Univerzita Karlova v Praze

Matematicko – Fyzikální Fakulta

Propositional Proof Complexity and Rewriting

Stefano Cavagnetto

Doktorská Disertační Práce
2008

Školitel: Prof. RNDr. Jan Krajíček, DrSc.

Obor M1 – Algebra, teorie čísel a matematická logika

Charles University in Prague

Faculty of Mathematics and Physics

Propositional Proof Complexity and Rewriting

Stefano Cavagnetto

Doctoral Dissertation
2008

Thesis Advisor: Prof. RNDr. Jan Krajíček, DrSc.

Branch M1 – Algebra, Number Theory and Mathematical Logic

Abstract

In this work we want to find a new framework for propositional proofs (and in particular for resolution proofs) utilizing rewriting techniques. We interpret the well-known propositional proof system resolution using string rewriting systems (semi-Thue system [70], [71]) Σ_n^* and Σ_n corresponding to tree-like proofs and sequence-like proofs, respectively. We prove that the system Σ_n^* is complete and sound with respect to tree-like Resolution R^* (and we show how it is possible to obtain the same result for R). Using this interpretation we give a representation of Σ_n^* using planar diagrams in van Kampen style. In this representation we show how the classical complexity measures for Resolution (size, width and space) can be interpreted.

Subsequently, we consider rewriting in a synchronous, parallel fashion as it is used in the theory of cellular automata. In this respect, we give a new proof of Richardson theorem [63](a global function $G_{\mathbb{A}}$ of a cellular automaton $\mathbb{A}$ is injective if and only if the inverse of $G_{\mathbb{A}}$ is a global function of a cellular automaton), a classical result in the field, exploiting only propositional logic. In particular, we show how compactness of propositional logic and Craig's interpolation theorem suffice in order to prove the theorem. Moreover, we show a way how to construct the inverse cellular automaton using the method of feasible interpolation from [49].

We also solve two problems regarding complexity of cellular automata formulated by Durand [32]. The first problem can be stated as follows: consider finite bounded configurations and a reversible cellular automaton that is given by a "simple" algorithm. Is the inverse automaton given by a "simple" algorithm too? The second problem is the following: the injectivity problem of cellular automata on bounded size is co$\mathcal{NP}$-complete, [32]; does the result still hold if we consider instead of the size of the transition table, the smallest program (circuit) which computes its transition table?

Finally, we present a new proof system based on cellular automata. Most of the results in this work have been written up in articles, see [16] and [17].

Acknowledgements

I am deeply grateful to my supervisor Jan Krajíček for many reasons. First, by him I have been introduced to the interesting field of Propositional Proof Complexity. Second, the variety and the beauty of topics he exposed to me have contributed to shape my knowledge on Mathematical Logic and increase my love for the Mathematics and its history. In fine, he gave me encouragement and I really got a vigorous help and every possible support from him during all these years of study.

I wish to thank also other people that during these last four years contributed to influence my view of relevant mathematics in various ways. I am indebted with Pavel Pudlák for many discussions on mathematics. I thank Neil Thapen for his effort on teaching me a lot of model theory and for being so patient and kind in front of all my questions on Mathematical Logic. I also thank Emil Jeřábek, Radek Honzik and Pavel Hrubeš of the Institute for many discussions over the years. Finally, I want to thank the Institute of Mathematics of the Academy of Sciences of Czech Republic for the environment of its seminars, activities and the financial support during all these years.[1]

I wish to thank and dedicate this work to Maddalena who has followed me to Prague for my doctoral studies in Mathematics and has enthusiastically supported me all this time.

[1]Grants #A1019401, AVOZ10190503, Institute of Mathematics, Academy of Sciences of Czech Republic.

Contents

Preface

In this work we take the basic idea of rewriting as transformation of some object by step by step activity and we embed it in the context of the complexity of propositional proofs. This transformation is obtained by the application of some rewriting rules suitably choosen. We interpret these applications of rewriting rules in sequence as a proof in the classical sense; and this offers some room for a proper mathematical investigation.

In more detail, we want understand how the formalism of rewriting allows us to formulate basic proof systems. We study Resolution and its tree-like version. This simulation by rewriting system is fairly straightforward but requires certain patience with technical details. Exploiting this new formalization we give a representation of the tree-like case by planar diagrams in van Kampen style. As a by-product we have an interpretation of several proof complexity measures such as the space or the width in essentially geometric terms. This in some sense extends to Resolution some geometric interpretations that were known only for the so called group-based proof systems considered in [47].

A second motivation for studying proof systems in terms of rewriting is the hope to gain, using also the diagrammatic interpretation mentioned earlier, some intuition for proof search heuristic. One may expect that a heuristic formulated in terms of strategies for rewriting systems could apply also to more complex rewriting systems that would simulate stronger proof systems than Resolution. Virtually no heuristic for proof search in strong proof systems is known. In particular, we also consider our present work as a first step toward using rewriting systems in proof complexity that operate synchronously in parallel on all symbols of a string (or an array) as for example, cellular automata do. These discrete dynamical systems and models of massively parallel computation [31] are away from the contemporary research in proof complexity and the area is rich of numerous "experimental/heuristic" methods. This is not the miraculous recipe for proving that $TAUT$ is polynomial size, but to enhance our proof-search methods, in particular, in searching for very long proofs. In the second part of this work we introduce cellular automata in the field of proof complexity. We show also that a powerful method such as that of feasible interpolation can be exploited in order to solve problems concerning cellular automata. Thus, it is a fundamental step to build a suitable framework in order to investigate properly their capability in the study of the complexity of proofs. The rewriting approach can give us this unified framework, since one of the basic ways to formalize them is to use tables of local rewriting rules [42]. Moreover, we recall that from the computability point of view Turing Machines and cellular

automata, the latter ones considered on finite configurations, are equivalent, but from the complexity point of view, cellular automata are much more efficient; see [31], Part 3 and [77]. This fact could also have some consequences in proof complexity concerning the way in which we formulate proof systems. At the moment, we do not prove new lower bounds (which are considered the most appreciated results of the field of propositional proof complexity) but we hope the approach proposed here can open a new perspective on the analysis of the complexity of propositional proofs.

The work is organized as follows. The first chapter is a self-contained exposition of some of the most important concepts in complexity theory and propositional proof complexity. Almost all the concepts from these two fields used later on in this work are presented here. As general references the reader can see [45], [59], [46], [26], [65], [57] and [77].

The second chapter deals with the rewriting techniques and it introduces the semi-Thue systems. We define a new semi-Thue system and we prove that this system is complete and sound with respect to tree-like Resolution R^*. Exploiting this new formalization we give a representation of the tree-like case by planar diagrams in van Kampen style. We give also a characterization of all the complexity measures regarding R using planar diagrams. Finally we consider the dag-like case for resolution proofs and we propose an example of formalization of usual proofs using rewriting.

In the third chapter we consider rewriting from a different perspective. Rewriting is not performed sequentially anymore, as it happens for classical semi-Thue system, but in parallel and in a synchronous way. Thus the natural place where to look at is the theory of cellular automata. In this chapter we consider several applications of propositional logic to cellular automata. We give a new proof of one of the classical result about cellular automata, the Richardson Theorem. Our proof exploits the compactness of propositional logic and Craig's interpolation theorem. In the same chapter we show how to use feasible interpolation to find the description of inverse cellular automata. We conclude this chapter by solving two problems about complexity of cellular automata left open in [32].

The last chapter deals with inverse cellular automata as propositional proofs. In this chapter we combine Richardson's theorem with a $co\mathcal{NP}$-completeness result obtained by Durand [32] and we define a new proof system. We show that this new proof system can be thought of also as a propositional proof system in the sense of Cook and Reckhow [25].

Chapter 1

Propositional Proof Complexity

Two fields connected with computers, automated theorem proving on one side and computational complexity theory on the other side, gave the birth to the field of propositional proof complexity in the late '60s and '70s. In this chapter we recall some basics about computational complexity theory and we introduce some fundamental concepts of propositional proof complexity. It is organized as follows: in the next section we recall some of the basic definitions in computational complexity theory; for a self-contained exposition of the field the interested reader can see [65], [57]. In section 2 we introduce some basic definitions from propositional proof complexity and we recall an important result by Cook and Reckhow [24] which gives an interesting link between complexity of propositional proofs and one of the most beautiful open problem in contemporary mathematics (the famous $\mathcal{P}$ versus $\mathcal{NP}$ problem, [26], [58], [66], [76], [67]). There are many survey papers on propositional proof complexity offering different emphasis, see [75], [21] and [59]. The reader interested in connections with bounded arithmetic can see [45]. Section 3 considers one of the most investigated proof system for propositional logic, the proof system Resolution R. Section 4 introduces feasible interpolation. This technique has been applied successfully in several part of the field in proving lower bounds and in order to gain a better understanding of automatizability of proof search. For a greater completness we recall in some detail the proof of feasible interpolation for R, [49]. We conclude the chapter with a section devoted to the idea of "mathematical" proof system; as an example in this section we present the proof system Cutting Planes CP.

1.1 Some technical preliminaries

In 1936 Alan Turing [74] introduced the standard computer model in computability theory, the Turing machine. A Turing machine M consists of a finite state control (a finite program) attached to read/write head which moves on an infinite tape. The tape is divided into squares. Each square is capable of storing one symbol from a finite alphabet Γ. $b \in \Gamma$, where b is the blank symbol. Each machine has a specified input alphabet $\Sigma \subseteq \Gamma$ where $b \notin \Sigma$. M is in some finite state q (in a specified finite set Q of possible states), at each step in a computation. At the beginning a finite input string over Σ is written on adjacent squares of the tape and all other squares are blank. The head scans the left-most symbol of the input string, and M is in the initial state q_0. At every step M is in some state q and the head is scanning a square on the tape containing some symbol s, and the action performed depends on the pair (q, s) and is specified by the machine's transaction function (or program) δ. The action consists of printing a symbol on the scanned square, moving the head left or right of one square, and taking a new state.

Formally the model introduced by Turing can be presented as follows. It is a tuple $\langle \Sigma, \Gamma, Q, \delta \rangle$ where Σ, Γ, Q are nonempty sets with $\Sigma \subseteq \Gamma$ and $b \in \Gamma - \Sigma$. The state set Q contains three special states q_0, q_{accept} and q_{reject}. The transition function δ satisfies:

$$\delta : (Q - \{q_{accept}, q_{reject}\}) \times \Gamma \to Q \times \Gamma \times \{-1, 1\}.$$

$\delta(q, s) = (q', s', h)$ is interpreted as: if M is in the state q scanning the symbol s then q' is the new state, s' is the new symbol printed on the tape, and the tape head moves left or right of one square (this depends whether h is -1 or 1). We assume $Q \cap \Gamma = \emptyset$. A configuration of M is a string xqy with $x, y \in \Gamma^*$, y is not the empty string, $q \in Q$. We interpret the configuration xqy as follows: M is in state q with xy on its tape, with its head scanning the left-most symbol of y.

Definition 1.1.1 *If C and C' are configurations, then $C \xrightarrow{M} C'$ if $C = xqsy$ and $\delta(q, s) = (q', s', h)$ and one of the following holds:*

1. *$C' = xs'q'y$ and $h = 1$ and y is nonempty.*
2. *$C' = xs'q'b$ and $h = 1$ and y is nonempty.*
3. *$C' = x'q'as'y$ and $h = -1$ and $x = x'a$ for some $a \in \Gamma$.*
4. *$C' = q'bs'y$ and $h = -1$ and x is empty.*

A configuration xqy is halting if $q \in \{q_{accept}, q_{reject}\}$.

Definition 1.1.2 *A computation of M on input $w \in \Sigma^*$, where Σ^* is the set of all finite string over Σ, is the unique sequence C_0, C_1, ... of configurations such that $C_0 = q_0 w$ (or $C_0 = q_0 b$ if w is empty) and $C_i \stackrel{M}{\rightarrow} C_{i+1}$ for each i with C_{i+1} in the computation, and either the sequence is infinite or it ends in a halting configuration.*

If the computation is finite, then the number of steps is one less than the number of configurations; otherwise the number of steps is infinite.

Definition 1.1.3 *M accepts w if and only if the computation is finite and the final configuration contains the state q_{accept}.*

Informally the complexity class $\mathcal{P}$ is the class of decision problems solvable by an some algorithm within a number of steps bounded by some fixed polynomial in the lenght of the input. Formally the elements belonging to the class $\mathcal{P}$ are languages. Let Σ be a finite alphabet with at least two elements, and Σ^*, as above, the set of all finite strings over Σ. A language over Σ is $L \subseteq \Sigma^*$. Each Turing machine M has an associated input alphabet Σ. For each string $w \in \Sigma^*$ there exists a computation associated with M and with input w. We said above[1] that M accepts w if this computation terminates in the accepting state.[2] The language accepted by M that we denote by $L(M)$ has associated alphabet Σ and is defined by

$$L(M) = \{w \in \Sigma^* |\ M \text{ accepts } w\}.$$

Let $t_M(w)$ be the number of steps in the computation of M on input w. If this computation never halts then $t_M(w) = \infty$. For $n \in \mathbb{N}$ we denote by $T_M(n)$ the worst case run time of M; i.e.

$$T_M(n) = max\{t_M(w) |\ w \in \Sigma^n\}$$

where Σ^n is the set of all strings over Σ of lenght n. Thus, we say that M runs in polynomial time if there exists k such that for all n, $T_M(n) \leq n^k + k$. Then the class $\mathcal{P}$ of languages can be defined by the condition that a language L is in $\mathcal{P}$ if $L = L(M)$ for some Turing machine M which runs in polynomial time.

[1] See Definition 1.1.3.

[2] Notice that M fails to accept w if this computation ends in the rejecting state, or if the computation fails to terminate.

The complexity class $\mathcal{NP}$ can be defined as follows using the notion of a checking relation, which is a binary relation $R \subseteq \Sigma^* \times \Sigma_1^*$ for some finite alphabets Σ and Σ_1. We associate with each such relation R a language L_R over $\Sigma \cup \Sigma_1 \cup \{\#\}$ defined by $L_R = \{w\#y |\ R(w,y)\}$, where the symbol $\# \notin \Sigma$. R is polynomial time if and only if $L_R \in \mathcal{P}$. The class $\mathcal{NP}$ of languages can be defined by the condition that a language L over Σ is in $\mathcal{NP}$ if there is $k \in \mathbb{N}$ and a polynomial time checking relation R such that for all $w \in \Sigma^*$,

$$w \in L \Longleftrightarrow \exists y(|y| \ \leq |w|^k \wedge R(w,y))$$

where $|w|$ and $|y|$ denote the lenghts of w and y, respectively.

The question of whether $\mathcal{P} = \mathcal{NP}$ is one of the greatest unsolved problem in theoretical computer science and in contemporary mathematics. Most researchers believe that the two classes are not equal (of course, it is easy to see that $\mathcal{P} \subseteq \mathcal{NP}$). At the beginning of the '70s Cook and Levin, independently, pointed out that the individual complexity of certain problems in $\mathcal{NP}$ is related to that of the entire class. If a polynomial time algorithm exists for any of these problems then all problems in $\mathcal{NP}$ would be polynomially solvable. These problems are called $\mathcal{NP}$-complete problems. Since that time thousands of $\mathcal{NP}$-complete problems have been discovered. We recall here only the first and probably one of the most famous of them, the satisfiability problem. For a collection of these problems the interested reader can see [35].

Let ϕ be a Boolean formula in the De Morgan language with constants 0, 1 (the truth values $FALSE$ and $TRUE$) and propositional connectives: unary $\neg$ (the negation) and binary $\wedge$ and $\vee$ (the conjunction and the disjunction, respectively). A Boolean formula is said to be satisfiable if some assignment of 0s and 1s to the variables makes the formula evaluate to 1. The satisfiability problem is to test whether a Boolean formula ϕ is satisfiable; this problem is denoted by SAT. Let $SAT = \{\langle\phi\rangle \,|\phi \text{ is a satisfiable Boolean formula}\}$.

Theorem 1.1.4 (Cook [23], Levin [50]) *$SAT \in \mathcal{P}$ if and only if $\mathcal{P} = \mathcal{NP}$.*

Suppose that L_i is a language over Σ_i, $i = 1, 2$. Then $L_1 \leq_p L_2$ (L_1 is polynomially reducible to L_2) if and only if there is a polynomial time computable function $f : \Sigma_1^* \to \Sigma_2^*$ such that

$$x \in L_1 \Longleftrightarrow f(x) \in L_2,$$

for all $x \in \Sigma_1^*$.

Definition 1.1.5 *A language L is $\mathcal{NP}$-complete if $L \in \mathcal{NP}$ and every language $L' \in \mathcal{NP}$ is polynomial time reducible to L.*

A language L is said $\mathcal{NP}$-hard if all languages in $\mathcal{NP}$ are polynomial time reducible to it, even though it may not be in $\mathcal{NP}$ itself.

The heart of Theorem 1.1.4 is the following one.

Theorem 1.1.6 *SAT is $\mathcal{NP}$-complete.*

Consider the complement of SAT. Verifying that something is not present seems more difficult than verifying that it is present, thus it seems not obviously a member of $\mathcal{NP}$. There is a special complexity class, *co*$\mathcal{NP}$, containing the languages that are complements of languages of $\mathcal{NP}$. This new class leads to another open problem in computational complexity theory. The problem is the following: is *co*$\mathcal{NP}$ different from $\mathcal{NP}$? Intuitively the answer to this problem, as in the case of the $\mathcal{P}$ versus $\mathcal{NP}$ problem, is positive. But again we do not have a proof of this.

Notice that the complexity class $\mathcal{P}$ is closed under complementation. It follows that if $\mathcal{P} = \mathcal{NP}$ then $\mathcal{NP} = co\mathcal{NP}$. Since we believe that $\mathcal{P} \neq \mathcal{NP}$ the previous implication suggests that we might attack the problem by trying to prove that the class $\mathcal{NP}$ is different from its complement. In the next section we will see that this is deeply connected with the study of the complexity of propositional proofs in mathematical logic.

We conclude this section by recalling some basic definitions from circuit complexity which will be used afterwards and the classical notation for the estimate of the running time of algorithms, the so called Big-O and Small-o notation for time complexity.

Definition 1.1.7 *A Boolean Circuit C with n inputs variables $x_1, \ldots, x_n$ and m outputs variables $y_1, \ldots, x_m$ and basis of connectives $\Omega = \{g_1, \ldots, g_k\}$ is a labelled acyclic directed graph whose out-degree 0 nodes are labeled by y_j's, in-degree 0 nodes are labeled by x_i's or by constants from Ω, and whose in-degree $\ell \geq 1$ nodes are labeled by functions from Ω of arity ℓ.*

The circuit computes a function $C : 2^n \to 2^m$ in an obvious way, where we identify $\{0,1\}^n = 2^n$.

Definition 1.1.8 *The size of a circuit is the number of its nodes. Circuit complexity $C(f)$ of a function $f : 2^n \to 2^m$ is the minimal size of a circuit computing f.*

In one form of estimation of the running time of algorithms, called the asymptotic analysis, we look for understanding the running time of the algorithm when large inputs are considered. In this case we consider just the highest order term of the expression of the running time, disregarding both coefficient of that term and any other lower term. Throughout this work we will use the asymptotic notation to give the estimate of the running time of algorithms and procedures. Thus we think that for a self-contained presentation it is perhaps worth to recall the Big-O and Small-o notation for time complexity. Let $\mathbb{R}^+$ be the set of real numbers greater than 0. Let f and g be two functions f, $g : \mathbb{N} \to \mathbb{R}^+$. Then $f(n) = O(g(n))$ if positive integers c and n_0 exist so that for every integer $n \geq n_0$, $f(n) \geq cg(n)$.[3] In other words, this definition points out that if $f(n) = O(g(n))$ then f is less than or equal to g if we do not consider differences up to a constant factor. The Big-O notation gives a way to say that one function is asymptotically no more than another. The Small-o gives a way to say that one function is asymptotically less than another. Formally, let f and g be two functions f, $g : \mathbb{N} \to \mathbb{R}^+$. Then $f(n) = o(g(n))$ if

$$\lim_{n\to\infty} f(n)/g(n) = 0.$$

1.2 The complexity of propositional proofs

The complexity of propositional proofs has been investigated systematically since late '60s.[4] Cook and Reckhow in [24], [25] gave the general definition of propositional proof system. To be able to introduce their definition that plays a central role in our work and is foundamental in the theory of complexity of the propositional proofs, we start from an example that must be familiar to anyone who has some basic knowledge of mathematical logic.

Let $TAUT$ be the set of tautologies in the De Morgan language[5] with constants 0, 1 (the truth values $FALSE$ and $TRUE$) and propositional connectives: unary $\neg$ (the negation) and binary $\wedge$ and $\vee$ (the conjunction and the disjunction, respectively). The language also contains auxiliary symbols such as brackets and commas. The formulas are built up using the constants, the atoms (propositional variables) p_0,..., p_n, and the connectives. Consider the following example of set of axioms taken from Hilbert's and Ackermann's work [37], where $A \to B$ is just the abbreviation of $\neg A \vee B$,

1. $A \vee (A \to A)$

[3] When $f(n) = O(g(n))$ we say that $g(n)$ is an asymptotic upper bound for $f(n)$.
[4] The earliest paper on the subject is an article by Tseitin [73].
[5] Introduced in the previous section when we defined the problem SAT

2. $A \to (A \vee B)$
3. $(A \vee B) \to (B \vee A)$
4. $(B \to C) \to ((A \vee B) \to A \vee C))$

The only inference rule is *modus (ponendo) ponens*[6] (MP), $A \to B, A/B$ (i.e. $A, \neg A \vee B/B$).

The literature of mathematical logic contains a wide variety of propositional proof systems formalized with a finite number of axiom schemes and a finite number of inference rules. The example above is just one of many possible different formalizations. Any of such systems is called a *Frege System* and denoted by F. A more general definition for Frege systems can be given using the concept of a Frege rule.

Definition 1.2.1 *A Frege rule is a pair* $(\{\phi_1(p_0, ..., p_n),\ ...,\ \phi_k(p_0, ..., p_n)\}$, $\phi(p_0,\ ...,p_n))$, *such that the implication*

$$\phi_1 \wedge ... \wedge \phi_k \to \phi$$

is a tautology. We use $p_0, \ldots, p_n$ *for propositional variables and usually we write the rule as*

$$\frac{\phi_1, \ldots, \phi_k}{\phi}.$$

Notice that a Frege rule can have zero premisses and in which case it is called an axiom schema (as the example above for the axioms (1) to (4)).

Definition 1.2.2 *A Frege system* F *is determined by a finite complete set of connectives and a finite set of Frege rules. A formula* ϕ *has a proof in* F *if and only if* $\phi \in TAUT$.[7] F *is implicationally complete.*[8]

As consequence of the schematic formalization we have that, the relation "w is a proof of ϕ in F" is a polynomial time relation of w and ϕ.

We consider all finite objects in our proofs as encoded in the binary alphabet $\{0, 1\}$. In particular, we consider $TAUT$ as a subset of $\{0, 1\}^*$. The length of a formula ϕ is denoted $|\phi|$. The properties above lead to a more abstract definition of proof system [24],

[6] In Latin, the mode that affirms by affirming.

[7] The "if" direction is the completeness and the "only" direction is the soundness of F.

[8] Recall that F is implicationally complete if and only if any ϕ can be proved in F from any set $\{\delta_1, \cdots, \delta_n\}$ if every truth assignment satisfying all δ_i's satisfies also ϕ.

Definition 1.2.3 (Cook Reckhow [24]) *A propositional proof system is any polynomial time computable function $P : \{0,1\}^* \to \{0,1\}^*$ such that $Rng(P) = TAUT$. Any $w \in \{0,1\}$ such that $P(w) = \phi$ is called a proof of ϕ in P.*

Any Frege system can be seen as a propositional proof system in this abstract perspective. In fact, consider the following function P_F,

$$P_F(w) = \begin{cases} \phi & \text{if } w \text{ is a proof of } \phi \text{ in } P \\ 1 & \text{otherwise} \end{cases}$$

Definition 1.2.4 *A propositional proof system P is polynomially bounded if there exists a polynomial $p(x)$ such that any $\phi \in TAUT$ has a proof w in P of size $|w| \leq p(|\phi|)$.*

In other words, any propositional proof system P that proves all tautologies in polynomial size is polynomially bounded. In [24] has been proved the following fundamental theorem relating propositional proof complexity to computational complexity theory. We report the theorem and the sketch of the proof.

Theorem 1.2.5 (Cook Reckhow [24]) *$\mathcal{NP} = co\mathcal{NP}$ if and only if there exists a polynomially bounded proof system P.*

Proof. Notice that since SAT is $\mathcal{NP}$-complete and for all $\neg\phi$, $\neg\phi \notin TAUT$ if and only if $\phi \in SAT$, $TAUT$ must be $co\mathcal{NP}$-complete. Assume $\mathcal{NP} = co\mathcal{NP}$. Then by hypothesis $TAUT \in \mathcal{NP}$. Hence there exists a polynomial $p(x)$ and a polynomial time relation R such that for all ϕ,

$$\phi \in TAUT \text{ if and only if } \exists y(R(\phi, y) \wedge |y| \leq p(|\phi|)).$$

Now define the propositional proof system as follows:

$$P(w) = \begin{cases} \phi & \text{if } \exists y(R(\phi, y) \text{ and } w = (\phi, y) \\ 1 & \text{otherwise} \end{cases}$$

It is clear that P is polynomially bounded.
For the opposite direction assume that P is a polynomially bounded propositional proof system for $TAUT$. Let $p(x)$ be a polynomial satisfying Definition 1.2.4. Since for all ϕ,

$$\phi \in TAUT \text{ if and only if } \exists w(P(w) = \phi \wedge |w| \leq p(|\phi|),$$

we get that $TAUT \in \mathcal{NP}$. Let $R \in co\mathcal{NP}$. By the $co\mathcal{NP}$-completeness of $TAUT$, R is polynomially reducible to $TAUT$. Since $TAUT \in \mathcal{NP}$ then so is R. This shows that $co\mathcal{NP} \subseteq \mathcal{NP}$ and consequently also that $co\mathcal{NP} = \mathcal{NP}$.

□

Hence, if we believe that $\mathcal{NP} \neq co\mathcal{NP}$ then there is no polynomially bounded propositional proof system for classical tautologies. Recall from the previous section that if $\mathcal{NP} \neq co\mathcal{NP}$ then $\mathcal{P} \neq \mathcal{NP}$. To prove that $\mathcal{NP} \neq co\mathcal{NP}$ is equivalent, by Theorem 1.2.5, to prove that there is no propositional proof system that proves all classical tautologies in polynomial size. This line of research gave rise to the program of proving lower bounds for many propositional proof systems. As mentioned in [46] it would be unlikely to prove that $\mathcal{NP} \neq co\mathcal{NP}$ in this incremental manner by showing exponential lower bounds for all the proof systems known.[9] This is like trying to prove a universal statement by proving all its instances. Despite that, we may hope to uncover some hidden computational aspect in these lower bounds and thus to reduce the conjecture to some intuitively more rudimentary one. For more discussion on this the reader can see [46].

We conclude this section with the notion of polynomial simulation introduced in [24]. The definition 1.2.6 is simply a natural notion of quasi-ordering of propositional proof systems by their strength.

Definition 1.2.6 *Let P and Q be two propositional proof systems. The system P polynomially simulates Q, $P \geq_p Q$ in symbols, if and only if there is polynomial time computable function $g : \{0,1\}^* \to \{0,1\}^*$ such that for all $w \in \{0,1\}^*$, $P(g(w)) = Q(w)$.*

The function g translates proofs in Q into proofs in P of the same formula. Since in the definition above g is a polynomial time function, then the length of the proofs in P will be at most polynomially longer than the length of the original proofs in the system Q.

1.3 Resolution

The logical calculus Resolution R is a refutation system for formulas in conjunctive normal form. This calculus is popularly credited to Robinson [64] but it was already contained in Blake's thesis [10] and is an immediate consequence of Davis and Putnam work [30].

[9] Unless there is an optimal proof system.

A literal ℓ is either a variable p or its negation $\bar{p}$. The basic object is a clause, that is a finite or empty set of literals, $C = \{\ell_1, \ldots, \ell_n\}$ and is interpreted as the disjunction $\bigvee_{i=1}^{n} \ell_i$. A truth assignment $\alpha : \{p_1, p_2, \ldots\} \to \{0, 1\}$ satisfies a clause C if and only if it satisfies at least one literal l_i in C. It follows that no assignment satisfies the empty clause, which it is usually denoted by $\{\}$. A formula ϕ in conjunctive normal form is written as the collection $\mathcal{C} = \{C_1, \ldots, C_m\}$ of clauses, where each C_i corresponds to a conjunct of ϕ. The only inference rule is the resolution rule, which allows us to derive a new clause $C \cup D$ from two clauses $C \cup \{p\}$ and $D \cup \{\bar{p}\}$

$$\frac{C \cup \{p\} \;\; D \cup \{\bar{p}\}}{C \cup D}$$

where p is a propositional variable. C does not contain p (it may contain $\bar{p}$) and D does not contain $\bar{p}$ (it may contain p). The resolution rule is sound: if a truth assignment $\alpha : \{p_1, p_2, \ldots\} \to \{0, 1\}$ satisfies both upper clauses of the rule then it also satisfies the lower clause.

A resolution refutation of ϕ is a sequence of clauses $\pi = D_1, \ldots, D_k$ where each D_i is either a clause from ϕ or is inferred from earlier clauses D_u, D_v, u, $v < i$ by the resolution rule and the last clause $D_k = \{\}$. Resolution is sound and complete refutation system; this means that a refutation does exist if and only if the formula ϕ is unsatisfiable.

Theorem 1.3.1 *A set of clauses $\mathcal{C}$ is unsatisfiable if and only if there is a resolution refutation of the set.*

Proof. The "only-if part" follows easily from the soundness of the resolution rule. Now, for the opposite direction, assume that $\mathcal{C}$ is unsatisfiable and such that only the literals p_1, $\neg p_1, \ldots, p_n$, $\neg p_n$ appear in $\mathcal{C}$. We prove by induction on n that for any such $\mathcal{C}$ there is a resolution refutation of $\mathcal{C}$.

Basis Case: If $n = 1$ there is nothing to prove: the set $\mathcal{C}$ must contains $\{p_1\}$ and $\{\neg p_1\}$ and then by the resolution rule we have $\{\}$.

Induction Step: Assume that $n > 1$. Partition $\mathcal{C}$ in for disjoints sets:

$$\mathcal{C}_{00} \cup \mathcal{C}_{01} \cup \mathcal{C}_{10} \cup \mathcal{C}_{11}$$

of those clauses which contain no p_n and no $\neg p_n$, no p_n but do contain $\neg p_n$, do contain p_n but not $\neg p_n$ and contain both p_n and $\neg p_n$, respectively. Produce a new set of clauses $\mathcal{C}'$ by:

(1) Delete all clauses from $\mathcal{C}_{11}$.

(2) Replace $\mathcal{C}_{01} \cup \mathcal{C}_{10}$ by the set of clauses that are obtained by the application of the resolution rule to all pairs of clauses $C_1 \cup \{\neg p_n\}$ from $\mathcal{C}_{01}$ and to $C_2 \cup \{p_n\}$ from $\mathcal{C}_{10}$.

The new set of clauses do not contain either p_n or $\neg p_n$. It is easy to see that the new set of clauses $\mathcal{C}'$ is also satisfiable. Any assignment $\alpha' : \{p_1, \ldots, p_{n-1}\} \to \{0,1\}$ satisfies all clauses C_1 such that $C_1 \cup \{\neg p_n\} \in \mathcal{C}_{01}$, or all clauses C_2 such that $C_2 \cup \{p_n\} \in \mathcal{C}_{01}$. Hence α' can be extended to a truth assignment α satisfying $\mathcal{C}$, which is a contradiction because by our hypothesis $\mathcal{C}$ is unsatisfiable.

□

A resolution refutation $\pi = D_1, \ldots, D_k$ can be represented as a directed acyclic graph (dag-like) in which the clauses are the vertices, and if two clauses $C \cup \{p\}$ and $D \cup \{\bar{p}\}$ are resolved by the resolution rule, then there exists a direct edge going from each of the two clauses to the resolvent $C \cup D$. A resolution refutation $\pi = D_1, \ldots, D_k$ is tree-like if and only if each D_i is used at most once as a hypothesis of an inference in the proof. The underlying graph of π is a tree. The proof system allowing exactly tree-like proofs is called tree-like resolution and denoted by R^*.

In propositional proof complexity, perhaps the most important relation between dag-like refutations and refutations in R^* is that the former can produce exponentially shorter refutations then the latter. A simple remark on this is that in a tree-like proof anything which is nedeed more than once in the refutation must be derived again each time from the initial clauses. A superpolynomial separation between R^* and R was given in [75], and later by others in [20] and [38]. Later on, in [11] has been presented a family of clauses for which R^* suffers an exponential blow-up with respect to R. For an improvement of the exponential separation the reader can see [7].

1.4 Interpolation and effective interpolation

The Craig interpolation theorem is a basic result in mathematical logic [28]. The theorem says that whenever an implication $A \to B$ is valid then there exists a formula I, called an interpolant, which contains only those symbols of the language occurring in A and B and such that the two implications $A \to I$ and $I \to B$ are both valid formulas. The theorem holds for propositional logic

as well as for first order logic.[10]

The problem of finding an interpolant for the implication is quite relevant to computational complexity theory. To see this, it is enough to observe what follows. Let U and V be two disjoints $\mathcal{NP}$-sets, subsets of $\{0,1\}^*$. By the proof of the $\mathcal{NP}$-completeness of satisfiability [23] there are sequences of propositional formulas $A_n(p_1,\ldots,\ p_n,\ q_1,\ldots,\ q_{s_n})$ and $B_n(p_1,\ldots,\ p_n,\ r_1,\ldots,\ r_{t_n})$ such that the size of A_n and B_n is $n^{O(1)}$ and such that

$$U_n := U \cap \{0,1\}^n = \{(\delta_1,\ldots,\delta_n \in \{0,1\}^n | \exists \alpha_1,\ldots,\alpha_{s_n} A_n(\bar{\delta},\bar{\alpha}) \text{ holds}\}$$

and

$$V_n := V \cap \{0,1\}^n = \{(\delta_1,\ldots,\delta_n \in \{0,1\}^n | \exists \beta_1,\ldots,\beta_{t_n} A_n(\bar{\delta},\bar{\beta}) \text{ holds}\}.$$

The assumption that the sets U and V are disjoint sets is equivalent to the statement that the implications $A_n \to \neg B_n$ are all tautologies. By Craig's interpolation theorem there is a formula $I_n(\bar{p})$ constructed only using atoms $\bar{p}$ such that

$$A_n \to I_n$$

and

$$I_n \to \neg B_n$$

are both tautologies. Thus the set

$$W := \bigcup\nolimits_n \{\bar{\delta} \in \{0,1\}^n | I_n(\bar{\delta}) \text{ holds}\}$$

defined by the interpolant I_n separates U from V: $U \subseteq W$ and $W \cap V = \emptyset$. Hence an estimate of the complexity of propositional interpolation formulas in terms of the complexity of an implication yields an estimate to the computational complexity of a set separating U from V. In particular, a lower bound to a complexity of interpolating formulas gives also a lower bound on the complexity of sets separating disjoint $\mathcal{NP}$-sets. Of course, we cannot really expect to polynomially bound the size of a formula or a circuit defining a suitable W from the lenght of the implication $A_n \to \neg B_n$. This is because, as remarked by Mundici [53], it would imply that $\mathcal{NP} \cap co\mathcal{NP} \subseteq \mathcal{P}/poly$. In fact, for $U \in \mathcal{NP} \cap co\mathcal{NP}$ we can take V to be the complement of U and hence it must hold that $W = U$.

Krajíček formulated the idea of effective interpolation in [47]. The idea is nice and more subtle than that one displayed above. For a given propositional proof system, try to estimate the circuit-size of an interpolant of an

[10] Throghout all this work by Craig interpolation's theorem we mean the propositional version of it.

implication in terms of the size of the shortest proof of the implication. In other words, for a given propositional proof system establish an upper bound on the computational complexity of an interpolant of A and B in terms of the size of a proof of the validity of $A_n \rightarrow \neg B_n$. Then any pair A and B which is hard to interpolate yields a formula which must have large proofs of validity. This fact can be exploited in proving lower bounds, and indeed several new lower bounds came out from its application, see [49], [60]. The idea has been also applied fruitfully in other areas such as bounded arithmetic in proving results of independence [62] and on establishing links between proof complexity and cryptography and in automatizability of proof search. The reader interested in some overviews can see [46] and [59].

Definition 1.4.1 *A propositional proof system P admits effective interpolation if and only if there is a polynomial $p(x)$ such that any implication $A \rightarrow B$ with a proof in P of size m has an interpolant of a circuit size $\leq p(m)$.*

The main point of the effective interpolation method is that by establishing a good upper bound for a proof system P in the form of the effective interpolation we prove lower bounds on the size of the proofs in P. That is,

Theorem 1.4.2 *Assume that U and V are two disjoints $\mathcal{NP}$-sets such that U_n and V_n are inseparable by a set of circuit complexity $\leq s(n)$, all $n \geq 1$. Assume that P admits effective interpolation. Then the implications $A_n \rightarrow \neg B_n$ require proofs in P of size $\geq s(n)^{\epsilon}$, for some $\epsilon > 0$.*

The main point in this section is to prove that R admits effective interpolation. To be able to give the proof in some detail we must recall few notions and facts from communication complexity. Let U_n,$V_n \subseteq \{0,1\}^n$ be two disjoint sets. Karchmer-Wigderson game [39] on U_n and V_n is played by two players A and B. Player A receives an element u from U_n and player B receives an element v from V_n. A and B have a protocol on which they agreed on. The two players communicate bits of information until both agree on the same $i \in [n]$ such that $u_i \neq v_i$. A measure of the complexity of the game is the minimum of the number of bits they need to communicate in the worst case over all protocols. This minimum is denoted by $C(U_n,V_n)$ and is called the communication complexity of the game.

Consider a propositional formula $\phi(p_1,\ldots,\ p_n)$ with $\neg$ just in front of atoms, that takes constantly value 1 on U_n and value 0 on V_n. Then ϕ separate U_n from V_n. The players can use ϕ as follows. They start from the principal connective and will, step by step, work down to smaller subformulas until a literal is not reached. The property that they will preserve is that the

current subformula takes value 1 on u and 0 on v. At the beginning this is true by hypothesis. If the principal connective is a conjunction the player B indicates to A which of the two subformulas takes value 0 on v. On the other hand, if the principal connective is a disjunction A indicates to B which of the two subformulas is 1 on u. The reader can find a proof of the following theorem in [39],

Theorem 1.4.3 (Karchmer-Wigderson [39]) *Let U_n,$V_n \subseteq \{0,1\}^n$ be two disjoint sets. Then $C(U_n,V_n)$ is equal to the minimal depht of a De Morgan formula separating U_n and V_n.*

Suppose that there is a circuit C separating U_n from V_n instead of ϕ. The players can use the same communication protocol. $C(U_n,V_n)$ will be bounded by the depth of C, but no information about the size of C is obtained. For this reason the notion of protocol has been generalized in [49], generalizing Razborov [62], as follows

Definition 1.4.4 *Let U_n,$V_n \subseteq \{0,1\}^n$ be two disjoint sets. A protocol for the Karchmer-Wigderson game on the pair (U_n, V_n) is a labelled directed graph G satisfying the following conditions:*

1. *G is acyclic and has one source denoted $\diamond$. The nodes with the out-degree 0 are leaves and all other are inner nodes.*

2. *Leaves are labeled by one of the following formulas:*

$$u_i = 1 \wedge v_i = 0 \text{ or } u_i = 0 \wedge v_i = 1$$

   for some $i = 1, \ldots, n$.

3. *There is a function $S(u, v, x)$ such that S assigns to a node x and a pair $(u, v) \in U_n \times V_n$ an edge from the node x (the function S is called the strategy).*

   Fixing a pair $(u, v) \in U_n \times V_n$ the strategy defines for every node x a directed path $P^x_{(u,v)} = x_1, \ldots, x_h$ in G: start at x and go toward a leaf x_h, always going from x_i using the edge $S(u, v, x_i)$.

4. *For every $(u, v) \in U_n \times V_n$ there is a set $F(u, v) \subseteq G$ satisfying:*

   (a) $\diamond \in F(u, v)$.

   (b) $x \in F(u, v) \rightarrow P^x_{(u,v)} \subseteq F(u, v)$.

   (c) The label of any leaf from $F(u, v)$ is valid for u, v.
   The set F is called the consistency condition.

Then given a protocol for the game on U_n and V_n a suitable circuit separating U_n and V_n can be found. The following theorem was stated and proved in [62].

Definition 1.4.5 *The communication complexity of G is the minimal number t such that for every $x \in G$ the players (one knowing u and x, the other one v and x) decide whether $x \in F(u,v)$ and compute $S(u,v,x)$ with at most t bits exchanged in the worst case.*

A new proof of Theorem 1.4.6 is contained in [46].

Theorem 1.4.6 *Let U_n,$V_n \subseteq \{0,1\}^n$ be two disjoint sets. Let G be a protocol for the game on U_n, V_n which has k nodes and the communication complexity t. Then there exists a circuit C of size $k2^{O(t)}$ separating U_n from V_n. On the other hand, any circuit C of size s separating U_n from V_n determines a protocol G with s nodes whose communication complexity is 2.*

Now we have all the essential background for proving effective interpolation for R. The proof of Theorem 1.4.7 below follows in detail [46].

Theorem 1.4.7 (Krajíček [49]) *Assume that the set of clauses*

$$\{A_1, ..., A_m, B_1, ..., B_l\}$$

where

1. $A_i \subseteq \{p_1, \neg p_1, ..., p_n, \neg p_n, q_1, \neg q_1, ..., q_s, \neg q_s\}$, *all* $i \leq m$
2. $B_j \subseteq \{p_1, \neg p_1, ..., p_n, \neg p_n, r_1, \neg r_1, ..., r_t, \neg r_t\}$, *all* $j \leq l$

has a resolution refutation with k clauses.

Then the implication

$$\bigwedge_{i \leq m} (\bigvee A_i) \rightarrow \neg \bigwedge_{i \leq l} (\bigvee B_j)$$

has an interpolant I whose circuit-size is $kn^{O(1)}$.

Proof. Let π be a resolution refutation with k clauses of $\{A_1, ..., A_m, B_1, ..., B_l\}$. Let U and V be the subsets of $\{0,1\}^n$ defined by

$$U := \{p \in \{0,1\}^n | \exists q \in \{0,1\}^s, \bigwedge_i \bigvee A_i\}$$

and

$$V := \{p \in \{0,1\}^n | \exists r \in \{0,1\}^t, \bigwedge_j \bigvee B_j\}$$

respectively. Before to show how to transform π into a protocol for the Karchmer-Wigderson game and the formal construction of the protocol, consider the following argument. Assume that $\pi = D_1,\ldots,D_k$. For a clause D we denote by $\tilde{D}$ the set of all truth assignment satisfying D. Now asssume that the player A gets $u \in U$ and the player B gets $v \in V$. The player A fixes some $q^u \in \{0,1\}^s$ such that $\bigwedge \bigvee A_i(u, q^u)$ holds. Similarly B picks a witness of the membership of v in V.

The two players will construct a path $P = S_0,\ldots,S_h$ through π from S_0 to the initial sequents. They will try to keep the following property: *the truth evaluations (u, q^u, r^v) and (v, q^u, r^v) do not satisfy the clauses on the path (that is, they are not in $\tilde{S_a}$, $a = 0,\ldots,h$.)*

Assume that A and B reach S_a which was deduced in π by the inference $X, Y/S_a$. They first determine whether $(u, q^u, r^v) \in \tilde{X}$ and $(v, q^u, r^v) \in \tilde{Y}$ and then continue depending on a possible outcome:

1. $(u, q^u, r^v) \in \tilde{X} \wedge (v, q^u, r^v) \in \tilde{X}$.
2. $(u, q^u, r^v) \notin \tilde{X} \wedge (v, q^u, r^v) \notin \tilde{X}$.
3. Exactly one of (u, q^u, r^v), (v, q^u, r^v) is in $\tilde{X}$.

In the first case none of the two tuples can be in $\tilde{Y}$, then the players put $S_{a+1} := Y$. In the second case they put $S_{a+1} := X$. The third case is more complicated. Since U and V are disjoint sets $u \neq v$ and the players stop constructing the path enter a protocol aimed at finding $i \leq n$ such that $u_i \neq v_i$. Each initial sequent is satisfied either by (u, q^u, r^v) or by (v, q^u, r^v). Then the players must sooner or later introduce the third possibility and find $i \leq n$ such that $u_i \neq v_i$. We need to show that each of the three following problem has small communication complexity. Let D be a clause,

1. Decide whether $(u, q^u, r^v) \in \tilde{D}$.
2. Decide whether $(v, q^u, r^v) \in \tilde{D}$.
3. If $(u, q^u, r^v) \in \tilde{D} \neq (v, q^u, r^v) \in \tilde{D}$ find $i \leq n$ such that $u_i \neq v_i$.

The first two can be decided by each player sending one bit and the third task needs only $\log(n)$ bits by a binary search. Now we show how to construct the protocol G formally. G has $(k + 2n)$ nodes, the k clauses from

π together $2n$ extra vertices. These extra vertices are labelled by formulas $u_i = 1 \wedge v_i = 0$ and $u_i = 0 \wedge v_i = 1$, $i = 1, \ldots, n$. The consistency condition is constituted by those clauses D_j that are not satisfied by (v, q^u, q^v), and also by those of extra $2n$ nodes whose label is valid for the pair (u, v). Finally, the strategy function $S(u, v, D)$ is defined as follows:

1. If $(u, q^u, r^v) \notin \tilde{D}_j$ then

$$S(u, v, D_j) := \begin{cases} X & \text{if } (v, q^u, r^v) \notin \tilde{X} \\ Y & \text{if } (v, q^u, r^v) \in \tilde{X} \text{ (and } (v, q^u, r^v) \notin \tilde{Y}). \end{cases}$$

2. If $(u, q^u, r^v) \in \tilde{D}_j$ then the players use binary search for finding $i \leq n$ such that $u_i \neq v_i$. $S(u, v, D_j)$ is then the one of the two node labeled by $u_i = 1 \wedge v_i = 0$ and $u_i = 0 \wedge v_i$ whose label is valid for the pair (u, v).

The strategy function $S(u, v, x)$ as well as the membership relation $x \in F(u, v)$ can be determined exchanging at most $\log(n)$ bits. Since G has $(k + 2n)$ nodes then by Theorem 1.4.6 we obtain a circuit separating U from V nad having the size $\leq (k + 2n)2^{O(\log(n))} = kn^{O(1)}$.

□

1.5 "Mathematical" proof systems

The set of propositional tautologies $TAUT$ is a $co\mathcal{NP}$-complete set. In general a proof system is a relation $R(x, y)$ computable in polynomial time such that

$$x \in TAUT \text{ if and only if } \exists y(R(x, y)).$$

A proof of x is a y such that $R(x, y)$ holds. Thus one can take an $co\mathcal{NP}$-complete set and a suitable relation R over it and investigate the complexity of such proofs. In this section we recall a few proof systems (only one in some detail) "mathematically" based on $co\mathcal{NP}$-complete sets.

A nice example of well-known "mathematical[11]" proof system is the proof system Cutting Plane CP. The Cutting Plane proof system (CP) is a refutation system based on showing the non-existence of solutions for a family of linear equalities. A line in a proof in th system CP is an expression of the form

$$\sum a_i \cdot x_i \geq B$$

[11] This expression is taken from Pudlák [59]

where a_1,..., a_n, B are integers. Then for a given clause C, and the variables x_i, $i \in P$, occur positively in C, and variables x_i, $i \in N$, occur negatively in C, then C is represented by the linear inequality

$$\sum_{i \in P} x_i - \sum_{i \in N} x_i \geq 1 - |N|.$$

A CNF formula is represented by the family of linear inequalities corresponding to its clauses. Thus for example the formula $(x_1 \vee \bar{x_2} \vee x_3) \wedge (\bar{x_1} \vee x_3 \vee x_4 \vee \bar{x_5})$ is represented by the inequalities $x_1 - x_2 + x_3 \geq 0$ and $-x_1 + x_3 + x_4 - x_5 \geq -1$. The axioms of the proof system are $x_i \geq 0$, $-x_i \geq -1$. The rules of inference are:

- $$\frac{\sum a_i \cdot x_i \geq A \;\; \sum b_i \cdot x_i \geq B}{\sum (a_i + b_i) \cdot x_i \geq A + B}$$

- $$\frac{\sum a_i \cdot x_i \geq A}{\sum (c \cdot a_i) \cdot x_i \geq c \cdot A}$$

  where $c \geq 1$ is an arbitrary integer;

- $$\frac{\sum (c \cdot a_i) \cdot x_i \geq A}{\sum a_i \cdot x_i \geq \left\lceil \frac{A}{c} \right\rceil}$$

  where $c > 1$ is an arbitrary integer.

A derivation D of the inequality I from inequalities I_1, ..., I_m is a sequence D_1,...,D_n such that $I = D_n$ and for all $i < n$ either D_i is an axiom, or one of I_i, ..., I_m or inferred from D_j, D_k for $j, k < i$ by means of a rule of inference. A CP refutation of I_1,...,I_m is a derivation of $0 \geq 1$ from I_1, ..., I_m.

We have seen above the soundness and the completeness of Resolution for CNF formulas, see Theorem 1.3.1. Soundness in the sense that given any formula ϕ which has a resolution refutation refutation π, ϕ is not satisfiable and completeness in the sense that given any unsatisfiable formula ϕ, there is a resolution refutation π of ϕ. Theorem 1.5.1 can be proved exploiting the completeness of R, since CP easily simulates resolution as observed in [27].

Theorem 1.5.1 *The proof system CP is sound and complete with respect to CNF formulas.*

For the soundness part we can argue as follows. Let ϕ be a CNF formula with a CP refutation γ. Suppose ϕ is satisfied by the assignment α. Instantiate each inequality in γ of ϕ by assigning the boolean variables their value under α. By induction on the length of γ we can prove that each instantiated inequality in the refutation γ holds. This is contradiction, because we cannot have the inequality $0 \geq 1$ as the last element of the refutation. Goerdt [36] proved that Frege systems polynomially simulate the CP proof system.

Other examples of mathematical proof systems are for instance the Nullstellensatz system introduced in [5], the Polynomial Calculus [19] and the Gaussian Calculus first defined in [6].

The interesting fact is that almost all the mathematical proof systems known are in some sense algebraically based. One of the aim behind this work is to find a new proof system but with a different origin. Indeed, we propose a new mathematical proof system and it comes out from the field of cellular automata.

Chapter 2

String Rewriting and Propositional Proof Complexity

Rewriting is a technique for defining complex objects by successively replacing parts of a simple initial object using a set of rewriting rules. The object can be a finite string of characters or more complicated, like for example a polygon. For instance, consider the continuous snowflakes curve of Figure 2.1 proposed by von Koch [44] as an example of curve not differentiable anywhere. It can be defined using this approach using as initial object an equilateral triangle and rewriting procedures replacing parts of its edges.

The most extensively studied and best understood rewriting systems deal with character strings. The main reason for that is Chomsky's work on formal grammars, in late 1950s, in which he applied the concept of rewriting in order to describe the syntactic features of natural languages, [18]. Thus a string rewriting system can be interpreted as a device for generating and recognizing formal languages; sometimes in the literature they are also called combinatorial systems, [29].

In this rewriting context we are considering transformations of some object by step by step activity. Given a finite alphabet and a definition of word over it, we allow a set of rewriting rules in order to transform words from the set of all words. The sequence of application of rules can be seen as a proof in the classical sense. In this chapter we show how this idea can be implemented and used to interpret the well known propositional calculus Resolution considered in the Section 3 of the previous chapter. We give a characterization of the rewriting system which represents tree-like resolution using diagrams in van Kampen style [12]; the dag-like case is less natural, but by allowing more complex rules it can be interpreted in the same manner.

Formerly van Kampen diagrams have been discussed by Krajíček, in [47]; he gave a link between the Dehn function of finitely presented groups and the

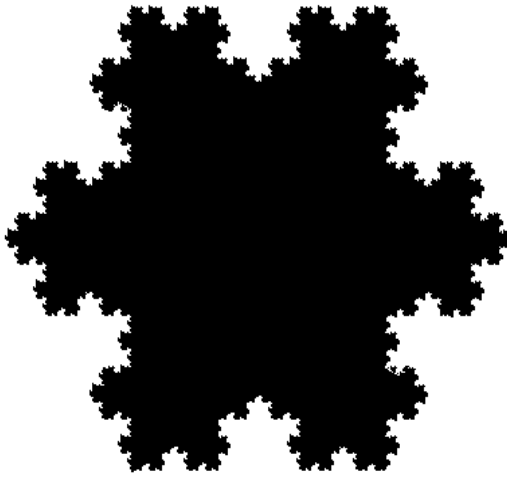

Figure 2.1: The von Koch curve.

length of proofs function in propositional proof complexity. A motivation for this is the hope to get a new perspective on proof systems using a different approach and exploiting also geometric interpretations to characterise several proof complexity measures.

The chapter is organized as follows. In section 1 we introduce string rewriting systems, also known as semi-Thue systems.[1] In section 2 we interpret tree-like resolution as semi-Thue system. We show that the new rewriting system is complete and sound with respect to Resolution, see Theorem 2.2.3 and 2.2.5 respectively. In this section we also discuss more in detail the motivation for considering rewriting systems in the context of proof complexity. Then section 3 introduces a representation by planar diagrams of the proofs in the semi-Thue system interpreting tree-like resolution. Section 4 is devoted to the dag-like case: we interpret dag-like resolution as string rewriting system. Section 5 has some concluding remarks on this topic.

[1] Axel Thue (1863-1922) introduced the first systematic treatement of string rewriting systems in the early 20th century, see [70], [71].

2.1 The String rewriting system Σ_n^*

In general, a string rewriting system is a substitution system used to transform a given string according to specified rewriting rules. A semi-Thue system Σ is a string rewriting system. Throughout this chapter, "semi-Thue system" and "string rewriting system" are used meaning the same mathematical concept. It is a tuple (A, Δ) where A is a finite alphabet and Δ is a set of ordered pairs $\Delta \subseteq A^* \times A^*$, where A^* is the set of all words over A. The elements of Δ, (q, z), are referred as string rewriting rules and denoted by $q \to z$. If a semi-Thue system Σ is symmetric, $\Delta = \Delta^{-1}$, then Σ is called a Thue system. In order to obtain the semi-Thue system, that we call Σ_n^*, we define its alphabet A_n.

Definition 2.1.1 *Let A_n be a finite alphabet containing two uppercase letters L, R, and two types of lowercase letters x_i and $\bar{x}_i$, for $i = 1, \ldots, n$. A word w over A_n is a finite string consisting of zero or more elements of A_n. The set of all words over A_n is denoted by A_n^*. The empty word is denoted by Λ.*

In Σ_n^* a special kind of words over A_n, called **regular-words**, will play an important role. By $u \subseteq w$ we mean that u is a subword of w.

Definition 2.1.2 *A word $w \in A_n^*$ is regular if and only if $w = \Lambda$ or it has the following form:*

$$w = Lu_1RLu_2R \ldots Lu_rR$$

where each $u_i \in (A_n \setminus \{L, R\})^*$, for $1 \leq i \leq r$. Any of u_i can be empty. A word w_i that is regular and $w_i \subseteq w_j$ is called a regular subword of w_j.

Of course, by the definition given above the following strings are regular words: LR, $Lx_1\bar{x}_3R$, and $Lx_1RL\bar{x}_2R$; by definition Lx_1R is a regular subword of $Lx_1RL\bar{x}_2R$, and $Lx_1\bar{x}_3R$ is a regular subword of $Lx_1\bar{x}_3R$. It is implicit in the definition above that L and R must alternate, i.e. we cannot have two consecutive Ls or Rs in the word if the word is regular. Moreover LR is also a regular subword of LR.

The size of a word w is the number of symbols contained in it[2]; excluding Λ which has size 0, the smallest size of a regular word is 2, namely is the first word given in the example above, LR. Then we define a special type of words which is minimal with respect to that of regular word. They are called **clause-words**.

[2]Note that the L,R letters in the regular words are just delimiters, but they contribute to the size.

Definition 2.1.3 *A clause-word is a regular word of A_n^* that starts with L and ends with R and it does not include any other occurrences of L and R in it.*

Thus given $Lx_1\bar{x}_2RLx_3\bar{x}_4\bar{x}_5RLx_5x_7R$, the words $Lx_1\bar{x}_2R$, $Lx_3\bar{x}_4\bar{x}_5R$ and Lx_5x_7R are clause-words. It follows easily that all clause-words are regular words; the viceversa does not hold. We distinguish between these two notions when needed, otherwise we will use the general notion of regular word introduced before. Then the semi-Thue system Σ_n^* can be defined as follows:

Definition 2.1.4 *Let Σ_n^* be a semi-Thue system in the alphabet A_n with rewriting rules, Σ_n^*-rules:*

1. ***Elimination rules**:*
   (a) $x_iRL\bar{x}_i \rightarrow \Lambda$;
   (b) $\bar{x}_iRLx_i \rightarrow \Lambda$;
   for any i.
2. ***Exchanging rules**:*
   (a) $x_ix_j \rightarrow x_jx_i$;
   (b) $\bar{x}_i\bar{x}_j \rightarrow \bar{x}_j\bar{x}_i$;
   (c) $x_i\bar{x}_j \rightarrow \bar{x}_jx_i$;
   (d) $\bar{x}_ix_j \rightarrow x_j\bar{x}_i$;
   for any i, j.

Some comments on the rewriting rules that we have chosen. Rules (1a) and (1b) are string rewriting rules simulating the resolution rule given in Section 1. The rules (2a)-(2d) can be useful because sometime we have regular words in which rules (1a), (1b) cannot be directly applied and we need to permute lower case letters first. Then by the exchanging rules we can move lower case letters in order to obtain suitable strings such that the elimination rules can be applied. Notice that from the previous definition follows easily that for any given finite set of lowercase characters $x_1 \dots x_n$ determining A_n we obtain a specific semi-Thue system Σ_n^*.

To be able to define properly the notion of proof in Σ_n^* we introduce a special set of regular words called **initial-words**. We denoted this set by $\mathcal{I}$. Moreover, we introduce in the definition below the rules (i) and (ii) that allow us to manipulate the elements of $\mathcal{I}$. These rules are called **introduction-rules** ($\mathcal{I}$-rules).

Definition 2.1.5 *Let $\mathcal{I} = \{w_1, \ldots, w_t\}$ be a non empty set of regular words. $\mathcal{I}$-rules are:*

(i) $L \to uL$, where $u \in \mathcal{I}$;

(ii)$R \to Ru$, where $u \in \mathcal{I}$.

The notion of Σ_n^*-proof is defined as follows

Definition 2.1.6 *A proof π' in Σ_n^* of the regular word w from a non empty set of regular words $\mathcal{I}$, denoted by*

$$\pi' : \mathcal{I} \vdash_{\Sigma_n^*} w$$

*is a finite sequence of regular words $w_1, \ldots, w_k$ such that $w_1 \in \mathcal{I}$, $w_k = w$ and each w_i, for $1 < i \leq k$, w_i is obtained from w_{i-1} by an $\mathcal{I}$-rule or a Σ_n^*-rule. w_1 is called the **source-word** of π'.*

Example. We conclude this section giving an example of proof of LR in Σ_n^*. Let $\mathcal{I} = \{Lx_1x_2R,\ Lx\bar{1}R,\ Lx\bar{2}R\}$ be a set of initial words. Let Lx_1x_2R be the source-word. The words $Lx\bar{1}R$, $Lx\bar{2}R$ are entered using $\mathcal{I}$-rules (i) and (ii):

By the introduction rule (ii), we have

$$Lx_1x_2R \to Lx_1x_2RLx\bar{2}R$$

then by rule (1a):

$$Lx_1x_2RLx\bar{2}R \to Lx_1R$$

by rule (ii) (or we can use also (i)):

$$Lx_1R \to Lx_1RLx\bar{1}R$$

Finally, by rule (1a) if we have used in the previous step the rule (ii) (otherwise, if rule (i) has been applied we use (1b)), we obtain:

$$Lx_1RLx\bar{1}R \to LR.$$

2.2 Σ_n^* and R^*: the tree-like case

In order to interpret R^* as the semi-Thue system Σ_n^* we need to set some correspondences.

Definition 2.2.1 *If $C = \{\ell_1, \ldots, \ell_l\}$ is a clause, then $w_C = L\ell_1 \ldots \ell_l R$ is a clause-word in the language of Σ_n^*. For definiteness we assume the ordering on variables given by their indices. If $\mathcal{C} = \{C_1, \ldots, C_t\}$ is a set of clauses, then $w_{\mathcal{C}} = w_{C_1} \ldots w_{C_t}$ is a regular word in the language of Σ_n^*. We assume clauses are canonically ordered in a fixed way.*

Using Definition 2.2.1 we take a clause C_i in the language of Resolution and we obtain a regular word w_{C_i} in the language of Σ_n^*. For example, take the clause $\{x_1 x_2\}$; by definition we have the regular word Lx_1x_2R. Notice that by our definition a set of clauses $\mathcal{C}$ corresponds to a regular word $w_{\mathcal{C}}$. In fact, let $\mathcal{C} = \{C_1, C_2\}$=$\{\{x_1\}, \{x_2x_3x_6\}\}$ be a set of clauses. Then by the definition above $w_{\mathcal{C}} = Lx_1RLx_2x_3x_6R$.

The clause-words defined in the Definition 2.1.3 give the correspondence with the basic object of the resolution calculus, clauses. Indeed, if w_i is a clause-word then it has the form $L\ell_1 \ldots \ell_n R$ that corresponds to a clause $\{\ell_1, \ldots, \ell_n\}$ where ℓ_i are literals (with $1 \leq i \leq n$). Notice that while the ordering of variables in the language of Resolution is not important, in the case of the language of Σ_n^* this has some relevance during the manipulation.

Definition 2.2.2 *Let $\mathcal{C} = \{C_1, \ldots, C_t\}$ be a set of clauses. Then*

$$\mathcal{I}_{\mathcal{C}} = \{w_{C_1}, \ldots, w_{C_t}\}$$

where $w_{C_1}, \ldots, w_{C_t}$ are clause-words corresponding to the clauses $C_1, \ldots, C_t$.

Thus a resolution refutation π starting from a set of clauses $\mathcal{C}$ and ending with $\{\}$ in R^* can be conceivably interpreted as a Σ_n^*-proof of LR from $\mathcal{I}_{\mathcal{C}}$. This is what we do next. Notice that the string rewriting system allowing $\mathcal{I}$-rules gives the opportunity to use words from $\mathcal{I}_{\mathcal{C}}$ when needed.

Remark: In classical Resolution we can always back-out from a dead-end. In this new approach the situation is analogous, in fact using $\mathcal{I}$-rules we can always reintroduce an element from $\mathcal{I}_{\mathcal{C}}$, the set of initial clause-words corresponding to the set of initial clauses, a clause-word many times anywhere in the proof.

Theorem 2.2.3 [3] *Let π be a resolution refutation in R^* of a set of clauses $\mathcal{C}$ in variables $x_1, \ldots, x_n$. Assume that π has k clauses. Then there exists a Σ_n^*-proof π' of LR from $\mathcal{I}_{\mathcal{C}}$ such that the number of steps k' in π' satisfies:*

$$k' < 2(kn) \ .$$

[3] Theorem 2.2.11 about width gives a more effective version of the simulation, estimating also the sizes of the lines of the rewriting proof.

Proof. Let a tree-like refutation π of $\mathcal{C}$ be fixed. For a clause D in π let $k(D)$ denotes the number of clauses in the subproof of π ending with D[4]. Then $k(D) = 1$ for initial clauses and $k(\{\}) = k$ for the end clause $\{\}$. If

$$\frac{D_1 \cup \{\ell\}\ D_2 \cup \{\bar{\ell}\}}{D_1 \cup D_2}$$

is an inference in π then, as π is tree-like,

$$k(D_1 \cup D_2) = k(D_1 \cup \{\ell\}) + k(D_2 \cup \{\bar{\ell}\}) + 1.$$

By induction on $k(D)$ we show that for any clause $D \in \pi$ there is π'_D, a derivation in Σ^*_n of w_D from $\mathcal{I}_\mathcal{C}$, such that the number of steps k'_D of π'_D satisfies:

$$k'_D < 2(k(D)n).$$

Taking for D the end-clause of π gives the theorem.

Basis Case: D is an initial clause in π. Then w_D is derived from $\mathcal{I}$ in one step using the $\mathcal{I}$-rules; w_D is the source-word of the derivation π'_D consisting of one step ($k'_D = 1$).
Induction Step: Assume D is in π derived from D_1 (containing ℓ) and D_2 (containing $\bar{\ell}$) by resolving ℓ. By induction assumption applied to D_1 there are a derivation π'_{D_1} of w_{D_1} and a derivation π'_{D_2} of w_{D_2} in Σ^*_n from $\mathcal{I}$ with

$$k'_{D_1} < 2(k(D_1)n)$$

and

$$k'_{D_2} < 2(k(D_2)n)$$

steps respectively. Construct a derivation π'_D in Σ^*_n from $\mathcal{I}$ as follows:

1. Initial part of π'_D is π'_{D_1}.
2. Then continue with derivation carrying w_{D_1} as the left-most clause-word of every step. This subderivation uses the same inferences as π'_{D_2} except for introducing the first line: instead of using a clause C for a source-word w_C as in π'_{D_2}, use $\mathcal{I}$-rule (ii) to infer from w_{D_1} the word $w_{D_1} w_C$.

[4] $k(D)$ is the size of the tree rooted at D.

3. After the steps (1) and (2) we have a derivation in Σ_n^* of $w_{D_1}w_{D_2}$. Now, use exchanging rules to move ℓ in w_{D_1} towards R and $\bar{\ell}$ in w_{D_2} towards L such that
$$\ell RL\bar{\ell}$$
becomes a subword. This proocess needs at most $2(n-1)$ application of exchanging rules.

4. Finally apply the elimination rules to delete the subword $\ell RL\bar{\ell}$, getting w_D.

The number of steps k'_D in this derivation is bounded above by:
$$k'_D \leq k'_{D_1} + k'_{D_2} + 2(n-1) + 1$$
As $k'_{D_1} < 2k(D_1)n$, $k'_{D_2} < 2k(D_2)n$ and $k(D) = 1 + k(D_1) + k(D_2)$, we also have $k' < 2k(D)n$.

□

Now, we want to establish that the simulation of R^* by Σ_n^* is sound, namely that any derivation of LR from $\mathcal{I}_\mathcal{C}$ in Σ_n^* can be transformed into a refutation of $\mathcal{C}$ in R^*. A small obvious technical lemma is the following:

Lemma 2.2.4 *Let $\mathcal{C}$ be a set of clauses and $\mathcal{I}_\mathcal{C}$ the corresponding set of clause-words. Then any derivation in Σ_n^* from $\mathcal{I}_\mathcal{C}$ contains only regular words.*

Theorem 2.2.5 *Let $\mathcal{C}$ be a set of clauses and $\mathcal{I}_\mathcal{C} = \{w_C : C \in \mathcal{C}\}$ be the corresponding set of clause-words. Assume that there is a derivation π' in Σ_n^* of LR from $\mathcal{I}_\mathcal{C}$. Then the set of clauses $\mathcal{C}$ is refutable in R^*. In fact, if π' has k' steps then there is a refutation π in R^* of $\mathcal{C}$ with $k \leq k'$ steps.*

Proof. Let π' be $w_1,\ldots, w_{k'}$. By induction on i, we prove that if $w_i = w_{D_1}\ldots w_{D_t}$, with D_j clauses (we know by Lemma 2.2.4 that w_i is a regular word), then there are derivations π_j, $j = 1,\ldots,t$, in R^* of D_j from $\mathcal{C}$ with $k(\pi_j)$ clauses each such that
$$\sum_{j=1}^{t} k(\pi_j) \leq i$$
For $i = k'$ this gives the theorem.

Basis Case: By definition the source-word w_1 is w_C for some $C \in \mathcal{C}$. Thus the claim holds for $i = 1$.

Induction Step: If w_{i+1} has been obtained by other than the elimination rules or the $\mathcal{I}$-rules it corresponds to the same set of clauses of w_i and there is nothing to prove. The elimination rule is simulated by the resolution rule and the $\mathcal{I}$-rules are initial clauses rule of R^*.

□

We can describe a direct way how to extract the derivation in R^* from the derivation in Σ_n^*, using the following procedure. The procedure has a derivation in Σ_n^* from $\mathcal{I}_\mathcal{C}$ as input and marks by † all occurrences of a clause-word that:

1. is not the source-word;
2. it was not derived by an $\mathcal{I}$-rule;
3. it was not derived by an elimination rule.

Then delete all clause-words marked by †, and replace each regular word $w_{D_1},\ldots, w_{D_t}$ that remains by a set of clauses $\{D_1,\ldots,D_t\}$.

The proof of the preceding theorem shows that the sequence of these sets of clauses contains not only the refutation in R^* but, in fact, also information what an algorithm needs to keep in memory in order to check the correctness of the refutation. Before we state a formal theorem we recall a relevant concept of space complexity of resolution derivations introduced in [13] and refined in [33]. We take the definition given by Esteban and Torán in [33] for tree-like proofs.

Definition 2.2.6 *Let $k \in \mathbb{N}$, we say that an unsatisfiable set of clauses $\mathcal{C}$ has a tree-like resolution refutation bounded by space k if there exists a sequence of clauses $\mathcal{C}_1,\ldots, \mathcal{C}_s$ such that $\mathcal{C}_1 \subseteq \mathcal{C}$, $\{\} \in \mathcal{C}_s$, in any $\mathcal{C}_i$ there are at most k clauses, and for each $i < s$, $\mathcal{C}_{i+1}$ is obtained from $\mathcal{C}_i$ by one of:*
(i) deleting some of its clauses,
(ii) adding the resolvent of two clauses of $\mathcal{C}_i$ and deleting the parent clauses,
(iii) adding some of the clauses of $\mathcal{C}$ (initial clauses).

The definition of space in tree-like resolution expresses the idea of considering list of clauses kept in memory during the refutation, with the particularity that when a clause is used to derive other clauses, it is removed from the memory.

Now, we state a theorem that follows immediately from the proofs of Theorems 2.2.3 and 2.2.5.

Theorem 2.2.7 *Let $\mathcal{C}$ be a set of clauses in variables $x_1,\ldots, x_n$. Assume that there is a refutation in R^* of $\mathcal{C}$ of space t. Then there is a derivation in Σ_n^* of LR from $\mathcal{I}_\mathcal{C}$ such that every line in it contains at most t clause-words.*

Assume, on the other hand, that there is a derivation in Σ_n^* of LR from $\mathcal{I}_\mathcal{C}$ in which all lines contain at most t clause-words. Then $\mathcal{C}$ has a refutation in R^* of space at most t.

In an effort to better understand Resolution another important measure has been introduced: resolution width. The notion of resolution width was made explicit by Galil in [34] and the importance of it was pointed out by Ben-Sasson and Wigderson in [8]. Roughly speaking, the width of a resolution is the largest number of literals in a clause used in the refutation to obtain {}. We recall the formal definition given in [34].

Definition 2.2.8 *Let $\mathcal{C}$ be a set of clauses, over variables $x_1,\ldots, x_n$. The width($\mathcal{C}$) is the number of literals in the largest clause in $\mathcal{C}$. If π is a resolution refutation of $\mathcal{C}$, width(π) is the number of literals in the largest clause in π. Let proofwidth($\mathcal{C}$) denote the minimum of width(π) over all refutations π of $\mathcal{C}$.*

Similar complexity characterizations can be given for Σ_n^*.

Definition 2.2.9 *Let $\mathcal{I}_\mathcal{C}$ be a set of regular word representing a set of clauses $\mathcal{C} = \{C_1,\ldots,C_m\}$. $w_\mathcal{C} = w_{C_1}\ldots w_{C_m}$. Then width($w_\mathcal{C}$) is the number of symbols in the largest clause-word $w_{C_j} \subseteq w_\mathcal{C}$, with $1 \leq j \leq m$. Thus, if π' is a proof in Σ_n^*, then width(π') is the number of symbols in the largest clause-word in π'. At the same manner can be defined the proof-width(π'), namely the minimum of width(π') over all proofs π' of LR from $\mathcal{I}_\mathcal{C}$.*

Thus, if $w_\mathcal{C} = Lx_1RLx_3x_5x_2\bar{x_2}x_4RLx_8\bar{x_1}\bar{x_5}R$, representing the set of clauses $\mathcal{C} = \{\{x_1\},\{x_3,x_5,x_2,\bar{x_2},x_4\},\{x_8,\bar{x_1},\bar{x_5}\}\}$, then width($w_\mathcal{C}$) = 7. Notice that it is easy to obtain from width($w_\mathcal{C}$) the width of the corresponding set of clauses $\mathcal{C}$. It is enough eliminate the upper case letters L, R from the designated subword with biggest size and then find the width of corresponding set clauses.

The following theorem is due to Ben-Sasson and Wigderson [8] and it relates size lower bounds on tree like resolution refutations to lower bounds on the width of resolution proofs:

Theorem 2.2.10 (Ben-Sasson Wigderson [8]) *Any tree-like resolution proof π of $\mathcal{C}$ of size k can be converted to one of width $\lceil \log_2(k) \rceil +$ width $\mathcal{C}$.*

Combining Theorem 2.2.10 with Theorems 2.2.3 and 2.2.5 yields a non-trivial estimate of the width of derivations in Σ_n^*.

Theorem 2.2.11 *Let $\mathcal{C}$ be a set of clauses and assume $w_0 =$width$(\mathcal{C})$. Assume there is a derivation in Σ_n^* of LR from $\mathcal{I}_\mathcal{C}$ with k' lines. Then there is a derivation in Σ_n^* of LR from $\mathcal{I}_\mathcal{C}$ of width bounded above by $\log(k') + w_0$.*

The previous theorem allow us to derive weak forms of automatizability as introduced in [2]. Recall that given a proof system P for a language L and a function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, we say that P is $f(n,k)$-automatizable if and only if there is an algorithm Π_P such that given any input x with $|x| = n$, if $x \in L$, then Π_P outputs a proof π in P of this fact in at most $f(n,k)$ steps, where k is the size of the shortest proof in P of the fact that $x \in L$.

Definition 2.2.12 *Σ_n^* is automatizable if and only if it is $f(n,k)$-automatizable for some function f that is $(n+k)^{O(1)}$.*

In this sense automatizable means that for Σ_n^* is possible to find a proof in polynomial time in the size of the smallest one. In fact, it follows by [8] that:

Theorem 2.2.13 *There is an algorithm Ω having the following properties:*

1. *On input $\mathcal{C}$ (an unsatisfiable set of clauses) it constructs a derivation in Σ_n^* of LR from $\mathcal{I}_\mathcal{C}$;*

2. *Ω runs in time $k^{O(\log(n))}$, where n is the number of variables and k is the number of clause-words in π'.*

We conclude this section with some observations on this restating of tree-like Resolution. The simulation by rewriting system is fairly straightforward for the tree-like case of R. As a by-product we obtain a different interpretation of several complexity measures such as the space and the width. This different interpretation gives us the possibility in the next section to characterize in essentially geometric terms these measures using planar diagrams. This extends to Resolution some geometric interpretations that were known only for the so called group-based proof systems considered in [47].

2.3 Planar Diagrams representing proofs in $\sum_n^*$

One motivation in [47] was the study of connections between the Dehn function[5], the word problem, various geometric constructions and propositional proof systems. In particular, with the exception of Cutting plane system and their generalizations to discretely ordered modules, these geometric situations are totally absents in proof complexity. Propositional proof complexity relies mostly on finite combinatorics, with connections to bounded arithmetic and computational complexity; to find geometric characterizations would be useful, as suggested in [47], because it could enlarge the set of methods that we have at our disposal. It should be mentioned that in the few cases in which this enlargement was achieved, using for example algebraic or model theoretic methods, not only lower bounds were obtained but also results of a structural type[6]. We propose a characterization using planar diagrams of the proofs in the semi-Thue system Σ_n^*. Finally, at the end of this section, we give an example of geometric construction (in three dimensions) of these proofs in Σ_n^*. In this case rules of construction are easily derived from the two-dimensional case.

A diagram representing a proof in Σ_n^* is a directed planar labeled graph, where every edge xy is labelled by a letter from A_n. The contour of every cell (face) is labelled by a regular word w. We describe how to construct the diagram starting with a proof π' in Σ_n^*. Let w_1 be the source-word and $w_2, \ldots, w_t$ be all the words introduced by $\mathcal{I}$-rules in this order. Note that the same clause-words may have several occurrences in the sequence. Each word w_i is a clause-word from $\mathcal{I}$. Let δ be a disc. We fix an origin (usually on the bottom of the disc), denoted by O. Let σ be the concatenation of all the words $w_1, w_2, \ldots, w_t$. Let q be the number of characters in σ; then we mark the disc δ with $q-1$ nodes (other than O). Thus we obtain a disc divided in q edges, since the origin gives the starting node. Then going counterclockwise from O we write for each edge e a character from σ. This means that if we had an original proof π' in which have been introduced the regular words

[5] A finitely presented group is given by a finite number of generators and a finite number of basic equations between words defined by the generators (the so called relators). Two words from generators (and their inverses) are equal in the group if their equivalence can be deduced using only the basic relators. The Dehn function measures the minimal number of deduction steps need. For details see [12] and [47].

[6] For example, lower bounds for algebraic systems like Nullstellensatz [5] or Polynomial Calculus (this system was originally introduced under the name Gröbner system, because of the well-known Gröbner basis algorithm) [19] also explicitely described the set of all "short provable"(i.e. in their degree) formulas by giving its basis as a linear space.

$L\bar{x}_2R$, $L\bar{x}_1R$ and the source-word is Lx_1x_2R then δ is divided in 10 labelled edges by 9 nodes plus the origin O.

Consider the following three rules of composition (Figure 2.2, Figure 2.3, Figure 2.4):

1. (*Join*) If there are four consecutive edges, xy, yz, zu and uv, labeled by x_i,R,L, and $\bar{x}_i$ (or labeled by $\bar{x}_i$,R,L, and x_i) then we can join the nodes x,v by a dashed edge.[7]

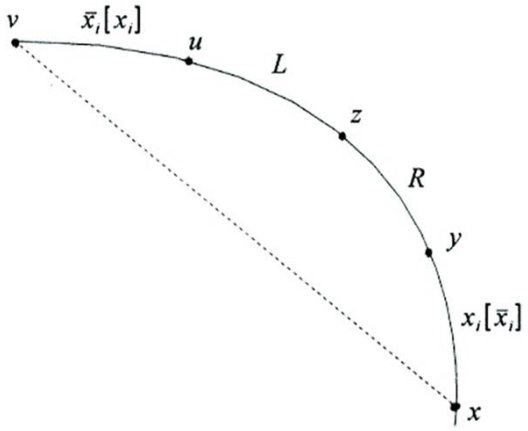

Figure 2.2: The rule Join.

2. (*Projection*) Let xv be two nodes connected by a dashed edge and let ve be the next edge, going counterclockwise. Then a new edge can be drawn connecting the nodes e, x. The new edge ex will be labeled by the same symbol labeling ve.[8]

3. (*Swap*) This rule allows to swap two edges which are consecutives. Let xy and yz be two consecutives edges, going counterclockwise, labeled by l_i and l_j. Then two new edges connecting x and z can be drawn such that the order of the labeling letters is reversed. This rule can be applied only to lowercase letters.[9]

[7]This is a simulation of Rule 1(a)[Rule 1(b)].
[8]This is a simulation of the contraction rule.
[9]This is a simulation of the exchanging rules.

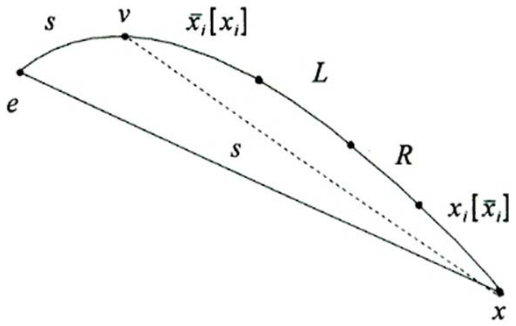

Figure 2.3: The rule Projection.

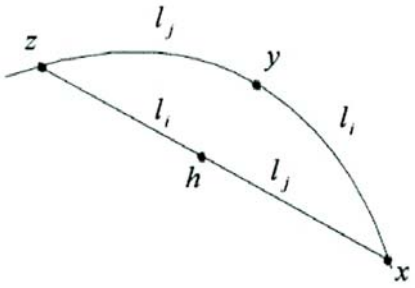

Figure 2.4: The rule Swap.

Definition 2.3.1 *A cell β is a region delimited by solid edges contained in δ; we denote it by $\beta \subseteq \delta$.*[10]

Definition 2.3.2 *The perimeter p of a given labeled disc δ is the number of edges on its border. The perimeter p of given cell β is the number of edges of β.*

Then in the example given in Figure 2.5, where $L\bar{x_1}R$, $L\bar{x_2}R$ and Lx_1x_2R are the initial clause-words, the perimeter of the disc is 10.

In case of application of the *Swap* rule the new cell contained in the disc has still perimeter 10. When we apply the *Join* rule by definition the cell has still perimeter 10; then the *Projection* rule gives a new cell with perimeter 6 (labeled by $Lx_1RL\bar{x_1}R$).

We search for configurations suitable for the application of the rule *Join*. We try to reduce the complexity of δ and the *Join* rule gives us the chance to use the *Projection* rule; this last one is the only rule (by creating a new edge with a new label) which reduces the perimeter of our disc. During search there are two possibilities:

[10]Note that by Definition 2.3.1 a cell can contain properly another cell. As a limit case the disc itself is a cell.

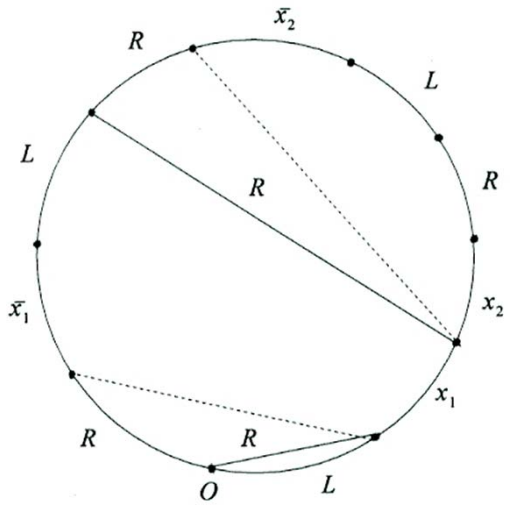

Figure 2.5: The proof given in section 2.

- The sequence of edges represented in Figure 2.1 exists.
- The sequence of edges represented in Figure 2.1 does not exist.

If the sequence exists we apply the rules *Join*; next we apply *Projection* and we consider the new cell which has perimeter $(p-4)$; in the second case we search, going counterclockwise, for edges labeled by letters x_i and $\bar{x}_i$; then we apply the *Swap* rule many times is necessary in order to have suitable sequences of edges on which we can apply *Join*. Then we consider the resulting cell with the labels attached and we write down, going counterclockwise from the origin O, the remaining letters. This is the last step of the proof π' in Σ_n^*. In order to discuss more extensively this construction, we need to introduce some additional definitions.

Definition 2.3.3 *A disc δ is regular if and only if the word attached to it is regular. A cell $\beta \subseteq \delta$ is regular if and only if the attached word to it is regular (going counterclockwise from the origin O).*

Lemma 2.3.4 *Let δ (β) be a regular disc (a regular cell). If one of the composition rules Join, Projection and Swap can be applied to δ (β), then the application creates a regular cell $\alpha \in \delta$ ($\alpha \in \beta$).*

Proof. For *Join* it is easy since no new edges are added to the disc; thus, after their application the disc (that is regular by hypothesis) it remains regular. The application of the *Swap* rule add edges connecting only edges

labeled by lowercase letters (this is the restriction on the rule) and do not involve movement of uppercase letters; then the new cell is still labeled by the same letters. So if by hypothesis the disc (or the previous cell) is regular then the cell β obtained by *Swap* is regular. When is applied the *Projection* rule has two cases.

1. The label of the edge is a lowercase letter.
2. The label of the edge is R.

In the first case, by hypothesis the disc (or the previous cell) is regular then is labeled by a regular word and by soundness the new cell is regular. In the second case a similar argument gives the claim.

□

Definition 2.3.5 *If a regular disc δ after finitely many steps ends with a cell labeled by LR (having $p = 2$) then the disc δ is called regular and complete.*

Theorem 2.3.6 *If there exists in Σ_n^* a proof*

$$\pi' : \mathcal{I} \vdash_{\Sigma_n^*} LR$$

such that $\mathcal{I} = \{w_{C_1}, w_{C_2}, \ldots, w_{C_t}\}$. Then there exists a regular disc δ labeled by words from $\mathcal{I}$, where for $1 \leq i \leq t$ each w_{C_i} may occur more than once, that is also complete.

Proof. Consider π'. By definition of proof in Σ_n^*, π' is a sequence of regular words $w_1, w_2, \ldots, w_k$. Start with a disc δ labeled by clause-words from $\mathcal{I}$ in the order they are introduced in π'. It is regular and will be regular at any stage, by Lemma 2.3.4. In order to show that the disc is also complete we must prove that after finitely many steps the disc ends with LR. It suffice to notice that the diagrams rules are used to simulate how the rewriting rules of Σ_n^* are applied in π' in an obvious way: *Swap* simulates the exchanging rules, *Join* and *Projection* the elimination rules.

□

Further, we have that

Theorem 2.3.7 *If there exists a regular and complete disc δ labelled by words from a non empty set $\mathcal{I}$ of clause-words, then there exists a proof*

$$\pi' : \mathcal{I} \vdash_{\Sigma_n^*} LR.$$

Proof. Let δ be a regular and complete disc whose border is labeled by clause-words from $\mathcal{I}$. We shall construct the Σ_n^*-proof of LR from $\mathcal{I}$ backwards. Let δ_1 be the cell labeled by LR with perimeter 2. Associate with it the word $w_1 = LR$. At any given stage we will have a subdisc δ_i of δ, a set $\mathcal{I}_i$ of clause-words and a regular word w_i such that:

1. $\mathcal{I}_i$ are the words occurring on the perimeter of δ_i;
2. clause subwords of w_i are in $\mathcal{I}_i$;
3. w_i,w_{i-1},...,w_1 is a valid Σ_n^*-derivation.

We write down every border going counterclockwise of each cell that we meet in the process. Every single line will be a regular word and to check if rules are applied correctly and they correspond to the rules in Σ_n^* is easy. The only point in this construction where we must be very careful is when we get the border of the disc. Recall that it collects all the application of the $\mathcal{I}$-rules. In order to get the original proof we must operate as follows; first we write down the complete border; then we consider the labels and we define properly the corresponding clause-words. Then we introduce step by step all the clause-words using the following procedure; if the number of clause-words is n then we obtain a sequence of n lines such that each $n-1$ line does not contain the last clause-word contained in the line n; at the end of this process we have w_1, the source-word of π'. This conclude our construction in the proof.

□

Combining the results obtained so far we can prove the following statement which give us the link between tree-like refutation proofs and regular and complete discs.

Theorem 2.3.8 *A set of clauses $\mathcal{C} = \{C_1, \ldots, C_m\}$ is unsatisfiable if and only if there exists a regular and complete disc δ such that its border is labelled by words from $\mathcal{I}_\mathcal{C}$.*

Definition 2.3.9 *Let δ be a complete and regular disc. The number of cells contained in δ is the size.*

Then the following theorem can be proved by inspection on π and δ.

Theorem 2.3.10 *Let k be the size of tree-like resolution refutation proof π. Let k' be the size of the corresponding regular and complete disc δ. Then $k' < k$.*

We may conceive the construction of our diagrams in a three-dimensional space. A nice representation in Euclidean solid geometry can be obtained. In this context, a proof is represented by a cylinder sectioned by polygons labeled with symbols from A_n. Thus, to represent a given proof[11] means to represent how the volume of the starting cylinder can be reduced. One example is given in Figure 2.6, where we consider the proof represented before in Figure 2.5. In this case the initial clause-words were Lx_1x_2R, $L\bar{x_1}R$ and $L\bar{x_2}R$.

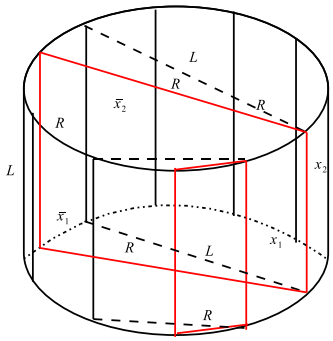

Figure 2.6: The same proof as in Figure 2.5 in three dimensions.

We conclude the section considering a more complicated proof with respect to that in Figure 2.5. Let $Lx_1x_2x_3R$ be the source-word. Let $L\bar{x_2}R$, $L\bar{x_1}R$, $L\bar{x_3}R$ be the words given by $\mathcal{I}$-rules. Having these informations we can construct the disc δ representing the proof in Σ_3^*, see Figure 2.7. Analyzing the resulting diagram we can obtain all the informations about the original proof. The number of steps in the proof is the number of cells labelled by regular words reading the diagram going counterclockwise with respect to O. The number of variables is given by the number of dashed lines contained in the disc. To find the size of the proof it is enough look at all the cells (starting from the one with smallest perimeter) and obtain all the regular words used in the proof and then find the clause-words. Thus we obtain the proof starting from the bottom, namely from the regular word LR. Then space and width can be defined similarly from the diagram in a geometric way. By the previous results linking Σ_n^* and R^*, we may find the estimate of complexity of resolution refutation proofs.

[11] It is easy to see how to translate the *Swap*, *Join* and *Projection* rules into the three-dimensional case.

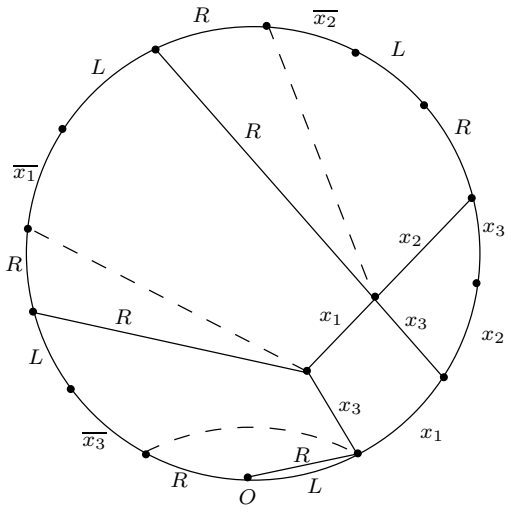

Figure 2.7: Proof of LR from $Lx_1x_2x_3R$, $L\bar{x_2}R$, $L\bar{x_1}R$ and $L\bar{x_3}R$.

2.4 Resolution and Σ_n: the dag-like case

We are going to outline in this section a construction of a rewriting system Σ_n that is equivalent to a general resolution R (i.e. no restriction to tree-like proofs) in the same way as Σ_n^* is to R^*.

Tree-like proofs have a very transparent structure. Once a clause is used in an inference it disappears; any other occurrence of the same clause has its own subproof and can be treated completely independently from other occurrences of the clause. In general, in dag-like proofs this is different. A clause can be used as a hypothesis in one inference but does not necessarily disappears: it can be reused as many times as needed.

Of course, we could simulate such proofs by transforming them first into tree-like proofs and then using the simulation outlined in the preceding section. However, this would blow-up the size of the proofs, sometimes exponentially. If we want to keep the simulation polynomial (computed by a polynomial time algorithm) we shall have to replace Σ_n^* by a more complicated rewriting system Σ_n.

The rewriting system Σ_n those definition we outline will be able to perform two tasks. The first is the **DOUBLING** process: having a regular word

$$uLvRw$$

with a clause-word LvR as a subword, rewrite it into

$$uLvRLvRw\Gamma$$

This will be used for saving an occurrence of the clause-word for possible future inferences.

The second process is **SHIFTING**: having a regular word

$$uLvRLwRe$$

with the two clause-words LvR and LwR as subwords rewrite it into

$$uLwRLvRe.$$

This procedure will enable us to move a clause-word which is a subword inside the regular word to a position where an elimination rule can be applied.

It is fairly obvious that if we augment Σ_n^* somehow to Σ_n such that the stronger rewriting system can perform the two procedures, the simulation from Theorem 2.2.3 can be extended to non-treelike proofs too. However, we want to construct such a simulation so that it is sound analogously to Theorem 2.2.5 about Σ_n^*. For this reason we have defined the system Σ_n not in as minimal way, using the smallest possible number alphabet and rewriting rules, but in a way in which the soundness is easy to verify. The drawback of this is that the systems has a bigger alphabet and the number of rewriting rules considerably increase.

Thus we explain the ideas how **DOUBLING** and **SHIFTING** are implemented and how the soundness is proved; we do not do it formally since full details are too tedious to follow. Now, we describe informally but with some precision how are the **DOUBLING** and **SHIFTING** procedures simulated so that the resulting system Σ_n properly simulates R (analogously with Theorem 2.2.5).

Let us consider the **DOUBLING**. The idea is to introduce in the alphabet colored versions of letters from A_n (four copies of different colors suffice). A clause-subword LvR of a regular word whose occurence is to be doubled is first colored green; the rules are formulated in a way that allows to color only one such subword (this use extra "super-script"symbol). Then green L is replaced by uncolored L and blue L. The rules allow to move blue letter right over all green and blue letters. Next the leftmost green letter (a literal unless v is the empty word) is replaced by its uncolored version followed by its blue version. The blue version is again moved as far right as possible over all green and blue letters, etc... At the end of this process the occurrence of LvR colored at the start green is now uncolored and it is followed by its blue

copy. Finally, the blue color is erased. In a very similar way can be treated the **SHIFTING** procedure.

We give below two examples to illustrate the procedures. In the first we formulate a possible set of rules that govern the **SHIFTING** procedure. In the second example we consider the **DOUBLING** procedure and we define a corresponding set of rewriting rules for it. In both examples we show how a step in a dag-like proof can be simulated by a string rewriting system.

Example 1. Let A'_n be the alphabet containing L, R, h, h^m, h^e and $x_1,\ldots,\ x_n$. We call h the 'head' symbol. The symbols h^m and h^e are called the 'head-moving' and the 'head-erasing' symbols respectively. If $a \in (A'_n \setminus h^e, h^m)$, then each a may have the following form: a, $\hat{a}$ and a^*. [12] Now, consider the rewriting system Σ^*_n extended by the following four sets of rewriting rules:

1. ***Structural rules***

   (Sa) $L \to hL$;

   (Sb) $Lh \to hL$;

   (Sc) $Rh \to hR$;

   (Sd) $hR \to Rh$;

   (Se) $hL \to Lh$;

   (Sf) $hx_i \to x_i h$;

   (Sg) $h\bar{x}_i \to \bar{x}_i h$;

   (Sh) $x_i h \to hx_i$;

   (Si) $\bar{x}_i h \to h\bar{x}_i$;

   (Sj) $h \to \Lambda$;

   These rules allow to introduce, erase and move through subwords the 'head' symbol.

2. ***Coloring rules***:

   (Ca) $hL \to \hat{L}\hat{h}$;

   (Cb) $\hat{h}x_i \to \hat{x}_i\hat{h}$;

   (Cc) $\hat{h}\bar{x}_i \to \hat{\bar{x}}_i\hat{h}$;

   (Cd) $\hat{h}R \to \hat{R}h^*$;

[12] We can interpret hat and star as green and blue.

(Ce) $h^*L \to L^*h^*$;

(Cf) $h^*x_i \to x_i^*h^*$;

(Cg) $h^*\bar{x_i} \to \bar{x_i}^*h^*$;

These rules are used to "color" (by hat and star) subwords.

3. ***Moving rules***:

(Ma) $h^*R \to R^*h^m$;

(Mb) $\hat{a}b^* \to b^*\hat{a}$, where $a, b \in \{L, R, x_i, x_j\}$ and $i, j = 1, \dots n$;

(Mc) $\hat{a}h^m \to h^m a$ where $a \neq L$;

These rules allow to move colored symbols.

4. ***Erasing rules***:

(Ea) $\hat{R}h^m \to h^e R$;

(Eb) $a^*h^e \to h^e a$, where $a \neq L$;

(Ec) $L^*h^e \to hL$;

(Ed) $\hat{a}h^e \to h^e a$;

These rules erase colors.

Now, consider the following step (i) of a sequence-like proof where clauses C_1 and C_3 are resolved to obtain the clause C_4 at the next step (i+1).

$$\begin{array}{ll} (i) & \underbrace{\{x_1x_2x_3\}}_{C_1}\underbrace{\{x_4x_5\}}_{C_2}\underbrace{\{\bar{x_1}x_6\}}_{C_3} \\ (i+1) & \underbrace{\{x_2x_3x_6\}}_{C_4}\underbrace{\{x_4x_5\}}_{C_2} \end{array}$$

Let $w_{C_1C_2C_3}$ be the regular word constituted of the clause-words w_{C_1}, w_{C_2} and w_{C_3} (apply Definition 2.2.1). Then, $w_{C_1C_2C_3}$ is

$$Lx_1x_2x_3RLx_4x_5RL\bar{x_1}x_6R$$

Now, we show how the step from (i) to (i+1) can be simulated using the rewriting rules of Σ_n^* extended by ***Structural***, ***Coloring***, ***Moving*** and ***Erasing*** rules.

We start our process by introducing the symbol h, thus:

$$hLx_1x_2x_3RLx_4x_5RL\bar{x_1}x_6R$$

by (1Sa);

$$\hat{L}\hat{h}x_1x_2x_3RLx_4x_5RL\bar{x_1}x_6R$$

by (2Ca);

$$\hat{L}\hat{x_1}\hat{h}x_2x_3RLx_4x_5RL\bar{x_1}x_6R$$

by (2Cb);

$$\hat{L}\hat{x_1}\hat{x_2}\hat{h}x_3RLx_4x_5RL\bar{x_1}x_6R$$

by (2Cb);

$$\hat{L}\hat{x_1}\hat{x_2}\hat{x_3}\hat{h}RLx_4x_5RL\bar{x_1}x_6R$$

by (2Cb);

$$\hat{L}\hat{x_1}\hat{x_2}\hat{x_3}\hat{R}h^*Lx_4x_5RL\bar{x_1}x_6R$$

by (2Cd);

$$\hat{L}\hat{x_1}\hat{x_2}\hat{x_3}\hat{R}L^*h^*x_4x_5RL\bar{x_1}x_6R$$

by (2Ce);

$$\hat{L}\hat{x_1}\hat{x_2}\hat{x_3}\hat{R}L^*x_4^*h^*x_5RL\bar{x_1}x_6R$$

by (2Cf);

$$\hat{L}\hat{x_1}\hat{x_2}\hat{x_3}\hat{R}L^*x_4^*x_5^*h^*RL\bar{x_1}x_6R$$

by (2Cf);

$$\hat{L}\hat{x_1}\hat{x_2}\hat{x_3}\hat{R}L^*x_4^*x_5^*R^*h^mL\bar{x_1}x_6R$$

by (3Ma);

Then apply the rule (3Mb) as many times we needed in order to obtain the following regular word:

$$L^*x_4^*x_5^*R^*\hat{L}\hat{x_1}\hat{x_2}\hat{x_3}\hat{R}h^mL\bar{x_1}x_6R.$$

Thus by application of (4Ea):

$$L^*x_4^*x_5^*R^*\hat{L}\hat{x_1}\hat{x_2}\hat{x_3}h^eRL\bar{x_1}x_6R$$

Now, rule (4Ed) can be applied until we obtain the following regular word:

$$L^*x_4^*x_5^*R^*h^eLx_1x_2x_3RL\bar{x_1}x_6R$$

Then we can apply (4Eb):

$$L^*x_4^*x_5^*h^eRLx_1x_2x_3RL\bar{x_1}x_6R$$

Rule (4Eb) can be applied until the symbol on the left hand side h^e is L;

$$L^*h^ex_4x_5RLx_1x_2x_3RL\bar{x_1}x_6R$$

Then by (4Ec) we obtain the regular word:

$$hLx_4x_5RLx_1x_2x_3RL\bar{x_1}x_6R$$

Then using the structural rule (1Sj) we obtain the regular regular word in which the first two clause-words are exchanged:

$$Lx_4x_5RLx_1x_2x_3RL\bar{x_1}x_6R.$$

In order to complete our simulation we apply the exchanging rules from Σ_n^* two times:

$$Lx_4x_5RLx_2x_3x_1RL\bar{x_1}x_6R$$

and then the elimination rule from Σ_n^* can be applied:

$$Lx_4x_5RLx_2x_3x_6R$$

Now, it is easy to check that the resulting regular word is composed by two clause-words w_{C_2} and w_{C_4}; these two words are by Definition 2.2.1 the clauses C_2 and C_4.

Example 2. Let A_n'' be the alphabet containing L, R, h, $h^\ddagger$ and $x_1,\ldots,x_n$. h is the 'head' symbol and $h^\ddagger$ is the 'head-stop' symbol. If $a \in (A_n' \backslash h, h^\ddagger)$, then each a may have the following form: a, $\breve{a}$ and $\tilde{a}$. [13] Now, consider the rewriting system Σ_n^* extended by the following two set of rewriting rules:

1. ***Structural rules***

   (Sa) $L \to hL$;

   (Sb) $Lh \to hL$;

   (Sc) $Rh \to hR$;

   (Sd) $hR \to Rh$;

   (Se)) $hL \to Lh$;

[13] We can interpret 'check' and 'tilde' as red and yellow.

(Sf) $hx_i \to x_i h$;

(Sg) $h\bar{x}_i \to \bar{x}_i h$;

(Sh) $x_i h \to h x_i$;

(Si) $\bar{x}_i h \to h \bar{x}_i$;

(Sj) $h \to \Lambda$;

These rules allow to introduce, erase and move through subwords the 'head' symbol.

2. ***Doubling rules***

(Da) $hL \to \check{L}\tilde{L}h$;

(Db) $ha \to \check{a}\tilde{a}h$, where $a \in (A_n \setminus \{L, R\})^*$;

(Dc) $hR \to \check{R}\tilde{R}h^{\ddagger}$;

(Dd) $\check{a}\tilde{b} \to \tilde{b}\check{a}$, where $a, b \in A_n$;

(De) $\tilde{a}h^{\ddagger} \to h^{\ddagger}a$, where $a \in A_n$;

(Df) $\check{a}h^{\ddagger} \to h^{\ddagger}a$, where $a \in A_n$;

(Dg) $h^{\ddagger}a \to ha$, where $a = L$.

These rules allow to make a copy of a subword.

Now, consider the following step (i) in a dag-like proof where the clause C_2 is used two times: first, to derive in the step (i+1) the clause C_4 and then in the step (i+2) to obtain C_5.

$$
\begin{array}{ll}
(i) & \underbrace{\{x_1 x_2\}}_{C_1}\underbrace{\{\bar{x}_2\}}_{C_2}\underbrace{\{\bar{x}_2 x_3\}}_{C_3} \\
(i+1) & \underbrace{\{x_1\}}_{C_4}\underbrace{\{\bar{x}_2\}}_{C_2}\underbrace{\{x_2 x_3\}}_{C_3} \\
(i+2) & \underbrace{\{x_1\}}_{C_4}\underbrace{\{x_3\}}_{C_5}
\end{array}
$$

Let $w_{C_1C_2C_3}$ be the regular word constituted of the clause-words w_{C_1}, w_{C_2} and w_{C_3} (apply Definition 2.2.1). Then, $w_{C_1C_2C_3}$ is

$$Lx_1x_2RL\bar{x}_2RLx_2x_3R.$$

We show how to simulate the previous derivation from (i) to (i+2) using the rewriting rules of Σ_n^* extended by ***Structural*** and ***Doubling*** rules. We introduce the symbol h by the rule (1Sa), then:

$$Lx_1x_2RhL\bar{x}_2RLx_2x_3R.$$

Then by (2Da):

$$Lx_1x_2R\check{L}\tilde{L}h\bar{x_2}RLx_2x_3R.$$

By rule (2Db) we obtain:

$$Lx_1x_2R\check{L}\tilde{L}\check{\bar{x}}_2\tilde{\bar{x}}_2hRLx_2x_3R.$$

Then we apply the rule (2Dc):

$$Lx_1x_2R\check{L}\tilde{L}\check{\bar{x}}_2\tilde{\bar{x}}_2\check{R}\tilde{R}h^{\ddagger}Lx_2x_3R.$$

After some applications of (2Dd), we obtain the following regular word:

$$Lx_1x_2R\tilde{L}\tilde{\bar{x}}_2\tilde{R}\check{L}\check{\bar{x}}_2\check{R}h^{\ddagger}Lx_2x_3R$$

then by (2Df) three times:

$$Lx_1x_2R\tilde{L}\tilde{\bar{x}}_2\tilde{R}h^{\ddagger}L\bar{x_2}RLx_2x_3R$$

and by (2De) three times we have:

$$Lx_1x_2Rh^{\ddagger}L\bar{x_2}RL\bar{x_2}RLx_2x_3R.$$

Then by (2Dg),

$$Lx_1x_2RhL\bar{x_2}RL\bar{x_2}RLx_2x_3R.$$

Thus by the structural rule (1Sj) we obtain:

$$Lx_1x_2RL\bar{x_2}RL\bar{x_2}RLx_2x_3R.$$

Then we can apply elimination rules from Σ_n^*

$$Lx_1RL\bar{x_2}RLx_2x_3R$$

and

$$Lx_1RLx_3R.$$

It is easy to verify that the simulation is correct.

Analyzing the previous examples it is fairly clear that it is possible to write down rules allowing the rewriting procedure described above. However, it cannot be enforced that the rules can be applied in a unique way. For example, anytime during the procedures we can insert few applications of the other rules from Σ_n^*. Or we can color the word and then uncolor a part and color again, etc ...

Thus instead of formulating particular rules it seems to us more convenient to formulate a general property of rules to be used in Σ_n that will guarantee the soundness, i.e. whenever there is a derivation in Σ_n of LR from $\mathcal{I}_\mathcal{C}$ then indeed $\mathcal{C}$ is unsatisfiable.

The property of Σ_n we want is:

There is a map associating to every word w that can occur in a Σ_n derivation from $\mathcal{I}_\mathcal{C}$ a set of clauses $\mathcal{D}_w$ such that

- Any truth assignment satisfying all clauses in $\mathcal{C}$ satisfies also all clauses in $\mathcal{D}_{w_\mathcal{C}}$.
- If a word v is derived in one step from a word u then any truth assignment satisfying all clauses in $\mathcal{D}_u$ satisfies also all clauses in $\mathcal{D}_v$.
- $\mathcal{D}_{LR}$ is unsatisfiable.

Such a map is constructed similarly as in the proof of Theorem 2.2.5. Notice that the colors allow us to reconstruct which literals, which may be being moved around, belong to the same clause-word. We skip the tedious details.

2.5 Some remarks

In this chapter we have shown that a propositional proof system such as Resolution can be interpreted as a string rewriting system; in particular as a Semi-Thue system. The interpretation Σ_n^* of the tree-like case has an interesting representation based on planar diagrams and they are similar to Van Kampen diagrams. This representation can be exploited more also to give a concrete geometrical representation, in Euclidean space. The system Σ_n^* is very elegant and his formal representation does not require too many rules and the proofs in Σ_n^* have a structure which is really transparent as the structure of proofs in R^*. Indeed, all the complexity measures study for R^* can be characterized in a very clear way also for the system Σ_n^*.

In the case of general resolution R things are less smooth, and this is because the proofs in R may have very complicated structure. Thus, in the simulation of them, using a string rewriting approach the number of rules substantially blow up and the resulting translation in Σ_n is much more tangled. Of course, in principle this is possible as we have outlined. We can simulate using string rewriting systems also general resolution and the gap between the proofs in R and in Σ_n is still polynomial, but it is not satisfactory in terms of its formal representation.

Chapter 3

Applications of Propositional Logic to Cellular Automata

In this chapter we consider the rewriting procedure applied in parallel and in a synchronous way. Perhaps the most natural way to think about this is to consider cellular automata. Cellular automata are dynamical systems that have been extensively studied as discrete models for natural systems. They can be described as large collections of simple objects locally interacting with each other. A d-dimensional cellular automaton consists of an infinite d-dimensional array of identical cells. Each cell is always in one state from a finite state set. The cells change their states synchronously in discrete time steps according to a local rule. The rule gives the new state of each cell as a function of the old states of some finitely many nearby cells, its neighbours. The automaton is homogeneous so that all its cells operate under the same local rule. The states of the cells in the array are described by a configuration. A configuration can be considered as the state of the whole array. The local rule of the automaton induces a global function that tells how each configuration is changed in one time step. [1] In literature cellular automata take various names according to the way they are used. They can be employed as computation models [31] or models of natural phenomena [72], but also as tessellations structures, iterative circuits [14], or iterative arrays [22]. The study of this computation model was initiated by von Neumann in the '40s [54], [55]. He introduced cellular automata as possible universal computing devices capable of mechanically reproducing themselves. Since that time cellular automata have also aquired some popularity as models for massively parallel computations.

The main reason why cellular automata have been extensively studied

[1] For surveys on cellular automata the interested reader can see [42], [43].

as discrete models for natural system is that they have several basic properties of the physical world: they are massively parallel, homogeneous and all interactions are local. Other physical properties such as reversibility and conservation laws can be programmed by selecting the local rule suitably. The main point is that cellular automata provide very simple models of complex natural systems encountered in physics and biology. As natural systems they consists of large numbers of very simple basic components that together produce the complex behaviour of the system. Then, in some sense, it is not surprising that several physical systems (spin systems, crystal growth process, lattice gasses, ...) have been modelled using these devices, see [72].

Probably the most popular of these automata is *Life* (or *Game of Life*), created by Conway in 1970, [3]. This cellular automaton operates on an infinite two-dimensional grid. Each cell is in one of two possible states, alive or dead, and interacts with its neighbours, which are the cells that are directly horizontally, vertically and diagonally adjacent. The eight neighbours form the so called Moore neighborhood [51] (see Figure 3.1 where two other neighborhoods widely used in literature are considered: the von Neumann [56] and the Smith neighborhood [68]). At every time step, each cell can

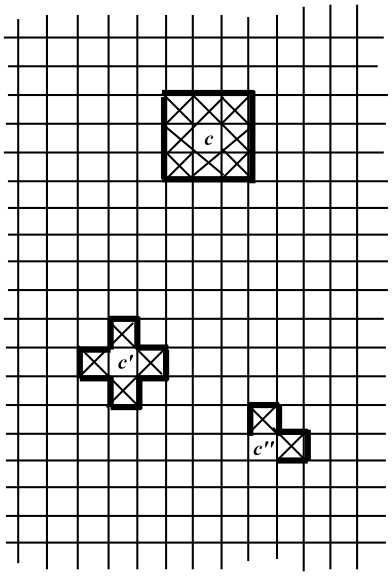

Figure 3.1: The Moore neighborhood of the cell c, the von Neumann neighborhood of the cell $c^{'}$ and the Smith neighborhood of the cell $c^{''}$.

change its state in a parallel and synchronous way according to the following local rules: (1) A cell that is dead at time t becomes alive at time $t+1$ if and only if three of its neighbours were alive at time t; (2) A cell that was alive

at time t will remain alive if and only if it had just two or three neighbours alive at time t; (3) A cell that is alive at time t and has four or more of its eight neighbours alive at t will be dead by time $t+1$; (4) A cell that has only one alive neighbour, or none at all, at time t, will be dead at time $t+1$.[2]

In this chapter we show how the study of propositional proof complexity and some of its techniques can be exploited in order to investigate cellular automata and their properties. The chapter is organized as follows. First, in Section 1, we present a formal definition of cellular automaton. In the same section we recall some of the most important results regarding cellular automata. In Section 2 we give a new proof of a foundamental theorem in the field, the Richardson theorem [63]. We use only compactness of propositional logic and the Craig interpolation theorem. In the same section we show how to apply feasible interpolation, discussed in Section 5 of the first chapter, to find description of inverse cellular automata. In Section 3 we solve two open problems formulated in [32]. The first can be stated as follows: consider finite bounded configurations and a reversible cellular automaton that is given by a "simple" algorithm. Is the inverse automaton given by a "simple"algorithm too? The second problem is the following: the injectivity problem of cellular automata on bounded size is $co\mathcal{NP}$-complete, [32]; does the result still hold if we consider instead of the size of the transition table, the smallest program (circuit) which computes its transition table?

3.1 Cellular Automata: definitions and some basic results

Formally a cellular automaton is an infinite lattice of finite automata, called cells. The cells are located at the integer lattice points of the d-dimensional Euclidean space. In general one can allow any Abelian group $\mathcal{G}$ in place of $\mathbb{Z}^d$. In particular, we may consider $(\mathbb{Z}/m)^d$, a toroidal space, where $\mathbb{Z}/m$ is the additive group of integers modulo m. In $\mathbb{Z}^d$ we identify the cells by their coordinates. This means that the cells are adressed by the elements of $\mathbb{Z}^d$.

Definition 3.1.1 *Let S be a finite set of states and $S \neq \emptyset$. A configuration of the cellular automaton is a function $c : \mathbb{Z}^d \to S$. The set of all configurations is denoted by $\boldsymbol{C}$.*

[2]As a game it can be seen as a zero-player game, i.e. a game whose evolution is determined by its initial state, needing no input from players.

The cells change their states synchronously at discrete time steps. Simply the next state of each cell depends on the current states of the neighboring cells according to an update rule. All the cells use the same rule, and the rule is applied to all cells in the same time. The neighboring cells may be the nearest cells surrounding the cell, but more general neighborhoods can be specified by giving the relative offsets of the neighbors.

Definition 3.1.2 *Let $N = (\vec{x_1}, ..., \vec{x_n})$ be a vector of n elements of $\mathbb{Z}^d$. Then the neighbors of a cell at location $\vec{x} \in \mathbb{Z}^d$ are the n cells at locations $\vec{x} + \vec{x_i}$, for $i = 1, ..., n$.*

The local transformation rule (transition function) is a function $f : S^n \to S$ where n is the size of the neighborhood. State $f(a_1, ..., a_n)$ is the new state of a cell at time $t + 1$ whose n neighbours were at states $a_1, ..., a_n$ at time t.

Definition 3.1.3 *A local transition function defines a global function $G : C \to C$ as follows,*

$$G(c)(\vec{x}) := f(c(\vec{x} + \vec{x_1}), \ldots, c(\vec{x} + \vec{x_n})).$$

The cellular automataton evolves from a starting configuration c^0 (at time 0), where the configuration c^{t+1} at time $(t + 1)$ is determined by c^t (at time t) by,

$$c^{t+1} := G(c^t).$$

Thus, cellular automata are dynamical systems that are updated locally and are homogeneous and discrete in time and space. Most frequently in literature cellular automata are specified by a quadruple

$$\mathbb{A} = (d, S, N, f),$$

where d is a positive integer, S is the set of states (finite), $N \in (\mathbb{Z}^d)^n$ is the neighborhood vector, and $f : S^n \to S$ is the local transformation rule.

Definition 3.1.4 *A cellular automaton $\mathbb{A}$ is said to be injective if and only if its global function $G_{\mathbb{A}}$ is one-to-one. A cellular automaton $\mathbb{A}$ is said to be surjective if and only if its global function $G_{\mathbb{A}}$ is onto. A cellular automaton $\mathbb{A}$ is bijective if its global function $G_{\mathbb{A}}$ is one-to-one and onto.*

Let $\mathbb{A}$ and $\mathbb{B}$ be cellular automata. Let $G_{\mathbb{A}}$ and $G_{\mathbb{B}}$ the two global functions. Suppose that d is the same for $\mathbb{A}$ and $\mathbb{B}$ and that they have in common

also S. We may compose $\mathbb{A}$ with $\mathbb{B}$ as follows: first run $\mathbb{A}$ and then run $\mathbb{B}$. Denoting the resulting cellular automaton by $\mathbb{B} \circ \mathbb{A}$ we have

$$G_{\mathbb{B}\circ\mathbb{A}} = G_{\mathbb{B}} \circ G_{\mathbb{A}}.$$

This composition can be formed effectively. If $N_{\mathbb{A}}$ and $N_{\mathbb{B}}$ are neighborhoods of $\mathbb{A}$ and $\mathbb{B}$, and $G_{\mathbb{A}}$ and $G_{\mathbb{B}}$ the global functions, then a neighborhood of $G_{\mathbb{B}} \circ G_{\mathbb{A}}$ consists of vectors $\vec{x} + \vec{y}$ for all $\vec{x} \in N_{\mathbb{A}}$ and $\vec{y} \in N_{\mathbb{B}}$.

To establish if two given cellular automata $\mathbb{A}$ and $\mathbb{B}$, with $G_{\mathbb{A}}$ and $G_{\mathbb{B}}$, are equivalent is decidable. In fact, if $N_{\mathbb{A}} = N_{\mathbb{B}}$ then the local transformation rules, $f_{\mathbb{A}}$ and $f_{\mathbb{B}}$, are identical. If $N_{\mathbb{A}} \neq N_{\mathbb{B}}$ then one can take $N_{\mathbb{A}} \cup N_{\mathbb{B}}$ and to test whether $\mathbb{A}$ and $\mathbb{B}$ agree on the expanded neighborhood.

The shift functions translate the configurations one cell down in one of the coordinate direction. Formally, for each dimension $i = 1, ..., d$ there is a corresponding shift function σ_i whose neighborhood contains only the unit coordinate vector $\vec{e_i}$ whose rule is the identity function id.[3] Translations are compositions of shift functions.

Often a particular state $q \in S$ is specified as a quiescent state (simulating empty cells). The state must be stable, i.e. $f(q, q, ..., q) = q$. A configuration c is said to be quiescent if all its cells are quiescent, $c(\bar{x}) = q$.

Definition 3.1.5 *A configuration $c \in S^{\mathbb{Z}^d}$ is finite if only a finite number of cells are non-quiescent, i.e. the set* [4]*,*

$$\{\vec{x} \in \mathbb{Z}^d |\ c(\vec{x}) \neq q\}$$

is finite.

Let $\mathbf{C}_F$ be the subset of $\mathbf{C}$ that contains only the finite configurations. Finite configurations remain finite in the evolution of the cellular automaton, because of the stability of q, hence the restriction G^F of G on the finite configurations is a function $G^F : \mathbf{C}_F \to \mathbf{C}_F$.

Definition 3.1.6 *A spatially periodic configuration is a configuration that is invariant under d linearly independent translations.*

This is equivalent to the existence of d positive integers $t_1, ..., t_d$ such that $c = \sigma_i^{t_i}(c)$ for every $i = 1, ..., d$. We denote the set of periodic configurations by $\mathbf{C}_P$. The restriction of G^P of G on the periodic configurations is hence a function $G^P : \mathbf{C}_P \to \mathbf{C}_P$.

[3] The one-dimensional shift function is the left shift $\sigma = \sigma_1$

[4] Usually in literature called the support.

Very often finite and periodic configurations are used in effective simulations of cellular automata on computers. Periodic configuarations are referred to as the periodic boundary conditions on a finite cellular array. For instance, when $d = 2$, this is equivalent to running the cellular automaton on a torus that is obtained by joining together the opposite sides of a rectangle. The relevant group is $(\mathbb{Z}/t_1) \times (\mathbb{Z}/t_2)$. This can be visualized as taping the left and right edges of the rectangle to form a tube, then taping the top and bottom edges of the tube to form a torus (doughnut shape), see Figure 3.2.

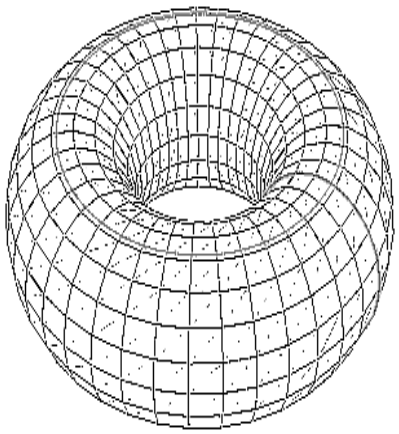

Figure 3.2: A toroidal arrangement: when one goes off the top, one comes in at the corresponding position on the bottom, and when one goes off the left, one comes in on the right.

Definition 3.1.7 *Let $\mathbb{A}$ be a cellular automaton. A configuration c is called a Garden of Eden configuration of $\mathbb{A}$, if c is not in the range of the global function $G_{\mathbb{A}}$.*

A basic property of the class of finite sets is the following: a function from a finite set into itself is injective if and only if the function is surjective. Surprisingly, when we deal with cellular automata this is also partially true, even if only in one direction: an injective cellular automaton is always surjective, but the converse does not hold. When we consider finite configurations the behaviour is more analogous to finite sets. The following theorem, a combination of two results proved by Moore [51] and by Myhill [52] respectively, points out exaclty this fact.

Definition 3.1.8 *A pattern α is a function $\alpha : P \to S$, where $P \subseteq \mathbb{Z}^d$ is a finite set. Pattern α agrees with a configuration c if and only if $c(\overline{x}) = \alpha(\overline{x})$ for all $\overline{x} \in P$.*

Theorem 3.1.9 (Moore [51], Myhill [52]) *Let $\mathbb{A}$ be a cellular automaton. Then $G^F_{\mathbb{A}}$ is injective if and only if $G^F_{\mathbb{A}}$ has the property that for any given pattern α there exists a configuration c in the range of $G^F_{\mathbb{A}}$ such that α agrees with c.*

The proof of the theorem is combinatorial and holds for any dimension d. The following theorem summarizes the situation regarding the injectivity and the surjectivity of cellular automata.

Theorem 3.1.10 (Richardson [63]) *Let $\mathbb{A}$ be a cellular a automaton. Let $G_{\mathbb{A}}$ be its global function and $G^F_{\mathbb{A}}$ be $G_{\mathbb{A}}$ restricted to the finite configurations. Then the following implications hold:*

1. *If $G_{\mathbb{A}}$ is one-to-one then $G^F_{\mathbb{A}}$ is onto.*
2. *If $G^F_{\mathbb{A}}$ is onto then $G^F_{\mathbb{A}}$ is one-to-one.*
3. *$G^F_{\mathbb{A}}$ is one-to-one if and only if $G_{\mathbb{A}}$ is onto.*

Definition 3.1.11 *A cellular automaton $\mathbb{A}$ with global function $G_{\mathbb{A}}$ is invertible if there exists a cellular automaton $\mathbb{B}$ with global function $G_{\mathbb{B}}$, such that $G_{\mathbb{B}} \circ G_{\mathbb{A}} = id$, where id is the identity function on $\boldsymbol{C}$.*

It is decidable whether two given cellular automata $\mathbb{A}$ and $\mathbb{B}$ are inverses of each other. This follows easily from the effectiveness of the composition and the decidability of the equivalence.

In 1972 Richardson proved the following quite remarkable and important theorem about cellular automata:[5]

Theorem 3.1.12 (Richardson [63]) *Let $\mathbb{A}$ be an injective cellular automaton. Then $\mathbb{A}$ is bijective and the inverse of $G_{\mathbb{A}}$, $G^{-1}_{\mathbb{A}}$, is the global function of a cellular automaton.*

In the next section we provide a new proof of Richardson's statement. The same year of Richardson's result Amoroso and Patt proved that:

[5]Recall that on finite configurations the global function may be onto and one-to-one even if the cellular automaton is not reversible.

Theorem 3.1.13 (Amoroso and Patt [1]) *Let $d = 1$. Then there exists an algorithm that determines, given a cellular automaton $\mathbb{A} = (1, S, N, f)$, if $\mathbb{A}$ is invertible or not.*

In the same paper they also provided an algorithm to determine if a given cellular automaton is surjective.[6] In higher spaces the problem of showing if a given cellular automaton is surjective or not, has been shown undecidable by Kari [41]. In the same paper Kari proved that the reversibility of cellular automata is undecidable too,

Theorem 3.1.14 (Kari [41]) *Let $d > 1$. Then there is no an algorithm that determines, given $\mathbb{A} = (d > 1, S, N, f)$, if $\mathbb{A}$ is invertible or not.*

The proof of Theorem 3.1.14 is based on the transformation of the tiling problem, which has been shown undecidable by Berger [9], into the invertibility problem on a suitable class of cellular automata.

3.2 A proof of the Richardson theorem via propositional logic

Richardson proved Theorem 3.1.12 by a topological argument plus the Garden of Eden theorem. Richardson's proof was non-constructive (it used compactness of a certain topological space) and our new proof is formally non-constructive too (we use compactness of propositional logic). This non-constructivity is unavoidable by Theorem 3.1.14. Nevertheless our proof offers a technical simplification: only basic logic is involved requiring straightforward formalism, and it allows us to apply an interpolation theorem. The compactness can be eliminated, and the proof made fully constructive, if we consider periodic configurations; considering $d = 2$ the working space becomes a torus. We will discuss this point below, after the proof of Theorem 3.2.1.

We shall concentrate on dimension $d = 2$ only and on the binary alphabet. This simplifies the notation but displays the idea of the proof in full generality.

Theorem 3.2.1 *Let $\mathbb{A}$ be a cellular automata over $\mathbb{Z}^2$ (with 0, 1 alphabet) whose global function $G_{\mathbb{A}}$ is injective. Then there is a cellular automata $\mathbb{B}$ (with 0, 1 alphabet) with global function $G_{\mathbb{B}}$ such that $G_{\mathbb{B}} \circ G_{\mathbb{A}} = id$.*

[6] Later Sutner designed elegant decision algorithms based on de Bruijn graphs, see [69].

Proof. For an n-tuple of $\mathbb{Z}^2$-points $N = ((u_1, v_1),\ldots,(u_n, v_n))$ defining the neighborhood of $\mathbb{A}$ denote by

$$(i, j) + N$$

the n-tuple $(i + u_1, j + v_1), \ldots, (i + u_n, j + v_n)$. For each i, j let $p_{i,j}$ be a propositional variable. Denote by $p_{(i,j)+N}$ the n-tuple of variables

$$p_{i+u_1,j+v_1}, \ldots, p_{i+u_n,j+v_n}.$$

Then the transformation function of $\mathbb{A}$ is a boolean function of n-variables

$$p^{t+1}_{(i,j)} = f(p^t_{(i,j)+N})$$

where the superscript t and $t + 1$ denote the discrete time. An array

$$(r_{(i,j)})_{(i,j)\in\mathbb{Z}^2}$$

(we shall skip the indices and write simply $\vec{r}$) of 0 and 1 describes the configurations obtained by $G_{\mathbb{A}}$ from an array

$$(p_{(i,j)})_{(i,j)\in\mathbb{Z}^2}$$

if and only if conditions

$$r_{(i,j)} = f(p_{(i,j)+N}),$$

for all (i, j) are satisfied.

Denote the infinite set of all these conditions $T_A(\vec{p}, \vec{r})$. We can define f by a CNF (or DNF) formula; hence we can think of $T_A(\vec{p}, \vec{r})$ as of a propositional theory consisting of clauses.

A basic observation is that the injectivity of $G_{\mathbb{A}}$ is equivalent to the fact that the theory

$$T(\vec{p}, \vec{r}) \cup T(\vec{q}, \vec{r})$$

(where $\vec{p}$, $\vec{q}$ and $\vec{r}$ are disjoints arrays of variables) logically implies all equivalences

$$p_{(i,j)} \equiv q_{(i,j)}$$

for all $(i, j) \in \mathbb{Z}^2$. As the theory remains unchanged if we replace all indices (i, j), by $(i, j) + (i_0, j_0)$, (any fixed $(i_0, j_0) \in \mathbb{Z}^2$) this is equivalent to the fact that

$$T(\vec{p}, \vec{r}) \cup T(\vec{q}, \vec{r})$$

implies that

$$p_{(0,0)} \equiv q_{(0,0)}.$$

This can be restated as follows:

$$T(\vec{p},\vec{r}) \cup \{p_{(0,0)}\} \cup T(\vec{q},\vec{r}) \cup \{\neg q_{(0,0)}\}$$

is unsatisfiable.

Now we can use the compactness theorem for propositional logic to deduce that there are finite theories

$$T_0(\vec{p},\vec{r}) \subseteq T(\vec{p},\vec{r})$$

and

$$T_0(\vec{q},\vec{r}) \subseteq T(\vec{q},\vec{r})$$

such that

$$T_0(\vec{p},\vec{r}) \cup \{p_{(0,0)}\} \cup T_0(\vec{q},\vec{r}) \cup \{\neg q_{(0,0)}\}$$

is unsatisfiable. (We may assume w.l.o.g. that these finite parts are identical up to renaming p's to q's.) By Craig's interpolation theorem there is a formula $I(\vec{r})$, such that:

$$T_0(\vec{p},\vec{r}) + p_{(0,0)} \vdash I(\vec{r})$$

and

$$I(\vec{r}) \vdash T_0(\vec{q},\vec{r}) \rightarrow q_{(0,0)}.$$

Although we write the whole array $\vec{r}$ in $I(\vec{r})$, the formula obviously contains only finitely many r variables (at most those appearing in $T_0(\vec{p},\vec{r})$).

Using deduction theorem four times we obtain

$$T_0(\vec{p},\vec{r}) \vdash p_{(0,0)} \rightarrow I(\vec{r})$$

and

$$T_0(\vec{q},\vec{r}) \vdash I(\vec{r}) \rightarrow q_{(0,0)}.$$

Renaming $\vec{q}$ to $\vec{p}$ the second fact gives

$$T_0(\vec{p},\vec{r}) \vdash I(\vec{r}) \rightarrow p_{(0,0)}$$

i.e. together

$$T_0(\vec{p},\vec{r}) \vdash I(\vec{r}) \equiv p_{(0,0)}.$$

In other words, the interpolant $I(\vec{r})$ computes the symbol of cell (0,0) in the configuration prior to $\vec{r}$, i.e. it defines the inverse to $\mathbb{A}$.

Let $M \subseteq \mathbb{Z}^2$ be the finite set of $(s,t) \in \mathbb{Z}^2$ such that $r_{(s,t)}$ appears in $I(\vec{r})$. Define the cellular automaton $\mathbb{B}$ as follows:

1. Alphabet is 0,1;
2. The neighborhood is M;
3. The transition function is given by $I(\vec{r})$, i.e.
$$p^{t+1}_{(i,j)} = I(p^t_{(i,j)+M}).$$

This concludes the proof.

□

Below we add some remarks about the previous proof.

1. The construction of the cellular automaton $\mathbb{B}$ has two key steps: (a) the use of the compactness theorem for propositional logic and (b) the application of Craig's interpolation. Compactness leads to a non-recusrive procedure while interpolation can be quite effective (see below).
2. The construction guarantees that
$$G_{\mathbb{B}}(G_{\mathbb{A}}(\vec{p})) = \vec{p}$$
but it does not - a priori - imply that also
$$G_{\mathbb{A}}(G_{\mathbb{B}}(\vec{r})) = \vec{r}.$$
That follows from the Garden of Eden theorem.
3. If the interpolant $I(\vec{r})$ contains M variables (i.e. the neighborood of $\mathbb{B}$ has size $|M|$) then the size of $\mathbb{B}$ (as defined in [32], see Definition 3.3.1 below) is $O(2^{|M|})$. This also bounds the size $|I|$ of any formulas defining the interpolant, but I could be in principle defined by a substantially smaller formula (e.g. of size $O(|M|)$.)

The same argument works for the version of the previous theorem with $(\mathbb{Z}/m)^2$ in place of $\mathbb{Z}^2$. In this case, already the starting theory $T(\vec{p},\vec{r})$ is finite: of size $O(m^2\, 2^n)$ where m^2 is the size of $(\mathbb{Z}/m)^2$ and $O(2^n)$ bound the sizes of CNFs/DNFs formulas for the transition function of $\mathbb{A}$. Hence we do not need to use the compactness theorem and we can apply the interpolation immediately. The interpolant may, in principle, be defined on all m^2 r-variables.

Given the finite subtheory $T_0(\vec{p},\vec{r})$ a method to find constructively the interpolant $I(\vec{r})$ is described below; hence we also give the description of the inverse cellular automaton $\mathbb{B}$. The construction is as follows:

1. Apply one of the "usual" automated theorem provers to verify that

$$T_0(\vec{p},\vec{r}) \cup \{p_{(0,0)}\} \cup T_0(\vec{q},\vec{r}) \cup \{\neg q_{(0,0)}\} \qquad (*)$$

   is unsatisfiable;

2. Extract from the run of the algorithm a resolution refutation π of clauses $(*)$. This can be done in polynomial time [4];

3. Apply Theorem 1.4.7 to get a polynomial time algorithm $W(\vec{r},\pi)$ that computes the interpolant $I(\vec{r})$.

Hence the circuit size of $I(\vec{r})$ is polynomial in the run-time of the algorithm from 1. Of course, we expect that in the worst case this is exponential in the size of $(*)$, but it may, in principle, be better than the exhaustive search.

3.3 Some complexity results

Durand [32] proved the first complexity results concerning a global property of cellular automata of dimension ≥ 2 (see Theorem 4.1.1). By Kari's result [40] the reversibility of a cellular automaton with $d \geq 2$ is not decidable. This implies that the inverse of a given cellular automaton cannot be found by an algorithm: its size can be greater than any computable function of the size of the reversible cellular automaton. Durand's result shows that even if we restrain the field of action of cellular automata (with $d = 2$) to finite configuration bounded in size, it is still very difficult to prove that the cellular automaton is invertible or not: the set of cellular automata invertible on finite configurations is *co*$\mathcal{NP}$-complete (see below). In [32] it is assumed that the size of a cellular automaton corresponds to the size of the table of its local function and of the size of its neighborhood. More precisely:

Definition 3.3.1 *If s is the number of states of a cellular automaton $\mathbb{A}$ and $N = (x_1, ..., x_n)$ then the size of a string necessary to code the table of the local function plus the vector N of $\mathbb{A}$ is $s^n \cdot \log(s) + o(s^n \cdot \log(s))$.*

Durand [32] proved that the decision problem concerning invertibility of cellular automata of dimension 2 belongs to the class of *co*$\mathcal{NP}$-hard problems or to the class of *co*$\mathcal{NP}$-complete problems if some bound is introduced on the size of the finite configurations considered.[7] For the *co*$\mathcal{NP}$-completeness

[7] Notice that result is obtained for a 2-dimensional cellular automata with von Neumann neighborhood, see Fig. 3.1.

we assume that the size of the neighborhood is lower than the size of the transition table of the cellular automaton, i.e. $\forall x \in N$, $|x| \leq s^n$.

Now, consider the following problem:

PROBLEM *(CA-FINITE-INJECTIVE):*
Instance: A 2-dimensional cellular automaton $\mathbb{A}$ with von Neumann neighborhood. Two integers p and q less than the size of $\mathbb{A}$.
Question: Is $\mathbb{A}$ injective when restricted to all finite configurations $\leq p \times q$?

The theorem below[8] is the main result in [32],

Theorem 3.3.2 (Durand [32]) *The problem CA-FINITE-INJECTIVE is co$\mathcal{NP}$-complete.*

If one drops the restriction on the bound of the size of the neighborhood then a proof of the co$\mathcal{NP}$-hardness of *CA-INFINITE-INJECTIVE* can be obtained; for more details on this the reader can see [32].

What is assumed in the previous result is basically that the size of the representation of a cellular automaton corresponds to the size of its transition table. Durand [32] asked if the co$\mathcal{NP}$-completeness result can be true also if we define the size of a cellular automaton as the length of the smallest program (circuit) which computes its transition table. A second question formulated in [32] is the following: suppose that we have an invertible cellular automaton given by a simple algorithm and that we restrict ourself to finite bounded configurations. Then is the inverse given by a simple algorithm too? In this section we give answers to both questions. We start with the latter problem first.

For succinctness we do it on $\mathbb{Z}^1$; it is fairly simple to get similar examples for $\mathbb{Z}^2$ or $(\mathbb{Z}/m)^2$.

Assume we have a boolean function $f : \{0,1\}^n \to \{0,1\}^n$ having the following properties:

1. f is a permutation;
2. f is computed by a polynomial size circuit.
3. The inverse function f^{-1} requires an exponential size circuit, $\exp(\Omega(n))$.

[8] For a discussion on this theorem with the idea behind the proof, see Chapter 4, Section 1.

For example, if f were a one-way permutation (e.g. conjecturally based on factoring or discrete logarithm) then it has the properties. Now define a cellular automaton $\mathbb{A}_f$ as follows:

1. Alphabet: 0, 1, #.

2. Neighborhood of $i \in \mathbb{Z}$:

$$N = \langle i-n, i-n+1, \ldots, i, i+1, \ldots, i+n \rangle$$

   i.e. $|N| = 2n+1$.

3. Transition function:

   (i) $p_i^t = \# \rightarrow p_i^{t+1} = \#$

   (ii) If $p_i^t \in \{0,1\}$ and there are j, k such that:

   (a) $j < i < k$ and $k - j = n + 1$;

   (b) $p_j^t = p_k^t = \#$

   (c) $p_r^t \in \{0,1\}$ for $r = j+1$, $j+2$,...,i,..., $k-1$

   define

$$p_i^{t+1} = (f(p_{j+1}^t, \ldots, p_{k-1}^t))_i$$

   where $(f(p_{j+1}^t, \ldots, p_{k-1}^t))_i$ is the i-th bit of $f(p_{j+1}^t, \ldots, p_{k-1}^t)$.

   (iii) If $p_i^t \in \{0,1\}$ and there are no j, k satisfying (ii) then put

$$p_i^{t+1} = p_i^t.$$

The informal description of the automaton $\mathbb{A}_f$ can be summarized as follows: every $0-1$ segment between two consecutives #'s that does not have the lenght exactly n is left unchanged. Segments of length n are trasformed according to the permutation f.

The inverse automaton $\mathbb{B}$ is defined analogously using f^{-1} in place of f ($\mathbb{B} = \mathbb{A}_{f^{-1}}$).

Theorem 3.3.3 *Assume that $f : 2^n \rightarrow 2^n$ is a permutation computable by a size poly(n) circuit such that any circuit computing the inverse function f^{-1} must have size at least $\exp(n^{\Omega(1)})$. Then the cellular automaton $\mathbb{A}_f$ is invertible but has an exponentially smaller circuit-size than its inverse cellular automaton.*

Proof. By the construction the inverse cellular automaton is $\mathbb{B} = \mathbb{A}_g$ where $g = f^{-1}$. That is, the transition table of $\mathbb{B}$ essentially defines the boolean function f^{-1}. Hence by hypothesis its circuit-size is exponential in n, while $\mathbb{A}_f$ has circuit-size poly(n).

□

Remark: The hypothesis of Theorem 3.3.3 follows from the existence of cryptographic one-way functions. In particular, it follows from the exponential hardness of factoring or of discrete logarithm.

Theorem 3.3.3 solves negatively one of the open problem formulated by Durand [32] that we have described above: a very "simple" algorithm giving a reversible cellular automaton (even if restricted to finite configurations) can have an inverse which is given by an algorithm which is exponentially bigger and then not "simple".[9]

Now we answer the other open problem formulated in [32]. The problem asks about *co$\mathcal{NP}$*-completeness of the injectivity of cellular automata when it is represented by a program (circuit) rather than by a transition table.

Consider the following problem:

PROBLEM (P1):
Input: A circuit $C(x_1, ..., x_n)$ defining the transition table function of $0-1$ cellular automaton $\mathbb{A}_C$ with a neighborhood N of size $|N| = n$.
*Question:*Is $\mathbb{A}_C$ injective on $\mathbb{Z}^1$?

Theorem 3.3.4 *Problem (P1) is co$\mathcal{NP}$-hard.*

Proof. We shall describe a polynomial reduction from $TAUT$ to *(P1)*. Let $\phi(x_1, ..., x_n)$ be a propositional formula. Let the alphabet be 0,1 and the neighborhood N be $\langle 0, ..., n\rangle$. Now define the cellular automaton $\mathbb{A}$ as follows:

$$p_i^{t+1} := \begin{cases} p_i^t, & \text{if} \quad \phi(p_{i+1}^t, ..., p_{i+n}^t) \\ 0, & \text{otherwise.} \end{cases}$$

Clearly the circuit defining $\mathbb{A}$ is

$$p_i^t \wedge \phi(p_{i+1}^t, ..., p_{i+n}^t)$$

[9]Where "simple" algorithm means polynomial time algorithm.

and has size $O(|\phi|)$. This means that the map $\phi \to \mathbb{A}$ is polynomial time.

If $\phi \in TAUT$ then always $p_i^{t+1} = p_i^t$. In this case $\mathbb{A}$ is a cellular automaton doing nothing, i.e. its global map is the identity and, in particular, it is invertible. Assume $\phi \notin TAUT$. We need to construct two different configurations mapped by $\mathbb{A}$ to the same configuration. Let $i_0 \geq 1$ be minimal i_0 such that there is a truth assignment $\bar{a} = (a_1, ..., a_n) \in \{0,1\}^n$ satisfying:

(i) $\neg\phi(\bar{a})$;

(ii) $a_{i_0} = \ldots = a_n = 0$;

(iii) either $i_0 = 1$ or $a_{i_0-1} = 1$.

Informally, $\bar{a}$ has the longest segment of 0's on the right hand side that is possible for assignments falsifying ϕ. Define two configurations (see Fig. 3.3):

$$C_0 : \langle ..., 0, 0, 0, a_1, ..., a_n, 0, 0, ... \rangle$$

and

$$C_1 : \langle ..., 0, 0, 1, a_1, ..., a_n, 0, 0, ... \rangle.$$

The two configurations differ only in the position 0. Easily the theorem follows from the following lemma.

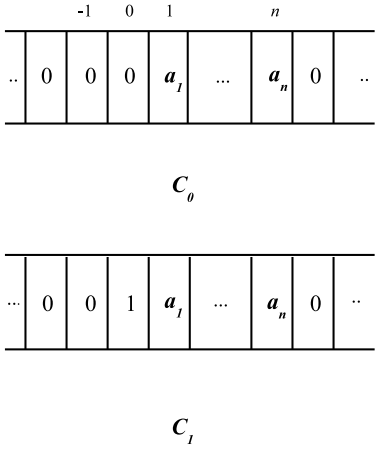

Figure 3.3: The configurations C_0 and C_1.

Lemma 3.3.5 *The two configurations C_0 and C_1 are both mapped by $\mathbb{A}$ to C_0.*

Proof. By the definition of $\mathbb{A}$, all 0's in C_0, C_1 stay 0. Also $\bar{a}$ remains the same: either it is $\vec{0}$ or also the n-string to the right side from a square $i = 1$, ..., n, i.e. q_{i+1}, ..., q_{i+n} contains more zeros on the right side than $\bar{a}$ does: this would contradict the definition of $\bar{a}$.

Finally, the 1 in the 0-square of C_1 changes to 0, as $\neg\phi(\bar{a})$. This proves the lemma.

□

Hence Theorem 3.3.4 follows.

□

Now, consider a finite modification of the problem *(P1)*:

PROBLEM (P2):
Input:

1. $\mathbb{A}_C$ as in *(P1)*;
2. $1^{(m)}$, such that $m > n$. (Notice that this condition implies that $\mathbb{A}_C$ is well-defined on $(\mathbb{Z}/m)$ tori.)

Question: Is $\mathbb{A}_C$ injective on $(\mathbb{Z}/m)$?

Theorem 3.3.6 *(P2) is co$\mathcal{NP}$-complete.*

Proof. That *(P2)* is in *co$\mathcal{NP}$* is obvious. The *co$\mathcal{NP}$*-hardness of *(P2)* is shown exactly as of *(P1)*.

□

Chapter 4

Inverse Cellular Automata as propositional proofs

In this last chapter we combine the Richardson theorem with the $co\mathcal{NP}$-completeness result of Durand [32] and we define a new type of a proof system $\mathbb{P}_{\mathbb{CA}}$. This proof system $\mathbb{P}_{\mathbb{CA}}$ is a proof system for the membership in a $co\mathcal{NP}$-complete language $\mathcal{L}_D$ (to be specified below). As the set $TAUT$ of propositional tautologies can be polynomially reduced to $\mathcal{L}_D$, $\mathbb{P}_{\mathbb{CA}}$ can be thought of also as a propositional proof system in the sense on Cook and Reckhow [25].

This last chapter is organized as follows. In Section 1 we briefly recall the proof of Durand's Theorem and we prepare the ground for the formulation of a proof system in terms of cellular automata. In particular we show that: there is polynomial time algorithm having a cellular automaton $\mathbb{A}$ with von Neumann neighborhood and a cellular automaton $\mathbb{B}$ with an arbitrary neighborhood and with the same alphabet of $\mathbb{A}$ as inputs, it can decide whether or not $\mathbb{B}$ is an inverse to $\mathbb{A}$. Then, in Section 2 we define a "mathematical" proof system for $\mathcal{L}_D$ satisfiyng the Cook and Reckhow definition [25]. At last, Section 3 has some concluding remarks when we consider our new proof system with respect to polynomial simulations.

4.1 Durand's Theorem

In the fourth section of Chapter 3 we considered Durand's result of $co\mathcal{NP}$-completeness. We recall the problem and we sketch the idea behind the proof of the theorem.

The problem that has been called (CA-FINITE-INJECTIVE) and it goes as follows:

PROBLEM *(CA-FINITE-INJECTIVE):*
Instance: A 2-dimensional cellular automaton $\mathbb{A}$ with von Neumann neighborhood. Two integers p and q smaller than the size of $\mathbb{A}$.
Question: Is $\mathbb{A}$ injective when restricted to all finite configurations $\leq p \times q$?

We shall also use the name CA-FINITE-INJECTIVE for the language of inputs with an affirmative answer. The main result in [32] is the following,

Theorem 4.1.1 (Durand [32]) *The problem CA-FINITE-INJECTIVE is co$\mathcal{NP}$-complete.*

The proof is based on tiling. A tile is a square and its sides are colored. The colors belong to a finite set called the color set. All tiles have the same size. A plane tiling is valid if and only if all pairs of adjacent sides have the same color.[1] A finite tiling can be defined as follows. We assume that the set of colors contains a special "blank color" and that the set of tiles contains a ''blank tile" (a tile whose sides are blank.) A finite tiling is an almost everywhere blank tiling of the plane. If there exist two integers i and j such that all the nonblank tiles of the tiling are located inside a square of size $i \times j$, then we say that the size of the finite tiling is lower than $i \times j$. Inside the $i \times j$ square, there can be blank and nonblank tiles. If we have at least one nonblank tile, then the tiling is called nontrivial.

Durand in its proof introduces a special tile set δ. The sides contain a color ("blank", "border", "odd", "even", or "the-end"), a label (N, S, E, W, $N+$, $S+$, $E+$ $W+$, or ω), and possibly an arrow. A tiling is valid with respect to δ if and only if all pairs of adjacent sides have the same color, the same label, and for each arrow of the plane, its head points out the tail of an arrow in the adjacent cell. A basic rectangle of size $p \times q$ is a finite valid tiling of the plane of size $p \times q$ with no size labeled "blank" or "border" inside the rectangle.

Then, given a finite set of colors B with a blank color and a collection $\tau \in B^4$ of tiles including a blank tile, Durand constructs a cellular automaton $\mathbb{A}_\tau$ and proves the following basic theorem which provides a link between tilings and cellular automata:

Theorem 4.1.2 (Durand [32]) *Let $n \geq 3$ be an integer and τ be a set of tiles. The cellular automaton $\mathbb{A}_\tau$ is not injective restricted to finite configurations of size smaller than $2n \times 2n$ if and only if the tile set τ can*

[1] Notice that is not allowed to turn tiles.

be used to form a finite nontrivial tiling of the plane of size smaller than $(2n-4) \times (2n-4)$.

Then using Theorem 4.1.2 he proves that ***PROBLEM*** *(CA-FINITE-INJECTIVE)* is *co*$\mathcal{NP}$-complete, i.e. Theorem 4.1.1.

Now we reformulate Durand's problem a bit in that we consider cellular automata operating on $(\mathbb{Z}/m)^2$ rather than on finite rectangles in $\mathbb{Z}^2$. We are replacing rectangles in $\mathbb{Z}^2$ by $(\mathbb{Z}/m)^2$ in order to be compatible with our treatement of Richardson's theorem given in the third chapter.

Consider a variant of the problem *CA-FINITE-INJECTIVE* in which the cellular automata operate on $(\mathbb{Z}/m)^2$ rather than on "finite configurations". We call this problem ***PROBLEM****(CA-TORI-INJECTIVE):*

PROBLEM*(CA-TORI-INJECTIVE):*
Instance: A 2-dimensional cellular automaton $\mathbb{A}$ with von Neumann neighborhood and $m \geq 3$, m is smaller than the size of $\mathbb{A}$.
Question: Is $\mathbb{A}$ injective when restricted to $(\mathbb{Z}/m)^2$?

Definition 4.1.3 *The language* $\mathcal{L}_D$ *is the set of pairs* $(m, \mathbb{A})$ *for which the* **PROBLEM**(CA-TORI-INJECTIVE) *has an affirmative answer.*

In terms of languages the problem above will be called $\mathcal{L}_D$. Thus, of course Theorem 4.1.1 by Durand can be simply stated as follows:

Theorem 4.1.4 $\mathcal{L}_D$ *is a co*$\mathcal{NP}$*-complete language.*

Lemma 4.1.5 *There is a polynomial time algorithm that on the two inputs:*

1. *a cellular automaton* $\mathbb{A}$ *with von Neumann neighborhood;*
2. *a cellular automaton* $\mathbb{B}$ *with an arbitrary neighborhood and the same alphabet as* $\mathbb{A}$*,*

decides whether or not $\mathbb{B}$ *is an inverse to* $\mathbb{A}$.

Proof. The automata $\mathbb{A}$ and $\mathbb{B}$ are presented to the algorithm by the tables of their local functions, see Definition 3.3.1. Assume that the alphabet of $\mathbb{A}$ and $\mathbb{B}$ has S symbols and that the size of $\mathbb{B}$ neighborhood is N. Hence the size of $\mathbb{A}$ and $\mathbb{B}$ are $O(S^5 \cdot \log(S))$ and $O(S^N \cdot \log(S))$, respectively.

To evaluate a cell (i, j) in $\mathbb{B} \circ \mathbb{A}$ we need to look at a von Neumann neighborhood of all N points in the neighborhood of (i, j) in $\mathbb{B}$, i. e. on at most $\leq 5N$ cells. Considering all the possible $\leq S^{5N}$ patterns on these cells yields in a list of all possible patterns ($\leq S^N$) on the neighborhood of (i, j) in $\mathbb{B}$, after the action of $\mathbb{A}$. Then we check that in all these patterns $\mathbb{B}$ produces in the cell (i, j) the original symbol.

The time they need is bounded above by $O(S^{5N} \cdot (N \cdot S^5 \cdot \log(S)) \cdot (S^N \log(S)) = S^{O(N)})$, where S^{5N} bounds the number of patterns to check, $N \cdot S^5 \cdot \log(S)$ bounds the time need to compute the pattern on the neighborhood of (i, j) in $\mathbb{B}$ after the action of $\mathbb{A}$ (for any fixed pattern), and $S^N \cdot \log(S)$ bounds the time need to compute the symbol of (i, j) after the action of $\mathbb{B}$. However, the quantity $S^{O(N)}$ is polynomial in terms of the size of $\mathbb{B}$, i.e. the algorithm is polynomial time.

□

Let us remark that the restriction on $\mathbb{A}$ to a von Neumann neighborhood is essential. If $\mathbb{A}$ was allowed to have an arbitrarily neighborhood M, then the algorithm would need time $S^{O(M \cdot N)}$ which is only quasi-polynomial in the sizes $O(S^M \cdot \log(S))$ and $O(S^N \cdot \log(S))$ of the inputs $\mathbb{A}$ and $\mathbb{B}$.

4.2 A proof system based on cellular automata

In this section we define a new proof system $\mathbb{P}_{\mathbb{CA}}$ based on cellular automata. As far as we know this is the first proof system based on cellular automata.

Definition 4.2.1 *$\mathbb{P}_{\mathbb{CA}}$ is a proof system for the language $\mathcal{L}_D$. A $\mathbb{P}_{\mathbb{CA}}$ proof for the pair $(m, \mathbb{A}) \in \mathcal{L}_D$ is cellular automaton $\mathbb{B}$ such that:*

1. *$\mathbb{B}$ has the same alphabet as $\mathbb{A}$;*
2. *$\mathbb{B}$ is inverse to $\mathbb{A}$.*

Lemma 4.2.2 *$\mathbb{P}_{\mathbb{CA}}$ is a proof system for the language $\mathcal{L}_D$.*

Proof. If $\mathbb{A} \in \mathcal{L}_D$, then a suitable cellular automaton $\mathbb{B}$ exists by Richardson's theorem, Theorem 3.2.1.[2] On the other hand the existence of $\mathbb{B}$ implies

[2]See Chapter 3, section 3, p. 62

that the cellular automaton $\mathbb{A}$ is injective, i.e. $\mathbb{A} \in \mathcal{L}_D$. Hence $\mathbb{P}_{\mathbb{CA}}$ is complete and sound.

Finally, the provability relation is polynomial time decidable by Lemma 4.1.5.

□

The statement $\mathbb{A} \in \mathcal{L}_D$ can be expressed in a propositional way, same as in our proof of Richardson's theorem. In particular, a proof of $\mathbb{A} \in \mathcal{L}_D$ is a proof of the unsatisfiability of the formula[3]:

$$T_0(\vec{p}, \vec{r}) \cup \{p_{(0,0)}\} \cup T_0(\vec{q}, \vec{r}) \cup \{\neg q_{(0,0)}\}$$

Hence any propositional proof system Q can be thought of also as a proof system for $\mathcal{L}_D$: a proof is a proof in Q of this formula.

We may observe at this place - reformulating the remark at the end of Section 3.2 - that having in particular a resolution proof of the formula gives us at least a circuit that describes the transition function of the inverse cellular automaton and whose size is polynomial in the size of the resolution proof: feasible interpolation (see Theorem 1.4.7) allows to extract a circuit computing the interpolant and the interpolant defines the transition function.

We remark that this leads to an interesting question about feasible interpolation. The size of the inverse automaton $\mathbb{B}$ is $O(S^N \log(S))$ where S is the size of the alphabet (common with the cellular automaton $\mathbb{A}$) and N is the size of the neighborhood of the inverse cellular automaton $\mathbb{B}$. Hence it is the quantity N that we would like to estimate. For this it would be very useful to have an estimate on the number of atoms the interpolant (produced by the feasible interpolation method or by any other specific method) depends on.

4.3 Some remarks

The main problems which remain open from this section are the followings: can we establish some polynomial simulation between $\mathbb{P}_{\mathbb{CA}}$ and some existing proof system such as Resolution? The investigation of this problem is hampered by the convoluted proof of Durand's theorem; a good place where to start thus would be to find a simple (or at least a simpler) proof of the latter. It would be desirable to have a proof which involves propositional logic, as

[3]See top page 66, the formula denoted by (*).

our proof of Richardson's theorem given in the second section of Chapter 3, since this could give us an elegant and unified framework.

Having such a simplified proof one could use the well-known relation between bounded arithmetic and proof systems (see [45]) and attempt to prove the soundness of $\mathbb{P}_{\mathbb{CA}}$ in the theory corresponding to R. Such a soundness proof would imply polynomial simulation of $\mathbb{P}_{\mathbb{CA}}$ by R via a universal argument. We remark that the proof of Durand's theorem appears to be formalizable in the theory V^0, if that is indeed the case this would imply a polynomial simulation of $\mathbb{P}_{\mathbb{CA}}$ by a constant-depth Frege system. [4]

[4] See [45] for background on bounded arithmetic.

Concluding remarks

We have shown that a propositional proof system such as Resolution can be interpreted as a string rewriting system; in particular as a Semi-Thue system. The interpretation Σ_n^* of the tree-like case has an interesting representation based on planar diagrams and they are similar to Van Kampen diagrams. This representation can be exploited more also to give a concrete geometrical representation, in Euclidean space in which we interpret the idea of giving a proof as that of reducing the volume of a given cylinder. This reduction is performed allowing planes sectioning the initial cylinder. We remark that the system Σ_n^* is very elegant and his formal representation does not require too many rules and the proofs in Σ_n^* have a structure which is really transparent as the structure of proofs in R^*. Indeed, all the complexity measures study for R^* can be characterized in a very clear way also for the system Σ_n^*.

In the case of general Resolution R things are less smooth, and this is because the proofs in R may have very complicated structure. Thus, in the simulation of them, using a string rewriting approach the number of rules substantially blow up and the resulting translation in Σ_n is much more tangled. Of course, in principle this is possible as we have outlined. We can simulate using string rewriting systems also general resolution and the gap between the proofs in R and in Σ_n is still polynomial, but it is not satisfactory in terms of its formal representation.

In 1972 Richardson proved the following remarkable result for cellular automata: let $\mathbb{A}$ be an injective cellular automaton. Then $\mathbb{A}$ is bijective and the inverse of $G_{\mathbb{A}}$, $G_{\mathbb{A}}^{-1}$, is the global function of a cellular automaton. We offered a new proof of this theorem exploiting only Craig's interpolation theorem and the compactness of propositional logic. Moreover, in the same chapter we solved two problems left open by Durand [32] about complexity of cellular automata. In particular the problems can be stated as follows:

1. Consider the case of finite bounded configurations. Let a reversible automaton be given by a "simple"algorithm. Is the inverse automaton given by a "simple"algorithm too?

2. The injectivity problem of cellular automata on bounded size is *co*$\mathcal{NP}$-complete, Theorem 4.1.1. Does the result still hold if we consider instead of the size of the transition table, the smallest program which computes its transition table?

In the first case the answer is negative, as shown in Theorem 3.3.3. The inverse automaton, even if we restrict our investigation to finite configurations, can be given by an algorithm which is exponentially bigger than the

algorithm of the invertible one's. The second problem has a positive answer. The injectivity (even if restricted to finite configurations) is indeed a very difficult problem though we consider the program (circuit) which computes the transition table of the cellular automaton instead of its transition table; it remains an $co\mathcal{NP}$-complete problem. Finally we define a new proof system based on cellular automata exploiting the $co\mathcal{NP}$-completeness of the injectivity problem plus the Richardson theorem.

To sum up, in this work we have investigated relations between propositional proof systems and various rewriting systems, and relations between complexity of both. We have constructed some concrete simulations (Chapter 2), found applications of propositional logic and Boolean complexity in cellular automata theory (Chapter 3), and invented an interpretation of an inverse automaton as a propositional proof (Chapter 4).

We view our results as initial steps in a new direction for research linking propositional logic (and its proof complexity) with rewriting systems and with cellular automata in particular. We think that studying the proof complexity strength (in the sense of Chapter 4) of some naturally occurring invertible cellular automata may be quite interesting.

List of Figures

Bibliography

[1] S. Amoroso and Y.N. Patt, Decision procedures for surjectivity and injectivity of parallel maps for tasselation structures, *Jour. Comput. System Scie.*, **6**, (1972), 448-464.

[2] A. Atserias, M. L. Bonet, On the automatizability of resolution and related Propositional Proof Systems, *Information and Computation*, **189(2)**, (2004), pp. 182-201.

[3] E. Berlekamp, J. Conway, R. Elwyn and R. Guy, *Winning way for your mathematical plays*, vol. 2, Academic Press, (1982).

[4] P. Beame, H. Kautz and A. Sabharwal, Towards Understanding and Harnessing the Potential of Clause Learning, *Journal of Artificial Intelligence Reasearch* (JAIR), **22**, (2004), pp. 319-351.

[5] P. Beame, R. Impagliazzo, J. Krajíček, T. Pitassi, and P. Pudlák, Lower bounds on Hilbert's Nullstellensatz and propositional proofs, in *Proc. London Math. Soc.*, **73(3)**, (1996), pp. 1-26.

[6] E. Ben-Sasson and R. Impagliazzo, Random CNF'S are hard for the polynomial calculus, in *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, (1999).

[7] E. Ben-Sasson, R. Impagliazzo, and A. Wigderson, Near-optimal separation of tree-like and general resolution, *ECCC*, Report TR02-005, (2000).

[8] E. Ben-Sasson, and A. Wigderson, Short proofs are Narrow–Resolution made Simple, *Journal of the ACM*, **48(2)**, (2001), pp.149-169.

[9] R. Berger, The undecidability of the domino problem, *Mem. Amer. Math. Soc.*, **66**, (1966), pp. 1-72.

[10] A. Blake, *Canonical expression in boolean Algebra*, Ph.D Thesis, (1937), University of Chicago.

[11] M.L. Bonet, J.L.Esteban, N. Galesi and J. Johannsen, Exponential separations between Restricted Resolution and Cutting Planes Proof Systems, In *39th Symposium on Foundations of Computer Science*, (FOCS 1998), pp.638-647.

[12] M. Bridson, The geometry of the word problem, in: *Invitations to Geometry and Topology*, Oxford University Press, (2002).

[13] K. Büning, T. Lettman, *Aussangenlogik: Deduktion und Algorithmen*, (1994), B.G Teubner Stuttgart.

[14] E. Burks, *Theory of Self-reproduction*, University of Illinois Press, Chicago, (1966).

[15] S. Buss (ed.), *Handbook of Proof Theory*, North-Holland, (1998).

[16] S. Cavagnetto, String Rewriting and Proof Complexity: an interpretation of Resolution, to appear in *Reports on Mathematical Logic*.

[17] S. Cavagnetto, Some Applications of Propositional Logic to Cellular Automata, submitted to *Mathematical Logic Quarterly*.

[18] N. Chomsky, Three models for the Description of Language, *IRE Transactions on Information Theory*, **(2)2**, (1956), pp.113-123.

[19] M. Clegg, J. Edmonds, and R. Impagliazzo, Using the Groebner basis algorithm to find proofs of unsatisfiability, in *Proceedings of 28th Annual ACM Symposium on Theory of Computing*, (1996), pp. 174-183.

[20] P. Clote and A. Setzer, On PHP st-connectivity and odd charged graphs, in P. Beame and S. Buss, editors,*Proof Complexity and Feasible Arithmetics*, AMS DIMACS Series Vol. 39, (1998), pp. 93-117.

[21] P. Clote and E. Kranakis, Boolean Functions and Computation Models, Texts in Theoretical Computer Science, Springer-Verlag, (2002).

[22] S. Cole, Real-time computation by n-dimensional iterative arrays of finite-state machine, *IEEE Trans. Comput*, **C(18)**, (1969), pp. 349-365.

[23] S. A. Cook, The complexity of theorem-proving procedures, in *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, (1971), pp. 151-158.

[24] S. A. Cook and A. R. Reckhow, On the lenghts of proofs in the propositional calculus, in *Prooceedings of the Sixth Annual ACM Symposium on the Theory of Computing*, (1974), pp. 15-22.

[25] S.A Cook and A.R. Reckhow, The relative efficiency of propositional proof systems, *Journal of Symbolic Logic*, **44(1)**, (1979), pp. 36-50.

[26] S. A. Cook, The P versus NP Problem, *Manuscript prepared for the Clay Mathematics Institute for the Millennium Prize Problems*, (2000).

[27] W. Cook, C. R. Cullard, and G. Turan, On the complexity of cutting planes proofs, *Discrete Applied mathematics*, **18**, (1987), pp.25-38.

[28] W. Craig, Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory, *Journal of Symbolic Logic*, **22(3)**, (1957), pp. 269-285.

[29] M. Davis, *Computability and Unsolvability, Dover Pubblications*, Inc, New York, (1958).

[30] M. Davis and H. Putnam, A computing procedure for quantification theory, *Journal of the ACM*, **7(3)**, pp. 210-215.

[31] M. Delorme and J. Mazoyer, editors, *Cellular Automata: a parallel model*, Mathematics and its Application, Springer, (1998).

[32] B. Durand, Inversion of 2D cellular automata: some complexity results, *Theoretical Computer Science*, **134**, (1994), pp.387-401.

[33] J.L. Esteban, J. Torán, Space bounds for resolution, *Information and Computation*, **171(1)**, (2001), pp. 84-97.

[34] Z. Galil, On resolution with clauses of bounded size, *SIAM Journal of Computing*, **6**, (1977), pp.444-459.

[35] M. R. Garey and D.S. Johnson, *Computers and Intractability - A guide to the theory of the NP-completeness*, W. H. Freeman, (1979).

[36] A. Goerdt, Cutting planes versus Frege proof systems, in: *Computer Science Logic:4th workshop, CSL '90*, E. borger and et al., eds, Lecture Notes in Computer Science, Spriger-verlag, (1991), pp.174-194.

[37] D. Hilbert and W. Ackermann, *Principles of Mathematical Logic*, New York, (1950).

[38] K. Iwama and S. Miyazaki, Tree-like Resolution is superpolinomially slower then dag-like resolution for the Pigeonhole Principle, in A. Aggarwal and C.P. Rangan, editors, *Proceedings: Algorithms and Computation, 10th International Symposium*, ISAAC'99, Vol. 1741, (1999), pp 133-143.

[39] M. Karchmer and A. Wigderson, Monotone circuits for connectivity require super-logarithmic depth, in *Proc. 20th Annual ACM Symp. on Theory of Computing*, ACM Press, (1988), pp. 539-550.

[40] J. Kari, Reversibility of 2D cellular automata undecidable, *Physica*, **D(45)**, (1990), 379-385.

[41] J. Kari, Reversibility and surjectivity problems of cellular automata, *Jour. Comput. System Scie.*, **48**, (1994), pp. 149-182.

[42] J. Kari, Reversible Cellular Automata, *Proceedings of DLT 2005, Developments in Language Theory*, Lecture Notes in Computer Science,**3572**, pp. 57-68, Springer-Verlag, (2005).

[43] J. Kari, Theory of cellular automata: A survey, *Theoretical Computer Science*, **334**, (2005), pp. 3-33.

[44] H. von Koch, Sur une courbe continue sans tangente obtenue par une construction géométrique élémentaire, *Archiv. für Matem. Fys.*, **1**, (1904), pp. 681-702.

[45] J. Krajíček, *Bounded arithmetic, propositional logic, and complexity theory*, Encyclopedia of Mathematics and Its Applications, **60**, Cambridge University Press, (1995).

[46] J. Krajíček, Propositional proof complexity I., *Lecture notes*, available at http://www.math.cas.cz/krajicek/biblio.html.

[47] J. Krajíček, Dehn function and length of proofs, *International Journal of Algebra and Computation*, **13(5)**,(2003), pp.527-542.

[48] J. Krajíček, Lower bounds to the size of constant-depth propositional proofs, *Journal of Symbolic Logic*, **59(1)**, (1994), pp.73-86.

[49] J. Krajíček, Interpolation theorems, lower bounds for proof systems, and indipendence results for bounded arithmetic, *Journal of Symbolic Logic*, **62(2)**, (1997), pp. 457-486.

[50] L. Levin, Universal search problem (in russian), *Problemy Peredachi Informatsii 9*, (1973), 115-116.

[51] E.F. Moore, Machine models of self-reproduction, *Proc. Symp. Appl. Math. Soc.*, **14**, (1962), pp. 13-33.

[52] J. Myhill, The converse to Moore's garden-of-Eden theorem, *Proc. Amer. Math. Soc.*, **14**, (1963), pp.685-686.

[53] D. Mundici, NP and Craig's interpolation theorem, *Proc. Logic Colloquium 1982*,North-Holland, (1984), pp. 345-358.

[54] J. von Neumann, The General and Logical Theory of Automata, in *Collected Works*, vol. 5, Pergamon Press, New York, (1963), pp. 288-328.

[55] J. von Neumann, *Theory of Self-reproducting automata*, ed. W. Burks, University of Illinois Press, Chicago, (1966).

[56] J. von Neumann, *Theory of automata: construction, reproduction and homogeneity*, unfinished manuscript edited for pubblication by W. Burks, see [14] pp. 89-250.

[57] C. H. Papadimitriu, *Computational Complexity*, Addison-Wesley, (1994).

[58] C. H. Papadimitriu, NP-completeness: A Retrospective, in *Proceedings of the 24th International Colloquium on Automata, Languages and Programming* **1256**, Lecture Notes in Computer Science, Springer, (1997), pp. 2-6.

[59] P. Pudlák, The Lenghts of Proofs, in *Handbook of Proof Theory*, ed. S. Buss, North-Holland, (1998), ch. 8, pp. 547-637.

[60] P. Pudlák, Lower bounds for resolution and cutting plane proofs and monotone computations, *Journal of Symbolic Logic*, (1997), pp. 981-998.

[61] R. Raz and P. McKenzie, Separation of the monotone NC hierarchy, in *Proc. 38th Symposium on Foundations of Computer Science*, (1997), pp. 234-243.

[62] A. A. Razborov, Unprovability of lower bounds on the circuits size in certain fragments of bounded arithmetic, *Izvestiya of the R. A. N.*, **59(1)**, (1995), pp. 201-224.

[63] D. Richardson, Tesselations with local transformations, *Jour. Comput. System Scie.*, **6**,(1972), pp. 373-388.

[64] J. A. Robinson, A machine-oriented logic based on the resolution principle, *Journal of the ACM*, **12(1)**, pp. 23-41.

[65] M. Sipser, *Introduction to the Theory of Computation*, PWS Publishing Company, Boston, (1997).

[66] M. Sipser, The history and the status of the P versus NP question, *STOC*, (1992), pp. 603-618.

[67] S. Smale, Mathematical problems for the next century, in *Mathematics: Frontiers and perspectives*, AMS, (2000), pp. 271-294.

[68] A. Smith III, A Simple computatio-universal spaces, *Journal of ACM*, (1971), **18**, pp. 339-353.

[69] K. Sutner, De Bruijn graphs and linear cellular automata, *Complex Systems*, **5**, (1991), 19-31.

[70] A. Thue, Die Lösung eines Spezialfalles eines gnerellen Logischen Problems, *Kra. Videnskabs-Selskabets Skriften I. Mat. Nat. Kl.*, **8**, (1910).

[71] A. Thue, Probleme über Veranderungen von Zeichenreihen nach gegeben Regeln, *Skr. Vid. Kristianaia I. Mat. Natarv. Klasse*, **10/13**, (1914).

[72] T. Toffoli and N. Margolus, *Cellular Automata Machines*, MIT Press, Cambridge MA, (1987).

[73] G. S. Tseitin, On the complexity of derivation in propositional calculus, in A. Slisenko ed., *Studies in Constructive Mathematics and Mathematical Logic*, (1970), Consultants Bureau, New York, pp. 115-125.

[74] A. Turing, On computable numbers with an application to the Enthscheidungsproblem, *Proc. London Math. Soc.*, **42**, (1936), pp. 230-265.

[75] A. Urquhart, The Complexity of Propositional Proofs, *Bulletin of Symbolic Logic*, **1(4)**,(1996), pp. 425-467.

[76] A. Wigderson, P, NP and Mathematics-a computational complexity perspective, http://www.math.ias.edu/avi/BOOKS/.

[77] K. Wagner and G. Wechsung, *Computational Complexity*, Riedel, (1986).