Propositional Proof Complexity and Cellular Automata

Stefano Cavagnetto
School of Computing and Prague College Research Centre, Prague College
Polská 10, Prague
Czech Republic

1. Introduction

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 consider how classic propositional logic and in particular Propositional Proof Complexity can be combined with the study of Cellular Automata. It is organized as follows: in the next section we recall some of the basic definitions in computational complexity theory [1]. In the same section we introduce some basic definitions from propositional proof complexity[2] and we recall an important result by Cook and Reckhow (20) which gives an interesting link between complexity of propositional proofs and one of the most beautiful open problem in contemporary mathematics[3]. Section 3 deals with cellular automata and on how Propositional Logic and techniques from Propositional Proof Complexity can be employed in order to give a new proof of a famous theorem in the field known as Richardson's Theorem[4]. In the chapter some complexity results regarding Cellular Automata are considered and described. The ending section of the chapter deals with a new proof system based on cellular automata and also it outlines some of the open problems related to it.

2. Some technical preliminaries

In 1936 Alan Turing (63) 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

[1] For a self-contained exposition of the field the interested reader can see (56), (49).
[2] There are many survey papers on propositional proof complexity offering different emphasis; the interested reader can see (64), (17)and (51).
[3] The famous $\mathcal{P}$ versus $\mathcal{NP}$ problem, (22), (50), (57), (65), (58).). In this chapter, the next section regarding computational complexity follows in detail Cook's paper (22)
[4] This new proof was given by the present author in (14); section 3 follows in detail this work.

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 2.1. *If C and C' are configurations, then $C \overset{M}{\to} C'$ 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 2.2. *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 $C_i \overset{M}{\to} C_{i+1}$ 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 2.3. *M accepts w if and only if the computation is finite and the final configuration contains the state q_{accept}.*

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[5] that M accepts w if this computation terminates in the accepting state.[6] The language accepted by M that we denote by $L(M)$ has associated alphabet Σ and is defined by

[5] See Definition 2.3.

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

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

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 length 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.

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 lengths 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 (29).

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 2.4 (Cook (19), Levin (42)). *$SAT \in \mathcal{P}$ 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 2.5. *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 2.4 is the following one.

Theorem 2.6. *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 introductory 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 2.7. *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 labelled 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 2.8. *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 $n \geq n_0$, $f(n) \geq cg(n)$.[7] 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 $\lim_{n\to\infty} f(n)/g(n) = 0$.

[7] When $f(n) = O(g(n))$ we say that $g(n)$ is an asymptotic upper bound for $f(n)$.

2.1 The complexity of propositional proofs

The complexity of propositional proofs has been investigated systematically since late ’60s.[8] Cook and Reckhow in (20), (21) 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[9] 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) $p_0, \ldots, p_n$, and the connectives. Consider the following example of set of axioms taken from Hilbert’s and Ackermann’s work (31), where $A \to B$ is just the abbreviation of $\neg A \vee B$,

1. $A \vee (A \to A)$
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*[10] (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 2.9. *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 premises and in which case it is called an axiom schema (as the example above for the axioms (1) to (4)).

Definition 2.10. *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 $\phi \in TAUT$.*[11] *F is implicationally complete.*[12]

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.

[8] The earliest paper on the subject is an article by Tseitin (62).
[9] Introduced in the previous section when we defined the problem SAT.
[10] In Latin, the mode that affirms by affirming.
[11] The “if” direction is the completeness and the “only” direction is the soundness of F.
[12] 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.

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 (20),

Definition 2.11 (Cook Reckhow (20))**.** *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 2.12. *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 (20) 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 2.13 (Cook Reckhow (20))**.** *$\mathcal{NP} = co\mathcal{NP}$ 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 2.12. 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 2.13, 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 (38) 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.[13] 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 (38).
We conclude this section with the notion of polynomial simulation introduced in (20). The definition 2.14 is simply a natural notion of quasi-ordering of propositional proof systems by their strength.

Definition 2.14. *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.

2.2 Resolution

The logical calculus Resolution R is a refutation system for formulas in conjunctive normal form. This calculus is popularly credited to Robinson (55) but it was already contained in Blake's thesis (9) and is an immediate consequence of Davis and Putnam work (26).
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, $C = \{\ell_1, \dots, \ell_n\}$ and is interpreted as the disjunction $\bigvee_{i=1}^{n} \ell_i$. A truth assignment $\alpha : \{p_1, p_2, \dots\} \to \{0,1\}$ 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 $\mathcal{C} = \{C_1, \dots, C_m\}$ 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 $\alpha : \{p_1, p_2, \dots\} \to \{0,1\}$ satisfies both upper clauses of the rule then it also satisfies the lower clause.
A resolution refutation of ϕ is a sequence of clauses $\pi = D_1, \dots, D_k$ 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 2.15. *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,

[13] Unless there is an optimal proof system.

$\neg p_1, \ldots, p_n, \neg p_n$ 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 contain $\{p_1\}$ and $\{\neg p_1\}$ and then by the resolution rule we have $\{\}$.

Induction Step: Assume that $n > 1$. Partition $\mathcal{C}$ in four disjoint 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 $\alpha' : \{p_1, \ldots, p_{n-1}\} \rightarrow \{0, 1\}$ 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, than 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 needed 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 (64), and later by others in (16) and (32). Later on, in (10) 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 (6).

2.3 Interpolation and effective interpolation

A basic result in mathematical logic is the Craig interpolation theorem (24). The theorem says that whenever an implication $A \rightarrow B$ 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 $A \rightarrow I$ and $I \rightarrow B$ are both valid formulas. Craig's interpolation theorem is a fundamental result in propositional logic and in predicate logic as well.[14]

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

Finding an interpolant for the implication is a problem of some relevance with respect to computational complexity theory. Mundici in (45) pointed out the following. Let U and V be two disjoints $\mathcal{NP}$-sets, subsets of $\{0,1\}^*$. By the proof of the $\mathcal{NP}$-completeness of satisfiability (19) 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

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

Assuming 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. Craig's interpolation theorem guarantees 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

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

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 length of the implication $A_n \to \neg B_n$. This is because, as remarked by Mundici (45), 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 $W = U$. In (39), Krajíček formulated the idea of effective interpolation as follows:

For a given propositional proof system, try to estimate the circuit-size of an interpolant of an 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 $A_n \to \neg B_n$. 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 (41), (52). Besides lower bounds, effective interpolation revealad to be an excellent idea in other areas in proving results of independence in bounded arithmetic (53) and in establishing links between proof complexity and modern cryptography.The footnote 15 can stay.[15]

Definition 2.16. *A propositional proof system P admits effective interpolation if and only if there is a polynomial $p(x)$ such that any implication $A \to B$ with a proof in P of size m has an interpolant of a circuit size $\leq p(m)$.*

[15] A general overview of these applications has been given in (38) and (51).

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 2.17. *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 system Resolution admits feasible interpolation, as it was proven in (41). In fact,

Theorem 2.18 (Krajíček (41))**.** *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 $kn^{O(1)}$.

The key idea of the proof of the previous theorem is that the structure of a resolution refutation allows one easily to decide which clauses cause the unsatisfiability under a specific assignment. Furthermore, it should be noticed that the interpolant can be computed by a polynomial time algorithm having an access to the resolution refutation. Theorem 2.18 is important because it is a theorem used later on in the next section.

2.4 "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 a well-known "mathematical[16] " 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

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

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

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

[16] This expression is taken from Pudlák (51).

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:

(a)

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

(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;

(c)

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

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 2.15. Soundness in the sense that given any formula ϕ which has a resolution 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 2.19 can be proved exploiting the completeness of R, since *CP* easily simulates resolution as observed in (23).

Theorem 2.19. *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 (30) proved that Frege systems polynomially simulate the *CP* proof system.
Other examples of mathematical proof systems are for instance the Nullstellensatz system introduced in (4), the Polynomial Calculus (15) and the Gaussian Calculus first defined in (5). At the end of this chapter a new mathematical proof system using cellular automata will be proposed.

3. Applications of propositional logic to cellular automata

Cellular automata 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. In the literature one can find different types of neighbourods depening upon which group of cells are taken into consideration during the application of the local rule. In the figure below the Von Neumann, Moore and Simth neighbourods are displayed in Figure 1.

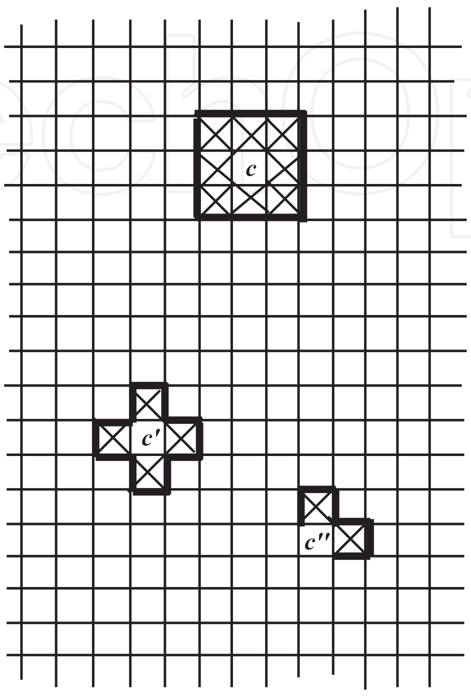

Fig. 1. The Moore neighborhood of the cell c, the von Neumann neighborhood of the cell $c^{'}$ and the Smith neighborhood of the cell $c^{''}$.

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. [17] In literature cellular automata take various names according to the way they are used. They can be employed as computation models (27) or models of natural phenomena (61), but also as tessellations structures, iterative circuits (12), or iterative arrays (18). The study of this computation model was initiated by von Neumann in the ’40s (46), (47). 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.
Cellular automata have been extensively studied as discrete models for natural system 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. They 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

[17] For surveys on cellular automata the interested reader can see (35), (36).

(spin systems, crystal growth process, lattice gasses, ...) have been modelled using these devices, see (61).
In this section 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. In this section we focus on a new proof of a foundamental theorem in the field, the Richardson theorem (54), given in (14). Then application of feasible interpolation shows how to find computational description of inverse cellular automata. Then we consider in some detail two complexity problems formulated in (28) and solved in (14) as well.

3.1 Cellular Automata: definitions and some basic results

From a formal point of view 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 any Abelian group $\mathcal{G}$ in place of $\mathbb{Z}^d$ can be used. 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. *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 $\boldsymbol{C}$.*

At discrete time steps the cells change their states synchronously. 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 3.2. *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 3.3. *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. Often 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 3.4. *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

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

Notice that 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 $\vec{y} \in N_{\mathbb{B}}$.
The reader can see that the problem of establishing whether or not two given cellular automata $\mathbb{A}$ and $\mathbb{B}$, with $G_{\mathbb{A}}$ and $G_{\mathbb{B}}$, are equivalent is decidable. To see this it is enough to observe that:

(i) if $N_{\mathbb{A}} = N_{\mathbb{B}}$ then the local transformation rules, $f_{\mathbb{A}}$ and $f_{\mathbb{B}}$, are identical;

(ii) 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.

Intuitively, 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.[18] Translations are compositions of shift functions.
In the literature quite often a particular state $q \in S$ is specified as a quiescent state (which usually simulates empty cells). The state must be stable, i.e. $f(q, q, ..., q) = q$. Thus a configuration c is said to be quiescent if all its cells are quiescent, $c(\bar{x}) = q$.

Definition 3.5. *A configuration $c \in S^{\mathbb{Z}^d}$ is finite if only a finite number of cells are non-quiescent, i.e. the set (support),*

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

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 3.6. *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 $G^P : \mathbf{C}_P \to \mathbf{C}_P$.
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 (see Figure 2) that is obtained by joining together the opposite sides of a rectangle. The relevant group is $(\mathbb{Z}/t_1) \times (\mathbb{Z}/t_2)$.

Definition 3.7. *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 $G_{\mathbb{A}}$.*

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

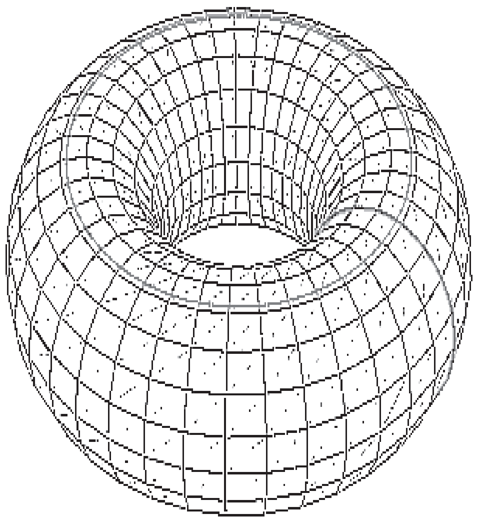

Fig. 2. A toroidal arrangement: when one goes off the top, one comes in at the corresponding position on the bottom, and when one goes off the left, one comes in on the right.

When one deals with finate sets a basic property that holds is the following: a function from a finite set into itself is injective if and only if the function is surjective. This property is partially tru for cellular automata. In fact, 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. Theorem 3.9, a combination, a combination of two results proved by Moore (43) and by Myhill (44) respectively, shows out exaclty this fact.

Definition 3.8. *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 3.9 (Moore (43), Myhill (44))**.** *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 3.10 (Richardson (54))**.** *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 3.11. *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 is a consequence of the effectiveness of the composition and the decidability of the equivalence. In 1972 Richardson proved the following important theorem about cellular automata:[19]

Theorem 3.12 (Richardson (54))**.** *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.*

The same year of Richardson's result Amoroso and Patt proved the following theorem:

Theorem 3.13 (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 in order to determine if a given cellular automaton is surjective.[20] In higher spaces the problem of showing if a given cellular automaton is surjective or not, has been shown undecidable by Kari (34). Kari proved also that the reversibility of cellular automata is undecidable too,

Theorem 3.14 (Kari (34))**.** *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 3.14 is based on the transformation of the tiling problem shown to be undecidable by Berger (8), into the invertibility problem on a suitable class of cellular automata.

3.2 A new proof of the Richardson theorem based on propositional logic

Theorem 3.12 was proven using a topological argument in combination with the Garden of Eden theorem. Richardson's proof was non-constructive (it used compactness of a certain topological space) and the new proof is formally non-constructive too (becaue of the use of compactness of propositional logic). One should notice that this non-constructivity is unavoidable because of the undecidibility theorem proved by Kari; see Theorem 3.14. The new proof given in (14) offers a technical simplification: only basic logic is involved requiring straightforward formalism, and it allows us to apply an interpolation theorem. It is important to point out that the proof can be made fully constructive, if we consider periodic configurations; considering $d = 2$ the working space becomes a torus.[21]
The proof use as a dimension $d = 2$ and on the binary alphabet. This simplifies the notation but displays the idea of the proof in full generality which works $d > 2$ and extended alphabets as well. We report below the statement of Richardson's theorem and the proof in some detail. The reader can see (14) for the complete one.

Theorem 3.15. *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 $G_{\mathbb{B}} \circ G_{\mathbb{A}} = id$.*

The new proof given by the present author in (14) goes as follows. For an n-tuple of $\mathbb{Z}^2$-points $N = ((u_1, v_1),\ldots,(u_n, v_n))$ defining the neighborhood of $\mathbb{A}$ denoted by

$$(i, j) + N$$

[19] Recall that on finite configurations the global function may be onto and one-to-one even if the cellular automaton is not reversible.
[20] Later Sutner designed elegant decision algorithms based on de Bruijn graphs, see (60).
[21] After Theorem 3.15 this point will be discussed in some detail.

the n-tuple $(i + u_1, j + v_1), \ldots, (i + u_n, j + v_n)$. Then we define a suitable embedding into propositional logic as follows: For each i, j let $p_{i,j}$ be a propositional variable. Denote by $p_{(i,j)+N}$ the n-tuple of variables

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

This embedding has the consequence of characterizing the transformation function of the cellular automata a as 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 $T_A(\vec{p}, \vec{r})$.
Then the next step is to define f by a CNF (or DNF) formula. In this manner 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 of the form:

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

for all $(i, j) \in \mathbb{Z}^2$. Since the theory is unaltered 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 reformulated in the following manner:

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

is unsatisfiable.
By applying 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. At this point in the proof we can use the Craig interpolation theorem. This result guarantees the existence of a formula $I(\vec{r})$, such that:
$$T_0(\vec{p},\vec{r}) + p_{(0,0)} \vdash I(\vec{r})$$
and
$$I(\vec{r}) \vdash T_0(\vec{q},\vec{r}) \to q_{(0,0)}.$$
Although we write the whole array $\vec{r}$ in $I(\vec{r})$, the formula obviously contains only finitely many r variables.
Thus by application of the deuction theorem we have that:
$$T_0(\vec{p},\vec{r}) \vdash p_{(0,0)} \to I(\vec{r})$$
and
$$T_0(\vec{q},\vec{r}) \vdash I(\vec{r}) \to q_{(0,0)}.$$
Then after suitably renaming the propositional variables we see that the the second implication gives:
$$T_0(\vec{p},\vec{r}) \vdash I(\vec{r}) \to p_{(0,0)}$$
i.e. together
$$T_0(\vec{p},\vec{r}) \vdash I(\vec{r}) \equiv p_{(0,0)}.$$
The interpolant $I(\vec{r})$ computes the symbol of cell (0,0) in the configuration prior to $\vec{r}$. This 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 $I(\vec{r})$. Finally the inverse cellular automaton $\mathbb{B}$ as follows:

1. Alphabet is 0,1;
2. The neighborhood is M;
3. The transition function is given by $I(\vec{r})$,

This concludes the proof.
We conclude this part about the Richardson theorem with a few remarks about the proof sketched above.

1. The construction of the inverse cellular automaton carried out in the proof given in (14) has two key moments. First the application of the compactness theorem after a suitable embedding into propositional logic. In a second moment the interpolation theorem is applied leading to some kind of effectiveness. Of course, the application of compactness leads to a non-recursive procedure but as we previously noticed this fact is unavoidable because of theorem 3.14.
2. The construction guarantees that
$$G_{\mathbb{B}}(G_{\mathbb{A}}(\vec{p})) = \vec{p}$$
but it does not - a priori - imply that also
$$G_{\mathbb{A}}(G_{\mathbb{B}}(\vec{r})) = \vec{r}.$$

This is a consequence of Theorem 3.9 (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 (28), see Definition 3.16 below) is $O(2^{|M|})$. This also bounds the size $|I|$ of any formulas defining the interpolant, but the interpolant I could be in principle defined by a substantially smaller formula (e.g. of size $O(|M|)$.)

It is interesting to notice that 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, 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 $\mathbb{A}$. In this case we can avoid the use of the compactness theorem and the interpolation can be directly applied.
In (14) is also given a constructive way how to to find the interpolant and the inverse automaton as well.

3.3 Some new complexity results

In this paper Durand (28) proved the first complexity results concerning a global property of cellular automata of dimension ≥ 2 (see Theorem 3.17). By Kari's result (33) 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 hard 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). Nevertheless ome open problems are left from Durand' work in (28). A solution to two of the open problems stated in (28) has been offered in (14) by the present author. Below we give in some detail a summary of Durand's theorem and of the two solution. For more details the reader can see (28) and (14). In (28) 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 3.16. *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 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.[22]
For the $co\mathcal{NP}$-completeness 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 is the main result in (28),

[22] Notice that result is obtained for a 2-dimensional cellular automata with von Neumann neighborhood, see Figure 1.

Theorem 3.17 (Durand (28))**.** *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.[23] 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 3.18 (Durand (28))**.** *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 be used to form a finite nontrivial tiling of the plane of size smaller than $(2n-4) \times (2n-4)$.*

Then using Theorem 3.18 he proves that ***PROBLEM*** *(CA-FINITE-INJECTIVE)* is co$\mathcal{NP}$-complete, i.e. Theorem 3.17.
Notice that 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 (28).
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 (28) 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 (28) 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 (14) both problem got a solution. Let us see in some detail the solutions. The second problem is solved for succintness on $d = 1$ it is fairly simple to get obtain examples for $\mathbb{Z}^2$ or $(\mathbb{Z}/m)^2$.
Let $f : \{0,1\}^n \to \{0,1\}^n$ be a Boolean function 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, $\exp(\Omega(n))$.

As an example of these type of functions one could take one-way permutations (e.g. conjecturally based on factoring or discrete logarithm). Now define a cellular automaton $\mathbb{A}_f$ as follows:

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

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) $p_i^t = \# \to p_i^{t+1} = \#$
(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 $r = j+1, j+2, \ldots, i, \ldots, k-1$
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

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

Informally the automaton $\mathbb{A}_f$ can be summarized as follows: every $0-1$ segment between two consecutives #'s that does not have the length exactly n is left unchanged. Segments of length n are trasformed according to the permutation f.
The inverse automaton $\mathbb{B}$ is defined in the same manner using f^{-1} in place of f ($\mathbb{B} = \mathbb{A}_{f^{-1}}$).

Theorem 3.19. *Assume that $f : 2^n \to 2^n$ 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.*

The proof is based on the fact that by the construction the inverse cellular automaton has a transition table which defines a boolean function which has a circuit-size exponential in n.
The hypothesis of Theorem 3.19 follows from the existence of cryptographic one-way functions. In particular, it follows from the exponential hardness of factoring or of discrete logarithm.
Thus Theorem 3.19 solves negatively one of the open problem formulated by Durand (28) 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".[24]
The second problem stated in (28) and solved in (14) asks 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

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

$\mathbb{A}_C$ with a neighborhood N of size $|N| = n$.
*Question:*Is $\mathbb{A}_C$ injective on $\mathbb{Z}^1$?

Theorem 3.20. *Problem (P1) is co$\mathcal{NP}$-hard.*

The proof offered in (14) describes a polynomial reduction from $TAUT$ to *(P1)*. Let $\phi(x_1, ..., x_n)$ be a propositional formula. A sketch of the proof can be the following. 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

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

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 $\phi \notin TAUT$. Then two different configurations mapped by $\mathbb{A}$ to the same configuration have been constructed. 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 as follows:

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

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.

Lemma 3.21. *The two configurations C_0 and C_1 are both mapped by $\mathbb{A}$ to C_0.*

The proof is given using the definition above and then Theorem 3.20 follows directly from the application of the lemma.
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 3.22. *(P2) is co$\mathcal{NP}$-complete.*

The proof of Theorem 3.22 is quite similar to the proof of 3.20

4. Inverse Cellular Automata as propositional proofs

In this last section of the chapter we combine the Richardson theorem with the $co\mathcal{NP}$-completeness result of Durand (28) 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 (21).
In this conclusive section 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, we define a "mathematical" proof system for $\mathcal{L}_D$ satisfiyng the Cook and Reckhow definition (21). We conclude this chapter with some concluding remarks and some open questions when we consider our new proof system with respect to polynomial simulations.
We considered Durand's result of $co\mathcal{NP}$-completeness in the section 3. Let us recall the problem because it will be useful below. The problem that has been called (CA-FINITE-INJECTIVE) 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.
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. *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 3.17 by Durand can be simply stated as follows:

Theorem 4.2. *$\mathcal{L}_D$ is a $co\mathcal{NP}$-complete language.*

Now, consider the following lemma which states the existence of a polynomial time algorithm deciding the inverse:

Lemma 4.3. *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 3.16. 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.

$\square$

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 $\mathbb{B}$.

4.1 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 4.4. *$\mathbb{P}_{\mathbb{CA}}$ 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 4.5. *$\mathbb{P}_{\mathbb{CA}}$ 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 3.15. On the other hand the existence of $\mathbb{B}$ implies 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 4.3.

$\square$

The statement $\mathbb{A} \in \mathcal{L}_D$ can be expressed in a propositional way, same as in the proof of Richardson's theorem proved in (14). In particular, a proof of $\mathbb{A} \in \mathcal{L}_D$ is a proof of the unsatisfiability of the formula[25]:

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

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

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 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 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.

4.2 Some concluding remarks regarding the new proof system

The main problems which remain open from this last part 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 desiderable to have a proof which involves propositional logic, as the proof of Richardson's theorem given in (14), since this could give us a very elegant and unified framework.
Having such a simplified proof one could use the well-known relation between bounded arithmetic and proof systems (see (37)) 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. [26]

5. Conclusions

In this chapter we have seen some interactions between the study of cellular automata and the study of propositional proofs. In some sense the first attempt was made in (14) and the ending part of the final section constitutes a development in that direction. Nevertheless as pointed out at the end of the previous section several problems concerning simulations remain open for the reasons given above.

6. Acknowledgements

I thank Jan Krajíček for helpful discussions.

7. References

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

[26] See (37) for an extensive background on bounded arithmetic.

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

[4] 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.

[5] 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).

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

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

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

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

[10] 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.

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

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

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

[14] S. Cavagnetto, Some Applications of Propositional Logic to Cellular Automata, *Mathematical Logic Quarterly*, 55, (2009), pp. 605-616.

[15] 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.

[16] 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.

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

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

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

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

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

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

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

[24] 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.
[25] M. Davis, *Computability and Unsolvability, Dover Pubblications*, Inc, New York, (1958).
[26] M. Davis and H. Putnam, A computing procedure for quantification theory, *Journal of the ACM*, 7(3), pp. 210-215.
[27] M. Delorme and J. Mazoyer, editors, *Cellular Automata: a parallel model*, Mathematics and its Application, Springer, (1998).
[28] B. Durand, Inversion of 2D cellular automata: some complexity results, *Theoretical Computer Science*, 134, (1994), pp.387-401.
[29] M. R. Garey and D.S. Johnson, *Computers and Intractability - A guide to the theory of the NP-completeness*, W. H. Freeman, (1979).
[30] 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.
[31] D. Hilbert and W. Ackermann, *Principles of Mathematical Logic*, New York, (1950).
[32] 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.
[33] J. Kari, Reversibility of 2D cellular automata undecidable, *Physica*, D(45), (1990), 379-385.
[34] J. Kari, Reversibility and surjectivity problems of cellular automata, *Jour. Comput. System Scie.*, 48, (1994), pp. 149-182.
[35] 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).
[36] J. Kari, Theory of cellular automata: A survey, *Theoretical Computer Science*, 334, (2005), pp. 3-33.
[37] J. Krajíček, *Bounded arithmetic, propositional logic, and complexity theory*, Encyclopedia of Mathematics and Its Applications, 60, Cambridge University Press, (1995).
[38] J. Krajíček, Propositional proof complexity I., *Lecture notes*, available at http://www.math.cas.cz/krajicek/biblio.html.
[39] J. Krajíček, Dehn function and length of proofs, *International Journal of Algebra and Computation*, 13(5),(2003), pp.527-542.
[40] J. Krajíček, Lower bounds to the size of constant-depth propositional proofs, *Journal of Symbolic Logic*, 59(1), (1994), pp.73-86.
[41] 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.
[42] L. Levin, Universal search problem (in russian), *Problemy Peredachi Informatsii 9*, (1973), 115-116.
[43] E.F. Moore, Machine models of self-reproduction, *Proc. Symp. Appl. Math. Soc.*, 14, (1962), pp. 13-33.
[44] J. Myhill, The converse to Moore's garden-of-Eden theorem, *Proc. Amer. Math. Soc.*, 14, (1963), pp.685-686.
[45] D. Mundici, NP and Craig's interpolation theorem, *Proc. Logic Colloquium 1982*,North-Holland, (1984), pp. 345-358.
[46] J. von Neumann, The General and Logical Theory of Automata, in *Collected Works*, vol. 5, Pergamon Press, New York, (1963), pp. 288-328.
[47] J. von Neumann, *Theory of Self-reproducting automata*, ed. W. Burks, University of Illinois Press, Chicago, (1966).

[48] J. von Neumann, *Theory of automata: construction, reproduction and homogeneity*, unfinished manuscript edited for pubblication by W. Burks, see (12) pp. 89-250.

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

[50] 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.

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

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

[53] 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.

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

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

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

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

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

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

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

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

[62] 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.

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

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

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

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