Санкт-Петербургский государственный университет

ГАЕВОЙ Никита Сергеевич

Выпускная квалификационная работа

Simulations between proof systems

Уровень образования: магистратура
Направление 01.04.01 “Математика”
Основная образовательная программа ВМ.5832.2019 “Современная математика”

Научный руководитель:
Профессор, СПбГУ,
д.ф.-м.-н.,
Гирш Эдуард Алексеевич

Рецензент: Postdoc,
Institute of Mathematics of CAS in Prague, PhD in Mathematics,
Парт Фёдор Александрович

Санкт-Петербург
2021

Contents

1 Introduction

The notion of a propositional proof system is a central notion of proof complexity. The classical version of the definition of a *proof system* by Cook and Reckhow [CR79] is a polynomial-time mapping of all strings ("proofs") onto "theorems" that are elements of a certain language L. If the language of theorems is the language of all propositional tautologies TAUT, then the proof system is called a *propositional proof system*. A propositional proof system is *polynomially bounded* if it has a polynomial size proof for every tautology. The problem of the existence of a polynomially bounded propositional proof system is the fundamental problem of proof complexity and it is equivalent to the problem of the equality of classes NP and coNP. A propositional proof system can also be equivalently defined as a polynomial-time function that takes a formula and a string and verifies whether the given string is a valid proof of the given formula. In order to be a valid proof system such a function should accept at least one string for each tautology (*completeness*) and reject all strings for all other formulas (*soundness*). The latter definition is more convenient for working with simulations, so we will use it throughout the entire work.

Similarly to the languages in complexity classes, propositional proof systems also have notions for "reduction". A proof system Π *simulates* another proof system Φ if for every tautology the size of its shortest Π-proof is at most polynomially longer than the size of the shortest Φ-proof. However, we are mostly interested in a stronger notion of *p-simulation* that additionally requires the existence of a polynomial-time mapping of all Π-proofs into Φ-proofs (the mapping can also access the tautology). The second major open problem in proof complexity is the problem of the existence of an *optimal* (*p-optimal*) proof system, thats it, a system that simulates (p-simulates) any other proof system. The existence of the optimal propositional proof system would not only reduce the problem $\mathsf{NP} \stackrel{?}{=} \mathsf{coNP}$ to proving proof size bounds for just one proof system, but also imply the existence of a complete disjoint NP pair [Raz94; Pud01].

Unfortunately, there is no known correspondence between the problem of the existence of an (p-)optimal proof system and any structural assumptions (like $\mathsf{NP} \stackrel{?}{=} \mathsf{coNP}$ for the problem of the existence of a polynomially bounded proof system). Krajíček and Pudlák [KP89] showed that the existence of an optimal (respectively, p-optimal) propositional proof system follows from the assumption $\mathsf{NE} = \mathsf{coNE}$ (respectively, $\mathsf{E} = \mathsf{NE}$) and Köbler, Messner and Torán [KMT03] weakened these conjectures to double exponential time, however, the converse implication is not known nor widely believed.

A proof system is *automatizable* if it has an algorithm that, given a tautology, is able to produce its proof in a time polynomial in the length of the shortest proof. Automatizability seems to be quite a restrictive trait of a proof system, that is, for many proofs systems it is known that they are not automatizable unless $\mathsf{P} = \mathsf{NP}$ [AM20; de +20; Göö+20; Gar20]. However, it is not known whether optimal proof systems (if they exist) can be automatizable.

Since the problem of the existence of (p-)optimal proof system is seemingly hard, it is also interesting to consider its extended or restricted cases. Cook and Krajíček showed that optimal proof systems with non-uniform advice exist and that even one bit of advice per proof length is sufficient to p-simulate all classical propositional proof systems [CK07]. Another way to extend the problem is to generalize the notion of p-simulation. The most prominent example of this approach is the notion of *effective simulation* introduced by Pitassi and Santhanam [PS10].

In this work we focus on another special case of the problem. We introduce a notion of C-*simulation*, a specialized version of p-simulation, for certain classes of circuits C and study the problem of the existence of C-optimal propositional proof systems (i.e. systems that C-simulate any other propositional proof system). In Section 2 we provide a positive result for this problem: the existence of p-optimal proof systems implies the existence of C-optimal proof systems for any reasonable class C. Then, we introduce a notion of a *bounded* proof system and dedicate Sections 4 and 5 to negative results. These results state that no efficiently bounded C-optimal (where C stands for monotone and AC^0-circuits, respectively) proof system could exist unless there exists an automatizable optimal proof system. Informally, it means that simulating other systems in C-optimal proof system using only efficient proofs is almost as hard as automatizing.

2 Definitions and the unbounded case

Let TAUT denote the language of all propositional tautologies. We use the following definition of a proof system.

Definition 1. A propositional proof system is a polynomial-time algorithm $\Pi(\varphi, w)$ such that for all φ

$$\varphi \in \mathsf{TAUT} \iff \exists w\ \Pi(\varphi, w) = 1$$

This definition is equivalent to the classic definition by Cook and Reckhow [CR79] for the case of the language of all tautologies.

The standard notion of reduction of proof systems is p-simulation. The problem of the existence of p-optimal proof systems is a major open question, therefore, we define a stronger notion of the C-simulation for relatively small classes C of circuits.

Definition 2. Let C be a class of circuits. A propositional proof system Π is C-simulated by propositional proof system Π' if and only if there exists a polynomial-time algorithm P such that for every formula $\varphi \in \mathsf{TAUT}$ algorithm $P(\varphi, m)$ generates a circuit from class C that maps all Π-proofs of φ of the size m into Π'-proofs of φ.

Note that the generated circuit takes only the proof as the input and does not see the formula, but it is compensated by the fact that P is able to produce different circuits for different formulas. It is also important to note that since the produced circuit takes only the proof as its input, if class C has some restrictions on the size of the circuit (e.g. AC^0 requires the circuit to be polynomial-size), these restrictions should be considered as a function of m and not $|\varphi| + m$.

Definition 3. A propositional proof system is C-optimal if and only if it C-simulates any other propositional proof system.

For C equal to the class of all circuits the notion of C-simulation differs from p-simulation only at the fact that the size of the output of C-simulation is determined by the formula and the size of the input proof, while for p-simulation it is not necessary. However, it does not affect the ability to simulate one system by another if the target system allows to add padding symbols to its proofs. Therefore, for such C, a C-optimal proof system exists if and only if there exists a p-optimal proof system. Thus, we are mostly interested in the problem of existence of C-optimal proof systems for smaller classes of circuits.

Let Copy be a class of monotone NC-circuits of depth 1. In other words, the output bits of circuits from Copy could be either constants or copies of the input bits.

Theorem 1. *Let Π be a p-optimal proof system. Then there exists a Copy-optimal proof system Π'.*

Proof. Let Π' be a proof system such that its proofs are tuples of three elements $\left\langle A, w, 1^{\tau(|w|)} \right\rangle$ with elements interpreted as follows. A is a number of some Turing machine with an alarm clock that works in time $\tau(|w|)$, w is

some string and $1^{\tau(|w|)}$ is just a padding to ensure that the size of the proof is at least $\tau(|w|)$. The tuple is a valid Π'-proof if and only if the string obtained as a result of running A on string w for $\tau(|w|)$ steps is a valid Π-proof. It is easy to see that Π' is a propositional proof system. On the other hand, if Π is p-optimal, then for any other proof system Φ there exists an algorithm A such that given Φ-proof w it runs in polynomial time $\tau(w)$ and outputs some valid Π-proof. As a consequence, the tuple $\langle A, w, 1^{\tau(|w|)} \rangle$ is a valid Π'-proof. It remains to check that this tuple could be generated by some circuit from Copy, but A and $1^{\tau(|w|)}$ are some constants depending only on Φ and w is just a copy of the input. □

It is important to note that it is crucial for the construction above to allow of an arbitrary length, because although A and 1^{τ} are constants depending only on Φ, they can be very long and their sizes cannot be bounded in any reasonable way unless Π is automatizable (then A and τ could be chosen as global constants, independent from Φ, see Lemma 2 below).

In order to formalize this observation, we provide the following definition.

Definition 4. An f-bounded propositional proof system is a proof system that given formula φ accepts only proofs of size at most $f(\varphi)$, where f is a polynomial-time function.

Note that any proof system could be made f-bounded just by rejecting too long proofs, if the size of the shortest proof never exceeds $f(\varphi)$. However, this operation does not preserve optimality.

Definition 5. An f-bounded proof system called exactly bounded if it accepts only proofs of size exactly $f(\varphi)$.

Note that any bounded proof system could be made exactly bounded by simply adding the padding to all short proofs, but it may cause two following problems. The first problem is that this operation changes the size of the shortest proof, which is important for automatizability. And the second problem is that we can lose optimality once again, if the bound is too large, because a simulation possibly would not be able to output all the padding in time. However, this problem could be avoided by allowing the simulations to write padding for free. This change of definition would make some trivial systems optimal (e.g. the system accepting proofs consisting of padding only), but it is not a problem, because all those systems are automatizable. Moreover, we would discuss mostly the case of simulation of proof systems with long proofs inside systems with short proofs, where this difference is not important (because we always able to output all padding in time), so all theorems below holds for both definitions of simulations.

Lemma 2. *If there exists an automatizable optimal proof system* Π*, then there exists an exactly bounded* Copy*-optimal proof system* Π'*.*

Proof. Let A be a polynomial-time algorithm that given formula φ finds its Π-proof and let τ be the running time of A. Consider the proof system Π' from Theorem 1 and the tuple $\langle A, \varepsilon, 1^\tau \rangle$ where ε is the empty string. By the definition of Π', the tuple $\langle A, \varepsilon, 1^\tau \rangle$ is a valid Π'-proof, its size polynomially bounded by the size of the shortest Π-proof and depends only on the size of the given formula. Therefore, this proof system could be made exactly bounded by simply rejecting all the proofs of size other than the size of the given tuple. □

The converse also holds, which is a corollary to Theorem 3.

In the rest of the work we will discuss the problem of the existence of an exactly bounded proof system that is optimal under simulations with relatively small classes of circuits. Out goal would be to prove that for certain classes C of circuits an exactly bounded C-optimal proof system exists if and only if it is automatizable in polynomial or expected polynomial time. The only way to make a proof system exactly bounded is to add padding for the short proofs and to reject long. As it was mentioned above, this operation does not preserve the size of the shortest proof, so if the bound was too large, the automatizability of the generated proof system does not necessarily mean the automatizability of the original proof system. But it is true in the most interesting case when the bound is polynomial in size of the shortest proof. Thus, informally speaking, results of this kind (namely, Theorems 3 and 5) state that generating efficient proofs in the optimal proof system (here optimal in the sense of monotone and AC^0-simulations, respectively) based on proofs in other systems is almost as hard as automatizing.

3 The structure of hard instances

In this section we present the family of proof systems that would be used in both Theorems 3 and 5 as proof systems that would be hard to simulate in the optimal proof system unless the optimal proof system was automatizable.

Let Π be an exactly bounded proof system and f be an arbitrary function. Consider an arbitrary formula φ. By the definition of an exactly bounded proof system all Π-proofs of φ have the same size m. Now, we construct proof system $\Phi_f(\Pi)$ using proof system Π (from now on we will omit parameter Π when it will be clear what proof system is meant). All the Φ_f-proofs of φ consist of m blocks of the size $f(m)$, thats it, all Φ_f-proofs are of the size $mf(m)$ and Φ_f is also an exactly bounded proof system. We match bits of Π-proofs to the blocks of Φ_f-proofs. A string x of the size $mf(m)$ is a valid Φ_f-proof if and only if there exists a Π-proof y such that each bit of y is equal to the parity of the bits of the corresponding block in x. In other words, if we consider x as a sequence of m binary strings $\langle x_1, x_2, \ldots, x_m \rangle$, where $|x_i| = f(m)$ for all i, the string $y = y_1 y_2 \ldots y_m$, where $y_i \coloneqq \sum_j x_{i,j} \mod 2$, would be a valid Π-proof if and only if x is a valid Φ_f-proof. Note

	Block 1	Block 2	Block 3	Block 4	Block 5	Π-proof
⋮			⋮			⋮
x_2	110000	110000	110000	110000	110000	00000
y_2	111000	110000	111000	111000	110000	10110
x_3	111000	111000	111000	111000	111000	11111
y_3	111000	111100	111000	111000	111100	10110
x_4	111100	111100	111100	111100	111100	00000
⋮			⋮			⋮

Figure 1: An example of (x_i) and (y_i) for $m = 5$ and Π-proof 10110

that this definition also describes an algorithm that can verify Φ_f-proof in time $f(m) \cdot \mathsf{poly}(m)$. Thus, Φ_f is a valid proof system for any function $f(m)$ that can be computed in time $\mathsf{poly}(m)$.

4 Optimality under monotone simulations

In this section we consider the case of simulations with monotone circuits.

Theorem 3. *If there exists an exactly bounded proof system* Π *that is optimal under simulations with monotone circuits, then* Π *is automatizable.*

Proof. Let m be the size of all Π-proofs. Note that m depends on formula φ. Consider proof system $\Phi_{m+1}(\Pi)$. Since Π is optimal, there is a monotone simulation that maps all valid Φ_{m+1} proofs into Π-proofs. The idea of the proof is to construct a set of polynomial quantity of Φ_{m+1}-proofs and then to prove that at least one of these proofs should be mapped to some valid Π-proof by the monotone simulation.

For two Boolean strings x, y of the same size, we say that $x \leq y$ if each bit of x is less or equal than the corresponding bit of y. Consider the sequence of strings $x_0, x_1, \ldots, x_{m+1}$ where x_i is the string consisting of m equal blocks that consist of i ones and $m + 1 - i$ zeroes in this exact order, $m(m + 1)$ bits in total (see Figure 1). Recall that each Φ_{m+1}-proof corresponds to the Π-proof obtained by computing parities of the bits in each block. Note that x_i corresponds to the string of m zeros if i is even and to the string of m ones if i is odd. Therefore, for every i, there exists a valid Φ_{m+1}-proof y_i such that $x_i \leq y_i \leq x_{i+1}$. Now, consider a monotone circuit C that maps Φ_{m+1}-proofs into Π-proofs. By monotonicity, for each i, either $C(x_{i+1})$ have strictly more ones than $C(x_i)$ or $C(x_i) = C(y_i) = C(x_{i+1})$. Obviously, $C(x_{i+1})$ cannot have strictly more ones than $C(x_i)$ for all i since all $C(x_i)$ are m-bit strings and m-bit string cannot have more than m ones. Therefore, C should map at least one of x_i into the same string as some y_j, that is, into a valid Π-proof.

Let A be the algorithm that does the following:

1. Generates a monotone circuit C that maps Φ_{m+1}-proofs into Π-proofs.
2. Computes $C(x_i)$ for all i, checks whether $\Pi(\varphi, C(x_i)) = 1$ and returns the first $C(x_i)$ that is a valid Π-proof.

Since one of the $C(x_i)$ maps into a valid Π-proof, the algorithm always finds a proof. It only remains to check whether A runs in polynomial time. By definition, the first step could be done in time polynomial in $m(m+1)$ which is clearly polynomial in m. The second step is also can be done in polynomial in m time since the size of C is polynomial in m and there are only $m+1$ strings x_i. Thus, A runs in polynomial in m time and hence Π is automatizable. □

Corollary 4. *An automatizable p-optimal proof systems exists if and only if there exists an exactly bounded proof system* Π *that is optimal under simulations with monotone circuits.*

Proof. Directly follows from Lemma 2 and Theorem 3 and the fact that Copy-simulation implies monotone simulation and monotone simulation implies p-simulation. □

5 Optimality under AC^0-simulations

In this section we prove the following theorem.

Theorem 5. *If there exists an exactly bounded* AC^0*-optimal proof system, then it is automatizable in expected polynomial time.*

Firstly, we prove the following simplified version of this theorem, and then we adapt the proof for Theorem 5.

Theorem 6. *If there exists an exactly bounded* AC^0*-optimal proof system such that its proofs are verifiable in* AC^0 *(not necessarily uniform), then it is automatizable in expected polynomial time.*

In order to prove this theorem we firstly deduce it from the following theorem and then prove the remaining using several lemmas.

Let $\mathcal{R}^l_n$ be a set of all partial assignments that leave exactly l unassigned variables. Note that $\mathcal{R}^0_n$ is then a set of all possible assignments of the size n. Let L_f be the language of Boolean strings divided into m blocks of the size $f(m)$ such that the parity of all bits in each block is 0.

Theorem 7. *There exists a function* $f = \mathsf{poly}(m)$ *such that for any language* L *containing* L_f *as a subset, if* $L \in \mathsf{AC}^0$*, then* $\Pr_{x \leftarrow \mathcal{R}^0_{mf(m)}}[x \in L] = 1 - o(1)$.

Proof of Theorem 6. Let Π be a system from the theorem statement. Now, we construct an algorithm that given a formula φ constructs its Π-proof. We would describe the algorithm for a fixed φ, however, it would be clear that the algorithm is polynomial in sum of the sizes of φ and its Π-proof. Consider the proof system $\Phi_f(\Pi)$ (from now on we will denote it as Φ_f) with a function f from Theorem 7. By optimality of Π there exists an AC^0-circuit C that maps Φ_f-proofs into Π-proofs. Consider the language L of Φ_f-proofs (not necessarily valid) that are mapped by C into valid Π-proofs. This language is clearly accepted by the AC^0-circuit obtained by concatenation of circuit C and the circuit that verifies Π-proofs. Now, choose an arbitrary valid Π-proof x. Every valid Φ_f-proof should be mapped by circuit C into a valid Π-proof, therefore, L contains all the strings such that the parity of all bits in each block is equal to the corresponding bit in x. Without loss of generality we can assume $x = 0^m$, because otherwise we can add negations to the first bits in blocks corresponding to ones in x. Now, we are able to apply Theorem 7 and conclude that the algorithm that chooses a random string y and returns $C(y)$ finds a valid proof with at least constant probability. Thus, the algorithm that does this procedure repeatedly until finds a valid Π-proof works in expected polynomial time. □

In order to prove Theorem 7, we need decision tree version of Håstad's Switching Lemma [Juk92; Bea95] that is a corollary to the proof of Håstad's Switching Lemma by Razborov [Hås87; Raz95].

Let $\mathrm{DT}(f)$ be the minimal height of a decision tree for a function f.

Lemma 8 (Decision tree version of Håstad's Switching Lemma [Juk92; Bea95])**.** *Let f be an arbitrary Boolean function that can be computed by a tree-like AC-circuit of depth d and size M. Let $s \geq 2$ be an arbitrary integer and $p = \frac{1}{128s}$. Then,*

$$\Pr_{\rho \leftarrow \mathcal{R}_n^{p^d n}}[\mathrm{DT}(f|_\rho) > s] \leq M 2^{-s}.$$

We also need the following lemma.

Lemma 9. *Consider n variables divided into k blocks of the size b (i.e. $n = kb$). Let $f(\rho)$ be a function that takes a partial assignment on these variables and counts the number of blocks without unassigned variables. If $l \leq n$ and $l = \Omega(k^3)$, then*

$$\Pr_{\rho \leftarrow \mathcal{R}_n^l}[f(\rho) \geq 1] = o(1)$$

Proof.

$$\begin{aligned}\Pr_{\rho\leftarrow\mathcal{R}_n^l}[f(\rho)\geq 1] &\leq \mathbb{E}_{\rho\leftarrow\mathcal{R}_n^l} f(\rho) = \frac{k\binom{(k-1)b}{l}}{\binom{kb}{l}} \\ &= k\frac{\prod_{j<l} kb-j-b}{\prod_{j<l} kb-j} \\ &= k\prod_{j<l}\left(1-\frac{b}{kb-j}\right) \\ &\leq k\left(1-\frac{1}{k}\right)^l = o(1).\end{aligned}$$

The second equality holds because of linearity of expectation and in the last equality we use the fact that $l=\Omega(m^3)$. □

The following proof is similar to the proof of the fact that no AC^0 circuit could compute parity on more than a constant part of all possible input strings, but has a couple of additional difficulties.

Proof of Theorem 7. Consider a tree-like AC-circuit C that accepts language L. Let M be its size and d be its depth. The proof consists of two following steps.

1. We apply a random partial assignment on C and use Lemma 8 to obtain a function with small DT with probability $1-o(1)$.
2. Then, knowing the fact that $L_f \subset L$, we prove that with probability $1-o(1)$ the obtained function is constant 1.

In order to use Lemma 8, we need to consider random partial assignment ρ chosen equiprobably from $\mathcal{R}_{mf(m)}^{p^d mf(m)}$, where $p=\frac{1}{128s}$ for some parameter s. Now, our goal is to choose parameters s and $f(m)$. We firstly state all restrictions on s and f generated by both steps 1 and 2 and after that choose proper values for s and f. Note that while f is a function of m, parameter s could be dependent on some parameters of circuit C, namely, d.

We start with step 2. Let $C|_\rho$ denote the function obtained after step 1. Clearly, if $\mathrm{DT}(C|_\rho)$ is less than numbers of unassigned variables in each block, then each leaf of the shortest decision tree corresponds to a partial assignment that does not make L_f constant. $L_f \subset L$ and the function in each leaf is constant, therefore, value in each leaf is 1 and $C|_\rho \equiv 1$. In order to ensure that $\mathrm{DT}(C|_\rho)$ is less than numbers of unassigned variables in each block with probability $1-o(1)$ it is sufficient to make sure that $s<m$ and the number of unassigned variables in each block is at least m also with probability $1-o(1)$. For the second restriction we can apply Lemma 9 with $k=m^2$ (we divide each of m blocks into m subblocks and use

them as blocks in the statement of the lemma) and replace it with restriction $p^d m f(m) \geq m^6$.

Step 1 does not add any additional specific restrictions, except for $s \geq 2$ and the restriction on the probability of success $M2^{-s} = o(1)$. As it easy to see, both these restrictions are satisfied if we choose $s = \Theta(m^{g(d)})$ for some function g.

Thus, our restrictions are $s = \Theta(m^{g(d)}), g(d) < 1$ and $p^d m f(m) \geq m^6$. For the last restriction we can use $p = \frac{1}{128s}$ and rewrite it in the form $m^5(128s)^d \leq f(m)$. Now it is clear that we can choose $f(m) = m^6$ and $s = \frac{\sqrt[d]{m}}{128}$ and satisfy all the restrictions. □

Now we can prove Theorem 5. This proof is a pure adaptation of the proof above to the general case, but it is more complicated.

Proof of Theorem 5. The algorithm is the same as in the proof of the theorem 6, but the parameter f would be different. Similarly to the proof above we consider AC-circuit C that maps Φ_f-proofs into Π-proofs and the function F that takes a Φ_f-proof x and answers whether $C(x)$ is a valid Π-proof. Our goal is to prove that $\Pr_{x \leftarrow \mathcal{R}^0_{m^{10}}}[F(x) = 1] = 1 - o(1)$ which would immediately imply that the algorithm that repeatedly computes $C(x)$ with a random string x finds a valid Π-proof in expected polynomial time.

Function F clearly can be computed in polynomial time, but possibly not by AC^0 circuit anymore. Hence, we have to manually verify that the proof of Theorem 7 can be applied to F. C is a circuit with m output gates. Without loss of generality we can replace it with m separate tree-like AC^0 circuits with one output gates each. It affects the size of C, but it remains polynomial in m, so we can afford it.

Similarly to the proof of Theorem 7 we apply Lemma 8 to these circuits and obtain m functions with $\mathrm{DT} \leq s$ with probability at least $1 - M2^{-s}$ (here M is the size of C, i.e. the sum of sizes of m circuits corresponding to the output bits). $F|_\rho$ is a function that depends only on the answers of the obtained m circuits, therefore, $\mathrm{DT}(F|_\rho)$ can never exceed the sum of DT of obtained circuits and the following inequality holds

$$\Pr_{\rho \leftarrow \mathcal{R}^{p^d n}_n}[\mathrm{DT}(F|_\rho) > sm] \leq M2^{-s}.$$

Now, in order to make sure that $F|_\rho \equiv 1$ with probability at least $1 - o(1)$ we need to add restriction $s < m$ and make sure that each block has at least m^2 unassigned variables with probability at least $1 - o(1)$. It also can be done using Lemma 9, but this time with $k = m^3$ (we divide each block into m^2 subblocks) and with restriction $p^d m f(m) \geq m^9$.

Now, we can choose $f(m) = m^9$ and $s = \frac{\sqrt[d]{m}}{128}$ to satisfy all restrictions and conclude that $\Pr_{x \leftarrow \mathcal{R}^0_{m^{10}}}[F(x) = 1] = 1 - o(1)$. □

Note that although the probabilities of the success in both Theorems 6 and 5 are close to 1, this approach cannot be generalized to the case of deterministic automatization, because the algorithm uses circuits in the simulation only as a black box, and can not guarantee that it finds the valid proof in any feasible time. Moreover, if there exists an optimal automatizable proof system, then for any deterministic algorithm of this type there exists an optimal proof system Π' and an AC^0-simulation of proof system $\Phi(\Pi')$ that answers invalid proofs on all queried proofs.

References

[AM20] Albert Atserias and Moritz Müller. "Automating Resolution is NP-Hard". In: *J. ACM* 67.5 (Sept. 2020). ISSN: 0004-5411. DOI: 10.1145/3409472.

[Bea95] Paul Beame. "A Switching Lemma Primer". In: *Technical Report UW-CSE* (1995), pp. 1–24.

[CK07] Stephen Cook and Jan Krajíček. "Consequences of the provability of NP $\subseteq$ P/poly". In: *Journal of Symbolic Logic* 72.4 (2007), pp. 1353–1371. DOI: 10.2178/jsl/1203350791.

[CR79] Stephen A. Cook and Robert A. Reckhow. "The relative efficiency of propositional proof systems". In: *Journal of Symbolic Logic* 44.1 (1979), pp. 36–50. DOI: 10.2307/2273702.

[de +20] Susanna F. de Rezende et al. *Automating Algebraic Proof Systems is NP-Hard*. English. WorkingPaper. Electronic Colloquium on Computational Complexity, May 2020.

[Gar20] Michal Garlík. *Failure of Feasible Disjunction Property for k-DNF Resolution and NP-hardness of Automating It*. 2020. arXiv: 2003.10230 [cs.CC].

[Göö+20] Mika Göös et al. "Automating Cutting Planes is NP-Hard". In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2020. Chicago, IL, USA: Association for Computing Machinery, 2020, pp. 68–77. ISBN: 9781450369794. DOI: 10.1145/3357713.3384248.

[Hås87] Johan Håstad. *Computational Limitations of Small-Depth Circuits*. Cambridge, MA, USA: MIT Press, 1987. ISBN: 0262081679.

[Juk92] Stasys Jukna. *Boolean Function Complexity*. Ed. by M. S. Paterson. Cambridge University Press, Nov. 1992. ISBN: 9780521408264. DOI: 10.1017/CBO9780511526633.

[KMT03] Johannes Köbler, Jochen Messner, and Jacobo Torán. "Optimal proof systems imply complete sets for promise classes". In: *Information and Computation* 184.1 (July 2003), pp. 71–92. ISSN: 08905401. DOI: 10.1016/S0890-5401(03)00058-0.

[KP89] Jan Krajíček and Pavel Pudlák. "Propositional proof systems, the consistency of first order theories and the complexity of computations". In: *Journal of Symbolic Logic* 54.3 (1989), pp. 1063–1079. DOI: 10.2307/2274765.

[PS10] Toniann Pitassi and Rahul Santhanam. "Effectively Polynomial Simulations". In: *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*. Ed. by Andrew Chi-Chih Yao. Tsinghua University Press, 2010, pp. 370–382. URL: http://conference.iiis.tsinghua.edu.cn/ICS2010/content/papers/29.html.

[Pud01] Pavel Pudlák. "On Reducibility and Symmetry of Disjoint NP-Pairs". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 2136. 2001, pp. 621–632. ISBN: 9783540446835. DOI: 10.1007/3-540-44683-4_54.

[Raz94] Alexander A Razborov. "On provably disjoint NP-pairs". In: *BRICS Report Series* 1.36 (Nov. 1994). ISSN: 1601-5355. DOI: 10.7146/brics.v1i36.21607.

[Raz95] Alexander A. Razborov. "Bounded Arithmetic and Lower Bounds in Boolean Complexity". In: *Feasible Mathematics II*. Boston, MA: Birkhäuser Boston, 1995, pp. 344–386. DOI: 10.1007/978-1-4612-2566-9_12.