional links and musings for future work 93
4.8.1 Links with the time hierarchy theorem with advice 93
4.8.2 Establishing lower bounds for any strategy 97

III Algebraic computation [Léc23] 100

5 NC vs P in the algebraic world 101
5.1 Overview 101
5.2 Introduction 103
5.2.1 Previous work 103
5.2.2 Our result 104
5.3 Equivalence between PRAMS and algebraic circuits 106
5.4 Limitations of log-depth algebraic circuits 109
5.5 Problems not in $\mathsf{NC}^{\mathsf{alg}}$ 112
5.5.1 Bit extraction 113
5.5.2 Bijections from $\mathbb{N}$ to $\mathbb{N}^2$ 115
5.5.3 Maxflow 115

A Reasonable functions 122

Chapter 1

Complexity preliminaries

Notations $\mathbb{N}$ denotes the set of integers, $\mathbb{N}_{>0}$ denotes the set of positive integers $\mathbb{Z}$ denotes the set of signed integers, $\mathbb{Q}$ the set of rational numbers, $\mathbb{R}$ denotes the set of real numbers. For any $a, b \in \mathbb{N}$, $[\![a, b]\!]$ denotes the set of integers i s.t. $a \leq i \leq b$. The size of a program p, a string w or a set S are respectively denoted by $|p|, |w|, |S|$, for sets we may also use Card S or $\#S$. There is a natural bijection between programs, Turing machines, binary strings, integers, thus when necessary we conflate programs with their binary representation, Turing Machines, and integer. Σ is used to denote an alphabet, Σ^* the words over that alphabet. $\{0, 1\}$ is the alphabet composed of letters 0 and 1. ϵ is the empty string. Let v and w be two words, vw, $v \cdot w$ or $v||w$ are all defined as the concatenation of v and w. A language L is a subset of Σ^*.

Let f and g be two functions of $\mathbb{N} \mapsto \mathbb{N}$, we write

$$f = o(g) \overset{\text{def}}{\equiv} \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

$$f = O(g) \overset{\text{def}}{\equiv} \exists c > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, |f(n)| \leq c\,|g(n)|$$

$$f = \Theta(g) \overset{\text{def}}{\equiv} \exists c_1, c_2 > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, c_1\,|g(n)| \leq |f(n)| \leq c_2\,|g(n)|$$

$$f = \omega(g) \overset{\text{def}}{\equiv} \forall c > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, |f(n)| > c\,|g(n)|$$

1.1 Boolean computation

1.1.1 Turing machines

Definition of Turing machines

We very promptly define Turing machines, for a more detailed explanation we refer the reader to introductory books on the subject [AB06].

Definition 1.1: Turing machine(TM)

A multi-tape Turing machine is a machine composed of k tapes and k heads for a certain $k \in \mathbb{N}$. It is formally defined as a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where:

- Q is a finite set of states.
- Σ is the input alphabet, excluding the blank symbol $\sqcup$.
- Γ is the tape alphabet, where $\Sigma \subseteq \Gamma$ and $\sqcup \in \Gamma$.
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$ is the transition function, with k tapes and heads.
- $q_0 \in Q$ is the initial state.
- $q_{\text{accept}} \in Q$ is the accept state.
- $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{accept}} \neq q_{\text{reject}}$.

Each of the k heads operates independently but synchronously on their own tape initially filled with blank symbols, with the transition function δ determining the new state, the symbols to write on each tape, and the movement direction (left, right, or stay) for each head. When one heads reaches state q_{accept} or q_{reject} the TM stops.

In the following we will only reason over Turing machines whose input alphabet is $\{0;1\}$.

Definition 1.2: Output of a Turing machine

Let M be a Turing machine whose input alphabet is $\{0;1\}$, let x be a bitstring of $\{0,1\}^*$. Consider the computation performed by M until it stops when x is initially written on its first tape. $M(x) \in \{0,1\}^*$ is defined as the bitstring written on M first tape when M reaches the accepting state and stops.

Turing machines can be endowed with oracles to grant them additional computing power. An oracle is any function in $\{0,1\}^* \mapsto \{0,1\}^*$, which the TM may query by writing the desired input on a specific oracle tape.

Definition 1.3: Oracle Turing machines

Let $A : \{0,1\}^* \mapsto \{0,1\}^*$ be an oracle, a machine M with access to A noted M^A, is a Turing machine with a special oracle tape and a special query state. Whenever M^A gets to the query state with x written on the oracle tape, the content of the oracle tape gets replaced with $A(x)$.

If a property stated for TM machines also holds for all TM with oracle access then the result is said to relativize. Several important problems in complexity theory, e.g. $\mathsf{P} vs. \mathsf{NP}$, are known not to relativize, meaning for some oracle $A, \mathsf{P}^A = \mathsf{NP}^A$ and for some oracle $B, \mathsf{P}^B \neq \mathsf{NP}^B$. Hence the need for proof techniques which do not relativize.

Encodings

Definition 1.4: Encoding of a Turing machine #

Let M be a Turing machine. The bitstring $\#M \in \{0,1\}^*$ is an encoding of M such that one can uniquely retrieve M from $\#M$. We leave out the specific implementation details. In the following, we will conflate Turing machines with their binary encodings.

It will happen, particularly in section 4, that we need to encode and retrieve a Turing machine ($\#M$) and its input (x) from an element of $\{0,1\}^*$, to that end we introduce encoding of tuples.

Definition 1.5: Prefix-free codes

We define functions E_1 and E_2 from $\{0,1\}^* \mapsto \{0,1\}^*$ as $E_1(x) = 1^{|x|}0x$ and $E_2(x) = E_1(|x|)x$.

Theorem 1.1: Prefix free code [LV19]

E_2 is a prefix code and $|E_2(x)| = x + 2log(|x|) + 1$. E_2 is being a prefix code means that $\forall x, y \in (\{0,1\}^*)^2$ one can uniquely retrieve x and y from the string $E_2(x)y$.

Definition 1.6: Encoding of tuples

Let $x, y, z \in \{0,1\}^*$. We define $< x, y > \overset{\text{def}}{=} E_2(x)y$ and $< x, y, z > \overset{\text{def}}{=} E_2(x)E_2(y)z$. This definition is naturally extended to any number of bitstrings inside $< \cdot >$

Run-time and simulation

Definition 1.7: Running time of a TM

A Turing machine M runs in time $T : \mathbb{N} \mapsto \mathbb{N}$ if $\exists c, \forall n, \forall x \in \{0,1\}^n$, $M(x)$ stops before $cT(n)$ steps, in which case we may write $M \in DTIME(T)$ (and sometimes abusively $M \in DTIME(T(n))$)

A fundamental theorem in computer science is that there is a universal Turing machine $\mathbb{U}$ which can, given a description of any Turing machine M encoded as $\#M$, and any input x, simulate the behavior of M on x. In other words, $\mathbb{U}(\#M, x) = M(x)$. Additionally, this simulation incurs a time slowdown of a logarithmic factor, such that if M runs in time $T(n)$ on x, then $\mathbb{U}(\#M, x)$ runs in time $O(T(n) \log T(n))$. To this day we have no lower bound on the optimal slowdown factor.

Theorem 1.2: Universal Turing Machine

There is a universal Turing machine $\mathbb{U}$ such that for any TM M, and any string $x \in \{0,1\}^*$, we have that $\mathbb{U}(< \#M, x >) = M(x)$. Moreover, if M runs in time T where $\forall n, T(n) \geq n$, then $\mathbb{U}(\#M, \cdot)$ runs in time $CT \log T$ where the constant depends only upon the number of tapes and alphabet's size of M .

We admit this theorem, one can find its proof in [AB06]. [1]

Complexity Classes

Definition 1.8: Time classes

Given a function $f : \{0,1\}^* \mapsto \{0,1\}$ and a time bound function $T : \mathbb{N} \mapsto \mathbb{N}$, we say that $f \in \mathsf{DTIME}(T)$ if it is computed by a TM, i.e. there is a TM M such that $\forall x \in \{0,1\}^*, M(x) = f(x)$ and $M \in \mathsf{DTIME}(T)$
Given an oracle $A : \{0,1\}^* \mapsto \{0,1\}^*$, we write $f \in \mathsf{DTIME}^A(T)$ if f is there exists a TM M such that f is computed by M^A.

The next complexity class is often presented as the class of easy problems.

Definition 1.9: Complexity class P

The complexity class P consists of all decision problems (languages) that can be solved by a deterministic Turing machine in polynomial time. Formally, P is defined as:

$$\mathsf{P} = \bigcup_{k \in \mathbb{N}} \mathsf{DTIME}(n^k)$$

Next we introduce time-constructible function. Time constructible functions are functions which are nice in the sense that the time required to compute $f(n)$ is not much greater than $f(n)$. All usual functions bigger than $n \mapsto n$ are time constructible. Functions not time constructible include the busy beaver function which grows faster than any computable function, or the halting function which given n outputs 1 if the n-th Turing machine stops 0 otherwise.

Definition 1.10: Time-Constructible Function

A function $f : \mathbb{N} \to \mathbb{N}$ is called time-constructible if there exists a deterministic Turing machine M such that, on input 1^n, M halts in $O(f(n))$ steps and outputs $f(n)$ in binary.

The Time Hierarchy Theorem is a fundamental result in computational complexity theory that demonstrates the existence of a strict hierarchy of complexity classes based on time bounds. Specifically, it shows that given more computational time, Turing machines can solve strictly more problems. The proof of this theorem is made through a technique called diagonalization which is a staple of many proofs in complexity theory. In chapter 4 we extend this technique.

Theorem 1.3: Time Hierarchy Theorem

Let f and g be time-constructible functions such that $\forall n, f(n) \geq n$ and $g = \omega(f \log f)$. Then there exists a language L that can be decided by a deterministic Turing machine in time g but not in time f. Formally,

$$\exists L \in \mathsf{DTIME}(g) \quad \text{such that} \quad L \notin \mathsf{DTIME}(f).$$

---

[1] Although in their proof they don't require $T(n) > n$ it seems necessary to us as you need to read the description of $\#M$ to have a constant C only dependent upon the number of tapes and alphabet's size of M.

Proof

Let f and g be as in the theorem. We construct a Turing machine M such that M runs in time g and such that $L(M) \notin \mathsf{DTIME}(f)$ (note that by definition $L(M) \in \mathsf{DTIME}(g)$). For any $w \in \{0,1\}^*$ we define M_w as the only Turing machine such that $\#M_w = w$. The Turing machine M on input w simulates for $g(|w|)$ *simulation* steps the universal Turing machine $\mathbb{U}(< w, w >) = \mathbb{U}(< \#M_w, w >)$ and calls a the output if there was any (note that it simulates M_w on input w for at least $\omega(f(|w|)))$ steps. If a belong to $\{0,1\}$ M returns $\neg a$, otherwise it returns 0.

Now we prove that for any $c \in \mathbb{R}$, $L(M)$ can't be decided by any machine N running for cf steps. Suppose for a contradiction that machine N runs for $cf(n)$ steps (at most) and always outputs 0 or 1 (otherwise it does not recognize a language). There is a constant c' such that for any $x \in \{0,1\}^*$, $n \overset{\text{def}}{=} |x|$, universal machine $\mathbb{U}$ runs in less than $c'cf \log f(n)$ on input $(\#N_i, x)$. By noticing that there are infinitely many strings w representing the Turing machine N there must exist n_0 such that there are infinitely many Turing machines $(N_i)_{i \geq n_0}$ such that $N_i = N$ and such that $|\#N_i| = i$. Take i such that $\forall j \geq i, g(j) > cc'f \log f(j)$ (it exists because $g = \omega(f \log f)$). Then we have a contradiction because:

$$\begin{aligned} &\text{Machine } N_i \text{ runs for less than } cf(i) \text{ steps on input } \#N_i \\ &\text{Thus simulating } \mathbb{U} \text{ on } < M_{\#N_i}, \#N_i > \text{ takes less than } cc'f \log f(i) < g(i) \text{ steps} \\ &\text{Thus } M(\#N_i) = \neg \mathbb{U}(< M_{\#N_i}, \#N_i >) = \neg N_i(\#N_i) = \neg N(\#N_i) \end{aligned}$$

Thus $L(M) \neq L(N)$

1.1.2 RAM model

Definition of RAMs The Random-Access Machine (RAM) is a theoretical computational model used in the analysis of algorithms and computational complexity. The main difference with the TM model is that getting to a location in memory is done in constant time in the RAM model (at least in this thesis), and several algebraic operations are considered to take unit cost in the RAM model.

Definition 1.11: RAM machine

A RAM is composed of infinitely many registers indexed by integers, it can execute a finite sequence of instructions labelled from 1 to k for a certain $k \in \mathbb{N}$. The instructions are of the following kinds:

1. $w = u \circ v$, where w denotes a memory location, u and v denote either memory locations or constants. Finally, $\circ$ denotes a binary operation $+$, $-$, $\times$ or $\oplus$ (denoting bit by bit xor-ing).
2. `go to` l, where l is the label of some instruction.
3. `if` $u \, \Delta \, 0$ `go to` l, where u denotes a memory location and Δ denotes either $<$, $\leq$, or $=$.
4. $u := v$: Store the contents of the memory location v into the location u.
5. $u :=\uparrow v$ (indirect reference): Treat the value in the memory location v as a pointer. Store into the memory location u the value in the pointed memory location.
6. `stop`.

When computing with a RAM a function from $\{0,1\}^n$ to $\{0,1\}^m$, we may say that the inputs are written on the first n registers and the outputs are written on the first m registers at the end of the computation. The implementations detail have no bearing on the expressivity of the RAM model. Each instruction listed in definition 1.11 takes unit time. From a computability standpoint there are no differences between RAM and Turing Machines.

Theorem 1.4: Equivalence between RAM and TM

Any function which can be computed by a RAM can also be computed by a Turing Machine and vice versa.

Contrary to Turing machines, there exists a universal RAM which can simulate any other RAM with a constant time slowdown.

Theorem 1.5: Universal RAM

There exists a universal RAM uR such that for any RAM R and any input x, $uR(< \#R, x >) = R(x)$ and if R runs in T steps uR runs in $O(T)$ steps. ($\#R$ is some encoding of R)

1.1.3 Circuits

Definition of Circuits We quickly and without details recall the notion of Boolean circuits for the reader.

Definition 1.12: Boolean Circuits

A circuit is a computational model that implements a Boolean function from $\{0,1\}^n \mapsto \{0,1\}^m$ for some $n, m \in \mathbb{N}^2$.
Formally, a Boolean circuit C is a quadruplet (G, Op, I, O) where:

- $G = (V, E)$ is a directed acyclic graph (DAG). The vertices V represent the logic gates, and the edges E represent connections between these logic gates.
- $I = (x_1, \dots, x_n) \in \{0,1\}^n$ denotes the tuple of n input variables of the circuit.
- $O = (y_1, \dots, y_m) \in \{0,1\}^m$ denotes the set of m outputs of the circuit.
- $\mathrm{Op} : V \to \{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}\} \cup I \cup O$ is a function that assigns a logical operation to each vertex or designates a vertex as one of the n inputs of the circuit or as one of the m outputs of the circuit.

Each NOT gate receives exactly one input, the AND and OR gates may receive arbitrarily many.
A boolean circuit computes a function in the obvious way. If C output $(y_1, \dots y_m)$ on input $(x_1, \dots, x_m)$ we may write $C(x_1, \dots, x_m) = (y_1, \dots y_m)$.

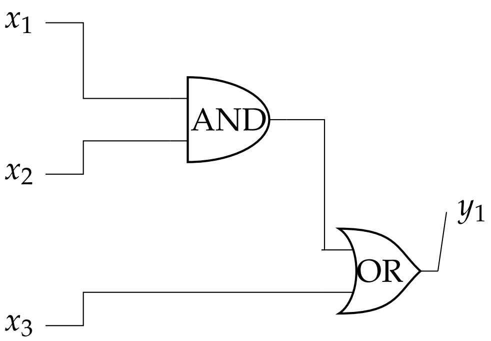


Figure 1.1: Circuit

Definition 1.13: Size, depth and fan-in of Circuits

- The **size** of a circuit is the total number of logic gates it contains.
- The **depth** of a circuit is the length of the longest path (in terms of the number of gates) from any input to any output.
- The **fan-in** of a circuit is the maximum number of input any gate can have.

Circuit complexity classes

Definition 1.14: Uniform sequence of circuits

Let $(C_n)_{n \in \mathbb{N}}$ be a sequence of circuits, it is said to be uniform if there is a Turing machine M such that $\forall n \in \mathbb{N}$, $M(n)$ is a description of circuit C_n.

Definition 1.15: Complexity class P/poly

The complexity class P/poly consists of all languages that can be decided by a non-uniform family of polynomial-size circuits. Formally, a language $L \subseteq \{0,1\}^*$ is in P/poly if there exists a sequence of Boolean circuits of fan-in 2 $\{C_n\}_{n\in\mathbb{N}}$ such that:

- Each circuit C_n has n input variables and exactly one output.
- There exists a polynomial p such that forall n the size of C_n is less than $p(n)$.
- For all $x \in \{0,1\}^n$, $x \in L$ if and only if $C_n(x) = 1$.

In part 5 we study the power of paralelizing computations, to that end we introduce the class NC which is intended to contain all problems solvable quickly in parallel (for a certain notion of quickness). More precisely the class NC represents all problems which you can solve in poly-logarithmic time if you have polynomially many processors working in parallel over a shared memory.

Definition 1.16: Complexity class NC

The complexity class NC (Nick's Class) consists of all languages that can be decided by a family of uniform Boolean circuits with polynomial size and polylogarithmic depth. Formally, a language $L \subseteq \{0,1\}^*$ is in NC if there exists a sequence of Boolean circuits of fan-in 2 $\{C_n\}_{n\in\mathbb{N}}$ such that:

- $(C_n)_{n\in\mathbb{N}}$ is a uniform sequence of circuits
- Each circuit C_n has n input variables and exactly one output.
- The size of C_n (i.e., the number of gates) is bounded by a polynomial in n.
- The depth of C_n (i.e., the length of the longest path from an input to the output) is bounded by $O(\log^k n)$ for some constant k.
- For all $x \in \{0,1\}^n$, $x \in L$ if and only if $C_n(x) = 1$.

This means that for every input length n, there is a corresponding polynomial-size, polylogarithmic-depth circuit C_n that correctly decides membership in L for all inputs of that length.

A very old and important question in computer science is determining if $NC = P$. Most researchers believe that $NC \neq P$. At some point the question became so old that people changed the question. Mulmuley showed that if the only operations you are allowed are algebraic $(+, -, \times, <)$ then you can show that $P \neq NC$. In part 5 we give a simpler proof of this theorem.

1.2 Algebraic computation

Algebraic models computation take as input numbers $(\mathbb{N}, \mathbb{R}, \ldots)$ and may perform algebraic operations $(+, \times, \ldots)$ in order to compute functions.

Algebraic circuits We define algebraic circuits in a similar fashion as boolean circuits. The main differences are that algebraic circuits take n real numbers as input, output a tuple of m

real numbers and their gates are operations over the reals.

	Boolean Circuits	**Algebraic Circuits**
Inputs	$x_1, x_2, \ldots, x_n$ in $\{0,1\}^n$ boolean bits	$x_1, x_2, \ldots, x_n$ in $\mathbb{R}^n$ real numbers
Gates	AND, OR, NOT (logical operations)	$+, -, \times, \leq, =,$ constants (arithmetic operations)

Table 1.1: Differences between Boolean circuits and algebraic circuits in terms of inputs and gates.

Definition 1.17: Algebraic circuit

An algebraic circuit is a circuit with gates $+, \times, -, /, =, <, \leq$ of fan-in 2, and constant gates n for all $n \in \mathbb{N}$. Its inputs are real numbers. The comparison gates $=, <, \leq$ return 1 when true, 0 otherwise. The division gate $/$ is over real numbers. All the other gates are interpreted in the obvious way.

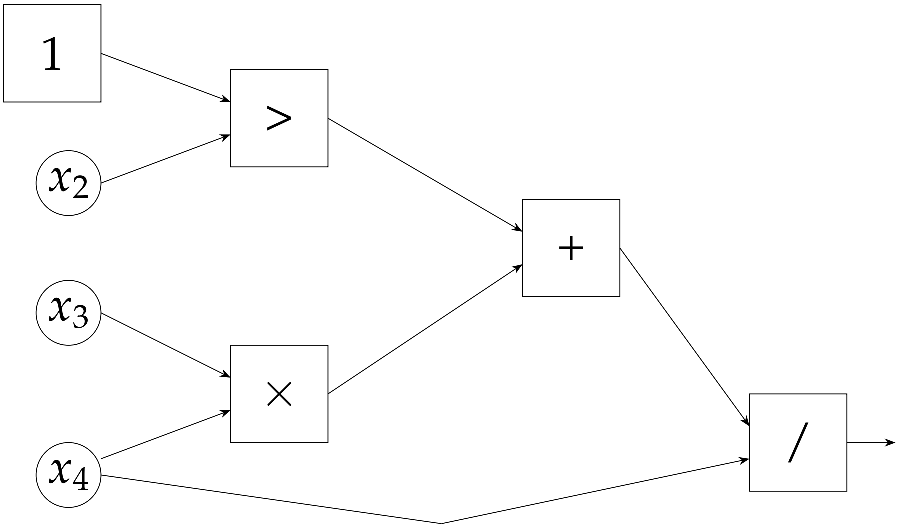


Figure 1.2: Algebraic circuit

Algebraic RAMS In part III we will consider algebraic RAMs where the set of operations and inputs are modified when compared to usual RAMs. In short $\oplus$ (xor-ing) is not longer part of the instruction set, and the inputs are numbers.

Part I

Extensions and limitations of the Agafonov theorem

MATHEMATICALLY INACCURATE STORY TO INTEREST THE READER

Suppose you want to make 1 million dollars 💰 but nobody will read your $\mathsf{P} \neq \mathsf{NP}$ proof 😢, out of spite you build a robot 🤖 to go to the casino 🎰 and play roulette with your life savings 💸. It seems like a bad plan, but you know that at this casino the infinite sequence of black-red 🟥 ⬛ outcome is given by a normal fixed number 0️⃣ 1️⃣. Actually, you know this number very well; it is Champernowne's 🍄 number $0 \cdot 1 \cdot 10 \cdot 11 \cdot 100 \cdot 101 \cdot 110 \cdot 111 \cdot 1000 \cdot 1001 \cdot 1010 \cdot \ldots$. You think you can make money, huh? 🤑 Just wait 😈. The robot, being a bounded sub-region of $\mathbb{R}^3$ 🔢, cannot have infinite ♾️ memory, and hence its computing power is that of a finite automaton. For similar reasons, it may only hold finitely many chips, let's say it can only bet 1 dollar at a time 💵 to fix ideas. Well, Agafonov' theorem says that if this robot plays for an infinitely long time, it may not win money [2]! Upon gaining knowledge of this fact, you send back a new robot equipped with a randomness-generating device (an unobserved electron's spin in a box ⬆️ ⬇️), convinced that you can make money this way. "Hmmmm, ackthuhally no ☝️ 🤓", we proved you cannot in this paper [LSS24] [3], and we talk about it in this chapter 2. Also, we talk about other strategies you could use in 3, maybe you should read it before sending out the next robot 🤖.

---

[2] Well, actually it may win $o(n)$ dollars over n plays but this is way less dramatic than our lie.

[3] Well actually only if the involved probabilities are rational numbers but saying it makes our paper less impressive.

In this chapter we study Agafonov theorem. In section 1.3 we define automata and normal sequences. In chapter 2 we present our result extending Agafonov theorem to probabilistic automata [LSS24]. In chapter 3 we study possible extension of the Agafonov theorem to languages sightly stronger than regular languages.

Notations The set of infinite bitstrings is noted $\{0,1\}^\omega$ its elements will be denoted by Greek letters: α, β, etc. Finite sequences (elements of $\{0,1\}^*$), or *words* are denoted by u, v, w, etc.

If $v, w \in \{0,1\}^*$, $v \cdot w$ denotes the concatenation of v and w; the definition extends to $v \cdot \alpha$ for $\alpha \in \{0,1\}^\omega$ *mutatis mutandis*.

The non-negative integers are denoted by $\mathbf{N}$, and the positive integers by $\mathbf{N}_{>0}$. If $n \in \mathbf{N}$ and $\alpha = a_1 a_2 \cdots \in \{0,1\}^\omega$, we denote by $\alpha_{\leq N}$ the finite sequence $a_1 a_2 \cdots a_N$.

Given a set S we write $\mathrm{Dist}(S)$ the space of probability distributions on S. Given a probability distribution $\delta \in \mathrm{Dist}(S)$, we say that δ is *dyadic* (resp. *rational*) when for all $s \in S$, $\delta(s)$ is a dyadic number (resp. a rational number), that is a number of the form $\frac{p}{2^k}$ for integers p, k.

We consider the standard probability *measure* $\mathrm{Prob}_{\rho \in \{0,1\}^\omega}$ on $\{0,1\}^\omega$ equipped with the least Σ-algebra induced by the cylinder sets $C_w = \{\alpha \mid \exists \alpha' \in \{0,1\}^\omega, \alpha = w \cdot \alpha'\}$ and such that $\mathrm{Prob}_{\rho \in \{0,1\}^\omega} C_w = 2^{-|w|}$ for $w \in \{0,1\}^*$. Elements of $\{0,1\}^\omega$ drawn according to this measure are called *fair random infinite sequence*.

1.3 Introduction to automata and normal sequences

1.3.1 Deterministic automata

Definition 1.18: Deterministic finite automaton (DFA)

An automaton is defined as a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where:

- Q is a finite set of states.
- Σ is a finite input alphabet.
- $\delta : Q \times \Sigma \to Q$ is the transition function, which defines the state transitions based on the current state and input symbol.
- $q_0 \in Q$ is the initial state, where the computation begins.
- $F \subseteq Q$ is the set of final or accepting states.

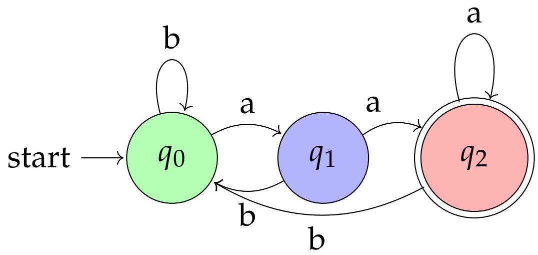


Figure 1.3: Deterministic Finite Automaton: q_0 is the initial state as denoted by start and q_2 is a final state as denoted by double circling

The extended transition function $\delta^* : Q \times \Sigma^* \to Q$ is defined recursively to describe the effect of processing a string of input symbols from any given state. The function δ^* thus extends δ to handle strings of input symbols, rather than just individual symbols.

Definition 1.19: Extended Transition Function

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton (DFA)

- Base case: For any state $q \in Q$, $\delta^*(q, \epsilon) = q$, where ϵ is the empty string.
- Recursive case: For any state $q \in Q$, any string $w \in \Sigma^*$, and any symbol $a \in \Sigma$, $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$.

Here, wa denotes the string w followed by the symbol a.

Definition 1.20: Language recognized by an automaton

Let $A = (Q, \Sigma, \delta, q_0, F)$ be an automaton. The language recognized by the automaton A, denoted $L(A)$, is the set of all strings $w \in \Sigma^*$ such that the automaton, starting from the initial state q_0, reaches an accepting state after reading the entire string w. Formally,

$$L(A) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\},$$

Explicitly, for a string $w = w_1 w_2 \dots w_n$, the path in the automaton is given by the sequence of states $q_0, q_1, q_2, \dots, q_n$ such that $q_{i+1} = \delta(q_i, w_{i+1})$ for all $i \in \{0, 1, \dots, n-1\}$. The string w is accepted if and only if $q_n \in F$.

Definition 1.21: Regular languages

Let L be a language, L is said to be regular if L is recognized by an automaton, i.e. there exists a DFA A such that $L = L(A)$.

We will next define the output of an automaton. We want to use DFAs as a means of extracting bitstrings from words. The bits that are extracted are those right after the DFA has visited an accepting state.

Definition 1.22: Output of an automaton

Let $A = (Q, \Sigma, \delta, q_0, F)$ be an automaton. Let $w \in \Sigma^*$, let $n = |w|$, we my write $w_1, \dots, w_n$. Let I be the set of indices $I = \{i + 1; w_1, \dots, w_i \in L(A)\}$. Let Card $I = m$, we define $A(w) = w_{j_1} \cdot w_{j_2} \cdots w_{j_m}$ where $\forall 1 \leq k \leq m, j_k \in I$ and the j_k are in ascending order.

Here we give an example of the output of automaton presented in figure 1.3 on the string $w = abaaababaaaaba$ The color of each letter corresponds to the state after reading the letter, then $A(w)$ is composed of all letters appearing after a red letter $A(w) = abaab$.

With automata it is easy to have a intuitive interpretation of the output of an automaton : when the automaton reaches a accepting state it outputs the next letter of the word. This definition may actually be extended to languages in general: Given a language L, the bits extracted in w by L are the w_{i+1} where the prefix $w_1 \dots w_i$ belong to L. Note that this extends Definition 1.22.

Definition 1.23: Output of a language

Let L be a language. Let $w \in \Sigma^*$, let $n = |w|$, we my write $w_1, \ldots, w_n$. Let I be the set of indices $I = \{i+1; w_1, \ldots, w_i \in L\}$. We define the string $L(w) \overset{\text{def}}{=} w_{j_1} \cdot w_{j_2} \cdots w_{j_m}$ where $\forall 1 \leq k \leq m, j_k \in I$ and the j_k are in ascending order.

In the following of this chapter we will see that outputs of automata verify some properties (namely Agafonov's theorem 2.1), we will be interested if these properties carry over to languages more general than regular languages.

Definition 1.23 may also be extended to infinite words in the intuitive way.

Definition 1.24: Output of a language for an infinite word

Let L be a language. Let $w \in \Sigma^*$, let $n = |w|$, we my write $w_1, \ldots, w_n$. Let I be the set of indices $I = \{i+1; w_1, \ldots, w_i \in L\}$. Let $|I| = m$, we define $L(w) = w_{j_1} \cdot w_{j_2} \cdots w_{j_m}$ where $j_1 = \min I$ and $\forall 2 \leq k, j_k = \min I \setminus \{j_h; 1 \leq h < k\}$.

Note that the output may be infinite or not (if we go through finitely many accepting states).

1.3.2 Probabilistic automata

A probabilistic automaton is a computational model that extends the concept of a deterministic finite automaton by incorporating probabilistic state transitions.

Definition 1.25: Probabilistic Automaton

A probabilistic automaton is a 5-tuple $P = (Q, \Sigma, \delta, q_0, F)$, where:

- Q is a finite set of states.
- Σ is a finite alphabet.
- $\delta : Q \times \Sigma \times Q \to [0,1]$ is the transition probability function, where $\delta(q, a, q')$ represents the probability of transitioning from state q to state q' upon reading symbol a, and for all $q \in Q$ and $a \in \Sigma$, $\sum_{q' \in Q} \delta(q, a, q') = 1$.
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of accepting (or final) states.

Definition 1.26: Dyadic and rational automata

P is said to be dyadic if the transition probabilities are all dyadic numbers, i.e. $\exists (m,k) \in \mathbb{N}^2, \forall (q, a, q') \in Q \times \Sigma \times Q, \delta(q, a, q') = \frac{m}{2^k}$.
P is said to be rational if the transition probabilities are all rational numbers, i.e. $\forall (q, a, q') \in Q \times \Sigma \times Q, \delta(q, a, q') \in \mathbb{Q}$.

The probability of acceptance of a word $w = w_1 w_2 \ldots w_n \in \Sigma^*$ by the probabilistic automaton P defined in 1.27 is the sum of the probabilities of all possible paths that lead from the initial state q_0 to any accepting state in F after reading the entire word w.

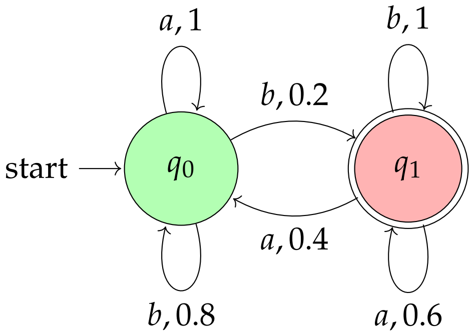


Figure 1.4: Probabilistic Automaton (this one is also rational)

Definition 1.27: Probability of acceptance of a word

Let $P = (Q, \Sigma, \delta, q_0, F)$ be a probabilistic automaton.
We let $\delta : Q \times \Sigma \times Q \to [0, 1]$ be the extended transition function that gives the probability of reaching state q' from the state q after reading the word w. It is recursively defined as follows :

- Base case: $\delta^*(q, \epsilon, q) = 1$ and $\delta^*(q, \epsilon, q') = 0$ for $q' \neq q$, where ϵ is the empty string.
- Recursive case: For $w = w_1 w_2 \dots w_n$ and $a \in \Sigma$,

$$\delta^*(q, wa, q') = \sum_{q'' \in Q} \delta^*(q, w, q'') \cdot \delta(q'', a, q').$$

The probability of acceptance of the word w by P is given by:

$$\Pr[\text{accept } w] = \sum_{q \in F} \delta^*(q_0, w, q),$$

The language recognized by a probabilistic automaton P is the set of all strings for which the automaton ends in an accepting state with a probability greater than a given threshold θ. For instance string *abba* is accepted with probability $1 \cdot 0.2 \cdot 1 \cdot 0.6 + 1 \cdot 0.8 \cdot 0.2 \cdot 0.6 = 0.216$ by the automaton of figure 1.4.

We will not be using this notion in this thesis, on the other hand we will be interested in the output of a probabilistic automaton. The definition is intuitively derived from the DFA one 1.22 : on input w a probabilistic automaton P goes from state to state according to its transition density function, when it reaches a final state it outputs the next bit read and carries on.

Definition 1.28: Output of a probabilistic automaton

Let $P = (Q, \Sigma, \delta, q_0, F)$ be a probabilistic automaton. Let $w \in \Sigma^*$, let $n = |w|$, Let $X_1, ...X_n \in Q^n$ be random variables where X_i is the state in which the automaton is in after reading w_i.
To each sequence of $X_1, ...X_n$ we may define $O(X_1, \dots X_n) = w_{j_1} \dots wj_k$ where k is the number of indices i such that $X_i \in F$ and forall h, X_{j_h} is the h-th X_i to be in F. We precise that $O(X_1, \dots X_n)$ is the empty string in case $k = 0$.
Then we define $P(w)$ to be the random variable $O(X_1, \dots X_n)$

We then extend naturally this definition to infinite sequence. But first we formally define our probability measure on $\{0,1\}^\omega$. Probability theory is not the main focus of this thesis and is only used in a very simple and direct fashion, therefore we do not incorporate all the details but leave some references for the interested reader.

Definition 1.29: Standard Probability Measure on $\{0,1\}^\omega$

Let $\{0,1\}^\omega$ denote the set of all infinite binary sequences. The standard probability measure μ on this set is defined using cylinder sets. A cylinder set $C(s)$ specified by a finite binary string $s = s_1 s_2 \ldots s_n$ consists of all infinite binary sequences that begin with the string s:

$$C(s) = \{\alpha \in \{0,1\}^\omega \mid \alpha_i = s_i \text{ for } i = 1, 2, \ldots, n\}.$$

The measure μ of a cylinder set $C(s)$ is given by:

$$\mu(C(s)) = \left(\frac{1}{2}\right)^n.$$

This measure can be uniquely extended to the σ-algebra generated by the cylinder sets, resulting in the standard probability measure on $\{0,1\}^\omega$. For a detailed explanation of this extension, see [Bil95] and [Kal02].

Thanks to this formal definition we may define fair random infinite sequences, intuitively those correspond to infinite sequences where each bit has probability $1/2$ of being 0 and $1/2$ of being 1.

Definition 1.30: Fair random infinite sequences

An infinite sequence α is said to be a fair (random) infinite sequence if it is drawn according the the probability measure defined in Definition 1.29.
Let A be an event depending on $\rho \in \{0,1\}^\omega$ we note $\mathrm{Prob}_{\rho \in \{0,1\}^\omega}(A(\rho))$ or $\mathrm{P}_{\rho \in \{0,1\}^\omega}(A(\rho))$ to speak of the probability that event A occurs where ρ is an infinite fair random sequence.

Definition 1.31: Output of a probabilistic automaton for infinite sequences

Let $P = (Q, \Sigma, \delta, q_0, F)$ be an automaton. Let $\alpha \in \Sigma^\omega$, let $n = |w|$, Let $X_1, X_2, \ldots \in Q^\omega$ be random variables where X_i is the state in which the automaton is in after reading w_i.
To each sequence of $X_1, ...X_n$ we may define $O(X_1, \ldots X_n) = w_{j_1} \ldots w j_k$ where k is the number of indices i such that $X_i \in F$ and forall h, X_{j_h} is the h-th X_i to be in F
Then we define $P(w)$ to be the random variable $\lim_{n\to\infty} O(X_1, \ldots X_n)$

In the following we will use definition 1.31 as follows: Given a set $A \subset \{0,1\}^\omega$ we are interested in the quantity $\mathbb{P}(P(w) \in A)$. Almost always set A will be the set of all normal sequences in $\{0,1\}^\omega$

1.3.3 Normal sequences

We now turn our attention to normal sequences [Bor09]. A normal sequence over $\{0,1\}$ is an infinite sequence for which every word of length k appears with frequency 2^{-k}. We give a formal definition next before resuming.

Definition 1.32: Occurrence

Let $a = a_1 \cdots a_m$ and $b = b_1 \cdots b_n$ be finite sequences such that $n < m$. An *occurrence* of b in a is an integer i with $1 \leq i \leq m - n + 1$ such that $a_i \ldots a_{i+n-1} = b_1 \ldots b_n$. If $\alpha = a_1 a_2 \cdots$ is an infinite sequence and $w = w_1 \cdots w_n$ is a word, we denote by $\sharp_N\{w\}(\alpha)$ the number of occurrences of w in $a_1 a_2 \cdots a_N$. I.e. $\sharp_N\{w\}(\alpha) = |\{i; \alpha_i \ldots \alpha_{i+n-1} = w_1 \ldots w_n \wedge\}|$

Definition 1.33: Normal sequences

$\alpha \in \{0,1\}^\omega$ is said to be *normal* if, $\forall m, \forall w \in \{0,1\}^m, \lim_{N\to\infty} \sharp_N\{w\}(\alpha)/N = 2^{-m}$.

If you take a fair random sequence of $\{0,1\}^\omega$, in the sense that each bit has probability $1/2$ of being 1, then this sequence is normal with probability 1. In a sense being normal is a prerequisite to being random. Concrete examples of normal sequences include Champernowne's binary sequence $0100011011000001 \cdots$ [Cha33] (each word of $\{0,1\}^*$ are concatenated in size-ascending-lexicographic order), and many more examples exist [Bug12].

With out current definition of normal sequences occurrences of a word may overlap, for instance the string 00 appears twice in 00011011 (at the beginning) and the two occurrences overlap. In the literature this is sometimes called the non-aligned version of normality. This may prove tedious to reason about normal sequences, for instance it is not immediately clear [4] why a fair random sequence must be normal in this way. A another intuitive way of defining normality would be to say that you only counts occurrences of $w \in \{0,1\}^n$ on indices multiples of n, this way you get rid of overlapping occurrences. In the literature this is sometimes called the aligned version of normality. Theorem 1.6 states that these two notions of normality are equivalent.

Definition 1.34: Blocks

Let $\alpha = \alpha_1 \alpha_2 \cdots$ be an infinite sequence. Let $i, n \in \mathbb{N}_{>0}$. The ith *block of size n in α*, denoted $B^i_n(\alpha)$, is the finite sequence $\alpha_{(i-1)n+1} \alpha_{(i-1)n+2} \cdots \alpha_{in}$.
Given a word $w \in \{0,1\}^n$, we write $\sharp^{(n)}_N\{w\}(\alpha)$ for the number of blocks of α of size n that are equal to w in the prefix of size $N \times n$:

$$\sharp^{(n)}_N\{w\}(\alpha) = \mathrm{Card}\{i \in [0, N-1] \mid B^i_n(\alpha) = w\},$$

for $k \in \mathbf{N}$ and $w \in \{0,1\}^k$. We define $\mathrm{freq}(\alpha, w)$ as the following limit, when it exists:

$$\mathrm{freq}(\alpha, w) = \lim_{N\to\infty} \frac{\sharp^{(n)}_N\{w\}(\alpha)}{N}.$$

Definition 1.35: Block normality for size k

Let $k \in \mathbf{N}$, the sequence is *length-k-*normal if for all words $w \in \Sigma^k$, $\mathrm{freq}(\alpha, w)$ is well defined and equal to 2^{-k}.

---

[4] at least to us

Definition 1.36: Block normality

The sequence α is said to be *block normal* if it is *length-k*-normal for all k.

Theorem 1.6: [NZ51; Pos61b]

Let $\alpha \in \{0,1\}^\omega$, α is normal iff it is block-normal

We can now get to the point of this chapter: Agafonov'theorem. Agafonov theorem states that selection by a finite state selector preserves normality, i.e. if α is a normal sequence and A is a DFA, then the subsequence $A(\alpha)$ is either finite or a normal sequence.

Theorem 1.7: Agafonov's theorem

Let $\alpha \in \{0,1\}^\omega$ be a sequence, then α is normal if and only if for all deterministic automaton A, the subsequence $A(\alpha)$ is either finite or normal.

In chapter 2 we extend this result to probabilistic selectors. In chapter 3 we consider models of computation which very sightly extend DFA and we consider whether Agafonov' theorem still stands.

Chapter 2

Extension of Agafonov' theorem to probabilistic automatas [LSS24]

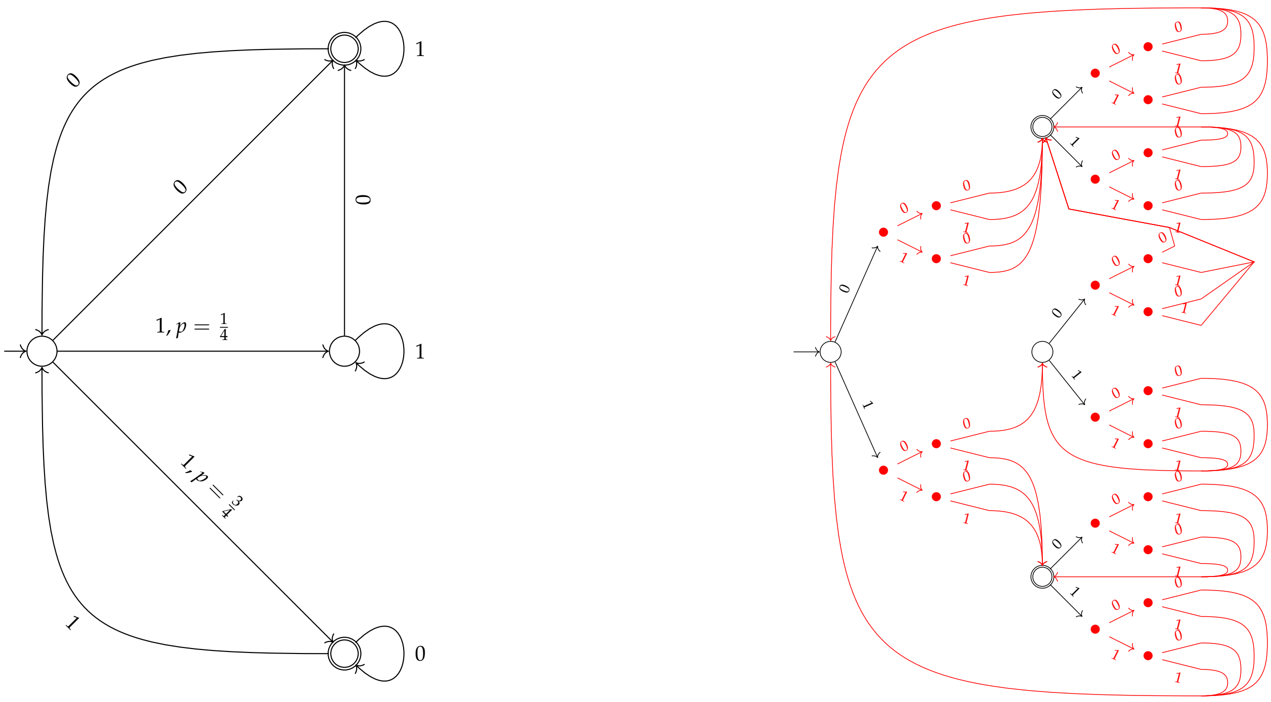
In this section, we address the following question: do probabilistic selector verify Agafonov' theorem. We provide a partial positive answer : in the case where the probabilities involved are rational. More formally, we prove that given a normal sequence α and a rational probabilistic selector P, the selected subsequence $P(\alpha)$ will be a normal sequence with probability 1.

2.1 Introduction

A *finite-state selector* is a deterministic finite automaton (DFA) that selects the nth symbol from α if the length $n-1$ prefix of α is accepted by the DFA. *Agafonov's Theorem* [Aga68a; Aga68b] is the celebrated result that a sequence α is normal iff any DFA that selects an infinite sequence from α, selects a normal sequence. While alternative proofs, generalizations [WK19] [KS21] – and counter-examples to generalizations [MR06] – abound, all results in the literature consider deterministic or non-deterministic DFAs, but none consider probabilistic computation.

The extension to probabilistic selection is quite natural – not only are the underlying notions probabilistic in nature (i.e., normality of the transformed sequence), but the machinery of finite automata and similar computational devices itself has a 60-year history [Rab63] of being extended to probabilistic devices.

In the present section we study finite-state selectors equipped with probabilistic transitions from each state. As finite-state selectors can be viewed as devices sequentially processing successively larger prefixes of infinite sequences, we eschew the machinery of stochastic languages (where the initial state is a probability distribution on the states, and a string is accepted according to threshold rules) — instead initial and accepting states are kept "as usual" in finite-state selectors. Probabilistic selection entails that normality may not be preserved in all runs of an automaton: For example, consider an automaton with two states S_1 and S_2, only one of which is accepting, and transitions on 0 and 1 from S_i to S_i ($i \in \{1,2\}$) with probability 1/2 and from S_i to $S_{i+1 \bmod 2}$ with probability 1/2; then for any normal sequence α, there is a run of the automaton on α that will select the sequence $0^\omega = 000\cdots$. The main result of the present paper is to show that the probability of having such runs is zero — in fact that for any probabilistic finite-state selector A with rational probabilities and any normal sequence α, *the probability that a run of* A on α *will select a normal sequence is 1*. The proof progresses by treating the relatively tame case of *dyadic* probabilities (i.e., of the form $a/2^k$ with a and k non-negative integers) first, and subsequently "simulating" finite-state selectors with arbitrary rational probabilities by "determinized" selectors with dyadic probabilities.



(a) A probabilistic selector

(b) Determinization

Figure 2.1: A probabilistic finite state selector and its determinization
Unlabelled edges correspond to blind transitions, i.e. transitions valid for both 0 and 1.

Contributions. We prove that (the pertinent analogue of) Agafonov's Theorem holds in the setting of probabilistic selection, namely that a probabilistic finite state selector with rational probabilities preserves normality with probability 1. An added contribution is that the proof methods involved are novel, and may be of independent interest. As in Agafonov's original paper, and to keep complexity simple, all results are stated for binary alphabets. We fully expect all results to hold for arbitrary finite alphabets, *mutatis mutandis*.

Related work. Agafonov's Theorem has been generalized in multiple ways beyond finite automata (see, e.g.,[AD79; AM15; BCH15; CV20; KW75; WK19]). Conversely, it is known that when adding trifling computational expressivity to finite-state selectors, counterexamples to Agafonov's Theorem for the resulting selectors can be constructed [MR06]. While some existing work considers preservation of more general measures by finite automata, or similar selectors [Car20b], and substantial work exists relating equidistribution to various types of automata [SS72; BCH15] no extant work considers stochastic *selection*. Agafonov's Theorem itself has been proved by a multitude of different techniques, e.g. [BL92; Car20a; BCH15]; it is conceivable that some of these can be adapted to alternative proofs, or extensions, of the results reported in the present paper.

2.2 Preliminaries and notation

We first define selectors which are just automata working on alphabet $\Sigma = \{0,1\}$, the name selector is there to emphasize that we use these automata as a means of extracting bits from an infite sequence. Our results can be adapted to bigger alphabet size (the notion of normality needs also some straight forward adapting)

Definition 2.1: Finite state Selector

A finite state selector (or just selector) $\mathcal{S}$ is a 4-uple (Q, δ, q_0, F) interpeted as probabilistic automaton $(Q, \Sigma, \delta, q_0, F)$ where $\Sigma = \{0,1\}$
A selector is dyadic (resp. rational, resp deterministic) if it is also a dyadic probabilistic automaton (resp. rational, resp. DFA) (see Definition 1.26).

We recall Agafonov theorem.

Theorem 2.1: Agafonov's theorem

Let $\alpha \in \{0,1\}^\omega$ be a sequence, and $\mathcal{S}$ a deterministic finite selector. Then α is normal if and only if for all deterministic finite selector $\mathcal{S}$, the subsequence $\mathcal{S}(\alpha)$ is either finite or normal.

2.2.1 Technical lemmas about normality

We will establish a few results on normal sequences that will be useful in later proofs. We first define notions that will be used in the proofs.

Definition 2.2: Normality relative to a word w

Let $\alpha \in \{0,1\}^\omega$ be a sequence, and $w \in \{0,1\}^*$ a word. We say that α is *w-normal* if $\mathrm{freq}(\alpha, w) \overset{\text{def}}{=} \lim_{N\to\infty} \frac{\sharp_N\{w\}(\alpha)}{N} = 2^{-|w|}$.
Given $\epsilon \in \mathbb{R}$, we say that α is *w-normal up to ϵ* if $\exists N_0, \forall N > N_0, \left|\frac{\sharp_N\{w\}(\alpha)}{N} - 2^{-|w|}\right| < \epsilon$.

We now restate a weaker property for sequences than normality: being normal for words of a fixed length k. The main lemma associated to that notion will be that if a sequence is normal for words of length mk for a fixed integer m and all integers k, then it is normal (i.e. normal for words of arbitrary length).

Lemma 2.1:

Let α be a sequence in $\{0,1\}^\omega$. The following are equivalent:

- α is block-normal
- there exists $m \in \mathbf{N}_{>0}$ such that α is *length-km* normal for all $k \in \mathbf{N}$.

Proof

Consider a sequence α which is length-km normal for all $k \in \mathbf{N}$, and fix a word $w \in \{0,1\}^n$. We will prove that α is w-normal. We will use the length-mn normality of α.

For this, we note that we can write:

$$\sharp_N^{(n)}\{w\}(\alpha) = \text{Card}\{i \in [0, N-1] \mid B_n^i(\alpha) = w\}$$

by grouping blocks $B_n^i(\alpha)$ by groups of size m

$$= \sum_{w_1,\ldots,w_m \in \{0,1\}^n} \text{Card}\{i \in \{1,\ldots,m\} \mid w_i = w\}.\sharp_{N/m}^{(mn)}\{w_1 \cdot \cdots \cdot w_m\}(\alpha)$$

by counting differently

$$= \sum_{w_1,\ldots,w_{m-1} \in \{0,1\}^n} \sum_{j=1}^{m} \sharp_{N/m}^{(mn)}\{w_1 \cdot \cdots \cdot w_{j-1} \cdot w \cdot w_j \cdot w_{m-1}\}(\alpha)$$

As a consequence:

$$\text{freq}(\alpha, w) = \lim_{N\to\infty} \frac{\sharp_N^{(n)}\{w\}(\alpha)}{N} \quad \text{by above}$$

$$= \lim_{N\to\infty} \frac{1}{N} \cdot \sum_{w_1,\ldots,w_{m-1} \in \{0,1\}^n} \sum_{j=1}^{m} \sharp_{N/m}^{(mn)}\{w_1 \cdot \cdots \cdot w_{j-1} \cdot w \cdot w_j \cdot w_{m-1}\}(\alpha)$$

We sum over finitely many elements hence the interchanging

$$= \sum_{w_1,\ldots,w_{m-1} \in \{0,1\}^n} \sum_{j=1}^{m} \lim_{N\to\infty} \frac{1}{N} \cdot \sharp_{N/m}^{(mn)}\{w_1 \cdot \cdots \cdot w_{j-1} \cdot w \cdot w_j \cdot w_{m-1}\}(\alpha)$$

by definition of freq

$$= \sum_{w_1,\ldots,w_{m-1} \in \{0,1\}^n} \sum_{j=1}^{m} \frac{1}{m} \text{freq}(\alpha, w_1 \cdot \cdots \cdot w_{j-1} \cdot w \cdot w_j \cdot w_{m-1})$$

By length-mn normality of α.

$$= \sum_{w_1,\ldots,w_{m-1} \in \{0,1\}^n} \sum_{j=1}^{m} \frac{2^{-mn}}{m} = 2^{n(m-1)} 2^{-mn} = 2^{-n}$$

Now the following lemma states that the proportion of blocks equal to a fixed word w in a prefix of size N of a normal sequence asymptotically behaves as a linear function. The proof is quite straightforward.

Lemma 2.2:

Let α be a normal sequence and $w \in \{0,1\}^n$. Then $\sharp_N^{(n)}\{w\}(\alpha) = 2^{-n}N + o(N)$.

Proof

If it were not true, we would have that there exists some $\epsilon > 0$ and a sequence $(N_i)_{i\in\mathbf{N}}$

such that $\left|\sharp_{N_i}^{(n)}\{w\}(\alpha) - 2^{-n}N_i\right| > \epsilon N_i$ for all $i \in \mathbf{N}$. In other words,

$$\left|\frac{\sharp_{N_i}^{(n)}\{w\}(\alpha)}{N_i} - 2^{-n}\right| > \epsilon.$$

This contradicts the normality of α since it implies that $\lim_{i\to\infty}\left|\frac{\sharp_{N_i}^{(n)}\{w\}(\alpha)}{N_i} - 2^{-n}\right| > \epsilon \neq 0$.

In future proofs we will often want to compute the frequency of a word $w \in \{0,1\}^n$ in α up to index N, to do so we will want to group blocks of size n $(B_n^j(\alpha))_{j\in\mathbb{N}}$ in sets E_i of size $|E_i| = 2^n$ such that each E_i contain all possible strings of size n. Those will be defined from a partition $(V_i^n(\alpha))_{i\in\mathbf{N}}$ of $\mathbf{N}$ such that the set $V_i^n(\alpha)$ contains the indices of the blocks of size n of α contained in E_i.

We introduce the $V_i^n(\alpha)$: Given a size n there are 2^n different words in $\{0,1\}^n$, name them $(u_j)_{j\in[\![1;2^n]\!]}$. $V_i^n(\alpha)$ is a set containing for all $j \in [\![1;2^n]\!]$ the index of the i-th block of size n in α equal to u_j

Definition 2.3: $V_i^n(\alpha)$

Let $\alpha \in \{0,1\}^\omega$ be a normal sequence, and $n \in \mathbb{N}$. We define $\theta_i(\alpha) : \{0,1\}^n \to \mathbf{n}$ as mapping a word w to the value j such that $B_n^j(\alpha)$ is exactly the i-th block of size n of α equal to w. (i.e. there are exactly $i-1$ indices $j_1 < j_2 < j_3 < \ldots < j_{i-1} < j$ such that $\forall k, B_n^{j_k}(\alpha) = w \wedge B_n^j(\alpha) = w$)
The sets of indices $(V_i^n(\alpha))_{i\in\mathbb{N}} \subset \mathbb{N}$ are then defined as the image $\mathrm{Im}(\theta_i(\alpha))$.

This is well defined as α is normal for any $i \in \mathbb{N}$ and $w \in \{0,1\}^n$ there always exists the i-th block of α equal to w.

Example : $\alpha = 00 \cdot 01 \cdot 01 \cdot 10 \cdot 00 \cdot 11 \cdot 01 \cdot 00 \cdot 11 \cdot 10 \cdots$

$V_1^2 = \{1;2;4;6\}$, $V_2^2 = \{3;7;9;10\}$, $V_3^2 = \{5;8;\ldots\}$. V_1^2 contains the index of the first block equal to 00, of the one equal to 01, of the one equal to 10, of the one equal to 11. V_2^2 contains the index of the second block equal to 00, of the one equal to 01 etc.

The next lemma gives useful bounds on the V_i^n.

Lemma 2.3:

Let α be a normal sequence, $n \in \mathbb{N}$. Consider the sets $V_i^n(\alpha)$ from Definition 2.3. We have that $\max_{i=1}^{N} \max V_i^n(\alpha) = N2^n + o(N)$, and $|[N] \setminus \bigcup_{i=1}^{N/2^n} V_i^n| = o(N)$.

Proof

This comes from the fact that for any $w \in \{0,1\}^n$, $\sharp_N^{(n)}\{w\}(\alpha) = 2^{-|w|}N + o(N)$.

The following probabilistic lemma is needed for the proof of Lemma 2.12.

Lemma 2.4:

Let Ω_1 and Ω_2 be two sample spaces. Let $(X_i)_{i\in\mathbb{N}}$ be an iid family of r.v. which take value in $\Omega_1^{\mathbb{N}}$, $(Y_i)_{i\in\mathbb{N}}$ another iid family of r.v. which take value in $\Omega_2^{\mathbb{N}}$. Let $f(X,Y)$ be a function in $\Omega_1 \times \Omega_2 \mapsto \mathbb{R}$. Suppose $\forall x \in \Omega_1, \forall y \in \Omega_2, f(X_0, y)$ and $f(x, Y_0)$ have finite expected value and variance then

$$\mathbb{P}_{Y_i}\left(\sum_x \left[\mathbb{P}(X = x)\sum_{i=1}^{N} f(x, Y_i)\right] = N \times \mathbb{E}_{X_0,Y_0}(f(X_0, Y_0)) + o(N)\right) = 1.$$

Proof

This is a direct application of the law of large numbers .

2.3 Dyadic case

We first restrict to dyadic selectors of Definition 2.1 (all the transition probabilities are dyadic numbers).

Definition 2.4: Dyadicity degree

Given a dyadic selector S, we define its dyadicity degree as the smallest integer D such that for all states $q, q' \in Q$ and element $a \in \{0,1\}$, there exists $m \in \mathbb{N}$ such that the probability $t(q,a,q')$ can be written as $\frac{m}{2^D}$.

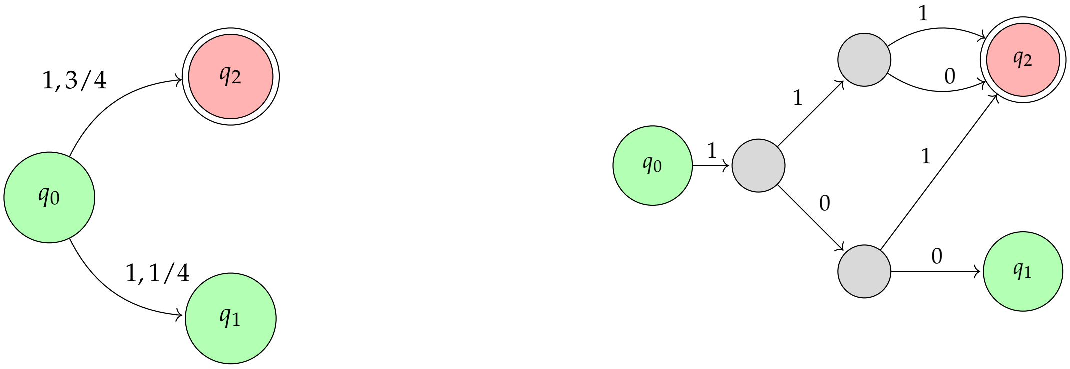
We first define the determinisation of a dyadic selector. The determinisation is based upon this classic idea. Suppose you have access to a random fair coin (and thus you have r.v. X such that $P(X = 1) = 1/2 = P(X = 0)$) how can you use this coin to simulate r.v Y such that $P(Y = 1) = \frac{m}{2^D} = 1 - P(Y = 0)$? Easy, you draw X at random D times, the sequence of throws spells out a number in binary, if it less than m then $Y = 1$ otherwise $Y = 0$. If Y can take more that two values (but always with dyadic probabilities) we can easily adapt the construction.

Definition 2.5:

Given a dyadic selector $\mathcal{S} = (Q, \delta, q_0, F)$ of dyadicity degree D, we define a *determinisation* $\mathrm{Det}(\mathcal{S})$ of S as the *deterministic selector* (Q', δ', q_0', F') where:

- $Q' = Q \cup Q \times \{0,1\} \times \{0,1\}^{\leq D-1}$;
- $q_0' = q_0$ and $A' = A$;
- the transition function δ' is defined as follows:
  - for all $q \in Q$, $a \in \{0,1\}$, $\delta'(q,a) = (q,a,\epsilon)$ where ϵ is the empty word;
  - for all $((q,b,w)$ with $q \in Q$, $b \in \{0,1\}$ and $w \in \{0,1\}^{\leq D-2}$, and $a \in \{0,1\}$, $\delta'((q,b,w),a) = (q,b,w\cdot a)$;
  - for all (q,b,w) with $w \in \{0,1\}^{D-1}$, $\delta'((q,b,w),a) = q'$ where $q' = \phi(w \cdot a)$ for a chosen $\phi_{q,b} : 2^D \to Q$ such that the preimage of any $s \in Q$ has cardinality m_s where m_s is defined by $t(q,b)(s) = \frac{m_s}{2^D}$.

Now, the principle is that the behaviour of a dyadic selector S on the sequence α can be simulated by the behaviour of a determinisation $\mathrm{Det}^D(\mathcal{S})$ computing on an interleaving of α and a random *advice* string ρ noted $I^D(\alpha,\rho)$ (ρ this corresponds to our coin that we launch D times). We give an example of a transition in figure 2.2. Note that this transformation is applied to every transition in the same way, in particular the number of "intermediate states" (in grey on the figure) is the same regardless of the transition probability (but of course the last outgoing edges may differ). Please look at figure 2.1 in the introduction for a full determinisation.



$w = 1$

$w' = 1r_0r_1$ (r_0,r_1 are random bits)

Figure 2.2: Determinisation of one transition. Reading w on the left probabilistic automaton is equivalent to reading w' on the right deterministic automaton

Definition 2.6: Interwoven sequence for dyadic selectors

Let α, ρ be sequences in $\{0,1\}^\omega$, and $D \in \mathbf{N}$. The interwoven sequence $I^D(\alpha,\rho)$ is defined as the sequence:

$$\alpha_0\rho_0\dots\rho_{D-1}\alpha_1\rho_D\dots\rho_{2D-1}\dots.$$

Note that the interweaving of two normal sequences can be a non-normal sequence, e.g. the interweaving of α with itself $I^1(\alpha,\alpha)$ is not normal.

Lemma 2.5:

Let $\mathcal{S}$ be a dyadic selector of dyadicity degree D. Then for all sequences $\alpha \in \{0,1\}^\omega$ the random variables $\mathcal{S}(\alpha)$ and $\mathrm{Det}(\mathcal{S})(I^D(\alpha,\rho))$ where ρ is a fair random infinite sequence of $\{0,1\}^\omega$ have the same distribution.

Proof

This is a special case of Lemma 2.8.

Consider given a normal sequence α. We now prove that for almost all random advice sequence $\rho \in \{0,1\}^\omega$, the interwoven sequence $I^D(\alpha,\rho)$ is normal. This is the key lemma in the proof of Theorem 2.2. In order to give a reader an intuition for why Lemma 2.6 is true: the lemma states that if you pick a normal sequence α and add (say) 10 random bits in between each bit of α then you get with probability 1 that this is still a normal number. If α was an infinite random sequence then the result would clearly be true, we now have to prove it for normal α

Lemma 2.6:

Let $\alpha \in \{0,1\}^\omega$ be a normal sequence. Then for all $D \in \mathbf{N}$,

$$\mathrm{Prob}_{\rho \in \{0,1\}^\omega}[I^D(\alpha,\rho)\text{is normal}] = 1.$$

Proof

In case $D = 0$, the interwoven sequence $I^D(\alpha,\rho)$ is equal to α. As a consequence, $\mathrm{Prob}_{\rho \in \{0,1\}^\omega}[I^D(\alpha,\rho)$ is normal$]$ is equal to 1.

We now suppose that $D \neq 0$. Given $m \in \mathbf{N}_{>0}$, we will show that $I^D(\alpha,\rho)$ is length-$(D+1)m$ normal with probability 1. This implies that for almost all $\rho \in \{0,1\}^\omega$, the sequence $I^D(\alpha,\rho)$ is length-$(D+1)m$ normal for every $m \in \mathbf{N}_{>0}$. By Lemma 2.1, this implies that for almost all $\rho \in \{0,1\}^\omega$, the sequence $I^D(\alpha,\rho)$ is normal.

We now fix $m \in \mathbf{N}_{>0}$, and $w \in \{0,1\}^{(D+1)m}$. We will consider the block decomposition of $I^D(\alpha,\rho)$ into blocks of size $(D+1)m$ and prove that:

$$\mathrm{Prob}_{\rho \in \{0,1\}^\omega}\left(\lim_{N\to\infty} \frac{\sharp_N^{((D+1)m)}\{w\}(I^D(\alpha,\rho))}{N} = 2^{-(D+1)m}\right) = 1.$$

We note that blocks of size $(D+1)m$ follow the pattern:

$$\alpha_i r_j \ldots r_{j+D} \alpha_{i+1} r_{j+D} \ldots r_{j+2D} \ldots \alpha_{i+m-1} r_{j+(m-1)D} \ldots r_{j+mD}.$$

We will consider $\lfloor w \rfloor_D = w_0 w_{D+1} w_{2(D+1)} \ldots w_{(m-1)(D+1)}$ the subword of w corresponding to the positions of bits from α in this pattern.

We will consider the block decomposition of α into blocks of size m. Let $\mathrm{idx}(i) = j$ where j is the i-th block such that $B_j^m(\alpha) = \lfloor w \rfloor_D$. Note that this function is well defined because α is a normal sequence. Note that if a given block $B_i^{(D+1)m}(I^D(\alpha,\rho))$ is equal to w, then $\lfloor B_i^{(D+1)m}(I^D(\alpha,\rho)) \rfloor_D$ should be equal to $\lfloor w \rfloor_D$. We write $\tilde{N} = \sharp_N^{(m)}\{\lfloor w \rfloor_D\}(\alpha)$, note that it is the maximal i such that $\mathrm{idx}(i) < N$. We also define $\bar{w}$ as the complementary subsequence of w:

$$\bar{w} = w_1 \ldots w_D w_{D+2} \ldots w_{2(D+1)-1} w_{2(D+1)+1} \ldots w_{m(D+1)-1}.$$

We introduce a new notation: we will write $\sharp_{\mathrm{Im}(\mathrm{idx})<N}^{((D+1)m)}\{w\}(I^D(\alpha,\rho))$ to denote the number of blocks of size $(D+1)m$ equal to w within the blocks indexed by some $j < N$ in $\mathrm{Im}(\mathrm{idx})$.

$$\begin{aligned} P &= \mathrm{Prob}_{\rho \in \{0,1\}^\omega}\left(\lim_{N\to\infty} \frac{\sharp_N^{((D+1)m)}\{w\}(I^D(\alpha,\rho))}{N} = 2^{-(D+1)m}\right) \\ &= \mathrm{Prob}_{\rho \in \{0,1\}^\omega}\left(\lim_{N\to\infty} \frac{\sharp_{\mathrm{Im}(\mathrm{idx})<N}^{((D+1)m)}\{w\}(I^D(\alpha,\rho))}{N} = 2^{-(D+1)m}\right) \end{aligned}$$

Now by Lemma 2.2 we have that $\lim_{N\to\infty}\tilde{N} = N.2^{-m}$. Hence:

$$\begin{aligned}
P &= \mathrm{Prob}_{\rho\in\{0,1\}^\omega}\left(\lim_{\tilde{N}\to\infty}\frac{\sharp^{((D+1)m)}_{\mathrm{Im}(\mathrm{idx})<N}\{w\}(I^D(\alpha,\rho))}{\tilde{N}.2^{-m}} = 2^{-(D+1)m}\right)\\
&= \mathrm{Prob}_{\rho\in\{0,1\}^\omega}\left(\lim_{\tilde{N}\to\infty}\frac{\sharp^{((D+1)m)}_{\mathrm{Im}(\mathrm{idx})<N}\{w\}(I^D(\alpha,\rho))}{\tilde{N}} = 2^{-Dm}\right)\\
&= \mathrm{Prob}_{\rho\in\{0,1\}^\omega}\left(\lim_{\tilde{N}\to\infty}\frac{\sharp^{(Dm)}_{\mathrm{Im}(\mathrm{idx})<N}\{\bar{w}\}(\rho)}{\tilde{N}} = 2^{-Dm}\right)
\end{aligned}$$

By the law of large numbers, we have that

$$\mathrm{Prob}_{\rho\in\{0,1\}^\omega}\left(\lim_{\tilde{N}\to\infty}\frac{\sharp^{(Dm)}_{\mathrm{Im}(\mathrm{idx})<N}\{\bar{w}\}(\rho)}{\tilde{N}} = 2^{-Dm}\right) = 1,$$

which concludes the proof.

This lemma then leads to the following theorem.

Theorem 2.2: Agafonov theorem for dyadic selectors

Let $\alpha \in \{0,1\}^\omega$ be a sequence. Then α is normal if and only if for all dyadic finite selector $\mathcal{S}$ the probability that $\mathcal{S}(\alpha)$ is either finite or normal is equal to 1.

Proof

The right to left implication is simply a consequence of Agafonov's theorem (Theorem 2.1) since if for all dyadic finite selector $\mathcal{S}$ the probability that $\mathcal{S}(\alpha)$ is either finite or normal is equal to 1, then for all deterministic finite selector $\mathcal{S}$ the selected subsequence $\mathcal{S}(\alpha)$ is either finite or normal.
Now, suppose that the above implication from left to right is false. Then by Lemma 2.5 there exists a subset $R \subset \{0,1\}^\omega$ of strictly positive measure such that $\mathrm{Det}(\mathcal{S})(I^D(\alpha,\rho))$ is infinite and not normal for all $\rho \in R$. Since almost for almost all $\rho \in \{0,1\}^\omega$ the interwoven sequence $I^D(\alpha,\rho)$ is normal, this implies that there exists a ρ such that $I^D(\alpha,\rho)$ is normal and $\mathrm{Det}(\mathcal{S})(I^D(\alpha,\rho))$ is infinite and not normal. But this contradict Agafonov's theorem (Theorem 2.1).

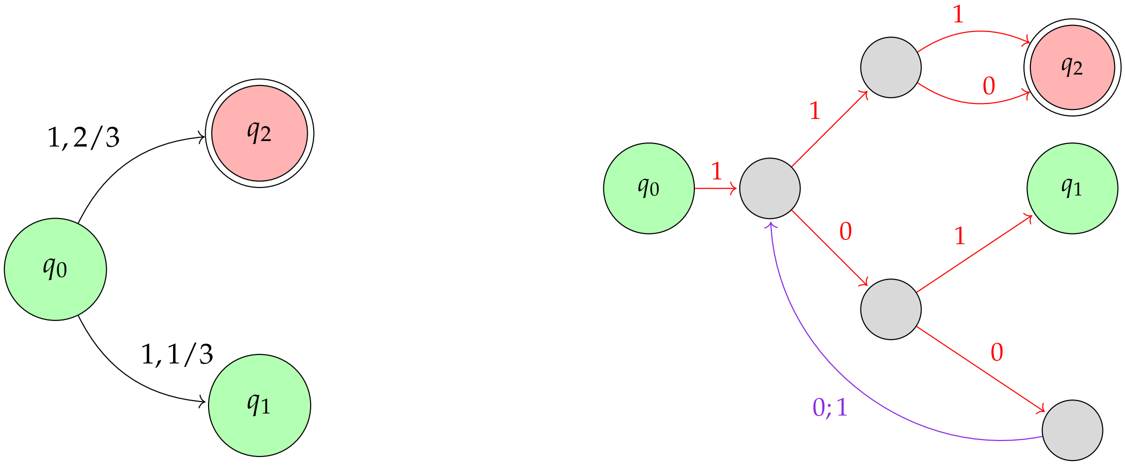
We will now consider the case of rational selectors. The difficulty in adapting the proof lies in the fact that the interwoven sequence has a less regular structure. In the above proof, each block of size $(D+1)m$ followed the same pattern. But in the case of rational selectors, the presence of feedback loops renders those pattern random, this makes the proof significantly harder. Indeed in the dyadic case the value of a block of size $(D+1)m$ was independent of the value of other blocks of size $(D+1)m$, in the rational case this is no longer true, thus we cannot apply the law of large numbers. Informally to make our proof work we divide $\mathcal{S}(\alpha)$ into non adjacent blocks whose values are independent, some bits are not contained in any blocks but we argue they are few of them and thus they don't prevent $\mathcal{S}(\alpha)$ from being normal.

2.4 Rational selector

2.4.1 Determinisation

The first step in extending the results to the rational case is to define the determinization. Let's go back to our coin parallel, let X be a random fair coin and now we want to simulate Y such that $P(Y = 1) = \frac{p}{q} = 1 - P(Y = 0)$ for $p, q \in \mathbb{N}$. Let D be the smallest integer such that $q \leq 2^D$, draw X at random D times, the sequence of random throws spells out a binary number $y \leq 2^D$, if $y \leq p$ then $Y = 1$, if $p < y \leq q$ then $Y = 0$, and if $q < y$ then you repeat the whole operation from the beginning (drawing X D times and all).

Thus there are now parts of the determinized automaton that simulates probabilistic choices will contain feedback loops. In figure 2.3 we show how to transform one probabilistic transition. When determinizing an automaton, we apply this transformation to every transition in the automaton. Note that this transformation will applied to every transition in the same way, in particular the number of "intermediate states" (in grey on the figure) is the same regardless of the transition probability (but of course the last outgoing edges may differ). To ensure that we put every probability over the same denominator before determinization.



$w = 1$

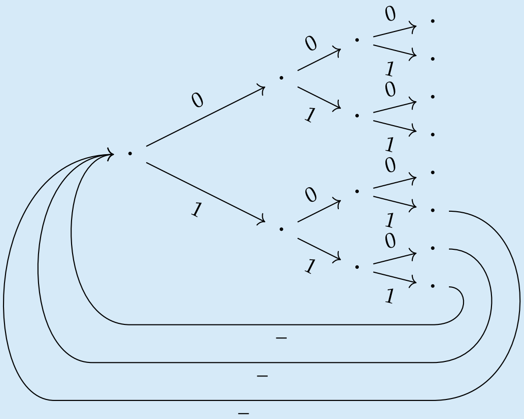
$w' = 1R$
where R is random in $\{00, 01, 10, 11\}$ and if $R = 00$ then $R = 00\tau R$ (where τ is random)

Figure 2.3: Reading w on the left is the same as reading w' on the right.

To have the Incorporating feedback loops is enough to represent any rational distribution, as shown in the next lemma.

Definition 2.7: Gadget

A (k, f)-gadget is a regular binary tree of depth k, extended with blind transitions (accepting 0 or 1) from the last f leaves to the root.



Lemma 2.7:

Any rational distribution $\mathrm{Dist}(S)$ is simulated by a gadget.

Proof

Let $p_1, \ldots, p_k$ be rationals with $\sum_i p_i = 1$, and suppose $p_i \le p_{i+1}$ for all i. Consider M the smallest common multiple of all denominators of the elements p_i, and write $p_i = \frac{\tilde{p}_i}{M}$. We will denote by $q_i = \sum_{j=1}^{i} \tilde{p}_i$. Note that $q_0 = 0$ and $q_k = \sum_i \tilde{p}_i = M$. Now consider P the smallest natural number such that $2^P \ge M$. We build the regular automaton of depth P with feedback loops on $2^P - M$ leaves. We will show that the probability p of reaching a leaf within $[q_i + 1, q_{i+1}]$ is equal to p_i. One only need to compute:

$$p = \frac{\tilde{p}_i}{2^P} \sum_{m \ge 0} \left(\frac{2^P - M}{2^P} \right)^m = \frac{\tilde{p}_i}{2^P} \frac{1}{1 - \frac{2^P - M}{2^P}} = \frac{\tilde{p}_i}{2^P} \frac{2^P}{2^P - (2^P - M)} = \frac{\tilde{p}_i}{2^P} \frac{2^P}{M} = \frac{\tilde{p}_i}{M} = p_i$$

As explained in out thought experiment with coins, and following figure 2.3 we define the interwoven sequence for rational selectors.

Definition 2.8: $I_k^D(\alpha,\rho)$ Interwoven sequence for rational selectors

Let $\alpha \in \{0,1\}^\omega$ an input sequence, and $\rho \in \{0,1\}^\omega$ an *advice* sequence. For any $k, D \in \mathbb{N}$ we define the interwoven sequence $I_k^D(\alpha,\rho)$ as:

$$\alpha_1\rho_1\ldots\rho_{i_1}\alpha_2\rho_{i_1+1}\ldots\rho_{i_2}\alpha_3\ldots,$$

where $i_1 < i_2 < \ldots$ is the sequence of indices i_j such that $\rho_{i_j+1}\ldots\rho_{i_{j+1}}$ is equal to $w_1r_1w_2r_2\ldots r_mw_{m+1}$ where:

- $w_1, w_2, \ldots, w_m$ are among the $2^D - k$ greatest elements in $\{0,1\}^D$ (considered with the natural alphabetical order);
- w_{m+1} belongs to the k smallest elements in $\{0,1\}^D$;
- r_i are bits in $\{0,1\}$ which we will call *return bits*, corresponding to feedback loops.

Now we define the determination $\mathrm{Det}(\mathcal{S})$ of rational selector $\mathcal{S}$. $\mathrm{Det}(\mathcal{S})(\alpha,\rho)$ is designed to simulate the execution of $\mathcal{S}(\alpha)$ when ρ is an infinite fair random sequence.

Since the selector is finite, one can write all rational numbers involved with a common denominator, say k. Given a rational selector $\mathcal{S}$, we will call k the *rationality degree* of $\mathcal{S}$ and the *dyadicity degree* D of a rational selector $\mathcal{S}$ as the smallest integer such that $2^D \geq k$. Each transition will be simulated by a gadget as defined above, the feedback edges of all gadgets will be the $2^D - k$ last edges in the gadget, this does not depend on the specific transition considered.

Definition 2.9: rationality degree and dyadicity degree

Let $\mathcal{S}$ be a rational selector, we define its rationality degree k as the smallest integer such that all transition probabilities may be written as $\frac{p}{k}$ (i.e. the common denominator of all probabilities in $\mathcal{S}$). The dyadicity degree of $\mathcal{S}$ is D where D is the smallest integer such that $k \leq 2^D$.

Definition 2.10: Determinization of rational selectors

Let $\mathcal{S} = (Q, \delta, q_0, F)$ be a rational selector of rationality degree k and dyadicity degree D. We define its *determinisation* $\mathrm{Det}(\mathcal{S})$ as the deterministic selector (Q', δ, q_0, F) where:

- $Q' = Q \cup Q \times \{0,1\} \times (\{0,1\}^{\leq k-1} \cup \{\text{return}\})$;
- $q'_0 = q_0$ and $F' = F$;
- the transition function δ is defined as follows:
  - for all $q \in Q$, $\delta'(q, a) = (q, a, \epsilon)$ where ϵ is the empty list;
  - for all $((q, b, w)$ with $w \in \{0,1\}^{\leq k-2}$, $\delta'((q, b, w), a) = (q, b, w \cdot a)$;
  - for all (q, b, w) with $w \in \{0,1\}^{k-1}$ and $a \in \{0,1\}$:
    - if $w \cdot a$ belongs to the $2^D - k$ last leaves (i.e. the $2^D - k$ largest elements of $\{0,1\}^D$ for the natural order), then $\delta'((q, b, w), a) = (q, b, \text{return})$;
    - otherwise, $\delta'((q, b, w), a) = q'$ where $q' = \phi(w \cdot a)$ for a chosen $\phi_{q,b} : k \to Q$ such that the preimage of any $s \in Q$ has cardinality m_s where m_s is defined by $\delta(q, b)(s) = \frac{m_s}{k}$;
  - for all (q, b, return) and any $a \in \{0,1\}$, $\delta'(q, b, \text{return}) = (q, b, \epsilon)$.

We note that this is a direct generalisation of the dyadic case, i.e. if the considered selector is dyadic, then the interwoven sequence $I^D_{2^D}$ just defined coincides with the definition from the previous section. Similarly, the determinisation of a dyadic selector is a special case of the determinisation of a rational selector. We can see here the difficulty in adapting the proof to the rational case arising: instead of interweaving one block of ρ of size D between each bit of α, we interweave a block of bits from ρ of variable length.

Note however that we carefully defined the determinisation so that the size of these blocks does not depend on the state we're in and thus does not depend upon the read values α_i.

First, we check that the determinisation simulates the rational selector when given random advice strings.

Lemma 2.8:

Let $\mathcal{S}$ be a rational selector of rationality degree k and dyadicity degree D. Then for all sequence $\alpha \in \{0,1\}^\omega$ the random variables $\mathcal{S}(\alpha)$ and $\mathrm{Det}(\mathcal{S})(I^D_k(\alpha, \rho))$ where ρ is an infinite fair random sequence, have the same distribution.

Proof

Let ρ be an infinite fair random sequence. By construction of $\mathrm{Det}(\mathcal{S})(I^D_k(\alpha, \rho))$, for a any two state q and q' and letter $q \in \{0,1\}$ the probability of going from q to q' whilst reading a in $\mathrm{Det}(\mathcal{S})(I^D_k(\alpha, \rho))$ (ignoring the gadget states in between) is equal to the probability of going from q to q' whilst reading a in $\mathcal{S}(\alpha)$.

2.4.2 Rational selectors preserve normality

In the following, α will be a infinite sequence, not considered normal unless explicitly stated. We will write w to denote a finite word. We denote by ρ and τ fair infinite random sequences, and by r a finite random sequence. Lastly, q will be the probability to loop back at the end of a gadget, equal to $1 - \frac{k}{2^D}$. We will denote by $A(N) \xrightarrow{N} B(N)(1 \pm \epsilon)$ the fact that

$$\exists N_0, \forall N > N_0, B(N)(1-\epsilon) < A(N) < B(N)(1+\epsilon).$$

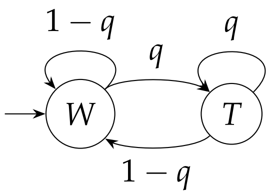
To prove that rational selectors preserve normality, we will prove in this section that $\mathbb{P}_\rho(I_k^D(\alpha,\rho) \text{is normal}) = 1$, that is the generalised version of Lemma 2.6. As in the dyadic case, this is the crux of the problem, and the proof of the main theorem will easily follow. In order to prove this technical lemma, we analyze a process we call $\mathcal{F}$ which takes a sequence α and inserts in between every bit of α a random amount of random bits. We will then show that if α is normal the sequence $\mathcal{F}(\alpha)$ obtained in this way is normal. Finally we will argue that normality of $I_k^D(\alpha,\rho)$ amounts to the normality of $\mathcal{F}(\alpha)$.

Definition 2.11: Random process $\mathcal{F}_q$

Suppose given $K \in \mathbb{N}$, $w \in \{0,1\}^K$, $q \in [0;1[$, and $\tau \in \{0,1\}^\omega$. We define $\mathcal{F}_q(w,\tau) \in \{0,1\}^*$ as the random variable described in fig. 2.4 where we consume a bit of w when we get to state W and a bit of τ when we get to state T. The process stops when the state W is reached and there are no more bits of w to be consumed. The output is all the consumed bit in timely order.
We denote by $\mathcal{F}_q(w)$ the random variable $\mathcal{F}_q(w,\tau)$ where τ is a fair random infinite sequence.

Remark 1. Note that τ needs to be infinite because we have no bound on how many bits of it we may consume.

In the following, we may not specify q and just write $\mathcal{F}(w)$ when the context is clear.



Example: if $w = 0110$, $\tau = 10010...$, then

$$\mathcal{F}(w) = 010110010,$$

with the sequence of states

$$WTTWWTWTTW.$$

Figure 2.4: The random process $\mathcal{F}$ viewed as a Markovian process.

For now $\mathcal{F}$ has only been defined on finite strings. We extend it to infinite strings in an intuitive way.

Definition 2.12: $\mathcal{F}(\alpha)$

Suppose given $\alpha \in \{0,1\}^\omega$, $K \in \mathbb{N}_{>0}$, and $q \in \mathbb{R}$. Let $(\mathcal{F}_i)_{i \in \mathbb{N}}$ be an iid family of random variables of law $\mathcal{F}$. The random variable $\mathcal{F}_q(\alpha)$ is the infinite sequence distributed as the concatenation of the $\mathcal{F}_i$ applied to the blocks $B_K^i(\alpha)$:

$$\mathcal{F}_0(B_K^0(\alpha))\mathcal{F}_1(B_K^1(\alpha))\mathcal{F}_2(B_K^2(\alpha))\ldots.$$

Note that the value of K does not change the distribution of the random variable $\mathcal{F}(\alpha)$, hence the definition is unambiguous.

In the next lemma we analyze the length of $\mathcal{F}(w)$.

Lemma 2.9: Expected size of $\mathcal{F}(w)$

Let $q \in [0;1[$ and $(\mathcal{F}_i)$ be an iid family of random variables of law $\mathcal{F}_q$. Then for all $K \in \mathbb{N}$ and for any family $(w_i)_{i\in\mathbb{N}} \in (\{0,1\}^K)^{\mathbb{N}}$,

$$\mathbb{P}\left(\sum_{i\leq N} |\mathcal{F}_i(w_i)| = NKq^{-1} + o(NKq)\right) = 1.$$

Proof

By standard Markov chain analysis, the expected value of $|\mathcal{F}_i(w_i)|$ is Kq^{-1} and its variance is finite, furthermore the $\mathcal{F}_i(w_i)$ are independent the strong law of large number therefore applies and we get the desired result.

In the next lemma we show that for any w we can approximate the number of w in $\mathcal{F}(\alpha) = \mathcal{F}_0(B_K^0(\alpha))\mathcal{F}_1(B_K^1(\alpha))\mathcal{F}_2(B_K^2(\alpha))\dots$ by adding up the number of w in each $\mathcal{F}_i(B_K^i(\alpha))$ separately. Note that this is not trivial as some occurences of w may straddle two consecutive $\mathcal{F}_i(B_K^i(\alpha))$. The larger K the more precise the approximation. What we gain from this separation is that the random variable $\sharp_{s_i-|w|}\{w\}(\mathcal{F}_i(B_K^i(\alpha)))$ are independent and we can apply the law of large numbers. In contrast in $\mathcal{F}(\alpha)$ where we concatenate the $\mathcal{F}_i(B_K^i(\alpha))$ we do not have independence because knowing that w appears at the end of $\mathcal{F}_1(B_K^1(\alpha))$ may influence that it appears at the beginning of $\mathcal{F}_2(B_K^2(\alpha))$.

Lemma 2.10: Counting block by block

Let $\alpha \in \{0,1\}^\omega$ be a normal sequence, $w \in \{0,1\}^M$ be a word, and $(\mathcal{F}_i)_{i\in\mathbb{N}}$ be an iid family of random variables of law $\mathcal{F}$. For all $K \in \mathbb{N}$, we write $\beta = \mathcal{F}_0(B_K^0(\alpha))\mathcal{F}_1(B_K^1(\alpha))\mathcal{F}_2(B_K^2(\alpha))\dots$ and for all i we define $s_i = |\mathcal{F}_i(B_K^i(\alpha))|$ and $S_N = \sum_{i=0}^N s_i$. Then we have that

$$\left[\sum_{i=0}^{N} \sharp_{s_i-|w|}\{w\}(\mathcal{F}_i(B_K^i(\alpha)))\right] - \sharp_{S_N}\{w\}(\beta) < MN.$$

Proof

First note that we count indices up to $s_i - |w|$ in $\sharp_{s_i-|w|}\{w\}(\mathcal{F}_i(B_K^i(\alpha)))$ because if w appears in $\mathcal{F}_i(B_K^i(\alpha))$ it must appear before the last $|w|$ bits. For this reason we also mention that $\sharp_{s_i-|w|}\{w\}(\mathcal{F}_i(B_K^i(\alpha))) = \sharp_{|w|}\{w\}(\mathcal{F}_i(B_K^i(\alpha)))$.
Then note that $\sharp_{S_N}\{w\}(\beta) \geq \left[\sum_{i=0}^N \sharp_{s_i-|w|}\{w\}(\mathcal{F}_i(B_K^i(\alpha)))\right]$ indeed if w appears somewhere in one of the $\mathcal{F}_i(B_K^i(\alpha))$ then it also appears in β.
Therefore every w is counted in $\sharp_{S_N}\{w\}(\beta)$ and not in $\left[\sum_{i=0}^N \sharp_{s_i-|w|}\{w\}(\mathcal{F}_i(B_K^i(\alpha)))\right]$ appears at an index in the $|w|$ last bits of an $F_i(B_K^i(\alpha))$. There are at most $|w| \times N = MN$ of those.

We have that $\sum_{i=0}^N |\mathcal{F}_i(B_K^i(\alpha))|$ tends to NKq^{-1}, thus by taking large values of K the discrepancy MN of the number of w noticed in the previous theorem can be made negligible when compared to the size of the string.

In the next theorem we just prove that for a random ρ the proportion of w in $\mathcal{F}_i(B_K^i(\rho))$

is approximately $2^{-|w|}|\mathcal{F}_i(B^i_K(\rho))|$ on average.

Lemma 2.11: Expected number of occurrences w in $\mathcal{F}(\rho)$

Let $\rho \in \{0,1\}^\omega$ be a fair random infinite sequence, $q \in [0;1[$, and $w \in \{0,1\}^M$. Let $(\mathcal{F}_i)_{i\in\mathbb{N}}$ be an iid family of random variables of law $\mathcal{F}_q$. For all i, we write $s_i = |\mathcal{F}_i(B^i_K(\rho))|$ and for all N, $S_N = \sum_{i=0}^N s_i$. Then for any $\epsilon > 0$, there exists $K \in \mathbb{N}$ such that:

$$\left|\mathbb{E}(\sharp_{s_i-|w|}\{K\}(\mathcal{F}_i(B^i_w(\rho)))) - 2^{-|w|}Kq^{-1}\right| < \epsilon.$$

Proof

This result can be shown by standard analysis of fair random sequences of size Kq^{-1}. Indeed for a random ρ, $\mathcal{F}_i(B^i_K(\rho))$ is just a random sequence of expected size Kq^{-1}. Let $\epsilon \in \mathbb{R}$. If K is large enough, there exists some $\epsilon' \in \mathbb{R}$ such that a random sequence of size Kq^{-1} contains on average $2^{-|w|}Kq^{-1} + \epsilon'$ occurrences of w where $|\epsilon'| < \epsilon$.

Lemma 2.12:

Let α be a normal sequence, for any $w \in \{0,1\}^*$ and $q \in [0,1[$, $\mathcal{F}_q(\alpha)$ is w-normal with probability 1.

Here is the intuition of the proof, given a word w and a normal number α we want to study the occurrences of w in $\mathcal{F}(\alpha)$. If α was a random number it would be trivial to show that $\mathcal{F}(\alpha)$ is normal, when α is only normal this becomes much harder because the probability of occurrence of w at a specific index i depends upon α and i in a way which is very hard to compute. So we first divvy α up into blocks $B^i_K(\alpha)$ and study the sum of occurrences of w in $\mathcal{F}(B^i_K(\alpha))$. Remember the sets V^K_i of definition 2.3, we group the $\mathcal{F}(B^i_K(\alpha))$ together according to the V^K_j and now we can look at the sum of $\mathcal{F}(B^i_K(\alpha))$ for all $i \in V^K_j$. So now we have no dependency on the specific α because $B^i_K(\alpha)$ takes value in one and every element of $\{0,1\}^K$ when i ranges across V^K_j, moreover we recognize the sum of an expected value. By arguing that α is normal we can get upper-bound the number of $i < N$ not included in a V^K_j where every element of V^K_j is less than N.

Proof of Lemma 2.12

Let $(\mathcal{F}_i)_{i\in\mathbb{N}}$ be random independent processes $\mathcal{F}$. Let α be a normal sequence. Let $w \in \{0,1\}^*$, $\epsilon' \in \mathbb{R}$, and $q \in [0;1[$. Then

$$\mathbb{P}\left(\mathcal{F}_q(\alpha) \text{is } w\text{-normal up to } \epsilon'\right) = 1 \Leftrightarrow \mathbb{P}\left(\lim_N \frac{\sharp_N\{w\}(\mathcal{F}_q(\alpha))}{N} = 2^{-|w|} \pm \epsilon'\right) = 1.$$

Using Lemma 2.10 by introducing independent random variables $\mathcal{F}_i$ of law $\mathcal{F}_q$, and writing $s_i = |\mathcal{F}_i(B^i_K(\alpha))|$, the above result is implied by:

$$\forall\epsilon, \exists K, \mathbb{P}\left(\sum_{i\leq N} \sharp_{s_i-|w|}\{w\}(\mathcal{F}_i(B^i_K(\alpha))) \xrightarrow{N} 2^{-|w|}KNq^{-1}(1\pm\epsilon)\right) = 1.$$

Take $V_i(\alpha)$ as defined in definition 2.3. Call $B_N = \{i \in [1; \frac{N}{2^K}] \mid \max(V_i(\alpha)) < N\}$. We group the indices of blocks $B^j_K(\alpha)$ into sets V_i of size 2^K and such that $|V_i| = 2^K$. We may also change $s_j - |w|$ to s_j as explained in the proof of Lemma 2.10. Then the above is equivalent to

$$\forall \epsilon, \exists K, \mathbb{P}\left(S_1 + S_2 \xrightarrow{N} 2^{-|w|} K N q^{-1}(1 \pm \epsilon)\right) = 1,$$

where

$$S_1 = \sum_{i \in B_N} \sum_{j \in V_i(\alpha)} \sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha))), \qquad S_2 = \sum_{j \in [N] \setminus \bigcup_{i \in B_N} V_i} \sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha))).$$

By Lemma 2.3, we have that $|B_N| = \frac{N}{2^K} + g(N)$, where $g(N) = o(N)$. The equation can thus be further rewritten as:

$$\mathbb{P}\left(T_1 + T_2 + T_3 \xrightarrow{N} 2^{-|w|} K N q(1 \pm \epsilon)\right) = 1,$$

where:

$$\begin{aligned} T_1 &= \sum_{i \in [N/2^K]} \sum_{j \in V_i(\alpha)} \sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha))), \\ T_2 &= \sum_{i=1+N/2^K}^{\frac{N}{2^K}+g(N)} \sum_{j \in V_i(\alpha)} \sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha))), \\ T_3 &= \sum_{j \in [N] \setminus \bigcup_{i \in B_N} V_i} \sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha))). \end{aligned}$$

We now consider each term separately.

The term T_1. By construction of the V_i, as j ranges across all values in V_i, $B^j_K(\alpha)$ takes all values in $\{0,1\}^K$. By creating an appropriate bijection between j and (i, r), we can write

$$\sum_{i \in [\frac{N}{2^K}]} \sum_{j \in V_i(\alpha)} \sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha))) = \sum_{i \in [\frac{N}{2^K}]} \sum_{r \in \{0,1\}^K} \sharp_{s_{i,r}}\{w\}(\mathcal{F}_{i,r}(r)).$$

We recognize a sum over expectations as in Lemma 2.4. By Lemma 2.11, we can take K big enough such that the expected value over r of $\sharp_{s_{i,r}}\{w\}(\mathcal{F}_{i,r}(r))$ is $Kq^{-1}2^{-|w|}(1 \pm \epsilon)$. Thus:

$$\forall \epsilon, \exists K, \mathbb{P}\left(\sum_{i \in [\frac{N}{2^K}]} \sum_{j \in V_i(\alpha)} \sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha))) \xrightarrow{N} 2^{-|w|} K N q^{-1}(1 \pm \epsilon)\right) = 1.$$

The term T_2. We have that

$$\forall \epsilon, \forall K, \mathbb{P}\left(\left[\sum_{i=1+N/2^K}^{\frac{N}{2^K}+g(N)} \sum_{j \in V_i(\alpha)} \sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha)))\right] = o(N)\right) = 1,$$

because $g(N) = o(N)$ and the random variable $\sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha)))$ has finite expected value and variance (in particular constant in N).

The term T_3. We have that

$$\mathbb{P}\left(\sum_{j\in[N]\setminus\bigcup_{i\in B_N}V_i} \sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha))) = o(n)\right) = 1.$$

The sum is over $o(N)$ terms by Lemma 2.3 and the random variable $\sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha)))$ is of finite expected value and variance.
By combining the three results we can get that [a]

$$\forall\epsilon' \in \mathbb{R}, \mathbb{P}(\mathcal{F}(\alpha)\text{is } w\text{-normal up to } \epsilon') = 1.$$

We define the sequence $(\epsilon_n)_{n\in\mathbb{N}} \in \mathbb{R}^{\mathbb{N}}$ as $\epsilon_n = 1/(n+1)$. We have that

$$\mathbb{P}(\forall n, \mathcal{F}(\alpha)\text{is } w\text{-normal up to } \epsilon_n) = 1$$

as an intersection of countably many events of probability 1. We then get, by considering the countable intersection of these event that $\mathbb{P}(\mathcal{F}(\alpha)\text{is } w\text{-normal}) = 1$.

---

[a] We remind the reader of definition 2.2 for normality up to ϵ

Lemma 2.13:

Let α be a normal number then $\mathcal{F}(\alpha)$ is normal with probability 1.

Proof

By lemma 2.12 $\forall w \in \{0,1\}^*$, $\mathcal{F}(\alpha)$ is w-normal with probability 1. $\mathbb{P}(\mathcal{F}(\alpha)\text{is normal}) = \mathbb{P}(\forall w \in \{0,1\}^*, \mathcal{F}(\alpha)\text{is } w\text{-normal})$, since this is an intersection of countably many event of probability 1, we have that $\mathcal{F}(\alpha)$ is normal with probability 1.

Lemma 2.14:

For any positive integer D, any $k \in [2^{D-1}; 2^D]$

$$\mathbb{P}_\rho(I^D_k(\alpha,\rho)\text{is normal}) = 1.$$

Proof

Let D be a positive integer, $k \in [2^{D-1}; 2^D]$. There are 3 kinds of bits in $I^D_k(\alpha,\rho)$: bits from α, bits from ρ appearing inside the gadgets (we call this sequence γ) and bits from ρ corresponding to return bits (we call this infinite sequence of bits τ). Note that γ and τ are both independent fair random infinite sequences.
Note that in $I^D_k(\alpha,\rho)$, we find every bit from α and τ at indices multiple of $D+1$. We define the infinite sequence y as such: $\forall i \in \mathbb{N}, y_i = I^D_k(\alpha,\rho)_{i(D+1)}$.
Notice that $I^D_k(\alpha,\rho) = I^D(y,\gamma)$ (where the second I is from defintion 2.6). Since γ is a fair infinite random sequence then by using theorem 2.6 if y is normal then so is $I^D_k(\alpha,\rho)$ with probability 1 over γ.
Thus now we only need to show that y is normal with probability 1. Notice that the

distribution of y is the same as $\mathcal{F}_q(\alpha)$ with $q = 1 - \frac{k}{2^D}$. Therefore by theorem 2.13 y is normal with probability 1.

This gives the main theorem. The proof follows the proof of Theorem 2.2, using Lemma 2.14 and and Lemma 2.8.

Theorem 2.3:

Let $\alpha \in \{0,1\}^\omega$ be a sequence. Then α is normal if and only if for all rational selector $\mathcal{S}$ the probability that $\mathcal{S}(\alpha)$ is either finite or normal is equal to 1.

While we think the equivalent statement to hold for general probabilistic selectors, we believe that establishing such a result would require a different proof method. We briefly explain why next. Given a fair random coin X simulating Y such that $P(Y = 0) = a = 1 - P(Y = 1)$ and a is irrational is possible as follows, write the binary decomposition of a in binary $a = 0.a_0a_1\ldots$. Next draw X repeatedly and see the sequence of draws (X_i) as a real number in $[0;1]$ whose (X_i) is its binary decomposition. As soon as we encounter j such that (X_j) differs from a_j we can conclude if (X_i) is bigger or less than a and compute Y accordingly. Back to determinizing automata, the issue with this way of simulating an irrational draw would require gadgets of infinite size and Agafonov theorem does not apply to infinite automata.

Chapter 3

Non-extensions of the Agafonov theorem

In this section, we study necessary and sufficient condition for languages to adhere to Agafonov theorem (Theorem 2.1). To that end we naturally introduce the notion of Agafonov languages (Definition 3.1) and ponder over possible necessary *or* sufficient conditions for languages to be Agafonovesque. Extending Agafonov result to languages other than regular has been studied before [KW75] [WK19], to the best of our knowledge languages presented here do not appear elsewhere in the literature (except language of 3.5 which is folklore).

Definition 3.1: Agafonov languages

Let L be a language, L is said to be Agafonovesque (or to be an Agafonov language, or to verify Agafonov' theorem) if $\forall \alpha \in \{0,1\}^\omega, \alpha$ normal $\Rightarrow L(\alpha)$ normal.

One important property which makes the Agafonov theorem true for automata is that the path a random infinite string takes through the automaton visits accepting states a constant fraction of the time. Said differently starting from an final state of automaton A and exploring A using a random string, we wait on average a fixed amount of time before *returning* to a final state (as stated later in lemma 3.1). By extending this notion of random walk to languages, and defining the *expected return time* (ERT) of a language we had the intuition that this new notion may yield necessary or sufficient conditions for a language to be Agafonovesque. We properly define return time in section 3.1

In sections 3.2 to 3.4, we go over several natural conjectures involving ERT and Agafonov languages. We try to give intuitions for the conjectures and subsequently disprove them. We believe the languages we build as counter-examples to the conjecture are interesting as they point towards difficulties worth knowing towards extending Agafonov theorem.

Notations : Let X be a random variable (r.v.) $\mathbb{E}(X)$ denotes its expectation.

3.1 Return time

As we explain in 3.2 we had at some point the conjecture that languages abiding by Agafonov theorem (see 3.1) all verified some condition about the average expected return time in the language. In this section we introduce the notions of return time and tree of a language which will be used in the following sections.

Definition 3.2: Binary labelled tree

- A finite binary labelled tree $\mathfrak{T}$ is a strict binary tree (each node has at 0 or 2 sons) such that each node t is labelled 0 or 1. The label of node t is written $\mathfrak{T}(t)$. The nodes labelled 1 are called accepting nodes. A node is represented by a bitstring in the intuitive way (see fig. 3.1).
- An infinite binary labelled tree $\mathfrak{T}$ is a complete infinite tree (each node has exactly two sons, there are no leaves) and each node t is labelled 0 or 1.

Definition 3.3: Tree of a language

To each language $L \subset \{0,1\}^*$ we may bijectively associate its infinite labelled binary tree $\mathfrak{T}(L)$ such that for all $t \in \{0,1\}^*$, $\mathfrak{T}(t) = 1$ if and only if $t \in L$.

Next, we define oblivious languages, a standard concept, where the membership of a string depends solely on its length.

Definition 3.4: Oblivious languages

A language L is said to be oblivious if $\forall n \in \mathbb{N}, \forall w \in \{0,1\}^n, w \in L \Leftrightarrow 1^n \in L$.

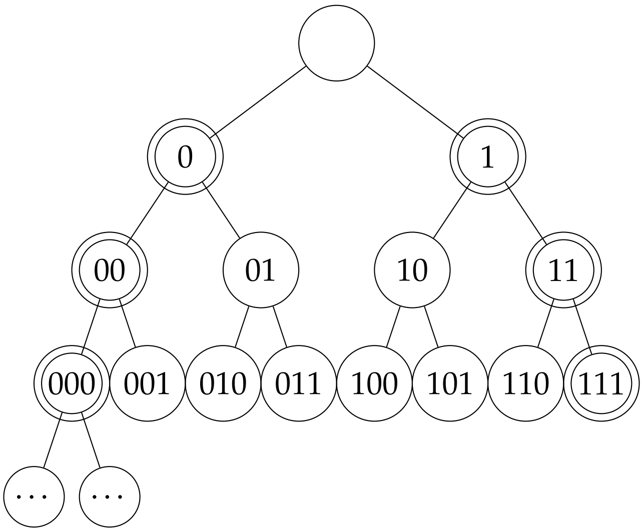


Figure 3.1: Infinite tree cutoff at depth 3 (accepting nodes are doubly circled)

We can now introduce the return time and its derivates.

Definition 3.5: Return time of a node

- Let $w \in \{0,1\}^*$, let L be a language. We construct the random variable W as such: $W \overset{\text{def}}{=} w \cdot r_1 r_2 \dots r_N$ where r_i are random fair bits and $N \geq 1$ is a random variable defined as the smallest integer such that $W \in L$.
  N is called the return time for node w.
- The expected return time $ERT_L(w)$ is defined as $\mathbb{E}(N)$.
- For any $k \in \mathbb{N}$, the k-moment of the return time at node w noted $R_L^k(w)$ is defined as $\mathbb{E}(N^k)$

When the language L is clear from context we may write $ERT(w)$. If one sees the language L as its tree $\mathfrak{T}(L)$, and starts exploring $\mathfrak{T}(L)$ at random starting from w, $ERT_L(w)$ can be seen as the expected time before this process gets back to a node labeled 1 in $\mathfrak{T}(L)$ (the expected time before we *return* in L). Note that the ERT may be infinite.

Definition 3.6: Expected return time in a tree for a node

- (For an infinite tree) Let $\mathfrak{T}$ be a labeled infinite tree, let L be the language such that $\mathfrak{T}(L) = \mathfrak{T}$, then $\forall w \in \{0,1\}^*, ERT_{\mathfrak{T}}(w) = ERT_L(w)$.
- (For a finite tree) Let T be a labeled finite tree. Let $\mathfrak{T}$ be an infinite binary labelled tree extending T in the natural way : all the nodes from T keep their label and all new nodes are labelled 0 then for all w node of T, $ERT_T(w) \overset{\text{def}}{=} ERT_{\mathfrak{T}}(w)$

Now we want to examine how expected return time grows as we go deeper along the tree.

Definition 3.7: Expected return time

- Let L be a language, the expected return time at depth $d \in \mathbb{N}$ noted $ERT_L(d)$, is said to be greater (resp. less) than $a \in \mathbb{R}$ if

$$\forall w \in \{0,1\}^d \cap L, ERT_L(w) > a \text{ (resp. } ERT_L(w) < a)$$

- Let L be a language, the k-moment time at depth $d \in \mathbb{N}$ noted $R_L^k(d)$, is said to be greater (resp. less) than $a \in \mathbb{R}$ if $\forall w \in \{0,1\}^d \cap L, R_L^k(w) > a$ (resp. $R_L^k(w) < a$).
- By conflating infinite binary labelled trees and languages we may speak of $ERT_{\mathfrak{T}}(d)$ where $\mathfrak{T}$ is an infinite binary labelled tree.
- Let $f : \mathbb{N} \mapsto \mathbb{N}$ we note $ERT_L = O(f)$ (or abusively $ERT_L(d) = O(f(d))$) if $\exists c, ERT_L(d) < cf(d)$. We have similar definitions for Θ and ω.

Notice the expected-return-time definition only takes into account words that are in L. Indeed let us take the oblivious languages $L = \{w \in \{0,1\}^*; \exists n \in \mathbb{N}, |w| = n^2\}$, we intuitively want to say that the return time of this language is linear (from an accepting node to another there is a path of length $2n+1$), but for words of length $n^2 - 1$ their ERT is 1, thus we only consider words already in the language. Secondly we don't have a precise value for ERT_d as at a given depth d there are 2^d nodes of possibly different ERT and its not clear how we should define expected return time at depth d.

Definition 3.8: Agafonov languages

Let L be a language, L is said to be Agafonovesque (or to be an Agafonov language, or to verify Agafonov' theorem) if $\forall \alpha \in \{0,1\}^\omega, \alpha$ normal $\Rightarrow L(\alpha)$ normal.

3.2 Languages of unbounded ERT

As said in the beginning of this chapter, one important property which makes the Agafonov theorem true for automaton is that if the automaton is in a strongly connected component containing an accepting state it will go infinitely many times through this accepting state (at least for normal or random infinite sequences). Even more than that, the automaton will go through this state a constant fraction of the time, as per theorem 3.1. In this section we try to understand if this is a sufficient condition.

Theorem 3.1: Automaton have finite return time

Let A be an automaton where every state is co-accessible (there is a path from this state to an accepting state), then $ERT_{L(A)} = \theta(1)$.

Proof

A random walk starting over automaton A and with a random string as input can be seen as a random walk on a Markov chain. Thus any state (including accepting state) are visited in finite expected time.

We expected languages not verifying this property, i.e. going through accepting states less and less frequently, to not abide by Agafonov' theorem, in a way we describe next.

Coming up with the conjecture We had at some point this conjecture : let L be a language, if L is agafonovesque then it cannot be that $ERT_L = \omega(1)$. The intuition behind this conjecture lies both in theorem 3.1 and in the proof of theorem 3.2 which states that this is indeed true for oblivious languages. It turns out that this intuition is wrong and there are agafonovesque languages of arbitrary ERT (theorem 3.3).

Theorem 3.2: Oblivious languages of superconstant ERT are not Agafonovesque

Let L be an oblivious language, f a function from $\mathbb{N} \mapsto \mathbb{N}$ such that $f = \omega(1)$, if $ERT_L = f$, then L is not agafonovesque.

Proof

We first say that $ERT_L = f$ is properly defined because L is oblivious so all accepting nodes at the same depth have the same ERT. To prove the theorem, we have to find a word $\alpha \in \{0,1\}^\omega$ such that α is normal but $L(\alpha)$ is not. The basic idea is to take any normal number and change only the bits right after an accepting state.

Take any normal word $\beta \in \{0,1\}^\omega$, define $w_0 \overset{\text{def}}{=} \epsilon$ and for all $i > 1$, w_i is defined as follows :

- Let $N \overset{\text{def}}{=} |w_{i-1}|$, let $w'_i \overset{\text{def}}{=} w_{i-1} \cdot \beta_{N+1} \cdots \beta_{N+h}$ where h is the first occurrence such that $w'_i \in L$. Then $w_i \overset{\text{def}}{=} w_i \cdot 0$

Let α be defined as the limit of $(w_i)_{i \in \mathbb{N}}$. By construction the bits selected by $L(\alpha)$ are all 0, and we select infinitely many bits because L is oblivious and L is infinite, thus $L(\alpha) = 000 \cdots$ which is not normal.
On the other hand $\alpha_{<N}$ is equal to $\beta_{<N}$ except for indices $(\sum_{j=0}^{n} f(j))_{n \in \mathbb{N}}$, since $|\{i <$

$N; \exists n \in \mathbb{N}, i = \sum_{j=0}^{n} f(n)\}| = o(N)$, one can easily check that the normality of β entails that of α.

Next we prove the main theorem of this section, disproving our original conjecture.

Theorem 3.3: Agafonov languages of arbitrary ERT

For any $E : \mathbb{N} \mapsto \mathbb{N}$, there exists a language L such that $ERT_L = \theta(E)$ and L is Agafonovesque.

Proof

This is a direct application of corollaries 3.1 and 3.2.

In order to prove theorem 3.3 we introduce several constructions

Definition 3.9: Rack and Vine trees

Let $k : \mathbb{N} \mapsto \mathbb{N}$, $E : \mathbb{N} \mapsto \mathbb{N}$ be two functions. Forall n:

- (Racks) Let $B_{k,E}(n)$ be the complete binary tree of depth $E(n)2^{k(n)}$ such that all leaves are accepting nodes (see figure 3.2)
- (Vines) Let $F_k(n)$ be the labelled tree such that (see figure 3.3)
  - $\forall 0 \leq i \leq n, 0^i$ is a node, $0^i 1$ is a node,
  - for two nodes u, v in $F_k(n)$, u is a child of v iff $v = u0$ or $v = u1$
  - the accepting nodes are exactly the $0^i 1$ for $i \leq k(n)$

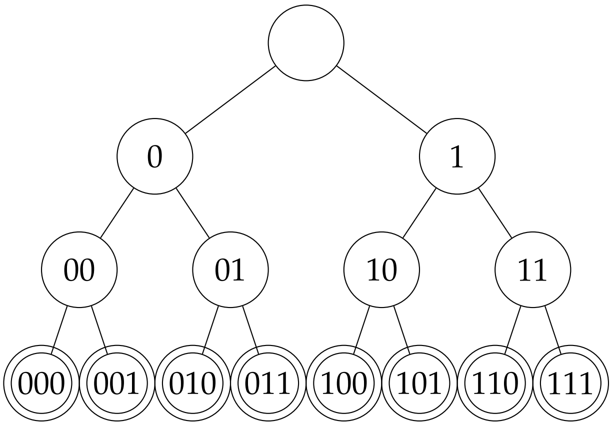


Figure 3.2: A rack of depth 3

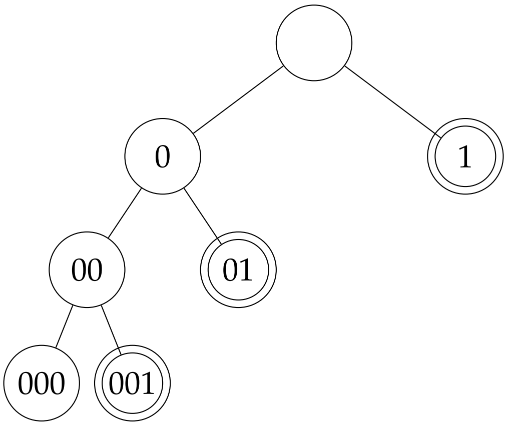


Figure 3.3: A vine of depth 3

Definition 3.10: Gluing/Sticking

Take T and H two binary labelled trees such that T is finite, H is not necessarily finite. Let w be a leaf of T, gluing H to node w yields the tree G composed of all nodes in $\{w'; (w' \text{ is a node of } T) \vee (w' = w \cdot h \wedge h \text{ is a node of } H\}$. The labels are defined as follows $G(w') = H(h)$ if $w' = w \cdot h$ otherwise $G(w') = T(w')$.

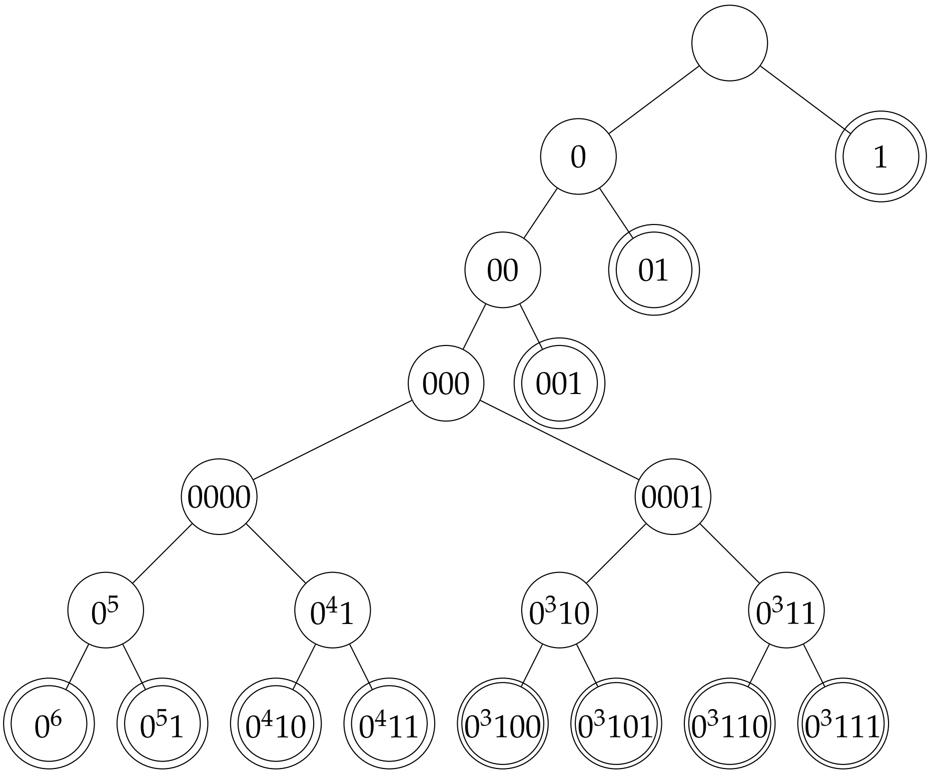


Figure 3.4: Gluing a rack to a vine

We have defined racks and vines such that when glued together the expected return time is $E(n)$ (see fig. 3.4). We make use of it to construct trees $\mathfrak{T}_{k,E}$ which are counter examples for theorem 3.3.

Definition 3.11: Construction of $\mathfrak{T}_{k,E}$

- We define $T_{k,E}(n)$ to be the finite tree obtained by sticking the root of $B_{k,E}(n)$ at the node $0^{k(n)}$ of $F_k(n)$. i.e. $T_{k,E}(n)$ is the tree which represents the language composed of all strings x such that $x = 0^i1$ for $i < k(n)$ or $x \in 0^{k(n)}(0+1)^{z(n)}$ where $z(n) = E(n)2^{k(n)}$.
- We define $\mathfrak{T}_{k,E}$ to be the *infinite* tree defined as the limiting $\mathfrak{T}_i$ as i tends towards infinity where $\mathfrak{T}_i$ defined as such: At step $0:$ $\mathfrak{T}_0$ is $T_{k,E}(0)$ At step $i:$ For each leaf f of $\mathfrak{T}_{i-1}$, call d_f the depth of f, glue tree $T_{k,E}(d_f)$ to leaf f. The resulting tree is $\mathfrak{T}_i$.

Next theorem justifies our construction by saying that the expected return time from the root of the tree $T_{k,E}(n)$ is equal to $E(n)$ (in the next theorem ϵ represents the empty word).

Lemma 3.1: $ERT(T_{k,E})$

$\forall n \in \mathbb{N}$ we have $E(n) \leq ERT_{T_{k,E}(n)}(\epsilon) \leq E(n) + 2$

Proof

This is done by direct analysis

$$ERT_{T_{k,E}(n)}(\epsilon) = \left[\sum_{i=0}^{k(n)-1} i2^{-i}\right] + 2^{-k(n)} \cdot ERT_{B_{k,E}(n)}(\epsilon)$$

$$\boxed{\text{the sum accounts for the vine part}}$$

$$= \left[\sum_{i=0}^{k(n)-1} i2^{-i}\right] + 2^{-k(n)} \cdot 2^{k(n)} E(n)$$

This quantity is more than $E(n)$ and less than $2 + E(n)$ (by convergence of $\sum_{i=0}^{N} i2^{-i}$)

Corollary 3.1:

Let k, E be two functions of $\mathbb{N} \mapsto \mathbb{N}$, met $d \in \mathbb{N}, ERT_{\mathfrak{T}_{k,E}}(d) = \theta(E(d))$

Proof

Let $w \in \{0,1\}^d$, if w is accepting then by construction of $\mathfrak{T}_{k,E}$, tree $T_{k,E}(d)$ starts at w. By lemma 3.1 we then get the desired result

Corollary 3.2:

Let E be a function from $\mathbb{N} \mapsto \mathbb{N}$, then there exists $k \in \mathbb{N} \mapsto \mathbb{N}$, such that the language $L(\mathfrak{T}_{k,E})$ is Agafonovesque.

Proof

Let E be a function from $\mathbb{N} \mapsto \mathbb{N}$. Let k be the function defined as such $\forall n \in \mathbb{N}, k(n) = 2n + 1$. Then we claim $L(\mathfrak{T}_{k,E})$ is Agafonovesque.
Recal that $\mathfrak{T}_{k,E}$ it is built by gluing racks ($B_{k,E}$) and vines together F_k). Let $\alpha \in \{0,1\}^\omega$ be an infinite sequence (not necessarily normal at this point). There are two possibilities as we examine the path of α through $\mathfrak{T}_{k,E}$:

- α takes finitely many racks i.e. $\exists n_0, \forall n, n \geq n_0 \Rightarrow \alpha_n$is in a vine of $\mathfrak{T}_{k,E}$. Then as far as α is concerned, starting from n_0, it is as if it is being selected by a finite automaton (the finite automaton which accepts each time it sees a 1), therefore by Agafonov theorem if α is normal $\mathfrak{T}_{k,E}(\alpha)$ is normal.
- α takes infinitely many racks, i.e. $\forall n \in \mathbb{N}, \exists m \in \mathbb{N}, m > n \wedge \alpha_m$is the root of $B_{k,E}(i)$ for a certain i. If we are in this situation we claim that α is not normal. Indeed let us consider m a integer such that the node α_m of $\mathfrak{T}_{k,E}$ is the beginning of rack $B_{k,E}(i)$ for a certain i then it must be that for a certain d, the last $k(d)$ bits of $\alpha_0, \ldots, \alpha_m$ were all 0 (because we go through a vine before arriving at a rack). Moreover d verifies $d + k(d) = m \Rightarrow 3d + 1 = m \Rightarrow d \approx m/3 \Rightarrow k(d) \approx 2m/3$. Thus we have that the fraction of 0 in $\alpha_0, \ldots, \alpha_m$ is bigger than $2/3$, since this happens for infinitely many m, α cannot be normal.

In conclusion if α is normal then $\mathfrak{T}_{k,E}(\alpha)$ is normal.

3.3 Almost regular languages of finite moments

If we analyze regular languages we can note that from any accepting state not only is the expected return time finite (as well any any k-moment) but we also have only access to finitely many states. If we look at the infinite labelled tree of a regular language the tree becomes self similar for almost all nodes as we descend deeper in the tree (the tree becomes self similar as soon as we enter a strongly connected component of the automaton). To mimic this self similarity we come up with the notion of *View*, the *View* of a node w in a tree what the node *sees* as it looks down the tree. We try to come up with a sufficient condition for Agafonov languages in this direction. If a language has finitely many views across all $w \in \{0,1\}^*$ then the language is regular and thus the Agafonov theorem is already established. To go sightly beyond we consider views until a certain depth called $View_n$ and consider languages with finitely many $View_n$ and call those almost regular languages (see the exact definition here 3.13). One conjecture could be that almost regular languages with finite k-moments for all k are agafonovesque but we disprove this conjecture in theorem 3.4. Assuming finite k-moments is important as otherwise there are trivial counter example to this conjecture. We have not seen the notion of almost regular language defined elsewhere and think it is interesting in its own right.

Definition 3.12: View of a language

$View_n(w, L) = \{w' \in \Sigma^{\leq n}; ww' \in L\}$

Definition 3.13: Almost regular languages

Let L be a language it is said to be almost regular iff $\exists C \in \mathbb{N}, \forall n \exists M, |\{View_n(w, L); w \in \Sigma^{\geq M} \bigcap L\}| \leq C$

Not all regular languages are regular as implied by theorem 3.4, but the converse is true.

Lemma 3.2: Regular languages are almost regular

Let L be a language, if L is almost regular then it is regular.

Proof

Let L be a regular language, it is recognized by automaton A with at most k states, therefore $\forall n \in \mathbb{N}, |\{View_n(w, L); w \in L\}| \leq k$ which is a stronger requirement than necessary for being almost regular.

Theorem 3.4: Almost regular languages of finite moments but not Agafonovesque

There exists an almost regular language, with finite k-moments for all k i.e. $\forall k \in \mathbb{N}, \exists C \in \mathbb{N}, \forall w \in L, R_L^k(w) < C$ (R is from definition 3.5) which does not abide by Agafonov theorem.

Proof

Let $k = n \mapsto n$, $E = n \mapsto \frac{n^3}{2^n}$, then $\mathfrak{T}_{k,E}$ verifies that:

- $\forall k \in \mathbb{N}, \exists C \in \mathbb{N}, \forall w \in L, R_L^k(w) < C$

- There exists a $\alpha \in \{0,1\}^\omega$ normal such that $T(\alpha)$ is not normal.
- $\mathfrak{T}$ is almost regular.

Proofs:

- Let $w \in L$, call $n \overset{\text{def}}{=} |w|$ then we have $R^k_L(x) < 2^{-n} * (n + n^3)^k$ which is strictly less than $C \in \mathbb{N}$ for a certain C depending on k
- Consider an infinite sequence $\alpha' \in \{0,1\}^\omega$ which is the limit of sequence $(\alpha_i) \in (\{0,1\}^*)^{\mathbb{N}})$ constructed as follows. At step 0, $\alpha_0 = \epsilon$. At step i, draw $r \in \{0,1\}^*$ of size $|\alpha_{i-1}|^3$ at random, we define $\alpha_i \overset{\text{def}}{=} \alpha_{i-1}0^{|\alpha_{i-1}|}r0$. The $0^{|\alpha_{i-1}|}$ part ensures that we get always get to a rack, and the random r ensures normality, the final 0 will be the selected bit and make the output not normal. It can be easily checked that α' is normal (idea : let $n, N \in \mathbb{N}$ if you draw index i at random in $[0;N]$ $\alpha'_i, \dots \alpha'_{i+n}$ were all drawn like random bits with probability closer to 1 as N tends to infinity). Moreover $\mathfrak{T}_{k,E}(\alpha') = 00000....$
- The idea is this : given $n \in \mathbb{N}$ at depth more than n every accepting node only sees racks for the next n nodes. Formally:
  Fix $C = 1$, let $n \in \mathbb{N}$, fix $M = n$ then $\forall w \in \{0,1\}^{\geq M} \cap L, View_n(w, L) = \{w' \in \{0,1\}^*; w' \in (\epsilon + (0+1)^{i-1}1) \wedge i \in [0;n]\}$ thus $\forall w \in \{0,1\}^{\geq M} \cap L, |\{View_n(w, L); w \in \Sigma^{\geq M} \bigcap L\}| \leq C$.

3.4 Languages of bounded return time

In the previous section we crucially relied on the fact that the return time of our language was not bounded (it was finite but could grow as we went down the tree). One may think that having a bounded return time must be a sufficient condition for being agafonovesque but as we show in Theorem 3.5 this is not true. Let us first recall the definition of bounded return time.

Definition 3.14: Bounded return time

Let L be a language, it is said to have a bounded return time if $\exists C \in \mathbb{N}, \forall w \in \{0,1\}^*$ if we call N the return time at node w (N is a random variable), then we have $N < C$. This is equivalent to saying $\exists C \in \mathbb{N}, \forall w \in L, \forall w' \in \{0,1\}^C, \exists i \leq C, w \cdot w'_1 \cdot w'_2 \cdots w'_i \in L$

Theorem 3.5:

There exists languages with bounded return time which are not Agafonovesque.

Proof

Let $\alpha \in \{0,1\}^\omega$ be a normal sequence. Define program p_α as follows :

```
Input: $w$
$n \leftarrow |w|$
if $w$ is not a prefix of $\alpha$ then
    ACCEPT
else
    if $\alpha_{n+1} = 0$ then
        ACCEPT
    else
        if $\forall i \in [1,5], \alpha_{n-i} \neq 0$ then
            ACCEPT
        end if
    end if
end if
```

Let L_α be the language recognized by p_α. Basically p_α accepts right before the zeros of α and if it hasn't been able to do that for the 5 last prefixes of α it accepts no matter what.

Clearly L_α has finite return time, that is because if we haven't accepted for the past 5 prefixes accept. Hence the return time is bounded by 5.

Moreover $L_\alpha(\alpha)$ is not normal. We sketch the proof : notice that $L_\alpha(\alpha_{<N})$ selects all 0 in $\alpha_{<N}$ so approximately $N/2$ zeroes as N goes to infinity, but it only selects the 1 which come right after a sequence of five consecutive ones, so only $1/32$ of them on as N goes to infinity. Thus the limiting ratio of 1 is not $1/2$ and thus $L_\alpha(\alpha)$ is not normal.

The language in the proof of theorem 3.5 is not almost regular. Our guess is that any almost regular language of bounded return time is Agafonovesque... But only because such languages are regular. While we do not believe this conjecture to be hard to prove we leave it for future research.

Question

Let L be a language, if L has bounded RT and is almost regular then it is regular.

Part II

Kolmogorov complexity

Chapter 4

Time hierarchy for Kolmogorov complexity

(this part should really be revisited (in particular it's log alpha in the intro and not alpha)

In this part we improve on the state-of-the-art time hierarchy for time-bounded Kolmogorov complexity. More precisely, we prove that there are infinitely many $n \in \mathbb{N}$ such that there are strings of size n which are the output of programs of size $f = o(\log n)$ running in $o(2^f T)$ steps but are not the output of any program of size f running in T steps. The previous gap was exponential in 2^f. This result is established by studying a new problem in combinatorics which we call novelty games: a list E of pk numbers in $[1; N]$ is shared evenly between p players, who each get to output an answer depending on their share of E. The players must establish a communication-free strategy ensuring that one of them outputs a number not belonging to E.

4.1 Introduction

What is Kolmogorov Complexity ? Kolmogorov complexity, named after Andreï Kolmogorov, is a measure of the complexity of a word. More precisely, the Kolmogorov complexity of a string $w \in \{0,1\}^*$, denoted as $K(w) \in \mathbb{N}$, is the length of the shortest program program (in a fixed programming language) that outputs w and halts. It provides a way to quantify the information content of a string by the length of its shortest description. For example, a string that consists of repeated patterns will have a lower Kolmogorov complexity than a random string of the same length. Let us give some examples for the unfamiliar reader. In the following paragraph we will abusively conflate programs with their binary descriptions.

- The string 1^n, which consists of n consecutive ones, has an upper bound on its Kolmogorov complexity of $K(1^n) \leq \log n + 0(1)$, since it can be described by a simple program specifying the number n in binary and a *for* loop (the *for* loop accounts for the $O(1)$ additional bits).

- The first n bits of the binary expansion of π have an upper bound of $K(\pi_1, \ldots, \pi_n) \leq \log n + O(1)$, because π is a well-known mathematical constant that can be generated by a fixed algorithm with a short description. By attaching n written in binary to a description of this algorithm we get the upper bound.

- Any string x of size n has Kolmogorov complexity less than $n + O(1)$ as you can always hardcode x and return it.

- For "almost all" random strings, their shortest description is the string itself plus a small constant overhead for the program (this constant may depend on the specific programming language, for instance instructions telling the program to print and stop). This fact can be proven by a simple counting argument : each program of size less than n prints at most one string, there are less than 2^{n+1} such programs but there are 2^{n+10} strings of size $n+10$, thus 99% of strings in $\{0,1\}^{n+10}$ have a Kolmogorov complexity of at least n.

Note that given a string $w \in \{0,1\}$ we may in general only upper bound its Kolmogorov complexity : we may know that there is a program of a certain size outputting w and stopping but we do not know if it is indeed the smallest one. To prove that the exact value of the Kolmogorov complexity of a string w is k we would have to prove that no programs of size less than k can output w. But the seemingly only way to do that would be to run all programs of size less than k and check that they don't output s. The issue is that with no upper time limit on the running time the programs of size less than k, how can we differentiate between a program running for very long and eventually outputting w and a program which never halts ? One seems to run into the halting problem which is known not to be computable... This argument can actually be made formal: there are no computable functions f growing to infinity such that $\forall w \in \{0,1\}^*, f(w) < K(w)$.

Before going any further, we want to emphasize the robustness and universal aspect of Kolmogorov complexity. Up until now, we have discussed Kolmogorov complexity without specifying the programming language used to describe the programs. This is because Kolmogorov complexity does not depend (up to a constant) on the specific choice of the programming language, as long as the language is Turing complete. This property implies that the complexity of a string s in any two Turing-complete languages differs by at most a constant independent of s. In other words, if $K_L(s)$ and $K_M(s)$ denote the Kolmogorov complexity of s in languages L and M, respectively ($K_L(s) = min\{|p|; L(p) = s\}$), then there exists a constant c_{LM} such that $\forall s, |K_L(s) - K_M(s)| \leq c_{LM}$. To sketch a proof of this, consider two Turing-complete languages L and M. Because L is Turing complete, there exists a program (a compiler) that translates any program written in M into an equivalent program in L. This means that for any program p in M that outputs s, there is a corresponding program in L whose length is at most the length of P plus a constant c_{LM}, which is the size of the compiler. Hence, $K_L(s) \leq K_M(s) + c_{LM}$. By symmetry, there also exists a constant c_{ML} such that $K_M(s) \leq K_L(s) + c_{ML}$.

Another property of Kolmogorov complexity relevant to our result is that for a certain $c \in \mathbb{N}$ there are strictly more strings output by programs of size $f + c$ than by programs of size f. This result is easily shown by considering random strings. A random string s of size $f + c$ is with high probability only output by a program of size at least $f + c$, but (for a well chosen c depending on the language) all strings of size f are output by a program of size less than $f + c$. This gives rise to a hierarchy in Kolmogorov complexity and by denoting $[f] = \{s \in \{0,1\}^*; K(s) \leq f\}$, we have that $[f + c] \not\subset [f]$, or informally $[f + c] > [f]$. In an intuitive sense we may say that there is a strict hierarchy between Kolmogorov complexity classes.

Prefix-free Kolmogorov complexity We mention as a clarification for the educated reader that we do *not* deal in this chapter with prefix free complexity, despite the notation K sometimes used for this notion. However, we have no reason to believe our techniques wouldn't hold for prefix free complexity.

Time bounded Kolmogorov complexity An aspect of Kolmogorov complexity is that it does not take into account the computational resources required to generate a string from its shortest description. To address this, the concept of time-bounded Kolmogorov complexity is introduced. Time-bounded Kolmogorov complexity, denoted as $K^t(w)$, considers both the length of the shortest program that outputs the string w and the time t within which the program must run. Informally, $K^t(w)$ is the length of the shortest binary program (in a fixed language) that produces w within a specified time bound t.

We have seen before that for plain Kolmogorov complexity the choice of any universal programming language has no significant bearing on the size of the smallest programs, up to constant term. For the time version we can likewise show that the notion is robust in the same sense : Let L be a Turing machine, let us denote $K^t_L(s) = min\{|p|; L(p) = s \text{and } L(p) \text{ stops in less than } t \text{ steps}\}$. We can show that for a universal Turing machine $\mathbb{U}$ abiding by theorem 1.2

$$\exists c, \forall w \in \{0,1\}^*, K^{ct\log t}_{\mathbb{U}}(w) \leq K^t_L(w).$$

We thus have a robust notion up to a log-factor slowdown. We fix a universal Turing machine $\mathbb{U}$ abiding by theorem 1.2 and we formally define forall string $w \in \{0,1\}^*$,

$$K^t(w) \overset{\text{def}}{=} min\{| < \#M, x > |; \mathbb{U}(< \#M, x >) = w \text{ and } \mathbb{U}(< \#M, x >) \text{ stops in less than } t \text{ steps}\}$$

.

We remind the reader that M denotes a Turing machine (or program), and $\#M$ is its description. Notice that the definition of K^t complexity is based on the running time of the Universal Turing machine, i.e. the number of *simulated* steps.

We can define the set $[f, T](n)$ as the set of strings of length n output by a program of length $f(n)$ running in time $T(n)$ (when simulated through $\mathbb{U}$).

RAM model The notions of Kolmogorov complexity and time bounded Kolmogorov complexity can be adapted simply to RAM machine rather than Turing machine. In this case since simulation through the universal RAM machine bears only a constant slowdown we can bypass all technical consideration over simulation time.

Context of the research Kolmogorov time bounded complexity has been studied these recent years and used for breakthrough results in meta complexity [Hir+23] [LP20] [Hir18], but not only [Che+23] [LO22]. The results are many and have strong implications regarding existence of OWF amongst other things. In a recent result [Hir+24] it is shown that finding a witness to K^t is as hard as evaluating K^t (under some assumptions), it is somewhat surprising to know this but yet have little insights over existence of a time-bounded-Kolmogorov hierarchy.

Novelty games we introduce in our this paper seem quite similar to a problem known as range avoidance problem: The Range Avoidance (AVOID) problem is: given a Boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$ for some stretch $m > n$, find an element $y \in \{0,1\}^m$ outside the range of C. This paper [RSW22] show some implications over the complexity of AVOID, in this paper [Gaj+23] a fast algorithm is given for AVOID for circuits of NC^3. We believe links between Novelty games and the AVOID problem should be studied, this thesis does not as the author was not aware of the existence of this problem at the (allocated) time of writing.

Question

What links are there between the range avoidance problem and novelty games ?

Neologisms for simulating programs In order not to say "program" a bigillion times we introduce the following nomenclature : program which simulates other programs are called *simulants*, their output is called the *outant*. Programs which are being simulated are called *simulees*, their output is called *outee*.

Nomenclature In order to alleviate any ambiguity we introduce the following notations: p will denote a program. $\#n$ will denote a representation of $n \in \mathbb{N}$, i.e. a program which outputs n, some additional argument may be added to the representation (e.g. running time). $p(x)$ denotes the program p launched on input x (x is a placeholder name), in particular $|p(x)|$ denotes the size of program p launched on x, so it is of size $|p| + |x| + \log p$. $\boldsymbol{\mathcal{O}}\mathbf{ut}\boldsymbol{(}p(x)\boldsymbol{)}$ denotes the output of program p launched on x.

For a property $A(n)$ depending upon integers: $\exists_\infty n, A(n) \overset{\text{def}}{\equiv} \forall N \in \mathbb{N}, \exists n \in \mathbb{N}, n > N \wedge A(n)$ and $\forall_\infty n, A(n) \overset{\text{def}}{\equiv} \exists N \in \mathbb{N}, \forall n \in \mathbb{N}, n > N \Rightarrow A(n)$.

A hierarchy for K^t complexity We have seen that for plain Kolmogorov complexity, there exists a hierarchy, i.e. for any $f \in \mathbb{N}$ there exists words having a description of size $f + c$ but no description of size f. Can this result be extended when considering K^t complexity? Meaning, for arbitrary $f, T \in \mathbb{N}$, are there words which can be printed by programs of size f running in $2T$ steps but not by programs of size f running in T steps? Stated more generally: For arbitrary $T_1, T_2 \in \mathbb{N}$ how do $[f, T_1](n)$ and $[f, T_2](n)$ compare? A first immediate remark is that we have to restrict our considerations to words of a given size, smaller than the allotted time bound. Indeed, the question becomes trivial otherwise: with 2000 time steps one can write the string 1^{1900}, but this is not possible in 1800 time steps. We therefore fix a size for our outputs and only consider time bounds larger than this size.

This open question, quoted in reference textbook [LV19], is a very natural question to ask when it comes to time bounded Kolmogorov complexity. It is similar to the time hierarchy theorem which states that one can solve strictly more decision problems if one allows Turing machines to run for longer.

In his thesis [Lon86], Luc Longpré gives a first answer by proving that with exponentially more time in the size of the programs one can output new strings. We present a sightly different proof version here.

Theorem 4.1: Time hierarchy for K^t complexity (exponential gap) [Lon86]

Let f, T be two reasonable functions from $\mathbb{N}$ to $\mathbb{N}$ in $\mathsf{DTIME}(T(n)$ additionally where $T(n) \geq n$ and $f(n) < n$ and f then

$$\exists c, \exists_\infty n, [f, T](n) \subsetneq [f, c2^{2^f} T](n),$$

If we have additionally that $f = \omega(\log(n))$ then

$$\exists c, \forall_\infty n, [f, T](n) \subsetneq [f, c2^{2^f} T](n),$$

Proof

We adapt slightly the proof of Longpré. The proof is made by diagonalization like in the proof of the time hierarchy theorem 1.3. Here follows a sketch of the proof

We will construct a program p (remember we conflate programs with their binary representation) which takes as input a representation of $n \in \mathbb{N}$ which is decodable in time $2n$ noted $\#n$. I.e. $\#n \in \{0,1\}^*$ is a program which runs in $2n$ steps and outputs n. We intend p to be such that $\mathbf{Out}(p(\#n)) \notin [f,T](n)$ but $\mathbf{Out}(p(\#n)) \in [f, 2^f T](n)$. Here is what program $p(\#n)$ does :

1. Compute the values $n, f(n), T(n)$
2. Iterate over all programs of size $\leq f(n)$ and simulates them for $T(n)$ *simulation* steps on the universal Turing machine U. These 2^{f+1} such programs are called $(h_i)_{i \in [\![0;2^n]\!]})$ and the program stores the output y_i of h_i if there was any and if $|y_i| = n$. Notice that there are at most 2^{f+1} such y_i the y_i are all the elements of $[f,T](n)$
3. Output a word in $\{0,1\}^n$ different from all y_i

Correctness: By construction $\mathbf{Out}(p(\#n))$ cannot be in $[f,T](n)$.

Running time analysis: Let us analyze the running time of $p(\#n)$:

- step 1 runs in time $O(T(n))$
- step 2 in time $O(2^{f(n)}T(n))$
- step 3 in time $O(n2^{2f(n)}) = O(T(n)2^{2f(n)})$ by theorem 4.6.

So the total running time is $O(2^{2f(n)}T(n))$.

Size analysis: We have that $|p(\#n)| \leq |p| + \log(|p|) + |\#n| \leq K^{2T}(n) + O(1)$. Because $f \in \mathsf{DTIME}(T)$ where $T > f$ we have by theorem A.3 that $\exists_\infty n, |p(\#n)| \leq K^{2n}(n) + O(1) \leq f(n)$.
If we know that $f \geq c\log(n)$ then $\forall_\infty |p(\#_{2T}n)| \leq K^{2n}(n) + O(1) \leq \log(n) + O(1) \leq f(n)$.

Conclusion: By correctness analysis $\forall n, \mathbf{Out}(p(\#n))$ cannot be in $[f,T](n)$, by running time and size analysis $\exists_\infty n, \mathbf{Out}(p(\#n)) \in [f, 2^{2f}T](n)$. Thus $\exists_\infty n, [f, 2^{2f}T](n) > [f,T](n)$.
If additionally we have that $f = \omega(log(n))$ then $\forall_\infty n, [f, 2^{2f}T](n) > [f,T](n)$.

Whether one can lower that $2^{2f(n)}$ to something subexponential has been an open question since Longpré's result. The contribution of this thesis is to provide a (partial) positive answer. We now present intuition about how to lower this gap in order to get $[f, \alpha T] > [f,T]$

Longpré proves his result using a standard complexity technique called diagonalization. This is the same technique as in the proof of the time hierarchy theorem (see 1.3). Let us describe it at a high level: one considers a program (called *simulant*) simulating the 2^f programs of size f (called *simulees*) for T steps each, hence the $2^f T$ time factor. The simulees may or may not output a string (called *outee*). The simulant then outputs a string (called *outant*) which is different from all the outees (see figure 4.1).

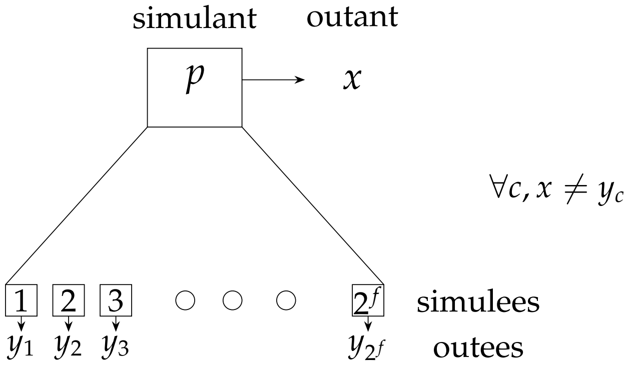


Figure 4.1: Longpré' construction

In order to improve on the result, we exploit the following idea: what if we had $2^f/\alpha$ simulants each simulating α simulees ? (see figure 4.2). The simulants would therefore run for αT, allowing us to get a better bound in the theorem. Because each simulant only knows a fraction of the outees, the problem now becomes ensuring that at least one outant is different from *all* the outees, i.e. $\exists j \in [\![1;\alpha]\!], \forall i \in [\![1;2^f]\!], x_j \neq y_i$. To do that the simulants must execute some kind of strategy to ensure one of them outputs an outant different from all outees. Establishing such a strategy is the main hurdle of this chapter.

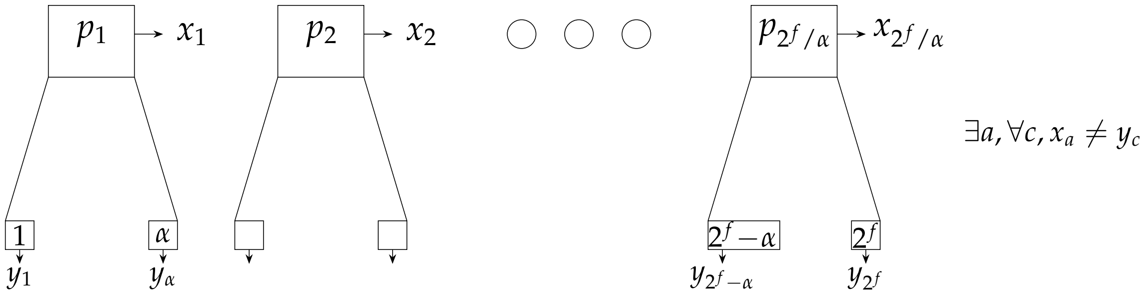


Figure 4.2: Our construction

Forgetting the kolmogorov setting for a moment, the question can be expressed as a new combinatorics problem presented in section 4.4, called novelty games. The problem goes as follows (we put in parenthesis the relating notions in our Kolmogorov problem): a list E of pk numbers (the outees) in $[1; N]$ (size of the outees) is shared evenly between p players (the simulants), who each get to output an answer (an outant) in $[1; N]$ depending on their share of E. The players must establish a communication-free strategy ensuring that at least one of them outputs a number not belonging to E.

We establish in section 4.5 a solution to this combinatorics problem which we then use to get a better Kolmogorov time hierarchy theorem than Longpré when $f = o(\log(n))$. More precisely, we prove that for any function $f = o(\log(n))$, there exists a function $\alpha = o(2^f)$ such that $\exists_\infty n, [f, \alpha T](n) > [f, T](n)$.

The bound $f = o(\log(n))$ and the associated α is directly related to bounds on the solution we obtain for novelty games. Better bounds for novelty games would thus translate (modulo some technicalities) into a better bounded time hierarchy theorem. The bounds come from the fact that N must be bigger than pk in order for a winning strategy to exist, how much bigger determines the bound.

The method described above and represented in figure 4.2 we name them "parallel diagonalization". In section 4.3 in figure 4.3 we present an even more general way of diagonalizing which we call "parallel diagonalization with advice", the "advice" part refers to the fact that we can give some additional information to the programs. We believe this technique may have implications elsewhere in complexity theory. The analysis of novelty games may also be of independent interest to researchers in combinatorics.

Question

Can parallel diagonalization with advice lead to new separation in complexity theory ? To better upper bounds for problems like MKtP ? [a]

---

[a] The MKtP problem ask is a string x if of K^t complexity at least k

4.1.1 Outline

Here is a succinct outline of this chapter. In section 4.2, we establish notations and basic facts. Section 4.3 then provides a high level view of the paper and of our technique. The remaining sections then contain the technical material. in Section 4.4 we define the new combinatorial problem – which we call *novelty games*. We also prove general results and give winning strategies for these games on specific cases. In Section, 4.6 we formally define the program in $[f, c2^f T](n)$ but not in $[f, T](n)$ assuming the existence of a general winning strategy for novelty games. In Section 4.5 we present a winning strategy for novelty games. The results are then combined in Section 4.7 to establish the main theorem of the paper. Lastly, Section 4.8 sketches some directions for future work.

4.2 Definitions

4.2.1 Notations

Notations We will use $\mathbb{N}$ to denote the set of integers. Intervals $\{N, N+1, \dots M\}$ will be denoted by $[N; M]$, and we will abusively write $[N]$ for $[1; N]$. We will write $\binom{[N]}{k}$ to denote the set of multisets of size at most k included in $[1, N]$.

For a given set $\{0,1\}$, we write $\{0,1\}^*$ the set of strings, i.e. finite sequences of elements of $\{0,1\}$. Given x and y in $\{0,1\}^*$, $|x|$ will denote the length of the string x, and we will write xy the concatenation of the strings x and y.

We will abusively speak of Turing machine as programs and vice versa. In the following we interchangeably view integers, programs and bitstrings as the same thing, meaning a bitstring represents an integer and a program and vice versa. The output of program $p(x)$ is noted **Out**$(p(x))$

We write $\forall_\infty n, A(n)$ to express that $A(n)$ is true for all but finitely many n. We will also write $\exists_\infty n, A(n)$ to express that there are infinitely many n such that $A(n)$.

Given two sets S and R we write $S \subsetneq R$ if S is included but not equal to R, we may also write with the same meaning $R > S$.

The function $\log^*(n)$ is defined recursively as

$$\log^* n = \begin{cases} 0 & \text{if } n \leq 1, \\ 1 + \log^*(\log_2 n) & \text{if } n > 1. \end{cases}$$

Naming conventions Time bounds will be written as T, program size bounds as f and time factor as α. Since any word of size n is output by a program of size $n+c$ and you need at least n steps to write it in order to have interesting theorems we will consider that $f \leq n \leq T$. We will often drop the input of functions and for instance abusively write T, f, and α instead of $T(n)$ $f(n)$ and $\alpha(n)$. Functions g and h will denote slow growing functions.

4.2.2 Kolmogorov related notions

We define Time-space bounded Kolmogorov complexity as it is usually defined, for instance in the book of Li and Vitanyi [LV19, section 7.1.2], or in Longpré [Lon86].

Definition 4.1: Time-space bounded Kolmogorov complexity

Let ϕ be a computable function from strings to strings. Let $T \in \mathbb{N}, x \in \{0,1\}^*$. We define $K_\phi^T(x) = min\{|p|; p \in \{0,1\}^* \wedge \phi(p)$ outputs x and stops in T steps or less $\}$

Theorem 4.2: Universal Kolmogorov complexity

There exists a recursive function ϕ_0 such that for every other recursive function ϕ, there is a constant c such that $K_{\phi_0}^{ctlog(t)}(x) \leq K_\phi^t(x)$ for all x. The constant c depends on ϕ only.

Proof

Machine $\mathbb{U}$ of Theorem 1.2 is a witness to this theorem.

In the following we fix $\mathbb{U}$ to be an efficient universal efficient Turing machine (it adheres to theorem 4.2).

Definition 4.2: Universal time-space bounded Kolmogorov complexity

Let $T \in \mathbb{N}^2, w \in \{0,1\}^*$. We define $K^T(w)$ as

$$\min\{| < \#M, x > |; \mathbb{U}(< \#M, x >) = w \text{ in } T \text{ steps or less}\},$$

Note that the input received by the Turing machine $\mathbb{U}$ is treated as the encoding of a pair consisting of a machine and its input, and the running time considered is the one of the universal Turing machine, not original program M. Note that if we worked in the RAM model this distinction would not make any difference, since simulation only incurs a constant slowdonw.

We next state a theorem useful for bounding the K^t complexity of our strings.

Theorem 4.3: K^t complexity of an output

Let $p, x, y \in \{0,1\}^*$, if program p on input x outputs y in t steps then for a $c \in \mathbb{R}$, $K^{ct\log t}(y) < |p| + |x| + O(\log(|p|)$

Proof

$\mathbb{U}(< p, x >)$ outputs y in $ct\log t$ steps and $| < p, x > | \leq |p| + 2\log|p| + 1 + |x|$

Finally we introduce the object of study of this chapter.

Definition 4.3: Class of time-space bounded Kolmogorov complexity with fixed size

Let T, f be functions from $\mathbb{N}$ to $\mathbb{N}$, and $n \in \mathbb{N}$. We define

$$[f, T](n) = \{w \in \{0,1\}^*; |w| = n \wedge K^{T(n)}(w) \leq f(n)\}.$$

Longpré denotes this class by $[f(n), T(n)]$ instead.

Reasonable functions For technical reasons we need our functions not to be too contrived, in particular we need them to be easy to compute and not to be growing too slowly. To that end introduce the two following notions:

Definition 4.4: Very-time-computable functions

Let f be a function from $\mathbb{N} \mapsto \mathbb{N}$, f is very-time-computable (VTC) if there exists a machine M in $\mathsf{DTIME}(f(n)+n)$ such that $\forall n, M(n) = f(n)$.

The usual notion of time computability is not suitable to functions $f < n$ hence our new definition 4.4. Next is our definition controlling the growth rate of our functions.

Definition 4.5: Reasonable functions

A function $f : \mathbb{N} \mapsto \mathbb{N}$ is reasonable if $\forall n, M(n) = f(n)$ and $\forall c, \exists_\infty n, cK^{2n}(n) \leq f(n)$

In appendix A we study reasonable functions. We have a slightly different definition of reasonable functions in the appendix than in definition 4.5, but no matter since the different definitions are shown to be equivalent. Reasonability imposes a certain growth rate for our function but this condition is not stringent at all: as shown in theorem 4.5 $\log^*$ is a reasonable function. For this chapter we only need the following fact whose proof is in the appendix:

Theorem 4.4: VTC functions and reasonability

If f is VTC and increasing and going to infinity then f is reasonable.

In the following all considered functions are considered to be VTC and increasing and going to infinity and reasonable.

Theorem 4.5: Growth rate of K^{2n}

$\forall c \in \mathbb{N}, \exists_\infty n \in \mathbb{N}\,, cK^{2n}(n) \leq \log^*(n)\,.$

Finding a word outside a list We introduce the problem OUTSIDE and efficient algorithms for it as we will need them in proofs later. This problem is called MISSING-STRING by the authors of [VW23]. It turns out that there seems to be deep connections between their work and ours, we discuss it at the end of subsection 4.8.1.

Definition 4.6: OUTSIDE$_{k,n}$

An instance of OUTSIDE$_{k,n}$ where $k < 2^n$ is given by k elements $y_1, \ldots, y_k \in \{0,1\}^n$. A solution to this instance is an element $y \in \{0,1\}^n$ such that $\forall i, y \neq y_i$.

Theorem 4.6: OUTSIDE$_{k,n}$ $\in \mathsf{DTIME}(nk^2)$

OUTSIDE$_{k,n}$ can be solved in time $O(nk^2)$ by a multi-tape Turing Machine

Proof

For each $y \in \{0,1\}^n$ check if $\forall i, y \neq y_i$ if that is the case output y. Checking for one y takes kn time (words have n bits here), since this happens for at most $k+1$ steps this problem is in $\mathsf{DTIME}(nk^2)$

In the following we prove that OUTSIDE$_{k,n}$ is an easier problem than sorting, thus any sorting algorithm on multi-tape Turing machines would yield faster algorithms for OUTSIDE$_{k,n}$. Using fusion sort we may show that sorting k numbers of size n on a multi-tape TM can be done in $O(nk\log(k))$.

Theorem 4.7: Sorting is in $O(nk\log(k))$ ([Rei90])

Sorting a list of size $[y_1, \ldots y_k]$ where $\forall i \leq k, y_i \in \{0,1\}^n$ can be done in time $O(nk\log k)$ by a TM

Using radix sort it might be possible to show that sorting can be performed in $O(n^2k)$ (apparently it's quoted in [Rei90] but it is in German [1])

Lemma 4.1: OUTSIDE is easier than sorting

Let f be the function such that the problem of sorting k numbers of $\{0,1\}^n$ is in $\mathsf{DTIME}(f(k,n))$ then OUTSIDE$_{k,n}$ $\in \mathsf{DTIME}(f(k,n))$

Proof

Let $y_1, \ldots, y_k$ be an instance of OUTSIDE$_{k,n}$, sort the y_i, then check for gaps of more than 1 in the sorted list, output any number within this gap.

Theorem 4.8: OUTSIDE$_{k,n}$ $\in \mathsf{DTIME}(nk\log k)$

OUTSIDE$_{k,n}$ can be solved in time $O(nk\log(k))$ by a multi-tape Turing Machine

Proof

Direct application of theorem 4.7 and lemma 4.1.

In the RAM model everything is much easier.

---

[1] if the reader wonders how we could quote this book for the previous theorem it is because we independently checked theorem 4.7

Theorem 4.9:

In the RAM model $\mathrm{OUTSIDE}_{k,n}$ can be solved in time $O(kn)$.

Proof

Let $y_1, \ldots, y_k$ be an instance of $\mathrm{OUTSIDE}_{k,n}$, create an array l of size k containing only 0 ; for each y_i, if $y_i \leq k$ do $l[y_i] = 1$ (where y_i is interpreted as a number in $[0; 2^n - 1]$). Traverse list l until you find an index j such that $l[i] = 0$, output j.

4.3 Goal and general strategy

High level description of the paper. Our goal is to study the question

$$\exists_\infty n, [f(n), T(n)] \subsetneq [f(n), \alpha T(n)] \quad ?$$

Longpré has shown

$$\exists_\infty n, [f(n), T(n)] \subsetneq [f(n), c2^{2f(n)} T(n)].$$

The proof of Longpré follows a standard diagonalization argument. One considers a simulant $p(n)$ which runs all simulees of size $f(n)$ or less for $T(n)$ steps (this takes time $2^{f+1}T$), and then outputs a string of length n which was not the outee of any of the 2^{f+1} simulees. The simulant $p(n)$ can be made of size $K^{T(n)}(n) + O(1)$ since we just gave its description (the term $O(1)$) and instead of giving n as an input we can give it a representation of n decodable in $T(n)$ steps (the term $K^{T(n)}(n)$). As a consequence the stronger result we obtain is in fact

$$\exists_\infty [f(n), T(n)] \subsetneq [K^{T(n)}(n) + O(1), c2^{2f(n)} T(n)],$$

When f is bigger than $\log(n) + c$, one can change the quantifier $\exists_\infty$ to $\forall_\infty$, since $K^{T(n)}(n) < \log n + c$ for some $c \in \mathbb{N}$.

One try variant. We restate here the idea given in section 4.1 described by figure 4.2. Our idea to prove

$$[f(n), T(n)] \subsetneq [f(n), \alpha T(n)]$$

is to do the following: instead of running every simulee of size $\leq f$ for T steps (which takes time $2^{f+1}T$), we only run α of them for T steps. First we divide the set of 2^{f+1} simulee of size f into chunks of size α, these chunks are called C_i for $i \in [0, 2^{f+1}/\alpha]$. We then consider $2^{f+1}/\alpha$ simulants $(p_i)_{i \in [2^{f+1}/\alpha]}$. Simulant p_i will run all the simulees of chunk C_i for T steps: some of those may stop before T steps and we remember their outputs (there are at most α such outees). Call those outputs $c_i^1, c_i^2 \ldots$. Then each program p_i will look at these outees and produce an outant which hopefully is not the output of *any* program of size f (not only the outees of chunk C_i). Now of course this is where the difficulty of the approach lies: each simulant p_i only knows a fraction of the possible outputs of programs of size f, and there is no reason to think that it can output something new. However, we only need *one* of those programs to output something new. To that effect, the programs will play a collective game ensuring that at least one of them outputs a "new" string. The second task of our programs, after computing the values c_i^j, is thus to apply a *strategy* ensuring that one of them succeeds in outputting a new string.

As a technical note, instead of considering multiple programs p_i we will have one global program [2] p taking as input a description of n decodable in $2n$ steps noted $\#n$ and i, n tells it which size we are currently looking at and i tells it to run the i-th chunk of size α of simulees of size $f(n)$.

Simulant $p(\#n, i)$ is of approximate size $K^{2n}(n) + f - \alpha$, this is smaller than f for infinitely many n for any reasonable α (which is what we need) . We even have some leeway which allows us to cram in some additional information which we'll use in the multiple tries variants.

As for the running time, simulating α programs for T steps takes αT steps in total. We then have to apply our strategy on the α inputs. The strategy needs to be described, which adds some extra size to program p Running this strategy also adds some extra running time. Depending upon our strategy, this extra time could be greater than αT and forbid us from concluding.

Novelty games Regardless of size and time constraint, it is not clear for now that a strategy may even exist. We will first explain again the combinatorial problem and then how it is relevant to our simulants. We start from an initial list $E \subset [1; N]$ of size pk. We then partition this list E into sets of size k and give those to p players. Each of the p players may produce an answer depending only on their k inputs. Their answer must be an element of $[N]$.The players collectively win if at least one of the answers given by the players is not in E. Is there a strategy the players can apply which makes them win no matter the initial list E ? Note that the players may agree on a common strategy, and no communications are allowed between players during the process. This game is noted $\mathcal{G}^N(p, k)$ and such games are called novelty games, we describe them in section 4.4.

Relating novelty games to our parameters in the Kolmogorov setting we get : each of the p players represent a simulant, therefore $p = 2^{f+1}/\alpha$, and each retrieves at most α outee, thus $k = \alpha$. The answer of each simulant is an outant of $\{0,1\}^n$, thus $N = 2^n$. List E should be understood as the set of all outees of size n of simulees of size f or less which we are trying to parallelize against, thus $|E| \leq 2^{f+1}$, let us pick the worst case and just say $E = 2^{f+1}$.

Remark 2. The combinatorial approach disregards any restriction on the number of bits necessary to describe the strategy and the running time of the strategy. As a consequence, once we will have a winning strategy, we will need to take care of these aspects in order to use it for proving our result about time-bounded Kolmogorov complexity classes.

Remark 3. Achieving a winning strategy for *any* list E is more than what is needed for our specific problem. Indeed, for the intended application for time-bounded Kolmogorov complexity, we only need a winning strategy when the initial list is composed of the outputs of the programs of size $\leq f(n)$ running for $T(n)$ steps.

Remark 4. The bound $[N]$ is really important. Our simulant *must* answer a bitstring of size n (recall the definition of $[f, T](n)$) i.e. a integer in $[N]$. We are given $f(n)$ and $\alpha(n)$ which determine the number $p(n)$ of players and the number $k(n)$ of inputs. Now it may be that the game $\mathcal{G}^N(p(n), k(n))$ has a solution when N is very large but not when N is small [3]. But for our purpose we need N to be less than 2^n. This leads us to an analysis in section 4.4 of the bound $\mathfrak{B}(p, k)$ which is the smallest N for which game $\mathcal{G}^N(p, k)$ has a solution. This bound $\mathfrak{B}(p, k)$ proves to be the crux and the limiting factor of our results.

---

[2] This adds some uniformity to the programs which is not formally needed and might hinder the approach. Without uniformity, one may give additional advice to the programs, for instance information about the inputs of the other programs (hence the terminology of parallel diagonalization *with advice*), which is morally equivalent to the multiple tries approach detailed next.

[3] We will establish later in the paper that the game is indeed easier when N gets large.

Multiple tries variant. Up until this point we have considered that simulants $p(\#n, i)$ of size $\approx K^{2n}(n) + f - \alpha$. As stated before, we only need the program to be of size f or less, and since all reasonable α are much bigger than $K^{2n}(n)$, we can provide some extra information (advice) to our programs. This technique can be thought of as parallel diagonalization with advice. We present here *one* way of using this extra information [4].

We consider simulants $p(\#^T n, i, j)$. There are to be thought of as the j-th simulant attributed to the i-th chunk of α simulees. Input j is in $[1; 2^\alpha / g]$, where g is any reasonable slow growing function. The size of $p(\#^T n, i, j)$ is therefore approximately $K^T(n) + f - \alpha + \alpha - \log(g) \approx K^T(n) + f - \log(g)$, which is smaller than f for infinitely many n. The program $p(\#^T n, i, j)$ runs the i-th chunk of α programs of size $\leq f(n)$ for $T(n)$ steps and then outputs an outant which depends on j. You may see figure 4.3

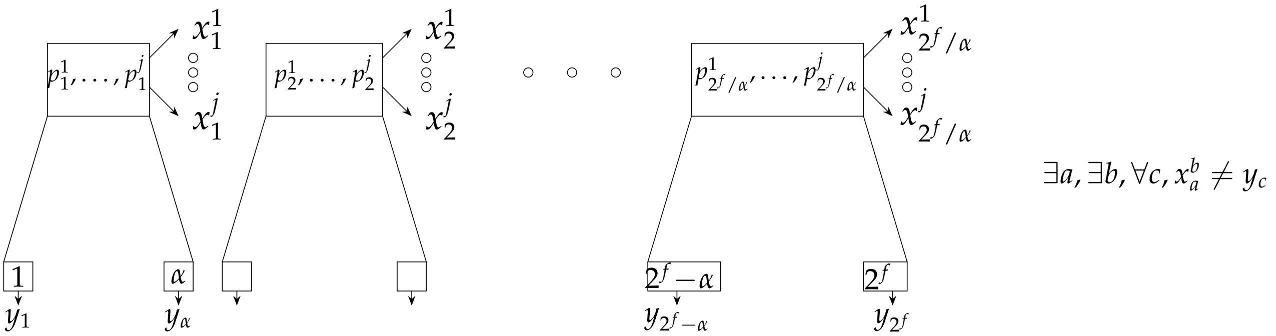


Figure 4.3: Multiple tries setting

Expressed using our combinatorics problem, each player can now propose m answers instead of giving out a unique answer. The players collectively win the game if one of the answers of one of the players is not in the initial list E. This game is denoted by $\mathcal{G}^N(p, k, m)$.

For the moment, we have not been able to use the ability to use multiple tries to improve on our solution for the one try variant, and we only came up with basic theorems. We nevertheless mention it since it is a natural extension of the underlying combinatorics problem. It might also allow us to lower the bound $\mathfrak{B}(p, k, m)$ presented in section 4.4.

4.4 Novelty games

In this section, we introduce the combinatorial game behind our technique, under the name of *Novelty games*.

We remind the reader that $\binom{[N]}{k}$ denotes the set of all multisets included in $[1, N]$ of size at most k.

---

[4] this may not be the only meaningful way to use this extra information, for instance we could consider extra simulants working with a different ordering of simulees. This way naturally extends the previously associated combinatorics game.

4.4.1 Definition

Definition 4.7: $\mathcal{G}^N(p,k,m)$

The game $\mathcal{G}^N(p,k,m)$ is defined as follows.

- There are p players.
- Player i receives a multiset $A_i \in \binom{[N]}{k}$; elements of A_i are called inputs.
- Each player has m tries.
- On each try a player answers an element of $[1;N]$

An *occurrence* of the game $\mathcal{G}^N(p,k,m)$ is a specific family $(A_i)_{i=1}^p$.
A *strategy* $\mathfrak{S}^N(p,k,m)$ for $\mathcal{G}^N(p,k,m)$ is a family of functions $(s_i)_{i=1,\dots,p}$ where

$$s_i : \binom{[N]}{k} \times [m] \to [N].$$

The players collectively win the occurrence $(A_i)_{i=1}^p$ of $\mathcal{G}^N(p,k,m)$ if there exists a pair $(i,j) \in [p] \times [m]$ such that $s_i(A_i,j) \notin \cup_{i=1}^p A_i$.
A *winning strategy* for $\mathcal{G}^N(p,k,m)$ is a strategy such that all occurrences of the game are won.

When $m=1$, we may write $\mathcal{G}^N(p,k)$ instead of $\mathcal{G}^N(p,k,1)$.

Definition 4.8: Oblivious strategy

When all players have the same strategy, i.e. when $\mathfrak{S} = (s_i)_{i=1}^p$ is such that there exists a function s with $s_i = s$ for all $i = 1,\dots,p$, we say that $\mathfrak{S}^M(p,k,m)$ is *oblivious*.

Definition 4.9: $\mathfrak{B}(p,k,m)$

We define $\mathfrak{B}(p,k,m)$ as the smallest integer N such that there exists a winning strategy for game $\mathcal{G}^N(p,k,m)$.

We note that $\mathfrak{B}(p,k,m)$ is always less than $+\infty$, as proven in section 4.5.2. Establishing an estimate of $\mathfrak{B}(p,k,m)$ in general is open.

4.4.2 General remarks about Novelty games

Sets. The first remark is that one may only consider sets of size exactly k instead of multisets of size at most k. This is because the hardest case for the strategy is when each player has k different numbers. Indeed, suppose there exists a strategy $\mathfrak{S}$ winning in all occurrences of the game for which players have a set of size k (or, equivalently, a multiset of size k without repetitions). Then one can easily construct a winning strategy for all multisets: when the considered multiset have less than k distinct numbers, you may always add arbitrary numbers to have k of them. One can apply the strategy $\mathfrak{S}$ on the resulting set. A routine check suffices to show that this strategy is winning.

In the following, we will therefore always assume that inputs are pairwise distinct, and that we may add arbitrary new inputs so that sets have size exactly k.

We also note that inputs are not ordered, since we work with sets and not tuples. But we may suppose, if need be, that they are by choosing an arbitrary order relation.

The next theorem states that the existence of strategy for inputs in $[1;N]$ implies the existence of a strategy for inputs in $[1;M]$ for all $M \geqslant N$. This motivates the definition of $\mathfrak{B}$.

Theorem 4.10:

Let $M, N \in \mathbb{N}$ such that $M \geqslant N$. If $\mathcal{G}^N(p,k,m)$ has a winning strategy then so does $\mathcal{G}^M(p,k,m)$.

Proof

We treat the case $M = N+1$ and then we can conclude by induction. Let $\mathfrak{S}$ be a winning strategy for $\mathcal{G}^N(p,k,m)$. We define a strategy $\mathfrak{S}'$ for the game $\mathcal{G}^M(p,k,m)$ as follows. If M is not an input of strategy $\mathfrak{S}'$, then each player simply runs $\mathfrak{S}$. Otherwise, the player replaces M by N and runs the strategy $\mathfrak{S}$ on the resulting set. One easily checks that $\mathfrak{S}'$ is a winning strategy for $\mathcal{G}^M(p,k,m)$ since the output of $\mathfrak{S}'$ is in $[N] \subset [M]$.

Next theorem proves the intuitive result that having more players, more inputs and less tries makes the game harder.

Theorem 4.11:

For all $p, k, m \in \mathbb{N}$, and all $a, b, c \in \{0,1\}^3$,

$$\mathfrak{B}(p-a, k-b, m+c) \leq \mathfrak{B}(p,k,m).$$

Proof

This is proven by noticing that a strategy for the game $\mathcal{G}^M(p,k,m)$ is also a strategy for $\mathcal{G}^M(p-a,k-b,m+c)$: it suffices to add b random inputs, apply the strategy on those (since there are less players, the strategy determines a map for each player – and the maps s_i for $i = p-a+1, \dots, p$ are unused), and guess c arbitrary additional outputs.

We believe establishing recurrent relations between values of $\mathfrak{B}$ is hard when varying the number of players, numbers, tries or the size of the set of inputs. For instance, it is not even clear how the inequality of theorem 4.11 can be made strict.

We now start by proving a simple lower bound on $\mathfrak{B}$.

Theorem 4.12: Trivial lower bound on $\mathfrak{B}$

For all $p, k, m \in \mathbb{N}$, $\mathfrak{B}(p,k,m) \geq pk+1$.

Proof

It is easy to realise that there are no winning strategy if $N = pk$: if for all $i \in [1;p]$, player i has as inputs $[(i-1)k+1; ik]$, then the set of all inputs is equal to $[1;pk]$ and no player may output a new number.

Another easy result to establish is that if one allows for enough tries, the value of $\mathfrak{B}$ can be determined (and is low).

Theorem 4.13:

For all $p, k \in \mathbb{N}$, $\mathfrak{B}(p, k, k) = pk + 1$.

Proof

Let $N = pk + 1$. Each player calls its inputs $i_1, \ldots, i_k$. On its j-th try each player does this: if $i_j + 1 \leq N$ it answers $i_j + 1$ otherwise it outputs 1. It's routine to check that this is a winning strategy.

The study of multiple tries has proven to be hard. Apart from quite obvious results, we have not found ways to improve on the results we obtain in the one-try variant. We therefore leave the following question for future work.

Question

Can we find a better bound for $\mathfrak{B}(p, k, k-1)$ than $\mathfrak{B}(p, k, 1)$?

We summarise the results of this paper in fig. 4.4. The proof are scattered in the paper.

Game	(p,k)	(p,k,i)	(p,k,k)
Upper bound	$(k^p)^{k^p}$	$(k^p)^{k^p}$	$pk+1$
Reference	section 4.5.2	section 4.5.2	section 4.4.2

(a) Bounds for general values p, k

Game	$(2,2)$	$(1,k)$	$(p,1)$	$(2,k)$
Upper bound	9	$k+1$	$p+1$	$k^3 2^k$
Reference	Theorem 4.14	section 4.4.2	section 4.4.2	section 4.4.5

(b) Bounds for specific values

Figure 4.4: Bound on $\mathfrak{B}$ (when $m = 1$ it is omitted)

4.4.3 Relation between novelty games and time hierarchy for K^t

In this section we establish links between $\mathfrak{B}$ and the best result we may get in order to show that $\exists_\infty n, [f, \alpha T](n) \nsubseteq [f, T](n)$. We ignore for now implementations of specific strategies (in particular running time and size). We remind the reader that the goal is to have $\alpha = o(2^f)$, and hopefully showing the conjecture for all $f < n$. All of this leads us to studying game $\mathcal{G}^N(p, k, m)$ where:

- $p = 2^f/\alpha$;
- $k = \alpha$;
- $m = k/g$ where g is any reasonable increasing function (thought of as slow growing);
- $N = 2^n$.

Our bound $\mathfrak{B}(p,k,m)$ should be less than N because a program must answer a string of size n (i.e. an integer of $[N]$). We are thus interested when α, f (or alternatively (p,k,m) with the above equalities) are such that :

$$\mathfrak{B}(p,k,m) < N \Leftrightarrow \exists g \text{ reasonable and increasing}, \exists_\infty n, \mathfrak{B}\left(\frac{2^{f(n)}}{\alpha(n)}, \alpha(n), \frac{\alpha(n)}{g(n)}\right) \leq 2^n.$$

Restricting our views to the one try setting, we make some hypothesis on the form of $\mathfrak{B}(p,k)$ from least to most likely, and see what is the best α and f we may get (we disregard all potential problems with running time and size of the winning strategy).

1. **Hypothesis** $\mathfrak{B}(p,k) \leq (pk)^c$ for $c \in \mathbb{R}$. We may establish the theorem for $f < \frac{n}{c}$ and any reasonable α.
2. **Hypothesis** $\mathfrak{B}(p,k) \leq p^k$. For any α reasonable we may establish the theorem for $f < \frac{n}{\alpha}$.
3. **Hypothesis** $\mathfrak{B}(p,k) \leq k^p$. We may establish the theorem when $2^f < \alpha n$. For $f = \omega(\log(n))$ this may only lead to a slight improvement when compared with Longpré's result.

4.4.4 Resolution of $\mathcal{G}(2,2)$ case

This partial solution was chronologically the first to be proposed, by Corentin Henriet during informal exchanges. We chose to present it since the strategy is different from the ones presented in the next sections and it provides better bounds than in the general case.

This strategy is oblivious. We will thus define the single map s used by both players. The strategy is winning as long as $N \geq 9$, showing that $\mathfrak{B}(2,2,1) \leq 9$ (much better than the general bound obtained in section 4.6, which gives 216). In this specific case, we can also prove that $\mathfrak{B}(2,2,1) > 5$ (to be compared to 4 – the general lower bound obtained in theorem 4.12).

Definition 4.10: Strategy s

We intuitively understand our inputs as being on a "clock" from 1 to N, i.e. the successor of N is 1. Let (x,y) be our inputs ordered such that the distance between x and y is less than the distance from y to x when read in the clockwise direction. If $y = x+1$ or $y = x+2$, then $s(x,y) = x-2$, otherwise we define $s(x,y) = x+1$.

Theorem 4.14:

$\mathfrak{B}(2,2,1) \leq 9$.

Proof

The proof consists in verifying that s is a winning strategy. This is done by checking every possible scenario.

Theorem 4.15:

$\mathfrak{B}(2,2,1) > 5$.

Proof

This is easily checked by pen and paper search.

While this strategy provides good bounds, we have not found a way to generalise it to more players or larger sets. The next section presents a general strategy for the 2-players case.

4.4.5 Graph strategies

This line of reasoning, and the corresponding winning strategy for $\mathcal{G}(2,k,1)$ was first obtained by Dmitriy Kunisky during informal exchanges.

We first define of colored graphs. These colored graphs have their *edges* colored with no restriction.

Definition 4.11: Colored graphs

Let $p \in \mathbb{N}$. A p-colored graph is a triple (V,E,C) where V is a set of vertices, $E \subseteq \{(x,y) \in V^2 \mid x \neq y\}$ is a set of edges, and $C : E \to [p]$ a coloring function mapping every edge to a color (an element of $[1;p]$).
The graph is said to be complete when $E = \{(x,y) \in V^2 \mid x \neq y\}$. We will moreover write $G_i = (V,E_i)$ the graph induced by selecting the edges of a given color i, i.e. $w \in E_i \Leftrightarrow w \in E \wedge C(w) = i$. Given $u,v \in V$, we write $u \sim v$ when $\{u;v\} \in E$; in this case we say that u and v are neighbors. We moreover write $u \overset{i}{\sim} v$ when $u \sim v$ and $C(\{u;v\}) = i$.

We will now define a property of colored graphs which will be used later to define a winning strategy.

Definition 4.12: (p,k)-set property

Let $p,k \in \mathbb{N}$. A p-colored graph $G = (V,E,C)$ is said to have the (p,k)-set property if for every subset of vertex $A \subseteq V$ of size k and every color $i \in [1;p]$, the vertices in A have a mutual neighbor v in G_i, i.e. such that for all $a \in A$, $a \overset{i}{\sim} v$.

We will now explain how the (p,k)-set property ensures that one can define a strategy for p players and sets of size k from a p-colored graph. The intuition is the following: each input will correspond to a vertex, each player will correspond to a color, and a player will use the (p,k)-set property instantiated on its color to produce an answer.

We now recall the notion of polychromatic cycle that will be useful to discuss winning strategies.

Definition 4.13: Polychromatic cycles

Let $p \in \mathbb{N}$ and $G = (V,E,C)$ be a p-colored graph. A cycle c of G is said to be polychromatic if no two edges of c have the same color.

Note that a polychromatic cycle does not necessarily use all colors. The non-existence of polychromatic cycles will ensure that the strategy will be winning (i.e that no all player's answer are an input of another one). For the interested reader, graphs with no polychromatic cycles have been studied here [Bat83; ALW11].

Definition 4.14: Good graphs

Let $p, k \in \mathbb{N}$, and let G be a p-colored graph. We say that G is (p,k)-good if it has the (p,k)-set property and has no polychromatic cycles.

We now formally describe the strategy induced by a (p,k)-good graph $G = (V, E, C)$ with $V = [1; N]$. The map $s_i^G : \binom{[N]}{k} \mapsto [N]$ defining the strategy of the i-th player is defined as $s_i^G(A) = y$ where y is any vertex (for instance, the smallest) such that $\forall x \in A, x \overset{i}{\sim} y$. Such a y always exists thanks to the (p,k)-set property. We can then establish that the absence of polychromatic cycles implies that this strategy is winning.

Theorem 4.16: Good graphs make winning strategies

Let $p, k \in \mathbb{N}$ and let G be a p-colored graph with N vertices. If G is (p,k)-good, the strategy $\mathfrak{S} = (s_i^G(A))_{i\in[1;p]}$ is winning for the game $\mathcal{G}^N(p,k,1)$.

Proof

Suppose the strategy is not winning. Then there exists an occurrence $A_1, \ldots, A_p$ of the game for which the strategy fails (here A_i denotes the set of inputs of player i). Consider a graph H where the vertices are players P_1 to P_p, with a directed edge from P_i to P_j if the answer of player i, i.e. $s_i^G(A_i)$, belongs to the set of inputs A_j of player j. By assumption, there exists a cycle in this graph (otherwise the strategy is winning on this occurrence). But one can check that a cycle in H implies the existence of a polychromatic cycle in G, leading to a contradiction.

The last step is now to show that $(2,k)$-good graphs exist. We will then discuss, in subsection 4.4.5, the unlikely existence of (p,k)-good graphs for $p > 2$ and $k \geq 2$.

Graph strategies for $(2,k)$

Theorem 4.17: Existence of $(2,k)$-good graphs

For any $k \in \mathbb{N}$, there exists a $(2,k)$-good graph.

Proof

We will prove that there exists an undirected uncolored graph $G = ([N], E)$ such that for every $A \in \binom{[N]}{k}$:

- $\exists a, \forall x \in A, x \sim a$;
- $\exists b, \forall x \in A, x \nsim b$.

This is sufficient to prove the existence of a $(2,k)$-good graph by converting every edge to a 1-colored edge and adding a 2-colored edge for every pair of vertices $x \neq y$ such that $x \nsim y$. Note that this graph has no polychromatic cycles of size 2 since two vertices cannot be simultaneously neighbors and non-neighbors.
We now prove the existence of such graphs using the probabilistic method. Let G be

an Erdős-Rényi graph with edge probability $\frac{1}{2}$ on the vertex set $[1;N]$. We have that :

$$\begin{aligned}
&\mathbb{P}[G \text{ is not } k\text{-good}] \\
&\quad \leq \mathbb{P}[\text{some } i_1, \ldots, i_k \in [N] \text{ have no mutual neighbor}] \\
&\qquad + \mathbb{P}[\text{some } i_1, \ldots, i_k \in [N] \text{ have no mutual non-neighbor}] \\
&\quad \boxed{\text{by symmetry}} \\
&\quad \leq 2 \cdot \mathbb{P}[\text{some } i_1, \ldots, i_k \in [N] \text{ have no mutual neighbor}] \\
&\quad \leq 2\binom{N}{k}\mathbb{P}[1, \ldots, k \text{ have no mutual neighbor}] \\
&\quad \leq 2\binom{N}{k}\left(1 - \frac{1}{2k}\right)^{N-k} \\
&\quad \leq 2\exp\left(k \log N - \frac{N-k}{2k}\right) \xrightarrow{N \to \infty} 0
\end{aligned}$$

Thus for large enough N there thus exist $(2,k)$-good graphs with N vertex

Looking more closely at the proof we notice that $2\exp\left(k \log N - \frac{N-k}{2k}\right)$ is less than 1 when $N = O(k^3 2^k)$, thus giving us $\mathfrak{B}(2,k) < O(k^3 2^k)$. The bound of section 4.5 gives a worse bound of $O(2^{2k^2 \log k})$.

Size and running time of the graph strategy. If we want to use a strategy based on graphs in the programs of section 4.6 to show $[f, \alpha T] \subsetneq [f, T]$, we need a succinct description of such graphs. Indeed, the program needs to be of size less than f. Note however that we have only demonstrated the existence of a $(2,k)$-good graph thanks to the probabilistic method. The best we can naively do is then to hardcode those graphs in the strategy. This would make the strategy really large in terms of bits (close to $N^2 = 2^{2n}$), but our programs can never be bigger than $f < n$ bits. Another approach is to let the programs find those graphs by itself: every program $p(\#^T, i)$ enumerates and tests graphs in ascending lexicographic order until it finds a good graph, and then use it for its strategy. While this strategy takes $O(1)$ bits to describe, it takes *at least* 2^n steps to run implying that one could only establish bounds for $T > 2^n$.

We also hint (but not formally prove) at a method of constructing $(2,k)$-good graphs without using probability theory. We briefly describe such a potential graph using the notion of pseudorandom *Paley graph* from number theory. A Paley graph is a graph G on a prime number $p \equiv 1 \mod 4$ of vertices. Vertices of G are identified with integers modulo p, and there exists an edge $i \sim j$ if and only if $i - j$ is congruent to a square modulo p, i.e. if and only if $\exists k, i - j = k^2 \mod p$. The condition $p \equiv 1 \mod 4$ ensures that the relation $\sim$ is symmetric.

Finding a mutual neighbor of $x_1, \ldots, x_k$ then amounts to solving the system of equations $\{y - x_i = z_i^2\}$ in variables $y, z_1, \ldots, z_k$. One may check, using character sum estimates, that many integers y satisfying these equations exist, for any choice of $z_1, \ldots, z_k$, as long as p is sufficiently large.

Graph Strategies for (p, k)

We now turn our attention to (p, k)-good graphs. Unfortunately, we have not been able to prove or disprove the existence of such graphs for $p > 2$. However, we can establish that their are no complete (p, k)-good graphs (note that the $(2, k)$-good graphs we came up with are complete).

Definition 4.15: Complete graph

A p-colored graph G is said to be complete if there is a (colored) edge between any two vertices.

Definition 4.16: Connectivity for color i

A p-colored graph G is said to be connected for color i if there is a i-colored path between any two vertices in G

Theorem 4.18: ([Bat83])

Let G be a complete p-colored graph. If G has no polychromatic cycles, then G is connected for at most 2 colors.

Theorem 4.19:

There are no complete (p, k)-good graphs for $p > 2$ and $k \geq 2$.

Proof

Let $G = (V, E, C)$ be a complete (p, k)-good graph. Then for any color i and any two vertices $u, v \in V$, there exists $w \in V$ such that $u \overset{i}{\sim} w$ and $v \overset{i}{\sim} w$. This implies that u and v are connected for every color. Whenever there are $p \geq 3$ colors, this contradicts Theorem 4.18

Can graph strategies be salvaged? Graph strategies provide, in the case $p = 2$, better bounds than the general strategy defined below. However it is unclear if (p, k)-good graphs exist.

Another graph based approach would be to consider uncolored but *directed* graphs with the following properties:

- there exists no cycles of length less than p;
- for any set $i_1, \ldots i_k$ there exists a vertex b such that $\forall j \in [1; k], i_j \rightarrow b$.

Such graphs can be used straightforwardly to define a winning strategy for the game $\mathcal{G}(p, k, 1)$:every player answers a common directed neighbor of its $k-$inputs using the second property. The fact that the strategy is winning follows from the first property. However, the existence of such graphs also remains an open question at the moment.

Question

Do (p, k)-good graphs exists ?

Question

Let $p, k > 3$, does there finite directed graphs $G = (V, E)$ such that no directed cycles of length less than p exist and for any set $i_1, \dots i_k \in V$ there exists a vertex b such that $\forall j \in [1; k], (i_j, b) \in E$?

4.5 General Strategy for (p, k)

We now exhibit an oblivious strategy for any number of players and inputs. We start by detailing the $(3, 2)$ case which is naturally generalized to any (p, k).

4.5.1 Strategy for (3,2)

We expose a strategy for game $\mathcal{G}^N(3, 2)$ when $N = 15^7$, the reason for such an N will be made apparent later. All player will execute function $F_{(3,2)} : N^2 \mapsto N$ defined in 4.18. We will see inputs as number in base $m = 15$.

Definition 4.17: Writing in base m

Let $x \in N$, $m \in \mathbb{N}$, $x = \overline{x_k ... x_2 x_1}^m$, where the $x_i \in [0; m-1]$ are uniquely defined to verify $x = \sum_{i=0}^{k-1} x_{i+1}.m^i$.
We may drop the $\overline{\quad\quad}^m$ if we specify the base beforehand.

Definition 4.18: $F_{(3,2)}$

The function $F_{(3,2)}$ is defined on $[0, 15^7]$ as follows:
Let $x, y \in N$, let $x_7 x_6 x_5 x_4 x_3 x_2 x_1$ and $y_7 y_6 y_5 y_4 y_3 y_2 y_1$ denote their notation in base 15.

$$F_{(3,2)}(x, y) \overset{\text{def}}{=} \overline{x_3 x_2 y_3 y_2 x_1 y_1 \nu}^{15} \tag{4.1}$$

$$= x_3.15^6 + x_2.15^5 + y_3.15^4 + y_2.15^3 + x_1.15^2 + y_1.15 + \nu \tag{4.2}$$

where $\nu \in \{0, \ldots, 14\}$ is the smallest number such that $\forall i \in [1; 7], \nu \neq x_i$ and $\nu \neq y_i$

Note that by definition $F_{(3,2)}(x, y)$ is necessarily different from x and y. And ν in the definition exists because $m = 15$. ν could have been taken to be any number different from all x_i and y_i, not necessarily the minimum.

Theorem 4.20:

The function $F_{(3,2)}$ provides an oblivious winning strategy for 3 players with 2 inputs.

Proof

We will denote by a, b the numbers given to player 1, by c, d the numbers given to player 2, and by e, f the numbers given to player 3. The goal is to show that if each player applies the function $F_{(3,2)}$ then they always collectively win the game.
As in the definition above, we write $a = a_7 a_6 a_5 a_4 a_3 a_2 a_1$ the base 15 representation of a, $b = b_7 b_6 b_5 b_4 b_3 b_2 b_1$ the base 15 representation of b, etc.
If $F_{(3,2)}(a, b)$ is different from c, d, e, f, the players have won the game. Suppose now that we are in the case where $F_{(3,2)}(a, b) \in \{c, d, e, f\}$. Without loss of generality, we can suppose that $F_{(3,2)}(a, b) = c$. We have in particular that: $c_0 \neq a_i$ for all $i \in [1, 7]$, $c_0 \neq b_i$ for all $i \in [0, 6]$ $c_3 = a_1$, $c_6 = a_2$, and $c_7 = a_3$.
We now consider the result produced by player 2, that is $F_{(3,2)}(c, d)$. Once again, if this value is different from a, b, e, f then the game is won. If this is not the case, there are two cases.

The first case is $F_{(3,2)}(c,d) \in \{a,b\}$. We now show that this is impossible. By definition, $F_{(3,2)}(c,d)$ is different modulo 15 from all c_i ($i \in [1,7]$), but c_2 is the remainder modulo 15 of b and and c_3 is the remainder modulo 15 of a. Hence the value of $F_{(3,2)}(c,d)$ modulo 15 is different from the values of a, b modulo 15.
The second case is $F_{(3,2)}(c,d) \in \{e,f\}$. Without loss of generality, we suppose $F_{(3,2)}(c,d) = e$. We will now have to show that $F_{(3,2)}(e,f) \notin \{a,b,c,d\}$. Let us write $z = z_7z_6z_5z_4z_3z_2z_1$ the representation of $F_{(3,2)}(e,f)$ in base 15. By definition of $F_{(3,2)}$, $z_0 \neq e_i$ for all $i \in [1,7]$. But since $F_{(3,2)}(c,d) = e$ and $F_{(3,2)}(a,b) = c$, we have the following properties: $e_2 = d_1$, $e_3 = c_1$, $e_6 = c_2 = b_1$, $e_7 = c_3 = a_1$. Thus z_1 is different from a_1, b_1, c_1, and d_1. This proves that $F_{(3,2)}(e,f) \neq a,b,c,d$, and therefore the game is won.

The function behind the strategy can be explained informally as follows. We call antecedent of x any a such that $\exists b, F(a,b) = x$. Given two arguments x, y, we make it so that their digits when written in base m correspond to the remainders modulo m of their antecedents through F. Here m will be chosen to be equal to 15, as explained below. Remind that we need no cycles of size $\leqslant 3$ to appear, i.e. player 1 answering an input of player 2, player 2 answering an input of player 3 and player 3 answering an input of player 1. Thus we keep track of the values of such remainder up to depth-2 antecedents (i.e. antecedents of antecedents). For instance, x will be written as $x = x_7x_6x_5x_4x_3x_2x_1$ in base m, and we understand:

- x_2, x_3 as the remainders of the antecedents of x,
- x_4, x_5 as the remainders of the antecedents of x_2,
- x_6, x_7 as the remainders of the antecedents of x_3.

Similarly, we suppose the writing of $y = y_7y_6y_5y_4y_3y_2y_1$ to keep track of the remainders modulo m of the antecedents of y.

The function $F_{(3,2)}$ then just implements this notion of keeping track of the antecedents, while producing a fresh value modulo m. More precisely, given $x = x_7x_6x_5x_4x_3x_2x_1$ and $y = y_7y_6y_5y_4y_3y_2y_1$, the function F will pick a value $v \notin \{x_1, \ldots, x_7, y_1, \ldots, y_7\}$. Since there are 14 values in the latter set, choosing $m = 15$ ensures that this is always possible. The function is then defined as

$$F_{(3,2)}(x_7x_6x_5x_4x_3x_2x_1, y_7y_6y_5y_4y_3y_2y_1) = x_3x_2y_3y_2x_1y_1v.$$

The following illustration should clarify the definition of $F_{(3,2)}$.

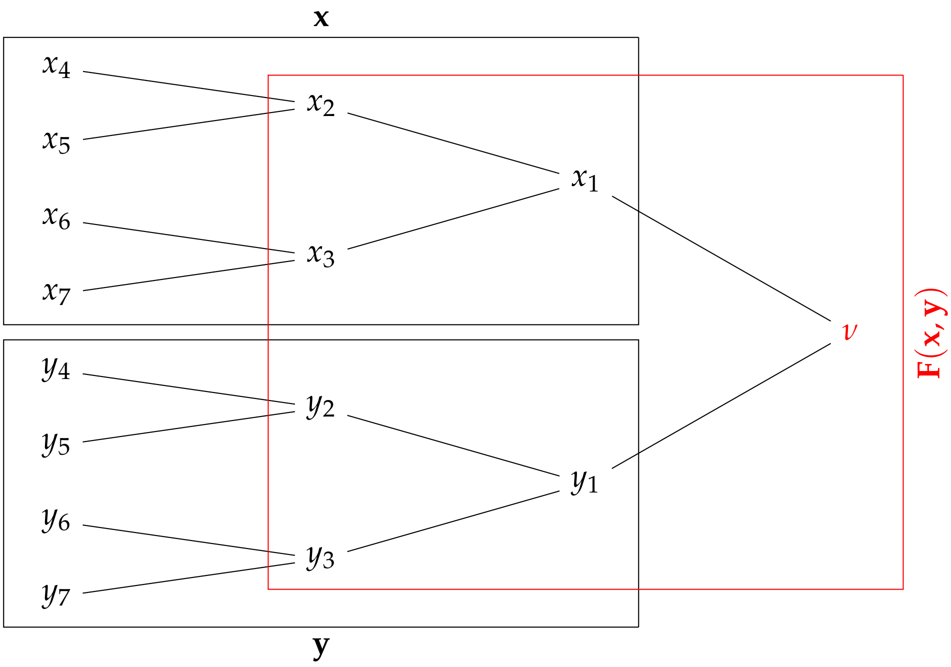


Figure 4.5: Construction tree for $F_{(3,2)}$

4.5.2 General strategy for (p,k)

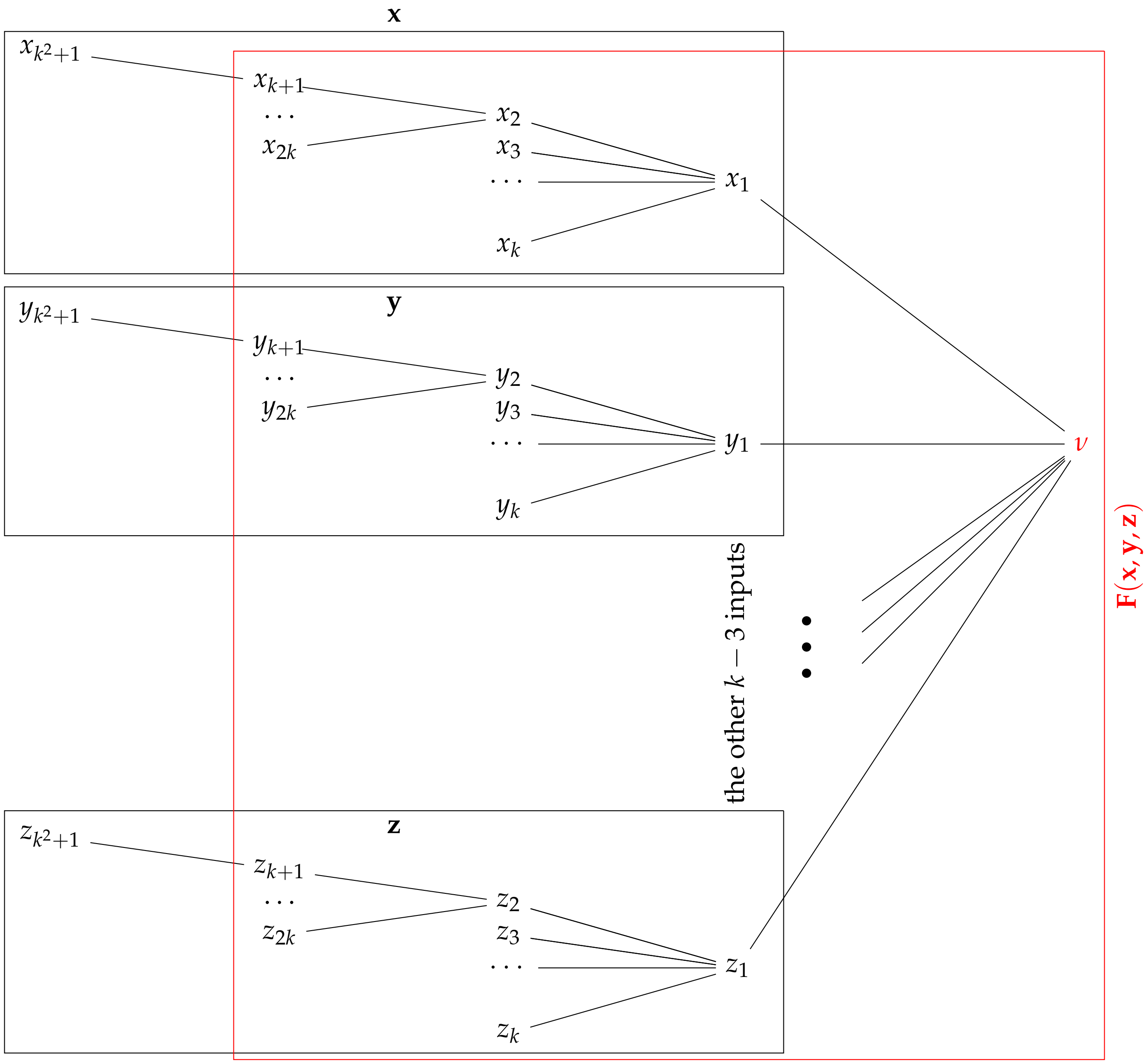


Figure 4.6: Construction for $F_{(3,k)}$

We now expose the general strategy. The construction follows the $(3,2)$ case of figure 4.5. The reader may look at figure 4.6 to see further generalization. Coming up with a formal strategy from figure 4.6 is then just a matter of counting how many digits we need to keep track of antecedent. This then leads to a base m big enough so that ν can be different from all those digits. Once we have these two ingredients we know N the smallest integer such that our inputs live in $[1;N]$ and our strategy works.

Let us explain the construction in plain English: the main insight of the strategy was to book-keep the remainders modulo m of the potential antecedents up to depth 2. For general (p,k), we will therefore keep track of the remainders of potential antecedents up to depth $p-1$. This means that for each of the k arguments, we will need values modulo m of the potential $1+k+k^2+k^3+\cdots+k^{p-1}$ antecedents. Since there are k arguments, this means that we will be manipulating $k(1+k+k^2+k^3+\cdots+k^{p-1})$ values modulo m, and produce a fresh value. This implies that m should be greater or equal to $1+k(1+k+k^2+k^3+\cdots+$

$k^{p-1}) = \frac{k^{p+1}-1}{k-1}$.

Now, since we need to book-keep $1 + k + k^2 + k^3 + \cdots + k^{p-1} = \frac{k^p-1}{k-1}$ antecedents, we need to work with numbers with $\frac{k^p-1}{k-1}$ digits and thus:

$$N \geq m^{\frac{k^p-1}{k-1}} \geq \left(\frac{k^{p+1}-1}{k-1}\right)^{\frac{k^p-1}{k-1}}.$$

We will here suppose that N is minimal, i.e. the above inequality is in fact an equality. We introduce the notation $k^{(d)} = \frac{k^d-1}{k-1}$; by convention, we define $k^{(0)} = 0$. Suppose given arguments $x_i \in [0, N]$. We write those in base $m = \frac{k^{p+1}-1}{k-1}$ as follows:

$$x_i = \sum_{d=0}^{p-1} \sum_{c=0}^{k^d-1} x_i(d,c) m^{k^{(d)}+c}.$$

As a quick sanity check this does have $k^{(p)}$ digits as $\sum_{d=0}^{p-1} \sum_{c=0}^{k^d-1} 1 = k^{(p)}$, and it's routine to check that $m^{k^{(d)}+c}$ does iterate through every m^i for $i \in [\![0; k^{(p)} - 1]\!]$. As a matter of a fact if you look at figure 4.6 and generalize it for (p,k), then you get that $x_i(d,c)m^{k^{(d)}+c}$ corresponds to the c-th digit at depth d of the tree.

The definition of $F_{(p,k)}$ then simply generalises the definition of $F(3,2)$ given above.

Definition 4.19: $F_{(p,k)}$

For any p, k we define the function $F_{(p,k)}$ as follows:

$$F_{(p,k)}(x_0, \ldots, x_{k-1}) = \nu + \sum_{d=1}^{p-1} \sum_{i=0}^{k-1} \sum_{c=0}^{k^{d-1}-1} x_i(d-1,c) m^{i.k^{(d-1)}+c},$$

where ν is a value different from all $x_i(d,c)$ (say the smallest one for a deterministic function).

Let us explain the formula: to construct $F_{(p,k)}(x_0, \ldots, x_{k-1})$ one simply reads the nodes of figure 4.6 in a Breadth First Search (BFS) way and interprets the labels as digits in base m.

Following the proof of Theorem 4.20, one can establish that the strategy where every player uses $F_{(p,k)}$ is winning, leading to the following result.

Theorem 4.21:

$\mathfrak{B}(p,k,1) \leq \left(\frac{k^{p+1}-1}{k-1}\right)^{\frac{k^p-1}{k-1}}$.

We will instead use this bound which is easier to manipulate.

Corollary 4.1:

For large values of p and k, $\mathfrak{B}(p,k,1) \leq (k^p)^{k^p}$.

4.6 Formal description and analysis of the programs

Here we described and analyze formally the programs used to show $[f, \alpha T] \nsubseteq [f, T]$. The functions $f(n)$, $\alpha(n)$ and $T(n)$ are all assumed to be VTC and reasonable as per definition 4.4 and 4.5.

In the following, we will often drop the arguments of the functions and write, for instance, f instead of $f(n)$. We also suppose w.l.o.g. that $T(n) > n$ and $f < n$. Indeed, if $T(n) < n$, then it is not possible to write any string of length n in the allocated time, and if $f \geq n$ then all inputs have a description of size bounded by f and there are no gap in the hierarchy. We also assume for simplicity that 2^{f+1} is divisible by α.

For all $p, k, m, N \in \mathbb{N}$, we have a winning strategy $\mathfrak{S}^N(p, k, m) = (\sigma_i^N(p, k, m))_{i \in [p]}$ for the game $\mathcal{G}^N(p, k, m)$, where σ_i^N is the map defining the strategy for player i. The following definition of programs does not depend on the specific strategy considered, but it should be clear that their analysis will be dependent on this choice.

4.6.1 One try variant

We formally define the program which is intended to output a string in $[f, \alpha T](n)$ and not in $[f, T](n)$ with strategy $\mathfrak{S}^N(p, k, 1)$. We then proceed with the size and running time analyses.

Algorithm 1 Program $p(< \#n, i >)$

```
Input: $< \#n, i >$ a string from $\{0,1\}^*$ (encoding $<>$ from definition 1.6) where:
    $\#n \in \{0,1\}^*$ – the smallest program outputting $n$ in $2n$ steps or less
    $i \in \{0,1\}^{f+1-\log(\alpha)}$ – the chunk number, ranging from 1 to $\frac{2^{f+1}}{\alpha}$
Procedure:
Run $\#n$ as a program and call $n$ its output
Compute $f(n)$, $T(n)$, and $\alpha(n)$
Create an empty list $l$ to store outees
for $j = 0$ to $\alpha - 1$ do
    Simulate program number $(i-1)\frac{2^{f+1}}{\alpha} + j$ for $T(n)$ simulation steps
    if it output a value of size $n$ then
        Add the output to list $l$
    end if
end for
Output: Run the strategy $\sigma_i^N\left(\frac{2^{f(n)}}{\alpha(n)}, \alpha(n), 1\right)$ on $l$ (seen as a multiset) where $N = 2^n$, and
    output the answer.
```

Size analysis. We now analyze the size of $p(\#n, i)$:

- $\#n$ is of size $K^{2n}(n)$ (by definition);
- i is an integer in $[1; 2^{f+1}/\alpha]$, hence it is of size $f + 1 - \log(\alpha)$.
- $< \#n, i >$ is of size $K^{2n}(n) + f - \log(\alpha) + O(\log(K^{2n}(n)))$ from definition 1.6;
- the program p is of size $O(1)$.

Thus for large n, $p(< \#n, i >)$ is of size less than $2K^{2n}(n) + f - \log(\alpha) + O(1)$. Since α is reasonable, this is strictly less than $f(n)$ for infinitely many values of n.

Running time analysis. Steps 3 to 11 take $O(\alpha T)$ steps to compute. The running time of step 12 depends on the specific strategy $\mathfrak{S}^N(2^f/\alpha, \alpha)$ we choose to implement. Hopefully this running time can be kept in $O(\alpha T)$, we assume it for now.

Correctness analysis. Suppose that the strategy $\mathfrak{S}^{2^n}(\frac{2^{f(n)}}{\alpha(n)}, \alpha(n))$ used in the program is a winning strategy for the game $\mathcal{G}^{2^n}(\frac{2^{f(n)}}{\alpha(n)}, \alpha(n))$. Let $\#n$ be a program outputting n. Then, by construction, there exists $i \in [1, 2^f/\alpha]$ such that $\mathbf{Out}(p(< \#n, i >))$ is a string of size n which is not the output of any program of size f in less than T steps.

Conclusion Size and running time analysis are meant to prove that $\mathbf{Out}(p(< \#n, i >))$ is in $[f, \alpha T](n)$. Correctness analysis shows that for at least one i, $\mathbf{Out}(p(< \#n, i >))$ is not in $[f, T](n)$. Thus showing $[f, T](n) \subsetneq [f, \alpha T](n)$.

4.6.2 Multiple tries variant

The multiple tries setting is quite similar to the one try setting, except that we will have an additional input j indicating which try is computed. We formally define the program which is intended to output a string in $[f, \alpha T](n)$ and not in $[f, T](n)$ when the strategy $\mathfrak{S}^N(p, k, m)$ is winning. In the following g is a function such that g and $\log g$ are VTC and reasonable, g is thought of as slowly growing (take for instance $g = \log^*$)

Algorithm 2 Program $p(E_2(\#n)ij)$

```
Input: $E_2(\#n)ij$ A string from $\{0,1\}^*$ where: $E_2$ from definition 1.5
    $\#n \in \{0,1\}^*$ – the smallest program outputting $n$ in $2n$ steps or less
    $i \in \{0,1\}^{f+1-\log(\alpha)}$ – the chunk number, ranging from 1 to $\frac{2^{f+1}}{\alpha}$
    $j \in \{0,1\}^{\log(\alpha)-\log(g)}$ – the try's number, ranging in $[1; \alpha/g]$.
Procedure:
Retrieve $\#n$ from $E_2(\#n)ij$
Run $\#n$ as a program and call $n$ its output
Compute $f(n)$, $T(n)$, and $\alpha(n)$
Retrieve $i$ and $j$ from $E_2(\#n)ij$ possible since we know $f(n)$ and $\alpha(n)$
Create an empty list $l$ to store outees
for $k = 0$ to $\alpha - 1$ do
    Simulate program number $(i-1)\frac{2^{f+1}}{\alpha} + k$ for $T(n)$ simulation steps
    if it output a value of size $n$ then
        Add the output to list $l$
    end if
end for
Output: Run the strategy $\sigma_i^N(\frac{2^{f(n)}}{\alpha(n)}, \alpha(n), j)$ on $l$ (seen as a multiset) where $N = 2^n$.
```

Size analysis. We analyze the size of $p(E_2(\#n)ij)$:

- $\#n$ is of size $K^{2n}(n)$ (by definition);
- i is of size $f - \log(\alpha)$;
- j is of size $\log(\alpha) - \log(g)$;

- $E_2(\#n)ij$ is of size $\leq 2K^{2n}(n) + f - \log\alpha + O(1)$;
- the program p is of size $O(1)$.

As a consequence, $p(E_2(\#n)ij)$ is of size less than $2K^{2n}(n) + f - \log(g) + O(1)$. By reasonability of $\log(g)$ this is $\leq f(n)$ for infinitely many n.

Running time and correctness. The analysis is similar to the one try case.

4.7 Results

We now combine the program of section 4.6 and the general strategy described in section 4.5.2 for the game $\mathcal{G}^N(p,k,1)$ with $N = (k^p)^{(k^p)}$. We obtain in this way the following result which improves on the result by Longpré.

Theorem 4.22: K^t hierarchy in the TM model

We place ourselves in the TM model. Let T and f be increasing VTC functions of $\mathbb{N} \mapsto \mathbb{N}$ in $\mathsf{DTIME}(T(n))$ such that $T(n) \geq n\log(n)$ and $f(n) = o(\log(n))$. Let $\alpha = \max(\frac{f2^f}{\log n}, \sqrt{f})$. Then:

$$\exists c \in \mathbb{R}, \exists_\infty n, [f, c\alpha T](n) \nsubseteq [f,T](n)$$

Or abusively:

$$\exists_\infty n, [f, O(\alpha T)](n) \nsubseteq [f,T](n).$$

$T(n)$ being greater than $n\log n$ comes from the running time of the strategy

Proof

The proof consists of two parts. First, we analyze the bound $\mathfrak{B}(p,k)$ which will give us the restrictions on f and α. We will then check that there are programs implementing the general strategy and having the proper size and running time.
Let us recall that that we need $\mathcal{B}(p,k) = (k^p)^{(k^p)}$ to be bounded above by 2^n since the outant needs to be of size N. Here we have:

- $p = 2^{f+1}/\alpha$
- $k = \alpha$
- $pk = 2^{f+1}$
- We need $(k^p)^{k^p} \leq 2^n$

We can then compute:

$$\begin{aligned}
(k^p)^{k^p} &\leq 2^n && \Leftrightarrow \\
k^p p\log(k) &\leq n && \Leftrightarrow \\
\alpha^{2^{f+1}/\alpha} 2^{f+1}/\alpha \log(\alpha) &\leq n && \Leftrightarrow \\
2^{f+1}/\alpha \log\alpha + f + 1 - \log\alpha + \log\log\alpha &\leq \log n && \Leftrightarrow
\end{aligned}$$

This inequality may only be satisfied if $f(n) = o(\log n)$. Under this assumption, taking any $\alpha \geq \frac{f2^f}{\log n}$ makes the inequality hold. We actually set $\alpha = \max(\frac{f2^f}{\log n}, \sqrt{f})$. Note that α and $\log(\alpha)$ are VTC and increasing to infinity and thus reasonable by theorem 4.4. The $\max(\cdot, \sqrt{f}))$ is necessary as we need α to be reasonable even when $f = o(\log\log(n))$, the square-root may be replaced by some other function. Note that $\alpha = o(2^f)$ which is necessary to improve on the result of Longpré.
Now, consider the programs $p(< \#n, i >)$ from subsection 4.6.1, with $f = o(\log(n))$ and $\alpha = \frac{f2^f}{\log n}$. We recall that i ranges in $[1; 2^{f+1}/\alpha]$ and that $\#n$ is the smallest program outputting n in $2n$ steps. The infinite sequence of $(n_j)_{j \in \mathbb{N}}$ we consider are all the values of n such that $3K^{2n}(n) < \log(\alpha(n))$ in ascending order. This is an infinite sequence by reasonability of α. We now claim that:

1. **Size of the program** $\forall_\infty, \forall i, j \in N$, $p(< \#n_j, i >)$ is of size $\leq f(n_j)$.
2. **Running time of the program** with a proper implementation of the strategy of section 4.4.5,$p(< \#n_j, i >)$ runs in $c\alpha T(n_j)$ for all $j \in \mathbb{N}$.
3. **Correctness** for all $j \in \mathbb{N}$, there exists $i \in [2^{f(n_j)}/\alpha(n_j)]$, such that no program of size $f(n_j)$ running for $T(n_j)$ steps outputs a string equal to $\mathbf{Out}(p(< \#^n_j, i >))$; moreover $\mathbf{Out}(p(< \#n_j, i >))$ is of size n_j.

Once those 3 points are established, it follows that $\forall j \in \mathbb{N}, \mathbf{Out}(p(< \#n_j, i >)) \in [f, c\alpha T](n_j) \setminus [f, T](n_j)$, and thus $\exists_\infty n, [f, O(\alpha T)](n) \nsubseteq [f, T](n)$.

Size of the program. We first check that $\forall_\infty j, \forall i, p(< \#n_j, i >)$ is of size less than $f(n_j)$. Our program has the same structure as the program described in section 4.6 and the strategy of section 4.5.2 may be described in $O(1)$ bits. Therefore the size of $p(< \#n_j, i >)$ is less than

$$2K^{2n_j}(n_j) + f - \log(\alpha) + c$$

By definition of the n_j this is less than than $f(n_j)$ for all but finitely many j.

Running time of the program. We check that our program runs in time less than $c\alpha T(n_j)$ for all n_j. In particular this involves checking that one can implement the strategy of section 4.5.2 in time O(αT).

- Retrieving n_j from $\#n_j$ is done in $\leq 2n_j \leq 2T(n_j)$ steps for all j by definition.
- Computing $f(n_j), T(n_j)$ is done in $\leq T(n_j)$ steps by supposition on f and T. One can check that it is also possible to compute $\alpha(n_j)$ in $O(T(n_j))$ steps.
- Simulating the execution of the i-th chunk of size α of programs of size f i.e. getting the α inputs for our strategy takes time $O(\alpha T)$ as explained in section 4.6.
- Lastly, we describe an efficient implementation of the strategy described in 4.5.2. First we have α inputs written in base 2 which we must convert base $m \geq k^p = o(n)$. Notice that the strategy works if we take base $m = 2^{\lceil \log k^p \rceil}$, and converting a binary number to a base 2^m can be done "for free" by considering chunks of size m. Thus we set $m = 2^{\lceil \log k^p \rceil} \leq 2k^p$

In the strategy we receive α inputs which are bitstrings of size n that we view as numbers of k^p digits written in base m. We call our inputs $x^1, \dots x^\alpha$ and denote by x^i_j the j-th digit of x^i in base m. We call the output of our strategy y; y written in base m is of the form $y = x^{i_1}_{j_1} \dots x^{i_K}_{j_K} \nu$ where $x^{i_l}_{j_l}$ are digits of our inputs taken according to the strategy and ν is chosen to be different from all $x^i_j \forall i, j \in [k^p]$. We represent our inputs in figure 4.7

To compute the digits $x^{i_l}_{j_l}$ we have to do a BFS on the tree described in figure 4.6. This can be done in $O(\alpha n \log(\alpha n)) = O(\alpha \log(n)) = O(\alpha T)$

To compute ν we have to solve $OUTSIDE_{\log(m), \alpha m}$, using theorem 4.6 and the fact that $m \log m = O(n)$ this can be done in time $O(n\alpha(\log(m) + \log(\alpha))) \leq O(n\alpha \log n) = O(\alpha T)$

Correctness. This follows from the argument outlined in paragraph correctness of section 4.6.1, and by the fact that $\mathfrak{S}^{2^{n_j}}(\frac{2^{f(n_j)+1}}{\alpha(n_j)}, \alpha(n_j), 1)$ is winning.

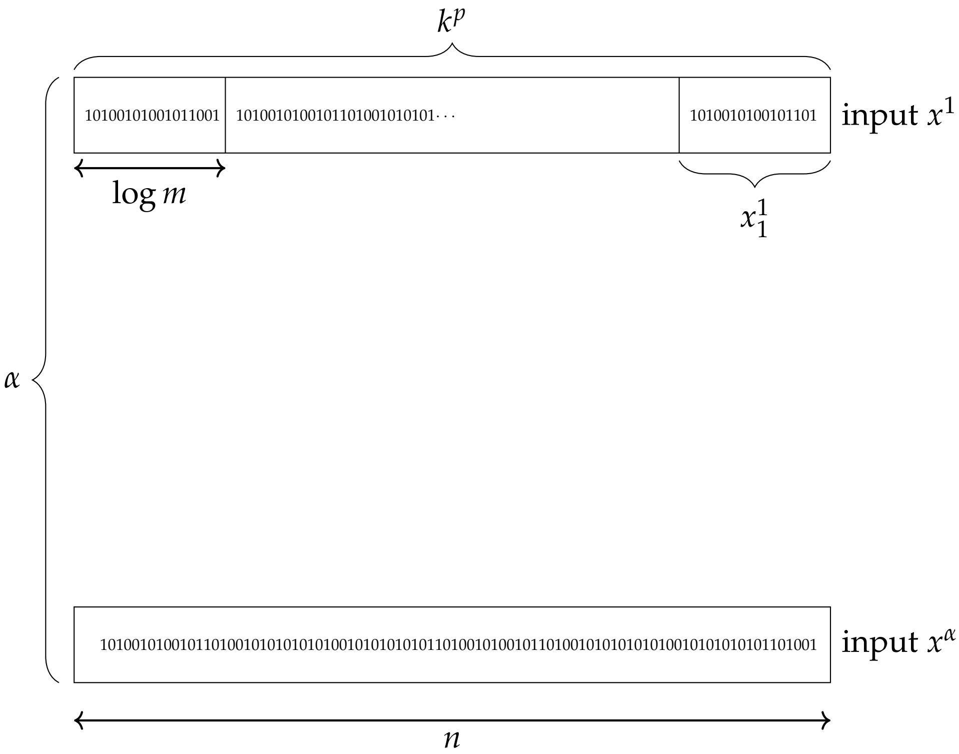


Figure 4.7: Binary representation of our inputs in the strategy.

Note several things about our result:

- In the proof the only part which makes our result $\exists_\infty$ instead of $\forall_\infty$ is that $f = o(log(n))$ for all other f the result would be $\forall_\infty$.
- When f is not in $\mathsf{DTIME}(T)$, the result may still stand with adjustments. For instance, we can indicate the value $f(n)$ to p for a cost of $\log f(n)$ bits.
- Our results hold under oracles.

Theorem 4.23: K^t hierarchy in the RAM model

Let T and f be increasing VTC functions of $\mathbb{N} \mapsto \mathbb{N}$ in $\mathsf{DTIME}(T(n))$ such that $T(n) \geq n$ and $f(n) = o(\log(n))$. Let $\alpha = \max(\frac{f2^f}{\log n}, \sqrt{f})$. Then:

$$\exists c \in \mathbb{R}, \exists_\infty n, [f, c\alpha T](n) \not\subseteq [f, T](n)$$

Or abusively:

$$\exists_\infty n, [f, O(\alpha T)](n) \not\subseteq [f, T](n).$$

Proof

The proof is the same as in the TM case. The part which differs is checking that we can implement the strategy in time $O(n\alpha)$. This presents no significant issue.

4.8 Additional links and musings for future work

4.8.1 Links with the time hierarchy theorem with advice

For completeness and potential future work, we show in this subsection what the time hierarchy (for Turing machine with advice) implies about a hierarchy for temporal Kolmogorov complexity. It turns out we get a hierarchy, but for a different notion of Kolmogorov complexity (see definition 4.24). We also show that the result of theorem 4.22 is indeed not easily obtainable from an analysis of the time hierarchy theorem. Lastly we mention recent advancements in the domain [5].

A function $g \in \{0,1\}^* \mapsto \{0,1\}$ is said to be computable in time T with advice of size f if there is a Turing machine which can compute g in time $O(T)$ when given an "*advice*" string of size $f(n)$ for inputs of size n (the advice is the same for every string x). We define complexity classes with advice in the following standard way.

Definition 4.20: $\mathsf{DTIME}(T)/f$

Let $T, f \in \mathbb{N} \mapsto \mathbb{N}$, a function $g : \{0,1\}^* \mapsto \{0,1\}$ is in $\mathsf{DTIME}(T)/f$ if there exists a TM M such that:

$$\exists c \in \mathbb{N}, \forall n, \exists a \in \{0,1\}^{f(n)}, \forall x \in \{0,1\}^n,\ M(x,a) = g(x) \text{ in } cT(n) \text{ steps or less}$$

By folklore results do have time hierarchy theorems for complexity classes with small amounts of advice.

Theorem 4.24: Time hierarchy theorem for complexity classes with advice [SW14]

Let $T : \mathbb{N} \mapsto \mathbb{N}$ be a time computable function such that $T(n) \geq n$ and $\alpha : \mathbb{N} \mapsto \mathbb{N}$ be an increasing function computable in $\mathsf{DTIME}(T(n))$ such that $\alpha(n) \overset{\infty}{\rightarrow} \infty$ and α is VTC then: $\mathsf{DTIME}(\alpha T \log T) \not\subset \mathsf{DTIME}(T)/n$

Proof

Let M be the TM defined as such : For all $n \in \mathbb{N}$ and all inputs $x \in \{0,1\}^n$, M simulate the advice-taking TM $N \overset{\text{def}}{=} M_{|x|}$ on input x and advice x for $\alpha T(n)$ steps ; if there was any output M reverses it, otherwise M outputs 1. This simulation can be done in $c_N \alpha T \log T$ where c only depends on the TM being simulated. Thus $M \in \mathsf{DTIME}(\alpha T \log T)$.
Consider TM $N \in \mathsf{DTIME}(T)/n$ with advice sequence $(a_n)_{n\in\mathbb{N}}$, note that a_n is a string of size n. We have that on all input N can be computed in exactly less than dT steps for a specific d. Consider n an integer such that n (in binary) is a representation of N and $c_N T\alpha \log T(n) > dT(n)$. Such n do exist. Now consider $x = a_n$, we have that $M(x) = M(a_n) = \neg M_{|x|}(x,x) = \neg N(a_n, a_n)$. Thus $M \neq N$ for all $N \in \mathsf{DTIME}(T)/n$.

In order to relate these notions to K^t complexity we introduce the following.

---

[5] This subsection is the results of informal talks with Shuichi Hirahara, it is a last minute addition, hence the very last paragraph of this section discussing recent advancements in the domain is not satisfying as the guy writing this thesis could not read the paper in the allocated time.

Definition 4.21: Truth table of a function $tt(f)$

Let $f : \{0,1\}^n \mapsto \{0,1\}$, we define $tt(f)$ as the string of size 2^n being equal to the truth table of f. Formally

$$\forall i \in [0;2^n - 1], tt(f)_i = f(i)$$

where i is interpreted as it binary representation over n bits in $f(i)$.

Definition 4.22: f_n

Given a function $f \in \{0,1\}^* \mapsto \{0,1\}$, we define f_n as the function f restricted to $\{0,1\}^n$

Next we defined an alternate definition of Kolmogorov complexity. We relate it to the usual notion in lemma 4.2.

Definition 4.23: Kb^t Bit-wise temporal Kolmogorov complexity [a]

---

[a] This may have another name in the literature

Let $x \in \{0,1\}^*$ and $t \in \mathbb{N}$, we define Kb^t as:

$$Kb^t(x) \overset{\text{def}}{=} \min\{|p| \in \{0,1\}^* | \forall i \in [0, |x|-1], \mathbb{U}(< p, i >) = x_i \text{ in less than t steps}\}$$

where $\mathbb{U}(< p, i >)$ is meant as program p running on input i as simulated by $\mathbb{U}$ an efficient universal TM.

With this new notion of Kolmogorov complexity we can define its associated class.

Definition 4.24: $[T,f]_b$

Let T, f be two functions of $\mathbb{N} \mapsto \mathbb{N}$ then we define

$$[T,f]_b(n) \overset{\text{def}}{=} \{x \in \{0,1\}^n | Kb^{T(n)}(x) \leq f(n)\}$$

Note that contrary to $[T,f]$ it makes sense to consider functions $T < n$ for $[T,f]_b$. For instance $\forall n, 0^n \in [O(1), O(1)]$. We will still restrict ourselves to function $f \leq n$ however, for similar reasons to the case $[T,f]$.

Relating definition 4.24 and 4.3 note that we have the following inclusions.

Lemma 4.2: Comparison between the two Kolmogorov classes $[T,f]_b$ and $[T,f]$

For all T, f functions of $\mathbb{N} \mapsto \mathbb{N}$ computable DTIME(T) we have:

$$\forall n \in \mathbb{N}, [T,f]_b(n) \subset [O(2^n T), f + 2K^{2^n}(n) + O(1)](n)$$

and

$$\forall n \in \mathbb{N}, [T,f](n) \subset [T+n, f + O(1)]_b(n)$$

With the precision that the "big $O(\cdot)$" mean that there exists a constant such that this is true.

Proof

$[T,f] \subset [T+n, f+O(1)]_b$: If x can be printed in time T by a program of size f, then so can a singular bit of x (with some minor time and program adjustments).

$[T,f]_b(n) \subset [O(2^n T), f + 2K^{2^n}(n) + O(1)](n)$: Take $x \in [T,f]_b(n)$, there is by definition a program p of size f which prints any bit of x in time T (through the simulation of a universal TM). Consider program p' which takes as input a representation of n decodable in 2^n steps, and then runs p for every bit $i \in [0; n-1]$ to retrieve x. This takes time $O(2^n T)$ and program p' is of size less than $f + 2K^{2^n}(n) + O(1)$.

We can now give an alternative definition of $\mathsf{DTIME}(T)/f$ which relates it to Kb

Lemma 4.3: Alternate definition of $\mathsf{DTIME}(T)/f$

Let f, T two functions of $\mathbb{N} \mapsto \mathbb{N}$, let

$$A \overset{\text{def}}{=} \bigcup_{c \in \mathbb{N}} \mathsf{DTIME}(T)/(f+c)$$

and

$$B \overset{\text{def}}{=} \{g : \{0,1\}^* \mapsto \{0,1\} | \exists c \in \mathbb{N}, \forall n \in \mathbb{N}, Kb^{cT(n)}(tt(g_n)) \leq f(n) + c\}$$

then

$$A = B.$$

Proof

$$A \subset B$$

Let $g \in \mathsf{DTIME}(T)/f$ then it is solved by a Turing machine $M \in DTIME(T(n))$ and with advice bits sequence α_n such that $\forall n, \alpha_n \leq f(n) + c$ thus we have $< M, \alpha_n > \leq O(1) + f(n)$ and M runs in time less than $cT(n)$. Ergo for a certain c $Kb^{cT(n)}(tt(g_n)) \leq f(n) + c$.

$$B \subset A$$

Let $g \in B$, consider α_n the program of size $\leq f(n) + c$ which computes g_n in $cT(n)$ steps or less, i.e. $\forall i \in \{0,1\}^n, \alpha(i) = tt(g_n)_i$. The result follows immediately.

Thus a function of $g \in \mathsf{DTIME}(T)/f + O(1)$ can be seen as a sequence of function $(g_n : \{0,1\}^n \mapsto \{0,1\})_{n \in \mathbb{N}}$ which verify $\exists c \in \mathbb{N}, \forall n \in \mathbb{N}, Kb^{cT(n)}(tt(g_n)) \leq f(n) + c\}$.

Theorem 4.25: Hierarchy for Kb^t

Let f, T two functions from $\mathbb{N} \mapsto \mathbb{N}$ such that $f(n) = o(\log(n))$,then

$$\exists c \in \mathbb{N}, \exists_\infty n, [cT \log T, c]_b(n) > [T, f]_b(n)$$

Proof

Let T be a time computable function, by theorem 4.24 there exists a function $g \in \mathsf{DTIME}(T\log T)$ and $g \notin \mathsf{DTIME}(T)/n$.

- $g \in \mathsf{DTIME}(T\log T)$ entails by lemma 4.3

$$\exists d, \forall n, Kb^{dT\log T(n)}(tt(g_n)) \leq d$$
$$\boxed{\text{Call } N = 2^n \text{ the size of } g_n}$$
$$\Rightarrow \exists d, \forall n, tt(g_n) \in [dT\log T(n), d](N)$$
$$\Rightarrow \exists d, \forall n, tt(g_n) \in [dT\log T(\log(N)), d](N) \qquad (*)$$

  In the rest of the proof we fix d a witness to property $(*)$.

- $g \notin \mathsf{DTIME}(T)/n$ entails

$$\forall c,\ g \notin \mathsf{DTIME}(T(n))/o(n)$$
$$\xRightarrow{\boxed{\text{by lemma 4.3}}} \forall c, \exists_\infty n,\ Kb^{cT(n)}(tt(g_n)) \leq o(n) + c.$$
$$\boxed{\text{Call } N = 2^n \text{ the size of } g_n}$$
$$\Rightarrow \forall c, \exists_\infty n,\ tt(g_n) \notin [cT(n), o(n) + c](N)$$
$$\Rightarrow \exists_\infty n,\ tt(g_n) \notin [T(\log N), o(\log N)](N)$$

Combining both results we get:

$$\exists_\infty n, tt(g_n) \in [dT\log T(\log N), d](N) \setminus [T(\log N), o(\log(N))](N)$$
$$\boxed{\text{Defining } T'(n) \overset{\text{def}}{=} T(2^n)}$$
$$\Rightarrow \exists_\infty N, [dT'\log(T'), d](N) > [T', o(\log N)](N)$$

Which proves the theorem.

Theorem 4.25 which is quite simple to prove is to be compared with theorem 4.22. It would seem that by combining lemma 4.2 and theorem 4.25 we can get a hierarchy for regular Kolmogorov classes $[T, f]$ à la theorem 4.22 but that is not the case. The reason is that a 2^n time factor appears when writing $[T, f]_b \subset [2^n T, f + O(K^{2^n}(n))]$ and 2^n is always bigger than 2^f so we don't get any improvements when compared with Longpré's result.

Recent advancement in the time hierarchy theorem (THT) with advice In paper [VW23] the authors come up with better "local" algorithms for the MISSING-STRING problem (the problem we call OUTSIDE in definition 4.6) and derive a better THT with advice. They also show various implications of the existence of better algorithms for the MISSING-STRING problem. Their approach seems very related to ours.

Question

How can we relate the analysis of games $\mathcal{G}(p, k, n)$ to the analysis of the MISSING-STRING problem on circuits ? To that of the THT with advice ?

4.8.2 Establishing lower bounds for any strategy

We now sketch a promising way to reason about winning strategies for novelty games. This could be used either to construct new ones, or establish lower bounds for $\mathfrak{B}(p,k,m)$. The idea is to study the smallest number of antecedents any element y in the image of $f : \binom{[N]}{k} \to [N]$ may have. The hope is that by analyzing such restrictions we get lower bounds on N.

Definition 4.25: $(\hat{x}_i^j(x))$

Given a sequence $(x_i)_{i\in k} \in \mathbb{N}^k$, an element $x \in \mathbb{N}$ and some $j \in [1;k]$, we will denote by $(\hat{x}_i^j(x))$ the sequence such that $\hat{x}_i^j(x) = x$ if $i = j$ and $\hat{x}_i^j(x) = x_i$ otherwise.

Definition 4.26: Antecedents of a function f $\mathcal{A}_f(x)$

Let f be a function, $\binom{[N]}{k} \to [N]$. We define $\mathcal{A}_f(y)$ as the set

$$\mathcal{A}_f(y) = \{x \mid \exists (x_i) \in [N]^k, \exists j \in [1;k], f(\hat{x}_1^j(x), \dots, \hat{x}_k^j(x)) = y\}.$$

The set $\mathcal{A}_f(y)$ contains all values of x such that there is a way to complete x with other numbers in order to produce y by the function f. Said differently: $\mathcal{A}_f(y) = \bigcup_{B\in f^{-1}(y)} B$.

We naturally extend the definition to apply to sets of number.

Definition 4.27: $\mathcal{A}_f^i(B)$

Let $B \subset [N]$, then $\mathcal{A}_f(B) = \bigcup_{y\in B} \mathcal{A}_f(y)$.
We define $\mathcal{A}_f^1(B) = \mathcal{A}_f(B)$ and for any $i \in \mathbb{N}^*$, $\mathcal{A}_f^i(B) = \mathcal{A}_f^{i-1}(\mathcal{A}_f(B))$.

If $B = \{x\}$ we abusively write $\mathcal{A}_f^i(x)$ instead of $\mathcal{A}_f^i(\{x\})$.

Definition 4.28: Non self-hitting (NSH) functions

Let f be a function $\binom{[N]}{k} \to [N]$. We say that f is non self-hitting if $\forall B \in \binom{[N]}{k}, f(B) \notin B$.

It is easy to prove that we need only consider strategies which are non self hitting.

Theorem 4.26:

If there is a winning strategy for the game $\mathcal{G}(p,k,m)$, then there is a strategy with no self hitting functions winning for game $\mathcal{G}(p,k,m)$.

As a consequence, all considered functions from now on will be supposed to be non-self hitting.

Theorem 4.27: Equivalence theorem for $(2, k)$

Let f be an NSH function $\binom{[N]}{k} \to [N]$, there exists g such that (f, g) is a valid strategy for the game $\mathcal{G}(2, k, 1)$ if and only if

$$\forall x_1, \ldots, x_k \in [N]^k, \bigcup_{i=0}^{n} \mathcal{A}_f(x_i) \neq [N].$$

Proof

For the right to left implication, it is clear that if the property is satisfied then there exists a function g such that

$$\forall x_1, \ldots, x_k, g(\{x_1, \ldots, x_k\}) \in [N] \setminus \bigcup_{i=0}^{n} \mathcal{A}_f(x_i).$$

To prove the the converse implication, suppose there exists $x_1, \ldots, x_k$ such that $\bigcup_{i=0}^{n} \mathcal{A}_f(x_i) = [N]$, and call $a = g(\{x_1, \ldots, x_k\})$. We know a belongs to $\bigcup_{i=0}^{n} \mathcal{A}_f(x_i)$ and we can WLOG suppose that it belongs to $\mathcal{A}_f(x_1)$. Then there exists a set B of size k such that $a \in B$ and $f(B) = x_1$. Therefore, the strategy fails on occurence $(B, \{x_1, \ldots, x_k\})$, leading to a contradiction.

Corollary 4.2:

Let f be an NSH function$\binom{[N]}{k} \to [N]$. If

$$\forall x_1, \ldots, x_k \in [N]^k, \bigcup_{i=0}^{n} |\mathcal{A}_f(x_i)| < \frac{N}{k},$$

then there exists g such that (f, g) is a valid strategy for the game $\mathcal{G}(2, k, 1)$.

Following this line of thought, our idea to establish lower bounds for $\mathfrak{B}$ is to study a subclass of strategies on which it is easier to reason and then claim that any winning strategy can be reduced to a function of this subclass. To that end we introduce balanced function, a balanced function is a function whose every image has the same number of antecedents.

Definition 4.29: Balanced functions

A function $f : \binom{[N]}{k} \to [N]$ is *balanced* if

$$\forall y \in [N], \left| |\mathcal{A}_f(y)| - \frac{1}{N}\binom{N}{k} \right| \leq 1.$$

A strategy is said to be balanced if the functions for players are all balanced functions.

The strategy we described for the game $\mathcal{G}(2, 2, 1)$ and for $\mathcal{G}(p, k, 1)$ are functions which on top of being NSH, oblivious and balanced have lots more symmetries. Thus we may have to consider additional symmetries to our functions in order to establish lower bounds efficiently. We leave the following questions for future work:

Question

1. Can we reduce our analysis of general winning strategies to winning strategies with many symmetries ?
2. For game $\mathcal{G}(p,k)$ and for a certain function h, can we get lower bounds for functions with many symmetries of the form

$$\forall x_1, \ldots, x_k \in [N]^k, \bigcup_{i=0}^{n} |\mathcal{A}_f^p(x_i)| < h(N,k,p) \text{ ?}$$

Part III

Algebraic computation [Léc23]

Chapter 5

NC vs P in the algebraic world

In this chapter we give an alternate and simpler proof of the fact that PRAM without bit operations (shortened to iPRAM for integer PRAM) as considered in paper [Mul99] cannot solve the maxflow problem. To do so we consider the model of PRAM working over real number (rPRAM) which is at least as expressive as the iPRAM models when considering integer inputs. We then show that the rPRAM model is as expressive as the algebraic version of NC : algebraic circuits of fan-in 2 and of polylog depth noted $\mathsf{NC}^{\mathsf{alg}}$. We go on to show limitations of the $\mathsf{NC}^{\mathsf{alg}}$ model using basic facts from real analysis : those circuits compute low degree piece wise polynomials. Then, using known results we show that the maxflow function is not a low-degree piece-wise polynomial. Finally we argue that $\mathsf{NC}^{\mathsf{alg}}$ is actually a really limited class which limits our hope of extending our results to the boolean version of NC.

5.1 Overview

A fundamental question in computer science is whether one can solve a problem faster if one has more computers, i.e. if the ability to parallelize your computations yields faster algorithm. It's evident that for some problem the answer is yes, parallelization does yield faster algorithm, for instance checking if every bit of a bitstring is 1 can be done in $O(1)$ in parallel but requires $O(n)$ with a TM. Though it's to be noted that apart from trivial "you have to check all the bits" argument (so for super-linear bounds) we do not know of any problem provably made faster by parallelization. In any case the converse case is of high interest, i.e. is there a problem which cannot be made faster through parallelization, where the steps to solve the problem are inherently sequential ? This problem is known as $P \overset{?}{=} NC$ where P is the class of problem solvable by a sequential algorithm running in polynomially many steps and NC is the class of problems solvable in polylog time by polynomially many computers. This question as most other non trivial questions in complexity theory is unsolved. One candidate problem not to be in NC is the maxflow problem which is proven P-complete under NC reductions, meaning if this problem is in any NC then so is all of P.

Definition 5.1: Maxflow problem

Given a directed graph $G = (V, E)$ with a source node $s \in V$, a sink node $t \in V$, and a capacity function $c : E \to \mathbb{N}_{>0}$ that assigns a non-negative capacity to each edge, the objective is to find a flow function $f : E \to \mathbb{N}_{>0}$ that satisfies the following conditions:

1. **Capacity Constraint**: For every edge $(u, v) \in E$,

$$0 \leq f(u, v) \leq c(u, v)$$

2. **Flow Conservation**: For every node $v \in V \setminus \{s, t\}$,

$$\sum_{u:(u,v)\in E} f(u, v) = \sum_{w:(v,w)\in E} f(v, w)$$

3. **Flow Value**: The value of the flow is defined as the total flow out of the source node s (which is equal to the total flow into the sink node t):

$$|f| = \sum_{v:(s,v)\in E} f(s, v) - \sum_{u:(u,s)\in E} f(u, s)$$

The goal is to maximize the flow value $|f|$.

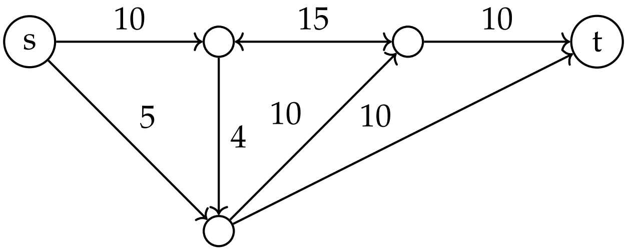


Figure 5.1: Instance of the maxflow problem. The optimal flow is 15.

In computer science when a lower bound is too hard to prove researchers may restrict algorithms to a certain class. For instance the fastest sorting algorithms we have run in time $\mathsf{DTIME}(n \log(n))$ and in the special case where we only consider algorithms based on comparisons we may prove that it is an optimal lower bound, but proving the same result for general algorithm would be a major result.

Back to maxflow, to get a hold of this problem Mulmuley in 1999 published a paper [Mul99] showing that if you consider only algorithms which perform algebra $(+, *, /, >)$ and basic loop and memory manipulation then you can prove that the maxflow problem cannot be solved in polylogarithmic time by polynomially many algebraic machines. These restrictions make sense as intuitively these would be the operations you want to use to solve the maxflow problem and indeed common algorithms for the maxflow problem are algebraic. The main thing algebraic algorithms prevent (or rather make costly) is accessing bits of the capacities in the flow-graph. Mulmuley's proof is based on quite advanced mathematics, our contribution is to simplify considerably his proof and give another insight on the gap between proofs in the algebraic world and the usual bit world.

5.2 Introduction

5.2.1 Previous work

An important question in complexity theory is determining if $P = NC$, where P is the set of languages solvable by a polynomial time Turing machine and NC the set of languages solvable by uniform family of poly-logarithmic depth, polynomial size circuit. It is thought that $P \neq NC$ but no one has come close to proving it. The maxflow problem is a problem in P where the inputs are directed graphs whose edges are labelled with integers called capacities, a specified node s called the source, a specified node t called the sink, and an integer threshold k, the input is in the language if a flow greater than k units may flow from s to t. The maxflow problem is known to be P complete with regards to NC reductions [GSS82], meaning that membership of the maxflow problem to NC is equivalent to $P = NC$. In his celebrated result [Mul99], Mulmuley defines a computation model called integer PRAMs (iPRAM) without bit operation which is akin to the PRAM model except that inputs are integers (instead of bits) and the usual bit operations are replaced by multiplication, addition and comparison (all of those take $\Omega(1)$ time regardless of the size of the integers). He then proves that the maxflow problem cannot be computed by iPRAM with polynomially many processors running in polylogarithmic time. The model of iPRAM with polynomially many processors running in polylogarithmic time is basically an algebraic version of NC, which we denote by $\mathsf{NC}^{\mathsf{alg}}$, where the inputs are now integers and the gates multiplication, addition and comparison, and the fan-in of each gate is at most 2.[1] Moreover it was also known that the maxflow problem does have algebraic algorithm (working only using multiplication and addition) running in polynomial time, so in short Mulmuley proved that the maxflow problem is not in the algebraic version of NC yet it is in the algebraic version of P. So in the algebraic world $NC \neq P$.

Size of inputs For now we have very loosely spoken of "polynomial time", "poly-logarithmic depth"... it is to be understood that we mean "polynomial time in the size of the input", but what is the size of an input? It may seem trivial, but we insist upon it to clarify any future misconceptions. In the bit world (where languages are subsets of $\{0,1\}^*$, and inputs bitstrings), the size of the input is its number of bits. In the algebraic world (where languages can be thought of as subsets of $\mathbb{N}^*$ and inputs strings of numbers) the size of an input is its number of numbers. For instance let us consider a specific instance of a maxflow problem, say a weighted directed graph G with 10 nodes given via its matrix representation, two integers to specify which node is the source and which is the sink, and a integer threshold k. In the algebraic world the size of this instance is $10^2 + 2 + 1 = 103$. In the bit world we do not have enough information to conclude because we do not know the bitlength of the capactities. This is just to shed light on the difference between the two models, in the algebraic one only the number of integers count, no matter their bitlength. Now when showing impossibility results, for instance that maxflow is not in the algebraic version of NC, we will on top of it prove that the impossibility still stand even if we restricts integers to be of bitlength polynomial in the size of the input (i.e. polynomial in the number of numbers). This is interesting because it "levels the playing field" between the algebraic world and the bit world, if the algebraic world comes short of solving the maxflow problem it will not be because of inputs of very large bitlengths[2].

---

[1] making the fan-in to be arbitrary wouldn't change the model, we can replace a large fan-in gate with a low height tree of gates

[2] This is how size was handled in [Mul99] hence our choice. Another way of defining the size of inputs in the algebraic world, perhaps more intuitive or familiar to some readers, is to include the bitlength of the inputs

5.2.2 Our result

In this chapter we show that maxflow is not in the algebraic version of NC called $\mathsf{NC}^{\mathsf{alg}}$ (subsequently also not in iPRAM). We detail things as follows: our technique for showing the result. Contributions and differences between our paper and Mulmuley's. Discussion on the meaning of this result.

Technique

The different PRAM models and algebraic circuits are equivalent. Firstly we consider (in order to use real analysis) real PRAM noted rPRAM (introduced in [SPL22]). rPRAM are akin to iPRAM except inputs are real numbers instead of integers, on which we can perform addition, multiplication, *division* and comparison[3]. We argue efficiently that rPRAM are at least as potent as iPRAM when we restrict the inputs to integers (iPRAM are not defined otherwise). Then we consider the non-uniform[4] family of algebraic circuits whose inputs are (multiple) real numbers, whose gates are multiplication, addition, division, comparison (all of fan-in 2), whose depth is poly-logarithmic in the size of the input and whose largeness is polynomial in the size of the input. We call this class $\mathsf{NC}^{\mathsf{alg}}$, it is to be thought of as the algebraic equivalent of NC. We then give an argument to show that under a reasonable time slowdown and largeness blow up, $\mathsf{NC}^{\mathsf{alg}}$ can simulate rPRAM with polynomially many processors running in poly-logarithmic time. Now we can focus our attention on proving lowerbounds for $\mathsf{NC}^{\mathsf{alg}}$ which is much simpler than with rPRAM or iPRAM (but any lower bounds for $\mathsf{NC}^{\mathsf{alg}}$ entail ones for rPRAM and iPRAM). Considering $\mathsf{NC}^{\mathsf{alg}}$ instead of iPRAM is one of the key ingredient to the simplicity of our proof, the other one being restricting our attention to algebraic circuit over one variable.

$\mathsf{NC}^{\mathsf{alg}}$ is really limited. First we prove by induction that a family of algebraic circuits whose input is only one number are really simple functions: they are piece-wise polynomials [5] and each of those polynomials is of small degree (we bound the degree and the number of different polynomials by the depth of the circuit, see 5.3). So for instance any function $f : \mathbb{R} \mapsto \mathbb{R}$ with too many slope changes cannot be computed by an algebraic circuit of small depth.

Problem Then we explain how one can use this result over one input algebraic circuits to limit the complexity of algebraic circuits over multiple inputs: let us say we want to prove that a family of functions $g_n : \mathbb{R}^n \mapsto \mathbb{R}$ is not computable in $\mathsf{NC}^{\mathsf{alg}}$ (i.e. with depth $\mathsf{polylog}(n)$) then one just needs to find a family of functions $f_n : \mathbb{R} \mapsto \mathbb{R}^n$ computable with a circuit of depth $\mathsf{polylog}(n)$ such that $g_n \circ f_n$ (which is now a function from $\mathbb{R}$ to $\mathbb{R}$ so we can use our theorem) is complex enough not to be computed by algebraic circuits of depth $\mathsf{polylog}(n)$. Since f_n is computable by such circuits then it must be g_n which is not. Now for the maxflow problem specifically: call $g_n : \mathbb{R}^{n^2} \mapsto \mathbb{R}$ the function which takes a weighted directed graph given as its matrix representation and returns its maximum flow (the source is node 1 and the sink node n). One simply needs to find a simple function $f_n : \mathbb{R} \mapsto \mathbb{R}^{n^2}$ such that $g_n \circ f_n$

---

in the size. As a side note changing size like this would strengthen the class $\mathsf{NC}^{\mathsf{alg}}$ and our proofs would still work.

[3] the division is probably superfluous when considering the decision version of a problem (we might be able to simulate it using multiplication and addition) but it makes our model more potent in the general case. In any case if the goal is only to prove the original result of Mulmuley, one may (on second reading) disregard the division operation in the rest of the paper, it would make proofs even simpler

[4] uniformity never plays a role so we may as well not restrict our circuits

[5] actually polynomial fractions but this is an unimportant detail

is too complex to be computed by circuits of depth $\mathsf{polylog}(n)$. Such a family of functions f_n has been know for a long time ([Car83], Mulmuley even gives his own in [Mul99]), and they are actually very simple since they are affine functions, so definitely computable by algebraic circuits even of depth 1. Then we simply conclude that g_n in not computable by circuits of $\mathsf{NC}^{\mathsf{alg}}$.

Contributions and differences

The contributions of this paper are twofold. Firstly the simplicity of the proof when compared with Mulmuley's. We use very basic (univariate) polynomial analysis to limit the complexity of algebraic circuits, and then we use a known result about the parametric complexity of the maxflow problem to immediately conclude. Thanks to our proof this quite famous result may be proven to researchers or student in an hour or so. Secondly, because of the simplicity of the proof we feel it is much easier to understand the true limitations of $\mathsf{NC}^{\mathsf{alg}}$ (therefore iPRAM) and reveal that they are basically humongous and why even basic functions cannot be computed in this model of computation. Mulmuley knew about these limitations but we are more pessimistic than him. Additionally our technique exposes the fundamental and non-trivial link between computing on inputs of large bitlength and computing functions on many inputs of smaller bitlengths : we show that functions over one input cannot be computed even if we restrict the inputs to $x \in [0, 2^{n^d}]$ (where d is an integer) and we use it to conclude that some function g over $e \in \mathbb{N}$ inputs cannot be computed even if we restrict its domain to $[0, 2^n]^e$. The technique that Mulmuley uses is based on much more complicated mathematics and with a model (iPRAM) on which proofs are clumsier that on circuits. He basically uses the same plan and gives the same argument for the non-membership of maxflow to iPRAM: iPRAM may only compute "simple" multivariate-polynomial-like functions but using parametrization we can show that maxflow is not a simple function. To be clear, most of our mathematical arguments may be - from a logically formal standpoint - already contained in Mulmuley's but we do not think it makes this paper redundant at all. Those arguments were not readily apparent in Mulmuley. It is only after careful examination that one realizes that our mathematical arguments are embedded in the ones of Mulmuley's. Our proof, because we limit the expressiveness of univariate circuits to polynomial of low degree, very clearly exposes the shortcomings of the expressivity of the algebraic model. Those shortcomings were was not as apparent in Mulmuley's paper, because he deals with harder concepts (more or less multivariate polynomials): limitations on such objects do not seem as restrictive, and it still seems that we can compute "interesting" functions in iPRAM, so it gives a false sense of how expressive the class iPRAM is. On the other hand the approach used by Mulmuley, although it lacks in simplicity, may allow one to add operations on top of addition and multiplication, for instance he claims his results are still valid if one allows computing the parity of an integer. It is not immediately clear that our proof techniques can be salvaged if we add such an operation (adding the parity function allows us to create a polynomial with exponentially many pieces). Also Mulmuley shows that randomness still does not allow us to compute the maxflow problem. We do not consider randomness in this paper, our educated guess is that our technique can be fitted to include randomness.

One last hiccup is that for what seems could be purely technical reasons we were not able to show that the *decision* version of the maxflow problem cannot be computed in $\mathsf{NC}^{\mathsf{alg}}$. We only show that the search version (taking a graph and outputting its maximal flow) is not in $\mathsf{NC}^{\mathsf{alg}}$. At the end of the chapter we give promising leads to solve this slight issue.

Discussion

As explained before we believe our results shed light on the lack of potency of the models considered by Mulmuley and consequently ours. We are still far from proving that $NC \neq P$. To cement that idea we show that trivial problem for bit complexity class, like checking the parity of a bitstring (its last bit is 0) is obviously in NC (its even in AC^0), but it is not in $\mathsf{NC}^{\mathsf{alg}}$. This was known of Mulmuley but he somewhat disregards it, we argue later in the paper that it should not be disregarded. We, very humbly, think it somewhat dampens the hope of using such techniques to prove $P \neq \mathsf{NC}$, or at the very least that closing the gap using such techniques will be quite involved. Secondly we warn the reader: although the maxflow is special in the bit complexity class P, as it is complete, it is as far as we can tell not special in the algebraic world: we could not find general reductions from problems in algebraic-P to the algebraic maxflow problem. The maxflow problem is only special in the sense as it has an intuitive translation from the bit world to the algebraic world: every capacity gets mapped to one number. If we had taken another problem like primality testing for instance, the translation would not make much sense. Indeed in primality testing we have a number x and must determine if it is prime, its intuitive translation in the algebraic world would be similar: we have an integer x and must determine if it is prime. The issue is that in the algebraic world our input size would now be 1 (we have 1 number) and so circuits should now be of constant size.

All of that being said it is still interesting in its own right to know that the maxflow problem cannot be solved in poly-logarithmic time by polynomially many processors. Indeed algorithms used to solve the maxflow problem are, to our knowledge, all algebraic, they do not use the bitwise representation of numbers, only their absolute value. So it gives credible reason to believe $P \neq NC$.

5.3 Equivalence between PRAMS and algebraic circuits

We will start with definitions of classes iPRAM, rPRAM and $\mathsf{NC}^{\mathsf{alg}}$. Our definition of iPRAM is taken almost verbatim from Mulmuley's paper [Mul99]. Our definition of rPRAM is taken from [SPL22]

The iPRAM model (what Mulmuley calls PRAM without-bit operations) is like the usual PRAM model [JaJ11], the main difference being that its input are integers (instead of bits) and it does not provide instructions for any bit operations such as $\wedge$, $\vee$, or extract-bit. The model provides usual arithmetic ($+$, $-$, $\times$), comparison ($=$, $\leq$, $<$), store, indirect reference, and branch operations. Each memory location can hold an integer; a rational number is represented by a pair of integers – its numerator and denominator – both of which can be accessed by the processors separately. The processors share a global memory. The model does not provide instructions for truncation or integer division with rounding. The lower bounds are proved in the conservative unit-cost model in which each operation is assigned unit cost regardless of the bitlengths of the operands. Our definition is also sightly stronger than Mulmuley's, in the sense that in his setting indirect addressing is not allowed to depend on the input, for instance we may not use a capacity as an address. There are no such restrictions in our model.

Definition 5.2: iPRAM

A iPRAM is composed of $p \in \mathbb{N}$ processors working over integers which share a global memory on top of their own memory, both kind of memories are made of infinitely many registers labelled by an integer and each contain an integer. These processors receive integers as inputs and don't have access to any bit operation. Each processor executes a program which is a sequence of instructions. The instructions are of the following kinds:

1. $w = u \circ v$, where w denotes a memory location, u and v denote either memory locations or constants. Finally, $\circ$ denotes a binary operation $+, -, \times, \leq, <, =$. The comparison operations return 1 if true 0 otherwise
2. `go to` l, where l is the label of some instruction.
3. `if` $u \, \Delta \, 0$ `go to` l, where u denotes a memory location and Δ denotes either $<, \leq$, or $=$.
4. $u := v$: Store the contents of the memory location v into the location u.
5. $u :=\uparrow v$ (indirect reference): Treat the value in the memory location v as a pointer. Store into the memory location u the value in the pointed memory location.
6. `stop`.

Instructions over memory may modify the internal memory of the processor or the global shared memory, a flag distinguishes the two cases. All instructions take unit time to operate.
The processors run synchronously in parallel. We work in the CREW model meaning that no two processors may write at the same time at the same location of the global memory.

Definition 5.3: rPRAM

The rPRAM model is like the iPRAM model except the inputs are real numbers and we have an additional / (divide) gate (it is *not* the euclidian division). Furthermore each value is now stored in a memory location represented by a real number, e.g. we have an instruction store value x in cell y where both x and y are real numbers. We may also retrieve the content of a memory cell indexed by y a real number.

Notice that even though our memory cells are labelled by real numbers, at most polynomially many labels can be instantiated over the course of a computation using polynomially many processors and running in poly-logarithmic time.

In the following definition we define what it means to compute with an iPRAM or a rPRAM, there is nothing special about our definition except that the programs are non uniform.

Definition 5.4: Computing with iPRAM and rPRAM

Let R be an iPRAM, let $(x_1, \ldots, x_n)$ be integers we write $R(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ if when starting R with global shared memory initialized with $(x_1, \ldots, x_n)$ (the i-th register contains x_i) we end up with $y_1, \ldots, y_n$ written in the first n register of the global shared memory when all the processors of R have stopped.
Let $(n_i)_{i \in \mathbb{N}}$ be a sequence of integers. Given a sequence of function $(f_i)_{i \in \mathbb{N}} : \mathbb{N}^i \mapsto \mathbb{N}^{n_i}$, we say (f_i) is computed by sequence $(R_i)_{i \in \mathbb{N}}$ if $\forall i, \forall x_1, \ldots x_n, f_i(x_1, \ldots, x_i) = R_i(x_1, \ldots, x_i)$.
The definition is similar for rPRAM working with $\mathbb{R}$ instead of $\mathbb{N}$and where "the first i registers" become the "registers of address 1 to i"

In order to meaningfully compare the two models (iPRAM and rPRAM) we may restrict our view to integer inputs, in this setting it is clear enough that any computation made by an iPRAM can be made by a rPRAM.

Theorem 5.1:

Let $(n_i)_{i \in \mathbb{N}}$ be a sequence of integers. Let $(R_i)_{i \in \mathbb{N}}$ be an iPRAM computing a function from $\mathbb{N}^i$ to $\mathbb{N}^{n_i}$, there is a sequence $(R'_i)_{i \in \mathbb{N}}$ of rPRAM such that $\forall i, \forall (x_1, \ldots, x_i) \in \mathbb{N}^i, R_i(x_1, \ldots, x_i) = R'_i(x_1, \ldots, x_i)$

Definition 5.5: Algebraic circuit

An algebraic circuit is a circuit with gates $+, \times, -, /, =, <, \leq$ of fan-in 2, and constant gates n for all $n \in \mathbb{N}$. Its inputs are real numbers. The comparison gates $=, <, \leq$ return 1 when true, 0 otherwise. All other gates are interpreted in the obvious way.

We define P^{alg} the class of polynomial size algebraic circuits in the natural way.

Definition 5.6: P^{alg}

C_n is a sequence of P^{alg} if each C_n is an algebraic circuit of fan-in 2, working over n inputs, the size of each circuits grows polynomial in n and the depth polynomial in n.

We define NC^{alg}, this class is non-uniform unlike its boolean counterpart.

Definition 5.7: NC^{alg}

C_n is a sequence of $\mathrm{NC}^{\mathrm{alg}}$ (polylog-depth algebraic circuits) if each C_n is an algebraic circuit of fan-in 2, working over n inputs, the size of each circuits grows polynomial in n and the depth polylogarithmically in n.

We restrict the largeness of log-depth algebraic circuits to polynomials. If unrestricted they could be meaningfully bigger, indeed, even if we work with fan in 2 a circuit of depth $\log^d(n)$ could be of largeness as big as $2^{\log^d(n)}$ which is bigger than any polynomial for $d > 1$. Mulmuley considers such circuits in his paper, for simplicity we do not although our results would still stand. We now sketch a proof that $\mathrm{NC}^{\mathrm{alg}}$ (circuits) and $rPRAM$ (prams) with polynomially many processors and of running time polylogarithmic define the same class of functions.

Theorem 5.2:

$\mathsf{NC}^{\mathsf{alg}}$ (circuits) and $rPRAM$ (prams) with polynomially many processors and of running time polylogarithmic define the same class of functions.

Proof

It is not hard to see that one can simulate a circuit using a real PRAM. Just have one processor per gate.

The other direction is not so trivial, here is a sketch of the proof. Let us consider p processors in parallel running for k steps. Our circuit will consist of k layers $(l_i)_{i=1}^{k}$, each layer is meant to simulate one instruction step for all processors, each layer contains a certain number of gates in charge of computation (described later) and a certain number of gates representing the internal memory state of each processor (each processor modifies at most k internal addresses over the whole run), on top of that there is an additional number of gates representing the state of the shared memory. In order to deal with the processor accessing and modifying the shared memory we need to be careful, indeed the index of the cell accessed by a processor is unknown across a range of infinitely many indices, to deal with this issue we use a trick known as dynamic addressing, each memory cell will be represented by a pair of gates one containing the actual content and one containing the index of the cell. Only kp memory cells can be used at most by the processors so we add to each layer $2kp$ gates representing the state of the shared memory (index and content). Moreover all operations on the memory (like checking if a certain memory cell given by its address is empty) can be simulated using a overhead of $O(\log(kp))$. We now consider that there is only one processor in our rPRAM and we describe how it can be simulated by a circuit in NC^{alg}. For p processors it will just be a matter of gluing the different circuits in parallel. For each processor and for each layer we have $O(1)$ gates representing the whole program run by the processor, a gate representing the label of the current instruction. If we knew in advance the instruction meant to be executed and since we can access the shared memory as described above it would be easy to see that we could simulate any of the 6 instructions. Since we can't predict the current instruction we glue circuits each implementing one of the 6 instructions with a condition to check if the instruction of the current label indeed corresponds to that implemented by the circuit. In total our circuit is of size polynomial in pk and of depth $O(k\log(pk))$.

5.4 Limitations of log-depth algebraic circuits

In this section we give a direct proof that log-depth algebraic circuits over one variable compute piece-wise polynomial fractions of low degree and few pieces (namely less than 2^n if the depth is $\mathrm{polylog}(n)$). This is detailed in main theorem 5.3.

Definition 5.8: Interval

A set I over $\mathbb{R}$ is an interval if there exists $a, b \in (\mathbb{R} \cup \{-\infty; +\infty\})^2$ two real numbers such that $I = [a, b]$ or $I =]a, b]$ or $I = [a, b[$ or $I =]a, b[$.[a]

---

[a] since some reader may no be familiar with these notations: $]a, b]$ is equal to $[a, b] \setminus a$

Definition 5.9: Comparison of intervals

Let I and J be two non-empty, non-intersecting intervals over $\mathbb{R}$. We write $I < J$ if $\forall x \in I, \forall y \in J, x < y$. Let us have a collection of r intervals $(I_i)_{i=0}^{r}$, we write $I_1 < I_2 < \ldots < I_r$ to indicate that they are given in order and they are all non-empty and non-intersecting.

Definition 5.10: Interval cut and pieces

Given a family $(I_i)_{1 \leq i \leq r}$ of r intervals, $I_1 < I_2 < \ldots < I_r$ we say it is an interval cut of size r if $I_1, I_2, \ldots, I_r$ partitions $\mathbb{R}$. Each I_i is called a piece.

In the following we may use as a shorthand the notation I for a family of r intervals $(I_i)_{1 \leq i \leq r}$.

Definition 5.11: Intertwining interval cuts

Given an interval cut of size n $I_1 < I_2 < \ldots < I_n$ and an interval cut of size m $J_1 < J_2 < \ldots < J_m$. There is at least one interval cut of k-pieces $K_1, \ldots, K_k$ such that $\forall h \in [k], \exists i, j, K_h \subset I_i \wedge K_h \subset J_j$, if k is minimal then we call $K_1, \ldots, K_k$ an intertwining of I and J.

Note in the previous definition that when k is minimal it is smaller than $n + m$ (this is not hard to see). As an example of what is an intertwining consider the family $I = (]-\infty; 0],]0; +\infty[)$ and $J = (]-\infty; 1],]1; +\infty[)$, then one possible value of an intertwining is $(]-\infty; 0],]0; 1],]1; +\infty[)$.

Definition 5.12: Piece-wise polynomial fraction

A function f over $D \subset \mathbb{R}$ is said to be a piece-wise polynomial fraction if there exists $n \in \mathbb{N} \setminus \{0\}$ and an interval cut $I_1 < I_2 < \ldots < I_n$ and n polynomial fractions $f_1, f_2, \ldots, f_n$ such that $\forall i \in [\![1; n]\!], \forall x \in I_i \cap D, f(x) = f_i(x)$. [a]

Given such an interval cut $I_1 < I_2 < \ldots < I_n$ we may say f is a piece-wise polynomial fraction over I. The number of pieces of a piece-wise polynomial fraction is the smallest number n such that there exist $I_1 < I_2 < \ldots < I_n$ as described above. Let us write c the function which takes a piece-wise polynomial fraction f and return its number of pieces (we extend c to also take circuits as inputs when they compute piece-wise polynomial fractions).

---

[a] $[\![1; n]\!]$ denotes the integers in the interval $[1; n]$

For clarification the poles of a polynomial fraction do not add new pieces, for instance the function $f(x) = \frac{1}{x^2-1}$ defined for all $x \in \mathbb{R} \setminus \{-1; 1\}$ is a one piece polynomial fraction. If one wanted to have functions defined over the whole of $\mathbb{R}$ it would be no issue as whenever performing a division we could add a comparison gate in the circuit to verify that we are not dividing by 0, and if so return an arbitrary value.

Definition 5.13: Augmented degree of a polynomial fraction

Let $F = \frac{P}{Q}$ be a polynomial fraction over $\mathbb{R}$, meaning that P and Q are coprime polynomials over $\mathbb{R}$, then we define its augmented degree noted $d(F)$ as $d(F) = d(P) + d(Q) + 1$ (where $d(P)$ and $d(Q)$ denote the usual degree of polynomials)

Definition 5.14: Augmented degree of a piece-wise polynomial fraction

Let f be a piece-wise polynomial fraction over $\mathbb{R}$ with a minimal interval cut $I_1 < \ldots < I_n$ and $f_1, \ldots, f_n$ the associated polynomial fractions then we define $d(f) = max_{0 \leq i \leq n} d(f_i)$.
It is routine to check that this definition does not depend on the chosen interval cut.

This is the main theorem of the paper on which we derive all impossibility results of section 5.5.

Theorem 5.3:

A circuit P of depth k taking only one input, computes a function which is a piece-wise polynomial fraction. Let us call with $c(P)$ its number of pieces and $d(P)$ its augmented degree, then $d(P) \leq 2^k$ and $c(P) \leq 2^{\frac{k^2+3k}{2}}$

Proof

We prove the result for each gate by induction on the depth of the gate, both that we do indeed compute a piece-wise polynomial fraction and that the bounds are correct. Let us denote c_k the maximal number of pieces of a function computed by a gate at depth k. Let us denote d_k the maximal augmented degree of a function computed by a gate at depth k. We are going to prove that $d_k \leq 2^k$ and $c_k \leq 2^{\frac{k^2+3k}{2}}$ by induction.

For depth $k = 1$ the function computed is either $f = (x \mapsto x)$ or $g = (x \mapsto 1)$. Notice that $d(f) = 2, c(f) = 1, d(g) = 1$ and $c(g) = 1$. Therefore $d_1 \leq 2^1$ and $c_1 \leq 2^2$

Going from k to $k+1$: Let us consider a gate called p at depth $k+1$. Our induction hypothesis is that $d_k \leq 2^k$ and $c_k \leq 2^{\frac{k(k+1)}{2}}$.

Multiplication gate: $p = p_1 * p_2$. Let $r = c(p_1)$ and $m = c(p_2)$ we consider the interval cuts of p_1 and p_2 which we denote $I_1 < \ldots < I_r$ and $J_1, < \ldots < J_m$, let us consider $K_1, \ldots, K_k$ an intertwining of I and J such that $k < r + m$, note that p is a piece-wise polynomial fractions over K. Therefore $c(p) \leq c(p_1) + c(p_2) \leq 2 * c_k \Rightarrow c_(p) \leq 2 * c_k \leq 2^{\frac{(k+1)(k+2)}{2}}$. By reasoning over each piece of the interval cut K we have that $d(p) \leq d(p_1) + d(p_2) \leq 2d_k$.

Addition and division gate work in a similar fashion.

Comparison gate: $p = (p_1 \leq p_2)$. Let us define $n = c(p_1)$ and $m = c(p_2)$ and let $A_0 \leq A_1 \leq \ldots \leq A_n$ be the pieces of p_1, $B_0 \leq B_1 \leq \ldots \leq B_m$ the pieces of p_2. Let $I_0 \leq I_1 \leq \ldots \leq I_r$ be an intertwining of A and B. For all i, p_1 and p_2 are both polynomial fraction over I_i and we can canonically write

$p_{i,1} = \frac{q_{i,1}}{t_{i,1}}$ and $p_{i,2} = \frac{q_{i,2}}{t_{i,2}}$. We want to count the number of pieces of p, to that end we focus on each I_i one by one. Notice that over I_i any new piece of p contained in I_i may only occur on places where $p_{i,1} - p_{i,2}$ changes sign, moreover $p_{i,1} = p_{i,2} \Leftrightarrow q_{i,1} * t_{i,2} - t_{i,1} * q_{i,1} = 0$. But the polynomial $q_{i,1} * t_{i,2} - t_{i,1} * q_{i,1}$ has at most $d(p_1) + d(p_2) - 1$ roots, therefore over I_i we "add" at most $d(p_1) + d(p_2)$ pieces. Since we have at most $r \leq n + m = c(p_1) + c(p_2)$ pieces I_i, we have that c_p is at most $(d(p_1) + d(p_2)) * (c(p_1) + c(p_2)) \leq 2d_k * 2c_k \leq 2 * 2^k * 2 * 2^{\frac{k^2+3k}{2}} \leq 2^{\frac{(k+1)^2+3(k+1)}{2}}$. And $d(p)$ is equal to $1 \leq 2^{k+1}$.

So no matter the last computation gate the induction hypothesis still holds at depth $k+1$ therefore $d_k \leq 2^{k+1}$ and $c_k \leq 2^{\frac{(k+1)^2+3(k+1)}{2}}$. This concludes the proof.

5.5 Problems not in $\mathsf{NC}^{\mathsf{alg}}$

In this section using theorem 5.3 we will present different problems not in $\mathsf{NC}^{\mathsf{alg}}$, including the maxflow problem. As a warm up we will begin by showing that the mod 2 function (or bit extraction as Mulmuley calls it in his paper) cannot be computed by algebraic circuits of small depth. But first let us properly define what we mean by computing a function.

In the beginning of this paper we have proved that $\mathsf{NC}^{\mathsf{alg}}$ is at least as expressive as iPRAM when only considering integer inputs. Therefore given a family of functions f_n and a family of intervals D_n such that $f_n : D_n \mapsto \mathbb{R}$ there are two ways to meaningfully analyze its computability in $\mathsf{NC}^{\mathsf{alg}}$. The first one is the obvious one: is there a sequence of circuits $C_n \in \mathsf{NC}^{\mathsf{alg}}$ such that $\forall \lambda \in D_n, C_n(\lambda) = f_n(\lambda)$. The second one focuses only on integers inputs: is there a sequence of circuits $C_n \in \mathsf{NC}^{\mathsf{alg}}$ such that $\forall k \in \mathbb{N} \bigcap D_n, C_n(k) = f_n(k)$. If a sequence of function cannot be computed on the integers by circuits of $\mathsf{NC}^{\mathsf{alg}}$ then it cannot be computed by iPRAM and it surely cannot be computed on the whole of D_n. When tackling problems we will often consider both version of computability.

Here is the general scheme for proving that a decision problem (the function returns either 0 or 1 depending on membership in a specified language) is not computed by an algebraic circuit of low depth. Take a subset L of $\mathbb{R}^n$ which will be the decision problem and restrict your view to $[0, 2^m]^n$, n is to be thought as the number of inputs and m as the bitlength of the integers we work with. Often in problems we only consider n and take m to be polynomial in n, it makes sense as in the bit world we may in polynomial time, only work with at most polynomial bit length integers. If L is complex enough the hope is that there will a lot of alternations inside the hypercube $[0, 2^m]^n$ between areas belonging to L and areas not belonging to L. For instance if $n = 1$ (which is the only case studied by theorem 5.3), L is complex if $[0, 2^m] \bigcap L$ is equal to the union of exponentially many disjoint intervals (exponential in m) [6] . Indeed in that case L cannot be recognized (meaning returning 1 when the output is in L, 0 otherwise) by an algebraic circuit of depth polylogarithmic in m because those cannot compute piece-wise polynomial fraction with exponentially many pieces. When n is not equal to 1 we need not worry, one must simply find an integer c and a one dimensional curve $f : [0; 2^{m^c}] \mapsto [0; 2^m]^n$ called parametrization, such that as λ ranges across $[0; 2^{m^c}]$, f goes through many alternations of L and L^C (the complementary of L), and such that f is expressible using a circuit of $\mathsf{NC}^{\mathsf{alg}}$. Then no circuit of $\mathsf{NC}^{\mathsf{alg}}$ may

---

[6] actually any interval is the union of exponentially many disjoint ones, so more precisely we should require that L is not equal to the union of less than exponentially many intervals

recognize L, otherwise a circuit for L composed with a circuit for f would yield a piece-wise polynomial function with exponentially many pieces. In our proof for maxflow we use a linear parametrization, but in general it does not need be linear. This parametrization trick may not work in some instances, meaning there probably exists problems over $\mathbb{R}^n$ not in $\mathsf{NC}^{\mathsf{alg}}$ but such that no parametrization goes through exponentially many alternation of L and L^C. For instance, in the decision version of the maxflow problem we fail to come up with a proper parametrization to show that it does not belong to $\mathsf{NC}^{\mathsf{alg}}$. Is it because such parametrization does not exist or because we could not find it ? We do not know.

5.5.1 Bit extraction

We start with proving that the mod 2 function cannot be computed by a circuit of small depth. Although the mod 2 function can be defined on $\mathbb{R}$ we only look at the mod 2 function on $[\![0;2^m]\!](=[0;2^m]\cap\mathbb{N})$ and we prove that on integer inputs no circuit of depth $\mathsf{polylog}(m)$ may always agree with the mod 2 function.

The next lemma states that no piece-wise polynomial fraction with few pieces or low augmented degree can agree with the mod 2 function on many consecutive integers.

Lemma 5.1:

Let f be a piece-wise polynomial fraction over $\mathbb{R}$, if $\forall k \in [\![0;N]\!], f(k) = k\%2$ then $d(f) * c(f) \geq \frac{N}{10}$

Proof

Let f be a piece-wise polynomial fraction as described in the theorem and call c its number of pieces, $I_1, \dots, I_c$ its pieces. Assume c is less than $N/10$ (otherwise the theorem is proved), there must be a piece I_j containing at least N/c integers of $[\![0;N]\!]$. We remind that over I_j f is a polynomial fraction which we note $\frac{P}{Q}$, since $k \mapsto k\%2$ has $N/2c$ zeroes over I_j, so must P, P also cannot be the zero function because $k \mapsto k\%2$ is equal to 1 for some integers in I_j, therefore P is of degree at least $N/2c$. Therefore $d(f) * c \geq (N/2c) * c \geq N/10$.

The next theorem states that no circuit of $\mathsf{polylog}(n)$ depth may compute the mod 2 function (we only consider the first 2^n integer inputs).

Theorem 5.4:

Let c and d be two positive real numbers, for any function family f_n computed by a circuit family of depth $c\log^d(n)$, there exists $N \in \mathbb{N}$ such that $\forall n > N, \exists k \in [\![0;2^n]\!], f_n(k) \neq k\%2$.

Proof

Let $(C_n)_n$ be a sequence of circuits of depth $k_n = O(c\log^d(n))$. Since $n \mapsto n\%2$ is a function with one input, we may consider $(C_n)_n$ to be a sequence of circuits with one input. For any n, the function computed by C_n is a piece-wise polynomial fraction f. By theorem 5.3 we have that $c(f) \leq 2^{\frac{k_n^2+3k_n}{2}}$ and $d(f) \leq 2^{k_n}$ therefore $c(f)d(f) = o(2^n)$. Therefore for large enough n, using Lemma 5.1, f cannot coincide with $n \mapsto n\%2$ for all integers in $[\![0;2^n]\!]$.

It was already known to Mulmuley that the problem of computing the last bit was a hard problem for $\mathsf{NC}^{\mathsf{alg}}$, from our understanding and if we recap his views informally : he dismisses it because there is no polynomial-depth circuit for this problem either[7], so it is not so interesting. Indeed he considers that the size of this problem is 1 : we are given 1 number and are asked to return its last bit. Ìn theorem 5.5 prove that a very similar set has no circuits in $\mathsf{NC}^{\mathsf{alg}}$ yet it has polynomial-depth circuits.

Theorem 5.5:

The set

$$\{(x_1,\dots,x_n) \in \mathbb{N}^n ; x_n \% 2 = 0 \wedge x_n < 2^n\}.$$

has polynomially bounded arithmetic circuits which agree with it over $[[0;2^n]]$, but none in $\mathsf{NC}^{\mathsf{alg}}$.

Proof

Let us call A the set of the theorem. One can check that x_n is less than 2^n and if so retrieve with a circuit of size linear in n the last bit of x_n (using standard dichotomic search). So set A has polynomially bounded circuits.
On the other hand setting all x_i but x_n to 0 we already know that the set

$$\{(0,\dots,0,x_n) \in \mathbb{N}^n ; x_n \% 2 = 0 \wedge x_n < 2^n\}.$$

is not in $\mathsf{NC}^{\mathsf{alg}}$ by a proof similar to the one of theorem 5.4. Therefore the set A is not in $\mathsf{NC}^{\mathsf{alg}}$ either.

Now we would like to take a moment to ponder over what the last theorem means with regards to the algebraic approach. When presented with Mulmuley's original result we were told this three-part story: One wants to prove that $P \neq \mathsf{NC}$. Maxflow is P-complete, so we just need to prove that Maxflow is not in NC. It is too hard, but fortunately we can prove that the algebraic version of Maxflow is not in the algebraic version of NC. Surely this has drawn us closer to proving $P \neq \mathsf{NC}$? Well it seems to us that the last theorem dampens that assumption, we have an example of a trivial problem for bit complexity classes which is not in $\mathsf{NC}^{\mathsf{alg}}$ (yet has polynomial size circuits !). So then arises the question why bother with maxflow when trivial problems already are not in $\mathsf{NC}^{\mathsf{alg}}$? In our opinion, this remark is valid in the context of assessing the computational power of $\mathsf{NC}^{\mathsf{alg}}$ and thus the extent to which impossibility results are meaningful. However there are still good reasons to care about the maxflow problem : first of all, even though maxflow is a P-complete problem in the bit world, there is no obvious guarantee (at least to us) that maxflow is complete in the algebraic world, so a proof that a trivial problem is not in $\mathsf{NC}^{\mathsf{alg}}$ could not entail anything regarding maxflow, secondly it is interesting in its own right to know if maxflow can be solved fast in parallel using only algebraic operations.

Taking the results to the bit world A direct way to get $\mathsf{NC} \neq \mathsf{P}$ would be to add gates in our model which can extract the bits of our inputs. Of course by adding such gates the whole advantage of the algebraic approach is lost: reasoning over boolean functions is notoriously hard. As a naive approach we propose the following: studying algebraic circuits endowed with sin gates (as in the sinus function). The justification is as follows : sin can be seen as a continuous version of the mod 2 function with period 2π. By dividing the input with appropriate multiples of $\frac{1}{\pi}$ we could in this way retrieve bits from our algebraic input, thus

---

[7] we note : unless the bitlength is part of the size

establishing bounds for NC^{alg} sin would yield bounds in the bit world. We can then restrict ourselves to the study of the set of functions polynomial in sin closed under application of sin. The hope is that because we deal with trigonometric functions and basic algebra we can leverage the trigonometric equations ($\sin a \sin b = 2 \sin \frac{a+b}{2} \cos \frac{a-b}{2}$ ...) to have a tractable analysis.

Question

Can we add gates to our models (e.g. the sinus function) and extend our results to the bit world ?

5.5.2 Bijections from $\mathbb{N}$ to $\mathbb{N}^2$

Before going on to the maxflow problem we focus on the problem of computing a bijection between $\mathbb{N}$ and $\mathbb{N}^2$. If our model could do this we could reduce problems over n inputs to problems over 1 input by repeatedly applying the bijection (the same way in the bit world problems over $\mathbb{N}^2$ might as well be problems over $\mathbb{N}$ by associating a number to each pair of number). We do not directly prove that no bijection exists however we prove that we cannot inverse the usual Cantor pairing function. Using similar arguments one may be able to prove that no bijection between $\mathbb{N}$ and $\mathbb{N}^2$ may be computed by a circuit of low depth.

Theorem 5.6:

Let $f : \mathbb{N} \times \mathbb{N} \mapsto \mathbb{N}$ be the usual Cantor pairing function $f(n, m) = \frac{1}{2}(m+n)(m+n+1)+m$. This function is bijective and its reciprocal f^{-1} cannot be computed over $[\![0; 2^n]\!]$ by an algebraic circuit of depth polylogarithmic in n .

Proof

Consider the function $g(x, y) = (x < y)$. (it returns either 0 or 1) The function $g(f^{-1})$ has $2^{\frac{n}{2}}$ alternations between 0 and 1 over $[\![0; 2^n]\!]$ therefore it cannot be computed by a circuit of $\mathsf{NC}^{\mathsf{alg}}$ (by arguments used to prove theorem 5.1 and 5.4) . But g can be computed by a circuit of $\mathsf{NC}^{\mathsf{alg}}$ ($<$ is a gate of $\mathsf{NC}^{\mathsf{alg}}$) therefore it must be f^{-1} which cannot be computed by a circuit of $\mathsf{NC}^{\mathsf{alg}}$.

5.5.3 Maxflow

Now we focus our attention to the maxflow problem. In the maxflow problem we are given a network (an directed graph) of edges with specified capacities, a threshold k, a two specified nodes s and t, the question then asked is can a flow of size k flow from s to t.

Definition 5.15: Maxflow problem

An instance of an **algebraic maxflow problem** of size n^2 is a directed graph with n nodes with each edge (as much as n^2) labelled with a real number called capacity. It is represented by its adjacency matrix representation. We say that a circuit A solves the **algebraic maxflow problem** on size n^2 if on any instance of size n^2 it returns the maximal flow from node 1 to node n. We say that a circuit A solves the **integer maxflow problem** on size n^2 if on any instance containing only integers it returns the maximal flow from node 1 to node n

Definition 5.16: Decision maxflow problem

An instance of a **decision maxflow problem** of size n^2 is an undirected graph with n nodes with each edge (as much as n^2) labelled with a real number called capacity and a threshold k. The graph is represented by its adjacency matrix representation. We say that a circuit A solves the **decision maxlow problem** on size n^2 if on any instance of size n^2 it returns 1 if the maximal flow from node 1 to node n is greater than k, 0 otherwise. We say that a circuit A solves the **integer decision maxflow problem** on size n^2 if on any instance containing only integers it returns whether the maximal flow from node 1 to node n is greater than k.

This gives us 4 variants of the maxflow problem. We can of course further limit the complexity of these problem by only requiring algorithms solving them to work on an interval D_n instead of the whole of $\mathbb{R}$. In his paper [Mul99] Mulmuley proves that all 4 cannot be solved by circuits of iPRAM even if we only consider polynomial size bit lengths integer, i.e $D_n = [0, 2^{n^c}]$. We will now give a simple proof that the algebraic maxflow problem cannot be solved by circuits of $\mathsf{NC}^{\mathsf{alg}}$. We introduce for that the linear parametrization version of the maxflow problem. For a given size n^2, each capacity c is now an affine function of parameter λ where the coefficients a and b are integers of bitlength polynomial in n ($c = a + b * \lambda$). Given $p, q \in \mathbb{Z}^2$ this defines for all $\lambda \in [p, q]$ an instance $G(\lambda)$ of the maxflow problem. Let I be the function such that $I(G(\lambda))$ returns the maximal flow from node 1 to node n. It has been proven in [Car83] and [Mul99] that there is a family $G_n(\lambda)$ of linearly parameterized graph with polynomial-bit-length parameters and $O(n^2)$ edges, such that $I(G_n(\lambda))$ has $O(2^n)$ breakpoints (slopes changes) over $[0, T]$ with $T = 2^{n+1}$.

Theorem 5.7:

The algebraic maxflow problem cannot be computed by circuits of $\mathsf{NC}^{\mathsf{alg}}$

Proof

Let us suppose there is a family of circuits $C_n \in \mathsf{NC}^{\mathsf{alg}}$ such that C_n computes the algebraic maxflow problem for size n^2 graphs. Then C_n must return the correct value on $G_n(\lambda)$ for all λ. Consider B_n a circuit of depth 1 and $O(n^2)$ largeness taking λ as input and outputting $G_n(\lambda)$ (represented via its adjacency matrix). Then the circuit $C_n(B_n(\lambda))$ is a circuit of $\mathsf{NC}^{\mathsf{alg}}$ with one input computing $I(G_n(\lambda))$ which is a linear function with 2^n breakpoints, but such a function cannot be computed by a circuit of $\mathsf{NC}^{\mathsf{alg}}$ by theorem 5.3, contradiction.

Theorem 5.8:

The integer algebraic maxflow problem cannot be computed by circuits of $\mathsf{NC}^{\mathsf{alg}}$

Proof

We sketch an idea of proof only. Consider the construction of Mulmuley, notice that the breakpoints of $I(G_n(\lambda))$ are more than distance 2 away from each other, scale everything up by a factor of 2^n (one may create the integer 2^n in $log(n)$ depth circuits using repeated squaring) so that now the breakpoints are 2^n away from each other, this gives you a function $I'(\lambda)$ which is piece wise linear with each piece containing

2^n integers, argue that any piece-wise polynomial fraction which agrees with I' must have a large number of pieces or a large augmented degree (like in Lemma 5.1), then conclude.

The last remaining piece remaining is the decision version, but we come across a slight hiccup: solvability of the decision version of a problem means being able to compute its epigraph, indeed the decision version asks us to compute $(x, y) \mapsto (y < f(x))$. Intuitively speaking it would be very strange if we could in our algebraic setting compute the epigraph of function but not its graph. The solvability of the decision version should entail the solvability of the search version (up to a given precision) by way of dichotomic search: if we can compute $(x, y) \mapsto (y < f(x))$ then we can approximately compute $x \mapsto f(x)$ in a similar time (for instance the time is multiplied by $\log(f(x))$ if we want to be precise up to the closest integer). Unfortunately in $\mathsf{NC}^{\mathsf{alg}}$ for functions outputting values exponential in n, we cannot reduce the complexity of the decision version to that of the search version via dichotomic search (we would need a depth $\log(2^n) = n$ but we only have $\mathsf{polylog}(n)$ depth). Looking closer at the construction of Mulmuley in [Mul99], the function $I(G_n(\lambda))$ does output for most $\lambda \in [0, 2^n]$ values bigger than $O(2^n)$.

We now give leads to tackle what looks to us to be only a technical issue. The most promising idea would be to find a family of function f_n computable with low depth circuits and close enough to $I(G_n(\lambda))$, i.e. such that the difference $f_n(\lambda) - I(G_n(\lambda)) = O(n^d)$ for many inputs, we could then use dichotomic search to find the value of $I(G_n(\lambda))$. This might actually be possible because looking closer at Mulmuley's construction $I(G_n(\lambda))$ vaguely grows in $\lambda^{\log(n)}$ which is a polynomial in λ (we can hardcode $\log(n)$ for every n), so an algebraic circuit of low depth may compute it. The technical part is properly analyzing the growth rate of the function $I(G_n(\lambda))$.

In case this fails another idea would be to fiddle around a bit with Mulmuley's construction to directly deal with the decision version. Yet another approach would be to analyse circuits of $\mathsf{NC}^{\mathsf{alg}}$ with 2 inputs but it would certainly complicate the mathematics. Finally there is hope that for non pathological functions f we may prove in general that uncomputability of f entails the uncomputability of $(y, x) \mapsto y < f(x)$.

Question

Can we prove that the decision versions of the maxflow problem cannot be decided by circuits of $\mathsf{NC}^{\mathsf{alg}}$ using our techniques ?

Bibliography

[AB06] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2006. ISBN: 978-0-521-42426-4. URL: `https://theory.cs.princeton.edu/complexity/book.pdf`.

[AD79] J. Auslander and Y. N. Dowker. "On disjointness of dynamical systems". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 85.3 (1979), pp. 477–491.

[Aga68a] V. N. Agafonov. "Normal Sequences and Finite Automata". In: *Reports of the Academy of Sciences of the USSR* 179.2 (1968), pp. 255–256.

[Aga68b] V. N. Agafonov. "Normal sequences and finite automata". English. In: *Sov. Math., Dokl.* 9 (1968). Originally published in Russian [Aga68a], pp. 324–325. ISSN: 0197-6788.

[ALW11] Atif Abueida, James Lefevre, and Mary Waterhouse. "Equitable edge colored Steiner triple systems". In: *The Australasian Journal of Combinatorics [electronic only]* 50 (June 2011).

[AM15] Dylan Airey and Bill Mance. "Normality preserving operations for Cantor series expansions and associated fractals, I". In: *Illinois J. Math.* 59.3 (2015), pp. 531–543.

[Bat83] N. G. Bate. "Complete graphs without polychromatic circuits". In: *Discret. Math.* 46.1 (1983), pp. 1–8. DOI: `10.1016/0012-365X(83)90263-7`. URL: `https://doi.org/10.1016/0012-365X(83)90263-7`.

[BCH15] Veronica Becher, Olivia Carton, and Pablo Ariel Heiber. "Normality and automata". In: *Journal of Computer and System Sciences* 81.8 (2015), pp. 1592–1613.

[Bil95] Patrick Billingsley. *Probability and Measure*. Wiley, 1995.

[BL92] A. Broglio and P. Liardet. "Predictions with automata. Symbolic dynamics and its applications". In: *Contemporary Mathematics* 135 (1992). Also appeared in Proceedings of the AMS Conference in honor of R. L. Adler. New Haven CT - USA 1991., pp. 111–124.

[Bor09] Émile Borel. "Les probabilités dénombrables et leurs applications arithmétiques". In: *Rend. Circ. Matem. Palermo* 27 (1909), pp. 247–271.

[Bug12] Yann Bugeaud. *Distribution modulo one and Diophantine approximation*. Cambridge Tracts in Mathematics. Cambridge University Press, 2012.

[Car20a] Olivier Carton. "A direct proof of Agafonov's theorem and an extension to shift of finite type". In: *CoRR* abs/2005.00255 (2020). arXiv: `2005.00255`. URL: `https://arxiv.org/abs/2005.00255`.

[Car20b] Olivier Carton. "A direct proof of Agafonov's theorem and an extension to shifts of finite type". In: *Preprint* (2020).

[Car83] Patricia June Carstensen. "The Complexity of Some Problems in Parametric Linear and Combinatorial Programming". AAI8314249. PhD thesis. USA: University of Michigan, 1983.

[Cha33] D. G. Champernowne. "The Construction of Decimals Normal in the Scale of Ten". In: *Journal of the London Mathematical Society* s1-8.4 (1933), pp. 254–260.

[Che+23] Lijie Chen et al. "Polynomial-Time Pseudodeterministic Construction of Primes". In: *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*. IEEE, 2023, pp. 1261–1270. DOI: `10.1109/FOCS57990.2023.00074`. URL: `https://doi.org/10.1109/FOCS57990.2023.00074`.

[CV20] Olivier Carton and Joseph Vandehey. "Preservation of Normality by Non-Oblivious Group Selection". In: *Theory of Computing Systems* (2020).

[Gaj+23] Karthik Gajulapalli et al. *Range Avoidance for Constant-Depth Circuits: Hardness and Algorithms*. 2023. arXiv: `2303.05044 [cs.CC]`. URL: `https://arxiv.org/abs/2303.05044`.

[GSS82] Leslie M. Goldschlager, Ralph A. Shaw, and John Staples. "The maximum flow problem is log space complete for P". In: *Theoretical Computer Science* 21.1 (1982), pp. 105–111. ISSN: 0304-3975. DOI: `https://doi.org/10.1016/0304-3975(82)90092-5`. URL: `https://www.sciencedirect.com/science/article/pii/0304397582900925`.

[Hir+23] Shuichi Hirahara et al. "A Duality between One-Way Functions and Average-Case Symmetry of Information". In: *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*. Ed. by Barna Saha and Rocco A. Servedio. ACM, 2023, pp. 1039–1050. DOI: `10.1145/3564246.3585138`. URL: `https://doi.org/10.1145/3564246.3585138`.

[Hir+24] Shuichi Hirahara et al. "Exact Search-To-Decision Reductions for Time-Bounded Kolmogorov Complexity". In: *39th Computational Complexity Conference (CCC 2024)*. Ed. by Rahul Santhanam. Vol. 300. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 29:1–29:56. ISBN: 978-3-95977-331-7. DOI: `10.4230/LIPIcs.CCC.2024.29`. URL: `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CCC.2024.29`.

[Hir18] Shuichi Hirahara. "Non-Black-Box Worst-Case to Average-Case Reductions within NP". In: *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*. 2018, pp. 247–258. DOI: `10.1109/FOCS.2018.00032`.

[JaJ11] Joseph F. JaJa. "PRAM (Parallel Random Access Machines)". In: *Encyclopedia of Parallel Computing*. Ed. by David Padua. Boston, MA: Springer US, 2011, pp. 1608–1615. ISBN: 978-0-387-09766-4. DOI: `10.1007/978-0-387-09766-4_23`. URL: `https://doi.org/10.1007/978-0-387-09766-4_23`.

[Kal02] Olav Kallenberg. *Foundations of Modern Probability*. Springer, 2002.

[KS21] Alexander Kozachinskiy and Alexander Shen. "Automatic Kolmogorov complexity, normality, and finite-state dimension revisited". In: *Journal of Computer and System Sciences* 118 (2021), pp. 75–107. ISSN: 0022-0000. DOI: `https://doi.org/10.1016/j.jcss.2020.12.003`. URL: `https://www.sciencedirect.com/science/article/pii/S0022000020301215`.

[KW75] T. Kamae and B. Weiss. “Normal numbers and selection rules”. In: *Israel Journal of Mathematics* (1975), pp. 101–110.

[Léc23] Ulysse Léchine. “Revisiting Mulmuley: Simple Proof That Maxflow Is Not in the Algebraic Version of NC”. In: *43rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2023)*. Ed. by Patricia Bouyer and Srikanth Srinivasan. Vol. 284. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 33:1–33:12. ISBN: 978-3-95977-304-1. DOI: `10.4230/LIPIcs.FSTTCS.2023.33`. URL: `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.FSTTCS.2023.33`.

[LO22] Zhenjian Lu and Igor C. Oliveira. “Theory and Applications of Probabilistic Kolmogorov Complexity”. In: *Bull. EATCS* 137 (2022). URL: `http://bulletin.eatcs.org/index.php/beatcs/article/view/700`.

[Lon86] Luc Longpré. “Resource Bounded Kolmogorov Complexity, A Link between Computational Complexity & Information Theory”. PhD thesis. Cornell University, USA, 1986.

[LP20] Yanyi Liu and Rafael Pass. *On One-way Functions and Kolmogorov Complexity*. 2020. arXiv: `2009.11514 [cs.CC]`.

[LSS24] Ulysse Léchine, Thomas Seiller, and Jakob Grue Simonsen. “Agafonov’s theorem for probabilistic selectors”. working paper or preprint. Apr. 2024. URL: `https://hal.science/hal-04563952`.

[LV19] Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications, 4th Edition*. Texts in Computer Science. Springer, 2019. ISBN: 978-3-030-11297-4. DOI: `10.1007/978-3-030-11298-1`. URL: `https://doi.org/10.1007/978-3-030-11298-1`.

[MR06] Wolfgang Merkle and Jan Reimann. “Selection Functions that Do Not Preserve Normality”. In: *Theory Comput. Syst.* 39.5 (2006), pp. 685–697.

[Mul99] Ketan Mulmuley. “Lower Bounds in a Parallel Model without Bit Operations”. In: *SIAM Journal on Computing* 28.4 (1999), pp. 1460–1509. DOI: `10.1137/S0097539794282` eprint: `https://doi.org/10.1137/S0097539794282930`. URL: `https://doi.org/10.1137/S0097539794282930`.

[NZ51] Ivan Niven and H. S. Zuckerman. “On the definition of normal numbers.” In: *Pacific J. Math.* 1.1 (1951), pp. 103–109.

[Pos61a] L. P. Postnikova. “On the Relationship between the Notions of von Mises–Church Collectives and Bernoulli Normal Sequences of Signs”. In: *Theory of Probability and its Applications* 6.2 (1961), pp. 232–234.

[Pos61b] LP Postnikova. “On the connection between the concepts of collectives of Mises-Church and normal Bernoulli sequences of symbols”. In: *Theory of Probability & Its Applications* 6.2 (1961). translation of [Pos61a] by Eizo Nishiura, pp. 211–213.

[Rab63] Michael O. Rabin. “Probabilistic automata”. In: *Information and Control* 6.3 (1963), pp. 230–245.

[Rei90] Rüdiger Reischuk. *Einführung in die Komplexitätstheorie*. Leitfäden und Monographien der Informatik. Stuttgart: Teubner, 1990, pp. XVII, 412. ISBN: 3-519-02275-3.

[RSW22] Hanlin Ren, Rahul Santhanam, and Zhikun Wang. "On the Range Avoidance Problem for Circuits". In: *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*. 2022, pp. 640–650. DOI: `10.1109/FOCS54457.2022.00067`.

[SPL22] Thomas Seiller, Luc Pellissier, and Ulysse Léchine. "Unifying lower bounds for algebraic machines, semantically". quoicoubeh. Nov. 2022. URL: `https://hal.science/hal-01921942`.

[SS72] Claus-Peter Schnorr and H. Stimm. "Endliche Automaten und Zufallsfolgen". In: *Acta Informatica* 1 (1972), pp. 345–359.

[SW14] Rahul Santhanam and Ryan Williams. "On Uniformity and Circuit Lower Bounds". In: *computational complexity* 23.2 (2014), pp. 177–205. DOI: `10.1007/s00037-014-0087-y`. URL: `https://doi.org/10.1007/s00037-014-0087-y`.

[VW23] Nikhil Vyas and Ryan Williams. "On oracles and algorithmic methods for proving lower bounds". In: *14th Innovations in Theoretical Computer Science Conference (ITCS 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik. 2023.

[WK19] Xiaowen Wang and Teturo Kamae. "Selection rules preserving normality". In: *Israel Journal of Mathematics* 232 (2019), pp. 427–442.

Appendix A

Reasonable functions

In this appendix we study reasonable functions for the purpose of use in part II. We first recall the definition of reasonable functions as stated in chapter 4, the notion of reasonability is meant to capture functions which to infinity not too slowly. The ultimate goal of this chapter would be to establish that the composition of two reasonable functions is a reasonable function. We leave it as an open question for now.

Definition A.1: Very-time-computable functions

Let f be a function from $\mathbb{N} \mapsto \mathbb{N}$, f is very-time-computable (VTC) if there exists a machine M in $\mathsf{DTIME}(f(n)+n)$ such that $\forall n, M(n) = f(n)$.

Definition A.2: d-reasonable function

Let $d \in \mathbb{N}$, and $f : \mathbb{N} \mapsto \mathbb{N}$ be an increasing function, it is d-reasonable if $\forall c, \exists_\infty n, cK^{dn}(n) \leq f(n)$

This is not the same definition as in chapter 4, but it is equivalent to it by theorem A.1.

Definition A.3: Reasonable functions

A function $f : \mathbb{N} \mapsto \mathbb{N}$ is reasonable if there exist $d \in \mathbb{N}$ such that f is d-reasonable.

One can notice in the definition of reasonable functions that there is an exponential discrepancy between the size of n which is $\log(n)$ and the running time which is dn. This discrepancy was all we needed in part 4 but may seem a bit arbitrary in general, in the following definition we define a more natural notion, this notion may be more deserving of the epithet "reasonable". In any case theorem A.1 states that all of these notions are actually equivalent.

Definition A.4: Ultra-reasonable functions

Let $f \in \mathbb{N} \mapsto \mathbb{N}$, f is said to be is ultra-reasonable if f is increasing and such that $\forall \epsilon \in \mathbb{R}^+, \forall c, \exists_\infty n, cK^{(1+\epsilon)\log(n)}(n) \leq f(n)$

Fact A.1:

Any ultra-reasonable function is 1-reasonable

Definition A.5: Iterated logarithm $\log^{(k)}(n)$

Let $k \in \mathbb{N}$. The k-iterated logarithm, denoted by $\log^{(k)}(n) : \mathbb{N} \mapsto \mathbb{N}$, is defined recursively as follows:

- $\log^{(1)}(n) = \lfloor \log(n) \rfloor$, where $\log(n)$ is the logarithm of n in base 2.
- For $k > 1$, $\log^{(k)}(n) = \lfloor \log(\log^{(k-1)}(n)) \rfloor$.

Next we define the arrow operation in a sightly unusual way

Definition A.6: $\uparrow_k$ (Arrow Operation)

For natural numbers a and k, $a \uparrow_k$: $\mathbb{N} \mapsto$ is defined recursively as:

$$a \uparrow_1 (n) = a^n,$$

$$a \uparrow_{k+1} (n) = a^{(a \uparrow_k (n))}.$$

Noted differently:

$$a \uparrow_k (n) = \underbrace{a^{a^{\cdot^{\cdot^{\cdot^{a^n}}}}}}_{k \text{ times}}.$$

Fact A.2: $2 \uparrow_k$ is easily computable

For any $k \in \mathbb{N}$, $n \mapsto 2 \uparrow_k (n)$ can be computed in less than $1.1 \cdot 2 \uparrow_{k-1} (n) steps$

Proof

This is entailed by the fact that for $k = 1$, $n \mapsto 2^n$ is computed in time $n + O(\log(n))$ (reminder that the bitsize of 2^n is n).

Lemma A.1:

Let f be d-reasonable, then $f(\log\log)$ is ultrareasonable.

Proof

Let $c \in \mathbb{N}$, Take n such that $\#^d n$ is a description of n of size $f(n)/c$ decodable in dn steps, infinitely many such n must exist by reasonability of f. Consider program p which takes as input $\#^d n$, decodes it, and outputs 2^{2^n}. Note that program p runs in time $dn + 2 \cdot 2^n$ (by fact A.2) and the size of p is $\leq \frac{f(n)}{2c} + O(1) \leq \frac{f(n)}{c}$ for n big enough. We thus have that:

$$\exists_\infty n,\ K^{dn+2^n}(2^{2^n}) \leq \frac{f(n)}{c},$$

$$\Rightarrow \exists_\infty m,\ K^{d \log\log m + \log m}(m) \leq \frac{f(\log\log(m))}{c}$$

Hence $f(\log\log)$ is ultrareasonable.

Corollary A.1:

Let f be reasonable, $k \in \mathbb{N}$ then $f(\log^{(k)})$ is reasonable.

Fact A.3:

Let f, g be two functions such that $f \leq g$, if f is reasonable then g is reasonable.

As a direct corollary of fact A.3 and lemma A.1 we have that :

Theorem A.1: Equivalence between d-reasonability and ultrareasonability

Let f be a reasonable function then it is 1-reasonable and ultrareasonable.

Lemma A.2:

Let f be reasonable, let $h : \mathbb{N} \mapsto \mathbb{N}$ be the function $h(n) \overset{\text{def}}{=} \min(f(n), n)$, then h is reasonable

Proof

We have two possibilities:
Either we have $\forall_\infty n, h(n) = f(n)$ and then h is reasonable.
Or $\exists_\infty n, h(n) = n$, but for all of these n we have a program p of size $\log(n)$ which prints n in $\log(n) + O(1)$ steps thus $\exists_\infty n, K^{2\log(n)}(n) \leq h(n)$

Theorem A.2:

Let f be d-reasonable, $k \in \mathbb{N}$, let $h : \mathbb{N} \mapsto \mathbb{N}$ be the function $h(n) \overset{\text{def}}{=} \min(f(n), \log^{(k)}(n))$, then h is reasonable.

Proof

$$\begin{aligned}
& f \text{ reasonable} \\
& \Rightarrow \min(f, n \mapsto n) \text{ reasonable} \\
\overset{\boxed{\text{corollary A.1}}}{\Rightarrow} & (\min(f, n \mapsto n)) \circ \log^{(k)} \text{ reasonable} \\
\overset{\boxed{\text{log is increasing}}}{\Rightarrow} & \min\left(f \circ \log^{(k)}, \log^{(k)}\right) \text{ reasonable} \\
\overset{\boxed{\text{fact A.3}}}{\Rightarrow} & \min(f, \log^{(k)}) \text{ reasonable}
\end{aligned}$$

Definition A.7: Inverse of a function in $\mathbb{N} \mapsto \mathbb{N}$

Let $f \in \mathbb{N} \mapsto \mathbb{N}$, its inverse noted f^{-1} is defined as $f^{-1}(n) = \min\{m; f(m) \geq n\}$

The following theorem is used liberally in chapter 4

Theorem A.3:

If f is VTC and increasing and going to infinity then f is ultrareasonable

Proof

We define first define program p which is meant to compute f^{-1}.

Algorithm 3 Program $p(n)$ to compute $f^{-1}(n)$

```
Input: $n$
Initialize: $i \leftarrow 0$
while True do
    if $f(i) \geq n$ then
        return $i$
    else
        $i \leftarrow i+1$
    end if
end while
```

Given that $h(n) \overset{\text{def}}{=} \min(f(n), \log(n))$ is VTC and less than f, h being reasonable entails reasonability of f, we may thus WLOG consider in the following that $f \leq \log(n)$. It's routine to check that p computes f^{-1}. The running time of p is $\leq \log(n) + \Sigma_{i=0}^{f^{-1}(n)} f(i) + i \leq 2(f^{-1}(n))^2$ ($\boxed{\text{we read } n \text{ and then execute the program, moreover } f^{-1}(n) \geq n}$). The function $n \mapsto 2^{f^{-1}(n)}$ is computed in less than $3(f^{-1}(n))^2 + \log(n)$ steps which is less than $2^{f^{-1}(n)}$ steps hence:

$$\begin{aligned} &\forall n, K^{2^{f^{-1}(n)}}(2^{f^{-1}(n)}) \leq n \\ &\boxed{\text{Using the fact that } f \text{ grows to infinity}} \\ &\Rightarrow \exists_\infty n, K^n(n) \leq f(\log(n)) \leq f(n) \end{aligned}$$

We leave the two following questions unanswered. We believe these questions to be tractable.

Question

- If f is reasonable then its inverse f^{-1} is reasonable ?
- Let f and g be two reasonable functions then $f \circ g$ is reasonable ?